BIRZEIT UNIVERSITY

Faculty of Engineering and Technology
Department of Electrical and Computer Engineering
ENCS5141—Intelligent Systems Laboratory

Assignment #1—Comparing the Time Complexity of Binary Search and Linear Search algorithms

Prepared by: Ahmad Ali—1102020

Instructor: Ali Ahmad
Assistant: Omar Ali
Date: March 9, 2024

Abstract

The aim of this study is to investigate the performance of binary and linear search algorithms. The method employed involves measuring the execution time of each algorithm across four sorted arrays with varying sizes. The results reveal that as the array size increases, binary search demonstrates significantly faster performance compared to linear search. However, for very small arrays, specifically those with less than 10 elements, linear search outperforms binary search. These findings underscore the importance of considering array size when selecting search algorithms.

Contents

1 Introduction

Time complexity measures the computational time an algorithm needs relative to input size. It's crucial for algorithm efficiency, aiding in selection and optimization for different computational tasks.

Two prominent searching algorithms, Binary Search and Linear Search, represent distinct approaches to locating elements within an array. Linear Search iteratively scans the array until it finds the target element. Its simplicity and versatility make it suitable for unsorted arrays. However, its time complexity grows linearly with the size of the array, rendering it less efficient for large arrays. In Listing 1.1, it's shown that the worst case of the algorithm is when the element isn't found. In such case, the algorithm would iterate over the full array. Which leads to a time complexity equal to $\mathcal{O}(n)$.

```
std::vector<int>::const_iterator linear_search(const
↪  std::vector<int>& array, const int target) {
        auto it = array.begin();
        while (it != array.end() && *it != target) ++it;
        return it;
}
```

Listing 1.1: Linear Search algorithm.

Binary Search, by contrast, operates on sorted arrays by repeatedly dividing the search space in half. This "divide and conquer" strategy results in a logarithmic time complexity. Despite the requirement for a sorted array, Binary Search offers superior performance for large arrays compared to Linear Search. In Listing 1.2, the algorithm splits the array into halves, then selects one of them based on the middle element to continue the search on. In the worst scenario, the algorithm has to perform $\log_2 n$ splits before being left of with an array with only one element. Thus, the time complexity of the binary search algorithm is $\mathcal{O}(\log n)$.

Figure 1.1 shows the theoretical complexity plot for linear and binary search. It shows that binary search algorithm would perform better than linear search in all cases.

```
std::vector<int>::const_iterator binary_search(const
↪  std::vector<int>& array, const int target) {
        auto left = array.begin();
        auto right = array.end() - 1;

        while (left <= right) {
                auto mid = left + (right - left) / 2;
                if (*mid == target) return mid;
                else if (*mid < target) left = mid + 1;
                else right = mid - 1;
        }
        return array.end();
}
```

Listing 1.2: Binary Search algorithm.

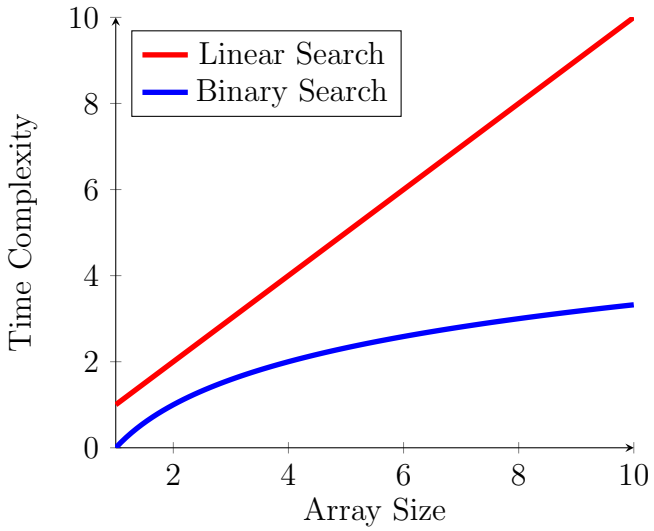

Figure 1.1: Time complexity plot for both algorithms.

The objective of this experiment is to compare the complexity of both algorithms and verify that the actual execution time of each algorithm matches the theoretical results stated in figure 1.1.

2 Procedure and Discussion

The study starts by preparing the data, which consists of four sorted arrays of sizes. 10, 100, 1000, and 10000. For each algorithm, the execution time of searching for a randomly sampled elements is measured.

Each experiment is formed using one of the algorithms on one of the arrays. Each experiment is repeated 100 times to measure the minimum, maximum, and average execution time.

It is worth mentioning that the execution time is measured around the search itself; to avoid any overhead from data preparation and measurement processing (finding the minimum, maximum, and average). Listing 2.1 shows how the execution time is being measured in C++.

```
auto start = std::chrono::high_resolution_clock::now();

search(array, element);

auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
```

Listing 2.1: Measuring the execution time of the search algorithm.

The results shows that an increase in the array size resulted in a proportional increase in the average execution time for the linear search. However, for binary search, the average execution time barely increased with the increase of the array size. Figure 2.1 shows that the linear search has increased almost linearly with the array size. While the binary search shows a barely noticed increase in the execution time.

The results in Table 2.1 show that linear search might be more effective in the smaller arrays (less than 10 elements).

Table 2.1: The minimum, maximum, and average execution time for each algorithm at specific array size. All measurements are in nanoseconds.

Array Size	**Linear Search**			**Binary Search**		
	Min.	**Max.**	**Avg.**	**Min.**	**Max.**	**Avg.**
10	54	559	121.96	65	327	129.31
100	110	948	505.52	98	420	225.21
1000	201	8349	4213.82	125	480	330.12
10000	84	92643	23783.3	65	644	233.5

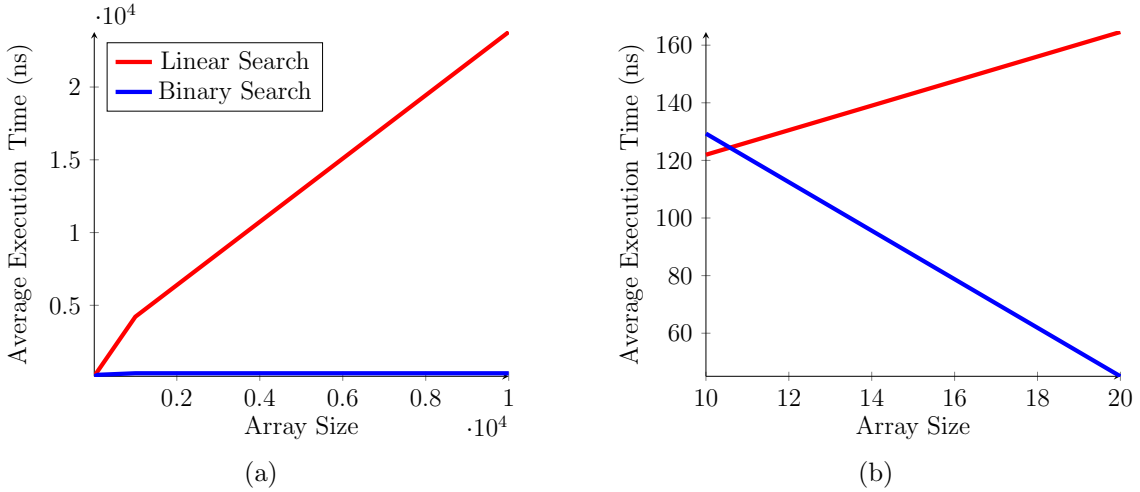

Figure 2.1: Average execution time (in nanoseconds) per array size plot for both algorithms.

Generally, linear search performs badly on its worst case compared to binary search when the array size is large. For example, the maximum execution time for linear search on array of size 10000 is 92643 ns compared to 644 ns for binary search.

Figure 2.1 shows some non-linearity in the linear search curve. However, this is normal due to the small number of samples (array sizes) that have been considered, so a noisy point can have a major effect on the rate of change for that curve.

3 Conclusion

In this assignment, the performance of binary and linear search have been studied. Binary search proved to be significantly faster on larger arrays, with more stable complexity curve. This is due to the fact that binary search has a time complexity of logarithmic order, while the linear search has a time complexity of linear order.

However, it's important to note that in smaller arrays, where the difference between logarithmic and linear order can be negligible, the linear search proved to be faster. A possible reason for this could be the overhead of calculating the middle element in the binary search algorithm.

In realistic scenarios, where the data is very large, search is considered as a crucial task in those systems. Thus, the complexity of the used search algorithm should be carefully selected. Unlike the exhaustive process of linear search, binary search is an example of algorithms that rely on the elements being indexed in advance in order to retrieve them faster. However, this improvement comes at the cost of indexing the elements in advance (ensuring that the arrays are sorted in our case).