A Comparison of Sequential and Binary Search
CS 317
Nathan Solomon

Introduction:

In this report we will look at the differences between Sequential and Binary search and demonstrate how their theoretical performance matches their experimental performance.

Sequential search is a search algorithm that attempts to find an element by starting at the beginning of the list and reading each element sequentially until it finds the element that it is looking for. This algorithm has an average time complexity of $Cavg(n) = (n + 1)/2 \in \Theta(n)$

Binary search is a search algorithm that starts by making an assumption that the list is sorted. Since it is sorted, the algorithm can start at the middle and make a comparison to determine if the element we are searching for is less than, equal to, or greater than the middle element. By doing this, if the element was not found, the algorithm can cut out about half of the possible elements in the list. The algorithm will then complete this action again with the remaining elements until the element is found. This algorithm has an average time complexity of $\Theta(\log_2 n)$

Experiment Setup:

An experiment was performed where each algorithm was run to find every single element in a random list of numbers. The average number of comparisons needed in order to find the element was then recorded. This experiment was performed on lists of 500 different lengths.

The project was set up and run using Java jdk1.8.0_231 and was built in IntelliJ IDEA 2019.3 (EDU). The following implementation of each algorithm was used to perform the search (Note: in both examples, `SearchResults result` is used to keep track of the number of comparisons throughout the experiment).

```
public static int search(int[] list, int value, SearchResult results){
   for(int i = 0; i < list.length; i++){
      results.addSequential();
      if(list[i] == value){
         return i;
      }
   }
   return -1;
}
```

Figure 1: Sequential Search Algorithm

```
public static int search(int[] list, int value, SearchResult results){
   return search(list, value, results, 0, list.length-1);
}

private static int search(int[] list, int value, SearchResult results, int left, int right){
   int middleIndex = ((right - left) / 2) + left;
   int middleValue = list[middleIndex];
   results.addBinary();
   if(middleValue == value){
      return middleIndex;
   } else if(value < middleValue) {
      return search(list, value, results, left, middleIndex-1);
   } else {
      return search(list, value, results, middleIndex + 1, right);
   }
}
```

Figure 2: Binary Search Algorithm

Data:

The complete raw data of the experiment can be found here: https://docs.google.com/spreadsheets/d/1oWzjkU6FNqrZ1viGVOZTJr0BUL3knLcB5cuK2PQgOFs/edit?usp=sharing

Selected Data Points:

List Length	Binary Search Comparisons	Theoretical Binary Search Comparisons	Sequential Search Comparisons	Theoretical Sequential Search Comparisons
1	1	0 ($\log_2 1 = 0$)	1	1
50	4.86	5.6439	25.5	25.5
100	5.8	6.6439	50.5	50.5
150	6.3423	7.2288	75.5	75.5
200	6.765	7.6439	100.5	100.5
250	7.012	7.9659	125.5	125.5
300	7.3267	8.2288	150.5	150.5
350	7.5657	8.4512	175.5	175.5
400	7.745	8.6439	200.5	200.5
450	7.8778	8.8138	225.4978	225.5
500	7.994	8.9658	250.4980	250.5

Table 1

Analysis:

Overall, much of the data is very close to what we expected to see, however there were a few differences between the theoretical and experimental data.

First, when looking at the data for sequential search, the experimental data is exactly the same as the theoretical average until right before the very end. After further investigation, this difference occurred due to duplicate values in the lists that were being searched. If there are duplicate values in the list, it was possible that a certain value could be found earlier in the search since it would be in the list multiple times. That accounts for the small difference in the sequential search.

Second, throughout my data for binary search, there is a small difference between the theoretical values and the experimental values. There are several factors that I think may have contributed to this. First is the same factor mentioned above regarding duplicate values in the list. The second factor is that our calculation for the average case for Binary Search assumes that the size of the remaining values gets cut exactly in half every single time. In our implementation (Figure 2), we

actually counted the number of three way comparisons, which meant that in every comparison, we could cut out slightly more than half the elements.

Despite these discrepancies, if we plot both our experimental and theoretical data on a graph, it reveals several important facts:

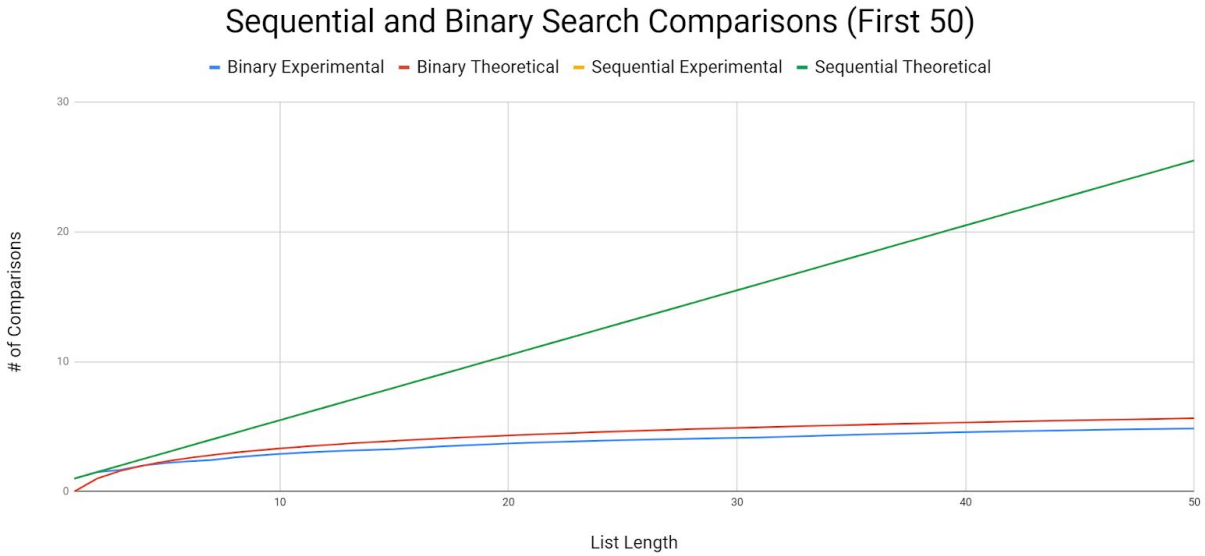

Figure 3

In the first 50 elements of the data (Figure 3), it becomes apparent that the experimental data sets have the same order as the theoretical data. Additionally, as we include large amounts of data (Figure 4), it becomes more apparent that this relationship is true.

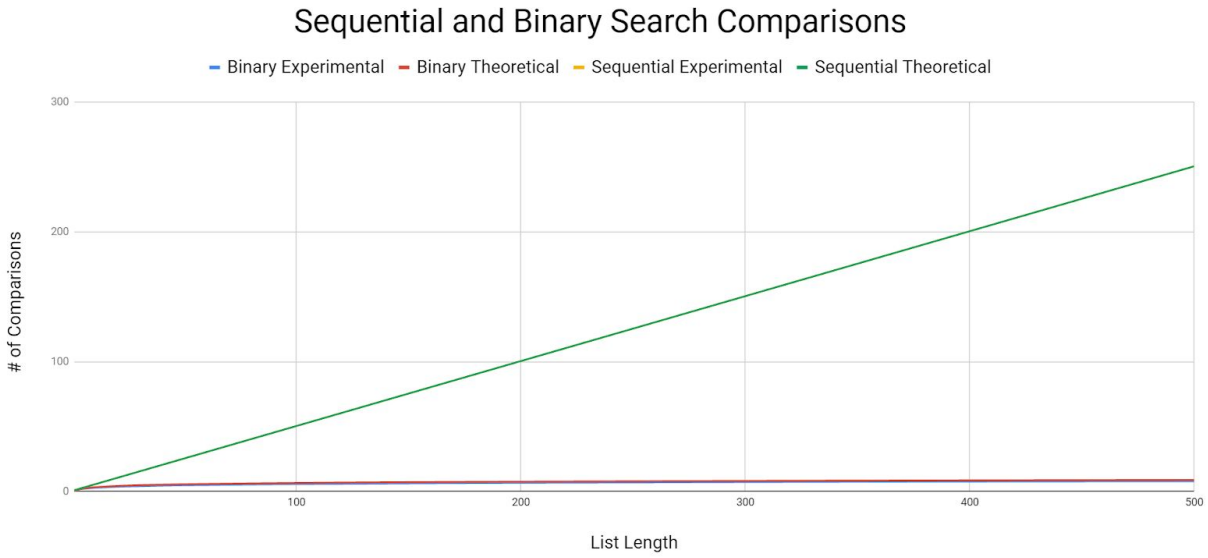

Figure 4

Conclusions:

In conclusion, Binary Search is capable of finding elements in a list much faster than a sequential search, assuming that the list is already sorted.

There are several different areas of further research that may be interesting to investigate. While lists of 500 different lengths were sufficient to demonstrate the differences between these two algorithms, it may be interesting to try much larger data sets (on the order of thousands for millions of different lengths). This could also help us to answer some of the questions pertaining to the differences between our experimental and theoretical data. Additionally, since it was assumed that the list was already sorted, it may be interesting to compare how these algorithms perform on an unsorted list. While Sequential search would be able to continue it's execution unchanged, the binary search would need to be modified to include a sorting algorithm before it could be run.