Harsh Kasliwal
2003085
C21

Experiment 10

AIM: Implement Linear and Binary Search algorithm.

Theory:

⇒ Linear Search: The linear search algorithm searches all elements in the array sequentially. When ~~a~~ data is unsorted, a linear search algorithm is preffered.

Space complexity for linear search is O(n) as it does not use any extra space where n is the no. of elements in an array.

Algorithm:

```
LinearSearch (A, n, x) {
1. for i=0 to n-1
2. { if (A[i] == x)
          return }
3. return -1
}.
```

Example :- We are given the following linear array.

Element 15 has to be searched in it using Linear search Algorithm.

92	87	53	10	15	21	77
0	1	2	3	4	5	6

LSA compares 15 with all the elements of the array one by one. It continues searching until either the element 15 is found or all the elements are searched. It first compares with 1st element '92' since 15 ≠ 92 it moves to next and so on...

Harsh Kaliwal
085

Binary Search: Binary is one of the fastest searching algorithms

To find an element;

=> Set the range on index in which element is to be searched, there are 3 cases
1. If it matches, element is at middle index.
2. If search value is less than value at middle index, then upper bound is shifted to the first half of the array.
3. If search value is greater than value at middle index, then lower bound is shifted to the second half of the array.

The process terminates when searched elements is found or process is repeated.

Algorithm: Iterative.

```
BinarySearch (A,n,x){
  start=0, end=n-1
  while (start<=end){
      mid= (start + end)/2
      if (A[mid] ==x)
          return mid
      else if (x < A[mid])
          end =mid-1
      else start = mid+1
  }
  return -1
}
```

Harsh Kasliwal
2003085
(2)

# Algorithm : Recursive

```
BinarySearch(A, start, end, x){
    if (start > end)
        return -1
    mid = (start + end)/2
    if (x == A[mid])
        return mid
    else if (x < A[mid])
    return BinarySearch(A, start, mid-1, x)
    else,
     return Binary Search(A, mid+1, end, x)
}.
```

⇒ In each iteration or in each recursive call, the search gets reduced to half of the array. So for n elements in the array, there are $\log_2 n$ iterations or recursive calls.

∴ Time complexity of Binary Search Algorithm is $O(\log_2 n)$.

# Example :

3	10	20	15	40	35	60
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]

15 has to be searched.

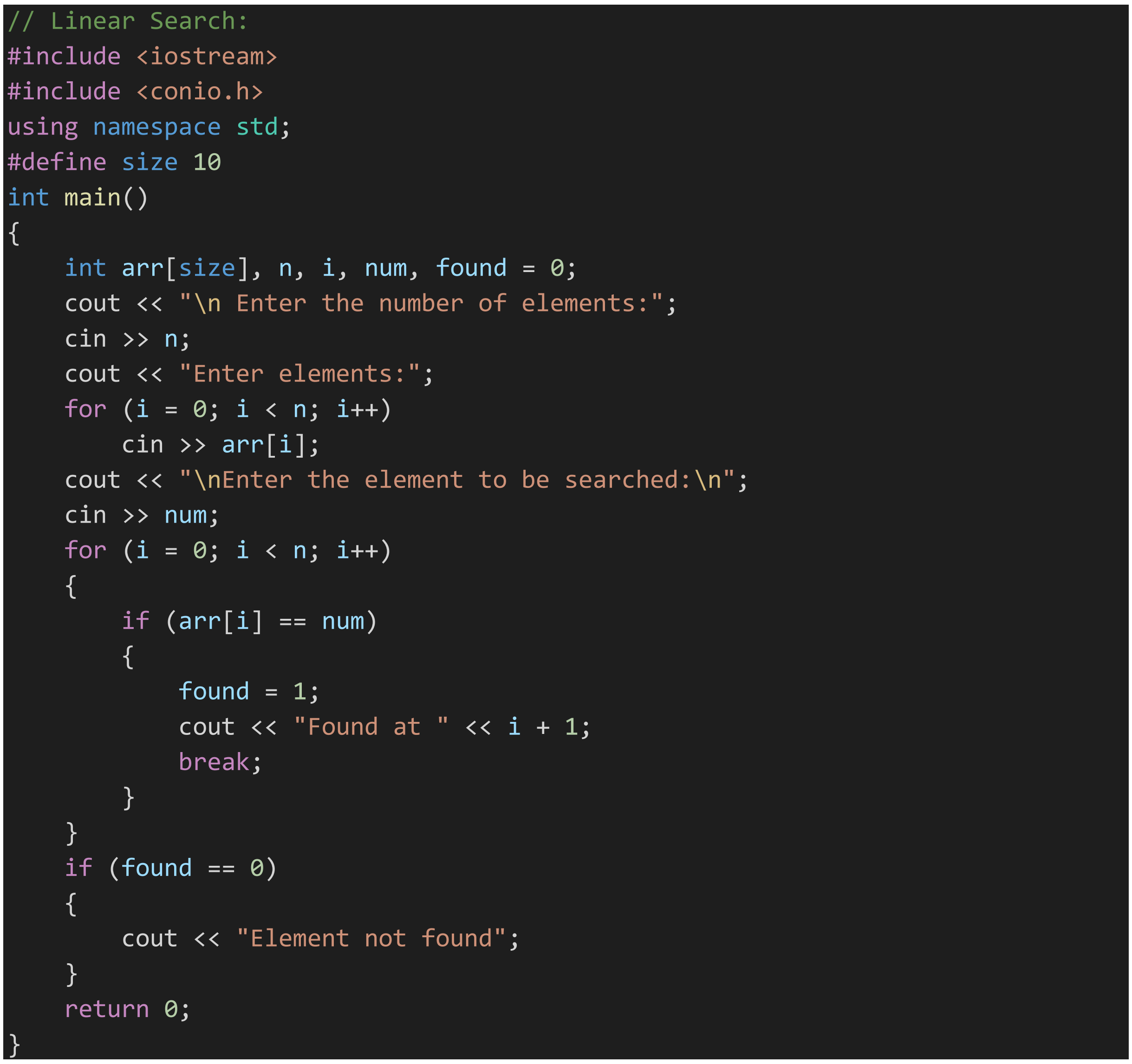
1. To begin with, we take start = 0 and end = 6.
2. mid = (start + end)/2
   = (0+6)/2 = 3.
3. Here, a[mid] = a[3] = 15 which matches to the element being searched.
4. So, our search terminates and index 3 is returned.

Harsh Kashiwal
2003085

Conclusion: Binary search is more efficient and takes minimum time to search an element than a linear search.
But linerary search is less complex than binary search.

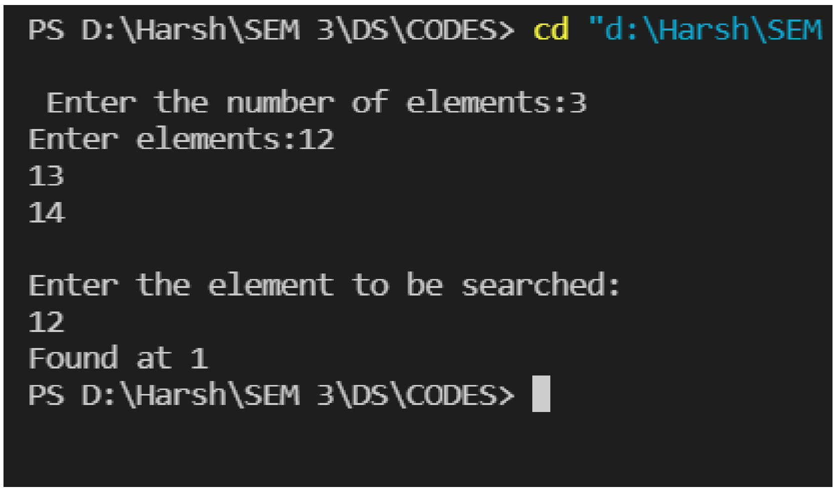
PROGRAM:

Linear Search



```cpp
// Linear Search:
#include <iostream>
#include <conio.h>
using namespace std;
#define size 10
int main()
{
    int arr[size], n, i, num, found = 0;
    cout << "\n Enter the number of elements:";
    cin >> n;
    cout << "Enter elements:";
    for (i = 0; i < n; i++)
        cin >> arr[i];
    cout << "\nEnter the element to be searched:\n";
    cin >> num;
    for (i = 0; i < n; i++)
    {
        if (arr[i] == num)
        {
            found = 1;
            cout << "Found at " << i + 1;
            break;
        }
    }
    if (found == 0)
    {
        cout << "Element not found";
    }
    return 0;
}
```

OUTPUT: Linear

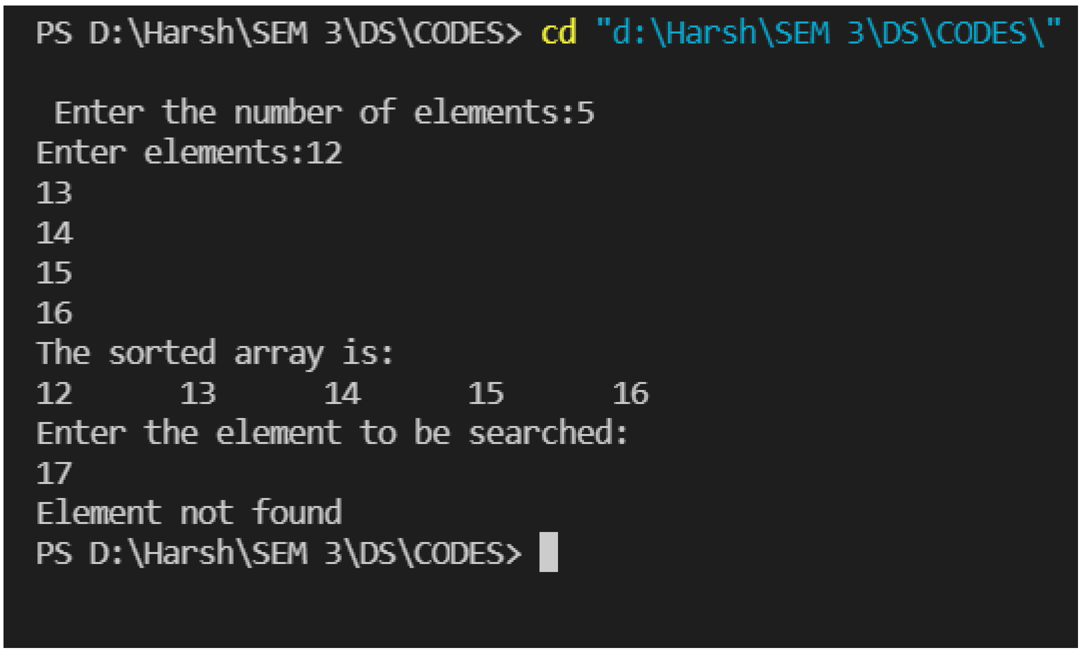


```
PS D:\Harsh\SEM 3\DS\CODES> cd "d:\Harsh\SEM

 Enter the number of elements:3
Enter elements:12
13
14

Enter the element to be searched:
12
Found at 1
PS D:\Harsh\SEM 3\DS\CODES>
```

Binary Search:

```
//Binary Search:
#include <iostream>
#include <conio.h>
using namespace std;
#define size 10
int smallest(int arr[], int k, int n){
        int pos = k, small = arr[k], i;
        for (i = k + 1; i < n; i++)
        {
            if (arr[i] < small)
            {
                small = arr[i];
                pos = i;
            }
        }
        return pos;
    }
 void selection_sort(int arr[], int n)
    {
        int k, pos, temp;
        for (k = 0; k < n; k++)
        {
            pos = smallest(arr, k, n);
            temp = arr[k];
            arr[k] = arr[pos];
            arr[pos] = temp;
        }
    }
int main()
{
    int arr[size], n, i, num, start, end, mid, found = 0;
    cout << "\n Enter the number of elements:";
    cin >> n;
    cout << "Enter elements:";
    for (i = 0; i < n; i++)
        cin >> arr[i];
    selection_sort(arr, n);
    cout << "The sorted array is:\n";
    for (i = 0; i < n; i++)
        cout << arr[i] << "\t";
    cout << "\nEnter the element to be searched:\n";
    cin >> num;
    start = 0;
    end = n - 1;
    while (start <= end)
    {
        mid = (start + end) / 2;
        if (arr[mid] == num)
        {
            cout << "Found at " << mid + 1;
            found = 1;
            break;
        }
        else if (arr[mid] > num)
            end = mid - 1;
        else
            start = mid + 1;
    }
    if (start > end && found == 0)
    {
        cout << "Element not found";
    }


    return 0;
}
```

OUTPUT: Binary



```
PS D:\Harsh\SEM 3\DS\CODES> cd "d:\Harsh\SEM 3\DS\CODES\"

 Enter the number of elements:5
Enter elements:12
13
14
15
16
The sorted array is:
12      13      14      15      16
Enter the element to be searched:
17
Element not found
PS D:\Harsh\SEM 3\DS\CODES> 
```