# Searching and Sorting Arrays/Lists

10-10-2001

## Opening Discussion

- What did we talk about last class?
- The delete algorithm we looked at has an average order of O(n). That is to say it is linear. The input size is the length of the list. How many operations it has to perform to do a delete scales as the length of the list.
- Today we will start talking about searching and sorting. We did a search yesterday. How would you sort an array?

## Searching Unsorted Data

- We have seen this before, we looked at the findElement function yesterday that searched a list for an element.
- When data isn't sorted you have to, on average, look at half of the items in a list or array before you find what you want. That implies the search is O(n). The number of things you have to compare to grows proportional to the number of things on the list or in the array.

## Sorting

- There are many reasons to want to sort data. One is that we can typically search sorted data more quickly. We'll talk about that later, today we will look at 3 $O(n^2)$ sorting algorithms.
- The basic process of sorting is easy, line everything up in order so that smaller elements all come before (or after) the larger elements.
- Can you think of methods/algorithms that you might use to sort data?

## Bubble Sort

- In this method you run through the array several times. Each time you look at adjacent elements and switch them if they are in the wrong order.
- You do this n-1 times or until no changes are made (later is a flagged bubble sort).
- It is called a bubble sort because elements slowly shift through like bubbles rising in water.

## Min/Max Sort

- In this method you go through the array and find either the minimum value then move it to the beginning. The next time through you do the same thing but skip over the previous smallest element.
- You can obviously alter this to use maximum or to find both.
- You have to keep track of WHERE the minimum was, not just what it is.

## Insertion Sort

- Insertion sort bears some resemblance to a min/max sort in that you slowly build a sorted section of the array at its front. In this case though you always pick the "next" item in the array to insert into its proper place in the sorted part of the array.
- After putting each item in it moves to the next and figures out where it goes.

## Searching Sorted Data

- When you are dealing with an array, once it is sorted you can search through it more quickly. This is because you know the ordering and arrays allow you "random access". That means you can read any location in constant time.
- In a binary search you narrow down the section you are looking in by a factor of 2 at each step and can find data in O(log n) time.

## Binary Search

- This is best described recursively. You have a "window" that you are looking at. You know the item is between a start and end location. Look at the middle if the window. If it is greater than what you are searching for you cut your window down to the region between start and middle. Otherwise your new window is between middle and end.

## Minute Essay

- Given the following array, trace through what one of the sorts we discussed would do to sort it properly. Sort it from smallest to largest and make sure to indicate what sort you are doing. You only have to show major steps though all swaps would be helpful. A=[7,3,10,2]
- On Friday we will look at more efficient (and more complex) sorting algorithms.