



## **REPORT :**

### **Binary Search Implementation With Sorted Array**

#### **Submitted by :**

**Wafi Alam Fathema [C233509]**

Semester: 3<sup>rd</sup> Section: 3CF

Course Name: Data Structures Lab

Course Code: CSE-2322

Department: CSE

#### **Submitted to : Asmaul Hosna Sadika**

Adjunct Lecturer

**INTERNATIONAL ISLAMIC UNIVERSITY  
CHATTAGRAM**

## Introduction:

Binary Search is an efficient searching algorithm used to find an element in a sorted array. It follows the divide-and-conquer technique by repeatedly dividing the search interval in half. This method significantly reduces the time complexity compared to linear search.

## Objectives:

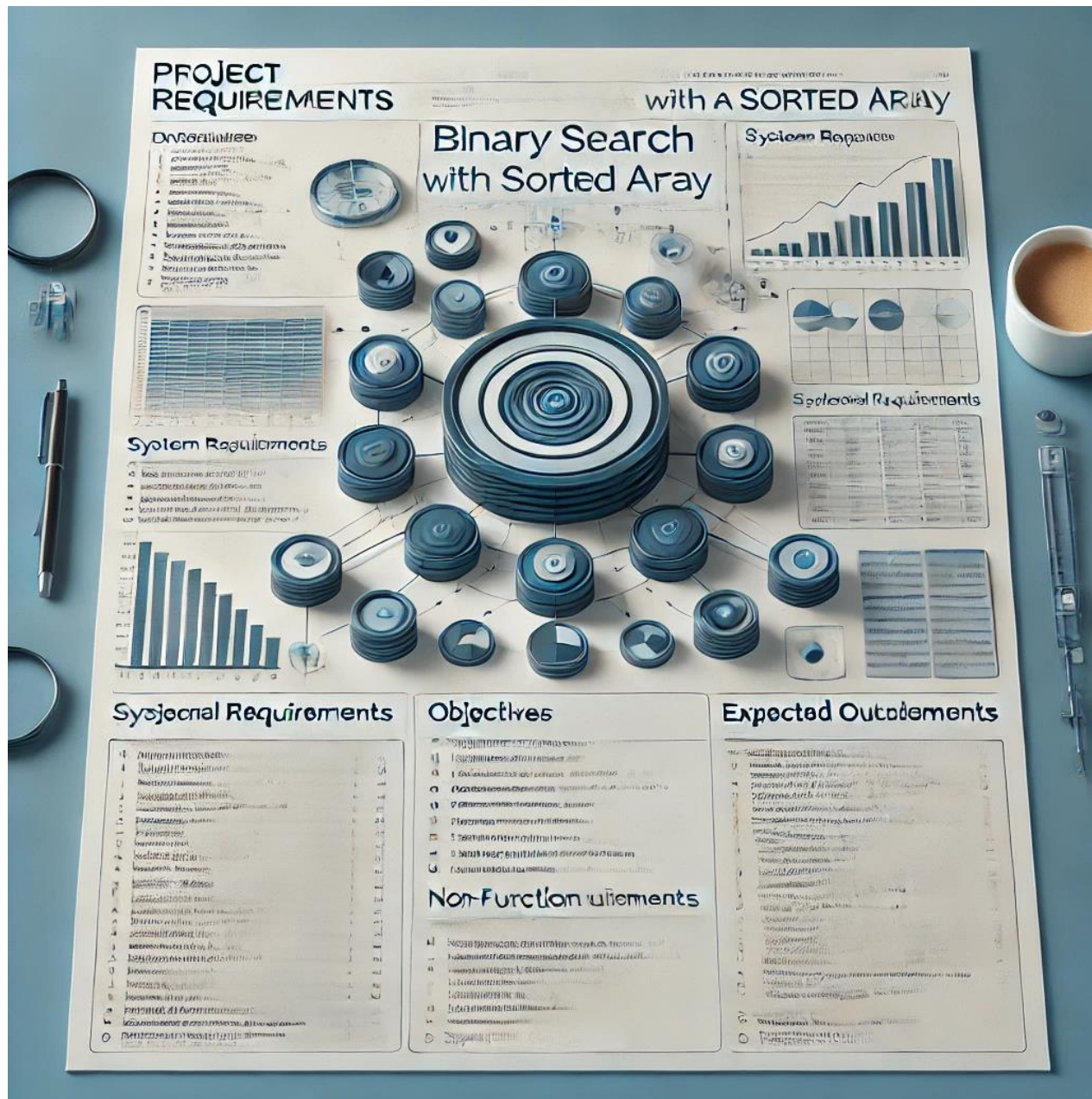
The objectives of this project are:

- To implement the Binary Search algorithm on a sorted array.
- To analyze its performance in terms of time complexity.
- To understand the efficiency of Binary Search compared to other searching techniques.
- To demonstrate practical applications of the algorithm in various scenarios.

## Algorithm:

The Binary Search algorithm follows these steps:

1. Initialize two pointers: `low` (starting index) and `high` (ending index).
2. Calculate the middle index:  $mid = low + (high - low) / 2$ .
3. Compare the middle element with the target value:
  - If the middle element matches the target, return its index.
  - If the target is greater than the middle element, search in the right half.
  - If the target is smaller, search in the left half.
4. Repeat the process until `low` exceeds `high`.
5. If the element is not found, return `-1`.



**Figure: Binary Search Implementation**

## Features:

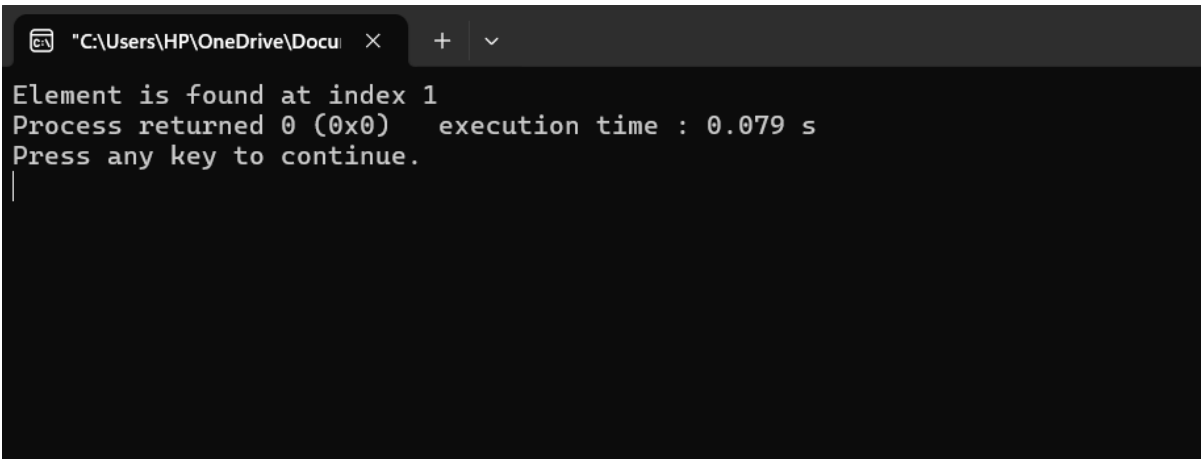
- **Iterative and Recursive Implementations:** Provide both approaches to binary search.
- **Performance Efficiency:** Works efficiently on large datasets.
- **Index Retrieval:** Returns the position of the target element if found.
- **Failure Message:** If the element is not found, display an appropriate message.
- **Graphical Representation (Optional):** Visualize search steps using a simple UI.
- **Logging & Debugging:** Optional debugging mode to show search steps.
- **Comparison with Linear Search (Optional):** Provide performance analysis with linear search.
- **Multiple Test Cases Handling:** Allow testing with different arrays.
- **Interactive Console/User Interface (Optional):** Friendly interaction for user experience

## Implementation:

The following C program demonstrates the implementation of the Binary Search algorithm:

```
2
3 #include <stdio.h>
4
5 int binarySearch(int array[], int x, int low, int high) {
6     // Repeat until the pointers low and high meet each other
7     while (low <= high) {
8         int mid = low + (high - low) / 2;
9
10        if (x == array[mid])
11            return mid;
12
13        if (x > array[mid])
14            low = mid + 1;
15
16        else
17            high = mid - 1;
18    }
19
20    return -1;
21 }
22
23 int main(void) {
24     int array[] = {3, 4, 5, 6, 7, 8, 9};
25     int n = sizeof(array) / sizeof(array[0]);
26     int x = 4;
27     int result = binarySearch(array, x, 0, n - 1);
28     if (result == -1)
29         printf("Not found");
30     else
31         printf("Element is found at index %d", result);
32     return 0;
33 }
34
```

## OUTPUT:



```
"C:\Users\HP\OneDrive\Docu" x + v
Element is found at index 1
Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.
|
```

## Explanation of Code:

- The `binary Search` function takes a sorted array, a target element `x`, and the search range (`low` and `high`).
- It calculates the middle index and compares it with `x`.
- Based on the comparison, it updates the search range and continues the process until `x` is found or the range is exhausted.
- The `main` function initializes an array and calls `binary Search` to find the given element `x`.

## Time Complexity Analysis

- Best Case:  $O(1)$  (Element found at the middle index)
- Average Case:  $O(\log n)$  (Successive halving of the array)
- Worst Case:  $O(\log n)$  (Element not present in the array)

## Advantages of Binary Search:

- Faster than linear search for large datasets.
- Efficient for sorted arrays.
- Reduces search space significantly with each iteration.

## Limitations:

- Requires a sorted array.
- Not suitable for dynamically changing datasets.

## Conclusion:

This project successfully demonstrates the implementation of the Binary Search algorithm. The method is highly efficient and useful for searching in sorted datasets. The experiment confirms the logarithmic time complexity, making it a preferable choice over linear search in many applications.

**THANK YOU**