Searching Algorithm Efficiency

Alex G, Ethan N, Victor O

CS404
Submitted: 4/26/25

Table of Contents

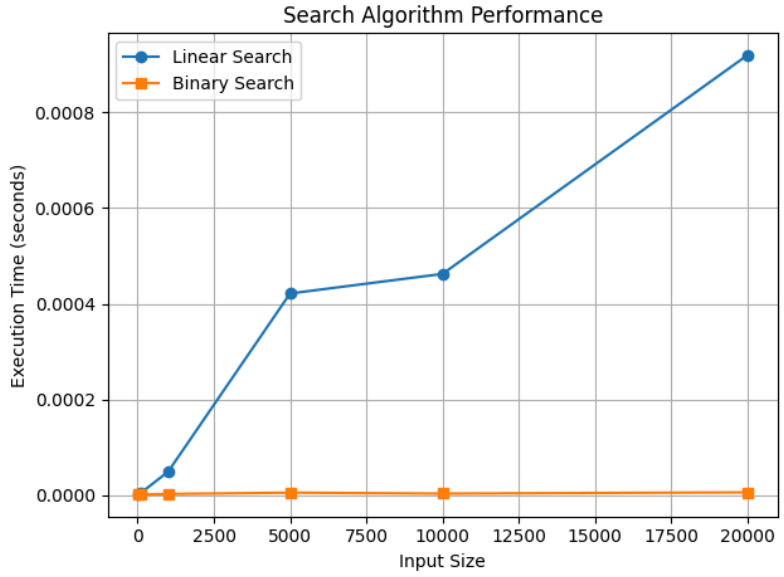
Abstract

Searching algorithms provides a systematic, efficient method to help users locate data and information within a database (*Algorithms for Efficient File Searching: Mastering the art of Quick Data Retrieval*). This paper explores some of the most common algorithms and compares the time efficiency of them. Measuring time efficiency identifies the algorithms that fit the needs of the user where n is the number of items in a dataset (*Algorithms for Efficient File Searching: Mastering the Art of Quick Data Retrieval*). The worst and average case of linear search is O(n), meaning that there is a proportional relationship between the database's size and time it takes to search for it due to how it searches from the first element to the last (*Linear Search vs Binary Search,* 2023). On the other hand, the worst and average case of binary search is O(log n), indicating that there is a logarithmic relationship due to how it recursively returns the midpoint of the dataset and subdatasets until it finds the targeted element (*Linear Search vs Binary Search,* 2023). For both of these algorithms, the best case is O(1) when the element searched for is the first element in linear search or the element whose index is the floor of the quotient of the first and last index. For sorting algorithms that organize data, insertion sort has a quadratic time complexity, merge sort has a log-linear time complexity, and radix sort has a linear time complexity. This paper will analyze these algorithms in a real-world environment and discuss the implications of the results. Understanding the strengths of the searching algorithms leads to more efficient solutions to current challenges.

Methodology

While many searching algorithms exist, linear and binary search algorithms are known well to be efficient algorithms. Using Visual Studios, both of these algorithms were programmed in Python. Implementing these in a consistent environment reduces the impact from the hardware and network. Additionally, the data sets used were measured from the same CPU and sorted, so binary search could be implemented without bugs. A series of input sizes were used from 10 to 1 million to observe how much impact size has on time efficiency. To get the best understanding of the time complexity, the last element was the target element (worst case). After recording the times, they were analyzed and graphed (Figure 1) to see how much impact input size affects the search algorithm's time to find the target element in the worst case and any practical implications of any differences.

Figure 1



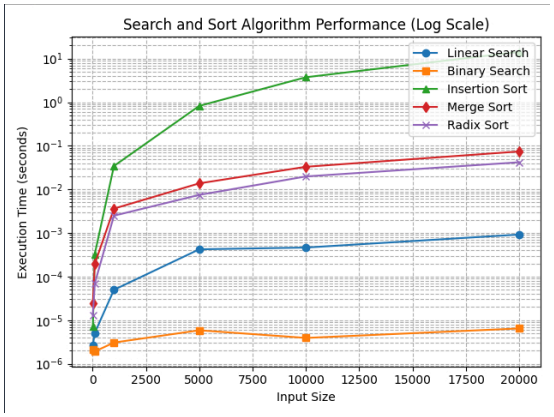


Figure 2

Results and Analysis

As each target element deviated further from 0, linear search became less efficient in searching the number, whereas binary search remained relatively constant. While the algorithm's methods may not be as impactful for smaller datasets as both algorithms performed similarly for input sizes 10 and 100 in Figure 3, the differences become larger when the input size increases. This increasing difference can be pointed to how each algorithm searches for a number. Linear search algorithms start from the first index to the last, so if the target element was always the last element, the time it takes to search for each input size would proportionally increase as shown in Figure 1. On the other hand, binary search algorithms repeatedly divide the search space to find the element, so order never matters and thus give out near constant results in Figure 1. In sorting algorithms, Radix Sort is the fastest, Merge Sort is the second fastest, and Insertion Sort is the slowest as input size increases. Similar to the searching algorithms, when the input size is small, the sorting algorithm method matters little, but the difference increases as input size increases. These findings remain consistent with the time complexity of each algorithm.

Figure 3

```
Analyzing search and sort algorithm performance...

Input Size |   Linear (s) |   Binary (s) |   Insertion (s) |    Merge (s) |    Radix (s)
------------------------------------------------------------------------------
        10 |   0.00000281 |   0.00000273 |      0.00000948 |   0.00002422 |   0.00002232
       100 |   0.00000719 |   0.00000269 |      0.00041083 |   0.00019124 |   0.00010040
      1000 |   0.00004434 |   0.00000294 |      0.08256522 |   0.00428464 |   0.00233272
      5000 |   0.00022104 |   0.00000515 |      1.92360680 |   0.06823886 |   0.01211916
     10000 |   0.00107449 |   0.00000809 |      8.08981261 |   0.08394549 |   0.07689826
     20000 |   0.00129382 |   0.00000829 |     17.02912430 |   0.09438843 |   0.05237424

Time Complexity Discussion:
- Linear Search: O(n)
- Binary Search: O(log n)
- Insertion Sort: O(n^2) worst case, O(n) best case
- Merge Sort: O(n log n) in all cases
- Radix Sort: O(nk), where k is number of digits

Space Complexity Discussion:
- Linear Search: O(1)
- Binary Search: O(1)
- Insertion Sort: O(1)
- Merge Sort: O(n) (needs temporary arrays)
- Radix Sort: O(n + k) (needs counting/output arrays)
```

Conclusion

This paper has demonstrated the differences in time complexity of linear and binary search algorithms in terms of increasing input size. It should also be acknowledged the choice of algorithms may not always depend on speed but also be guided with the nature of the dataset. Nevertheless, while linear search performs at a similar level as binary search for smaller datasets, the logarithmic time complexity of binary search becomes more advantageous than the linear time complexity of linear search as binary search consistently performs significantly faster at larger datasets if they are sorted. In terms of sorting algorithms, while the sorting method may play little role in smaller datasets, when input size increases, radix sort experiences a linear time complexity (fastest), merge sort has a log-linear time complexity (second fastest), and insertion sort has a quadratic time complexity (slowest). By comparing and understanding the difference between sorting algorithms, it becomes easier to decide on which to use for the most optimal solution.

References

Algorithms for Efficient File Searching: Mastering the Art of Quick Data Retrieval. AlgoCademy Blog. (n.d.). https://algocademy.com/blog/algorithms-for-efficient-file-searching-mastering-the-art-of-quick-data-retrieval/

GeeksforGeeks. (2023, December 19). *Linear Search vs Binary Search*. GeeksforGeeks. https://www.geeksforgeeks.org/linear-search-vs-binary-search/