

Assignment -1:

- a) WAP to implement LINEAR SEARCH iterative as well as recursive method.
- b) WAP to implement BINARY SEARCH iterative as well as recursive method.

```
(a) Linear Search –
<u>Iterative</u> –
<u> Algorithm</u> –
       Function: linearSearch(arr, size, element)
       Input: arr – an array whose elements are not ordered.
              size – number of elements in arr.
              element – the element to be searched.
       Output: Returns -1 if search is unsuccessful or location of element in the array
       arr if search is successful.
       1. for i \leftarrow 0 to size - 1 do
              if arr[i] = element then
       3.
                      return i.
       4.
              end if.
              return -1.
       6. end for.
<u>Program</u> –
       #include<stdio.h>
```

```
int linearSearch(int arr[], int size, int element)
    for (int i = 0; i < size; i++)
        if(arr[i] == element)
            return i;
    return -1;
}
int main()
    int element, size, searchIndex;
```

```
printf("Give the size of the array: ");
scanf("%d", &size);
int arr[size];
printf("\n");
for (int i = 0; i < size; i++)
    printf("Enter the array element: ");
    scanf("%d", &arr[i]);
}
printf("\nEnter the element to be searched: ");
scanf("%d", &element);
searchIndex = linearSearch(arr, size, element);
if (searchIndex == -1)
{
    printf("Element not found.");
}
else
{
    printf("Index is %d.", searchIndex);
}
return 0;
```

}

```
Recursive -
<u> Algorithm</u> –
      Function: linearSearch(arr, l, r, x)
      Input: arr – an array whose elements are not ordered.
            I – lower bound of arr.
            r – higher bound of arr.
            x – the element to be searched.
      Output: Returns -1 if search is unsuccessful or location of x in the array arr if
      search is successful.
      1. if r < I then
      2.
            return -1.
      3. end if.
      4. if arr[l] = element then
      5.
            return I.
      6. end if.
      7. if arr[r] = element then
      8.
            return r.
      9. end if.
      10. return linearSearch(arr, l + 1, r - 1, x)
Program -
      #include<stdio.h>
      int linearSearch(int arr[], int l, int r, int x)
      {
           if (r < 1)
                return -1;
           if (arr[1] == x)
                return 1;
           if (arr[r] == x)
                return r;
           return linearSearch(arr, 1 + 1, r - 1, x);
      }
      int main()
           int element, size, searchIndex;
           printf("Give the size of the array: ");
           scanf("%d", &size);
           int arr[size];
```

Falguní Sarkar_Roll No.: 11900119031_CSE (A)

for (int i = 0; i < size; i++)

printf("\n");

```
{
    printf("Enter the array element: ");
    scanf("%d", &arr[i]);
}

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

searchIndex = linearSearch(arr, 0, size-1, element);

if (searchIndex == -1)
{
    printf("Element not found.");
}
else
{
    printf("Index is %d.", searchIndex);
}
return 0;
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

Loading personal and system profiles took 1471ms.
Falguni Sarkar@MELOPHILE CANSESTER-UNDSIGN & Analysis of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } if ($?) { \linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } if ($?) { \linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } if ($?) { \linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRecursive.c -0 linearSearchRecursive } diversity of Algorithm\Lab\Assignment 1\" ; if ($?) { gcc linearSearchRe
```

```
(b) Binary Search -
<u>Iterative</u> –
Algorithm –
      Function: binarySearch(arr, low, high, element)
      Input: arr – a linear array containing sorted elements.
             low – lower bound of array arr.
             high – upper bound of array arr.
             element – the element to be searched.
      Output: Returns -1 if search is unsuccessful or location of element in the array
      arr if search is successful.
      1. while low ≤ high do
      2.
             mid \leftarrow (low + high)/2
      3.
             if arr[mid] = element then
      4.
                    return mid.
      5.
             else if arr[mid] < element then
      6.
                    low \leftarrow mid + 1.
      7.
             else
      8.
                    high \leftarrow mid − 1.
      9.
             end if.
      10. end while.
      11. return -1.
Program -
      #include<stdio.h>
      int binarySearch(int arr[], int low, int high, int element)
      {
            int mid;
            while(low <= high)</pre>
            {
                 mid = (low+high)/2;
                 if(arr[mid] == element)
                 {
                      return mid;
                 else if(arr[mid] < element)</pre>
                       low = mid+1;
                 }
                 else
                 {
                      high = mid-1;
                 }
            }
```

```
return -1;
}
int main()
    int element, size, searchIndex;
    printf("Give the size of the array: ");
    scanf("%d", &size);
    int arr[size];
    printf("\n");
    for (int i = 0; i < size; i++)
    {
        printf("Enter the array element: ");
        scanf("%d", &arr[i]);
    }
    printf("\nEnter the element to be searched: ");
    scanf("%d", &element);
    searchIndex = binarySearch(arr, 0, size-1, element);
    if (searchIndex == -1)
    {
        printf("Element not found.");
    }
    else
    {
        printf("Index is %d.", searchIndex);
    return 0;
}
```

<u>Recursive</u> –

Algorithm -

Function: binarySearch(arr, low, high, element)

Input: arr – a linear array containing **sorted** elements.

low – lower bound of array **arr**.

high – upper bound of array arr.

element – the element to be searched.

Output: Returns -1 if search is unsuccessful or location of **element** in the array **arr** if search is successful.

- 1. if low ≤ high then
- 2. $mid \leftarrow (low + high)/2$
- 3. if arr[mid] = element then
- 4. return mid.
- 5. else if arr[mid] < element then
- 6. return binarySearch(arr, mid+1, high, element).
- 7. else
- 8. return binarySearch(arr, low, mid-1, element)
- 9. end if.
- 10. end if.
- 11. return -1.

<u>Program</u> –

```
#include<stdio.h>
int binarySearch(int arr[], int low, int high, int element)
    int mid;
    if(low <= high)</pre>
    {
        mid = (low+high)/2;
        if(arr[mid] == element)
        {
            return mid;
        }
        else if(arr[mid] < element)</pre>
            return binarySearch(arr, mid+1, high,
                                                 element);
        }
        else
        {
            return binarySearch(arr, low, mid-1, element);
        }
    }
    return -1;
}
int main()
    int element, size, searchIndex;
    printf("Give the size of the array: ");
    scanf("%d", &size);
    int arr[size];
    printf("\n");
    for (int i = 0; i < size; i++)
    {
        printf("Enter the array element: ");
        scanf("%d", &arr[i]);
    }
    printf("\nEnter the element to be searched: ");
    scanf("%d", &element);
```

```
searchIndex = binarySearch(arr, 0, size-1, element);

if (searchIndex == -1)
{
    printf("Element not found.");
}
else
{
    printf("Index is %d.", searchIndex);
}
return 0;
}
```