WEEK-2

Date:25-06-2025

List of programs:

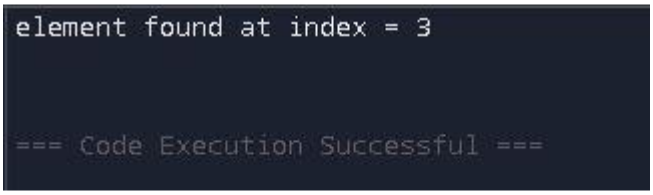
1. Write a C program to search an element in the given list using recursive Linear Search technique.
2. Write a C program to search an element in the given list using non-recursive Linear Search technique.
3. Write a C program to search an element in the given sorted list using recursive Binary Search technique.
4. Write a C program to search an element in the given sorted list using recursive Binary Search technique.

1. **Aim:** To write a C program to search an element int the given list using recursive Linear Search technique.

Program:

```
#include<stdio.h>
int LinearSearch(int arr[],int size,int key,int index)
{
if(index>=size)
    return -1;
if(arr[index]==key)
    return index;
return LinearSearch(arr,size,key,index+1);
}
int main()
{
int arr[]={5,3,8,4,2};
int size=sizeof(arr)/sizeof(arr[0]);
int key=4;
int result=LinearSearch(arr,size,key,0);
if(result!=-1)
       printf(“element found at index = %d\n”,result);
else
      printf(“element not found in the list\n”);
return 0;
}
```

Output:



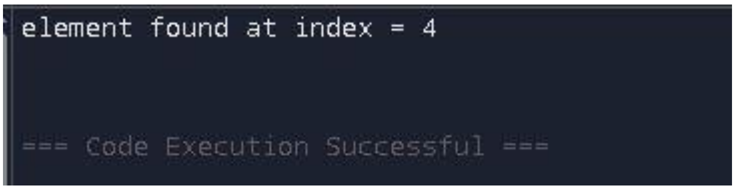
```
element found at index = 3


=== Code Execution Successful ===
```

2. **Aim:** To Write a C program to search an element in the given list using non-recursive Linear Search technique.

Program:

```
#include<stdio.h>
int LinearSearch(int arr[],int size,int key)
{
 int i;
  for(i=0;i<size;i++)
   {
    if(arr[i]==key)
   {
     return i;
    }
   }
  return -1;
}
int main()
{
int arr[]={5,3,8,4,2};
int size=sizeof(arr)/sizeof(arr[0]);
int key=2;
int result=LinearSearch(arr,size,key);
if(result!=-1)
    printf("element found at index = %d\n",result);
else
    printf("element not found in the list\n");
return 0;
}
```

Output:



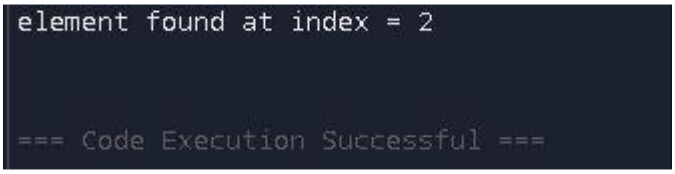
```
element found at index = 4


=== Code Execution Successful ===
```

3. **Aim:** Write a C program to search an element in the given sorted list using recursive Binary Search technique.

Program:

```
#include<stdio.h>
int BinarySearch(int arr[],int left,int right,int x)
{
if(right>=left)
{
  int mid=(left+right)/2;
  if(arr[mid]==x)
      return mid;
  if (arr[mid]>x)
     return BinarySearch(arr,left,mid-1,x);
 return BinarySearch(arr,mid+1,right,x);
 }
return -1;
}
int main()
{
int arr[]={2,3,4,10,40};
int n=sizeof(arr)/sizeof(arr[0]);
int x=4;
int result=BinarySearch(arr,0,n-1,x);
if(result!=-1)
     printf("element found at index = %d\n",result);
else
    printf("element not found in the list\n");
return 0;

}
```

Output:



```
element found at index = 2


=== Code Execution Successful ===
```

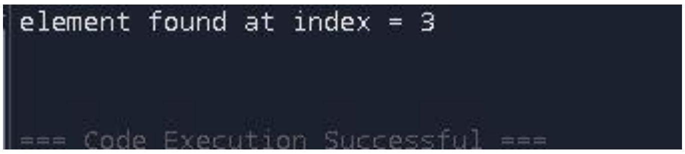
4. **Aim:** Write a C program to search an element in the given sorted list using non-recursive Binary Search technique.

Program:

```
#include<stdio.h>

int BinarySearch(int arr[],int size,int target)

{

int left=0;

int right=size-1;

while(left<=right)

{

  int mid=(left+right)/2;

  if(arr[mid]==target)

       return mid;

  if (arr[mid]<target)

      left=mid+1;

 else

      right=mid-1;

 }

return -1;

}

int main()

{

int arr[]={2,3,4,10,40};

int size=sizeof(arr)/sizeof(arr[0]);

int target=4;

int result=BinarySearch(arr,size,target);

if(result!=-1)
```

```
      printf(“element found at index = %d\n”,result);
else
     printf(“element not found in the list\n”);
return 0;
}
```

Output:



```
element found at index = 3


=== Code Execution Successful ===
```

Inferences:

- Linear search is a method of checking each element of the array sequentially until the element is found or the list ends.
- Recursive linear search is simple to implement and works on unsorted array.
- Non-recursive linear search use a simple loop to scan through the array.
- In this non-recursive linear search No recursion overhead but still **O(n)** time complexity.
- Binary search is a method Works only on **sorted arrays**. Repeatedly divide the array into halves and check the middle element.
- Recursive binary search has time complexity **O(log n)** that is much faster than linear search and has clear recursive structure.
- Non recursive binary search is implemented using a **while loop** (iterative approach).
- No recursion overhead (stack saving) and still **O(log n)** and efficient.