Searching and Sorting

Aaron Friesen
afriesen6@gatech.edu

Searching and Sorting

- Searching
- Why Searching is Hard... but also Cool
- Various Types of Searching
- Sorting
- Who needs to sort?

Searching

The technical definition of searching as it relates to computer science:

Finding an item with specified properties among a collection of items. The items may be stored individually as records in a database, or may be elements of a search space defined by a mathematical formula or procedure (such as all the roots of an equation).

At first, this seems pretty simple, especially given what we know about arrays - you can access them instantly by their indices.

```
    int[] arr = new int[100];
    //populate arr with numbers
    int chosen = arr[50]; //getting numbers out is easy!
```

Why Searching is Hard... but also Cool

Searching nearly omnipresent in the computer science industry. Any time you're dealing with lots of data, you'll probably need to find a way to search through that data.

Imagine you have a list of subscribers to an online service, such as a lottery system. Every five minutes one of them gets a prize. You need to find that person in your system and award them their prize. What happens if you have 1,000,000 users? 2,000,000? 3,000,000?

Searching through that could take a long time, depending on the efficiency of your algorithm. This is why searching is a big deal in computer science. Systems are out there with even more than 3 million subscribers - take Google, for instance. How many web pages does Google index?

Various Types of Searching

Computer scientists tend to categorize searching into different types:

- Linear Search: This is the slow kind. It usually requires iterating through every element of your collection until you find what you are looking for.
- Logarithmic Search: This is the faster kind! Logarithmic searches usually use some method to allow you to ignore parts of your collection you already know ahead of time will not be what you are looking for.
  - There is one catch to the logarithmic search: Usually, your data needs to be sorted ahead of time for them to work.
- Graph Search: Any time you have a graph of data and you want to find a particular element (or node) of that graph. There are a lot of algorithms for this, actually, but I'm going to be focusing on the other two. If you want to look some up yourself for the funsies, Dijkstra's Algorithm and A* are both good places to start.

Linear Search

Again, this is usually the worst kind of search. You've already needed to do it so this should be relatively straightforward.

```
    //given an int array arr
    public int search(int element) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == element) {
                return arr[i];
            }
        }
        return -1;//or some other value indicating that the element
    was not found
    }
```

This should seem relatively silly (you ask it to search and it returns exactly what you told it to look for). Similar logic could be used on a `contains(int element)`, for example.

Completely Unrelated Game Interlude

Game time.

Completely Unrelated Game Interlude

Game time.
What was the optimal way to play this guessing game? Why?

Game Interlude - Secretly Binary Search

Game time.
What was the optimal way to play this guessing game? Why?

- Able to eliminate a large number of possible choices every time you guess
- Able to guess from a large data set without looking at every single possible element

What we've just done is what's called Binary Search. It's called Binary Search because every guess allows you to eliminate HALF of the remaining choices.

Game Interlude - Secretly Binary Search

Game time.
What was the optimal way to play this guessing game? Why?

- Able to eliminate a large number of possible choices every time you guess
- Able to guess from a large data set without looking at every single possible element

What we've just done is what's called Binary Search. It's called Binary Search because every guess allows you to eliminate HALF of the remaining choices. Which apparently means Binary?

Hashing

Logarithmic searching is nice and all, but couldn't it be *even faster?* This is why hashing is basically the coolest thing ever.

Remember the definition of `Object.hashcode()`: it returns an int which corresponds to the object `hashcode()` is called upon. The same object should return the same number until it is modified somehow, and objects that `.equals()` considers equal should return the same value.
Let's do a quick delve into how things like `HashSet`s work to see why this is cool.

Hashing 2: Electric Boogaloo

Adding to a `HashSet` is pretty simple: You have a backing array of some generic type `T`, and the index at which an object goes roughly corresponds to its `hashcode()`. (Not exactly, but close enough.)

```
    public void add(T element) {
        int index = element.hashcode(); //ask the element to give a number ID.
    Think of this like the license plate for that specific object.
        arr[index] = element;
    }
```

Now that we've added an element, checking to see if the `HashSet` contains that element is easy. Why? <u>Because we already know where it should go!</u>

```
    public boolean contains(T element) {
        int index = element.hashcode();
        return arr[index] != null; //as long as something is in that spot,
    then we know that it's in the array! No looping necessary.
    }
```

Because we only need to check that single spot, it takes the same amount of time when we store 5 objects as it does when we store 5,000,000. Which is pretty rad if you ask me.

Sorting

Things like Binary Search can only be done to data that has already been sorted. Sometimes it's just nice to have sorted data to make everything look pretty and nice. In either case, sorting is just as (if not more) important to understand. There are hundreds of different sorting algorithms, but I'm going to be talking about a few popular ones, from worst to best:

- `BubbleSort`
- `SelectionSort`
- `InsertionSort`
- `BinarySort`
- `QuickSort`

Actually probably not that many. That's a lot of algorithms. You should look most of these up on your own though, as they are pretty interesting and Wikipedia has some kickass psuedocode and visualizations.

BubbleSort and Why You Should Already Be Cringing

The basic idea behind BubbleSort is simple: iterate through your collection, and if you ever find two side-by-side elements in the wrong order, reverse them. Repeat until you don't reverse anything.

Let's do it!

SelectionSort

SelectionSort loops through the array until it finds the smallest element, then puts that element into the beginning of the array. It loops again, finding the second smallest element, and put that in the second spot. It repeats this process until the list is sorted.

Let's do this one too.

InsertionSort

InsertionSort loops through the array, putting each element into a different temporary array in the spot that it should go. This is a little bit harder than SelectionSort, but is much faster.

You guys should know the drill by now.

Who needs to sort?

EVERYONE.