Topic 1
CS314 Course Introduction

Chapman: I didn't expect a kind of Spanish Inquisition.
Cardinal Ximinez: NOBODY expects the Spanish Inquisition! Our chief weapon is surprise...surprise and fear...fear and surprise.... Our two weapons are fear and surprise...and ruthless efficiency.... Our three weapons are fear, surprise, and ruthless efficiency...and an almost fanatical devotion to the Pope.... Our four...no... Amongst our weapons.... Amongst our weaponry...are such diverse elements as fear, surprise....

Mike Scott, Gates 6.304
scottm@cs.utexas.edu
www.cs.utexas.edu/~scottm/cs314/
Who Am I?

- Lecturer in CS department since 2000
- Undergrad Stanford, MSCS RPI
- US Navy for 8 years, submarines
- 2 years Round Rock High School
What We Will Do Today

- Discuss
  - course content
  - procedures
  - tools

- For your TO DO list:
  - complete items on the startup page

www.cs.utexas.edu/~scottm/cs314/handouts/startup.htm
Prerequisites

- Formal: CS312 with a grade of C- or higher
- Informal: Ability to design and implement programs in Java using the following:
  - variables and data types
  - expressions, order of operations
  - decision making (if statements)
    - including boolean logic and boolean expressions
  - loops (fixed and variable repetition)
  - procedures or functions
  - parameters (reference and value parameters, local variables, scope, problem generalization)
  - structures or records or objects
  - arrays (vectors, lists)
  - top down design (breaking big rocks into little rocks)
  - algorithm and data design
  - create and implement program of at least 200 - 300 loc
  - could you write a program to let 2 people play connect 4?
CS314 Topics

1. Introduction
2. Complexity
3. Encapsulation
4. Inheritance
5. Polymorphism
6. Generics
7. Interfaces
8. Iterators
9. Abstract Classes
10. Maps, Sets
11. Linked Lists
12. Recursion
13. Recursive Backtracking
14. Searching, Simple Sorts
15. Stacks
16. Queues
17. Fast Sorting
18. Trees
20. Graphs
21. Hash tables
22. Red-Black Trees
23. Heaps
24. Dynamic Programming
Data Structures

- simple definition:
  - Variables that store other variables
- We will learn a toolbox full of data structures
- … and how to build them …
- … and how to use new ones.
Clicker Question

Which of the following is a data structure?
A. a method
B. a try / catch block
C. a double
D. an array
E. more than one of A - D
Resources

- Class web site – most course material
- Schedule – readings
- Class discussion group – Piazza
- Labs, software (Java, Eclipse, Canvas)
- Teaching staff, lab hours
- Canvas, Grades and Program Submissions
Books and equipment

- clicker is required
- books are recommended, not required
- free alternatives on the web, see schedule
  - BJP (CS312 book)
  - Thinking Recursively in Java - recursion
Clicker Question 1

Which of these best describes you?

A. First year at UT and first year college student
B. First year at UT, transferring from another college or university.
C. In second year at UT.
D. In third year at UT
E. Other
Graded Course Components

- clicker participation, Attend your lecture
  - 42 or 28 lectures with clicker, **42 points total**

- Discussion section quizzes (Go to your section. Canvas LIES!!)
  - 9 quizzes, 10 points each, 1 online quiz 5 points: **95 points total**

- Programming projects
  - 11 projects, 20 points each: **220 points total**

- Midterms: Outside of class, UTC 2.112A
  - Exam 1, Wednesday 10/10, 7 – 9 pm, **160 points**
  - Exam 2, Thursday 11/7, 7 – 9 pm, **200 points**

- Final: **300 points**, Day, Time, Location TBD
  - \(42 + 95 + 220 + 160 + 200 + 300 = 1017\)

- clicker, Quizzes, Programming Assignments capped at 340 pts
  - 17 points of “slack” among those 3 components

- No points added! Grades based on 1000 points, not 1017

- Grades posted to Canvas
Grades and Performance

- Final grade determined by final point total and a 900 – 800 – 700 – 600 scale
  - plusses and minuses if within 25 points of cutoff:
    - 875 – 899: B+, 900 – 924: A-

- CS314 Historical Grades - my sections only
  - 76% C- or higher:
    - 27% A's, 31% B's, 18% C's
  - 10% D or F
  - 14% Q or W (drop)
  - ON CIS WORK LOAD EVALUATED AS HIGH
Assignments

- Non trivial programming projects
- Individual – do your own work
  – okay to share tests you write
- Programs checked automatically with plagiarism detection software
- Turn in the right thing - correct name, correct format or you will lose points / slip days
- Slip days
  – 6 for term, max 2 per assignment
  – don’t use frivolously
Randy Pausch, CS Professor at CMU said:

"When I got tenure a year early at Virginia, other Assistant Professors would come up to me and say, 'You got tenure early!?!?! What's your secret?!?!?' and I would tell them, 'Call me in my office at 10pm on Friday night and I'll tell you.'"

Meaning:
Some things don't have an easy solution.
Some things simply require a lot of hard work.
Succeeding in the Course - Meta

- “Be the first penguin”
  - Ask questions!!!
  - lecture, section, Piazza, lab hours

- “It is impossible to be perfect”
  - Mistakes are okay.
  - That is how we learn.
  - Trying to be perfect means not taking risks.
  - no risks, no learning

- “Find a Pack”
  - Make friends.
  - Study with them!
How to Get Help

- Piazza Post
- Lab Hours
- Class examples
- Examples from book
- Discuss with other students at a high level
Succeeding in the Course - Concrete

- Former student:
  - "I really like the boot camp nature of your course."
- do the readings
- start on assignments early
- get help from the teaching staff when you get stuck on an assignment
- attend lecture and discussion sections
- participate on the class discussion group
- do extra problems - http://tinyurl.com/pnzp28f
- study for tests using the old tests
- study for tests in groups
- ask questions and get help
Course Materials and Procedures

- Software
  - can work in CS department microlab, 1st or 3rd floor of GDC, Dell hall (north wing)
  - login via CS account name and password
  - can work at home if you wish
  - Java.
    • Web page has details under Software. - JDK 8.0
  - Optional IDE.
    • Recommended IDE is Eclipse, also free
Clicker Question 2

Which computer programming language are you most comfortable with?

A. Java
B. C or C++
C. Python
D. PHP
E. Other

"bit twiddling: 1. (pejorative) An exercise in tuning (see tune) in which incredible amounts of time and effort go to produce little noticeable improvement, often with the result that the code becomes incomprehensible."

- The Hackers Dictionary, version 4.4.7
Clicker Question 1

- "My program finds all the primes between 2 and 1,000,000,000 in 1.37 seconds."
  - how efficient is my solution, in terms of time?
  
A. Good
B. Bad
C. It depends
Efficiency

- Computer Scientists don’t just write programs.
- They also *analyze* them.
- How efficient is a program?
  - How much time does it take for a program to complete?
  - How much memory does a program use?
  - How do these change as the amount of data changes?
  - What is the difference between the best case and worst case efficiency if any?
Technique

- Informal approach for this class
  - more formal techniques in theory classes
- Many simplifications
  - view algorithms as Java programs
  - count executable statements in program or method
  - find number of statements as function of the amount of data
  - focus on the *dominant term* in the function
Counting Statements

```java
int x; // one statement
x = 12; // one statement
int y = z * x + 3 % 5 * x / i; // 1
x++; // one statement
boolean p = x < y && y % 2 == 0 ||
    z >= y * x; // 1
int[] data = new int[100]; // 100
data[50] = x * x + y * y; // 1
```
Clicker Question 2

What is output by the following code?

```java
int total = 0;
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 11; j++)
        total += 2;
System.out.println(total);
```

A. 24  
B. 120  
C. 143  
D. 286  
E. 338
What is output when method `sample` is called?

```java
public static void sample(int n, int m) {
    int total = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            total += 5;
    System.out.println(total);
}
```

A. 5  
B. \(n \times m\)  
C. \(n \times m \times 5\)  
D. \(n^m\)  
E. \((n \times m)^5\)
Example

```java
public int total(int[] values) {
    int result = 0;
    for (int i = 0; i < values.length; i++)
        result += values[i];
    return result;
}
```

- How many statements are executed by method `total` as a function of `values.length`?

  - Let $N = \text{values.length}$
    - $N$ is commonly used as a variable that denotes the amount of data.
Counting Up Statements

- `int result = 0; 1`
- `int i = 0; 1`
- `i < values.length; N + 1`
- `i++ N`
- `result += values[i]; N`
- `return total; 1`
- `T(N) = 3N + 4`
- `T(N) is the number of executable statements in method total as function of values.length`
Another Simplification

- When determining complexity of an algorithm we want to simplify things
  - hide some details to make comparisons easier

- Like assigning your grade for course
  - At the end of CS314 your transcript won’t list all the details of your performance in the course
  - it won’t list scores on all assignments, quizzes, and tests
  - simply a letter grade, B- or A or D+

- So we focus on the dominant term from the function and ignore the coefficient
Big O

- The most common method and notation for discussing the execution time of algorithms is \textit{Big O}, also spoken \textit{Order}.
- Big O is the \textit{asymptotic execution time} of the algorithm.
- Big O is an upper bounds.
- It is a mathematical tool.
- Hide a lot of unimportant details by assigning a simple grade (function) to algorithms.
Formal Definition of Big O

- $T(N)$ is $O(F(N))$ if there are positive constants $c$ and $N_0$ such that $T(N) \leq cF(N)$ when $N \geq N_0$
  - $N$ is the size of the data set the algorithm works on
  - $T(N)$ is a function that characterizes the *actual* running time of the algorithm
  - $F(N)$ is a function that characterizes an upper bounds on $T(N)$. It is a limit on the running time of the algorithm. (The typical Big functions table)
  - $c$ and $N_0$ are constants
What it Means

- $T(N)$ is the actual growth rate of the algorithm
  - can be equated to the number of executable statements in a program or chunk of code
- $F(N)$ is the function that bounds the growth rate
  - may be upper or lower bound
- $T(N)$ may not necessarily equal $F(N)$
  - constants and lesser terms ignored because it is a bounding function
Showing $O(N)$ is Correct

- Recall the formal definition of Big O
  - $T(N)$ is $O(F(N))$ if there are positive constants $c$ and $N_0$ such that $T(N) \leq cF(N)$ when $N > N_0$
- Recall method $total$, $T(N) = 3N + 4$
  - show method $total$ is $O(N)$.
  - $F(N)$ is $N$
- We need to choose constants $c$ and $N_0$
- how about $c = 4$, $N_0 = 5$ ?
vertical axis: time for algorithm to complete. (simplified to number of executable statements)

- $T(N)$, actual function of time. In this case $3N + 4$
- $F(N)$, approximate function of time. In this case $N$
- $c \cdot F(N)$, in this case, $c = 4$; $c \cdot F(N) = 4N$

horizontal axis: $N$, number of elements in data set

$N_0 = 5$
## Typical Big O Functions – "Grades"

<table>
<thead>
<tr>
<th>Function</th>
<th>Common Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N!$</td>
<td>factorial</td>
</tr>
<tr>
<td>$2^N$</td>
<td>Exponential</td>
</tr>
<tr>
<td>$N^d, d &gt; 3$</td>
<td>Polynomial</td>
</tr>
<tr>
<td>$N^3$</td>
<td>Cubic</td>
</tr>
<tr>
<td>$N^2$</td>
<td>Quadratic</td>
</tr>
<tr>
<td>$\sqrt{N}$</td>
<td>$N$ Square root $N$</td>
</tr>
<tr>
<td>$N \log N$</td>
<td>$N$ log $N$</td>
</tr>
<tr>
<td>$N$</td>
<td>Linear</td>
</tr>
<tr>
<td>$\sqrt{N}$</td>
<td>Root - $n$</td>
</tr>
<tr>
<td>$\log N$</td>
<td>Logarithmic</td>
</tr>
<tr>
<td>$1$</td>
<td>Constant</td>
</tr>
</tbody>
</table>
Clicker Question 4

Which of the following is true?

A. Method total is $O(N^{1/2})$
B. Method total is $O(N)$
C. Method total is $O(N^2)$
D. Two of A – C are correct
E. All of three of A – C are correct
Dealing with other methods

- What do I do about method calls?

```java
double sum = 0.0;
for (int i = 0; i < n; i++)
    sum += Math.sqrt(i);
```

- Long way
  - go to that method or constructor and count statements

- Short way
  - substitute the simplified Big O function for that method.
  - if Math.sqrt is constant time, O(1), simply count
    ```java
    sum += Math.sqrt(i);
    ``` as one statement.
Dealing With Other Methods

public int foo(int[] data) {
    int total = 0;
    for (int i = 0; i < data.length; i++)
        total += countDups(data[i], data);
    return total;
}

// method countDups is O(N) where N is the
// length of the array it is passed

What is the Big O of foo?

A. O(1)  B. O(N)  C. O(NlogN)
D. O(N^2) E. O(N!)
Independent Loops

// from the Matrix class
public void scale(int factor) {
    for (int r = 0; r < numRows(); r++)
        for (int c = 0; c < numCols(); c++)
            iCells[r][c] *= factor;
}

Assume numRows() = numCols() = N.
In other words, a square Matrix.
numRows and numCols are O(1)

What is the T(N)? What is the Big O?
A. O(1)       B. O(N)       C. O(NlogN)
D. O(N^2)     E. O(N!)

CS 314  Efficiency - Complexity  20
Just Count Loops, Right?

// assume mat is a 2d array of booleans
// assume mat is square with N rows,
// and N columns

int numThings = 0;
for (int r = row - 1; r <= row + 1; r++)
    for (int c = col - 1; c <= col + 1; c++)
        if (mat[r][c])
            numThings++;

What is the order of the above code?
A. O(1)  B. O(N)  C. O(N^2)  D. O(N^3)  E. O(N^{1/2})
It is Not Just Counting Loops

// Second example from previous slide could be rewritten as follows:
int numThings = 0;
if (mat[r-1][c-1]) numThings++;
if (mat[r-1][c]) numThings++;
if (mat[r-1][c+1]) numThings++;
if (mat[r][c-1]) numThings++;
if (mat[r][c]) numThings++;
if (mat[r][c+1]) numThings++;
if (mat[r+1][c-1]) numThings++;
if (mat[r+1][c]) numThings++;
if (mat[r+1][c+1]) numThings++;
Sidetrack, the logarithm

- Thanks to Dr. Math
- \(3^2 = 9\)
- likewise \(\log_3 9 = 2\)
  - "The log to the base 3 of 9 is 2."
- The way to think about log is:
  - "the log to the base \(x\) of \(y\) is the number you can raise \(x\) to to get \(y\)."
  - Say to yourself "The log is the exponent." (and say it over and over until you believe it.)
  - In CS we work with base 2 logs, a lot
- \(\log_2 32 = ?\) \(\log_2 8 = ?\) \(\log_2 1024 = ?\) \(\log_{10} 1000 = ?\)
When Do Logarithms Occur

- Algorithms tend to have a logarithmic term when they use a divide and conquer technique.
- The data set keeps getting divided by 2.

```java
public int foo(int n) {
    // pre n > 0
    int total = 0;
    while (n > 0) {
        n = n / 2;
        total++;
    }
    return total;
}
```

- What is the order of the above code?
  A. O(1)  B. O(logN)  C. O(N)
  D. O(Nlog N)  E. O(N^2)
Significant Improvement – Algorithm with Smaller Big O function

- Problem: Given an array of ints replace any element equal to 0 with the maximum positive value to the right of that element. (if no positive value to the right, leave unchanged.)

Given:
[0, 9, 0, 13, 0, 0, 7, 1, -1, 0, 1, 0]

Becomes:
[13, 9, 13, 13, 7, 7, 7, 1, -1, 1, 1, 0]
Replace Zeros – Typical Solution

public void replace0s(int[] data) {
    for (int i = 0; i < data.length - 1; i++) {
        if (data[i] == 0) {
            int max = 0;
            for (int j = i + 1; j < data.length; j++)
                max = Math.max(max, data[j]);
            data[i] = max;
        }
    }
}

Assume all values are zeros. (worst case)
Example of a dependent loops.
Replace Zeros – Alternate Solution

```java
public void replace0s(int[] data){
    int max =
        Math.max(0, data[data.length - 1]);
    int start = data.length - 2;
    for (int i = start; i >= 0; i--)
        if (data[i] == 0)
            data[i] = max;
        else
            max = Math.max(max, data[i]);
}

Big O of this approach?
A. O(1)          B. O(N)          C. O(NlogN)
D. O(N^2)        E. O(N!)
```
A Useful Proportion

- Since $F(N)$ is characterizes the running time of an algorithm the following proportion should hold true:

$$F(N_0) / F(N_1) \approx \text{time}_0 / \text{time}_1$$

- An algorithm that is $O(N^2)$ takes 3 seconds to run given 10,000 pieces of data.
  - How long do you expect it to take when there are 30,000 pieces of data?
  - common mistake
  - logarithms?
Why Use Big O?

- As we build data structures Big O is the tool we will use to decide under what conditions one data structure is better than another.
- Think about performance when there is a lot of data.
  - "It worked so well with small data sets..."
  - Joel Spolsky, Schlemiel the painter's Algorithm
- Lots of trade offs
  - some data structures good for certain types of problems, bad for other types
  - often able to trade SPACE for TIME.
  - Faster solution that uses more space
  - Slower solution that uses less space
Big O Space

- Big O could be used to specify how much space is needed for a particular algorithm
  - in other words how many variables are needed

- Often there is a *time – space tradeoff*
  - can often take less time if willing to use more memory
  - can often use less memory if willing to take longer
  - truly beautiful solutions take less time and space

*The biggest difference between time and space is that you can't reuse time.* - Merrick Furst
Quantifiers on Big O

- It is often useful to discuss different cases for an algorithm
  - Best Case: what is the best we can hope for?
    - least interesting
  - Average Case (a.k.a. expected running time): what usually happens with the algorithm?
  - Worst Case: what is the worst we can expect of the algorithm?
    - very interesting to compare this to the average case
Best, Average, Worst Case

To Determine the best, average, and worst case Big O we must make assumptions about the data set

- Best case -> what are the properties of the data set that will lead to the fewest number of executable statements (steps in the algorithm)

- Worst case -> what are the properties of the data set that will lead to the largest number of executable statements

- Average case -> Usually this means assuming the data is randomly distributed
  - or if I ran the algorithm a large number of times with different sets of data what would the average amount of work be for those runs?
Another Example

public double minimum(double[] values)
{
    int n = values.length;
    double minValue = values[0];
    for(int i = 1; i < n; i++)
        if(values[i] < minValue)
            minValue = values[i];
    return minValue;
}

- If no other information, assume asking average case
Example of Dominance

- Look at an extreme example. Assume the actual number as a function of the amount of data is:
  \[ N^2/10000 + 2N\log_{10} N + 100000 \]
- Is it plausible to say the \( N^2 \) term dominates even though it is divided by 10000 and that the algorithm is \( O(N^2) \)?
- What if we separate the equation into \( (N^2/10000) \) and \( (2N \log_{10} N + 100000) \) and graph the results.
For large values of $N$ the $N^2$ term dominates so the algorithm is $O(N^2)$.

When does it make sense to use a computer?
Comparing Grades

- Assume we have a problem
- Algorithm A solves the problem correctly and is $O(N^2)$
- Algorithm B solves the same problem correctly and is $O(N \log_2 N)$
- Which algorithm is faster?
- One of the assumptions of Big O is that the data set is large.
- The "grades" should be accurate tools if this is true
Running Times

Assume $N = 100,000$ and processor speed is $1,000,000,000$ operations per second

<table>
<thead>
<tr>
<th>Function</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^N$</td>
<td>$3.2 \times 10^{30,086}$ years</td>
</tr>
<tr>
<td>$N^4$</td>
<td>3171 years</td>
</tr>
<tr>
<td>$N^3$</td>
<td>11.6 days</td>
</tr>
<tr>
<td>$N^2$</td>
<td>10 seconds</td>
</tr>
<tr>
<td>$\sqrt{N}$</td>
<td>0.032 seconds</td>
</tr>
<tr>
<td>$N \log N$</td>
<td>0.0017 seconds</td>
</tr>
<tr>
<td>$\sqrt{N}$</td>
<td>0.0001 seconds</td>
</tr>
<tr>
<td>$\log N$</td>
<td>$3.2 \times 10^{-7}$ seconds</td>
</tr>
<tr>
<td></td>
<td>1.2 \times 10^{-8}$ seconds</td>
</tr>
</tbody>
</table>
Theory to Practice OR
Dykstra says: "Pictures are for the Weak."

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>4000</th>
<th>8000</th>
<th>16000</th>
<th>32000</th>
<th>64000</th>
<th>128K</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(N)</td>
<td>2.2x10⁻⁵</td>
<td>2.7x10⁻⁵</td>
<td>5.4x10⁻⁵</td>
<td>4.2x10⁻⁵</td>
<td>6.8x10⁻⁵</td>
<td>1.2x10⁻⁴</td>
<td>2.3x10⁻⁴</td>
<td>5.1x10⁻⁴</td>
</tr>
<tr>
<td>O(NlogN)</td>
<td>8.5x10⁻⁵</td>
<td>1.9x10⁻⁴</td>
<td>3.7x10⁻⁴</td>
<td>4.7x10⁻⁴</td>
<td>1.0x10⁻³</td>
<td>2.1x10⁻³</td>
<td>4.6x10⁻³</td>
<td>1.2x10⁻²</td>
</tr>
<tr>
<td>O(N³/²)</td>
<td>3.5x10⁻⁵</td>
<td>6.9x10⁻⁴</td>
<td>1.7x10⁻³</td>
<td>5.0x10⁻³</td>
<td>1.4x10⁻²</td>
<td>3.8x10⁻²</td>
<td>0.11</td>
<td>0.30</td>
</tr>
<tr>
<td>O(N²) ind.</td>
<td>3.4x10⁻³</td>
<td>1.4x10⁻³</td>
<td>4.4x10⁻³</td>
<td>0.22</td>
<td>0.86</td>
<td>3.45</td>
<td>13.79</td>
<td>(55)</td>
</tr>
<tr>
<td>O(N²) dep.</td>
<td>1.8x10⁻³</td>
<td>7.1x10⁻³</td>
<td>2.7x10⁻²</td>
<td>0.11</td>
<td>0.43</td>
<td>1.73</td>
<td>6.90</td>
<td>(27.6)</td>
</tr>
<tr>
<td>O(N³)</td>
<td>3.40</td>
<td>27.26</td>
<td>(218)</td>
<td>(1745)</td>
<td>(13,957)</td>
<td>(112k)</td>
<td>(896k)</td>
<td>(7.2m)</td>
</tr>
</tbody>
</table>

Times in Seconds. Red indicates predicated value.
## Change between Data Points

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>2000</th>
<th>4000</th>
<th>8000</th>
<th>16000</th>
<th>32000</th>
<th>64000</th>
<th>128K</th>
<th>256k</th>
<th>512k</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>O(N)</strong></td>
<td>-</td>
<td>1.21</td>
<td>2.02</td>
<td>0.78</td>
<td>1.62</td>
<td>1.76</td>
<td>1.89</td>
<td>2.24</td>
<td>2.11</td>
<td>1.62</td>
</tr>
<tr>
<td><strong>O(NlogN)</strong></td>
<td>-</td>
<td>2.18</td>
<td>1.99</td>
<td>1.27</td>
<td>2.13</td>
<td>2.15</td>
<td>2.15</td>
<td>2.71</td>
<td>1.64</td>
<td>2.40</td>
</tr>
<tr>
<td><strong>O(N^{3/2})</strong></td>
<td>-</td>
<td>1.98</td>
<td>2.48</td>
<td>2.87</td>
<td>2.79</td>
<td>2.76</td>
<td>2.85</td>
<td>2.79</td>
<td>2.82</td>
<td>2.81</td>
</tr>
<tr>
<td><strong>O(N^2) ind</strong></td>
<td>-</td>
<td>4.06</td>
<td>3.98</td>
<td>3.94</td>
<td>3.99</td>
<td>4.00</td>
<td>3.99</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>O(N^2) dep</strong></td>
<td>-</td>
<td>4.00</td>
<td>3.82</td>
<td>3.97</td>
<td>4.00</td>
<td>4.01</td>
<td>3.98</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>O(N^3)</strong></td>
<td>-</td>
<td>8.03</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Value obtained by $\frac{\text{Time}_x}{\text{Time}_{x-1}}$
Okay, Pictures

Results on a 2Ghz laptop

<table>
<thead>
<tr>
<th>Value of N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>NlogN</td>
<td></td>
</tr>
<tr>
<td>NsqrtN</td>
<td></td>
</tr>
<tr>
<td>N^2</td>
<td></td>
</tr>
<tr>
<td>N^2</td>
<td></td>
</tr>
</tbody>
</table>

CS 314 Efficiency - Complexity
Put a Cap on Time

Results on a 2Ghz laptop

Value of N vs Time graph:
- N
- N\log N
- N^{\sqrt N}
- N^2

Graph showing the comparison of different time complexities on a 2Ghz laptop.
No $O(N^2)$ Data

Results on a 2GhZ laptop

Value of $N$ vs. Time

- $N$
- $N\log N$
- $N\sqrt{N}$
Just $O(N)$ and $O(N \log N)$
Just $O(N)$
10^9 instructions/sec, runtimes

<table>
<thead>
<tr>
<th>N</th>
<th>O(log N)</th>
<th>O(N)</th>
<th>O(N log N)</th>
<th>O(N^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.00000000003</td>
<td>0.000000001</td>
<td>0.0000000033</td>
<td>0.00000001</td>
</tr>
<tr>
<td>100</td>
<td>0.0000000007</td>
<td>0.000000010</td>
<td>0.000000664</td>
<td>0.0001000</td>
</tr>
<tr>
<td>1,000</td>
<td>0.000000010</td>
<td>0.00001000</td>
<td>0.00010000</td>
<td>0.001</td>
</tr>
<tr>
<td>10,000</td>
<td>0.00000013</td>
<td>0.00010000</td>
<td>0.00132900</td>
<td>0.1 min</td>
</tr>
<tr>
<td>100,000</td>
<td>0.00000017</td>
<td>0.00100000</td>
<td>0.01661000</td>
<td>10 seconds</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.00000020</td>
<td>0.001</td>
<td>0.0199</td>
<td>16.7 minutes</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.00000030</td>
<td>1.0 second</td>
<td>30 seconds</td>
<td>31.7 years</td>
</tr>
</tbody>
</table>
Formal Definition of Big O (repeated)

- \( T(N) \) is \( O( F(N) ) \) if there are positive constants \( c \) and \( N_0 \) such that \( T(N) \leq cF(N) \) when \( N \geq N_0 \)
  - \( N \) is the size of the data set the algorithm works on
  - \( T(N) \) is a function that characterizes the *actual* running time of the algorithm
  - \( F(N) \) is a function that characterizes an upper bounds on \( T(N) \). It is a limit on the running time of the algorithm
  - \( c \) and \( N_0 \) are constants
There is a point $N_0$ such that for all values of $N$ that are past this point, $T(N)$ is bounded by some multiple of $F(N)$.

Thus if $T(N)$ of the algorithm is $O(N^2)$ then, ignoring constants, at some point we can bound the running time by a quadratic function.

Given a linear algorithm it is technically correct to say the running time is $O(N^2)$. $O(N)$ is a more precise answer as to the Big O of the linear algorithm.

- thus the caveat “pick the most restrictive function” in Big O type questions.
What it All Means

- **T(N)** is the actual growth rate of the algorithm
  - can be equated to the number of executable statements in a program or chunk of code
- **F(N)** is the function that bounds the growth rate
  - may be upper or lower bound
- **T(N)** may not necessarily equal **F(N)**
  - constants and lesser terms ignored because it is a bounding function
Other Algorithmic Analysis Tools

- **Big Omega** \( T(N) \) is \( \Omega( F(N) ) \) if there are positive constants \( c \) and \( N_0 \) such that \( T(N) \geq cF(N) \) when \( N \geq N_0 \)
  - Big O is similar to less than or equal, an upper bounds
  - Big Omega is similar to greater than or equal, a lower bound

- **Big Theta** \( T(N) \) is \( \theta( F(N) ) \) if and only if \( T(N) \) is \( O( F(N) ) \) and \( T( N ) \) is \( \Omega( F(N) ) \).
  - Big Theta is similar to equals
### Relative Rates of Growth

<table>
<thead>
<tr>
<th>Analysis Type</th>
<th>Mathematical Expression</th>
<th>Relative Rates of Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big $O$</td>
<td>$T(N) = O(F(N))$</td>
<td>$T(N) \leq F(N)$</td>
</tr>
<tr>
<td>Big $\Omega$</td>
<td>$T(N) = \Omega(F(N))$</td>
<td>$T(N) \geq F(N)$</td>
</tr>
<tr>
<td>Big $\theta$</td>
<td>$T(N) = \theta(F(N))$</td>
<td>$T(N) = F(N)$</td>
</tr>
</tbody>
</table>

"In spite of the additional precision offered by Big Theta, Big $O$ is more commonly used, except by researchers in the algorithms analysis field" - Mark Weiss
“And so, from Europe, we get things such as ... object-oriented analysis and design (a clever way of breaking up software programming instructions and data into small, reusable objects, based on certain abstraction principles and design hierarchies.)”

-Michael A. Cusumano, The Business Of Software
Creating large programs that work turns out to be very difficult
- DIA Automated baggage handling system
- Ariane 5 Flight 501
- More

Object oriented programming is one way of managing the complexity of programming and software projects

Break up big problems into smaller, more manageable problems
Object Oriented Programming

- "Object-oriented programming is a method of programming based on a hierarchy of classes, and well-defined and cooperating objects."

- What is a class?

- "A class is a structure that defines the data and the methods to work on that data. When you write programs in the Java language, all program data is wrapped in a class, whether it is a class you write or a class you use from the Java platform API libraries."
  - a new data type
Object Oriented Programming

- In other words break the problem up based on the things / data types that are part of the problem
- Not the only way
- One of many different kinds of strategies or paradigms for software development
  - functional, procedural, event driven, data flow, formal methods, agile or extreme, ...
- In 314 we will do a lot of object based programming
Example - Monopoly

If we had to start from scratch what classes would we need to create?
Encapsulation

- One of the features of object oriented languages
- Hide the data of an object (variable)
- Group operations and data together into a new data type
- Usually easier to *use* something than understand *exactly how it works*
  - microwave, car, computer, software, mp3 player
The IntList Class

- We will develop a class that models a list of ints
  - initially a poor man’s ArrayList
- Improvement on an array of ints
  - resize automatically
  - insert easily
  - remove easily
- A list - our first *data structure*
  - a variable that stores other variables
- Lists maintain elements in a definite order and duplicates are allowed
Clicker Question 1

Our `IntList` class will have an array of ints instance variable (`int[] container`). What should the capacity of this internal array be?

A. less than or equal to the size of the list
B. greater than or equal to the size of the list
C. equal to the size of the list
D. some fixed amount that never changes
E. 0
Clicker Question 2

When adding a new element to a list what should be the default location for the new element?

A. The beginning
B. The end
C. The middle
D. A random location
E. Don’t bother to actually add
IntList Design

- Create a new, empty IntList
  \texttt{new IntList} -> []

- The above is not code. It is a notation that shows what the results of operations. [] is an empty list.
  
- add to a list.
  
  [] \texttt{.add(1)} -> [1]
  [1] \texttt{.add(5)} -> [1, 5]
  [1, 5] \texttt{.add(4)} -> [1, 5, 4]

- elements in a list have a definite order and a position.
  
  – zero based position or 1 based positioning?
The IntList Class

- instance variables
- constructors
  - default
  - initial capacity
    - preconditions, exceptions, postconditions, assert
  - meaning of static
- add method
- get method
- size method
The IntList Class

- testing!!
- toString
  - “beware the performance of String concatenation” – Joshua Bloch
- insert method (int pos, int value)
- remove method (int pos)
- insertAll method
  (int pos, IntList other)
  – king of the IntLists
Timing Experiment

- Add N elements to an initially empty IntList then call toString. Time both events. How does the total time to add compare to the time to complete toString?

```java
IntList list = new IntList();
for (int i = 0; i < N; i++) {
    list.add(i);
    String s = list.toString();
}
```

A. time to add << time for toString()
B. time to add < time for toString()
C. time to add ~= time for toString()
D. time to add > time for toString()
E. time to add >> time for toString()
Instance Variables

- Internal data
  - also called instance variables because every instance (object) of this class has its own copy of these
  - something to store the elements of the list
  - size of internal storage container?
  - if not what else is needed

- Must be clear on the difference between the internal data of an IntList object and the IntList that is being represented

- Why make internal data private?
Abstract view of list of integers

[42, 12, 37]

The wall of abstraction.

```java
IntList aList = new IntList();
aList.add(42);
aList.add(12);
aList.add(37);
```
Constructors

- For initialization of objects
- IntList constructors
  - default
  - initial capacity?
- redirecting to another constructor
  this(10);
- class constants
  - what static means
Default add method

- where to add?
- what if not enough space?

```java
[].add(3) -> [3]
[3].add(5) -> [3, 5]
[3, 5].add(3) -> [3, 5, 3]
```

- Testing, testing, testing!
  - a `toString` method would be useful
toString method

- return a Java String of list
- empty list -> []
- one element -> [12]
- multiple elements -> [12, 0, 5, 4]
- Beware the performance of String concatenation.
- StringBuilder alternative
Clicker Question 3

What is output by the following code?

```java
IntList list = new IntList(25);
System.out.println( list.size() );
```

A. 25  
B. 0  
C. -1  
D. unknown  
E. No output due to runtime error.
get and size methods

- **get**
  - access element from list
  - preconditions?

  \[\textsf{[3, 5, 2].get(0)}\] \textbf{returns} 3
  \[\textsf{[3, 5, 2].get(1)}\] \textbf{returns} 5

- **size**
  - number of elements in the list
  - Do not confuse with the capacity of the internal storage container
  - The array is not the list!

  \[\textsf{[4, 5, 2].size()}\] \textbf{returns} 3
insert method

- add at someplace besides the end

\[ [3, 5].\text{insert}(1, 4) \rightarrow [3, 4, 5] \]

\[ [3, 4, 5].\text{insert}(0, 4) \rightarrow [4, 3, 4, 5] \]

- preconditions?
- overload add?
- chance for internal loose coupling
Clicker Question 4

What is output by the following code?

```java
IntList list = new IntList();
list.add(3);
list.insert(0, 4); // position, value
list.insert(1, 1);
list.add(5);
list.insert(2, 9);
System.out.println( list.toString() );
```

A. [4, 1, 3, 9, 5]
B. [3, 4, 1, 5, 9]
C. [4, 1, 9, 3, 5]
D. [3, 1, 4, 9, 5]
E. No output due to runtime error.
remove method

- remove an element from the list based on location

\[ [3, 4, 5].remove(0) \rightarrow [4, 5] \]
\[ [3, 5, 6, 1, 2].remove(2) \rightarrow [3, 5, 1, 2] \]

- preconditions?

- return value?
  - accessor methods, mutator methods, and mutator methods that return a value
Clicker Question 5

What is output by the following code?

```java
IntList list = new IntList();
list.add(12);
list.add(15);
list.add(12);
list.add(17);
list.remove(1);
System.out.println( list );
```

A. [15, 17]
B. [12, 17]
C. [12, 0, 12, 17]
D. [12, 12, 17]
E. [15, 12, 17]
insertAll method

- add all elements of one list to another starting at a specified location

\[ [5, 3, 7].insertAll(2, [2, 3]) \rightarrow [5, 3, 2, 3, 7] \]

The parameter \([2, 3]\) would be unchanged.

- Working with other objects of the same type
  - this?
  - where is private private?
  - loose coupling vs. performance
Class Design and Implementation – Another Example

This example will not be covered in class.
The Die Class

- Consider a class used to model a die
- What is the interface? What actions should a die be able to perform?

- The methods or behaviors can be broken up into constructors, mutators, accessors
The Die Class Interface

- Constructors (used in creation of objects)
  - default, single int parameter to specify the number of sides, int and boolean to determine if should roll

- Mutators (change state of objects)
  - roll

- Accessors (do not change state of objects)
  - getResult, getNumSides, toString

- Public constants
  - DEFAULT_SIDES
Visibility Modifiers

- All parts of a class have visibility modifiers
  - Java keywords
  - **public**, protected, **private**, (no modifier means package access)
  - do not use these modifiers on local variables (syntax error)

- **public** means that constructor, method, or field may be accessed outside of the class.
  - part of the interface
  - constructors and methods are generally public

- **private** means that part of the class is hidden and inaccessible by code outside of the class
  - part of the implementation
  - data fields are generally private
The Die Class Implementation

- Implementation is made up of constructor code, method code, and private data members of the class.
- scope of data members / instance variables
  - *private data members may be used in any of the constructors or methods of a class*
- Implementation is hidden from users of a class and can be changed without changing the interface or affecting clients (other classes that use this class)
  - Example: Previous version of Die class, DieVersion1.java
- Once Die class completed can be used in anything requiring a Die or situation requiring random numbers between 1 and N
  - DieTester class. What does it do?
DieTester method

```java
public static void main(String[] args) {
    final int NUM_ROLLS = 50;
    final int TEN_SIDED = 10;
    Die d1 = new Die();
    Die d2 = new Die();
    Die d3 = new Die(TEN_SIDED);
    final int MAX_ROLL = d1.getNumSides() +
                      d2.getNumSides() + d3.getNumSides();
    for(int i = 0; i < NUM_ROLLS; i++)
    {
        d1.roll();
        d2.roll();
        System.out.println("d1: " + d1.getResult() + " d2: " + d2.getResult() + " Total: " + (d1.getResult() + d2.getResult()) );
    }
```
DieTester continued

```java
int total = 0;
int numRolls = 0;
do {
    d1.roll();
    d2.roll();
    d3.roll();
    total = d1.getResult() + d2.getResult()
             + d3.getResult();
    numRolls++;
} while (total != MAX_ROLL);

System.out.println("\n\nNumber of rolls to get "
                 + MAX_ROLL + " was " + numRolls);
```
Correctness Sidetrack

- When creating the public interface of a class give careful thought and consideration to the *contract* you are creating between yourself and users (other programmers) of your class.

- Use *preconditions* to state what you assume to be true before a method is called.
  - caller of the method is responsible for making sure these are true.

- Use *postconditions* to state what you guarantee to be true after the method is done if the preconditions are met.
  - implementer of the method is responsible for making sure these are true.
Precondition and Postcondition Example

/************************************************************************
 * pre: numSides > 1
 * post: getResult() = 1, getNumSides() = sides
 */

public Die(int numSides)
{
    assert (numSides > 1) : "Violation of precondition: Die(int)";
    iMyNumSides = numSides;
    iMyResult = 1;
    assert getResult() == 1 && getNumSides() == numSides;
}
Object Behavior - Instantiation

Consider the DieTester class

```java
Die d1 = new Die();
Die d2 = new Die();
Die d3 = new Die(10);
```

When the new operator is invoked control is transferred to the Die class and the specified constructor is executed, based on parameter matching.

Space (memory) is set aside for the new object's fields.

The memory address of the new object is passed back and stored in the object variable (pointer).

After creating the object, methods may be called on it.
Creating Dice Objects

Die Tester class. Sees interface of Die class

Die class. Sees implementation. (of Die class.)
Objects

- Every Die object created has its own instance of the variables declared in the class blueprint
  
  ```java
  private int iMySides;
  private int iMyResult;
  ```

- thus the term *instance variable*

- the instance vars are part of the hidden implementation and may be of *any* data type
  – unless they are public, which is almost always a bad idea if you follow the tenets of information hiding and encapsulation
Complex Objects

- What if one of the instance variables is itself an object?
- add to the Die class

```java
private String myName;
```

- d1 can hold the memory address of a Die object. The instance variable myName inside a Die object can hold the memory address of a String object.
The Implicit Parameter

- Consider this code from the Die class

```java
public void roll()
{
    iMyResult =
        ourRandomNumGen.nextInt(iMySides) + 1;
}
```

- Taken in isolation this code is rather confusing.
- what is this iMyResult thing?
  - It's not a parameter or local variable
  - why does it exist?
  - it belongs to the Die object that called this method
  - if there are numerous Die objects in existence
  - Which one is used depends on which object called the method.
The *this* Keyword

- When a method is called it may be necessary for the calling object to be able to refer to itself
  - most likely so it can pass itself somewhere as a parameter
- when an object calls a method an implicit reference is assigned to the calling object
- the name of this implicit reference is *this*
- *this* is a reference to the current calling object and may be used as an object variable (may not declare it)
// in some class other than Die
Die d3 = new Die();
d3.roll();

// in the Die class
public void roll()
{
    iMyResult =
        ourRandomNumGen.nextInt(iMySides) + 1;
    /* OR
        this.iMyResult...
    */
}

this Visually

memory address
d3

memory address
this

a Die object
6 1

iMySides iMyResult
An equals method

- working with objects of the same type in a class can be confusing
- write an equals method for the Die class. assume every Die has a myName instance variable as well as iMyNumber and iMySides
A Possible Equals Method

```java
public boolean equals(Object otherObject) {
    Die other = (Die)otherObject;
    return iMySides == other.iMySides
        && iMyResult == other.iMyResult
        && myName.equals(other.myName);
}
```

- Declared Type of Parameter is Object not Die
- override (replace) the equals method instead of overload (present an alternate version)
  - easier to create generic code
- we will see the equals method is *inherited* from the Object class
- access to another object's private instance variables?
Another equals Methods

```java
public boolean equals(Object otherObject)
{
    Die other = (Die)otherObject;
    return this.iMySides == other.iMySides
        && this.iMyNumber == other.iMyNumber
        && this.myName.equals( other.myName );
}
```

Using the this keyword / reference to access the implicit parameters instance variables is unnecessary.
If a method within the same class is called within a method, the original calling object is still the calling object
public boolean equals(Object otherObject) {
    // check if objects identical
    if (this == otherObject)
        return true;
    // must return false if explicit parameter null
    if (otherObject == null)
        return false;
    // if objects not of same type they cannot be equal
    if (getClass() != otherObject.getClass())
        return false;
    // we know otherObject is a non null Die
    Die other = (Die)otherObject;
    return iMySides == other.iMySides
            && iMyNumber == other.iMyNumber
            && myName.equals(other.myName);
}
the instanceof Operator

- instanceof is a Java keyword.
- part of a boolean statement
  ```java
  public boolean equals(Object otherObj) {
    if (otherObj instanceof Die) {
      // now go and cast
      // rest of equals method
    }
  }
  ```
- Should not use instanceof in equals methods.
- instanceof has its uses but not in equals because of the contract of the equals method
Class Variables and Class Methods

- Sometimes every object of a class does not need its own copy of a variable or constant
- The keyword `static` is used to specify class variables, constants, and methods

```java
private static Random ourRandNumGen = new Random();
public static final int DEFAULT_SIDES = 6;
```

- The most prevalent use of static is for class constants.
  - if the value can't be changed why should every object have a copy of this non changing value
Class Variables and Constants

All objects of type Die have access to the class variables and constants.

A public class variable or constant may be referred to via the class name.
public class UseDieStatic
{
    public static void main(String[] args)
    {
        System.out.println("Die.DEFAULT_SIDES "+Die.DEFAULT_SIDES);
        // Any attempt to access Die.ourRandNumGen
        // would generate a syntax error

        Die d1 = new Die(10);

        System.out.println("Die.DEFAULT_SIDES "+Die.DEFAULT_SIDES);
        System.out.println("d1.DEFAULT_SIDES "+d1.DEFAULT_SIDES);

        // regardless of the number of Die objects in
        // existence, there is only one copy of DEFAULT_SIDES
        // in the Die class
    }
} // end of main method
} // end of UseDieStatic class
Static Methods

- `static` has a somewhat different meaning when used in a method declaration.
- Static methods may not manipulate any instance variables.
- In non-static methods, some object invokes the method `d3.roll();`.
- The object that makes the method call is an implicit parameter to the method.
Static Methods Continued

- Since there is no implicit object parameter sent to the static method it does not have access to a copy of any objects instance variables
  - unless of course that object is sent as an explicit parameter
- Static methods are normally utility methods or used to manipulate static variables (class variables)
- The Math and System classes are nothing but static methods
static and this

- Why does this work (added to Die class)

```java
public class Die {
    public void outputSelf() {
        System.out.println( this );
    }
}
```

- but this doesn't?

```java
public class StaticThis {
    public static void main(String[] args) {
        System.out.println( this );
    }
}
```
"Question: What is the object oriented way of getting rich?
Answer: Inheritance."
Features of OO Programming

- Encapsulation
  - abstraction
  - information hiding
  - breaking problem up based on data types

- Inheritance
  - code reuse
  - specialization
  - "New code using old code."
Encapsulation

- Create a program to allow people to play the game Monopoly
  - Create classes for money, dice, players, the bank, the board, chance cards, community chest cards, pieces, etc.

- Some classes use other classes:
  - the board consists of spaces
  - a player has money
  - a piece has a position
  - also referred to as composition
Inheritance

- Another kind of relationship exists between data types

- There are properties in Monopoly
  - a street is a kind of property
  - a railroad is a kind of property
  - a utility is a kind of property
In Monopoly there is the concept of a Property

All properties have some common traits
- they have a name
- they have a position on the board
- they can be owned by players
- they have a price to buy

But some things are different for each of the three kinds of property
- How to determine rent when another player lands on the Property
What to Do?

- If we have a separate class for Street, Railroad, and Utility there is going to be a lot of code copied
  - hard to maintain
  - an anti-pattern

- Inheritance is a programming feature to allow data types to build on pre-existing data types without repeating code
Inheritance in Java

- Java is designed to encourage object oriented programming
- all classes, except one, **must** inherit from exactly one other class
- The **Object** class is the *cosmic super class*
  - The **Object** class does not inherit from any other class
  - The **Object** class has several important methods: `toString`, `equals`, `hashCode`, `clone`, `getClass`

implications:
- all classes are descendants of **Object**
- all classes and thus all objects have a **toString**, `equals`, `hashCode`, `clone`, and `getClass` method
  - `toString`, `equals`, `hashCode`, `clone` normally overridden
Nomenclature of Inheritance

- In Java the `extends` keyword is used in the class header to specify which preexisting class a new class is inheriting from.
  
  ```java
  public class Student extends Person
  ```

- Person is said to be
  - the parent class of Student
  - the super class of Student
  - the base class of Student
  - an ancestor of Student

- Student is said to be
  - a child class of Person
  - a sub class of Person
  - a derived class of Person
  - a descendant of Person
Clicker Question 1

What is the primary reason for using inheritance when programming?

A. To make a program more complicated
B. To copy and paste code between classes
C. To reuse pre-existing code
D. To hide implementation details of a class
E. To ensure preconditions of methods are met.
Clicker Question 2

What is output when the `main` method is run?

```java
public class Foo {
    public static void main(String[] args) {
        Foo f1 = new Foo();
        System.out.println(f1.toString());
    }
}
```

A. 0
B. null
C. Unknown until code is actually run.
D. No output due to a syntax error.
E. No output due to a runtime error.
Simple Code Example

- Create a class named Shape
  - what class does Shape inherit from
  - what methods can we call on Shape objects?
  - add instance variables for a position
  - *override* the toString method

- Create a Circle class that extends Shape
  - add instance variable for radius
  - debug and look at contents
  - try to access instance var from Shape
  - constructor calls
  - use of key word *super*
Overriding methods

- any method that is not `final` may be overridden by a descendant class
- same signature as method in ancestor
- may not reduce visibility
- may use the original method if simply want to add more behavior to existing
Constructors

- Constructors handle initialization of objects
- When creating an object with one or more ancestors (every type except Object) a chain of constructor calls takes place
- The reserved word `super` may be used in a constructor to call a one of the parent's constructors
  – must be first line of constructor
- If no parent constructor is explicitly called the default, 0 parameter constructor of the parent is called
  – if no default constructor exists a syntax error results
- If a parent constructor is called another constructor in the same class may no be called
  – no `super();this();` allowed. One or the other, not both
  – good place for an initialization method
The Keyword `super`

- `super` is used to access something (any protected or public field or method) from the super class that has been overridden.
- Rectangle's `toString` makes use of the `toString` in `ClosedShape` my calling `super.toString()` without the super calling `toString` would result in infinite recursive calls.
- Java does not allow nested supers.
  ```java
  super.super.toString()
  ```
  results in a syntax error even though technically this refers to a valid method, `Object's toString`
- Rectangle *partially* overrides `ClosedShapes toString`
Creating a SortedIntList
A New Class

Assume we want to have a list of ints, but that the ints must always be maintained in ascending order:

\[-7, 12, 37, 212, 212, 313, 313, 500\]

- `sortedList.get(0)` returns the min
- `sortedList.get(list.size() - 1)` returns the max
Implementing `SortedIntList`

- Do we have to write a whole new class?
- Assume we have an `IntList` class.
- Which of the following methods would have to be changed?
  
  ```java
  public void add(int value)
  int get(int location)
  String toString()
  int size()
  int remove(int location)
  ```
Overriding the `add` Method

- First attempt
- Problem?
- solving with insert method
  - double edged sort
- solving with `protected`
  - What `protected` really means
Problems

- What about this method?
  ```java
  void insert(int location, int val)
  ```

- What about this method?
  ```java
  void insertAll(int location, IntList otherList)
  ```

- `SortedIntList` is not a good application of inheritance given the `IntList` we developed
More Example Code

ClosedShape and Rectangle classes
Shape Classes

- Declare a class called `ClosedShape`
  - assume all shapes have x and y coordinates
  - override `Object's version of toString`

- Possible sub classes of `ClosedShape`
  - Rectangle
  - Circle
  - Ellipse
  - Square

- Possible hierarchy
  
  
  `ClosedShape <- Rectangle <- Square`
public class ClosedShape
{
    private double myX;
    private double myY;

    public ClosedShape()
    {
        this(0,0);
    }

    public ClosedShape (double x, double y)
    {
        myX = x;
        myY = y;
    }

    public String toString()
    {
        return "x: " + getX() + " y: " + getY();
    }

    public double getX(){
        return myX;
    }
    public double getY(){
        return myY;
    }
}
// Other methods not shown
A Rectangle Constructor

public class Rectangle extends ClosedShape
{
    private double myWidth;
    private double myHeight;

    public Rectangle( double x, double y, double width, double height )
    {
        super(x, y);
        // calls the 2 double constructor in ClosedShape
        myWidth = width;
        myHeight = height;
    }

    // other methods not shown
}

Inheritance
public class Rectangle extends ClosedShape
{
    private double myWidth;
    private double myHeight;

    public Rectangle()
    {
        this(0, 0);
    }

    public Rectangle(double width, double height)
    {
        myWidth = width;
        myHeight = height;
    }

    public Rectangle(double x, double y,
                        double width, double height)
    {
        super(x, y);
        myWidth = width;
        myHeight = height;
    }

    public String toString()
    {
        return super.toString() + " width " + myWidth
            + " height " + myHeight;
    }
}
public class Rectangle extends ClosedShape
{
    private double myWidth;
    private double myHeight;

    public Rectangle()
    {
        init(0, 0);
    }

    public Rectangle(double width, double height)
    {
        init(width, height);
    }

    public Rectangle(double x, double y, double width, double height)
    {
        super(x, y);
        init(width, height);
    }

    private void init(double width, double height)
    {
        myWidth = width;
        myHeight = height;
    }
}
Result of Inheritance

Do any of these cause a syntax error? What is the output?

```java
Rectangle r = new Rectangle(1, 2, 3, 4);
ClosedShape s = new CloseShape(2, 3);
System.out.println( s.getX() );
System.out.println( s.getY() );
System.out.println( s.toString() );
System.out.println( r.getX() );
System.out.println( r.getY() );
System.out.println( r.toString() );
System.out.println( r.getWidth() );
```
### A Rectangle object

<table>
<thead>
<tr>
<th>The Real Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fields from Object class</strong></td>
</tr>
<tr>
<td>Instance variables declared in Object</td>
</tr>
</tbody>
</table>

| **Fields from ClosedShape class** |
| Instance Variables declared in ClosedShape |

| **Fields from Rectangle class** |
| Instance Variables declared in Rectangle |

Available methods are all methods from Object, ClosedShape, and Rectangle.
Access Modifiers and Inheritance

- **public**
  - accessible to all classes

- **private**
  - accessible only within that class. Hidden from all sub classes.

- **protected**
  - accessible by classes within the same package and all descendant classes

- **Instance variables** *should* be private

- protected methods are used to allow descendant classes to modify instance variables in ways other classes can't
Why private Vars and not protected?

- In general it is good practice to make instance variables private
  - hide them from your descendants
  - if you think descendants will need to access them or modify them provide protected methods to do this

- Why?

- Consider the following example
public class GamePiece
{
    private Board myBoard;
    private Position myPos;

    // whenever my position changes I must
    // update the board so it knows about the change

    protected void alterPos( Position newPos )
    {
        Position oldPos = myPos;
        myPos = newPos;
        myBoard.update( oldPos, myPos );
    }
}
"Inheritance is new code that reuses old code. Polymorphism is old code that reuses new code."
Polymorphism

- Another feature of OOP
- literally “having many forms”
- object variables in Java are *polymorphic*
- object variables can refer to objects or their declared type AND any objects that are descendants of the declared type

```java
Property p = new Property();
p = new Railroad(); // legal!
p = new Utility(); // legal!
p = new Street();
Object obj1; // = what?
```
Data Type

- object variables have:
  - a **declared type**. Also called the static type.
  - a **dynamic type**. What is the actual type of the pointee at run time or when a particular statement is executed.

- Method calls are syntactically legal if the method is in the declared type or any ancestor of the declared type.

- The actual method that is executed at runtime is based on the dynamic type
  - dynamic dispatch
Clicker Question 1

Consider the following class declarations:

```java
public class BoardSpace
public class Property extends BoardSpace
public class Street extends Property
public class Railroad extends Property
```

Which of the following statements would cause a syntax error? (Assume all classes have a zero argument constructor.)

A. Object obj = new Railroad();
B. Street s = new BoardSpace();
C. BoardSpace b = new Street();
D. Railroad r = new Street();
E. More than one of these
Method LookUp

- To determine if a method is legal the compiler looks in the class based on the declared type
  - if it finds it great, if not go to the super class and look there
  - continue until the method is found, or the Object class is reached and the method was never found. (Compile error)

- To determine which method is actually executed the runtime system:
  - starts with the actual run time class of the object that is calling the method
  - search the class for that method
  - if found, execute it, otherwise go to the super class and keep looking
  - repeat until a version is found

- Is it possible the runtime system won't find a method?
Clicker Question 2

What is output by the code to the right when run?

A. !!live
B. !eggegg
C. !egglive
D. !!!
E. eggegglive

```java
public class Animal{
    public String bt(){ return "!"; } 
}

public class Mammal extends Animal{
    public String bt(){ return "live"; } 
}

public class Platypus extends Mammal{
    public String bt(){ return "egg";} 
}

Animal a1 = new Animal();
Animal a2 = new Platypus();
Mammal m1 = new Platypus();
System.out.print( a1.bt() );
System.out.print( a2.bt() );
System.out.print( m1.bt() );
```
Why Bother?

- Inheritance allows programs to model relationships in the real world
  – if the program follows the model it may be easier to write
- Inheritance allows code reuse
  – complete programs faster (especially large programs)
- Polymorphism allows code reuse in another way
- Inheritance and polymorphism allow programmers to create *generic algorithms*
Genericity

- One of the goals of OOP is the support of code reuse to allow more efficient program development.
- If an algorithm is essentially the same, but the code would vary based on the data type, genericity allows only a single version of that code to exist.
- In Java, there are 2 ways of doing this:
  1. polymorphism and the inheritance requirement
  2. generics
A Generic List Class
We may find IntList useful, but what if we want a List of Strings? Rectangles? Lists?

– What if I am not sure?

Are the List algorithms different if I am storing Strings instead of ints?

How can we make a generic List class?
Generic List Class

- required changes
- How does `toString` have to change?
  - why?!?!
  - A good example of why keyword `this` is necessary from `toString`
- What can a `List` hold now?
- How many `List` classes do I need?
Writing an `equals` Method

- How to check if two objects are equal?
  
  ```java
  if(objA == objA)
      // does this work?
  ```

- Why not this
  
  ```java
  public boolean equals(List other)
  ```

- Because
  
  ```java
  public void foo(List a, Object b)
      if( a.equals(b) )
          System.out.println( same )
  ```

  - what if b is really a `List`?
**equals method**

- read the javadoc carefully!
- Must handle **null**
- Parameter must be **Object**
  - otherwise overloading instead of overriding
  - causes
- must handle cases when parameter is not same data type as calling object
  - `instanceof` or `getClass()`
- **do not rely on** `toString` and `String's equals`
public Object[] createASet(Object[] items)
{
    /*
     * pre: items != null, no elements of items = null
     * post: return an array of Objects that represents a set of the elements in items. (all duplicates removed)
     */

    {5, 1, 2, 3, 2, 3, 1, 5} -> {5, 1, 2, 3}
createASet examples

String[] sList = {"Texas", "texas", "Texas", "Texas", "UT", "texas"};
Object[] sSet = createASet(sList);
for(int i = 0; i < sSet.length; i++)
    System.out.println(sSet[i]);

Object[] list = {"Hi", 1, 4, 3.3, true,
                 new ArrayList(), "Hi", 3.3, 4};
Object[] set = createASet(list);
for(int i = 0; i < set.length; i++)
    System.out.println(set[i]);
"Get your data structures correct first, and the rest of the program will write itself."
- David Jones
Back to our Array Based List

- Started with a list of ints
- Don't want to have to write a new list class for every data type we want to store in lists
- Moved to an array of `Objects` to store the elements of the list

```java
// from array based list
private Object[] myCon;
```
Using Object

- In Java, all classes inherit from exactly one other class except Object which is at the top of the class hierarchy.
- Object variables can refer to objects of their declared type and any descendants – polymorphism.
- Thus, if the internal storage container is of type Object it can hold anything – primitives handled by wrapping them in objects.
  - int – Integer, char – Character.
Difficulties with Object

- *Creating* generic containers using the Object data type and polymorphism is relatively straightforward
- Using these generic containers leads to some difficulties
  - Casting
  - Type checking
- Code examples on the following slides
Clicker Question 1

What is output by the following code?

```java
GenericList list = new GenericList(); // 1
String name = "Olivia";
list.add(name); // 2
System.out.print(list.get(0).charAt(2)); // 3
```

A. i
B. No output due to syntax error at line // 1
C. No output due to syntax error at line // 2
D. No output due to syntax error at line // 3
E. No output due to runtime error.
Code Example - Casting

Assume a list class

```java
GenericList li = new GenericList();
li.add("Hi");
System.out.println( li.get(0).charAt(0) );
// previous line has syntax error
// return type of get is Object
// Object does not have a charAt method
// compiler relies on declared type
System.out.println(
    ((String)li.get(0)).charAt(0) );
// must cast to a String
```
Code Example – type checking

//pre: all elements of li are Strings
public void printFirstChar(GenericList li) {
    String temp;
    for(int i = 0; i < li.size(); i++) {
        temp = (String)li.get(i);
        if( temp.length() > 0 )
            System.out.println(
                temp.charAt(0) );
    }
}

// what happens if pre condition not met?
Too Generic?

Does this code compile?

```java
GenericList list = new GenericList();
list.add( "Olivia" );
list.add( new Integer(12) );
list.add( new Rectangle(1, 2, 3, 4) );
list.add( new GenericList() );
```

A. No
B. Yes
Is this a bug or a feature?
"Fixing" the Method

//pre: all elements of li are Strings
public void printFirstChar(GenericList li) {
    String temp;
    for(int i = 0; i < li.size(); i++) {
        if( li.get(i) instanceof String ){
            temp = (String)li.get(i);
            if( temp.length() > 0 )
                System.out.println(
                temp.charAt(0) );
        }
    }
}

CS314  Generics  10
Generic Types

- Java has syntax for *parameterized data types*
- Referred to as *Generic Types* in most of the literature
- A traditional parameter *has* a data type and can store various values just like a variable
  
  ```java
  public void foo(int x)
  ```
- Generic Types are like parameters, but the data type for the parameter is *data type*
  - like a variable that stores a data type
  - this is an abstraction. Actually, all data type info is erased at compile time
Making our Array List Generic

- Data type variables declared in class header

```java
public class GenericList<E> {
  // The <E> is the declaration of a data type parameter for the class
  // - any legal identifier: Foo, AnyType, Element, DataTypeThisListStores
  // - Sun style guide recommends terse identifiers

  // The value E stores will be filled in whenever a programmer creates a new

  GenericList<String> li = new GenericList<String>();
```
Modifications to GenericList

- instance variable
  
  ```java
  private E[] myCon;
  ```

- Parameters on
  - add, insert, remove, insertAll

- Return type on
  - get

- Changes to creation of internal storage container
  
  ```java
  myCon = (E[]) new Object[DEFAULT_SIZE];
  ```

- Constructor header does not change
Modifications to GenericList

- Careful with the equals method
- Recall type information is actually erased
- Use of wildcard
- Rely on the elements equals methods
Using Generic Types

Back to Java's ArrayList

ArrayList list1 = new ArrayList();
  – still allowed, a "raw" ArrayList
  – works just like our first pass at GenericList
  – casting, lack of type safety
Using Generic Types

ArrayList<String> list2 =
    new ArrayList<String>();

    – for list2 E stores String

list2.add( "Isabelle" );
System.out.println(
    list2.get(0).charAt(2) );  // ok
list2.add( new Rectangle() );
// syntax error
Parameters and Generic Types

- Old version
  ```java
  //pre: all elements of li are Strings
  public void printFirstChar(ArrayList li){
  ```

- New version
  ```java
  //pre: none
  public void printFirstChar(ArrayList<String> li){
  ```

- Elsewhere
  ```java
  ArrayList<String> list3 = new ArrayList<String>();
  printFirstChar( list3 ); // ok
  ArrayList<Integer> list4 = new ArrayList<Integer>();
  printFirstChar( list4 ); // syntax error
  ```
Generic Types and Subclasses

ArrayList<Shape> list5 =
    new ArrayList<Shape>();
list5.add( new Rectangle() );
list5.add( new Square() );
list5.add( new Circle() );
// all okay

- list5 can store Shape objects and any descendants of Shape
I once attended a Java user group meeting where James Gosling (one of Java's creators) was the featured speaker. During the memorable Q&A session, someone asked him: "If you could do Java over again, what would you change?" "I'd leave out classes," he replied. After the laughter died down, he explained that the real problem wasn't classes per se, but rather implementation inheritance (the extends relationship). Interface inheritance (the implements relationship) is preferable.

- Allen Holub
How Many Sorts?

How many sorts do you want to have to write?

```java
public static void SelSort(double[] list) {
    for(int i = 0; i < list.length - 1; i++) {
        int small = i;
        for(int j = i + 1; j < list.length; j++) {
            if( list[j] < list[small] )
                small = j;
        }
        double temp = list[i];
        list[i] = list[small];
        list[small] = temp;
    }
} // end of i loop
```
Why interfaces?

- Interfaces allow the creation of *abstract data types*
  - "A set of data values and associated operations that are precisely specified independent of any particular implementation."
  - multiple implementations allowed

- Interfaces allow a data type to be specified without worrying about the implementation
  - do design first
  - What will this data type do?
  - Don’t worry about implementation until design is done.
  - separation of concerns
  - allow us to create *generic algorithms*
 Interfaces

public interface List<E> {
  ‣ No constructors
  ‣ No instance variables
  ‣ abstract methods
  ‣ default methods
  ‣ static methods
  ‣ class constants (prefer enums)

    public static final int DEFAULT_CAP = 10;
    public void add(E val);
Implementing Interfaces

- A class inherits (extends) exactly one other class, but …
- A class can *implement* as many interfaces as it likes

```java
public class ArrayList implements List
```
- A class that implements an interface must provide implementations of all method declared in the interface or the class must be abstract
- interfaces can extend other interfaces
The Java Standard Library contains a number of interfaces – names are italicized in the class listing

One of the most important interfaces is the Comparable interface
Comparable Interface

```java
package java.lang;

public interface Comparable<T> {
    public int compareTo(T other);
}
```

- `compareTo` must return
  - an int <0 if the calling object is less than the parameter,
  - 0 if they are equal
  - an int >0 if the calling object is greater than the parameter

- `compareTo` should be consistent with `equals` but this isn't required.
"Use interfaces to ensure a class has methods that other classes or methods will use."

– Anthony, Spring 2013

The other classes or methods are already done.

The other methods or classes call interface type methods

POLYMORPHISM

– old code using new code
What is output by the following code?

```java
Comparable c1 = new Comparable();
Comparable c2 = new Comparable();
System.out.println(c1.compareTo(c2));
```

A. A value < 0  
B. 0  
C. A value > 0  
D. Unknown until program run  
E. Other
Example `compareTo`

- Suppose we have a class to model playing cards
  - Ace of Spades, King of Hearts, Two of Clubs
- each card has a suit and a value, represented by ints
- this version of `compareTo` will compare values first and then break ties with suits
compareTo in a Card class

```java
public class Card implements Comparable<Card> {
    public int compareTo(Card otherCard) {
        return this.rank - other.rank;
    }
    // other methods not shown
}
```

Assume ints for ranks (2, 3, 4, 5, 6,...) and suits (0 is clubs, 1 is diamonds, 2 is hearts, 3 is spades).
Interfaces and Polymorphism

- Interfaces may be used as the data type for object variables
- Can’t simply create objects of that type
- Can refer to any objects that implement the interface or descendants

Assume `Card` implements `Comparable`

```java
Card c = new Card();
Comparable comp1 = new Card();
Comparable comp2 = c;
```
Clicker Question 1

Which of the following lines of code causes a syntax error?

Comparable c1; // A
cl = new Comparable(); // B
Comparable c2 = "Kelly"; // C
int x = c2.compareTo("Ann"); // D
// E more than one of A - D

// what is x after statement?
Why Make More Work?

- Why bother implementing an interface such as Comparable
  - objects can use method that expect an interface type

Example if I implement Comparable:

```java
Arrays.sort(Object[] a)
public static void sort(Object[] a)
```

All elements in the array must implement the Comparable interface. Furthermore, all elements in the array must be *mutually comparable*

- objects of my type can be stored in data structures that accept Comparables
A List Interface

- What if we wanted to specify the operations for a List, but no implementation?
- Allow for multiple, different implementations.
- Provides a way of creating abstractions.
  - a central idea of computer science and programming.
  - specify "what" without specifying "how"
  - "Abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time."
List Interface

```java
public interface List<E> {
    public void add(E val);
    public int size();
    public E get(int location);
    public E remove(int location);
    public void insert(int location, E val);
}
```
public static void sort(Comparable[] list) {
    final int LIMIT = list.length - 1;
    for(int i = 0; i < LIMIT; i++) {
        int small = i;
        for(int j = i + 1; j < list.length; j++) {
            int d = list[j].compareTo(list[small]);
            if( d < 0)
                small = j;
        }
        Comparable temp = list[i];
        list[i] = list[small];
        list[small] = temp;
    } // end of i loop
}
"First things first, but not necessarily in that order"

-Dr. Who
Iterators

- ArrayList is part of the Java Collections Framework
- Collection is an interface that specifies the basic operations every collection (data structure) should have
- Some Collections don’t have a definite order
  - Sets, Maps, Graphs
- How to access all the items in a Collection with no specified order?
An iterator object is a “one shot” object
– it is designed to go through all the elements of a Collection once
– if you want to go through the elements of a Collection again you have to get another iterator object

Iterators are obtained by calling a method from the Collection
The Iterator interface specifies 3 methods:

- `boolean hasNext()`  
  //returns true if this iteration has more elements

- `E next()`  
  //returns the next element in this iteration
  //pre: hasNext()

- `void remove()`  
  /*Removes from the underlying collection the last element returned by the iterator.  
  pre: This method can be called only once per call to next.  
  After calling, must call next again before calling remove again.  
  */
Clicker Question 1

Which of the following produces a syntax error?

```java
ArrayList<String> list = new ArrayList<String>();
Iterator<String> it1 = new Iterator(); // I
Iterator<String> it2 = new Iterator(list); // II
Iterator<String> it3 = list.iterator(); // III
```

A. I
B. II
C. III
D. I and II
E. II and III
Imagine a fence made up of fence posts and rail sections.
Fence Analogy

- The iterator lives on the fence posts
- The data in the collection are the rails
- Iterator created at the far left post
- As long as a rail exists to the right of the Iterator, hasNext() is true

Iterator object
Fence Analogy

ArrayList<String> names =
    new ArrayList<String>();

names.add(“Jan”);
names.add(“Levi”);
names.add(“Tom”);
names.add(“Jose”);
Iterator<String> it = names.iterator();
int i = 0;
Fence Analogy

```java
while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}
```

// when i == 1, prints out Jan

first call to next moves iterator to next post and returns “Jan”
Fence Analogy

while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}

// when i == 2, prints out Levi
Fence Analogy

while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}

// when i == 3, prints out Tom
Fence Analogy

while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}
// when i == 4, prints out Jose
Fence Analogy

while( it.hasNext() ) {
    i++;
    System.out.println( it.next() );
}

// call to hasNext returns false
// while loop stops

"Jan"  "Levi"  "Tom"  "Jose"
Typical Iterator Pattern

```java
public void printAll(Collection<String> list) {
    Iterator<String> it = list.iterator();
    while (it.hasNext()) {
        String temp = it.next();
        System.out.println(temp);
    }
}
```
Clicker Question 2

What is output by the following code?

ArrayList<Integer> list;
list = new ArrayList<Integer>();
list.add(3);
list.add(3);
list.add(5);
Iterator<Integer> it = list.iterator();
System.out.print(it.next() + " ");
System.out.print(it.next() + " ");
System.out.print(it.next());

A. 3  
B. 3 5  
C. 3 3 5  
D. 3 3  
E. 3 3 then a runtime error
**remove method**

- An Iterator can be used to remove things from the Collection.
- Can only be called once per call to `next()`.

```java
public void removeWordsOfLength(int len) {
    Iterator<String> it = myList.iterator
    while( it.hasNext() ) {
        String temp = it.next();
        if(temp.length() == len)
            it.remove();
    }
}
```

// original list = ["dog", "cat", "hat", "sat"]
// resulting list after `removeWordsOfLength(3)` ?
Clicker Question 3

```java
public void printTarget(ArrayList<String> names, int len) {
    Iterator<String> it = names.iterator();
    while (it.hasNext())
        if (it.next().length() == len)
            System.out.println(it.next());
}
```

Given names = [“Jan”, “Ivan”, “Tom”, “George”] and len = 3 what is output by the printTarget method?

A. Jan Ivan Tom George
B. Jan Tom
C. Ivan George
D. No output due to syntax error
E. No output due to runtime error
The Iterable Interface

- A related interface is `Iterable`
- One method in the interface:
  ```java
  public Iterator<T> iterator()
  ```
- Why?
- Anything that implements the `Iterable` interface can be used in the for each loop.

```java
ArrayList<Integer> list;
// code to create and fill list
int total = 0;
for( int x : list )
  total += x;
```
Iterators

If you simply want to go through all the elements of a Collection (or Iterable thing) use the for each loop
– hides creation of the Iterator

```java
public void printAllOfLength(ArrayList<String> names, int len){
    //pre: names != null, names only contains Strings
    //post: print out all elements of names equal in
    //length to len
    for(String s : names){
        if( s.length() == len )
            System.out.println( s );
    }
}
```
Implementing an Iterator

- Implement an Iterator for our GenericList class
  - Nested Classes
  - Inner Classes
  - Example of encapsulation
  - checking precondition on remove
  - does our GenericList need an Iterator?
Comodification

- If a Collection (ArrayList) is changed while an iteration via an iterator is in progress an Exception will be thrown the next time the next() or remove() methods are called via the iterator

```java
ArrayList<String> names = new ArrayList<String>();
names.add("Jan");
Iterator<String> it = names.iterator();
names.add("Andy");
it.next(); // exception will occur here
```
"He's off the map!"

- Stan (Mark Ruffalo) *Eternal Sunshine of the Spotless Mind*
Data Structures

- More than arrays and lists
- Write a program to count the frequency of all the words in a file.
- Make a simplification: assume words are anything set off by whitespace
## Performance using ArrayList

<table>
<thead>
<tr>
<th>Title</th>
<th>Size (kb)</th>
<th>Total Words</th>
<th>Distinct Words</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>small sample</td>
<td>0.6</td>
<td>89</td>
<td>25</td>
<td>0.001</td>
</tr>
<tr>
<td>2BR02B</td>
<td>34</td>
<td>5,638</td>
<td>1,975</td>
<td>0.051</td>
</tr>
<tr>
<td>Alice in Wonderland</td>
<td>120</td>
<td>29,460</td>
<td>6,017</td>
<td>0.741</td>
</tr>
<tr>
<td>Adventures of Sherlock Holmes</td>
<td>581</td>
<td>107,533</td>
<td>15,213</td>
<td>4.144</td>
</tr>
<tr>
<td>2008 CIA Factbook</td>
<td>10,030</td>
<td>1,330,100</td>
<td>74,042</td>
<td>173.000</td>
</tr>
</tbody>
</table>
Order?

- Express change in size as factor of previous file

<table>
<thead>
<tr>
<th>Title</th>
<th>Size</th>
<th>Total Words</th>
<th>Distinct Words</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>small sample</td>
<td>0.6</td>
<td>89</td>
<td>25</td>
<td>0.001</td>
</tr>
<tr>
<td>2BR02B</td>
<td>57x</td>
<td>63x</td>
<td>79x</td>
<td>51x</td>
</tr>
<tr>
<td>Alice in Wonderland</td>
<td>3.5x</td>
<td>5.2x</td>
<td>3.0x</td>
<td>14.5x</td>
</tr>
<tr>
<td>Adventures of Sherlock Holmes</td>
<td>4.8x</td>
<td>3.7x</td>
<td>2.5x</td>
<td>6.0x</td>
</tr>
<tr>
<td>2008 CIA Factbook</td>
<td>17x</td>
<td>12.3x</td>
<td>5x</td>
<td>42x</td>
</tr>
</tbody>
</table>

\[ O(\text{Total Words} \times \text{Distinct Words}) \] ??
Clicker Question

Given 3 minutes for the 2008 CIA Factbook with 1,330,100 total words and 74,042 distinct words, how long for 1,000x total words and 100x distinct words?

A. an hour
B. a day
C. a week
D. a month
E. half a year
Why So Slow

- Write a contains method for an array based list

```java
public boolean contains(E target) {
```
A Faster Way - Maps

Also known as:
- table, search table, dictionary, associative array, or associative container

A data structure optimized for a very specific kind of search / access

In a *map* we access by asking "give me the *value* associated with this *key.*"

Recall, in the ArrayList example we did not count the number of occurrences of each word
Keys and Values

Dictionary Analogy:
- The key in a dictionary is a word: foo
- The value in a dictionary is the definition: First on the standard list of metasyntactic variables used in syntax examples

A key and its associated value form a pair that is stored in a map

To retrieve a value the key for that value must be supplied
- A List can be viewed as a Map with integer keys
More on Keys and Values

- Keys must be unique, meaning a given key can only represent one value
  - but one value may be represented by multiple keys
  - like synonyms in the dictionary.

Example:

*factor*: *n.* See *coefficient of X*

- *factor* is a key associated with the same value (definition) as the key *coefficient of X*
The Map<K, V> Interface in Java

- void clear()
  - Removes all mappings from this map (optional operation).
- boolean containsKey(Object key)
  - Returns true if this map contains a mapping for the specified key.
- boolean containsValue(Object value)
  - Returns true if this map maps one or more keys to the specified value.
- Set<K> keySet()
  - Returns a Set view of the keys contained in this map.
The Map Interface Continued

- $V \text{ get}(\text{Object key})$
  - Returns the value to which this map maps the specified key.

- boolean isEmpty()
  - Returns true if this map contains no key-value mappings.

- $V \text{ put}(K \text{ key, V value})$
  - Associates the specified value with the specified key in this map
The Map Interface Continued

- `V remove(Object key)`
  - Removes the mapping for this key from this map if it is present

- `int size()`
  - Returns the number of key-value mappings in this map.

- `Collection<V> values()`
  - Returns a collection view of the values contained in this map.
## Results with HashMap

<table>
<thead>
<tr>
<th>Title</th>
<th>Size (kb)</th>
<th>Total Words</th>
<th>Distinct Words</th>
<th>Time List</th>
<th>Time Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>small sample</td>
<td>0.6</td>
<td>89</td>
<td>25</td>
<td>0.001</td>
<td>0.0008</td>
</tr>
<tr>
<td>2BR02B</td>
<td>34</td>
<td>5,638</td>
<td>1,975</td>
<td>0.051</td>
<td>0.0140</td>
</tr>
<tr>
<td>Alice in Wonderland</td>
<td>120</td>
<td>29,460</td>
<td>6,017</td>
<td>0.741</td>
<td>0.0720</td>
</tr>
<tr>
<td>Adventures of Sherlock Holmes</td>
<td>581</td>
<td>107,533</td>
<td>15,213</td>
<td>4.144</td>
<td>0.2500</td>
</tr>
<tr>
<td>2008 CIA Factbook</td>
<td>10,030</td>
<td>1,330,100</td>
<td>74,042</td>
<td>173.000</td>
<td>4.0000</td>
</tr>
<tr>
<td>Title</td>
<td>Size</td>
<td>Total Words</td>
<td>Distinct Words</td>
<td>Time List</td>
<td>Time Map</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------</td>
<td>-------------</td>
<td>----------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>small sample</td>
<td>0.6</td>
<td>89</td>
<td>25</td>
<td>0.001</td>
<td>0.0008</td>
</tr>
<tr>
<td>2BR02B</td>
<td>57x</td>
<td>63x</td>
<td>79x</td>
<td>51x</td>
<td>18x</td>
</tr>
<tr>
<td>Alice in Wonderland</td>
<td>3.5x</td>
<td>5.2x</td>
<td>3.0x</td>
<td>14.5x</td>
<td>5x</td>
</tr>
<tr>
<td>Adventures of Sherlock Holmes</td>
<td>4.8x</td>
<td>3.7x</td>
<td>2.5x</td>
<td>5.6x</td>
<td>3.5x</td>
</tr>
<tr>
<td>2008 CIA Factbook</td>
<td>17x</td>
<td>12.3x</td>
<td>5x</td>
<td>42x</td>
<td>16x</td>
</tr>
</tbody>
</table>

O(Total Words)?
Topic 10
Abstract Classes

“I prefer Agassiz in the abstract, rather than in the concrete.”
Back to the Property Example

- There are properties on a monopoly board
- Railroads, Utilities, and Streets are kinds of properties
- One behavior we want in Property is the getRent method
- problem: How do I get the rent of something that is “just a Property”?
The Property class

```java
public class Property {
    private int cost;
    private String name;

    public int getRent() {
        return hmmmmm??????;
    }
}
```

Doesn’t seem like we have enough information to get the rent if all we know is it is a Property.
Potential Solutions

1. Just leave it for the sub classes.
   ‣ Have each sub class define getRent()

2. Define getRent() in Property and simply return -1.
   ‣ Sub classes override the method with more meaningful behavior.
Leave it to the Sub - Classes

// no getRent() in Property
// Railroad and Utility DO have getRent() methods

public void printRents(Property[] props) {
    for(Property p : props)
        System.out.println(p.getRent());
}

Property[] props = new Property[2];
props[0] = new Railroad("NP", 200, 1);
props[1]
    = new Utility("Electric", 150, false);
printRents(props);

What is result of above code?
A. 200150
B. different every time
C. Syntax error
D. Class Cast Exception
E. Null Pointer Exception
Fix by Casting

// no getRent() in Property
public void printRents(Property[] props) {
    for (Property p : props) {
        if (p instanceof Railroad)
            System.out.println(((Railroad) p).getRent());
        else if (p instanceof Utility)
            System.out.println(((Utility) p).getRent());
        else
            System.out.println(((Street) p).getRent());
    }
}

Property[] props = new Property[2];
props[0] = new Railroad("NP", 200, 1);
props[1] = new Utility("Electric", 150, false);
printRents(props);

What happens as we add more sub classes of Property?

What happens if one of the objects is just a Property?
Fix with Dummy Method

// getRent() in Property returns -1

public void printRents(Property[] props) {
    for(Property p : props)
        System.out.println(p.getRent());
}

Property[] props = new Property[2];
props[0] = new Railroad("NP", 200, 1);
props[1] = new Utility("Electric", 150, false);
printRents(props);

What happens if sub classes don’t override getRent()?

Is that a good answer?
A Better Fix

- We know we want to be able to find the rent of objects that are instances of `Property`.
- The problem is we don’t know how to do that if all we know is it a `Property`.
- Make `getRent` an abstract method.
- Java keyword.
Making getRent Abstract

public class Property {

    private int cost;
    private String name;

    public abstract int getRent();
    // I know I want it.
    // Just don’t know how, yet...

}

Methods that are declared abstract have no body an undefined behavior.

All methods in a Java interface are abstract.
Problems with Abstract Methods

Given `getRent()` is now an abstract method what is wrong with the following code?

```java
Property s = new Property();
System.out.println(s.getRent());
```
Undefined Behavior = Bad

- Not good to have undefined behaviors
- If a class has 1 or more abstract methods, the class must also be declared abstract.
  - version of Property shown would cause a compile error
- Even if a class has zero abstract methods a programmer can still choose to make it abstract
  - if it models some abstract thing
  - is there anything that is just a “Mammal”?
public abstract class Property {

    private int cost;
    private String name;

    public abstract double getRent();
    // I know I want it.
    // Just don’t know how, yet...

} // Other methods not shown

If a class is abstract the compiler will not allow constructors of that class to be called
Property s = new Property(1, 2);
//syntax error
Abstract Classes

- In other words you can’t create instances of objects where the lowest or most specific class type is an abstract class.
- Prevents having an object with an undefined behavior.
- Why would you still want to have constructors in an abstract class?
- Object variables of classes that are abstract types may still be declared:
  ```
  Property s; //okay
  ```
Sub Classes of Abstract Classes

- Classes that extend an abstract class must provided a working version of any abstract methods from the parent class
  - or they must be declared to be abstract as well
  - could still decide to keep a class abstract regardless of status of abstract methods
Implementing getRent()

```java
public class Railroad extends Property {

    private static int[] rents = {25, 50, 10, 200};

    private int numOtherRailroadsOwned;

    public double getRent() {
        return rents[numOtherRailroadsOwned];
    }

    // other methods not shown
}
```
public class Utility extends Property {

    private static final int ONE_UtilIty_RENT = 4;
    private static final int TWO_UtilIty_RENT = 10;

    private boolean ownOtherUtility;

    public Utility(String n, int c, boolean other) {
        super(n, c);
    }

    public String toString() {
        return "Utility. own other utility? " + ownOtherUtility;
    }

    public int getRent(int roll) {
        return ownOtherUtility ? roll * TWO_UtilIty_RENT :
                             roll * TWO_UtilIty_RENT;
    }
}
Polymorphism in Action

// getRent() in Property is abstract

class Property {
    abstract double getRent();

    void printRents(Property[] props) {
        for(Property p : props)
            System.out.println(p.getRent());
    }
}

• Add the Street class. What needs to change in printRents method?
• Inheritance is can be described as new code using old code.
• Polymorphism can be described as old code using new code.
Comparable in Property

public abstract class Property
    implements Comparable<Property> {
    private int cost;
    private String name;

    public abstract int getRent();

    public int compareTo(Property other) {
        return this.getRent() - other.getRent();
    }
}
We suggested having a list interface

```java
public interface IList<E> extends Iterable<E> {
    public void add(E value);
    public int size();
    public E get(int location);
    public E remove(int location);
    public boolean contains(E value);
    public void addAll(IList<E> other);
    public boolean containsAll(IList<E> other);
}
```
Data Structures

When implementing data structures:

- Specify an interface
- Create an abstract class that is \textit{skeletal implementation} interface
- Create classes that extend the skeletal interface
"All the kids who did great in high school writing pong games in BASIC for their Apple II would get to college, take CompSci 101, a data structures course, and when they hit the pointers business their brains would just totally explode, and the next thing you knew, they were majoring in Political Science because law school seemed like a better idea."

-Joel Spolsky

Thanks to Don Slater of CMU for use of his slides.
What is output by the following code?

```java
ArrayList<Integer> a1 = new ArrayList<Integer>();
ArrayList<Integer> a2 = new ArrayList<Integer>();
a1.add(12);
a2.add(12);
System.out.println( a1 == a2 );
```

A. No output due to syntax error  
B. No output due to runtime error  
C. false  
D. true
Dynamic Data Structures

- **Dynamic** data structures
  - They grow and shrink one element at a time, normally without some of the inefficiencies of arrays
  - as opposed to a static container such as an array

- **Big O of Array Manipulations**
  - Access the kth element
  - Add or delete an element in the middle of the array while maintaining relative order
  - adding element at the end of array? space avail? no space avail?
  - add element at beginning of an array
Object References

- Recall that an *object reference* is a variable that stores the address of an object.
- A reference can also be called a *pointer*.
- They are often depicted graphically:
References as Links

- Object references can be used to create *links* between objects.

- Suppose a *Student* class contained a reference to another *Student* object.

```
John Smith
40725
3.57

Jane Jones
58821
3.72
```
References as Links

- References can be used to create a variety of linked structures, such as a linked list:

```
studentList
```

```
+-----+-----+-----+-----+
|     |     |     |   . |
+-----+-----+-----+-----+
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
+-----+-----+-----+-----+
```

```
A **linear** collection of self-referential objects, called nodes, connected by other links

- **linear**: for every node in the list, there is one and only one node that precedes it (except for possibly the first node, which may have no predecessor,) and there is one and only one node that succeeds it, (except for possibly the last node, which may have no successor)

- **self-referential**: a node that has the ability to refer to another node of the same type, or even to refer to itself

- **node**: contains data of any type, including a reference to another node of the same data type, or to nodes of different data types

- Usually a list will have a beginning and an end; the first element in the list is accessed by a reference to that class, and the last node in the list will have a reference that is set to `null`
Advantages of linked lists

- Linked lists are dynamic, they can grow or shrink as necessary.

- Linked lists are *non-contiguous*; the logical sequence of items in the structure is decoupled from any physical ordering in memory.
Nodes and Lists

- A different way of implementing a list
- Each element of a Linked List is a separate Node object.
- Each Node tracks a single piece of data plus a reference (pointer) to the next
- Create a new Node very time we add something to the List
- Remove nodes when item removed from list and allow garbage collector to reclaim that memory
A Node Class

```java
public class Node<E> {
    private E myData;
    private Node<E> myNext;

    public Node() {
        myData = null; myNext = null;  
    }

    public Node(E data, Node<E> next) {
        myData = data; myNext = next;  
    }

    public E getData() {
        return myData;  
    }

    public Node<E> getNext() {
        return myNext;  
    }

    public void setData(E data) {
        myData = data;  
    }

    public void setNext(Node<E> next) {
        myNext = next;  
    }
}
```
One Implementation of a Linked List

- The Nodes show on the previous slide are *singly linked*
  - a node refers only to the next node in the structure
  - it is also possible to have *doubly linked* nodes.
  - The node has a reference to the next node in the structure and the *previous* node in the structure as well

- How is the end of the list indicated
  - myNext = null for last node
  - a separate dummy node class / object
public class LinkedList<E> implements IList<E>
    private Node<E> head;
    private Node<E> tail;
    private int size;

    public LinkedList(){
        head = null;
        tail = null;
        size = 0;
    }
}

LinkedList<String> list = new LinkedList<String>();

<table>
<thead>
<tr>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>myHead</td>
</tr>
<tr>
<td>myTail</td>
</tr>
<tr>
<td>iMySize</td>
</tr>
</tbody>
</table>
Writing Methods

- When trying to code methods for Linked Lists *draw pictures*!
  - If you don't draw pictures of what you are trying to do it is very easy to make mistakes!
add method

- add to the end of list
- special case if empty
- steps on following slides
- public void add(E obj)
Add Element - List Empty (Before)

head: null

Object

item

tail: null

size: 0
Add Element - List Empty (After)

head

String

Node

myData

myNext

tail

null

size

1
Add Element - List Not Empty (Before)

head

Node

- myData
- myNext

- String

- size

- 1

- tail

- String

- item
Add Element - List Not Empty (After)

head

[Diagram showing the structure of a linked list with a head node and a tail node. The head node contains a size of 2.]

Node

<table>
<thead>
<tr>
<th>myData</th>
<th>myNext</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>null</td>
</tr>
</tbody>
</table>

Node

<table>
<thead>
<tr>
<th>myData</th>
<th>myNext</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
Code for default add

- public void add(E obj)
What is the worst case Big O for adding to the end of an array based list and a linked list? The lists already contain N items.

**Array based** | **Linked**
--- | ---
A. O(1) | O(1)
B. O(N) | O(N)
C. O(logN) | O(1)
D. O(1) | O(N)
E. O(N) | O(1)
Code for addFront

- add to front of list
- public void addFront(E obj)
- How does this compare to adding at the front of an array based list?
Clicker Question 3

What is the Big O for adding to the front of an array based list and a linked list? The lists already contain N items.

<table>
<thead>
<tr>
<th>Array based</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>B. O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>C. O(logN)</td>
<td>O(1)</td>
</tr>
<tr>
<td>D. O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>E. O(N)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>
Code for Insert

- public void insert(int pos, E obj)
- Must be careful not to break the chain!
- Where do we need to go?
- Special cases?
Clicker Question 4

What is the Big O for inserting an element into the middle of an array based list and into the middle of a linked list? Each list already contains N items.

<table>
<thead>
<tr>
<th>Array based</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>B. O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>C. O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>D. O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>E. O(N)</td>
<td>O(logN)</td>
</tr>
</tbody>
</table>
Clicker Question 5

- What is the Big O for getting an element based on position from an array based list and from a linked list? Each list contains N items. In other words: \( E \text{ get(position)} \)

<table>
<thead>
<tr>
<th>Array based</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. ( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>B. ( O(1) )</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>C. ( O(N) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>D. ( O(\log N) )</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>E. ( O(N) )</td>
<td>( O(N) )</td>
</tr>
</tbody>
</table>
Code for get

- public E get(int pos)
- The downside of Linked Lists
Code for remove

- public E remove(int pos)
Why Use Linked List

- What operations with a Linked List faster than the version from ArrayList?
Getting All Elements in Order From a Linked Lists

What is the Order (Big O) of the following code?

```java
LinkedList<Integer> list;
list = new LinkedList<Integer>();
// code to fill list with N elements

//Big O of following code?
for(int i = 0; i < list.size(); i++)
    System.out.println(list.get(i));
```

A. O(N)  B. O(2^N)  C. O(NlogN)
D. O(N^2)  E. O(N^3)
Other Possible Features of Linked Lists

- Doubly Linked
- Circular
- Dummy Nodes for first and last node in list

```java
public class DLNode<E> {
    private E myData;
    private DLNode<E> myNext;
    private DLNode<E> myPrevious;
}
```
Dummy Nodes

- Use of Dummy Nodes for a Doubly Linked List removes most special cases
- Also could make the Double Linked List circular
Doubly Linked List add

- public void add(E obj)
Insert for Doubly Linked List

- public void insert(int pos, E obj)
"To a man with a hammer, everything looks like a nail"

-Mark Twain
Underneath the Hood.
The Program Stack

When you invoke a method in your code what happens when that method is done?

```java
public static void start() {
    int x = 5;
    int y = -5;
    Point pt = new Point(x, y);
    pt.scale(2);
    String s = pt.toString();
}
```
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void scale(int v) {
        x *= v;
        y *= v;
    }

    public int getX() { return x; }
    public int getY() { return y; }

    public String toString() {
        return "x: " + getX() + ", y: " + getY();
    }
}
The Program Stack

- When your program is run on a processor, the commands are converted into another set of instructions and assigned memory locations.
  - normally a great deal of expansion takes place

```java
public static void start() {
    int x = 5; // 50
    int y = -5; // 51
    Point pt = new Point(x, y); // 52
    pt.scale(2); // 53
    String s = pt.toString(); // 54
}
```
Basic CPU Operations

- A CPU works via a fetch command / execute command loop and a program counter
- Instructions stored in memory (Instructions are data!)

```java
int x = 5;
int y = -5;
Point pt = new Point(x, y);
pt.scale(2);
String s = pt.toString();
```

- What if the first instruction of the scale method is stored at memory location 103?
More on the Program Stack

50    int x = 5;
51    int y = -5;
52    Point pt = new Point(x, y);
53    pt.scale(2);
54    String s = pt.toString();

• Instruction 53 is really saying *jump to instruction 103 with pt as the implicit parameter and 2 as the explicit parameter*

• In general when method scale is done what happens?

A. Program ends  B. goes to instruction 54  C. Goes back to whatever method called it
Activation Records and the Program Stack

- When a method is invoked all the relevant information about the current method (variables, values of variables, next line of code to be executed) is placed in an activation record.

- The activation record is pushed onto the program stack.

- A stack is a data structure with a single access point, the top.
The Program Stack

- Data may either be added (*pushed*) or removed (*popped*) from a stack but it is always from the top.
  - A stack of dishes
  - which dish do we have easy access to?
Using Recursion
A Problem

- Write a method that determines how much space is taken up by the files in a directory
- A directory can contain files and directories
- How many directories does our code have to examine?
- How would you add up the space taken up by the files in a single directory
  - Hint: don't worry about any subdirectories at first
How many levels of directories have to be visited?

A. 0

B. Unknown

C. Infinite

D. 1

E. 8
Java File Class

- **File**(String pathname) Creates a new File instance by converting the given pathname.
- **boolean isDirectory()** Tests whether the file denoted by this abstract pathname is a directory.
- **File[] listFiles()** Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
Code for `getDirectorySpace()`

```java
// assert dir is a directory and dir != null
public static long spaceUsed(File dir) {
    assert dir != null && dir.isDirectory();
    long spaceUsed = 0;
    File[] subFilesAndDirs = dir.listFiles();
    if(subFilesAndDirs != null)
        for(File sub : subFilesAndDirs)
            if(sub != null)
                if(!sub.isDirectory()) // sub is a plain old file
                    spaceUsed += sub.length();
                else // else sub is a directory
                    spaceUsed += spaceUsed(sub);
    return spaceUsed;
}
```
Attended Question 3

- Is it possible to write a non-recursive method to do this (determine space taken up by files in a directory, including its subdirectories, and their subdirectories, and their subdirectories, and so forth)?

A. Yes
B. No
Iterative `getDirectorySpace()`

```java
public int getDirectorySpace(Directory d)
{
    ArrayList dirs = new ArrayList();
    File[] fileList;
    Directory[] dirList;
    dirs.add(d);
    Directory temp;
    int total = 0;
    while( ! dirs.isEmpty() )
    {
        temp = (Directory)dirs.remove(0);
        fileList = temp.getFiles();
        for(int i = 0; i < fileList.length; i++)
            total += fileList[i].getSize();
        dirList = temp.getSubdirectories();
        for(int i =0; i < dirList.length; i++)
            dirs.add( dirList[i] );
    }
    return total;
}
```
Wisdom for Writing Recursive Methods
The 3 plus 1 rules of Recursion

1. Know when to stop
2. Decide how to take one step
3. Break the journey down into that step and a smaller journey
4. Have faith

From *Common Lisp: A Gentle Introduction to Symbolic Computation* by David Touretzky
Writing Recursive Methods

- Rules of Recursion
  1. Base Case: Always have at least one case that can be solved without using recursion.
  2. Make Progress: Any recursive call must progress toward a base case.
  3. "You gotta believe." Always assume that the recursive call works. (Of course you will have to design it and test it to see if it works or prove that it always works.)

A recursive solution solves a small part of the problem and leaves the rest of the problem in the same form as the original.
the classic first recursion problem / example

\[ N! \]

5! = 5 \times 4 \times 3 \times 2 \times 1 = 120

```c
int res = 1;
for(int i = 2; i <= n; i++)
    res *= i;
```
Factorial Recursively

- Mathematical Definition of Factorial

\[ 0! = 1 \]
\[ N! = N \times (N - 1)! \]

The definition is recursive.

```java
// pre n >= 0
public int fact(int n) {
    if(n == 0)
        return 1;
    else
        return n * fact(n-1);
}
```
Tracing Fact With the Program Stack

System.out.println( fact(4) );
Calling fact with 4

System.out.println( fact(4) );

partial result = n * fact(n-1)

n 4 in method fact
Calling fact with 3

\[
\begin{align*}
\text{top} & \quad n = 3 \quad \text{in method fact} \\
\text{partial result} & = n \times \text{fact}(n-1) \\
\hline
n & = 4 \quad \text{in method fact} \\
\text{partial result} & = n \times \text{fact}(n-1) \\
\hline
\text{System.out.println( fact(4) );}
\end{align*}
\]
Calling fact with 2

System.out.println( fact(4) );

Top

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>partial result = n * fact(n-1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>3</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>partial result = n * fact(n-1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>4</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>partial result = n * fact(n-1)</td>
</tr>
</tbody>
</table>

System.out.println( fact(4) );
Calling fact with 1

\[
\begin{array}{|c|}
\hline
n & 1 \\
\hline
\end{array}
\]

in method fact

partial result = n * fact(n-1)

\[
\begin{array}{|c|}
\hline
n & 2 \\
\hline
\end{array}
\]

in method fact

partial result = n * fact(n-1)

\[
\begin{array}{|c|}
\hline
n & 3 \\
\hline
\end{array}
\]

in method fact

partial result = n * fact(n-1)

\[
\begin{array}{|c|}
\hline
n & 4 \\
\hline
\end{array}
\]

in method fact

partial result = n * fact(n-1)

System.out.println( fact(4) );
Calling fact with 0 and returning 1

<table>
<thead>
<tr>
<th>n</th>
<th>in method fact</th>
<th>partial result = n * fact(n-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>in method fact</td>
<td>returning 1 to whatever method called me</td>
</tr>
<tr>
<td>1</td>
<td>in method fact</td>
<td>partial result = n * fact(n-1)</td>
</tr>
<tr>
<td>2</td>
<td>in method fact</td>
<td>partial result = n * fact(n-1)</td>
</tr>
<tr>
<td>3</td>
<td>in method fact</td>
<td>partial result = n * fact(n-1)</td>
</tr>
<tr>
<td>4</td>
<td>in method fact</td>
<td>partial result = n * fact(n-1)</td>
</tr>
</tbody>
</table>

System.out.println( fact(4) );
Returning 1 from fact(1)

<table>
<thead>
<tr>
<th>n</th>
<th>1</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td>partial result = n * 1, return 1 to whatever method called me</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>2</td>
<td>in method fact</td>
</tr>
<tr>
<td></td>
<td>partial result = n * fact(n-1)</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>3</td>
<td>in method fact</td>
</tr>
<tr>
<td></td>
<td>partial result = n * fact(n-1)</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>4</td>
<td>in method fact</td>
</tr>
<tr>
<td></td>
<td>partial result = n * fact(n-1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>System.out.println( fact(4) );</td>
<td></td>
</tr>
</tbody>
</table>
Returning 2 from fact(2)

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>partial result = 2 * 1, return 2 to whatever method called me</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>3</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>partial result = n * fact(n-1)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>4</th>
<th>in method fact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>partial result = n * fact(n-1)</td>
<td></td>
</tr>
</tbody>
</table>

System.out.println( fact(4) );
Returning 6 from fact(3)

n \[ \boxed{3} \] in method fact

partial result = 3 * 2,
return 6 to whatever method called me

n \[ \boxed{4} \] in method fact

partial result = n * fact(n-1)

System.out.println( fact(4) );
Returning 24 from fact(4)

n \[ \boxed{4} \] in method fact

partial result = 4 \times 6,
return 24 to whatever method called me

System.out.println( fact(4) );
Calling System.out.println

System.out.println( 24 );

??
Evaluating Recursive Methods
Evaluating Recursive Methods

- you must be able to evaluate recursive methods

```java
public static int mystery (int n)
{
    if( n == 0 )
        return 1;
    else
        return 3 * mystery(n-1);
}

// what is returned by mystery(5)
```
Evaluating Recursive Methods

- Draw the program stack!

  \[
  m(5) = 3 \times m(4) \\
  m(4) = 3 \times m(3) \\
  m(3) = 3 \times m(2) \\
  m(2) = 3 \times m(1) \\
  m(1) = 3 \times m(0) \\
  m(0) = 1
  \]

  \[-> 3^5 = 243\]

- with practice you can see the result
Attendance Question 4

What is returned by fact(-3)?

A. 0
B. 1
C. Infinite loop
D. Syntax error
E. Runtime error

```java
public static int fact(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * fact(n - 1);
    }
}
```
Evaluating Recursive Methods

What about multiple recursive calls?

```java
public static int bar(int n){
    if( n <= 0 )
        return 2;
    else
        return 3 + bar(n-1) + bar(n-2);
}
```

What does bar(5) return?

A. 2  
B. 5  
C. 13 
D. 62 
E. 127
Evaluating Recursive Methods

What is returned by $\text{bar}(5)$?

\begin{align*}
\text{b}(5) &= 3 + \text{b}(4) + \text{b}(3) \\
\text{b}(4) &= 3 + \text{b}(3) + \text{b}(2) \\
\text{b}(3) &= 3 + \text{b}(2) + \text{b}(1) \\
\text{b}(2) &= 3 + \text{b}(1) + \text{b}(0) \\
\text{b}(1) &= 3 + \text{b}(0) + \text{b}(-1) \\
\text{b}(0) &= 2 \\
\text{b}(-1) &= 2
\end{align*}
Evaluating Recursive Methods

What is returned by \( \text{bar}(5) \)?

\[
\begin{align*}
\text{b}(5) &= 3 + \text{b}(4) + \text{b}(3) \\
\text{b}(4) &= 3 + \text{b}(3) + \text{b}(2) \\
\text{b}(3) &= 3 + \text{b}(2) + \text{b}(1) \\
\text{b}(2) &= 3 + \text{b}(1) + \text{b}(0) \quad //\text{substitute in results} \\
\text{b}(1) &= 3 + 2 + 2 = 7 \\
\text{b}(0) &= 2 \\
\text{b}(-1) &= 2
\end{align*}
\]
Evaluating Recursive Methods

- What is returned by \( \text{bar}(5) \)?

\[
\begin{align*}
\text{b}(5) &= 3 + \text{b}(4) + \text{b}(3) \\
\text{b}(4) &= 3 + \text{b}(3) + \text{b}(2) \\
\text{b}(3) &= 3 + \text{b}(2) + \text{b}(1) \\
\text{b}(2) &= 3 + 7 + 2 = 12 \\
\text{b}(1) &= 7 \\
\text{b}(0) &= 2 \\
\text{b}(-1) &= 2
\end{align*}
\]
Evaluating Recursive Methods

- What is returned by \( \text{bar}(5) \)?

\[
\begin{align*}
b(5) &= 3 + b(4) + b(3) \\
b(4) &= 3 + b(3) + b(2) \\
b(3) &= 3 + 12 + 7 = 22 \\
b(2) &= 12 \\
b(1) &= 7 \\
b(0) &= 2 \\
b(-1) &= 2
\end{align*}
\]
Evaluating Recursive Methods

- What is returned by \( \text{bar}(5) \)?

\[
\begin{align*}
b(5) &= 3 + b(4) + b(3) \\
b(4) &= 3 + 22 + 12 = 37 \\
b(3) &= 22 \\
b(2) &= 12 \\
b(1) &= 7 \\
b(0) &= 2 \\
b(-1) &= 2
\end{align*}
\]
Evaluating Recursive Methods

- What is returned by $\text{bar}(5)$?

\[
\begin{align*}
b(5) &= 3 + 37 + 22 = 62 \\
b(4) &= 37 \\
b(3) &= 22 \\
b(2) &= 12 \\
b(1) &= 7 \\
b(0) &= 2 \\
b(-1) &= 2
\end{align*}
\]
Recursion Practice

- Write a method `raiseToPower(int base, int power)`
  - //pre: power >= 0

- Simple recursion (also called tail recursion)
Finding the Maximum in an Array

- public int max(int[] values) {
- Helper method or create smaller arrays each time
Attendance Question 5

- When writing recursive methods what should be done first?

A. Determine recursive case
B. Determine recursive step
C. Make recursive call
D. Determine base case(s)
E. Determine Big O
Your Meta Cognitive State

- Remember we are learning to use a tool.
- It is not a good tool for *all* problems.
  - In fact we will implement several algorithms and methods where an iterative (looping without recursion) solution would work just fine
- After learning the mechanics and basics of recursion the real skill is knowing what problems or class of problems to apply it to
Big O and Recursion

- Determining the Big O of recursive methods can be tricky.
- A *recurrence relation* exits if the function is defined recursively.
- The $T(N)$, actual running time, for $N!$ is recursive
  
  - $T(N)_{\text{fact}} = T(N-1)_{\text{fact}} + O(1)$
  - This turns out to be $O(N)$
    - There are $N$ steps involved
Common Recurrence Relations

- \( T(N) = T(N/2) + O(1) \rightarrow O(\log N) \)
  - binary search
- \( T(N) = T(N-1) + O(1) \rightarrow O(N) \)
  - sequential search, factorial
- \( T(N) = T(N/2) + T(N/2) + O(1) \rightarrow O(N), \)
  - tree traversal
- \( T(N) = T(N-1) + O(N) \rightarrow O(N^2) \)
  - selection sort
- \( T(N) = T(N/2) + T(N/2) + O(N) \rightarrow O(N \log N) \)
  - merge sort
- \( T(N) = T(N-1) + T(N-1) + O(1) \rightarrow O(2^N) \)
  - Fibonacci
"In ancient times, before computers were invented, alchemists studied the mystical properties of numbers. Lacking computers, they had to rely on dragons to do their work for them. The dragons were clever beasts, but also lazy and bad-tempered. The worst ones would sometimes burn their keeper to a crisp with a single fiery belch. But most dragons were merely uncooperative, as violence required too much energy. This is the story of how Martin, an alchemist’s apprentice, discovered recursion by outsmarting a lazy dragon."

- David S. Touretzky, Common Lisp: A Gentle Introduction to Symbolic Computation
Problem space consists of states (nodes) and actions (paths that lead to new states). When in a node can only see paths to connected nodes.

If a node only leads to failure go back to its "parent" node. Try other alternatives. If these all lead to failure then more backtracking may be necessary.
A More Concrete Example

- Sudoku

- 9 by 9 matrix with some numbers filled in

- all numbers must be between 1 and 9

- Goal: Each row, each column, and each mini matrix must contain the numbers between 1 and 9 once each
  - no duplicates in rows, columns, or mini matrices
Solving Sudoku – Brute Force

- A **brute force** algorithm is a simple but general approach
- Try all combinations until you find one that works
- This approach isn’t clever, but computers are fast
- Then try and improve on the brute force results
Solving Sudoku

- Brute force Sudoku Solution
  - if not open cells, solved
  - scan cells from left to right, top to bottom for first open cell
  - When an open cell is found start cycling through digits 1 to 9.
  - When a digit is placed check that the setup is legal
  - now solve the board
Attendance Question 1

- After placing a number in a cell is the remaining problem very similar to the original problem?
  
  A. Yes
  
  B. No
Solving Sudoku – Later Steps

uh oh!
Sudoku – A Dead End

- We have reached a dead end in our search.

```
5 3 1 2 7 4 8 9
6 9 8 1 9 5 6
9 8 6 3
8 6 3 1
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9
```

- With the current set up none of the nine digits work in the top right corner.
Backing Up

- When the search reaches a dead end in **backs up** to the previous cell it was trying to fill and goes onto to the next digit.
- We would back up to the cell with a 9 and that turns out to be a dead end as well so we back up again—so the algorithm needs to remember what digit to try next.
- Now in the cell with the 8. We try and 9 and move forward again.
Characteristics of Brute Force and Backtracking

- Brute force algorithms are slow
- The don't employ a lot of logic
  – For example we know a 6 can't go in the last 3 columns of the first row, but the brute force algorithm will plow ahead any way
- But, brute force algorithms are fairly easy to implement as a first pass solution
  – many backtracking algorithms are brute force algorithms
Key Insights

- After trying placing a digit in a cell we want to solve the new sudoku board
  - Isn't that a smaller (or simpler version) of the same problem we started with?!?!?!?

- After placing a number in a cell the we need to remember the next number to try in case things don't work out.

- We need to know if things worked out (found a solution) or they didn't, and if they didn't try the next number

- If we try all numbers and none of them work in our cell we need to report back that things didn't work
Recursive Backtracking

- Problems such as Suduko can be solved using recursive backtracking
- recursive because later versions of the problem are just slightly simpler versions of the original
- backtracking because we may have to try different alternatives
Recursive Backtracking

Pseudo code for recursive backtracking algorithms

If at a solution, report success
for( every possible choice from current state / node)
  Make that choice and take one step along path
  Use recursion to solve the problem for the new node / state
  If the recursive call succeeds, report the success to the next lower level
  Back out of the current choice to restore the state at the beginning of the loop.
Report failure
Goals of Backtracking

- Possible goals
  - Find a path to success
  - Find all paths to success
  - Find the best path to success

- Not all problems are exactly alike, and finding one success node may not be the end of the search
The 8 Queens Problem
The 8 Queens Problem

A classic chess puzzle

– Place 8 queen pieces on a chess board so that none of them can attack one another
The N Queens Problem

- Place N Queens on an N by N chessboard so that none of them can attack each other
- Number of possible placements?
- In 8 x 8
  
  \[ 64 \times 63 \times 62 \times 61 \times 60 \times 59 \times 58 \times 57 \]
  
  \[ = 178,462,987,637,760 / 8! \]
  
  \[ = 4,426,165,368 \]

\[ \binom{n}{k} = \frac{n \cdot (n - 1) \cdots (n - k + 1)}{k \cdot (k - 1) \cdots 1} = \frac{n!}{k!(n - k)!} \quad \text{if } 0 \leq k \leq n \] (1)

n choose k
- How many ways can you choose k things from a set of n items?
- In this case there are 64 squares and we want to choose 8 of them to put queens on
For valid solutions how many queens can be placed in a given column?

A. 0
B. 1
C. 2
D. 3
E. 4
F. Any number
Reducing the Search Space

- The previous calculation includes set ups like this one

- Includes lots of set ups with multiple queens in the same column

- How many queens can there be in one column?

- Number of set ups
  \[8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 = 16,777,216\]

- We have reduced search space by two orders of magnitude by applying some logic
A Solution to 8 Queens

- If number of queens is fixed and I realize there can't be more than one queen per column I can iterate through the rows for each column

```java
for(int r0 = 0; r0 < 8; r0++){
    board[r0][0] = 'q';
    for(int r1 = 0; r1 < 8; r1++){
        board[r1][1] = 'q';
        for(int r2 = 0; r2 < 8; r2++){
            board[r2][2] = 'q';
            // a little later
            for(int r7 = 0; r7 < 8; r7++){
                board[r7][7] = 'q';
                if( queensAreSafe(board) )
                    printSolution(board);
                board[r7][7] = ' '; //pick up queen
            }
            board[r6][6] = ' '; // pick up queen
        }
    }
}
```
N Queens

- The *problem* with N queens is you don't know how many for loops to write.
- Do the problem recursively
- Write recursive code with class and demo
  - show backtracking with breakpoint and debugging option
Recursive Backtracking

- You must practice!!!
- Learn to recognize problems that fit the pattern
- Is a *kickoff* method needed?
- All solutions or a solution?
- Reporting results and acting on results
Minesweeper
Minesweeper Reveal Algorithm

- Minesweeper
- click a cell
  - if bomb game over
  - if cell that has 1 or more bombs on border then reveal the number of bombs that border cell
  - if a cell that has 0 bombs on border then reveal that cell as a blank and click on the 8 surrounding cells
Another Backtracking Problem
A Simple Maze

Search maze until way out is found. If no way out possible report that.
The Local View

Which way do I go to get out?

Behind me, to the South is a door leading South.
Modified Backtracking Algorithm for Maze

- If the current square is outside, return TRUE to indicate that a solution has been found.
- If the current square is marked, return FALSE to indicate that this path has been tried.
- Mark the current square.
- for (each of the four compass directions)
  {  
  if ( this direction is not blocked by a wall )
  { 
    Move one step in the indicated direction from the current square.
    Try to solve the maze from there by making a recursive call.
    If this call shows the maze to be solvable, return TRUE to indicate that fact.
  }
  }
- Unmark the current square.
- Return FALSE to indicate that none of the four directions led to a solution.
Backtracking in Action

The crucial part of the algorithm is the for loop that takes us through the alternatives from the current square. Here we have moved to the North.

```c
for (dir = North; dir <= West; dir++)
{
    if (!WallExists(pt, dir))
    {
        if (SolveMaze(AdjacentPoint(pt, dir)))
            return(TRUE);
    }
}
```
Backtracking in Action

Here we have moved North again, but there is a wall to the North. East is also blocked, so we try South. That call discovers that the square is marked, so it just returns.
So the next move we can make is West.

Where is this leading?
This path reaches a dead end.

Time to backtrack!

Remember the program stack!
The recursive calls end and return until we find ourselves back here.
And now we try South
Path Eventually Found
More Backtracking Problems
Other Backtracking Problems

- Knight's Tour
- Regular Expressions
- Knapsack problem / Exhaustive Search
  - Filling a knapsack. Given a choice of items with various weights and a limited carrying capacity find the optimal load out. 50 lb. knapsack. items are 1 40 lb, 1 32 lb. 2 22 lbs, 1 15 lb, 1 5 lb. A greedy algorithm would choose the 40 lb item first. Then the 5 lb. Load out = 45lb. Exhaustive search 22 + 22 + 5 = 49.
The CD problem

- We want to put songs on a Compact Disc. 650MB CD and a bunch of songs of various sizes.

If there are no more songs to consider return result
else{
    Consider the next song in the list.
    Try not adding it to the CD so far and use recursion to evaluate best without it.
    Try adding it to the CD, and use recursion to evaluate best with it
    Whichever is better is returned as absolute best from here
}
Another Backtracking Problem

- Airlines give out frequent flier miles as a way to get people to always fly on their airline.
- Airlines also have partner airlines. Assume if you have miles on one airline you can redeem those miles on any of its partners.
- Further assume if you can redeem miles on a partner airline you can redeem miles on any of its partners and so forth...
  - Airlines don't usually allow this sort of thing.
- Given a list of airlines and each airlines partners determine if it is possible to redeem miles on a given airline A on another airline B.
Airline List – Part 1

- Delta
  - partners: Air Canada, Aero Mexico, OceanAir
- United
  - partners: Aria, Lufthansa, OceanAir, Quantas, British Airways
- Northwest
  - partners: Air Alaska, BMI, Avolar, EVA Air
- Canjet
  - partners: Girjet
- Air Canda
  - partners: Areo Mexico, Delta, Air Alaska
- Aero Mexico
  - partners: Delta, Air Canda, British Airways
Airline List - Part 2

- Ocean Air
  - partners: Delta, United, Quantas, Avolar
- AlohaAir
  - partners: Quantas
- Aria
  - partners: United, Lufthansa
- Lufthansa
  - partners: United, Aria, EVA Air
- Quantas
  - partners: United, OceanAir, AlohaAir
- BMI
  - partners: Northwest, Avolar
- Maxair
  - partners: Southwest, Girjet
Airline List - Part 3

- Girjet
  - partners: Southwest, Canjet, Maxair

- British Airways
  - partners: United, Aero Mexico

- Air Alaska
  - partners: Northwest, Air Canada

- Avolar
  - partners: Northwest, Ocean Air, BMI

- EVA Air
  - partners: Northwest, Luftansa

- Southwest
  - partners: Girjet, Maxair
Problem Example

- If I have miles on Northwest can I redeem them on Aria?
- Partial graph:

```
BMI                  Avolar
|                   /                 |
|                  /                  |
|                 /                   |
|                /                    |
|               /                     |
|              /                      |
|             /                       |
|           /                         |
|          /                          |
|         /                            |
|        /                             |
|       /                              |
|      /                               |
|     /                                |
|    /                                 |
|   /                                  |
|  /                                   |
| /                                    |
|Ocean Air                             |
```

- Northwest
- Air Alaska
- EVA Air
"There's nothing in your head the sorting hat can't see. So try me on and I will tell you where you ought to be."

- The Sorting Hat, *Harry Potter and the Sorcerer's Stone*
Sorting and Searching

- Fundamental problems in computer science and programming
- Sorting done to make searching easier
- Multiple different algorithms to solve the same problem
  - How do we know which algorithm is "better"?
- Look at searching first
- Examples use arrays of ints to illustrate algorithms
Searching

Google

recursive backtracking

Advanced Search
Preferences
Language Tools

Grep in Project

Searching for "automatic"

11 matching lines:

automatically terminates your rights under this License.
automatically receives a license from the
automatically applies SmartyPants to the bodies of your entries;
automatically creates a 'index.html' file for each directory; automatically converts all entries into
automatically converts all entries into
Searching

- Given an array or list of data find the location of a particular value or report that value is not present
- linear search
  - intuitive approach?
  - start at first item
  - is it the one I am looking for?
  - if not go to next item
  - repeat until found or all items checked
- If items not sorted or unsortable this approach is necessary
Linear Search

/*
 * pre: data != null
 * post: return the index of the first occurrence of target in data or -1 if target not present in data
 */

public int linearSearch(int[] data, int target) {
    for(int i = 0; i < data.length; i++)
        if(data[i] == target)
            return i;
    return -1;
}


Linear Search, Generic

/*
pre: data != null
post: return the index of the first occurrence
of target in data or -1 if target not present in
data
*/

public int linearSearch(Object[] data, Object target) {
    for (int i = 0; i < data.length; i++)
        if (target.equals(data[i]))
            return i;
    return -1;
}

T(N)? Big O? Best case, worst case, average case?
What is the average case Big O of linear search in an array with N items, if an item is present once?

A. O(N)
B. O(N^2)
C. O(1)
D. O(logN)
E. O(NlogN)
Searching in a Sorted Array or List

- If items are sorted then we can *divide and conquer*
- dividing your work in half with each step
  - generally a good thing
- The Binary Search on List in Ascending order
  - Start at middle of list
  - is that the item?
  - If not is it less than or greater than the item?
  - less than, move to second half of list
  - greater than, move to first half of list
  - repeat until found or sub list size = 0
Binary Search

Is middle item what we are looking for? If not is it more or less than the target item? (Assume lower)

and so forth…
public static int bsearch(int[] data, int target) {
    int indexOfTarget = -1;
    int low = 0;
    int high = data.length - 1;
    while(indexOfTarget == -1 && low <= high) {
        int mid = low + ((high - low) / 2);
        if(data[mid] == target)
            indexOfTarget = mid;
        else if(data[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return indexOfTarget;
}

// mid = (low + high) / 2; // may overflow!!!
// or mid = (low + high) >>> 1; using bitwise op
Trace When Key == 3
Trace When Key == 30

Variables of Interest?
What is the worst case Big O of binary search in an array with N items, if an item is present?

A. $O(N)$
B. $O(N^2)$
C. $O(1)$
D. $O(\log N)$
E. $O(N\log N)$
public static <T extends Comparable<? super T>> int bsearch(T[] data, T target) {

    int result = -1;
    int low = 0;
    int high = data.length - 1;
    while( result == -1 && low <= high ) {
        int mid = low + ((high - low) / 2);
        int compareResult = target.compareTo(data[mid]);
        if(compareResult == 0)
            result = mid;
        else if(compareResult > 0)
            low = mid + 1;
        else
            high = mid - 1; // compareResult < 0
    }
    return result;
}
Recursive Binary Search

public static int bsearch(int[] data, int target) {
    return bsearch(data, target, 0, data.length - 1);
}

public static int bsearch(int[] data, int target, int low, int high) {
    if (low <= high) {
        int mid = low + ((high - low) / 2);
        if (data[mid] == target)
            return mid;
        else if (data[mid] > target)
            return bsearch(data, target, low, mid - 1);
        else
            return bsearch(data, target, mid + 1, high);
    }
    return -1;
}

// is this a recursive backtracking algorithm?
Other Searching Algorithms

- Interpolation Search
  - more like what people really do
- Indexed Searching
- Binary Search Trees
- Hash Table Searching
- best-first
- A*
Sorting

U.S. All-time List - Marathon

As of 4/24/08

Women

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2:19:36</td>
<td>Deena Kastor nee Drossin</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2:21:16</td>
<td>Drossin (2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2:21:21</td>
<td>Joan Benoit Samuelson</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2:21:25</td>
<td>Kastor (3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2:22:43</td>
<td>Benoit (2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2:24:52</td>
<td>Benoit (3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2:26:11</td>
<td>Benoit (4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2:26:26</td>
<td>Julie Brown</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2:26:40</td>
<td>Kim Jones</td>
<td></td>
</tr>
</tbody>
</table>

Tasks

- Enter new task here
- New Task
- Beam Task
- Delete Task
- Status
  - Priority
  - Subject
  - Start Date
  - Due Date
- Menu

Song Name | Time | Track # | Artist | Album
---|------|---------|--------|------
Letters from the Wasteland | 4:20 | 1 of 10 | The Wallflowers | Red Letter Days
When You're On Top | 3:54 | 1 of 13 | The Wallflowers | Red Letter Days
Hand Me Down | 3:35 | 1 of 10 | The Wallflowers | Red Letter Days
My Name Is | 3:40 | 1 of 13 | The Wallflowers | Red Letter Days
Sleepwalker | 3:31 | 3 of 10 | The Wallflowers | Red Letter Days
How Good It Can Get | 4:11 | 2 of 13 | The Wallflowers | Red Letter Days
Close To You | 3:17 | 3 of 13 | The Wallflowers | Red Letter Days
I've Been Delivered | 5:01 | 4 of 10 | The Wallflowers | Red Letter Days
Witness | 3:24 | 5 of 10 | The Wallflowers | Red Letter Days
Three Ways | 4:19 | 5 of 13 | The Wallflowers | Red Letter Days
Some Flowers Bloom Dead | 4:43 | 6 of 13 | The Wallflowers | Red Letter Days
Too Late to Quit | 3:54 | 6 of 13 | The Wallflowers | Red Letter Days
Mourning Train | 4:04 | 7 of 10 | The Wallflowers | Red Letter Days
If You Never Got Sick | 3:44 | 7 of 13 | The Wallflowers | Red Letter Days
Up from Under | 3:38 | 8 of 10 | The Wallflowers | Red Letter Days
Health and Happiness | 4:03 | 8 of 13 | The Wallflowers | Red Letter Days
Murder 101 | 2:31 | 9 of 10 | The Wallflowers | Red Letter Days
See You When I Get There | 3:09 | 9 of 13 | The Wallflowers | Red Letter Days
Birdcage | 7:42 | 10 of 10 | The Wallflowers | Red Letter Days
Feel Like Summer Again | 3:48 | 10 of 13 | The Wallflowers | Red Letter Days
Everything I Need | 3:37 | 11 of 13 | The Wallflowers | Red Letter Days
Here in Pleasantville | 3:40 | 12 of 13 | The Wallflowers | Red Letter Days
Empire in My Mind (Bonus Track) | 3:31 | 13 of 13 | The Wallflowers | Red Letter Days

Red Letter Days
Sorting

› A fundamental application for computers
› Done to make finding data (searching) faster
› Many different algorithms for sorting
› One of the difficulties with sorting is working with a fixed size storage container (array)
  – if resize, that is expensive (slow)
› The simple sorts are slow
  – bubble sort
  – selection sort
  – insertion sort
Selection sort

- Search through the data and find the smallest element
- swap the smallest element with the first element
- repeat starting at second element and find the second smallest element

```java
public static void selectionSort(int[] data) {
    for (int i = 0; i < data.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < data.length; j++)
            if (data[j] < data[min])
                min = j;
        int temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}
```
Selection Sort in Practice

What is the $T(N)$, *actual* number of statements executed, of the selection sort code, given an array of $N$ elements? What is the Big O?
Generic Selection Sort

```java
public static <T extends Comparable<? super T>>
void selectionSort(T[] data) {

    for(int i = 0; i < data.length - 1; i++) {
        int min = i;
        for(int j = i + 1; j < data.length; j++)
            if( data[min].compareTo(data[j]) > 0 )
                min = j;
        T temp = data[i];
        data[i] = data[min];
        data[min] = temp;
    }
}
```
Insertion Sort

- Another of the $O(N^2)$ sorts
- The first item is sorted
- Compare the second item to the first
  - if smaller swap
- Third item, compare to item next to it
  - need to swap
  - after swap compare again
- And so forth…
Insertion Sort Code

```java
public void insertionSort(int[] data) {
    for(int i = 1; i < data.length; i++) {
        int temp = data[i];
        int j = i;
        while( j > 0 && temp < data[j - 1]){
            // swap elements
            data[j] = data[j - 1];
            data[j - 1] = temp;
            j--;
        }
    }
}
```

- Best case, worst case, average case Big O?
Comparing Algorithms

- Which algorithm do you think will be faster given random data, selection sort or insertion sort?

A. Insertion Sort
B. Selection Sort
C. About the same
"stack n.
The set of things a person has to do in the future. "I haven't done it yet because every time I pop my stack something new gets pushed." If you are interrupted several times in the middle of a conversation, "My stack overflowed" means "I forget what we were talking about."

-The Hacker's Dictionary

Friedrich L. Bauer
German computer scientist who proposed "stack method of expression evaluation" in 1955.
Sharper Tools

Lists

Stacks
Stacks

- Access is allowed only at one point of the structure, normally termed the top of the stack
  - access to the most recently added item only

- Operations are limited:
  - push (add item to stack)
  - pop (remove top item from stack)
  - top (get top item without removing it)
  - isEmpty

- Described as a "Last In First Out" (LIFO) data structure
Assume a simple stack for integers.

```java
Stack<Integer> s = new Stack<Integer>();
s.push(12);
s.push(4);
s.push( s.top() + 2 );
s.pop();
s.pop();
s.push( s.top() );
//what are contents of stack?
```
Stack Operations

Write a method to print out contents of stack in reverse order.
Uses of Stacks

- The runtime stack used by a process (running program) to keep track of methods in progress
- Search problems
- Undo, redo, back, forward
What is Output?

Stack<Integer> s = new Stack<Integer>();
// put stuff in stack
for(int i = 0; i < 5; i++)
    s.push( i );
// Print out contents of stack
// while emptying it.
// Assume there is a size method.
for(int i = 0; i < s.size(); i++)
    System.out.print( s.pop() + " ");

A 0 1 2 3 4  
B 4 3 2 1 0  
C 4 3 2  
D 2 3 4  
E No output due to runtime error
Corrected Version

Stack<Integer> s = new Stack<Integer>();
// put stuff in stack
for(int i = 0; i < 5; i++)
    s.push( i );
// print out contents of stack
// while emptying it
int limit = s.size();
for(int i = 0; i < limit; i++)
    System.out.print( s.pop() + " ");

//or

// while( !s.isEmpty() )
//    System.out.println( s.pop() );
Implementing a stack

- need an underlying collection to hold the elements of the stack
- 2 obvious choices
  - array (native or ArrayList)
  - linked list

Adding a *layer of abstraction*. A big idea.

- array implementation
- linked list implementation
Applications of Stacks
Mathematical Calculations

- What does $3 + 2 \times 4$ equal?
  $2 \times 4 + 3$?  $3 \times 2 + 4$?

- The precedence of operators affects the order of operations.

- A mathematical expression cannot simply be evaluated left to right.

- A challenge when evaluating a program.

- *Lexical analysis* is the process of interpreting a program.

What about $1 - 2 - 4^5 \times 3 \times 6 / 7^2^3$
**Infix and Postfix Expressions**

- The way we are used to writing expressions is known as infix notation.
- Postfix expression does not require any precedence rules.
- \(32 * 1 +\) is postfix of \(3 * 2 + 1\).
- Evaluate the following postfix expressions and write out a corresponding infix expression:
  - \(2 3 2 4 * + *
  - 1 2 - 3 2 ^ 3 * 6 / +
  - 2 5 ^ 1 - 1 2 3 4 ^ * +
  - 2 5 ^ 1 -\)
Clicker Question 2

What does the following postfix expression evaluate to?

6 3 2 + *

A. 18  
B. 36  
C. 24  
D. 11  
E. 30
Evaluation of Postfix Expressions

- Easy to do with a stack
- given a proper postfix expression:
  - get the next token
  - if it is an operand push it onto the stack
  - else if it is an operator
    • pop the stack for the right hand operand
    • pop the stack for the left hand operand
    • apply the operator to the two operands
    • push the result onto the stack
  - when the expression has been exhausted the result is the top (and only element) of the stack
Infix to Postfix

Convert the following equations from infix to postfix:

- \( 2 \wedge 3 \wedge 3 + 5 \times 1 \)
- \( 11 + 2 - 1 \times 3 / 3 + 2 \wedge 2 / 3 \)

Problems:
- Negative numbers?
- Parentheses in expression
Infix to Postfix Conversion

- Requires operator precedence parsing algorithm
  - parse v. To determine the syntactic structure of a sentence or other utterance

Operands: add to expression
Close parenthesis: pop stack symbols until an open parenthesis appears

Operators:
  - Have an on stack and off stack precedence
  - Pop all stack symbols until a symbol of lower precedence appears. Then push the operator

End of input: Pop all remaining stack symbols and add to the expression
Simple Example

Infix Expression: 3 + 2 * 4

PostFix Expression:

Operator Stack:

Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression: + 2 * 4

PostFix Expression: 3

Operator Stack:

Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression: 2 * 4
PostFix Expression: 3
Operator Stack: +

Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression: * 4
PostFix Expression: 3 2
Operator Stack: +

Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression: 4
PostFix Expression: 3 2
Operator Stack: + *

Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression:
PostFix Expression: 3 2 4
Operator Stack: + *

Precedence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression: 
PostFix Expression: 3 2 4 *
Operator Stack: +

Precendence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Simple Example

Infix Expression:
PostFix Expression:  3 2 4 * +

Operator Stack:

Precendence Table

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Off Stack Precedence</th>
<th>On Stack Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>*</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>^</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>(</td>
<td>20</td>
<td>0</td>
</tr>
</tbody>
</table>
Example

\[ 11 + 2^4^3 - ((4 + 5) \times 6)^2 \]

Show algorithm in action on above equation
Balanced Symbol Checking

- In processing programs and working with computer languages there are many instances when symbols must be balanced: 

{ }, [ ], ( )

A stack is useful for checking symbol balance. When a closing symbol is found it must match the most recent opening symbol of the same type.

- Applicable to checking html and xml tags!
Algorithm for Balanced Symbol Checking

- Make an empty stack
- read symbols until end of file
  - if the symbol is an opening symbol push it onto the stack
  - if it is a closing symbol do the following
    - if the stack is empty report an error
    - otherwise pop the stack. If the symbol popped does not match the closing symbol report an error
- At the end of the file if the stack is not empty report an error
Algorithm in practice

- list[i] = 3 * ( 44 - method( foo( list[ 2 * (i + 1) + foo( list[i - 1] ) ] ) / 2 * ) - list[ method(list[0])];

- Complications
  - when is it not an error to have non matching symbols?

- Processing a file
  - *Tokenization*: the process of scanning an input stream. Each independent chunk is a token.

- Tokens may be made up of 1 or more characters
"FISH queue: n. [acronym, by analogy with FIFO (First In, First Out)] ‘First In, Still Here’. A joking way of pointing out that processing of a particular sequence of events or requests has stopped dead. Also FISH mode and FISHnet; the latter may be applied to any network that is running really slowly or exhibiting extreme flakiness."

-The Jargon File 4.4.7
Queues

- A sharp tool like stacks
- Like a line
  - In England people don’t “get in line” they “queue up”.
Queue Properties

- Queues are a first in first out data structure
  - FIFO (or LILO, but that sounds a bit silly)
- Add items to the end of the queue
- Access and remove from the front
  - Access to the element that has been in the structure the \textit{longest} amount of time
- Used extensively in operating systems
  - Queues of processes, I/O requests, and much more
Queues in Operating Systems

- On a computer with $N$ cores on the CPU, but more than $N$ processes, how many processes can actually be executing at one time?
- One job of OS, schedule the processes for the CPU
Queue operations

- enqueue(E item)
  - a.k.a. add(E item)

- E front()
  - a.k.a. E peek()

- E dequeue()
  - a.k.a. E remove()

- boolean isEmpty()

Specify methods in an interface, allow multiple implementations.
public interface Queue<E> {
    // place item at back of this Queue
    enqueue(E item);
    
    // access item at front of this Queue
    // pre: !isEmpty()
    E front();
    
    // remove item at front of this Queue
    // pre: !isEmpty()
    E dequeue();
    
    boolean isEmpty();
}
Implementing a Queue

Given the internal storage container and choice for front and back of queue what are the Big O of the queue operations?

- ArrayList
- LinkedList (Singly Linked)
- LinkedList (Doubly Linked)

enqueue
front
decqueue
isEmpty
Clicker Question 1

If implementing a queue with a singly linked list with references to the first and last nodes (head and tail) which end of the list should be the front of the queue in order to have all queue operations O(1)?

A. The front of the list should be the front of the queue
B. The back of the list should be the front of the queue.
C. Either end will work to make all ops O(1)
D. Neither end will allow all ops to be O(1)
Alternate Implementation

- How about implementing a Queue with a native array?
  - Seems like a step backwards
Application of Queues

- Radix Sort
  - radix is a synonym for base. base 10, base 2
- Multi pass sorting algorithm that only looks at individual digits during each pass
- Use queues as buckets to store elements
- Create an array of 10 queues
- Starting with the least significant digit place value in queue that matches digit
- empty queues back into array
- repeat, moving to next least significant digit
Radix Sort in Action: 1s

- original values in array
  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Look at ones place
  9, 113, 70, 86, 12, 93, 37, 40, 252, 7, 79, 12

- Queues:
  0 70, 40 5
  1 6 86
  2 12, 252, 12 7 37, 7
  3 113, 93 8
  4 9 9, 79
Radix Sort in Action: 10s

- Empty queues in order from 0 to 9 back into array
  70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79

- Now look at 10's place
  70, 40, 12, 252, 12, 113, 93, 86, 37, 7, 9, 79

- Queues:
  0    7, 9
  1 12, 12, 113
  2
  3 37
  4 40
  5 252
  6
  7 70, 79
  8 86
  9 93
Radix Sort in Action: 100s

- Empty queues in order from 0 to 9 back into array
  7, 9, 12, 12, 113, 37, 40, 252, 70, 79, 86, 93

- Now look at 100's place
  7, 9, 12, 12, 113, 37, 40, 252, 70, 79, 86, 93

- Queues:
  0 7, 9, 12, 12, 37, 40, 70, 79, 86, 93 5
  1 113 6
  2 252 7
  3 8 8
  4 9
Radix Sort in Action: Final Step

- Empty queues in order from 0 to 9 back into array
  7, 9, 12, 12, 40, 70, 79, 86, 93, 113, 252
public static void sort(int[] list){
    ArrayList<Queue<Integer>> queues = new ArrayList<Queue<Integer>>() ;
    for(int i = 0; i < 10; i++)
        queues.add( new LinkedList<Integer>());
    int passes = numDigits(list[0]);  // helper method
    // or int passes = (int) Math.log10(list[0]);
    for(int i = 1; i < list.length; i++){
        int temp = numDigits(list[i]);
        if( temp > passes )
            passes = temp;
    }
    for(int i = 0; i < passes; i++){
        for(int j = 0; j < list.length; j++)
            queues.get(valueOfDigit(list[j], i)).add(list[j]);
        int pos = 0;
        for(Queue<Integer> q : queues){
            while(!q.isEmpty())
                list[pos++] = q.remove();
        }
    }
}
"The bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems."

- Don Knuth
Previous Sorts

- Insertion Sort and Selection Sort are both average case $O(N^2)$
- Today we will look at two faster sorting algorithms.
  - quicksort
  - mergesort
A property of sorts

If a sort guarantees the relative order of equal items stays the same then it is a stable sort

\[ [7_1, 6, 7_2, 5, 1, 2, 7_3, -5] \]

– subscripts added for clarity

\[ [-5, 1, 2, 5, 6, 7_1, 7_2, 7_3] \]

– result of stable sort

Real world example:

– sort a table in Wikipedia by one criteria, then another
– sort by country, then by major wins
Quicksort

- Invented by C.A.R. (Tony) Hoare
- A divide and conquer approach that uses recursion

1. If the list has 0 or 1 elements it is sorted
2. otherwise, pick any element p in the list. This is called the *pivot* value
3. **Partition** the list minus the pivot into two sub lists according to values less than or greater than the pivot. (equal values go to either)
4. return the quicksort of the first list followed by the quicksort of the second list
Quicksort in Action

39 23 17 90 33 72 46 79 11 52 64 5 71
Pick middle element as pivot: 46
Partition list
23 17 5 33 39 11 46 79 72 52 64 90 71
quick sort the less than list
Pick middle element as pivot: 33
23 17 5 11 33 39
quicksort the less than list, pivot now 5
{} 5 23 17 11
quicksort the less than list, base case
quicksort the greater than list
Pick middle element as pivot: 17
and so on....
Quicksort on Another Data Set

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>44</td>
<td>68</td>
<td>191</td>
<td>119</td>
<td>119</td>
<td>37</td>
<td>83</td>
<td>82</td>
<td>191</td>
<td>45</td>
<td>158</td>
<td>130</td>
<td>76</td>
<td>153</td>
<td>39</td>
<td>25</td>
</tr>
</tbody>
</table>

Big O of Quicksort?
private static void swapReferences( Object[] a, int index1, int index2 ) {
    Object tmp = a[index1];
    a[index1] = a[index2];
    a[index2] = tmp;
}

private void quicksort( Comparable[] list, int start, int stop ) {
    if(start >= stop)
        return; //base case list of 0 or 1 elements
    int pivotIndex = (start + stop) / 2;

    // Place pivot at start position
    swapReferences(list, pivotIndex, start);
    Comparable pivot = list[start];

    // Begin partitioning
    int i, j = start;

    // from first to j are elements less than or equal to pivot
    // from j to i are elements greater than pivot
    // elements beyond i have not been checked yet
    for(i = start + 1; i <= stop; i++ )
    {
        // is current element less than or equal to pivot
        if(list[i].compareTo(pivot) <= 0)
            j++;
        swapReferences(list, i, j);
    }

    //restore pivot to correct spot
    swapReferences(list, start, j);
    quicksort( list, start, j - 1 );    // Sort small elements
    quicksort( list, j + 1, stop );    // Sort large elements
Clicker Question 1

What are the best case and worst case Orders (Big O) for quicksort?

<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. O(N\log N)</td>
<td>O(N^2)</td>
</tr>
<tr>
<td>B. O(N^2)</td>
<td>O(N^2)</td>
</tr>
<tr>
<td>C. O(N^2)</td>
<td>O(N!)</td>
</tr>
<tr>
<td>D. O(N\log N)</td>
<td>O(N\log N)</td>
</tr>
<tr>
<td>E. O(N)</td>
<td>O(N\log N)</td>
</tr>
</tbody>
</table>
Clicker Question 2

Is quicksort always stable?
A. Yes
B. No
Merge Sort Algorithm

Don Knuth cites John von Neumann as the creator of this algorithm

1. If a list has 1 element or 0 elements it is sorted
2. If a list has more than 1 split into into 2 separate lists
3. Perform this algorithm on each of those smaller lists
4. Take the 2 sorted lists and merge them together
Merge Sort

When implementing one temporary array is used instead of multiple temporary arrays.

Why?
Merge Sort code

/**
 * perform a merge sort on the data in c
 * @param c c != null, all elements of c
 * are the same data type
 */
public static void mergeSort(Comparable[] c) {
    Comparable[] temp = new Comparable[ c.length ];
    sort(c, temp, 0, c.length - 1);
}

private static void sort(Comparable[] list, Comparable[] temp, int low, int high) {
    if( low < high ) {
        int center = (low + high) / 2;
        sort(list, temp, low, center);
        sort(list, temp, center + 1, high);
        merge(list, temp, low, center + 1, high);
    }
}
private static void merge( Comparable[] list, Comparable[] temp,
        int leftPos, int rightPos, int rightEnd) {
    int leftEnd = rightPos - 1;
    int tempPos = leftPos;
    int numElements = rightEnd - leftPos + 1;
    // main loop
    while( leftPos <= leftEnd && rightPos <= rightEnd){
        if( list[ leftPos ].compareTo(list[rightPos]) <= 0) {
            temp[ tempPos ] = list[ leftPos ];
            leftPos++;
        } else{
            temp[ tempPos ] = list[ rightPos ];
            rightPos++;
        }
        tempPos++;
    }
    // copy rest of left half
    while( leftPos <= leftEnd){
        temp[ tempPos ] = list[ leftPos ];
        tempPos++;
        leftPos++;
    }
    // copy rest of right half
    while( rightPos <= rightEnd){
        temp[ tempPos ] = list[ rightPos ];
        tempPos++;
        rightPos++;
    }
    // Copy temp back into list
    for(int i = 0; i < numElements; i++, rightEnd--)
        list[ rightEnd ] = temp[ rightEnd ];
}
Clicker Question 3

What are the best case and worst case Orders (Big O) for mergesort?

<table>
<thead>
<tr>
<th>Best</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. $O(N \log N)$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>B. $O(N^2)$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>C. $O(N^2)$</td>
<td>$O(N!)$</td>
</tr>
<tr>
<td>D. $O(N \log N)$</td>
<td>$O(N \log N)$</td>
</tr>
<tr>
<td>E. $O(N)$</td>
<td>$O(N \log N)$</td>
</tr>
</tbody>
</table>
Clicker Question 4

- Is mergesort always stable?
  A. Yes
  B. No
Clicker Question 5

- You have 1,000,000 items that you will be searching. How many searches need to be performed before the data is changed to make it worthwhile to sort the data before searching?

A. 5
B. 40
C. 1,000
D. 10,000
E. 500,000
## Comparison of Various Sorts (2001)

<table>
<thead>
<tr>
<th>Num Items</th>
<th>Selection</th>
<th>Insertion</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>16</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2000</td>
<td>59</td>
<td>49</td>
<td>6</td>
</tr>
<tr>
<td>4000</td>
<td>271</td>
<td>175</td>
<td>5</td>
</tr>
<tr>
<td>8000</td>
<td>1056</td>
<td>686</td>
<td>0</td>
</tr>
<tr>
<td>16000</td>
<td>4203</td>
<td>2754</td>
<td>11</td>
</tr>
<tr>
<td>32000</td>
<td>16852</td>
<td>11039</td>
<td>45</td>
</tr>
<tr>
<td>64000</td>
<td>expected?</td>
<td>expected?</td>
<td>68</td>
</tr>
<tr>
<td>128000</td>
<td>expected?</td>
<td>expected?</td>
<td>158</td>
</tr>
<tr>
<td>256000</td>
<td>expected?</td>
<td>expected?</td>
<td>335</td>
</tr>
<tr>
<td>512000</td>
<td>expected?</td>
<td>expected?</td>
<td>722</td>
</tr>
<tr>
<td>1024000</td>
<td>expected?</td>
<td>expected?</td>
<td>1550</td>
</tr>
</tbody>
</table>

Times in milliseconds, 1000 milliseconds = 1 second
## Comparison of Various Sorts (2001)

<table>
<thead>
<tr>
<th>Num Items</th>
<th>Selection</th>
<th>Insertion</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.016</td>
<td>0.005</td>
<td>0 ??</td>
</tr>
<tr>
<td>2000</td>
<td>0.059</td>
<td>0.049</td>
<td>0.006</td>
</tr>
<tr>
<td>4000</td>
<td>0.271</td>
<td>0.175</td>
<td>0.005</td>
</tr>
<tr>
<td>8000</td>
<td>1.056</td>
<td>0.686</td>
<td>0 ??</td>
</tr>
<tr>
<td>16000</td>
<td>4.203</td>
<td>2.754</td>
<td>0.011</td>
</tr>
<tr>
<td>32000</td>
<td>16.852</td>
<td>11.039</td>
<td>0.045</td>
</tr>
<tr>
<td>64000</td>
<td>expected?</td>
<td>expected?</td>
<td>0.068</td>
</tr>
<tr>
<td>128000</td>
<td>expected?</td>
<td>expected?</td>
<td>0.158</td>
</tr>
<tr>
<td>256000</td>
<td>expected?</td>
<td>expected?</td>
<td>0.335</td>
</tr>
<tr>
<td>512000</td>
<td>expected?</td>
<td>expected?</td>
<td>0.722</td>
</tr>
<tr>
<td>1024000</td>
<td>expected?</td>
<td>expected?</td>
<td>1.550</td>
</tr>
</tbody>
</table>

*times in seconds*
## Comparison of Various Sorts (2011)

<table>
<thead>
<tr>
<th>Num Items</th>
<th>Selection</th>
<th>Insertion</th>
<th>Quicksort</th>
<th>Merge</th>
<th>Arrays.sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.002</td>
<td>0.001</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2000</td>
<td>0.002</td>
<td>0.001</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4000</td>
<td>0.006</td>
<td>0.004</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8000</td>
<td>0.022</td>
<td>0.018</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16000</td>
<td>0.086</td>
<td>0.070</td>
<td>0.002</td>
<td>0.002</td>
<td>0.002</td>
</tr>
<tr>
<td>32000</td>
<td>0.341</td>
<td>0.280</td>
<td>0.004</td>
<td>0.005</td>
<td>0.003</td>
</tr>
<tr>
<td>64000</td>
<td>1.352</td>
<td>1.123</td>
<td>0.008</td>
<td>0.010</td>
<td>0.007</td>
</tr>
<tr>
<td>128000</td>
<td>5.394</td>
<td>4.499</td>
<td>0.017</td>
<td>0.022</td>
<td>0.015</td>
</tr>
<tr>
<td>256000</td>
<td>21.560</td>
<td>18.060</td>
<td>0.035</td>
<td>0.047</td>
<td>0.031</td>
</tr>
<tr>
<td>512000</td>
<td>86.083</td>
<td>72.303</td>
<td>0.072</td>
<td>0.099</td>
<td>0.066</td>
</tr>
<tr>
<td>1024000</td>
<td>???</td>
<td>???</td>
<td>0.152</td>
<td>0.206</td>
<td>0.138</td>
</tr>
<tr>
<td>2048000</td>
<td></td>
<td></td>
<td>0.317</td>
<td>0.434</td>
<td>0.287</td>
</tr>
<tr>
<td>4096000</td>
<td></td>
<td></td>
<td>0.663</td>
<td>0.911</td>
<td>0.601</td>
</tr>
<tr>
<td>8192000</td>
<td>1.375</td>
<td>1.885</td>
<td>1.246</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concluding Thoughts

- Language libraries often have sorting algorithms in them
  - Java Arrays and Collections classes
  - C++ Standard Template Library
  - Python sort and sorted functions

- Hybrid sorts
  - when size of unsorted list or portion of array is small use insertion sort, otherwise use $O(N \log N)$ sort like Quicksort or Mergesort
Concluding Thoughts

- Sorts still being created!
- Timsort (2002)
  - created for python version 2.3
  - now used in Java version 7.0
  - takes advantage of real world data
  - real world data is usually partially sorted, not totally random
- Library Sort (2006)
  - Like insertion sort, but leaves gaps for later elements
"A tree may grow a thousand feet tall, but its leaves will return to its roots."

-Chinese Proverb
A *tree* is an abstract data type

- one entry point, the **root**
- Each node is either a **leaf** or an **internal node**
- An internal node has 1 or more **children**, nodes that can be reached directly from that internal node.
- The internal node is said to be the **parent** of its child nodes
Properties of Trees

- Only access point is the root
- All nodes, except the root, have one parent
  - like the inheritance hierarchy in Java
- Traditionally trees drawn upside down
Properties of Trees and Nodes

- **siblings**: two nodes that have the same parent
- **edge**: the link from one node to another
- **path length**: the number of edges that must be traversed to get from one node to another

Path length from root to this node is 3
More Properties of Trees

- **depth**: the path length from the root of the tree to this node

- **height of a node**: The maximum distance (path length) of any leaf from this node
  - a leaf has a height of 0
  - the height of a tree is the height of the root of that tree

- **descendants**: any nodes that can be reached via 1 or more edges from this node

- **ancestors**: any nodes for which this node is a descendant
Tree Visualization

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O
Clicker Question 1

What is the depth of the node that contains M on the previous slide?

A. 0
B. 1
C. 2
D. 3
E. 4
There are many variations on trees but we will start with \textit{binary trees}.

\textit{Binary tree}: each node has at most two children

- the possible children are usually referred to as the left child and the right child.
Full Binary Tree

- full binary tree: a binary tree is which each node was exactly 2 or 0 children
Clicker Question 2

What is the maximum height of a full binary tree with 11 nodes?

A. 1
B. 3
C. 5
D. 7
E. Not possible to construct a full binary tree with 11 nodes.
Complete Binary Tree

- *complete binary tree:* a binary tree in which every level, except possibly the deepest is completely filled. At depth n, the height of the tree, all nodes are as far left as possible.

Where would the next node go to maintain a complete tree?
Clicker Question 3

- What is the height of a complete binary tree that contains N nodes?

A. $O(1)$
B. $O(\log N)$
C. $O(N^{1/2})$
D. $O(N)$
E. $O(N \log N)$
Perfect Binary Tree

- *perfect binary tree:* a binary tree with all leaf nodes at the same depth. All internal nodes have exactly two children.

- a perfect binary tree has the maximum number of nodes for a given height

- a perfect binary tree has \(2^{(n+1)} - 1\) nodes where \(n\) is the height of the tree
  - height = 0 -> 1 node
  - height = 1 -> 3 nodes
  - height = 2 -> 7 nodes
  - height = 3 -> 15 nodes
A Binary Node class

public class Bnode <E> {
    private E myData;
    private Bnode<E> myLeft;
    private Bnode<E> myRight;

    public BNode();
    public BNode(E data, Bnode<E> left, Bnode<E> right)
    public E getData()
    public Bnode<E> getLeft()
    public Bnode<E> getRight()

    public void setData(E data)
    public void setLeft(Bnode<E> left)
    public void setRight(Bnode<E> right)
}
Binary Tree Traversals

- Many algorithms require all nodes of a binary tree be visited and the contents of each node processed or examined.

- There are 4 traditional types of traversals
  - preorder traversal: process the root, then process all sub trees (left to right)
  - in order traversal: process the left sub tree, process the root, process the right sub tree
  - post order traversal: process the left sub tree, process the right sub tree, then process the root
  - level order traversal: starting from the root of a tree, process all nodes at the same depth from left to right, then proceed to the nodes at the next depth.
Results of Traversals

- To determine the results of a traversal on a given tree draw a path around the tree.
  - start on the left side of the root and trace around the tree. The path should stay close to the tree.
  
  pre order: process when pass down left side of node
  12 49 13 5 42

  in order: process when pass underneath node
  13 49 5 12 42

  post order: process when pass up right side of node
  13 5 49 42 12
Tree Traversals

A
  /   \
 C   D
 /     /
 F     G

H
 /     /
 K     L

J

Binary Trees
What is the result of a post order traversal of the tree on the previous slide?

A.  F C G A K H L D J
B.  F G C K L H J D A
C.  A C F G D H K L J
D.  A C D F G H J K L
E.  L K J H G F D C A
Implement Traversals

- Implement preorder, inorder, and post order traversal
  - Big O time and space?
- Implement a level order traversal using a queue
  - Big O time and space?
- Implement a level order traversal without a queue
  - target depth
Breadth First - Depth First

- from NIST - DADS

- **breadth first search**: Any search algorithm that considers neighbors of a *vertex* (node), that is, outgoing *edges* (links) of the vertex's predecessor in the search, before any outgoing edges of the vertex.

- **depth first search**: Any search algorithm that considers outgoing *edges* (links to *children*) of a *vertex* (node) before any of the vertex's (node) *siblings*, that is, outgoing edges of the vertex's predecessor in the search. Extremes are searched first.
Breadth First

- A level order traversal of a tree could be used as a breadth first search
- Search all nodes in a level before going down to the next level
Breadth First Search of Tree

- C
- A
- G
- X
- Z
- W
- Q
- P
- O
- U
- K
- B
- Z
- M
- R
- L
Breadth First Search

Find Node with B

search level 0 first
Breadth First Search

Find Node with B

search level 1 next
Breadth First Search

Find Node with B

search level 2 next
Breadth First Search

Find Node with B

search level 3 next
Depth First Search

Find Node with B
BFS - DFS

- Breadth first search typically implemented with a Queue
- Depth first search typically implemented with recursion
- which technique do I use?
  – depends on the problem
"Yes. Shrubberies are my trade. I am a shrubber. My name is 'Roger the Shrubber'. I arrange, design, and sell shrubberies."

-Monty Python and The Holy Grail
The Problem with Linked Lists

- Accessing a item from a linked list takes $O(N)$ time for an arbitrary element.
- Binary trees can improve upon this and reduce access to $O(\log N)$ time for the average case.
- Expands on the binary search technique and allows insertions and deletions.
- Worst case degenerates to $O(N)$ but this can be avoided by using balanced trees (AVL, Red-Black).
A binary search tree is a binary tree in which every node's left subtree holds values less than the node's value, and every right subtree holds values greater than the node's value.

A new node is added as a leaf.
BST Insertion

- Add the following values one at a time to an initially empty binary search tree using the naïve algorithm:

  90  20  9  98  10  28  -25

- What is the resulting tree?
Traversals

- What is the result of an inorder traversal of the resulting tree?
- How could a preorder traversal be useful?
After adding N distinct elements in random order to a Binary Search Tree what is the expected height of the tree?

A. \( O(N^{1/2}) \)
B. \( O(\log N) \)
C. \( O(N) \)
D. \( O(N\log N) \)
E. \( O(N^2) \)
Clicker Question 1

- After adding N distinct elements to a Binary Search Tree what is the worst case height of the tree?

A. $O(N^{1/2})$
B. $O(\log N)$
C. $O(N)$
D. $O(N \log N)$
E. $O(N^2)$
Node for Binary Search Trees

```java
class BSTNode<E extends Comparable<E> {  
    private Comparable<E> myData;  
    private BSTNode<E> myLeft;  
    private BSTNode<E> myRight;

    public BSTNode(E item)  
    {  
        myData = item;  
    }

    public E getValue()  
    {  
        return myData;  
    }

    public BSTNode<E> getLeft()  
    {  
        return myLeft;  
    }

    public BSTNode<E> getRight()  
    {  
        return myRight;  
    }

    public void setLeft(BSTNode<E> b)  
    {  
        myLeft = b;  
    }
    // setRight not shown
```
Worst Case Performance

- Insert the following values into an initially empty binary search tree using the traditional, naïve algorithm:

  2 3 5 7 11 13 17

- What is the height of the tree?
- What is the worst case height of a BST?
More on Implementation

- Many ways to implement BSTs
- Using nodes is just one and even then many options and choices

```java
public class BinarySearchTree<E extends Comparable<E>> { 
    private BSTNode<E> root;
    private int size;
}
```
Add an Element, Recursive
Add an Element, Iterative
Clicker Question 2

What are the best case and worst case order to add N distinct elements, one at a time, to an initially empty binary search tree?

Best | Worst
--- | ---
A. $O(N)$ | $O(N)$
B. $O(N \log N)$ | $O(N \log N)$
C. $O(N)$ | $O(N \log N)$
D. $O(N \log N)$ | $O(N^2)$
E. $O(N^2)$ | $O(N^2)$

// given int[] data
// no duplicates in
// data
BST<Integer> b =
    new BST<Integer>();
for(int x : data)
    b.add(x);
Performance of Binary Trees

- For the three core operations (add, access, remove) a binary search tree (BST) has an average case performance of $O(\log N)$
- Even when using the *naïve insertion / removal algorithms*
  - no checks to maintain balance
  - balance achieved based on the randomness of the data inserted
Remove an Element

- Three cases
  - node is a leaf, 0 children (easy)
  - node has 1 child (easy)
  - node has 2 children (interesting)
Properties of a BST

- The minimum value is in the left most node
- The maximum value is in the right most node
  - useful when removing an element from the BST
Alternate Implementation

- In class examples of dynamic data structures have relied on *null terminated ends*.
  - Use null to show end of list, no children
- Alternative form
  - use structural recursion and polymorphism
BST Interface

```java
public interface BST {
    public int size();
    public boolean contains(Comparable obj);
    public boolean add(Comparable obj);
}
```
public class EmptyBST implements BST {

    private static EmptyBST theOne = new EmptyBST();

    private EmptyBST(){}

    public static EmptyBST getEmptyBST(){ return theOne; }

    public NEBST add(Comparable obj) { return new NEBST(obj); }

    public boolean contains(Comparable obj) { return false; }

    public int size() { return 0; }
}

Binary Search Trees
public class NEBST implements BST {

    private Comparable data;
    private BST left;
    private BST right;

    public NEBST(Comparable d) {
        data = d;
        right = EmptyBST.getEmptyBST();
        left = EmptyBST.getEmptyBST();
    }

    public BST add(Comparable obj) {
        int val = obj.compareTo(data);
        if (val < 0)
            left = left.add(obj);
        else if (val > 0)
            right = right.add(obj);
        return this;
    }
}

public boolean contains(Comparable obj) {
    int val = obj.compareTo(data);
    if (val == 0)
        return true;
    else if (val < 0)
        return left.contains(obj);
    else
        return right.contains(obj);
}

public int size() {
    return 1 + left.size() + right.size();
}
Topic 20: Huffman Coding

The author should gaze at Noah, and ... learn, as they did in the Ark, to crowd a great deal of matter into a very small compass.

Sydney Smith, Edinburgh Review
Agenda

- Encoding
- Compression
- Huffman Coding
Encoding

- UT CS
- 85 84 32 67 83
- 01010101 01010100 00100000 01000011 01010011

- what is a file?
- open a bitmap in a text editor
<table>
<thead>
<tr>
<th>Binary</th>
<th>Oct</th>
<th>Dec</th>
<th>Hex</th>
<th>Glyph</th>
</tr>
</thead>
<tbody>
<tr>
<td>010 0000</td>
<td>040</td>
<td>32</td>
<td>20</td>
<td>(space)</td>
</tr>
<tr>
<td>010 0001</td>
<td>041</td>
<td>33</td>
<td>21</td>
<td>!</td>
</tr>
<tr>
<td>010 0010</td>
<td>042</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
</tr>
<tr>
<td>010 0011</td>
<td>043</td>
<td>35</td>
<td>23</td>
<td>#</td>
</tr>
<tr>
<td>010 0100</td>
<td>044</td>
<td>36</td>
<td>24</td>
<td>$</td>
</tr>
<tr>
<td>010 0101</td>
<td>045</td>
<td>37</td>
<td>25</td>
<td>%</td>
</tr>
<tr>
<td>010 0110</td>
<td>046</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
</tr>
<tr>
<td>010 0111</td>
<td>047</td>
<td>39</td>
<td>27</td>
<td>'</td>
</tr>
<tr>
<td>010 1000</td>
<td>050</td>
<td>40</td>
<td>28</td>
<td>(</td>
</tr>
<tr>
<td>010 1001</td>
<td>051</td>
<td>41</td>
<td>29</td>
<td>)</td>
</tr>
<tr>
<td>010 1010</td>
<td>052</td>
<td>42</td>
<td>2A</td>
<td>*</td>
</tr>
<tr>
<td>010 1011</td>
<td>053</td>
<td>43</td>
<td>2B</td>
<td>+</td>
</tr>
<tr>
<td>010 1100</td>
<td>054</td>
<td>44</td>
<td>2C</td>
<td>.</td>
</tr>
<tr>
<td>010 1101</td>
<td>055</td>
<td>45</td>
<td>2D</td>
<td>-</td>
</tr>
<tr>
<td>010 1110</td>
<td>056</td>
<td>46</td>
<td>2E</td>
<td></td>
</tr>
</tbody>
</table>
Project Gutenberg's The Adventures of Sherlock Holmes

This eBook is for the use of anyone anywhere at almost no restrictions whatsoever. You may copy and re-use it under the terms of the Project Gutenberg e-book policy or online at www.gutenberg.net

Title: The Adventures of Sherlock Holmes

Author: Arthur Conan Doyle

Posting Date: April 18, 2011 [EBook #1661]
First Posted: November 29, 2002
Text File???
Bitmap File
JPEG File
# JPEG VS BITMAP

- **JPEG File**

<table>
<thead>
<tr>
<th>F</th>
<th>D8</th>
<th>FF</th>
<th>E0</th>
<th>00</th>
<th>10</th>
<th>4A</th>
<th>46</th>
<th>49</th>
<th>46</th>
<th>00</th>
<th>01</th>
<th>01</th>
<th>01</th>
<th>00</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>48</td>
<td>00</td>
<td>00</td>
<td>FF</td>
<td>DB</td>
<td>00</td>
<td>43</td>
<td>00</td>
<td>02</td>
<td>01</td>
<td>01</td>
<td>02</td>
<td>01</td>
<td>01</td>
<td>02</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>02</td>
<td>02</td>
<td>02</td>
<td>02</td>
<td>02</td>
<td>03</td>
<td>05</td>
<td>03</td>
<td>03</td>
<td>03</td>
<td>03</td>
<td>06</td>
<td>04</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>03</td>
<td>05</td>
<td>07</td>
<td>06</td>
<td>07</td>
<td>07</td>
<td>07</td>
<td>06</td>
<td>07</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>0B</td>
<td>09</td>
<td>08</td>
</tr>
<tr>
<td>8</td>
<td>0A</td>
<td>08</td>
<td>07</td>
<td>07</td>
<td>0A</td>
<td>0D</td>
<td>0A</td>
<td>0A</td>
<td>0B</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>07</td>
<td>09</td>
</tr>
<tr>
<td>E</td>
<td>0F</td>
<td>0D</td>
<td>0C</td>
<td>0E</td>
<td>0B</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>FF</td>
<td>DB</td>
<td>00</td>
<td>43</td>
<td>01</td>
<td>02</td>
<td>02</td>
</tr>
<tr>
<td>2</td>
<td>03</td>
<td>03</td>
<td>06</td>
<td>03</td>
<td>03</td>
<td>06</td>
<td>0C</td>
<td>08</td>
<td>07</td>
<td>08</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
</tr>
<tr>
<td>C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
</tr>
<tr>
<td>C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
</tr>
<tr>
<td>C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
<td>0C</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>08</td>
<td>03</td>
<td>00</td>
<td>04</td>
<td>00</td>
<td>03</td>
<td>01</td>
<td>22</td>
<td>00</td>
<td>02</td>
<td>11</td>
<td>01</td>
<td>03</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>FF</td>
<td>C4</td>
<td>00</td>
<td>1F</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>05</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
</tr>
<tr>
<td>A</td>
<td>0B</td>
<td>FF</td>
<td>C4</td>
<td>00</td>
<td>B5</td>
<td>10</td>
<td>00</td>
<td>02</td>
<td>01</td>
<td>03</td>
<td>03</td>
<td>02</td>
<td>04</td>
<td>03</td>
<td>05</td>
</tr>
<tr>
<td>5</td>
<td>04</td>
<td>04</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>7D</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>00</td>
<td>04</td>
<td>11</td>
<td>05</td>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>41</td>
<td>06</td>
<td>13</td>
<td>51</td>
<td>61</td>
<td>07</td>
<td>22</td>
<td>71</td>
<td>14</td>
<td>32</td>
<td>81</td>
<td>91</td>
<td>A1</td>
<td>08</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>B1</td>
<td>C1</td>
<td>15</td>
<td>52</td>
<td>D1</td>
<td>F0</td>
<td>24</td>
<td>33</td>
<td>62</td>
<td>72</td>
<td>82</td>
<td>09</td>
<td>0A</td>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>1A</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>2A</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>3A</td>
</tr>
</tbody>
</table>

JPEG and Bitmap images are two different file formats used for storing images. JPEG is a lossy compression format that reduces file size significantly, making it ideal for the web and other applications where high image quality is not critical. Bitmap, on the other hand, is a lossless format that preserves all image information and is best suited for graphic design and high-quality prints. Here are the differences:

**JPEG Advantages:**
- Smaller file sizes, which is crucial for web applications where bandwidth is a concern.
- Supports different levels of quality, allowing for a balance between file size and image quality.

**Bitmap Advantages:**
- Perfect quality, as no loss of information occurs during compression.
- Ideal for applications requiring high image fidelity, such as graphic design and print media.

In summary, JPEG is the go-to format for web applications, while Bitmap is preferred for high-quality printing and graphic design work.
Encoding Schemes

- "It's all 1s and 0s"
- What do the 1s and 0s mean?
- 50 121 109
- ASCII -> 2ym
- Red Green Blue->
dark teal?
Altering files

- Tower bit map. Alter the first 300 characters of line 16784 to all 0’s

000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000
000000000000000000000000000000000000000

Agenda

- Encoding
- Compression
- Huffman Coding
Compression

- Compression: Storing the same information but in a form that takes less memory
- lossless and lossy compression
- Recall:

<table>
<thead>
<tr>
<th>Filename</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tower-number1-1024x768.bmp</td>
<td>Bitmap image</td>
<td>2,305 KB</td>
</tr>
<tr>
<td>Tower-number1-1024x768.jpg</td>
<td>JPEG Image</td>
<td>283 KB</td>
</tr>
</tbody>
</table>
Lossy Artifacts
Why Bother?

- Is compression really necessary?

Toshiba Canvio® Connect
2TB Portable Hard Drive (Silver)
Item: 193434  Model: HDTC720XS3C1

2 Terabytes
500 HD, 2 hour movies or 500,000 songs
Price? About $100.00
Little Pipes and Big Pumps

Home Internet Access
- 40 Mbps roughly $40 per month
- 12 months * 3 years * $40 = $1,440
- 40,000,000 bits / second = 5.0 * 10^6 bytes / sec

CPU Capability
- $1,500 for a laptop or desktop
- Intel i7 processor
- Assume it lasts 3 years.
- Memory bandwidth 25.6 GB / sec = 2.6 * 10^10 bytes / sec
- on the order of 5.0 * 10^10 instructions / second
Mobile Devices?

Cellular Network

- Your mileage may vary …
- Mega bits per second
- AT&T
  - 17 mbps download, 7 mbps upload
- T-Mobile & Verizon
  - 12 mbps download, 7 mbps upload
- 17,000,000 bits per second = 2.125 x 10^6 bytes per second

iPhone CPU

- Apple A6 System on a Chip
- Coy about IPS
- 2 cores
- Rough estimates: 1 x 10^{10} instructions per second

http://tinyurl.com/q6o7wan
Little Pipes and Big Pumps

Data In
From Network

CPU
Compression - Why Bother?

- Apostolos "Toli" Lerios
- Facebook Engineer
- Heads image storage group
- jpeg images already compressed
- look for ways to compress even more
- 1% less space = millions of dollars in savings
Agenda

• Encoding
• Compression
• Huffman Coding
Purpose of Huffman Coding

• Proposed by Dr. David A. Huffman
  – A Method for the Construction of Minimum Redundancy Codes
  – Written in 1952

• Applicable to many forms of data transmission
  – Our example: text files
  – still used in fax machines, mp3 encoding, others
The Basic Algorithm

- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency!
- Yet in ASCII all characters are allocated the same amount of space
  - 1 char = 1 byte, be it e or X
The Basic Algorithm

- Any savings in tailoring codes to frequency of character?
- Code word lengths are no longer fixed like ASCII or Unicode
- Code word lengths vary and will be shorter for the more frequently used characters
The Basic Algorithm

1. Scan file to be compressed and tally occurrence of all values.

2. Sort or prioritize values based on frequency in file.

3. Build Huffman code tree based on prioritized values.

4. Perform a traversal of tree to determine new codes for values.

5. Scan file again to create new file using the new Huffman codes
Building a Tree
Scan the original text

- Consider the following short text

  Eerie eyes seen near lake.

- Count up the occurrences of all characters in the text
Eerie eyes seen near lake.

- What characters are present?
Building a Tree
Scan the original text

Eerie eyes seen near lake.

- What is the frequency of each character in the text?

<table>
<thead>
<tr>
<th>Char</th>
<th>Freq.</th>
<th>Char</th>
<th>Freq.</th>
<th>Char</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>e</td>
<td>8</td>
<td>y</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>s</td>
<td>2</td>
<td>n</td>
<td>2</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>i</td>
<td>1</td>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>space</td>
<td>4</td>
<td>l</td>
<td>1</td>
</tr>
</tbody>
</table>
Building a Tree
Prioritize characters

• Create binary tree nodes with a value and the frequency for each value
• Place nodes in a priority queue
  – The lower the frequency, the higher the priority in the queue
Building a Tree

- The queue after inserting all nodes

<table>
<thead>
<tr>
<th>front</th>
<th>back</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>sp</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>i</td>
<td>e</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>k</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>sp</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
</tr>
</tbody>
</table>

- Null Pointers are not shown
Building a Tree

• While priority queue contains two or more nodes
  – Create new node
  – Dequeue node and make it left subtree
  – Dequeue next node and make it right subtree
  – Frequency of new node equals sum of frequency of left and right children
  – Enqueue new node back into queue
Building a Tree

E 1
i 1
k 1
l 1
y 1
. 1
a 2
n 2
r 2
s 2
sp 4
e 8
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree
Building a Tree

```
2
 / 
\ / 
1 1

sp
4

4
 / 
\ / 
a 2
 / 
1 2

4
 / 
\ / 
r 2
 | 
 s 2
```

```
4
 / 
\ / 
2 2

E 1
 / 
\ / 
i 1
 | 
k 1
 | 
l 1
```
Building a Tree
Building a Tree

```
[4]
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
[2]  [2]
   an
[4]
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
[2]  [2]
   rs
[4]
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
[1]  [1]  [1]  [1]
   Eiik
[2]
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
[1]
   6
[4]
|   |
[1]  [1]
   y.
```
What is happening to the characters with a low frequency?
Building a Tree

4

2

E 1

i 1

k 1

l 1

2

6

sp 4

2

1

1

y 1

. 1

E 1

i 1

k 1

l 1

2

6

sp 4

8

4

8

4

a 2

n 2

r 2

s 2
Building a Tree
Building a Tree

```
8
  /\    \\
4   4
  /\    \\
2   2
```

```
E  i  k  l  y  .
  1   1   1   1   1
```

```
sp  4
```
Building a Tree

```
  e
   \   /
    8  /
     /  ∩
    4  4
   / \ / \  
 a  r  n  s
     \   /
      2  2
```

```
  8
   ∩
   ∩
  4  4
   /  /
  a  r  n  s
   / \ / \  
  2  2  2  2
```

```
  10
   ∩
   ∩
  4  4
   /  /
  E  i  k  y
   / \ / \ \ 
  1  1  1  1 .
```

```
  6
   /  ∩
  2  2  2
   / \ / \  
 sp i y
   / / \ \ \ 
  4 1 1 1 1
```
Building a Tree

Diagram of a tree structure with nodes labeled with letters and numbers.
Building a Tree

Diagram of a tree with nodes labeled with letters and numbers.
Building a Tree
• After enqueuing this node there is only one node left in priority queue.
Building a Tree

Dequeue the single node left in the queue.

This tree contains the new code words for each character.

Frequency of root node should equal number of characters in text.

Eerie eyes seen near lake. 4 spaces, 26 characters total
Encoding the File
Traverse Tree for Codes

- Perform a traversal of the tree to obtain new code words
- left, append a 0 to code word
- right append a 1 to code word
- code word is only completed when a leaf node is reached
Encoding the File
Traverse Tree for Codes

<table>
<thead>
<tr>
<th>Char</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>0000</td>
</tr>
<tr>
<td>i</td>
<td>0001</td>
</tr>
<tr>
<td>k</td>
<td>0010</td>
</tr>
<tr>
<td>l</td>
<td>0011</td>
</tr>
<tr>
<td>y</td>
<td>0100</td>
</tr>
<tr>
<td>.</td>
<td>0101</td>
</tr>
<tr>
<td>space</td>
<td>011</td>
</tr>
<tr>
<td>e</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>1100</td>
</tr>
<tr>
<td>n</td>
<td>1101</td>
</tr>
<tr>
<td>r</td>
<td>1110</td>
</tr>
<tr>
<td>s</td>
<td>1111</td>
</tr>
</tbody>
</table>
### Encoding the File

- Rescan text and encode file using new code words

Eerie eyes seen near lake.

<table>
<thead>
<tr>
<th>Char</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>0000</td>
</tr>
<tr>
<td>i</td>
<td>0001</td>
</tr>
<tr>
<td>k</td>
<td>0010</td>
</tr>
<tr>
<td>l</td>
<td>0011</td>
</tr>
<tr>
<td>.</td>
<td>0100</td>
</tr>
<tr>
<td>space</td>
<td>0101</td>
</tr>
<tr>
<td>e</td>
<td>0110</td>
</tr>
<tr>
<td>a</td>
<td>0111</td>
</tr>
<tr>
<td>n</td>
<td>1000</td>
</tr>
<tr>
<td>s</td>
<td>1001</td>
</tr>
<tr>
<td>n</td>
<td>1010</td>
</tr>
<tr>
<td>r</td>
<td>1011</td>
</tr>
<tr>
<td>s</td>
<td>1100</td>
</tr>
<tr>
<td>e</td>
<td>1101</td>
</tr>
<tr>
<td>a</td>
<td>1110</td>
</tr>
<tr>
<td>n</td>
<td>1111</td>
</tr>
</tbody>
</table>
• Have we made things any better?
• 82 bits to encode the text
• ASCII would take $8 \times 26 = 208$ bits

• If modified code used 4 bits per character are needed. Total bits $4 \times 26 = 104$. Savings not as great.
Decoding the File

• How does receiver know what the codes are?
• Tree constructed for each text file.
  – Considers frequency for each file
  – Big hit on compression, especially for smaller files
• Tree predetermined
  – based on statistical analysis of text files or file types
Decoding the File

- Once receiver has tree it scans incoming bit stream
- 0 ⇒ go left
- 1 ⇒ go right

```
1010001001111000111111
11011100001010
```

A. elk nay sir
B. eek a snake
C. eek kin sly
D. eek snarl nil
E. eel a snarl
Assignment Hints

• reading chunks not chars
• header format
• the pseudo eof character
• the GUI
Assignment Example

- "Eerie eyes seen near lake." will result in different codes than those shown in slides due to:
  - adding elements in order to PriorityQueue
  - required pseudo eof character (PEOF)
### Assignment Example

<table>
<thead>
<tr>
<th>Char</th>
<th>Freq.</th>
<th>Char</th>
<th>Freq.</th>
<th>Char</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>y</td>
<td>1</td>
<td>k</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>8</td>
<td>s</td>
<td>2</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>r</td>
<td>2</td>
<td>n</td>
<td>2</td>
<td>PEOF</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>a</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>space</td>
<td>4</td>
<td>l</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assignment Example
Assignment Example
Assignment Example
Assignment Example
Assignment Example
Assignment Example
Assignment Example
Assignment Example
Codes

value: 32, equivalent char: , frequency: 4, new code 011
value: 46, equivalent char: ., frequency: 1, new code 11110
value: 69, equivalent char: E, frequency: 1, new code 11111
value: 97, equivalent char: a, frequency: 2, new code 0101
value: 101, equivalent char: e, frequency: 8, new code 10
value: 105, equivalent char: i, frequency: 1, new code 0000
value: 107, equivalent char: k, frequency: 1, new code 0001
value: 108, equivalent char: l, frequency: 1, new code 0010
value: 110, equivalent char: n, frequency: 2, new code 1100
value: 114, equivalent char: r, frequency: 2, new code 1101
value: 115, equivalent char: s, frequency: 2, new code 1110
value: 121, equivalent char: y, frequency: 1, new code 0011
value: 256, equivalent char: ?, frequency: 1, new code 0100
"Hopefully, you've played around a bit with The Oracle of Bacon at Virginia and discovered how few steps are necessary to link just about anybody who has ever been in a movie to Kevin Bacon, but could there be some actor or actress who is even closer to the center of the Hollywood universe?.

By processing all of the almost half of a million people in the Internet Movie Database I discovered that there are currently 1160 people who are better centers than Kevin Bacon! … By computing the average of these numbers we see that the average (Sean) Connery Number is about 2.682 making Connery a better center than Bacon"

-Who is the Center of the Hollywood Universe?, University of Virginia

That was in 2001.

In 2013 Harvey Keitel has become the center of the Hollywood Universe. Connery is 136th. Bacon has moved up to 370th.
An Early Problem in Graph Theory

- Leonhard Euler (1707 - 1783)
  - One of the first mathematicians to study graphs
- The Seven Bridges of Konigsberg Problem
  - Konigsberg now called Kaliningrad
- A puzzle for the residents of the city
- The river Pregel flows through the city
- 7 bridges cross the river
- Can you cross all bridges while crossing each bridge only once?
Konigsberg and the River Pregel
Clicker Question 1

- How many solutions does the Seven Bridges of Konigsberg Problem have?

A. 0
B. 1
C. 2
D. 3
E. >= 4
How to Solve

- Brute Force?
- Euler's Solution
  - Redraw the map as a graph (really a multigraph)
Euler's Proposal

- A connected graph has an Euler tour (possible to cross each edge only once while traversing each edge only once and returning to starting point) if and only if every vertex has an even number of edges
  - Eulerian Circuit

- What if we reduce the problem to only crossing each edge (bridge) exactly once?
  - Doesn't matter if we end up where we started
    - Eulerian Trail
Graph Definitions

- A graph is comprised of a set of *vertices* (nodes) and a set of *edges* (links, arcs) connecting the vertices

- In a *directed* graph edges are one-way
  - movement allowed from first node to second, but not second to first
  - directed graphs also called *digraphs*

- In an *undirected* graph edges are two-way
  - movement allowed in either direction
Definitions

- In a weighted graph the edge has cost or weight that measures the cost of traveling along the edge.
- A path is a sequence of vertices connected by edges.
  - The path length is the number of edges.
  - The weighted path length is the sum of the cost of the edges in a path.
- A cycle is a path of length 1 or more that starts and ends at the same vertex.
  - a directed acyclic graph is a directed graph with no cycles.
Graphs We've Seen

Graphs

link → link → link → link

19 → 12 → 3
19 → 35 → 21 → 56

link → link → link
Example Graph

- Computer Scientists use graphs to model all kinds of things

Arpanet 1969, 1971
Example Graph

Roman Land Transportation Network
Example Graph

Enron emails 2001
Example Graph

US Airport Network
At Facebook's core is the social graph; people and the connections they have to everything they care about. The Graph API presents a simple, consistent view of the Facebook social graph, uniformly representing objects in the graph (e.g., people, photos, events, and pages) and the connections between them (e.g., friend relationships, shared content, and photo tags).

Every object in the social graph has a unique ID. You can access the properties of an object by requesting https://graph.facebook.com/ID. For example, the official page for the Facebook Platform has id 19292868552, so you can fetch the object at https://graph.facebook.com/19292868552:
Example Graph

"Jefferson" High School, Ohio and Sexual Networks, 2005, Chains of Affection: The Structure of Adolescent Romantic Graphs
Representing Graphs

- How to store a graph as a data structure?
Adjacency Matrix Representation

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>Br</th>
<th>Bl</th>
<th>Ch</th>
<th>Co</th>
<th>E</th>
<th>FG</th>
<th>G</th>
<th>Pa</th>
<th>Pe</th>
<th>S</th>
<th>U</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Br</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Bl</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ch</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Co</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FG</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Pa</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pe</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Country</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argentina</td>
<td>A</td>
</tr>
<tr>
<td>Brazil</td>
<td>Br</td>
</tr>
<tr>
<td>Bolivia</td>
<td>Bl</td>
</tr>
<tr>
<td>Chile</td>
<td>Ch</td>
</tr>
<tr>
<td>Columbia</td>
<td>Co</td>
</tr>
<tr>
<td>Ecuador</td>
<td>E</td>
</tr>
<tr>
<td>French Guiana</td>
<td>FG</td>
</tr>
<tr>
<td>Guyana</td>
<td>G</td>
</tr>
<tr>
<td>Paraguay</td>
<td>Pa</td>
</tr>
<tr>
<td>Peru</td>
<td>Pe</td>
</tr>
<tr>
<td>Suriname</td>
<td>S</td>
</tr>
<tr>
<td>Uruguay</td>
<td>U</td>
</tr>
<tr>
<td>Venezuela</td>
<td>V</td>
</tr>
</tbody>
</table>
The Map Coloring Problem

- How many colors do you need to color a map, so that no 2 countries that have a common border (not a point) are colored the same?
- How to solve using Brute Force?
A Solution

Green

Blue

Yellow

Green

Blue

Yellow

Red

Green

Blue

Yellow
What About the Ocean?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>Br</th>
<th>Bl</th>
<th>Ch</th>
<th>Co</th>
<th>E</th>
<th>FG</th>
<th>G</th>
<th>Pa</th>
<th>Pe</th>
<th>S</th>
<th>U</th>
<th>V</th>
<th>Oc</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Br</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Bl</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ch</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Co</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FG</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Pa</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pe</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Oc</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Make the Ocean Blue

Blue

Green

Red

Yellow

Green

Red

Yellow

Blue

Green

Red

Yellow

Blue
More Definitions

- A *dense* graph is one with a large number of edges
  - maximum number of edges?
- A *sparse* graph is one in which the number of edges is much less than the maximum possible number of edges
  - No standard cutoff for dense and sparse graphs
Graph Representation

- For dense graphs the adjacency matrix is a reasonable choice
  - For weighted graphs change booleans to cost
  - Can the adjacency matrix handle directed graphs?
- Most graphs are sparse, not dense
- For sparse graphs an *adjacency list* is an alternative that uses less space
- Each vertex keeps a list of vertices it is connected to.
Graph Implementation

public class Graph
{
    private static final double INFINITY = Double.MAX_VALUE;

    private Map<String, Vertex> vertices;

    public Graph() // create empty Graph
    {
    }

    public void addEdge(String source, String dest, double cost)
    {
    }

    // find all paths from given vertex
    public void findUnweightedShortestPath(String startName)
    {
    }

    // called after findUnweightedShortestPath
    public void printPath(String destName)
    {
    }
}
Graph Class

- This Graph class stores vertices
- Each vertex has an adjacency list
  - what vertices does it connect to?
- shortest path method finds all paths from start vertex to every other vertex in graph
- after shortest path method called queries can be made for path length from start node to destination node
**Vertex Class (nested in Graph)**

```java
private static class Vertex
    private String name;
    private List<Edge> adjacent;

    public Vertex(String n)

    // for shortest path algorithms
    private double double distance;
    private Vertex Vertex prev;
    private int int scratch;

    // call before finding new paths
    public void void reset()
```
private static class Edge
    private Vertex dest;
    private double cost;

    private Edge(Vertex d, double c)
Unweighted Shortest Path

- Given a vertex, S (for start) find the shortest path from S to all other vertices in the graph
- Graph is unweighted (set all edge costs to 1)
Word Ladders

- Agree upon dictionary
- Start word and end word of same length
- Change one letter at a time to form step
- Step must also be a word
- Example: Start = silly, end = funny

\[
\begin{array}{l}
\text{silly} \\
\text{sully} \\
\text{sulky} \\
\text{hulky} \\
\text{hunky} \\
\text{funky} \\
\text{funny}
\end{array}
\]
### Graph Representation

What are the vertices and when does an edge exist between two vertices?

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Letters</td>
<td>Words</td>
</tr>
<tr>
<td>B. Words</td>
<td>Words that share one or more letters</td>
</tr>
<tr>
<td>C. Letters</td>
<td>Words that share one or more letters</td>
</tr>
<tr>
<td>D. Words</td>
<td>Words that differ by one letter</td>
</tr>
<tr>
<td>E. Words</td>
<td>Letters</td>
</tr>
</tbody>
</table>
Portion of Graph
Size of Graph

- Number of vertices and edges depends on dictionary
- Modified Scrabble dictionary, 5 letter words
- Words are vertices
  - 8660 words
- Edge exists between word if they are one letter different
  - 24,942 edges

Is this graph sparse or dense?
A. Sparse
B. Dense

Max number of edges = \( N \times (N - 1) / 2 \)
37,493,470
Unweighted Shortest Path Algorithm

- Problem: Find the shortest word ladder between two words if one exists
- What kind of search should we use?

A. Breadth First Search
B. Depth First Search
C. Either one
Unweighted Shortest Path Algorithm

- Set distance of start to itself to 0
- Create a queue and add the start vertex
- while the queue is not empty
  - remove front
  - loop through all edges of current vertex
    - get node edge connects to
    - if this node has not been visited
      - sets its distance to current distance + 1
      - sets its previous node to current node
      - add new node to queue
Portion of Graph
Start at "smart" and enqueue it [smart]
Dequeue (smart), loop through edges [swart]
Dequeue (smart), loop through edges [swart, start]
Dequeque (smart), loop through edges
[swart, start, scart]
Dequeque (smart), loop through edges [swart, start, scart, smalt]
Dequeue (smart), loop through edges
[swart, start, scart, smallt, smarm]
Done with smart, dequeue (swart) [start, scart, smalt, smarm]
loop through edges of swart (start already present)
[start, scart, smalt, smarm]
loop through edges of swart (scart already present) [start, scart, smalt, smarm]
loop through edges of swart
[start, scart, smalt, smarm, swarm]
loop through edges of swart
[start, scart, smalt, smarm, swarm, sware]
Unweighted Shortest Path

- Implement method
- demo
- how is path printed?
- The diameter of a graph is the longest shortest path in the graph
- How to find?
- How to find center of graph?
  - vertex connected to the largest number of other vertices with the shortest average path length
Positive Weighted Shortest Path

- Edges in graph are weighted and all weights are positive
- Similar solution to unweighted shortest path
- Dijkstra's algorithm
- Edsger W. Dijkstra (1930–2002)
- UT Professor 1984 - 2000
- Algorithm developed in 1956 and published in 1959.
private static class Vertex
    private String name;
    private List<Edge> adjacent;

    public Vertex(String n)

    // for shortest path algorithms
    private double distance;
    private Vertex prev;
    private int scratch;

    // call before finding new paths
    public void reset()
Dijkstra's Algorithm

- Pick the start vertex
- Set the cost of the start vertex to 0 and all other vertices to INFINITY
- While there are unvisited vertices:
  - Let the current vertex be the lowest cost vertex that has not yet been visited
  - Mark current vertex as visited
  - For each edge from the current vertex
    - If the sum of the cost of the current vertex and the cost of the edge is less than the cost of the destination vertex
      - Update the cost of the destination vertex
      - Set the previous of the destination vertex to the current vertex
Dijkstra's Algorithm

- Example of a *Greedy Algorithm*
  - A Greedy Algorithm does what appears to be the best thing at each stage of solving a problem
- Gives best solution in Dijkstra's Algorithm
- Does NOT always lead to best answer
- Fair teams:
  - (10, 10, 8, 8, 8), 2 teams
- Making change with fewest coins
  - (1, 5, 10) 15 cents
  - (1, 5, 12) 15 cents
What is the lowest cost path from A to E?
A. 3
B. 17
C. 20
D. 28
E. 37
A is start vertex
Set cost of A to 0, all others to INFINITY
Place A in a priority queue
[(A,0)] pq
dequeue (A,0)
Mark A as visited
current vertex A:
loop through A's edges
if sum of cost from A to edge is less than current cost
update cost and prev
[ ] pq
A -> B, 0 + 1 < INFINITY
[(B,1)] pq
[(B,1)] pq
A -> C, 0 + 7 < INFINITY
[(B,1), (C, 7)] pq
[(B,1), (C, 7)] \text{ pq}
A \rightarrow G, 0 + 17 < \text{INFINITY}
[(B,1), (C, 7), (G, 17)] \text{ pq}
[(B,1), (C, 7), (G, 17)]

current vertex B:
loop through B's edges
if sum of cost from B to edge is less than current cost
update cost and prev
[(C, 7), (G, 17)] pq
B -> C, 1 + 3 < 7
update C's cost and previous
[(C, 4), (C, 7), (G, 17)] pq
[(C, 4), (C, 7), (G, 17)] pq
B -> D, 1 + 21 < INFINITY
[(C, 4), (C, 7), (G, 17), (D, 22)] pq
[(C, 4), (C, 7), (G, 17), (D, 22)]

pq

current vertex is C, cost 4

loop through C's edges
[((C, 7), (G, 17), (D, 22)] pq
C -> A, A already visited so skip
[(C, 7), (G, 17), (D, 22)] pq
C -> B, B already visited so skip
[(C, 7), (G, 17), (D, 22)] \( pq \)

C \( \rightarrow \) F, \( 4 + 3 < \text{INFINITY} \)

[(C, 7), (F, 7), (G, 17), (D, 22)] \( pq \)
[(C, 7), (F, 7), (G, 17), (D, 22)] pq

current vertex is C

Already visited so skip
[(F, 7), (G, 17), (D, 22)] pq
current vertex is F
loop through F's edges
[(G, 17), (D, 22)] pq

F -> C, already visited so skip
[(G, 17), (D, 22)] pq
F \to D, 7 + 4 < 22
update D's cost and previous
[(D, 11), (G, 17), (D, 22)] pq
[(D, 11), (G, 17), (D, 22)]

pq

current vertex is D

loop through D's edges
[(G, 17), (D, 22)] pq
D -> B, already visited so skip
[(G, 17), (D, 22)] \text{ pq}

D \rightarrow E, \ 11 + 6 < \text{ INFINITY}

update E's cost and previous

[(G, 17), (E, 17), (D, 22)] \text{ pq}
[ (G, 17), (E, 17), (D, 22)] pq
D -> F, already visited so skip
[(G, 17), (E, 17), (D, 22)] \text{pq}

D \rightarrow G, \ 11 + 5 < 17

update G's cost and previous

[(G, 16), (G, 17), (E, 17), (D, 22)] \text{pq}
[\((G, 17), (E, 17), (D, 22)\)]
pq

current vertex is G

loop though edges, already visited all neighbors
[(E, 17), (D, 22)] pq

current vertex is E

loop though edges, already visited all neighbors
No unvisited vertices.
Each Vertex stores cost (distance) of lowest cost path from start Vertex to itself and previous vertex in path from start vertex to itself.
Implementing Dijkstra's

- Create a Path class to allow for multiple distances to a given vertex

```java
private static class Path implements Comparable<Path> {
    private Vertex dest;
    private double cost;
}
```

- Use a priority queue of Paths to store the vertices and distances
Alternatives to Dijkstra's Algorithm

- A*, pronounced "A Star"
- A heuristic, goal of finding shortest weighted path from single start vertex to goal vertex
- Uses actual distance like Dijkstra's but also estimates remaining cost or distance
  - distance is set to current distance from start PLUS the estimated distance to the goal
- For example when finding a path between towns, estimate the remaining distance as the straight-line (as the crow files) distance between current location and goal.
Spanning Tree: A tree of edges that connects all the vertices in a graph.
Minimum Spanning Tree

- **Minimum Spanning Tree**: A spanning tree in a weighted graph with the lowest total cost
  - used in network design, taxonomy, Image registration, and more!

Cost of spanning tree shown?
A. 6  
B. 7  
C. 29  
D. 61  
E. None of These
Prim's Algorithm

- Pick a vertex arbitrarily from graph
  - In other words, it doesn't matter which one
- Add lowest cost edge between the tree and a vertex that is not part of the tree UNTIL every vertex is part of the tree
- Greedy Algorithm, very similar to Dijkstra's
Prim's Algorithm

Pick D as root
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree? 2 from D to A (or C)
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree? 2 from D to C (OR from A o B)
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree? 2 from A to B
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
5 from D to G
Lowest cost edge from tree to vertex not in Tree? 
1 from G to F
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree? 6 from G to E
Prim's Algorithm

Pick D as root
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree? 4 from D to F
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
3 from F to C
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
3 from C to B
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree? 1 from B to A
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
5 from D to G
Prim's Algorithm

Lowest cost edge from tree to vertex not in Tree?
6 from D to E
Prim's Algorithm

Cost of Spanning Tree?
Other Graph Algorithms

- Lots!
- [http://en.wikipedia.org/wiki/Category:Graph_algorithms](http://en.wikipedia.org/wiki/Category:Graph_algorithms)
"hash collision n. [from the techspeak] (var. `hash clash') When used of people, signifies a confusion in associative memory or imagination, especially a persistent one (see thinko).

True story: One of us was once on the phone with a friend about to move out to Berkeley. When asked what he expected Berkeley to be like, the friend replied: 'Well, I have this mental picture of naked women throwing Molotov cocktails, but I think that's just a collision in my hash tables.'"

- The Hacker's Dictionary
Jon was senior programmer on a large programming project.

Senior programmer spend a lot of time helping junior programmers.

Junior programmer to Jon: "I need help writing a sorting algorithm."
A Problem

From *Programming Pearls (Jon in Italics)*

*Why do you want to write your own sort at all? Why not use a sort provided by your system?*

I need the sort in the middle of a large system, and for obscure technical reasons, I can't use the system file-sorting program.

*What exactly are you sorting? How many records are in the file? What is the format of each record?*

The file contains at most ten million records; each record is a seven-digit integer.

*Wait a minute. If the file is that small, why bother going to disk at all? Why not just sort it in main memory?*

Although the machine has many megabytes of main memory, this function is part of a big system. I expect that I'll have only about a megabyte free at that point.

*Is there anything else you can tell me about the records?*

Each one is a seven-digit positive integer with no other associated data, and no integer can appear more than once.
Questions

- When did this conversation take place?
- What were they sorting?
- How do you sort data when it won't all fit into main memory?
- Speed of file i/o?
A Solution

/* phase 1: initialize set to empty */
for i = [0, n)
    bit[i] = 0

/* phase 2: insert present elements into the set */
for each i in the input file
    bit[i] = 1

/* phase 3: write sorted output */
for i = [0, n)
    if bit[i] == 1 write i on the output file
Some Structures so Far

- **ArrayLists**
  - O(1) access
  - O(N) insertion (average case), better at end
  - O(N) deletion (average case)

- **LinkedLists**
  - O(N) access
  - O(N) insertion (average case), better at front and back
  - O(N) deletion (average case), better at front and back

- **Binary Search Trees**
  - O(log N) access if balanced
  - O(log N) insertion if balanced
  - O(log N) deletion if balanced
Why are Binary Trees Better?

- Divide and Conquer
  - reducing work by a factor of 2 each time
- Can we reduce the work by a bigger factor? 10? 1000?
- An ArrayList does this in a way when accessing elements
  - but must use an integer value
  - each position holds a single element
Hash Tables

- Hash Tables overcome the problems of ArrayList while maintaining the fast access, insertion, and deletion in terms of N (number of elements already in the structure.)

- Hash tables use an array and hash functions to determine the index for each element.
Hash Functions

- Hash: "From the French hatcher, which means 'to chop'."
- *to hash* to mix randomly or shuffle (To cut up, to slash or hack about; to mangle)
- Hash Function: Take a large piece of data and reduce it to a smaller piece of data, usually a single integer.
  - A function or algorithm
  - The input need not be integers!
Hash Function

5/5/1967
555389085
5122466556
"Mike Scott"
scottm@gmail.net
"Isabelle"
Simple Example

- Assume we are using names as our key
  - take 3rd letter of name, take int value of letter
    (a = 0, b = 1, ...), divide by 6 and take remainder

- What does "Bellers" hash to?

- L -> 11 -> 11 % 6 = 5
Result of Hash Function

- Mike = (10 % 6) = 4
- Kelly = (11 % 6) = 5
- Olivia = (8 % 6) = 2
- Isabelle = (0 % 6) = 0
- David = (21 % 6) = 3
- Margaret = (17 % 6) = 5 (uh oh)
- Wendy = (13 % 6) = 1

This is an imperfect hash function. A perfect hash function yields a one to one mapping from the keys to the hash values.

What is the maximum number of values this function can hash perfectly?
Another Hash Function

- Assume the hash function for String adds up the Unicode value for each character.

```java
public int hashCode(String s) {
    int result = 0;
    for(int i = 0; i < s.length(); i++)
        result += s.charAt(i);
    return result;
}
```

- Hashcode for "DAB" and "BAD"?

A. 301 103
B. 4 4
C. 412 214
D. 5 5
E. 199 199
More on Hash Functions

- transform the key (which may not be an integer) into an integer value
- The transformation can use one of four techniques
  - Mapping
  - Folding
  - Shifting
  - Casting
Hashing Techniques

- Mapping
  - As seen in the example
  - integer values or things that can be easily converted to integer values in key

- Folding
  - partition key into several parts and the integer values for the various parts are combined
  - the parts may be hashed first
  - combine using addition, multiplication, shifting, logical exclusive OR
Shifting

More complicated with shifting

```java
int hashVal = 0;
int i = str.length() - 1;
while (i > 0)
    { hashVal = (hashVal << 1) + (int) str.charAt(i);
      i--;
    }
```

different answers for "dog" and "god"

Shifting may give a better range of hash values when compared to just folding

Casts

- Very simple
  - essentially casting as part of fold and shift when working with chars.
public int hashCode() {
    int h = hash;
    if (h == 0 && value.length > 0) {
        char[] val = value;
        int len = count;
        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
Mapping Results

- Transform hashed key value into a legal index in the hash table
- Hash table is normally uses an array as its underlying storage container
- Normally get location on table by taking result of hash function, dividing by size of table, and taking remainder
  
  \[ \text{index} = \text{key} \mod n \]
  
  n is size of hash table

  empirical evidence shows a prime number is best
  
  1000 element hash table, make 997 or 1009 elements
Mapping Results

"Isabelle" \rightarrow 230492619

hashCode method

230492619 \% 997 = 177

0 1 2 3 ............177................. 996

"Isabelle"
Handling Collisions

- What to do when inserting an element and already something present?
Open Addressing

- Could search forward or backwards for an open space

- **Linear probing:**
  - move forward 1 spot. Open?, 2 spots, 3 spots
  - reach the end?
  - When removing, insert a blank
  - null if never occupied, blank if once occupied

- **Quadratic probing**
  - 1 spot, 2 spots, 4 spots, 8 spots, 16 spots

- Resize when *load factor* reaches some limit
Closed Addressing: Chaining

- Each element of hash table be another data structure
  - linked list, balanced binary tree
  - More space, but somewhat easier
  - everything goes in its spot

- What happens when resizing?
  - Why don't things just collide again?
Hash Tables in Java

- **hashCode method in Object**
- **hashCode and equals**
  - "If two objects are equal according to the equals (Object) method, then calling the hashCode method on each of the two objects must produce the same integer result."
  - if you override equals you need to override hashCode

- Overriding one of equals and hashCode, but not the other, can cause logic errors that are difficult to track down.
Hash Tables in Java

- **HashTable class**
- **HashSet class**
  - implements Set interface with internal storage container that is a HashTable
  - compare to TreeSet class, internal storage container is a Red Black Tree
- **HashMap class**
  - implements the Map interface, internal storage container for keys is a hash table
Comparison

- Compare these data structures for speed:
  - Java HashSet
  - Java TreeSet
  - our naïve Binary Search Tree
  - our HashTable
  - Insert random ints
Clicker Question

What will be order from fastest to slowest?
A. HashSet TreeSet HashTable BST
B. HashSet HashTable TreeSet BST
C. TreeSet HashSet BST HashTable
D. HashTable HashSet BST TreeSet
E. None of these
"People in every direction
No words exchanged
No time to exchange
And all the little ants are marching
Red and Black antennas waving"

- *Ants Marching*, Dave Matthew's Band

"Welcome to L.A.'s Automated Traffic Surveillance and Control Operations Center. See, they use video feeds from intersections and specifically designed algorithms to predict traffic conditions, and thereby control traffic lights. So all I did was come up with my own... kick ass algorithm to sneak in, and now we own the place."

- Lyle, the Napster, (Seth Green), *The Italian Job*
Clicker Question 1

- 2000 elements are inserted one at a time into an initially empty binary search tree using the traditional, naive algorithm. What is the maximum possible height of the resulting tree?

A. 1  
B. 11  
C. 21  
D. 1999  
E. 2000
Binary Search Trees

- Average case and worst case Big O for
  - insertion
  - deletion
  - access

- Balance is important. Unbalanced trees give worse than $\log N$ times for the basic tree operations

- Can balance be guaranteed?
Red Black Trees

- A BST with more complex algorithms to ensure balance
- Each node is labeled as Red or Black.
- Path: A unique series of links (edges) traverses from the root to each node.
  - The number of edges (links) that must be followed is the path length
- In Red Black trees paths from the root to elements with 0 or 1 child are of particular interest
Paths to Single or Zero Child Nodes

- How many?

```
19
```
```
  12
```
```
    3
    16
    21
```
```
      56
```
```
  35
```
```
    1
```

Red Black Trees
Red Black Tree Rules

1. Every node is colored either Red or black
2. The root is black
3. If a node is red its children must be black. (a.k.a. the red rule)
4. Every path from a node to a null link must contain the same number of black nodes (a.k.a. the path rule)
Example of a Red Black Tree

- The root of a Red Black tree is black
- Every other node in the tree follows these rules:
  - Rule 3: If a node is Red, all of its children are Black
  - Rule 4: The number of Black nodes must be the same in all paths from the root node to null nodes
Red Black Tree?
Clicker Question 2

- Is the tree on the previous slide a binary search tree? Is it a red black tree?
  
  BST? Red-Black?
  
  A. No No
  B. No Yes
  C. Yes No
  D. Yes Yes

CS314

Red Black Trees
Red Black Tree?

Perfect?
Full?
Complete?
Clicker Question 3

Is the tree on the previous slide a binary search tree? Is it a red black tree?

BST? Red-Black?

A. No No
B. No Yes
C. Yes No
D. Yes Yes
Implications of the Rules

- If a Red node has any children, it must have two children and they must be Black. (Why?)
- If a Black node has only one child that child must be a Red leaf. (Why?)
- Due to the rules there are limits on how unbalanced a Red Black tree may become.
  – on the previous example may we hang a new node off of the leaf node that contains 0?
Properties of Red Black Trees

- If a Red Black Tree is complete, with all Black nodes except for Red leaves at the lowest level the height will be minimal, \( \sim \log N \).
- To get the max height for \( N \) elements there should be as many Red nodes as possible down one path and all other nodes are Black.
  - This means the max height would be \( < 2 \times \log N \).
  - See example on next slide.
Max Height Red Black Tree

![Red Black Tree Diagram]

- Red nodes are shown in red color.
- Black nodes are shown in black color.
- The tree has a maximum height of 14.
Maintaining the Red Black Properties in a Tree

- Insertions
- Must maintain rules of Red Black Tree.
- New Node always a leaf
  - can't be black or we will violate rule 4
  - therefore the new leaf must be red
  - If parent is black, done (trivial case)
  - if parent red, things get interesting because a red leaf with a red parent violates rule 3
Insertions with Red Parent - Child

Must modify tree when insertion would result in Red Parent - Child pair using color changes and rotations.
Case 1

- Suppose sibling of parent is Black.
  - by convention null nodes are black

- In the previous tree, true if we are inserting a
  3 or an 8.
  - What about inserting a 99? Same case?

- Let X be the new leaf Node, P be its Red
  Parent, S the Black sibling and G, P's and
  S's parent and X's grandparent
  - What color is G?
Case 1 - The Picture

Relative to G, X could be an *inside* or *outside* node.
Outside -> left left or right right moves
Inside -> left right or right left moves
Fixing the Problem

If X is an outside node a single rotation between P and G fixes the problem. A rotation is an exchange of roles between a parent and child node. So P becomes G's parent. Also must recolor P and G.
Single Rotation

Apparent rule violation?
Case 2

- What if $X$ is an inside node relative to $G$?
  - a single rotation will not work
- Must perform a double rotation
  - rotate $X$ and $P$
  - rotate $X$ and $G$
After Double Rotation

Apparent rule violation?
Case 3
Sibling is Red, not Black

Any problems?
Fixing Tree when S is **Red**

- Must perform single rotation between parent, P and grandparent, G, and then make appropriate color changes.
More on Insert

Problem: What if on the previous example G's parent had been red?

Easier to never let Case 3 ever occur!

On the way down the tree, if we see a node X that has 2 Red children, we make X Red and its two children black.
  – if recolor the root, recolor it to black
  – the number of black nodes on paths below X remains unchanged
  – If X's parent was Red then we have introduced 2 consecutive Red nodes. (violation of rule)
  – to fix, apply rotations to the tree, same as inserting node
Example of Inserting Sorted Numbers

1 2 3 4 5 6 7 8 9 10

Insert 1. A leaf so red. Realize it is root so recolor to black.
Insert 2

make 2 red. Parent is black so done.
Insert 3. Parent is red. Parent's sibling is black (null) 3 is outside relative to grandparent. Rotate parent and grandparent.
Insert 4

On way down see 2 with 2 red children. Recolor 2 red and children black. Realize 2 is root so color back to black

When adding 4 parent is black so done.
Insert 5

5's parent is red. Parent's sibling is black (null). 5 is outside relative to grandparent (3) so rotate parent and grandparent then recolor
Finish insert of 5

1  2  4
  |   |   |
  3  5 4

Red Black Trees
Insert 6

On way down see 4 with 2 red children. Make 4 red and children black. 4's parent is black so no problem.
Finishing insert of 6

6's parent is black so done.
7's parent is red. Parent's sibling is black (null). 7 is outside relative to grandparent (5) so rotate parent and grandparent then recolor.
Finish insert of 7
On way down see 6 with 2 red children. Make 6 red and children black. This creates a problem because 6's parent, 4, is also red. Must perform rotation.
Still Inserting 8

Rocolored now need to rotate
Finish inserting 8

Recolored now need to rotate

1
2
3
4
5
6
7
8
Insert 9

On way down see 4 has two red children so recolor 4 red and children black. Realize 4 is the root so recolor black
Finish Inserting 9

After rotations and recoloring
Insert 10

On way down see 8 has two red children so change 8 to red and children black
Again a rotation is needed.
Finish inserting 11

```
1  2  3  4  5  6  7  8  9  10  11
```

Red Black Trees
"You think you know when you can learn, are more sure when you can write even more when you can teach, but certain when you can program."

- Alan Perlis
Priority Queue

- Recall priority queue
  - elements enqueued based on priority
  - dequeue removes the highest priority item

- Options?
  - List? Binary Search Tree?

<table>
<thead>
<tr>
<th>Linked List enqueue</th>
<th>BST enqueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>B. O(N)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>C. O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>D. O(logN)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>E. O(1)</td>
<td>O(logN)</td>
</tr>
</tbody>
</table>
Another Option

- A heap
  - not to be confused with the runtime heap (portion of memory for dynamically allocated variables)

- A complete binary tree
  - all levels have maximum number of nodes except deepest where nodes are filled in from left to right

- Maintains the heap order property
  - in a min heap the value in the root of any subtree is less than or equal to all other values in the subtree
Clicker Question 2

In a max heap with no duplicates where is the largest value?

A. the root of the tree
B. in the left-most node
C. in the right-most node
D. a node in the lowest level
E. None of these
Example Min Heap

- 12
- 17
- 16
- 19
- 52
- 37
- 25
- 21
- 45
Enqueue Operation

- Add new element to next open spot in array
- Swap with parent if new value is less than parent
- Continue back up the tree as long as the new value is less than new parent node
Enqueue Example

- Add 15 to heap (initially next left most node)
Enqueue Example

- Swap 15 and 52
Enqueue Example

- Swap 15 and 17, then stop
Interestingly heaps are often implemented with an array instead of nodes.

For element at position i:
- Parent index: $i / 2$
- Left child index: $i \times 2$
- Right child index: $i \times 2 + 1$

Example array:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12</td>
<td>17</td>
<td>16</td>
<td>19</td>
<td>52</td>
<td>37</td>
<td>25</td>
<td>21</td>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
public class PriorityQueue<E extends Comparable<E>> {

    private int size;
    private E[] con;

    public PriorityQueue() {
        heap = getArray(2);
    }

    private E[] getArray(int size) {
        return (E[]) (new Comparable[size]);
    }
}
public void enqueue(E val) {
    if ( size >= con.length - 1 )
        enlargeArray( con.length * 2 + 1 );

    size++;
    int indexToPlace = size;
    while ( indexToPlace > 1
            && val.compareTo( con[indexToPlace / 2] ) < 0 ) {
        con[indexToPlace] = con[indexToPlace / 2]; // swap
        indexToPlace /= 2; // change indexToPlace to parent
    }
    con[indexToPlace] = val;
}

private void enlargeArray(int newSize) {
    E[] temp = getArray(newSize);
    System.arraycopy(con, 1, temp, 1, size);
    con = temp;
}
Dequeue

- min value / front of queue is in root of tree
- swap value from last node to root and move down swapping with smaller child unless values is smaller than both children
Dequeue Example

- Swap 35 into root (save 12 to return)
Dequeue Example

- Swap 35 into root (save 12 to return)
Dequeue Example

- Swap 35 with smaller child (15)
Dequeue Example

- Swap 35 with smaller child (17)
Dequeue Example

- Swap 35 with smaller child (21)
public E dequeue( ) {  
    E top = con[1];
    int hole = 1;
    boolean done = false;
    while ( hole * 2 < size && !done ) {  
        int child = hole * 2;
        // see which child is smaller
        if ( con[child].compareTo( con[child + 1] ) > 0 )  
            child++;    // child now points to smaller

        // is replacement value bigger than child?
        if (con[size].compareTo( con[child] ) > 0 ) {  
            con[hole] = con[child];
            hole = child;
        }
        else  
            done = true;
    }
    con[hole] = con[size];
    size--;
    return top;
}
PriorityQueue Comparison

- Run a Stress test of PQ implemented with Heap and PQ implemented with BinarySearchTree
- What will result be?

A. Heap takes half the time or less of BST
B. Heap faster, but not twice as fast
C. About the same
D. BST faster, but not twice as fast
E. BST takes half the time or less of Heap
Data Structures

- Data structures we have studied
  - arrays, array based lists, linked lists, maps, sets, stacks, queue, trees, binary search trees, graphs, hash tables, red-black trees, priority queues, heaps

- Most program languages have some built in data structures, native or library

- Must be familiar with performance of data structures
  - best learned by implementing them yourself
We have not covered every data structure.

- Bag (multiset)
- Set
- List
- Map
- Multiset
- Priority queue
- Queue
- Deque
- Stack
- String
- Tree
- Graph

Some properties of abstract data types:

<table>
<thead>
<tr>
<th>Structure</th>
<th>Stable</th>
<th>Unique</th>
<th>Cells per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bag (multiset)</td>
<td>no</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>Set</td>
<td>no</td>
<td>yes</td>
<td>1</td>
</tr>
<tr>
<td>List</td>
<td>yes</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>Map</td>
<td>no</td>
<td>yes</td>
<td>2</td>
</tr>
</tbody>
</table>

“Stable” means that input order is retained. Other str

Data Structures

- deque, b-trees, quad-trees, binary space partition trees, skip list, sparse list, sparse matrix, union-find data structure, Bloom filters, AVL trees, trie, 2-3-4 trees, and more!

- Must be able to learn new and apply new data structures
"Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities"

- Richard E. Bellman
Origins

- A method for solving complex problems by breaking them into smaller, easier, sub problems

- Term *Dynamic Programming* coined by mathematician Richard Bellman in early 1950s
  - employed by Rand Corporation
  - Rand had many, large military contracts
  - Secretary of Defense, Charles Wilson “against research, especially mathematical research”
  - how could any one oppose "dynamic"?
Dynamic Programming

- Break big problem up into smaller problems ...

- Sound familiar?

- Recursion?
  \[ N! = 1 \text{ for } N == 0 \]
  \[ N! = N \times (N - 1)! \text{ for } N > 0 \]
Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 114, ...
- $F_1 = 1$
- $F_2 = 1$
- $F_N = F_{N-1} + F_{N-2}$

Recursive Solution?
Failing Spectacularly

- Naïve recursive method

```java
// pre: n > 0
// post: return the nth Fibonacci number
public int fib(int n) {
    if (n <= 2)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

- Order of this method?
A. O(1)  B. O(log N)  C. O(N)  D. O(N^2)  E. O(2^N)
<table>
<thead>
<tr>
<th>Fibonacci Number</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.467E-6</td>
</tr>
<tr>
<td>2</td>
<td>4.47E-7</td>
</tr>
<tr>
<td>3</td>
<td>4.46E-7</td>
</tr>
<tr>
<td>4</td>
<td>4.46E-7</td>
</tr>
<tr>
<td>5</td>
<td>4.47E-7</td>
</tr>
<tr>
<td>6</td>
<td>4.47E-7</td>
</tr>
<tr>
<td>7</td>
<td>1.34E-6</td>
</tr>
<tr>
<td>8</td>
<td>1.787E-6</td>
</tr>
<tr>
<td>9</td>
<td>2.233E-6</td>
</tr>
<tr>
<td>10</td>
<td>3.573E-6</td>
</tr>
<tr>
<td>11</td>
<td>1.2953E-5</td>
</tr>
<tr>
<td>12</td>
<td>8.934E-6</td>
</tr>
<tr>
<td>13</td>
<td>2.9033E-5</td>
</tr>
<tr>
<td>14</td>
<td>3.7966E-5</td>
</tr>
<tr>
<td>15</td>
<td>5.0919E-5</td>
</tr>
<tr>
<td>16</td>
<td>7.1464E-5</td>
</tr>
<tr>
<td>17</td>
<td>1.08984E-4</td>
</tr>
<tr>
<td>Position</td>
<td>Fibonacci Number</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
</tr>
<tr>
<td>36th</td>
<td>14930352</td>
</tr>
<tr>
<td>37th</td>
<td>24157817</td>
</tr>
<tr>
<td>38th</td>
<td>39088169</td>
</tr>
<tr>
<td>39th</td>
<td>63245986</td>
</tr>
<tr>
<td>40th</td>
<td>102334155</td>
</tr>
<tr>
<td>41th</td>
<td>165580141</td>
</tr>
<tr>
<td>42th</td>
<td>267914296</td>
</tr>
<tr>
<td>43th</td>
<td>433494437</td>
</tr>
<tr>
<td>44th</td>
<td>701408733</td>
</tr>
<tr>
<td>45th</td>
<td>1134903170</td>
</tr>
<tr>
<td>46th</td>
<td>1836311903</td>
</tr>
<tr>
<td>47th</td>
<td>-1323752223</td>
</tr>
<tr>
<td>48th</td>
<td>512559680</td>
</tr>
<tr>
<td>49th</td>
<td>-811192543</td>
</tr>
<tr>
<td>50th</td>
<td>-298632863</td>
</tr>
<tr>
<td>51st</td>
<td>-1109825406</td>
</tr>
</tbody>
</table>
How long to calculate the 70th Fibonacci Number with this method?

A. 37 seconds
B. 74 seconds
C. 740 seconds
D. 14,800 seconds
E. None of these
Aside - Overflow

- at 47th Fibonacci number overflows int
- Could use BigInteger class instead

```java
private static final BigInteger one = new BigInteger("1");

private static final BigInteger two = new BigInteger("2");

public static BigInteger fib(BigInteger n) {
    if (n.compareTo(two) <= 0) {
        return one;
    } else {
        BigInteger firstTerm = fib(n.subtract(two));
        BigInteger secondTerm = fib(n.subtract(one));
        return firstTerm.add(secondTerm);
    }
}
```
Aside - BigInteger

- Answers correct beyond 46th Fibonacci number
- Even slower due to creation of so many objects

37th fibonacci number: 24157817 - Time: 2.406739213
38th fibonacci number: 39088169 - Time: 3.680196724
39th fibonacci number: 63245986 - Time: 5.941275208
40th fibonacci number: 102334155 - Time: 9.63855468
41th fibonacci number: 165580141 - Time: 15.659745756
42th fibonacci number: 267914296 - Time: 25.404417949
43th fibonacci number: 433494437 - Time: 40.867030512
44th fibonacci number: 701408733 - Time: 66.391845965
45th fibonacci number: 1134903170 - Time: 106.964369924
46th fibonacci number: 1836311903 - Time: 178.981819822
47th fibonacci number: 2971215073 - Time: 287.052365326
Slow Fibonacci

- Why so slow?
- Algorithm keeps calculating the same value over and over
- When calculating the 40\textsuperscript{th} Fibonacci number the algorithm calculates the 4\textsuperscript{th} Fibonacci number 24,157,817 times!!!
Fast Fibonacci

- Instead of starting with the big problem and working down to the small problems...
- ... start with the small problem and work up to the big problem

```java
public static BigInteger fastFib(int n) {
    BigInteger smallTerm = one;
    BigInteger largeTerm = one;
    for (int i = 3; i <= n; i++) {
        BigInteger temp = largeTerm;
        largeTerm = largeTerm.add(smallTerm);
        smallTerm = temp;
    }
    return largeTerm;
}
```
Fast Fibonacci

1th fibonacci number: 1 - Time: 4.467E-6
2th fibonacci number: 1 - Time: 4.47E-7
3th fibonacci number: 2 - Time: 7.146E-6
4th fibonacci number: 3 - Time: 2.68E-6
5th fibonacci number: 5 - Time: 2.68E-6
6th fibonacci number: 8 - Time: 2.679E-6
7th fibonacci number: 13 - Time: 3.573E-6
8th fibonacci number: 21 - Time: 4.02E-6
9th fibonacci number: 34 - Time: 4.466E-6
10th fibonacci number: 55 - Time: 4.467E-6
11th fibonacci number: 89 - Time: 4.913E-6
12th fibonacci number: 144 - Time: 6.253E-6
13th fibonacci number: 233 - Time: 6.253E-6
14th fibonacci number: 377 - Time: 5.806E-6
15th fibonacci number: 610 - Time: 6.7E-6
16th fibonacci number: 987 - Time: 7.146E-6
17th fibonacci number: 1597 - Time: 7.146E-6
## Fast Fibonacci

<table>
<thead>
<tr>
<th>Number</th>
<th>Fibonacci Number</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>45th</td>
<td>1134903170</td>
<td>1.7419E-5</td>
</tr>
<tr>
<td>46th</td>
<td>1836311903</td>
<td>1.6972E-5</td>
</tr>
<tr>
<td>47th</td>
<td>2971215073</td>
<td>1.6973E-5</td>
</tr>
<tr>
<td>48th</td>
<td>4807526976</td>
<td>2.3673E-5</td>
</tr>
<tr>
<td>49th</td>
<td>7778742049</td>
<td>1.9653E-5</td>
</tr>
<tr>
<td>50th</td>
<td>12586269025</td>
<td>2.01E-5</td>
</tr>
<tr>
<td>51st</td>
<td>20365011074</td>
<td>1.9207E-5</td>
</tr>
<tr>
<td>52th</td>
<td>32951280099</td>
<td>2.0546E-5</td>
</tr>
<tr>
<td>67th</td>
<td>44945570212853</td>
<td>2.3673E-5</td>
</tr>
<tr>
<td>68th</td>
<td>72723460248141</td>
<td>2.3673E-5</td>
</tr>
<tr>
<td>69th</td>
<td>117669030460994</td>
<td>2.412E-5</td>
</tr>
<tr>
<td>70th</td>
<td>190392490709135</td>
<td>2.4566E-5</td>
</tr>
<tr>
<td>71th</td>
<td>308061521170129</td>
<td>2.4566E-5</td>
</tr>
<tr>
<td>72th</td>
<td>498454011879264</td>
<td>2.5906E-5</td>
</tr>
<tr>
<td>73th</td>
<td>806515533049393</td>
<td>2.5459E-5</td>
</tr>
<tr>
<td>74th</td>
<td>1304969544928657</td>
<td>2.546E-5</td>
</tr>
<tr>
<td>200th</td>
<td>280571172992510140037611932413038677189525</td>
<td>1.0273E-5</td>
</tr>
</tbody>
</table>
Memoization

- Store (cache) results from functions for later lookup

- Memoization of Fibonacci Numbers

```java
public class FibMemo {

    private static List<BigInteger> lookupTable = new ArrayList<>();

    private static final BigInteger one = new BigInteger("1");

    static {
        // no fib for n == 0
        lookupTable.add(null);
        lookupTable.add(one);
        lookupTable.add(one);
    }
}
```
public static BigInteger fib(int n) {
    // check lookup table
    if (n < lookupTable.size())
        return lookupTable.get(n);

    // must calculate nth fibonacci
    // don't repeat work
    BigInteger smallTerm
        = lookupTable.get(lookupTable.size() - 2);
    BigInteger largeTerm
        = lookupTable.get(lookupTable.size() - 1);
    for (int i = lookupTable.size(); i <= n; i++) {
        BigInteger temp = largeTerm;
        largeTerm = largeTerm.add(smallTerm);
        lookupTable.add(largeTerm); // memo
        smallTerm = temp;
    }
    return largeTerm;
}
Dynamic Programming

- When to use?
  - When a big problem can be broken up into sub problems.

- **Solution to original problem can be calculated from results of smaller problems.**

- Sub problems have a natural ordering from smallest to largest OR simplest to hardest.
  - larger problems depend on previous solutions

- Multiple techniques within DP
DP Algorithms

- Step 1: Define the *meaning* of the subproblems (in English for sure, Mathematically as well if you find it helpful).
- Step 2: Show where the solution will be found.
- Step 3: Show how to set the first subproblem.
- Step 4: Define the order in which the subproblems are solved.
- Step 5: Show how to compute the answer to each subproblem using the previously computed subproblems. (This step is typically polynomial, once the other subproblems are solved.)
Dynamic Programming Example

- Another simple example
- Finding the best solution involves finding the best answer to simpler problems
- Given a set of coins with values \( (V_1, V_2, \ldots, V_N) \) and a target sum \( S \), find the fewest coins required to equal \( S \)
- What is Greedy Algorithm approach?
- Does it always work?
- \( \{1, 5, 12\} \) and target sum = 15
- Could use recursive backtracking …
Minimum Number of Coins

- To find minimum number of coins to sum to 15 with values \{1, 5, 12\} start with sum 0
  - recursive backtracking would likely start with 15
- Let \( M(S) = \) minimum number of coins to sum to \( S \)
- At each step look at target sum, coins available, and previous sums
  - pick the smallest option
Minimum Number of Coins

- $M(0) = 0$ coins
- $M(1) = 1$ coin (1 coin)
- $M(2) = 2$ coins (1 coin + $M(1)$)
- $M(3) = 3$ coins (1 coin + $M(2)$)
- $M(4) = 4$ coins (1 coin + $M(3)$)
- $M(5) =$ interesting, 2 options available:
  - 1 + others OR single 5
  - if 1 then 1 + $M(4) = 5$, if 5 then 1 + $M(0) = 1$
  - clearly better to pick the coin worth 5
Minimum Number of Coins

- $M(0) = 0$
- $M(1) = 1$ (1 coin)
- $M(2) = 2$ (1 coin + $M(1)$)
- $M(3) = 3$ (1 coin + $M(2)$)
- $M(4) = 4$ (1 coin + $M(3)$)
- $M(5) = 1$ (1 coin + $M(0)$)
- $M(6) = 2$ (1 coin + $M(5)$)
- $M(7) = 3$ (1 coin + $M(6)$)
- $M(8) = 4$ (1 coin + $M(7)$)
- $M(9) = 5$ (1 coin + $M(8)$)
- $M(10) = 2$ (1 coin + $M(5)$) options: 1, 5
- $M(11) = 2$ (1 coin + $M(10)$) options: 1, 5
- $M(12) = 1$ (1 coin + $M(0)$) options: 1, 5, 12
- $M(13) = 2$ (1 coin + $M(12)$) options: 1, 12
- $M(14) = 3$ (1 coin + $M(13)$) options: 1, 12
- $M(15) = 3$ (1 coin + $M(10)$) options: 1, 5, 12

CS314 Dynamic Programming 22
KNAPSACK PROBLEM - RECURSIVE BACKTRACKING AND DYNAMIC PROGRAMMING
Knapsack Problem

- A *bin packing* problem
- Similar to fair teams problem from recursion assignment
- You have a set of items
- Each item has a weight and a value
- You have a knapsack with a weight limit
- Goal: Maximize the *value* of the items you put in the knapsack without exceeding the weight limit
Knapsack Example

- Items:

<table>
<thead>
<tr>
<th>Item Number</th>
<th>Weight of Item</th>
<th>Value of Item</th>
<th>Value per unit Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6.0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>11</td>
<td>5.5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>12</td>
<td>3.0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>19</td>
<td>3.167</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>12</td>
<td>1.714</td>
</tr>
</tbody>
</table>

- Weight Limit = 8

- One greedy solution: Take the highest ratio item that will fit: (1, 6), (2, 11), and (4, 12)

- Total value = 6 + 11 + 12 = 29

- Is this optimal?  A. No  B. Yes
private static int knapsack(ArrayList<Item> items, int current, int capacity) {

    int result = 0;
    if (current < items.size()) {
        // don't use item
        int withoutItem = knapsack(items, current + 1, capacity);
        int withItem = 0;
        // if current item will fit, try it
        Item currentItem = items.get(current);
        if (currentItem.weight <= capacity) {
            withItem += currentItem.value;
            withItem += knapsack(items, current + 1, capacity - currentItem.weight);
        }
        result = Math.max(withoutItem, withItem);
    }
    return result;
}
Knapsack - Dynamic Programming

- Recursive backtracking starts with max capacity and makes choice for items: choices are:
  - take the item if it fits
  - don't take the item
- Dynamic Programming, start with simpler problems
- Reduce number of items available
- AND Reduce weight limit on knapsack
- Creates a 2d array of possibilities
Knapsack - Optimal Function

- OptimalSolution(items, weight) is best solution given a subset of items and a weight limit

2 options:

- OptimalSolution does not select $i^{th}$ item
  - select best solution for items 1 to $i - 1$ with weight limit of $w$

- OptimalSolution selects $i^{th}$ item
  - New weight limit = $w -$ weight of $i^{th}$ item
  - select best solution for items 1 to $i - 1$ with new weight limit
Knapsack Optimal Function

- \( \text{OptimalSolution}(\text{items, weight limit}) = \)

  0 if 0 items

  \( \text{OptimalSolution}(\text{items} - 1, \text{weight}) \) if weight of \( i \)th item is greater than allowed weight \( w_i > w \) (In others \( i \)th item doesn't fit)

  \( \max \) of \( (\text{OptimalSolution}(\text{items} - 1, w), \) value of \( i \)th item + \\
  \text{OptimalSolution}(\text{items} - 1, w - w_i) \)
Knapsack - Algorithm

- Create a 2d array to store value of best option given subset of items and possible weights

- In our example 0 to 6 items and weight limits of 0 to 8

- Fill in table using OptimalSolution Function
Knapsack Algorithm

Given N items and WeightLimit

Create Matrix M with N + 1 rows and WeightLimit + 1 columns

For weight = 0 to WeightLimit
    M[0, w] = 0

For item = 1 to N
    for weight = 1 to WeightLimit
        if(weight of ith item > weight)
            M[item, weight] = M[item - 1, weight]
        else
            M[item, weight] = max of M[item - 1, weight] AND value of item + M[item - 1, weight - weight of item]
## Knapsack - Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Weight</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>items / capacity</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>{1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1,2}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3, 4}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5, 6}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>items / weight</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>---------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>{}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>{1}</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>[1, 6]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2}</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>[2, 11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3}</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>[4, 1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>{1, 2, 3, 4}</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>[4, 12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5}</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>[6, 19]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5, 6}</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>[7, 12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Knapsack - Items to Take

<table>
<thead>
<tr>
<th>items / weight</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>{1} [1, 6]</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>{1,2} [2, 11]</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>{1, 2, 3} [4, 1]</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>{1, 2, 3, 4} [4, 12]</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5} [6, 19]</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5, 6} [7, 12]</td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>17</td>
<td>17</td>
<td>18</td>
<td>23</td>
<td>29</td>
<td>30</td>
</tr>
</tbody>
</table>
public static int knapsack(ArrayList<Item> items, int maxCapacity) {
    final int ROWS = items.size() + 1;
    final int COLS = maxCapacity + 1;
    int[][][] partialSolutions = new int[ROWS][COLS];

    for(int item = 1; item <= items.size(); item++) {
        for(int capacity = 0; capacity <= maxCapacity; capacity++) {
            Item currentItem = items.get(item - 1);
            int best = partialSolutions[item - 1][capacity];
            if(currentItem.weight <= capacity) {
                int withItem = currentItem.value;
                int capLeft = capacity - currentItem.weight;
                withItem += partialSolutions[item - 1][capLeft];
                if(withItem > best)
                    best = withItem;
            }
            partialSolutions[item][capacity] = best;
        }
    }
    return partialSolutions[ROWS - 1][COLS - 1];
}
<table>
<thead>
<tr>
<th>Number of items</th>
<th>Capacity</th>
<th>Recursive knapsack</th>
<th>Time</th>
<th>Dynamic knapsack</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>258</td>
<td>Answer: 433</td>
<td>111.77610595</td>
<td>Answer: 433</td>
<td>2.6353E-5</td>
</tr>
<tr>
<td>35</td>
<td>199</td>
<td>Answer: 318</td>
<td>154.049166387</td>
<td>Answer: 318</td>
<td>2.3673E-5</td>
</tr>
<tr>
<td>37</td>
<td>238</td>
<td>Answer: 411</td>
<td>636.560835011</td>
<td>Answer: 411</td>
<td>3.5285E-5</td>
</tr>
<tr>
<td>38</td>
<td>308</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is FP?

- **functional programming**: A style of programming that emphasizes the use of **functions** (methods) to decompose a complex task into subtasks.
  - Examples of functional languages:
    - LISP, Scheme, ML, Haskell, Erlang, F#, Clojure, ...

- Java is considered an object-oriented language, not a functional language.

- But Java 8 adds several language features to facilitate a partial functional programming style.
Java 8 FP features

• 1. Effect-free programming

• 2. Processing structured data via functions

• 3. First-class functions

• 4. Function closures

• 5. Higher-order operations on collections
Effect-free code (19.1)

- **side effect**: A change to the state of an object or program variable produced by a call on a function (i.e., a method).
  - example: modifying the value of a variable
  - example: printing output to System.out
  - example: reading/writing data to a file, collection, or network

```java
int result = f(x) + f(x);
int result = 2 * f(x);
```

- Are the two above statements equivalent?
  - Yes, if the function f() has no *side effects*.
  - One goal of functional programming is to minimize side effects.
public class SideEffect {
    public static int x;

    public static int f(int n) {
        x = x * 2;
        return x + n;
    }

    // what if it were 2 * f(x)?
    public static void main(String[] args) {
        x = 5;
        int result = f(x) + f(x);
        System.out.println(result);
    }
}
First-class functions (19.2)

- **first-class citizen**: An element of a programming language that is tightly integrated with the language and supports the full range of operations generally available to other entities in the language.

- In functional programming, functions (methods) are treated as first-class citizens of the languages.
  - can store a function in a variable
  - can pass a function as a parameter to another function
  - can return a value from a function
  - can create a collection of functions
  - ...
Lambda expressions

• **lambda expression** ("lambda"): Expression that describes a function by specifying its parameters and return value.
  – Java 8 adds support for lambda expressions.

• Syntax: 
  
  \[(parameters) \rightarrow expression\]

• Example:
  
  \( (x) \rightarrow x \times x \)  // squares a number

  – The above is roughly equivalent to:

  ```java
  public static int squared(int x) {
    return x * x;
  }
  ```
• Recall the `MathMatrix` class:

```java
public MathMatrix add(MathMatrix rhs) {
    int[][] res = new int[cells.length][cells[0].length];
    for (int r = 0; r < res.length; r++)
        for (int c = 0; c <= res[0].length; c++)
            res[r][c] = cells[r][c] + rhs.cells[r][c];
    return new MathMatrix(res);
}

public MathMatrix subtract rhs) {
    int[][] res = new int[cells.length][cells[0].length];
    for (int r = 0; r < res.length; r++)
        for (int c = 0; c <= res[0].length; c++)
            res[r][c] = cells[r][c] - rhs.cells[r][c];
    return new MathMatrix(res);
}
```
• GACK!!!

• How do we generalize the idea of "add or subtract"?
  – How much work would it be to add other operators?
  – Would functional programming help?
• We can represent the math operation as a lambda:

```java
public MathMatrix add(MathMatrix rhs) {
    return getMat(rhs, (x, y) -> x + y);
}

public MathMatrix subtract(MathMatrix rhs) {
    return getMat(rhs, (x, y) -> x - y);
}
```
private MathMatrix getMat(MathMatrix rhs, IntBinaryOperator operator) {

    int[][] res = new int[cells.length][cells[0].length];

    for (int r = 0; r < cells.length; r++) {
        for (int c = 0; c < cells[0].length; c++) {
            int temp1 = cells[r][c];
            int temp2 = rhs.cells[r][c];
            res[r][c] = operator.applyAsInt(temp1, temp2);
        }
    }

    return new MathMatrix(res);
}

// IntBinaryOperator Documentation
Streams (19.3)

- **stream**: A sequence of elements from a data source that supports aggregate operations.

- Streams operate on a data source and modify it:

  - example: print each element of a collection
  - example: sum each integer in a file
  - example: concatenate strings together into one large string
  - example: find the largest value in a collection
  - ...

```plaintext
source → stream1 → modifier → stream2 → ... → terminator
```
• Non-functional programming sum code:

```java
// compute the sum of the squares of integers 1-5
int sum = 0;
for (int i = 1; i <= 5; i++) {
    sum = sum + i * i;
}
```
The map modifier

- The `map` modifier applies a lambda to each stream element:
  - **higher-order function**: Takes a function as an argument.
- Abstracting away loops

```java
// compute the sum of the squares of integers 1-5
int sum = IntStream.range(1, 6)
    .map(n -> n * n)
    .sum();
```

// the stream operations are as follows:

```
IntStream.range(1, 6) -> [1, 2, 3, 4, 5]
    -> map -> [1, 4, 9, 16, 25]
    -> sum -> 55
```
The filter stream modifier removes/keeps elements of the stream using a boolean lambda:

```java
// compute the sum of squares of odd integers
int sum =
    IntStream.of(3, 1, 4, 1, 5, 9, 2, 6, 5, 3)
    .filter(n -> n % 2 != 0)
    .map(n -> n * n)
    .sum();
```

// the stream operations are as follows:
IntStream.of  -> [3, 1, 4, 1, 5, 9, 2, 6, 5, 3]
    -> filter  -> [3, 1, 1, 5, 9, 5, 3]
    -> map     -> [9, 1, 1, 25, 81, 25, 9]
    -> sum     -> 151
• using streams as part of a regular method:

```java
// Returns true if the given integer is prime. // Assumes n >= 0.
public static boolean isPrime(int n) {
    return IntStream.range(1, n + 1)
        .filter(x -> n % x == 0)
        .count() == 2;
}
```
The reduce modifier combines elements of a stream using a lambda combination function.

- Accepts two parameters: an initial value and a lambda to combine that initial value with each next value in the stream.

```java
// Returns n!, or 1 * 2 * 3 * ... * (n-1) * n.
// Assumes n is non-negative.
public static int factorial(int n) {
    return IntStream.range(2, n + 1)
        .reduce(1, (a, b) -> a * b);
}
```
## Stream operators

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>anyMatch(f)</code></td>
<td>returns true if any elements of stream match given predicate</td>
</tr>
<tr>
<td><code>allMatch(f)</code></td>
<td>returns true if all elements of stream match given predicate</td>
</tr>
<tr>
<td><code>average()</code></td>
<td>returns arithmetic mean of numbers in stream</td>
</tr>
<tr>
<td><code>collect(f)</code></td>
<td>convert stream into a collection and return it</td>
</tr>
<tr>
<td><code>count()</code></td>
<td>returns number of elements in stream</td>
</tr>
<tr>
<td><code>distinct()</code></td>
<td>returns unique elements from stream</td>
</tr>
<tr>
<td><code>filter(f)</code></td>
<td>returns the elements that match the given predicate</td>
</tr>
<tr>
<td><code>forEach(f)</code></td>
<td>performs an action on each element of stream</td>
</tr>
<tr>
<td><code>limit(size)</code></td>
<td>returns only the next <code>size</code> elements of stream</td>
</tr>
<tr>
<td><code>map(f)</code></td>
<td>applies the given function to every element of stream</td>
</tr>
<tr>
<td><code>noneMatch(f)</code></td>
<td>returns true if zero elements of stream match given predicate</td>
</tr>
</tbody>
</table>
# Stream operators

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel()</td>
<td>returns a multithreaded version of this stream</td>
</tr>
<tr>
<td>peek(f)</td>
<td>examines the first element of stream only</td>
</tr>
<tr>
<td>reduce(f)</td>
<td>applies the given binary reduction function to stream elements</td>
</tr>
<tr>
<td>sequential()</td>
<td>single-threaded, opposite of parallel()</td>
</tr>
<tr>
<td>skip(n)</td>
<td>omits the next n elements from the stream</td>
</tr>
<tr>
<td>sorted()</td>
<td>returns stream's elements in sorted order</td>
</tr>
<tr>
<td>sum()</td>
<td>returns sum of elements in stream</td>
</tr>
<tr>
<td>toArray()</td>
<td>converts stream into array</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>concat(s1, s2)</td>
<td>glues two streams together</td>
</tr>
<tr>
<td>empty()</td>
<td>returns a zero-element stream</td>
</tr>
<tr>
<td>iterate(seed, f)</td>
<td>returns an infinite stream with given start element</td>
</tr>
<tr>
<td>of(values)</td>
<td>converts the given values into a stream</td>
</tr>
<tr>
<td>range(start, end)</td>
<td>returns a range of integer values as a stream</td>
</tr>
</tbody>
</table>
Optional results

• Some stream terminators like max return an "optional" result because the stream might be empty or not contain the result:

```java
// print largest multiple of 10 in list
// (does not compile!)
int largest =
    IntStream.of(55, 20, 19, 31, 40, -2, 62, 30)
    .filter(n -> n % 10 == 0)
    .max();
System.out.println(largest);
```
Optional results fix

• To extract the optional result, use a "get as" terminator.
  – Converts type OptionalInt to Integer

// print largest multiple of 10 in list
// (this version compiles and works.)
int largest =
    IntStream.of(55, 20, 19, 31, 40, -2, 62, 30)
    .filter(n -> n % 10 == 0)
    .max()
    .getAsInt();
System.out.println(largest);
Stream exercises

• Write a method `sumAbsVals` that uses stream operations to compute the sum of the absolute values of an array of integers. For example, the sum of \{-1, 2, -4, 6, -9\} is 22.

• Write a method `largestEven` that uses stream operations to find and return the largest even number from an array of integers. For example, if the array is \{5, -1, 12, 10, 2, 8\}, your method should return 12. You may assume that the array contains at least one even integer.
Closures (19.4)

- **bound/free variable**: In a lambda expression, parameters are bound variables while variables in the outer containing scope are free variables.
- **function closure**: A block of code defining a function along with the definitions of any free variables that are defined in the containing scope.

```java
// free variables: min, max, multiplier
// bound variables: x, y
int min = 10;
int max = 50;
int multiplier = 3;
compute((x, y) -> Math.max(x, min) * Math.max(y, max) * multiplier);
```
Streams and arrays

- An array can be converted into a stream with Arrays.stream:

```java
// compute sum of absolute values of even ints
int[] numbers = {3, -4, 8, 4, -2, 17,
                 9, -10, 14, 6, -12};
int sum = Arrays.stream(numbers)
            .map(n -> Math.abs(n))
            .filter(n -> n % 2 == 0)
            .distinct()
            .sum();
```
A method reference lets you pass a method where a lambda would otherwise be expected:

```java
// compute sum of absolute values of even ints
int[] numbers = {3, -4, 8, 4, -2, 17,
                 9, -10, 14, 6, -12};
int sum = Arrays.stream(numbers)
    .map(Math::abs)
    .filter(n -> n % 2 == 0)
    .distinct()
    .sum();
```
A collection can be converted into a stream by calling its `stream` method:

```java
// compute sum of absolute values of even ints
ArrayList<Integer> list =
    new ArrayList<Integer>();
list.add(-42);
list.add(-17);
list.add(68);
list.stream()
    .map(Math::abs)
    .forEach(System.out::println);
```
// convert into set of lowercase words
List<String> words = Arrays.asList(
    "To", "be", "or", "Not", "to", "be");
Set<String> words2 = words.stream()
    .map(String::toLowerCase)
    .collect(Collectors.toSet());
System.out.println("word set = " + words2);

output:
word set = [not, be, or, to]
// find longest line in the file
int longest = Files.lines(Paths.get("haiku.txt"))
    .mapToLong(String::length)
    .max()
    .getAsInt();

stream operations:
Files.lines -> ["haiku are funny",
               "but sometimes they don't make sense",
               "refrigerator"]
               -> mapToLong -> [15, 35, 12]
               -> max -> 35
Stream exercises

• Write a method `fourLetterWords` that accepts a file name as a parameter and returns a count of the number of unique lines in the file that are exactly four letters long. Assume that each line in the file contains at least one word.

• Write a method using streams that finds and prints the first 5 perfect numbers. (Recall a perfect number is equal to the sum of its unique integer divisors, excluding itself.)