Topic 14
Searching and Simple Sorts

"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

- Given a 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

```java
/* pre: list != null
post: return the index of the first occurrence
       of target in list or -1 if target not present in
       list */
public int linearSearch(int[] list, int target) {
    for(int i = 0; i < list.length; i++)
        if(list[i] == target)
            return i;
    return -1;
}
```

Linear Search, Generic

```java
/* pre: list != null
post: return the index of the first occurrence
       of target in list or -1 if target not present in
       list */
public int linearSearch(Object[] list, Object target) {
    for(int i = 0; i < list.length; i++)
        if(target.equals(list[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?
  A. O(N)
  B. O(N^2)
  C. O(1)
  D. O(logN)
  E. O(NlogN)

Searching in a Sorted 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

<table>
<thead>
<tr>
<th>list</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>low item</td>
<td>middle item</td>
<td>high item</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>list</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>middle</td>
<td>high</td>
</tr>
<tr>
<td>item</td>
<td>item</td>
<td>item</td>
</tr>
</tbody>
</table>

and so forth...

Trace When Key == 3
Trace When Key == 30

Variables of Interest?

Binary Search in Action

```
public static int bsearch(int[] list, int target) {
    int result = -1;
    int low = 0;
    int high = list.length - 1;
    while (result == -1 && low <= high) {
        int mid = low + ((high - low) / 2);
        if (list[mid] == target)
            result = mid;
        else if (list[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return result;
}
```  

// mid = (low + high) / 2; // may overflow!!!  
// or mid = (low + high) >> 1; using bitwise op

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(NlogN)
Generic Binary Search
public static int bsearch(Comparable[] list, Comparable target) {
    int result = -1;
    int low = 0;
    int high = list.length - 1;
    while (result == -1 && low <= high) {
        int mid = low + ((high - low) / 2);
        int compareResult = target.compareTo(list[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[] list, int target) {
    return bsearch(list, target, 0, list.length - 1);
}

public static int bsearch(int[] list, int target, int low, int high) {
    if (low <= high) {
        int mid = low + ((high - low) / 2);
        if (list[mid] == target)
            return mid;
        else if (list[mid] > target)
            return bsearch(list, target, low, mid - 1);
        else
            return bsearch(list, 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
- Grover's Algorithm (Waiting for quantum computers to be built)
- best-first
- A*

Sorting

<table>
<thead>
<tr>
<th>U.S. All-time List - Marathon</th>
</tr>
</thead>
<tbody>
<tr>
<td>As of 4/24/08</td>
</tr>
<tr>
<td>Women</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1 1 1 2:19:26 Costa Faniottie (Italy)</td>
</tr>
<tr>
<td>2 2 22:16 Cristal (2)</td>
</tr>
<tr>
<td>3 2 22:16 Jen Becht-Samuelson</td>
</tr>
<tr>
<td>4 2 22:30 Antin (1)</td>
</tr>
<tr>
<td>5 2 24:29 Antin (2)</td>
</tr>
<tr>
<td>6 2 24:51 Antin (3)</td>
</tr>
<tr>
<td>7 2 24:51 Antin (4)</td>
</tr>
<tr>
<td>8 3 25:38 Lara Brown</td>
</tr>
<tr>
<td>9 4 25:44 Kim James</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Men</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 2:17:54 Jurgen Danker (Italy)</td>
</tr>
<tr>
<td>2 2 22:06 Michael (1)</td>
</tr>
<tr>
<td>3 2 22:16 Jen Becht-Samuelson</td>
</tr>
<tr>
<td>4 2 22:30 Antin (1)</td>
</tr>
<tr>
<td>5 2 24:29 Antin (2)</td>
</tr>
<tr>
<td>6 2 24:51 Antin (3)</td>
</tr>
<tr>
<td>7 2 24:51 Antin (4)</td>
</tr>
<tr>
<td>8 3 25:38 Lara Brown</td>
</tr>
<tr>
<td>9 4 25:44 Kim James</td>
</tr>
</tbody>
</table>
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 run in quadratic time \( O(N^2) \)
  - bubble sort
  - selection sort
  - insertion sort

Algorithm

- Search through the list 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[] list) {
    for (int i = 0; i < list.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < list.length; j++)
            if (list[j] < list[min])
                min = j;
        int temp = list[i];
        list[i] = list[min];
        list[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 a list of \( N \) elements? What is the Big O?

Generic Selection Sort

```java
public void selectionSort(Comparable[] list) {
    Comparable temp;
    for (int i = 0; i < list.length - 1; i++) {
        int min = i;
        for (int j = i + 1; j < list.length; j++)
            if (list[min].compareTo(list[j]) > 0)
                min = j;
        temp = list[i];
        list[i] = list[min];
        list[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[] list) {
    for (int i = 1; i < list.length; i++) {
        int temp = list[i];
        int j = i;
        while (j > 0 && temp < list[j - 1]) {
            // swap elements
            list[j] = list[j - 1];
            list[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