****

**PA4 Report**

By

**Andrew Vinci**

**CSI 281**

**Data Structures & Algorithms**

**Spring 2021-22**

1. **Introduction**

For this experiment, I am testing two different searching algorithms, Sequential Search and Binary Search. The goal of this experiment is to figure out which Searching Algorithm is faster under different circumstances. Each algorithm will be tested with a search key of Low, Middle, and High. With Low being a search key that appears in the beginning of the list, Middle being a search key that appears in the middle of the list, and High being a search key that appears at the end of the list. Each Algorithm will be testing the same list of numbers. Sequential Search will be using a version that requires the list to be sorted before the search takes place, while Binary Search will be using a version that is recursive, meaning that it calls itself during the run time of the algorithm. Both lists will be sorted with Quick Sort before they are implemented. The hypothesis for this experiment is that Binary Search will outperform Sequential Search. Using Big O notation, which is a mathematical formula used to determine the run time of a given algorithm [1], my hypothesis is that Binary Search will outperform Sequential Search with each search key placement, given that the average run time of Sequential Search is O (n) and the average run time of Binary Search is O (log n).

1. **Background**

There are two different Searching Algorithms that are going to implemented in this experiment. The first is Sequential Search and the second is Binary Search.

* **Sequential Search:** A few characteristics of Sequential Search are that it is one of the easier searching algorithms to implement [2]. Another characteristic of Sequential Search is that it does not need the list that it is performing a search on to be sorted [2]. However, for this lab the Sequential Search that will be used is one that does require the list to be sorted beforehand. Both characteristics can be considered advantageous, as the algorithm is easy to implement and does not necessarily require the list to be sorted before implementing. However, one disadvantage of Sequential Search is that as the list grows, the run time depends on the number of items in the list [3]. This leads into the performance analysis of Sequential Search. The average run time of Sequential Search is O (n), with a best-case of O (1) and a worst-case of O (n).
* **Binary Search:** A few characteristics of Binary Search are that the data must be sorted before this search can be implemented [4]. Another characteristic of Binary Search is that with every search it does, the number of items are cut in half each time [4]. This leads into one of the benefits of Binary Search. Since it cuts the numbers its searching through in half each time, it is often useful when dealing with larger lists [4]. A disadvantage of this searching algorithm is the fact that it cannot be used on a random list of numbers. It can only be used on a list that has already been sorted. The average run time of Binary Search is O (log n), with a best-case of O (1), and a worst-case of O (log n).

1. **Implementation Detail**

**Sequential Search:**

* Sequential Search is a searching algorithm that searches through a list of items and compares every item in the list to a pre-determined search key. The algorithm goes through the entire list until it either finds the search key or until it reaches the end of the list. There are different versions of Sequential Search. One version, which is not used in this lab, is a search that can be implemented on a random list of numbers and does not require the list to be sorted beforehand. This is one of the benefits of Sequential Search as it can be implemented on any list to find a pre-determined search key. Another version, which will be used in this lab, searches through the entire list for the search key and requires that the list be sorted before implementing. This version also tests whether or not the item in the list is greater than the search key. Since the list is sorted for this version, if that item in the list turns out to be greater than the search key, then the search ends because the item would not be in that specified list.

**Binary Search:**

* Binary Search is a searching algorithm that searches through a list of items, it finds the middle element of that list, and cuts the list in half each time that the search takes place, depending on whether or not the indexed spot is greater than or less than the search key. Since Binary Search requires that the list be sorted beforehand, the search can determine whether or not the search key is greater than or less than the indexed spot in the list that the search key is being compared to. If the search key is greater than the indexed spot in the list, then it ignores the items that are all smaller than the indexed spot and concentrates its search on the items that are greater than the indexed spot. Likewise, if the search key is less than the indexed spot in the list, it ignores the items that are greater than the indexed spot and concentrates its search on the items that are smaller than the indexed spot. The version of Binary Search that is used in this lab is a recursive version of Binary Search, meaning that the function calls itself within its implementation. If the indexed spot is greater than the search key, then the upper bound and lower bound are altered so that the search takes place only in the lower set of items in the list and the function then calls itself. If the indexed spot is less than the search key, then the upper bound and lower bound are altered so that the search takes place only in the higher set of items in the list and the function then calls itself. It does this until the entire list has been searched through or until the search key has been found.

1. **Experimentation Detail**

The platform that was used to conduct this experiment is a Dell Inspiron 15 7000 series laptop that has an Intel Core i7-10510U with a CPU speed of 1.80GHz 2.30GHz. The amount of memory on board is 16gb and the operating system is a 64-bit system with a x64-based processor.

**Summary Data**

**Sequential Search:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: Sequential Search** | | | |
| **N** | **Dataset #1 – Search Key Low** | **Dataset #2 – Search Key Mid** | **Dataset #3 – Search Key High** |
| 100 | 0.0001741s | 0.0001538s | 0.0001459s |
| 100,000 | 0.0005696s | 0.0005691s | 0.0006685s |
| 1,000,000 | 0.0049169s | 0.0049856s | 0.0050683s |

**Binary Search:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm: Binary Search** | | | |
| **N** | **Dataset #1 – Search Key Low** | **Dataset #2 – Search Key Mid** | **Dataset #3 – Search Key High** |
| 100 | 0.0003534s | 0.0001460s | 0.0001377s |
| 100,000 | 0.0002058s | 0.0002005s | 0.0002156s |
| 1,000,000 | 0.0002152s | 0.0001965s | 0.0002481s |

1. **Discussion and Conclusion**

The results of this experiment fell in line with my original hypothesis, which was that Binary Search would outperform Sequential Search. For the most part, Binary Search did outperform Sequential Search within the experiment. The size of the list made the most difference when it came to Sequential Search. This experiment proved that as the size of the list grows, so does the time that it takes for Sequential Search to complete its initial search. One thing that is important to note is that with Binary Search, the size of the list does not seem to have had a big impact. In fact, the average times from 100,000 items and 1,000,000 items are not that different when it comes to Binary Search. Sequential Search performed the best when it was dealing with search keys that appeared in the beginning of the list. With Binary Search, it performed the best when search keys were at the middle of the list.

Overall, the experiment did produce the results that I expected going into it. Sequential Search was slower than Binary Search and as the number of items in the list grew, so did the time it took for Sequential Search to complete its search. Alternatively, Binary Search did not have a major time difference when it came to the size of the list. The time differences with Binary Search were more affected when it came to where the search key was within the list, although this did affect Sequential Search as well.

If I had to rank these two searching algorithms from best to worst, I would rank them as follows:

1. Binary Search
2. Sequential Search

If I had to choose an algorithm that I preferred, it would have to be Binary Search. The fact that Binary Search produced fairly similar results all the way through the experiment, no matter what the size of the list was, lends me to believe that it would be a preferred choice for myself. However, it should be considered that Binary Search needs to have a sorted list before it can even be performed, so there are times when Sequential Search might be the preferred method. With that being said, pairing Binary Search with a fast-sorting algorithm, such as Quick Sort, would be my preferred method of sorting and searching.

1. **References**

[1] E. Team, “What is big-O notation?,” *Educative*, 2021. [Online]. Available: https://www.educative.io/edpresso/what-is-big-o-notation. [Accessed: 05-Mar-2022].

[2] “Linear Search,” *GeeksforGeeks*, 15-Sep-2021. [Online]. Available: https://www.geeksforgeeks.org/linear-search/?ref=gcse. [Accessed: 05-Mar-2022].

[3] “Introduction to searching algorithms,” *Studytonight.com*, 2021. [Online]. Available: https://www.studytonight.com/data-structures/search-algorithms. [Accessed: 05-Mar-2022].

[4] “Comparison of searches - searching - KS3 computer science revision - BBC Bitesize,” *BBC News*, 2021. [Online]. Available: https://www.bbc.co.uk/bitesize/guides/zgr2mp3/revision/4#:~:text=One%20of%20the%20main%20advantages%20of%20a%20binary%20search%20is,within%2010%20steps%2C%20every%20time. [Accessed: 05-Mar-2022].

1. **Appendix**

**Sequential Search:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Sequential Search** | | | **Dataset #: Search Key Low** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0.0001992s | 0.0001367s | 0.0001864s | 0.0001741s |
| 100,000 | 0.0006522s | 0.0004856s | 0.0005711s | 0.0005696s |
| 1,000,000 | 0.0053353s | 0.0032077s | 0.0062079s | 0.0049169s |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Sequential Search** | | | **Dataset #: Search Key Mid** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0.0001468s | 0.0001536s | 0.0001612s | 0.0001538s |
| 100,000 | 0.0004375s | 0.0004924s | 0.0007775s | 0.0005691s |
| 1,000,000 | 0.0030745s | 0.0056898s | 0.0061926s | 0.0049856s |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Sequential Search** | | | **Dataset #: Search Key High** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0.0001778s | 0.0001401s | 0.0001198s | 0.0001459s |
| 100,000 | 0.0006901s | 0.0004993s | 0.0008163s | 0.0006685s |
| 1,000,000 | 0.002831s | 0.0062296s | 0.0061444s | 0.0050683s |

**Binary Search:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Binary Search** | | | **Dataset #: Search Key Low** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0.0007885s | 0.0001457s | 0.0001261s | 0.0003534s |
| 100,000 | 0.0002045s | 0.0001900s | 0.0002231s | 0.0002058s |
| 1,000,000 | 0.0002293s | 0.0002257s | 0.0001907s | 0.0002152s |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Binary Search** | | | **Dataset #: Search Key Mid** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0.0001410s | 0.0001442s | 0.0001530s | 0.0001460s |
| 100,000 | 0.0002146s | 0.0001828s | 0.0002041s | 0.0002005s |
| 1,000,000 | 0.0001946s | 0.0002296s | 0.0001653s | 0.0001965s |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm: Binary Search** | | | **Dataset #: Search Key High** | |
| **N** | **Run #1** | **Run #2** | **Run #3** | **Average** |
| 100 | 0.0001259s | 0.0001513s | 0.0001361s | 0.0001377s |
| 100,000 | 0.0001962s | 0.0002326s | 0.0002180s | 0.0002156s |
| 1,000,000 | 0.0002500s | 0.0002129s | 0.0002814s | 0.0002481s |