Lab 3

Steve Boucher

November 5, 2022

Finders Keepers!

1 Introduction

The goal of lab 3 was to write search algorithms. The three search functions required to code for this lab were linear, binary, and hash lookup. For my initial setup I created a main.java program, My CWN (Completed, WorkInProgress, And Notes) txt file, along with the magic items text file. I also brought in some previous code from lab two, merge sort being a must, as the lists had to be sorted to be searched, as well as my File Reader function which I improved upon in this lab.

2 Reused Code

I was quite proud of my optimizations to my file reader program. In my previous labs, I had quite a large and clunkly main program, as it contained all the code for the file reader as well as The rest of the program calls I needed to make. In this lab, I was able to take that code, and condense it into its own function to be called on once, and easily passed to my other commands. Below is the slimmed down main function, which is no only 5 lines of code.

```
7  public static void main(String[] args) throws FileNotFoundException {
8    String[] magicalArray = FileReader();
9    LinearSearch(magicalArray);
10    BinarySearch(magicalArray);
11   hashSearch(hashyTime(magicalArray), magicalArray);
12  }
```

My merge sort command remained untouched, so that was the lowest function I put into my main file and I didn't need to see it constantly. I did reuse 2 pieces of code consistently between linear and binary search which were amount of items required to find, and comparison calculators. I was required to search for 42 items at random, and to do that I put my search function in a simple 4 loop for 42 times, and added the comparisons, and calculated the average from there.

2.1 Linear Search

Coding linear search was quite simple. The process the codes through can be boiled down to 2 simple steps. 1. Check if the current item is what we're looking for. 2. If we found it, Hooray! If not, Go to the next spot and repeat until either it's found out we run out of list. The Big O for linear search is O(N), as the worst case scenario requires the program to look at every item in the data set. Horribly inefficient, but can be easy and time effective for smaller sets of data.

```
26      while (found != true) {
27
28        if (magicItem == magicalArray[counter]) {
29          found = true;
30          total = total + counter;
31        }
32        counter++;
```

2.1.1 Binary Search

I actually had a lot of fun coding binary search. Binary search requires 3 pointers to lob of as much excess data as possible to require a smaller amount of total comparisons. Binary search does that my have a left pointer (lp), right pointer (rp), and mid point (mp). as it searches the data set, it compares the item to the midpoint. if its higher, it adjusts lp or rp accordingly, and then recalculates the mp until it finds what its looking for. This process cuts down on time and brings its big Oh to O(log N).

```
53      while (found == false) {
54        int mp = lp + (rp - lp) / 2;
55        counter++;
56        if (magicalArray[mp] == item) {
57          found = true;
58        } else if ((item.compareToIgnoreCase(magicalArray[mp])) < 0) {
59          rp = mp - 1;
60        } else {
61          lp = mp + 1;
62        }
63        total = total + counter;
64      }
```

2.1.2 Linked List Plus Hash

After my horrible linked list flop of lab one, I am happy to say I redeemed my self, and was able to put together a linked list! a Linked List is a type of list where each item contains data and a pointer to the next

```
1    public class Node {
2       int data;
3        Node next;
4    }
```

now, some extra code had to be written to fill in the list with items, which I did in my linked List file, creating a insert command, which creates a node, fills it, and a pointer to the next.

After creating the linked list, the next step was to create a hash table using the code provided on the classroom website. I struggled to figure out how to correct implement this code for a while, but I was able to correctly implement this. My big issue came down to how to search the items once they are hashed. In my head, if I'm searching for something, I put it in the hash function, it tells me where to go. It doesn't feel like that's the correct way to do it, but the resources I've found online say to put it through a key, and besides just hashing that, I don't know what else they would mean. The Average complexity for a hash search is the same as O(log N) as the goal is to not look at as much data as possible.

3 Results and Conclusion

I feel like I did a really good job on Binary and Linear Searching. I thoroughly enjoyed the challenged they both provided, and seeing the results. The average for searching for 42 items was around 325 comparisons for linear search and 42 comparisons for Binary Search. Sadly I was unable to correctly calculate the average for Hash Search, but the total for 42 should be around 7 comparisons.