S. B. JAIN INSTITUTE OF TECHNOLOGY,

MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 2

Aim: Implement a Binary Search algorithm for finding an element's position in a sorted array. Also find its complexities.

Name of Student:

Roll No:

Semester/Year: V/III

Academic Session:2024-2025

Date of Performance:

Date of Submission:

AIM: Implement a Binary Search algorithm for finding an element's position in a sorted array. Also find its complexities.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

- To understand the concepts of searching and sorting techniques.
- To understand and implement the binary search to find the position of an element.

THEORY:

Binary search is a searching algorithm which is used for searching specific data from an array (or any index accessible data structure). There are two preconditions to perform binary search on an array.

1. Data must be stored in indexed based data structure, like array, vector etc.
2. Data must be sorted in ascending or descending order.

When we perform a binary search we divide the array into two segments. In every operation, we continue searching only one segment and skip other one until we find the value.

Let consider an array {1, 2, 5, 6, 8, 9, 12, 15, 20}. Suppose we need to find the value 12. We will check the middle value. Here "middle" value means which element is now at the middle index. Let us consider 1 is the lowest index and 9 is the highest index. So, middle index will be (1+9)/2(integer division) = 5. Below we can see at 5th index the value is 8. 8 is less than 12. So it is confirmed that the value we are seeking is on the right side of the array. So next we'll search from index 6 to index 9. Now middle becomes (6+9)/2(integer division) = 7. At 7th index value is 12. So value is found at seventh index.

Index	1	2	3	4	5	6	7	8	9
Value	1	2	5	6	8	9	12	15	20

Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Binary search works on a sorted array. The value is compared with the middle element of the array. If equality is not found, then the half part is eliminated in which the value is not there. In the same way, the other half part is searched.
The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log n).

Applications of Binary Search

- This algorithm is used to search element in a given sorted array with more efficiency.
- It could also be used for few other additional operations like- to find the smallest element in the array or to find the largest element in the array.

Advantages:

- It is a much faster algorithm
- It works on the divide and conquers principle
- It is efficient

Disadvantages:

- It can be used only when data is sorted.
- It is more complicated.
- If random access is not supported then efficiency might be lost.

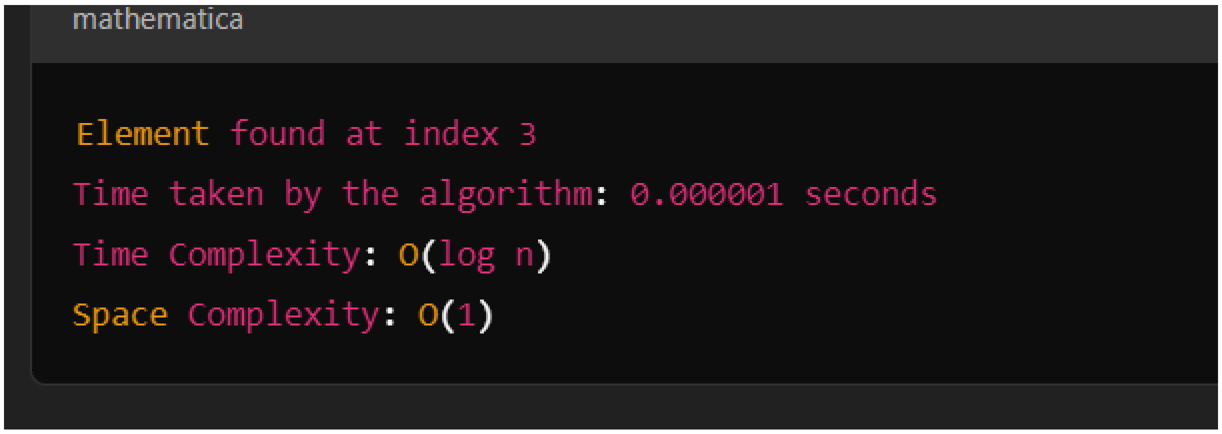
ALGORITHM:

```
binary_search(A, target):
    beg = 1, end = size(A)
    mid = (beg + end)/2
    while (beg <= end&& A[mid]!= target):
        if A[mid] < target:
            beg = mid+1
        else:
            end = mid-1
        mid=(beg+end)/2

    if A[mid] == target:
        return mid
    else:
        target was not found
```

CODE:

```
#include <stdio.h>
#include <time.h>  // For measuring time

// Function to perform binary search
int binarySearch(int arr[], int size, int target) {
    int left = 0, right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2; // To avoid potential overflow

        // Check if target is present at mid
        if (arr[mid] == target)
            return mid;

        // If target is greater, ignore left half
        if (arr[mid] < target)
            left = mid + 1;

        // If target is smaller, ignore right half
        else
            right = mid - 1;
    }

    // Target is not present in the array
    return -1;
}

int main() {
    int arr[] = {2, 3, 4, 10, 40, 50, 60};  // Example sorted array
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 10;  // Element to search for

    // Start measuring time
    clock_t start, end;
    double cpu_time_used;

    start = clock();  // Record the starting time

    // Call the binary search function
    int result = binarySearch(arr, size, target);

    end = clock();  // Record the ending time
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;  // Calculate time in seconds

    // Output result of binary search
    if (result != -1)
        printf("Element found at index %d\n", result);
    else
        printf("Element not found in the array\n");

    // Print time taken for the binary search
    printf("Time taken by the algorithm: %f seconds\n", cpu_time_used);

    // Print complexities
    printf("Time Complexity: O(log n)\n");
    printf("Space Complexity: O(1)\n");

    return 0;
}
```

INPUT & OUTPUT WITH DIFFERENT TEST CASES:



```
mathematica

Element found at index 3
Time taken by the algorithm: 0.000001 seconds
Time Complexity: O(log n)
Space Complexity: O(1)
```

CONCLUSION:

DISCUSSION AND VIVA VOCE:

- What is a search algorithm and explain the types of search?
- What is a binary search?
- Discuss the differences between binary search and linear search.
- What are pros and cons of binary search?
- What are the time and space complexity of binary search?

REFERENCES:

- *https://www.geeksforgeeks.org/binary-search/*
- *https://www.javatpoint.com/binary-search*
- *https://www.tutorialspoint.com/data_structures_algorithms/binary_search_algorithm.htm*
- *https://en.wikipedia.org/wiki/Binary_search_algorithm*

Post_Practical: Implement the binary search of an element in a circularly sorted array.