

BINARY SEARCH – ALGORITHM IMPLEMENTATION AND PERFORMANCE ANALYSIS

Objective:

The objective of this task is to implement the Binary Search algorithm using C++ and analyze its performance by measuring execution time for different input sizes. The experiment also aims to verify the theoretical time complexity of the algorithm.

Algorithm Description:

Binary Search is an efficient searching algorithm that works on a **sorted array**. It repeatedly divides the search interval in half. If the target element is equal to the middle element, the search is successful. If the target is smaller than the middle element, the search continues in the left half; otherwise, it continues in the right half. This process continues until the element is found or the search interval becomes empty.

Steps involved:

1. Ensure the array is sorted.
2. Initialize two pointers: `low` (start index) and `high` (end index).
3. Find the middle index: `mid = (low + high) / 2`.
4. Compare the target element with `arr[mid]`.
5. If equal, return the index.
6. If the target is smaller, set `high = mid - 1`.
7. If the target is larger, set `low = mid + 1`.
8. Repeat until `low > high`.

Time and Space Complexity:

Case	Time Complexity
Best Case	$O(1)$
Average Case	$O(\log n)$
Worst Case	$O(\log n)$

Space Complexity:

- $O(1)$ (for iterative binary search)

Where n is the number of elements in the array.

Sample Input and Output:

```
-2\output'  
PS C:\Users\Roshini\Documents\GitHub\Alfido-Tech-Internship-2\out  
put> & .\'BinarySearch.exe'  
Enter number of elements: 10  
Enter element to search: 8  
Element found at index 4  
Time taken (nanoseconds): 0  
PS C:\Users\Roshini\Documents\GitHub\Alfido-Tech-Internship-2\out  
put>
```

Runtime Table:

Input Size	Time Taken (ms)
100	0.001
1,000	0.002
10,000	0.004

Compile and Run Commands:

Compile (Using g++):

g++ binarysearch.cpp -o binarysearch

Run:

./binarysearch

Conclusion:

The Binary Search algorithm was successfully implemented in C++. The measured runtime confirms that Binary Search performs in **$O(\log n)$** time for different input sizes, making it a highly efficient searching algorithm for large sorted datasets. However, it requires the input data to be sorted before searching.