"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
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

```java
/*
 * 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?
Attendance Question 1

- 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?
Attendance Question 2

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(logN)
E. O(N log N)
Generic Binary Search

genericBinarySearch(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
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

Algorithm
- 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

44 68 191 119 119 37 83 82 191 45 158 130 76 153 39 25

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

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…
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