Experiment No.11
Application of Binary Search Technique
Name: Shravani Sandeep Raut
Roll No: 48
Date of Performance:
Date of Submission:

Experiment No. 11: Application of Binary Search Technique

Aim: Application of Binary Search Technique.

Objective:

1) Understand the optimal search algorithm in terms of time.

2) Understand the method searching technique.

Theory:

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique. Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

Binary search is much faster than linear search for large datasets, with a time complexity of O(log n). The iterative version of binary search uses O(1) additional space, making it memory efficient.

Algorithm:

Binary_Search(a, lower_bound, upper_bound, val) // 'a' is the given array, 'lower_bound' is the index of the first array element, 'upper_bound' is the index of the last array element, 'val' is the value to search

Step 1: set beg = lower_bound, end = upper_bound, pos = - 1

Step 2: repeat steps 3 and 4 while beg <=end

Step 3: set mid = (beg + end)/2

Step 4: if a[mid] = val

set pos = mid

print pos

go to step 6

else if a[mid] > val

set end = mid - 1

else

set beg = mid + 1

[end of if]

[end of loop]

Step 5: if pos = -1

print "value is not present in the array"

[end of if]

Step 6: exit

Code:

```
#include <stdio.h>
#define MAX 100
int a[MAX], n , i, x;
int low = 0, high = 0, mid = 0, found =0;

void main()
{
   printf("Enter the size of array: ");
   scanf("%d", &n);

   printf("Enter sorted array only\n");
   for(i=0; i<n; i++)
   {
      printf("Enter value: ");
      scanf("%d", &a[i]);
   }

   printf("Enter the element to be searched: ");
   scanf("%d", &x);

   low= 0;
   high = n-1;
   while(high >= low)
   {
      mid = (high + low)/2;
      if( a[mid] == x)
      {
         printf("Search successful\nFound the element at %d",mid);
         found =1;
         break;
      }
      else if(a[mid]> x)
      {
         high = mid-1;
      }
      else
      {
         low = mid +1;
```

```
    }
  }
  if(low>high && found ==0)
  {
    printf("Search unsuccessful!! Element not found");
  }

}
```

Output:

```
Enter the size of array: 5
Enter sorted array only
Enter value: 3
Enter value: 9
Enter value: 15
Enter value: 40
Enter value: 67
Enter the element to be searched: 9
Search successful
Found the element at 1

=== Code Exited With Errors ===
```

Conclusion:

1) What is searching?

Searching is the process of finding a specific element or value within a collection of data, such as an array, list, or database. The goal is to determine whether the element exists in the collection and, if so, its location.

2) Differentiate between binary search and linear search.

1. Algorithm Type:

- **Linear Search:** A **sequential** search algorithm. It checks each element of the list one by one until the desired element is found or the list ends.
- **Binary Search:** A **divide-and-conquer** algorithm. It works by repeatedly dividing the sorted list in half, eliminating half of the remaining elements at each step.

2. Array Requirement:

- **Linear Search:** Works on unsorted or sorted arrays.
- **Binary Search:** Requires the array to be sorted beforehand.

3. Efficiency:

- **Linear Search:** Less efficient for large datasets as it has to potentially check every element.
- **Binary Search:** More efficient for large datasets, especially when the array is already sorted, as it reduces the number of checks required.