Homework Assignment: Binary Search Algorithm Analysis and Implementation

Mauricio Estrella

September 19, 2025

1 Task 1 - Derivation

The average cost of the algorithm is known to be O(lg n). Derive this conclusion by yourself, step by step, and explicitly explain the assumptions on probability behind the conclusion.

I begin with assumptions to reach our $O(\log n)$ conclusion:

- **Assumption 1**: The search is always successful. We are only considering cases where the value being searched for exists within the array.
- **Assumption 2**: We assume that the probability of finding an element is of uniform probability. So $\frac{1}{n}$ where n is the size of the array.

Thus our average cost can be categorized as:

$$\sum_{i=0}^{m} \frac{c_i}{m} \tag{1}$$

where m is the number of possible instances, which for binary search is n in this case. Binary Search works by halving our search at each iteration, and thus we can visualize this binary search process onto a binary tree. If we think of the number of comparisons that happen to get to each level (depth) of the tree, then we see that the number of comparisons to find a node at depth d is $d + 1$. In addition, we know that at any depth, there are 2^d nodes. Thus, the total number of comparisons for all nodes at depth d is $2^d \times (d+1)$.
To get the total cost for the entire tree, we have to sum this expression across all the depths. If a tree has a height of h, the depths go from $d = 0$ to $d = h$ - 1. This gives us a total cost,

$$\sum_{i=1}^{n} c_i = \sum_{d=0}^{h-1} 2^d \times (d+1) \tag{2}$$

The approximation of this sum is $n \log n$. So, the average cost is $\frac{n \log n}{n}$, which simplifies to $\log n$.

2 Task 2 - Average Running Time

Decide whether the algorithm has the same average running time in the following two situations, and if not, which one is higher. Justify your conclusion (in English).

A. The value to be searched appears at any position in the array with the same probability (p_a), which is also the probability that there is no such value in the array.
B. The probability of the value being in the array (p_b) is the same as the probability of the opposite case (no such value in the array); when the value is in the array, the probability for it to appear in any position (p_c) is the same for all positions.

No, these two scenarios do not have the same average running time. Scenario B has the higher average running time and here is why:
In Scenario A, we have n places the number could be, plus 1 for "no such value in the array". So there are $n+1$ total and equally likely outcomes, thus the probability of an unsuccessful search is $\frac{1}{n+1}$ and the probability of a successful search is $\frac{n}{n+1}$. We can establish that an unsuccessful search is always a "worst-case" search and goes to the deepest level of the tree if we visualize our binary search process, again, as a binary tree. For an array of n elements there are $n+1$ gaps, so our probability of landing in a gap is $\frac{1}{n+1}$. In Scenario B, the probability of an unsuccessful search is $\frac{1}{2}$ and the probability of a successful search is $\frac{1}{2}$. Putting it all together:

- **Scenario A**: gives a very low probability $\frac{1}{n+1}$ to that slower, unsuccessful search. The average time will be very close to the faster successful search time.
- **Scenario B**: gives a much higher probability $\frac{1}{2}$ to the slower, unsuccessful search. This pulls the overall average up.

3 Task 3 - Average Cost Discussion

Explain (in English) why the average cost and the worst cost of this algorithm are both $O(lgn)$– shouldn't the worst cost be higher than the average cost?

As we have seen from the above task, the worst case scenario is about $\log n$ while the average case scenario is $\log n - 1$. They are both still directly proportional to $\log n$ and as our n grows to infinity, both our average and worst case grow logarithmically.

Since both are fundamentally tied to the same growth curve, we say they belong to the same complexity class: $O(\log n)$.

4 Task 4 - Program Results and Discussion

Write a program to estimate the average cost of the algorithm using random numbers in arrays of various sizes. Discuss the results and compare them with the above analysis. The requirements are similar to the programming part of Assignment 1.

Here are results of running binary sort on input sizes of 100 to 10000:

```
Input Size (n)  | Avg Comparisons      | Theoretical (log2 n)
-----------------------------------------------------------------
100             | 5.79                 | 6.64
500             | 7.96                 | 8.97
1000            | 8.87                 | 9.97
5000            | 10.90                | 12.29
10000           | 11.52                | 13.29
```

The main comparison in binary search happens when our while loop decides to go either left or right. I gathered all the comparisons that occurred for each input size as a total and then divided by n to get our average comparisons. I inserted the theoretical average in the next column to visualize the similarities and differences. To summarize:

- My results clearly confirm that the algorithm has a logarithmic $O(\log n)$ time complexity. The most obvious sign is when the input size n increased 100-fold (from 100 to 10,000), the average number of comparisons barely doubled (from 5.79 to 11.52).
- **Theoretical**: This value represents the "worst-case" scenario. It's the number of comparisons needed to find an element at the very end of a search path (a "gap" in the search tree).
- **My Average**: My experiment calculated the average cost by searching for every element. Some elements (like the one in the middle) are found very quickly in just one comparison. Others take longer. My average includes the easy, medium, and hard to find, so it ends up being slightly lower than the absolute "worst case" which is shown under the Theoretical column.

  Sample runs:

```
--- Starting Sample Run for Binary Search ---
Sorted list to search in: [2, 3, 10, 14, 23, 55, 89, 99]
Key to find: 10
Result: Found at index 2
Total comparisons: 3
--- End of Sample Run ---
--- Starting Sample Run for Binary Search ---
Sorted list to search in: [2, 3, 10, 14, 23, 55, 89, 99, 207, 308, 500]
Key to find: 500
```

```
Result: Found at index 10
Total comparisons: 4
--- End of Sample Run ---
```