Ryan Swift
16 January 2023

Time Measurement Analysis for Project 1

This analysis compares the runtimes of the linear, non-recursive binary, and recursive binary search methods. All recorded times are in microseconds, and represent the averages across three executions each (for a total of 9 executions across the three search types), with all executions within the same Trial using the same input data files. In this analysis, the input size of the requested books is always held constant at 2 books, and the input size of the list of new books varies across the three trials: Trial 1 searches through 10 books, Trial 2 searches through 1000 books, and Trial 3 searches through 100,000 books.

Trial 1: 10 New Books

Search Method	Linear	Non-Recursive Binary	Recursive Binary
Time	4.513	83.229	76.165

Trial 2: 1000 New Books

Search Method	Linear	Non-Recursive Binary	Recursive Binary
Time	101.623	10633.767	11338.733

Trial 3: 100,000 New Books

Search Method	Linear	Non-Recursive Binary	Recursive Binary
Time	7218.977	1,505,203.333	1,495,333.333

Ryan Swift
16 January 2023

Conclusion:

When I first saw these data, I was almost certain that I had done something incorrectly. After all, how could it possibly be the case that the performance of a linear search was *multiple orders of magnitude* superior to that of a binary search? After looking though my source code for any sign of something that might cause my code to behave so strangely, I realized that the huge performance disadvantage for the two binary search algorithms was due to the need to sort the list of books before they could be input into the algorithms. When designing my program, I made the decision to include the execution of the sort function in the elapsed time for both of the binary search functions. It seemed unfair to me to exclude the sorting time from the binary searches, as the linear search does not require sorting to be able to execute properly, whereas the binary searches will not work without the book list being sorted.

As these data demonstrate, if the list of new books provided is unsorted, it is much faster to avoid sorting and simply use a linear search. Though, if the list of new books is meant to be used in a context outside of this assignment- say, in an actual library- that time spent sorting would only need to occur when the list is initialized. After that, the efficiency of the two binary searches would undoubtedly reveal them as superior to linear searches on large lists. This is made evident in an analysis of the runtimes of linear and binary searches. Assuming the variable **m** represents the number of requested books in our search, and **n** represents the number of new books, then the runtime of the linear search is **m(n)**, as the full list of **n** new books will be traversed for each of the **m** books requested (assuming worst case scenario. As for our binary search, the search itself is **m(log n)**, as the size of the list being traversed (**n**) is halved at each iteration, and run **m** times. However, binary searches also require that the list be sorted prior to the initiation of the search algorithm. As such, the sorting runtime adds **n(log n)** to the runtime, and it is this extra sorting time included in the binary search functions which produces the longer runtimes, relative to linear search. Therefore, if a given list is short or unsorted, a linear search will be the most efficient method. Otherwise, if the list is long and has already been sorted, then the binary searches become the optimal search method.