AMITY SCHOOL OF ENGINEERING & TECHNOLOGY

AMITY UNIVERSITY CAMPUS, SECTOR-125, NOIDA-201303

ANALYSIS AND DESIGN OF ALGORITHMS LAB

PRACTICAL FILE

COURSE CODE: CSE303

Submitted to:
Dr. Sumita Gupta

Submitted by:
Boddu Asmitha Bhavya
5CSE6-X
A2305221386

INDEX

S.no	Experiment	Date of Allotment	Date of Evaluation	Max Marks	Marks Obtained	Sign

EXPERIMENT-1

Program:
WAP to implement linear search using c/c++ and write the time complexity.

Psuedo code:

```
Start

linear_search ( Array , value)

For each element in the array

    If (searched element == value)

        Return's the searched lament location

    end if

end for

end
```

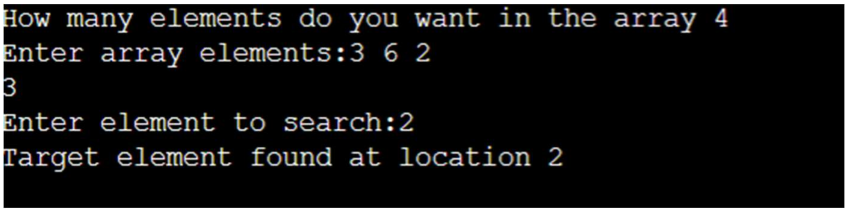
Input:

```
#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

int main()

{

    int array[50],i,target,num;

    printf("How many elements do you want in the array");

    scanf("%d",&num);

    printf("Enter array elements:");

    for(i=0;i<num;++i)

        scanf("%d",&array[i]);

    printf("Enter element to search:");
    scanf("%d",&target);

    for(i=0;i<num;++i)

        if(array[i]==target)

            break;

    if(i<num)

        printf("Target element found at location %d",i);

    else

        printf("Target element not found in an array");

    return 0;

}
```

Time Complexity:

- Best Case: In the best case, the key might be present at the first index. So the best case complexity is O(1)
- Worst Case: In the worst case, the key might be present at the last index i.e., opposite the end from which the search has started in the list. So, the worst-case complexity is O(N) where N is the size of the list.
- Average Case: O(N)

Output:



```
How many elements do you want in the array 4
Enter array elements:3 6 2
3
Enter element to search:2
Target element found at location 2
```

EXPERIMENT-2

Program:
WAP to implement the binary search (NON RECURSIVE AND RECURSIVE) using c/c++ and write thee time complexity.

I)NON-RECURSIVE / ITERATIVE METHOD:

Pseudo Code:

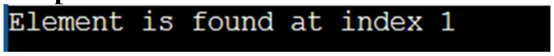
```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid])
        low = mid + 1
    else
        high = mid – 1
```

Input:

```
#include <stdio.h>
int binarySearch(int array[], int x, int low, int high) {
  while (low <= high) {
    int mid = low + (high - low) / 2;
    if (array[mid] == x)
      return mid;
    if (array[mid] < x)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return -1;
}
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
  return 0;
}
```

Time Complexity:
O(log N)

Output:



```
Element is found at index 1
```

II)RECURSIVE METHOD:

Pseudo Code:

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]
            return binarySearch(arr, x, mid + 1, high)
        else
            return binarySearch(arr, x, low, mid - 1)
```

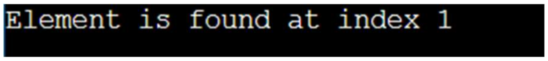
Input:

```
#include <stdio.h>
int binarySearch(int array[], int x, int low, int high) {
  if (high >= low) {
    int mid = low + (high - low) / 2;
    if (array[mid] == x)
      return mid;
    if (array[mid] > x)
      return binarySearch(array, x, low, mid - 1);
    return binarySearch(array, x, mid + 1, high);
  }
  return -1;
}
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
}
```

Time Complexity:

Best Case: O(1)
Average Case: O(log N)
Worst Case: O(log N)

Output:



```
Element is found at index 1
```