Term work

on

Design & Analysis of Algorithms

LAB

(PCS 409)

2021-22

Submitted to: **Submitted by:**

Dilip Kumar Gangwar	Devansh Rautela
Asst. Professor	University Roll. No.:2018314
GEHU, D.DUN	CLASS ROLLNO./SECTION: 20/C

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, DEHRADUN

ACKNOWLEDGMENT

I would like to particularly thank my DAA Lab Faculty **Mr Dilip Kumar Gangwar** for his patience, support and encouragement throughout the completion of this Term work.

At last but not the least I greatly indebted to all other persons who directly or indirectly helped me during this course.

Devansh Rautela

Univ. Roll No.- 2018314

B.Tech CSE-C-IV Sem

Session: 2021-2022

GEHU, Dehradun

Graphic Era HILL UNIVERSITY

Established by an Act of the State Legislature of Uttarakhand (Adhiniyam Sankhya 12 of 2011)

Table of Contents

PROGRAM 1

I. Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(n), where n is the size of input).

Algorithm :

```
Algorithm linearSearch(a[0…n-1],n,key)
{
no_of_comparisons = 0
for i=0 to n
{
no_of_comparisons++
if(a[i] == key)
Print " key is found"
return no_of_comparisons
}
if(i==n)
Print " key is NOT found"
return no_of_comparisons
}
```

Complexity Analysis :

Time Complexity :

Worst Case: T(n) = θ(n)

Best Case: T(n)= Ω(1)

Average Case:T(n) = θ(n)

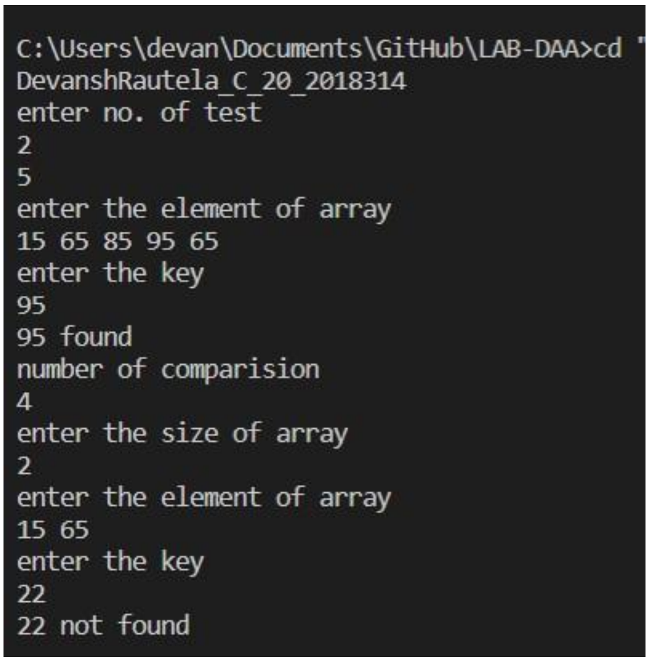
Space Complexity:

Worst Case=Best Case = Average case = θ(1)

Source Code :

```
#include <iostream>
using namespace std;
int main()
{
cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0)
 {
    int n;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    cout<<"enter the element of array"<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
    int key;
    cout<<"enter the key"<<endl;
    cin>>key;
    int c=0,f=0;
    for(int i=0;i<n;i++)
    {       c++;
        if(arr[i]==key)
        {
            cout<<key<<" found"<<endl;
            cout<<"number of comparision"<<endl;
            cout<<c;
            f=1;
            break;
        }
    }
    if(f==0)
    cout<<key<<" not found"<<endl;
    t--;
 }
    return 0;
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "
DevanshRautela_C_20_2018314
enter no. of test
2
5
enter the element of array
15 65 85 95 65
enter the key
95
95 found
number of comparision
4
enter the size of array
2
enter the element of array
15 65
enter the key
22
22 not found
```

PROGRAM 2

2. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case. (Time Complexity = O(logn), where n is the size of input).

Algorithm:

```
Binary_Search(a, lower_bound, upper_bound, val)
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
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = $\theta\log(n)$

Best Case: T(n)= $\Omega(1)$

Average Case:T(n) = $\theta\log(n)$

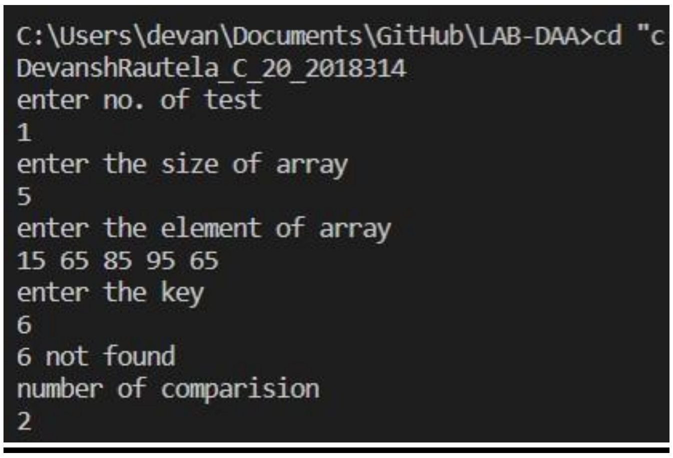
Space Complexity:

Worst Case=Best Case = Average case = $\theta(1)$ [Constant]

Source code :

```
#include <iostream>
using namespace std;
void binary(int *arr,int n,int key)
{
    int l=0,r=n-1;
    int c=0;
    while(l<r)
    {
        int mid=(l+r)/2;
        c++;
        if(arr[mid]==key)
        {
            cout<<key<<" found"<<endl;
            cout<<"number of comparision"<<endl;
            cout<<c;
            return ;
        }
        if(arr[mid]>key)
          r=mid-1;
        else
          l=mid+1;
    }
    cout<<key<<" not found"<<endl;
    cout<<"number of comparision"<<endl;
    cout<<c;
    return ;
}
int main()
{
cout<<"DevanshRautela_C_20_2018314"<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0)
 {
    int n;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    cout<<"enter the element of array"<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
    int key;
    cout<<"enter the key"<<endl;
    cin>>key;
    int c=0;
    binary(arr,n,key);
    t--;
 }
    return 0;
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
5
enter the element of array
15 65 85 95 65
enter the key
6
6 not found
number of comparision
2
```

PROGRAM 3

3. Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array arr[n], search at the indexes arr[0], arr[2], arr[4],. ,arr[2 k] and so on. Once the interval (arr[2 k] < key < arr[2 k+1]) is found, perform a linear search operation from the index 2 k to find the element key. (Complexity < O(n), where n is the number of elements need to be scanned for searching): (Jump Search)

Algorithm:

```
Algorithm JumpSearch(a[0…n-1],n,key)
Step 1: set r=2^1,i=1
Step 2: repeat steps 3 and 4 while r <=n
Step 3: if a[r]<key
            for j=i-1 to r
               if a[j]== key
                   print found
                       exit
Step 4:i=r, r=2^i
Step 5:if r>n then
               for j=i-1 to n
               if a[j]== key
                   print found
                   exit
Step 6:print not found
Step 7:exit
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ(sqrt n)

Best Case: T(n)= Ω(sqrt n)

Average Case:T(n) = θ(sqrt n)

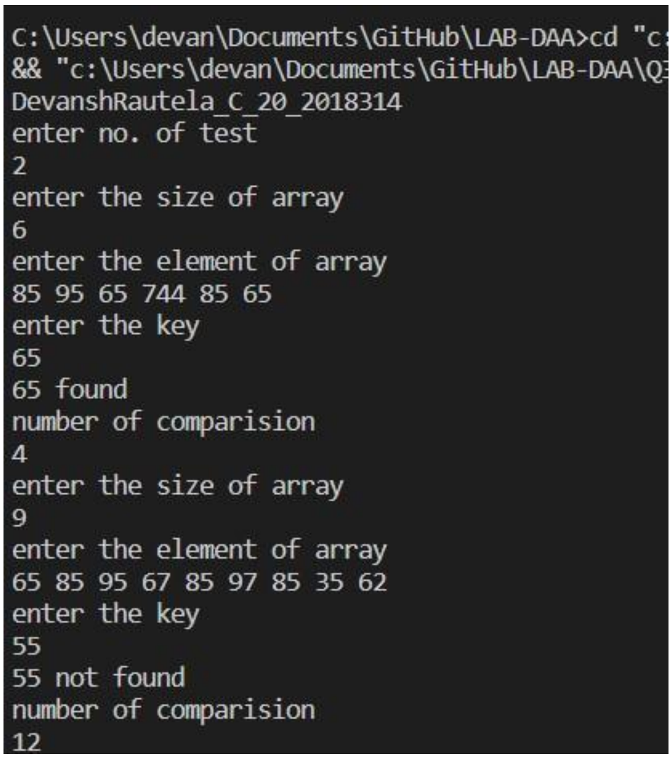
Space Complexity:

Worst Case=Best Case = Average case = θ(1)

Source code :

```
#include <iostream>
#include <math.h>
using namespace std;
void jumpsearch(int *arr,int n,int key)
{int r=pow(2,1);
 int l=0;
 int i=1;
 int c=0;
 while(r<=n)
 {
   c++;
   if(arr[r]<arr[i])
   {
    for(int j=l;j<=r;j++)
    {
    c++;
    if(arr[j]==key)
    {
       cout<<key<<" found"<<endl;
       cout<<"number of comparision"<<endl;
       cout<<c;
       return ;
    }
    }
   }
   i++;
   l=r;
   r=pow(2,i);
 }
 if(r>n){
       for(int j=l;j<n;j++){
        c++;
        if(arr[j]==key) {
            cout<<key<<" found"<<endl;
            cout<<"number of comparision"<<endl;
            cout<<c;
            return ;
        }  }  }
    cout<<key<<" not found"<<endl;
    cout<<"number of comparision"<<endl;
    cout<<c;
    return ;}
int main(){
cout<<"DevanshRautela_C_20_2018314"<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0) {
    int n, key;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    cout<<"enter the element of array"<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
    cout<<"enter the key"<<endl;
    cin>>key;
    jumpsearch(arr,n,key);
    t--;
 } return 0;
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c
&& "c:\Users\devan\Documents\GitHub\LAB-DAA\Q
DevanshRautela_C_20_2018314
enter no. of test
2
enter the size of array
6
enter the element of array
85 95 65 744 85 65
enter the key
65
65 found
number of comparision
4
enter the size of array
9
enter the element of array
65 85 95 67 85 97 85 35 62
enter the key
55
55 not found
number of comparision
12
```

PROGRAM 4

4. Given a sorted array of positive integers containing few duplicate elements, design an algorithm and implement it using a program to find whether the given key element is present in the array or not. If present, then also find the number of copies of given key. (Time Complexity = O(log n))

Algorithm:

```
Step 1: find fist occurance of key using  binary search
Step 2:find last occurance of key using binary search
Step 3: set freq=last occurance – first occurance +1
Step 4: if freq >1
            Print freq
        Else
            Print no duplicate
Step 5 exit
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θlog(n)

Best Case: T(n)= Ω(1)

Average Case:T(n) = θlog(n)

Space Complexity:

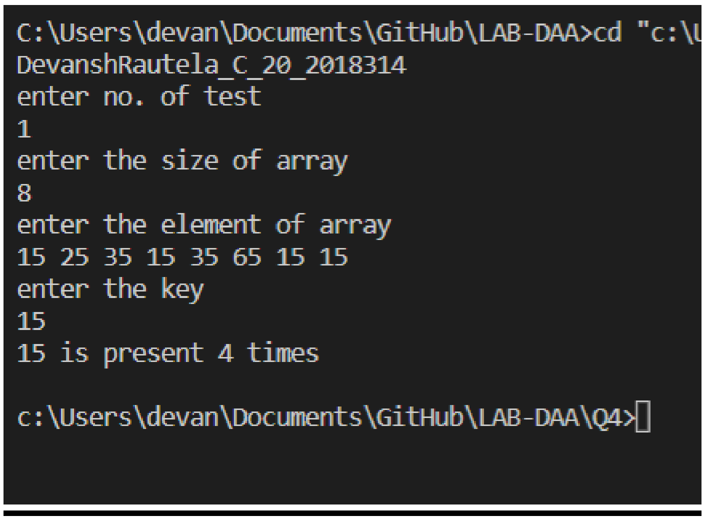
Worst Case=Best Case = Average case = θ(1) [Constant]

<u>Source code</u> :

```
#include <iostream>
using namespace std;
int first(int *arr,int l,int r,int key,int n)
{
    if(l<=r)
    {
        int mid1=l+(r-l)/2;
        if((mid1==0||arr[mid1-1]<key)&&arr[mid1]==key)
           return mid1;
        else if(arr[mid1]<key)
           return first(arr,mid1+1,r,key,n);
        else
          return first(arr,l,mid1-1,key,n);
    }
    return -1;
}
int last(int  *arr,int l,int r,int key,int n)
{
    if(l<=r)
    {
        int mid2=l+(r-l)/2;
        if((mid2==n-1||arr[mid2+1]>key)&&arr[mid2]==key)
           return mid2;
        else if(arr[mid2]>key)
           return last(arr,l,mid2-1,key,n);
        else
          return last(arr,mid2+1,r,key,n);
    }
    return -1;
}
```

```
int main(){
cout<<"DevanshRautela_C_20_2018314"<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0)
 {
    int n;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    cout<<"enter the element of array"<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
    int key;
    cout<<"enter the key"<<endl;
    cin>>key;
    int diff=last(arr,0,n-1,key,n)-first(arr,0,n-1,key,n)+1;
    if(diff>0)
    {
        cout<<key<<"-"<<diff<<endl;
    }
    else
       cout<<key<<" not found"<<endl;
    t--;
}
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
15 25 35 15 35 65 15 15
enter the key
15
15 is present 4 times

c:\Users\devan\Documents\GitHub\LAB-DAA\Q4>
```

PROGRAM 5

5. Given a sorted array of positive integers, design an algorithm and implement it using a program to find three indices i, j, k such that arr[i] + arr[j] = arr[k].

Algorithm:

```
Step 1: for i=0 to n-2
            for j=i+1 to n-1
             for k=j+1 to n
                if arr[i]+arr[j]==arr[k]
                print i j k
                 exit
Step 2: print not found
Step 3: exit
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ(n^3)

Best Case: T(n)= Ω(n^3)

Average Case:T(n) = θ(n^3)

Space Complexity:

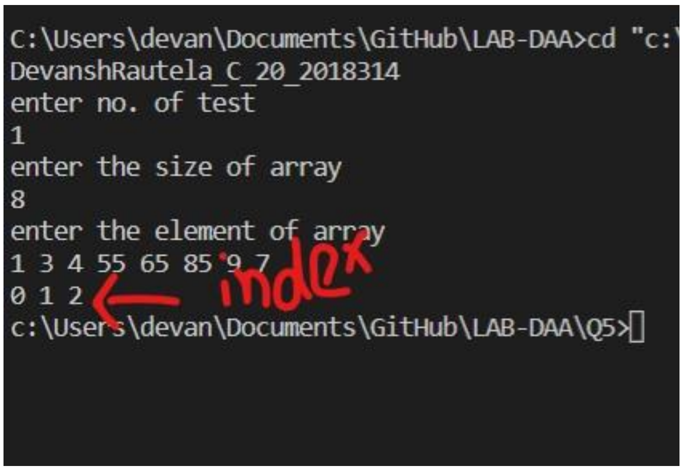
Worst Case=Best Case = Average case = θ(1) [Constant]

Source code:

```
#include <iostream>
using namespace std;
int main(){
cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0)
 {
    int n;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    cout<<"enter the element of array"<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
      int f=0;
    for(int i=0;i<n-2;i++)
    {
        for(int j=i+1;j<n-1;j++)
        {
            for(int k=j+1;k<n;k++)
            {
                if(arr[i]+arr[j]==arr[k])
                {
                    cout<<i<<" "<<j<<" "<<k;
                    f=1;
                    break;
                }
            }
            if(f==1)
```

```
              break;
         }
         if(f==1)
          break;
     }
     if(f==0)
      cout<<"not found";
      t--;
 }
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
1 3 4 55 65 85 9 7
0 1 2
c:\Users\devan\Documents\GitHub\LAB-DAA\Q5>
```

PROGRAM 6

6. Given an array of nonnegative integers, design an algorithm and a program to count the number of pairs of integers such that their difference is equal to a given key, K.

Algorithm:

```
Step 1: for i=0 to n-1
            for j=i+1 to n
              if arr[i]+arr[j]==key
                print arr[i] arr[j]
Step 2: print not such pair
Step 3 : exit
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ(n^2)

Best Case: T(n)= Ω(n^2)

Average Case:T(n) = θ(n^2)

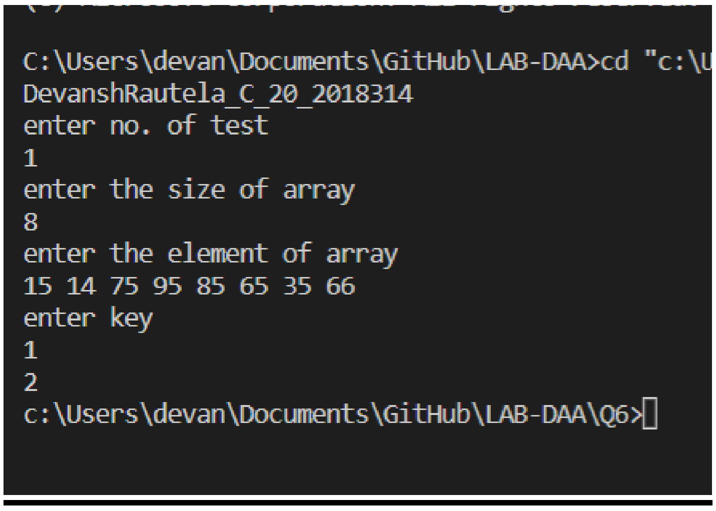
Space Complexity:

Worst Case=Best Case = Average case = θ(1) [Constant]

Source code :

```
#include <iostream>
using namespace std;
int main(){
cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0){
    int n;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    cout<<"enter the element of array"<<endl;
    int arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
      int f=0;
      int k;
      cout<<"enter key"<<endl;
      cin>>k;
      for(int i=0;i<n-1;i++)
      {
          for(int j=i+1;j<n;j++)
          {
              if(arr[i]-arr[j]==k||arr[j]-arr[i]==k)
               f++;
          }
      }
      cout<<f;
      t--;
 }
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\U
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
15 14 75 95 85 65 35 66
enter key
1
2
c:\Users\devan\Documents\GitHub\LAB-DAA\Q6>
```

PROGRAM 7

7. Given an unsorted array of integers, design an algorithm and a program to sort the array using insertion sort. Your program should be able to find number of comparisons and shifts (shifts - total number of times the array elements are shifted from their place) required for sorting the array.

Algorithm:

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a key.

Step3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ(n^2)

Best Case: T(n)= Ω(n)

Average Case:T(n) = θ(n^2)

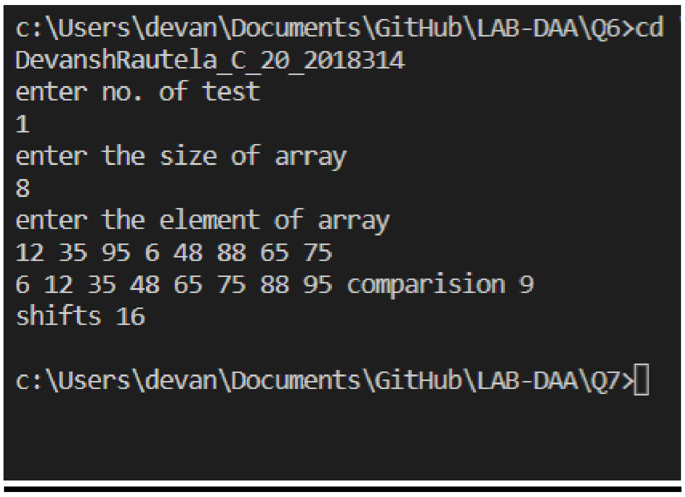
Space Complexity:

Worst Case=Best Case = Average case = θ(1) [Constant]

Source code :

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0)
 {
 int n;
 cout<<"enter the size of array"<<endl;
 cin>>n;
 int arr[n];
 int j;
 int c=0,s=0;
 cout<<"enter the element of array"<<endl;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
     int temp=arr[i];
     j=i-1;
     while(j>=0&&temp<arr[j])
     {
         arr[j+1]=arr[j];
         c++;
         s++;
         j--;
     }
     arr[j+1]=temp;
     s++;
 }
  for(int i=0;i<n;i++)
 {
     cout<<arr[i]<<" ";
 }
 cout<<"comparision "<<c<<endl;
 cout<<"shifts "<<s-1<<endl;
 t--;
 }
}
```

OUTPUT



```
c:\Users\devan\Documents\GitHub\LAB-DAA\Q6>cd
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
12 35 95 6 48 88 65 75
6 12 35 48 65 75 88 95 comparision 9
shifts 16

c:\Users\devan\Documents\GitHub\LAB-DAA\Q7>
```

PROGRAM 8

8. Given an unsorted array of integers, design an algorithm and implement a program to sort this array using selection sort. Your program should also find number of comparisons and number of swaps required.

Algorithm:

```
Step 1 − Set MIN to location 0
Step 2 − Search the minimum element in the list
Step 3 − Swap with value at location MIN
Step 4 − Increment MIN to point to next element
Step 5 − Repeat until list is sorted
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ(n^2)

Best Case: T(n)= Ω(n^2)

Average Case:T(n) = θ(n^2)

Space Complexity:

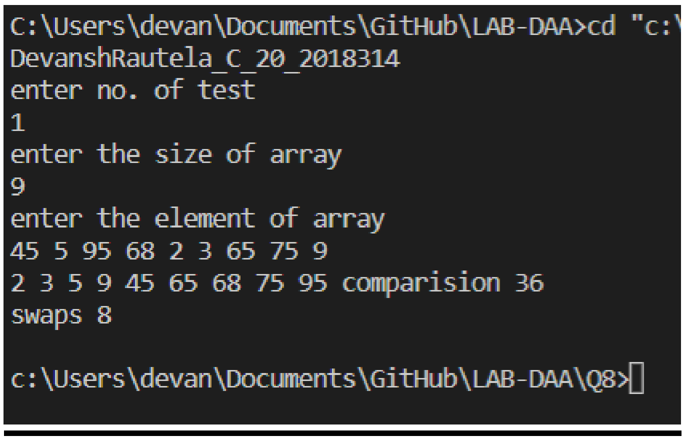
Worst Case=Best Case = Average case = θ(1) [Constant]

Source code :

```
#include <iostream>
#include <algorithm>
using namespace std;
int main(){
cout<<”DevanshRautela_C_20_2018314”<<endl;
 int t;
cout<<"enter no. of test"<<endl;
cin>>t;
 while(t!=0)
 {
 int n;
 cout<<"enter the size of array"<<endl;
 cin>>n;
 int arr[n];
 int j;
 int c=0,s=0;
 cout<<"enter the element of array"<<endl;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
 }
 for(int i=0;i<n;i++)
 {
     int k=i;
     for(int j=i+1;j<n;j++)
     {
         c++;
         if(arr[k]>arr[j])
           k=j;
     }
     s++;
```

```
    swap(arr[i],arr[k]);
 }
  for(int i=0;i<n;i++)
 {
    cout<<arr[i]<<" ";
 }
 cout<<"comparision "<<c<<endl;
 cout<<"swaps "<<s-1<<endl;
 t--;
 }
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
9
enter the element of array
45 5 95 68 2 3 65 75 9
2 3 5 9 45 65 68 75 95 comparision 36
swaps 8

c:\Users\devan\Documents\GitHub\LAB-DAA\Q8>
```

PROGRAM 9

9. Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements in the array or not. (use sorting) (Time Complexity = O(n log n))

Algorithm:

```
Step1: sort the array using merge sort
Step2:repeat for i=0 to n
        If array[i]==array[i+1] then
            Print duplicate found
            Jump to step 4
Step3: print no duplicate found
Step4:exit
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ($n$logn)

Best Case: T(n)= Ω(nlogn)

Average Case:T(n) = θ(nlogn)

Space Complexity:

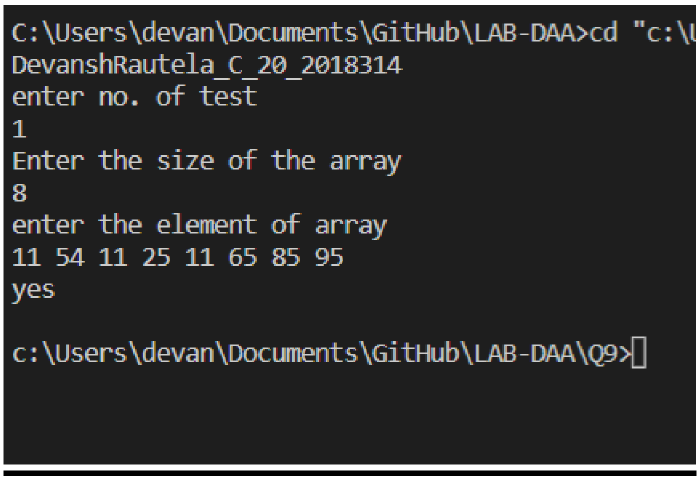
Worst Case=Best Case = Average case = θ(n) [Constant]

Source code :

```
#include <iostream>
#include <algorithm>
using namespace std;
void merge(int *arr,int l,int e,int mid)
{
    int i=0,j=0,k=l;
    int n1=mid-l+1;
    int n2=e-mid;
    int L[n1];
    int R[n2];
    for(int i=0;i<n1;i++)
    {
        L[i]=arr[l+i];
    }
    for(int i=0;i<n2;i++)
    {
        R[i]=arr[mid+1+i];
    }
    while(i<n1&&j<n2)
    {
        if(L[i]<=R[j])
        {
            arr[k]=L[i];
            i++;
        }
        else
        {
          arr[k]=R[j];
            j++;
        }
        k++;
    }
    while(i<n1)
    {
          arr[k]=L[i];
            i++;
        k++;
    }
    while(j<n2)
    {
          arr[k]=R[j];
            j++;
        k++;
    }

}
void mergesort(int *arr,int l,int e)
{
    if(l<e)
    {
        int mid=l+(e-l)/2;
        mergesort(arr,l,mid);
        mergesort(arr,mid+1,e);
        merge(arr,l,e,mid);
    }
}
int main(){
cout<<"DevanshRautela_C_20_2018314"<<endl;
 int t;
 cout<<"enter no. of test"<<endl;
 cin>>t;
 while(t!=0)
 {
```

```
 int n;


 cin>>n;
 int arr[n];
 int j;
 cout<<"enter the element of array"<<endl;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
 }
   mergesort(arr,0,n-1);
   int f=1;
  for(int i=0;i<n;i++)
 {
    if(arr[i]==arr[i+1])
    {
        cout<<"yes"<<endl;
        f=0;
        break;
    }
 }
 if(f==1)
 {
     cout<<"no"<<endl;
 }
 t--;
 }
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\
DevanshRautela_C_20_2018314
enter no. of test
1
Enter the size of the array
8
enter the element of array
11 54 11 25 11 65 85 95
yes

c:\Users\devan\Documents\GitHub\LAB-DAA\Q9>
```

PROGRAM 10

10. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array. (Merge sort)

<u>Algorithm:</u>

Step 1 − divide the list recursively into two halves until it can no more be divided.

```
MERGE_SORT(arr, beg, end)
    if beg < end
    set mid = (beg + end)/2
    MERGE_SORT(arr, beg, mid)
    MERGE_SORT(arr, mid + 1, end)
    MERGE (arr, beg, mid, end)
    end of if
END MERGE_SORT
```

Step 2 − merge the smaller lists into new list in sorted order.

```
while (i < n1 && j < n2)
    if (LeftArray[i] <= RightArray[j]) then
        a[k] = LeftArray[i];
        i++;
    else  then
        a[k] = RightArray[j];
        j++;    k++;
end of while
```

<u>Complexity Analysis:</u>

Time Complexity:

Worst Case: T(n) = θ($n$logn)

Best Case: T(n)= Ω(nlogn)

Average Case:T(n) = θ($n$logn)

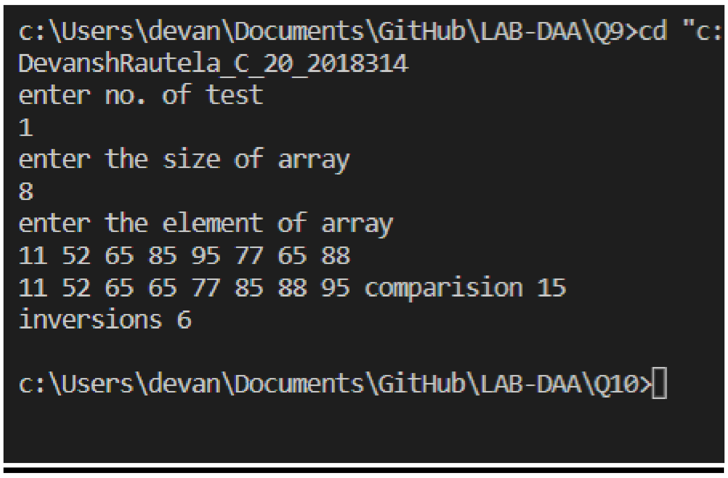
Space Complexity:

Worst Case=Best Case = Average case = θ(n)

Source code :

```
#include <bits/stdc++.h>
using namespace std;
int _mergeSort(int arr[], int temp[], int left, int right,int *c);
int merge(int arr[], int temp[], int left, int mid,int right,int *c);
int mergeSort(int arr[], int array_size,int *c)
{
    int temp[array_size];
    return _mergeSort(arr, temp, 0, array_size - 1,c);
}
int _mergeSort(int arr[], int temp[], int left, int right,int *c)
{
    int mid, inv_count = 0;
    if (right > left) {

        mid = (right + left) / 2;
        inv_count += _mergeSort(arr, temp, left, mid,c);
        inv_count += _mergeSort(arr, temp, mid + 1, right,c);
        inv_count += merge(arr, temp, left, mid + 1, right,c);
    }
    return inv_count;
}
int merge(int arr[], int temp[], int left, int mid,int right,int *c)
{
    int i, j, k;
    int inv_count = 0;

    i = left;
    j = mid;
    k = left;
    while ((i <= mid - 1) && (j <= right)) {
        *c=*c+1;
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        }
        else {
            temp[k++] = arr[j++];

            inv_count = inv_count + (mid - i);
        }
    }
    while (i <= mid - 1)
        temp[k++] = arr[i++];

    while (j <= right)
        temp[k++] = arr[j++];

    for (i = left; i <= right; i++)
        arr[i] = temp[i];

    return inv_count;
}
int main(){
 cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
 cout<<"enter no. of test"<<endl;
 cin>>t;
 while(t!=0)
 {
 int n;
 cout<<"enter the size of array"<<endl;
 cin>>n;
 int arr[n];
 int j;
 int c=0;
 cout<<"enter the element of array"<<endl;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
 }
    int s = mergeSort(arr, n,&c);
    for(int i=0;i<n;i++)
      cout<<arr[i]<<" ";
    cout<<"comparision "<<c<<endl;
    cout << "inversions " <<s<<endl;
  t--;
}
}
```

OUTPUT



```
c:\Users\devan\Documents\GitHub\LAB-DAA\Q9>cd "c:
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
11 52 65 85 95 77 65 88
11 52 65 65 77 85 88 95 comparision 15
inversions 6

c:\Users\devan\Documents\GitHub\LAB-DAA\Q10>
```

PROGRAM 11

11. Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array. (Randomised QuickSort)

Algorithm:

```
QUICKSORT (array A, start, end)
{
 1 if (start < end)
 2 {
3 p = partition(A, start, end)
4 QUICKSORT (A, start, p - 1)
5 QUICKSORT (A, p + 1, end)
6 }
}
PARTITION (array A, start, end)
{
 1 pivot ? A[end]
 2 i ? start-1
 3 for j ? start to end -1 {
 4 do if (A[j] < pivot) {
 5 then i ? i + 1
 6 swap A[i] with A[j]
 7  }}
 8 swap A[i+1] with A[end]
 9 return i+1
}
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ(n^2)

Best Case: T(n)= Ω(nlogn)

Average Case:T(n) = θ($n$logn)

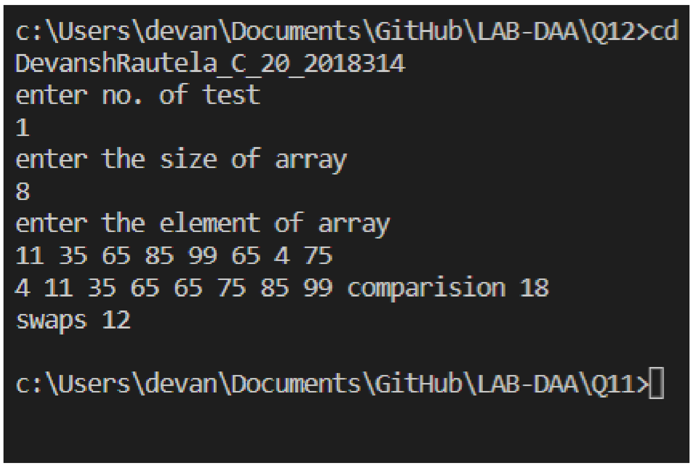
Space Complexity:

Worst Case=Best Case = Average case = θ(nlogn)

Source code :

```
#include<iostream>
#include<algorithm>
using namespace std;
int partition(int *arr,int l,int r,int *c,int *s){
   int piovt=arr[r];
   int i=l-1;
   for(int j=l;j<=r-1;j++)
   {
     (*c)++;
      if(arr[j]<=piovt)
      {
        i++;
        swap(arr[i],arr[j]);
        (*s)++;
      }
   }
   swap(arr[i+1],arr[r]);
   (*s)++;
   return (i+1);
}
void quicksort(int *arr,int l,int r,int *c,int *s)
{
if(l<r)
{
int p=partition(arr,l,r,c,s);
quicksort(arr,l,p-1,c,s);
quicksort(arr,p+1,r,c,s);
}
}
int main(){
 cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
 cout<<"enter no. of test"<<endl;
 cin>>t;
 while(t!=0)
 {
 int n;
 cout<<"enter the size of array"<<endl;
 cin>>n;
 int arr[n];
 int j;
 int s=0,c=0;
 cout<<"enter the element of array"<<endl;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
 }
   quicksort(arr,0,n-1,&c,&s);
  for(int i=0;i<n;i++)
 {
     cout<<arr[i]<<" ";
 }
 cout<<"comparision "<<c<<endl;
 cout<<"swaps "<<s<<endl;
 t--;
}
}
```

OUTPUT



```
c:\Users\devan\Documents\GitHub\LAB-DAA\Q12>cd
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
11 35 65 85 99 65 4 75
4 11 35 65 65 75 85 99 comparision 18
swaps 12

c:\Users\devan\Documents\GitHub\LAB-DAA\Q11>
```

PROGRAM 12

12. Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

Algorithm:

```
function quickSelect(list, left, right, k)
  if left = right
    return list[left]
  Select a pivotIndex between left and right
  pivotIndex := partition(list, left, right,pivotIndex)
  if k = pivotIndex
    return list[k]
  else if k < pivotIndex
    right := pivotIndex - 1
  else
    left := pivotIndex + 1
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ($n\log n$)

Best Case: T(n)= Ω(n)

Average Case:T(n) = θ(n^2)

Space Complexity:

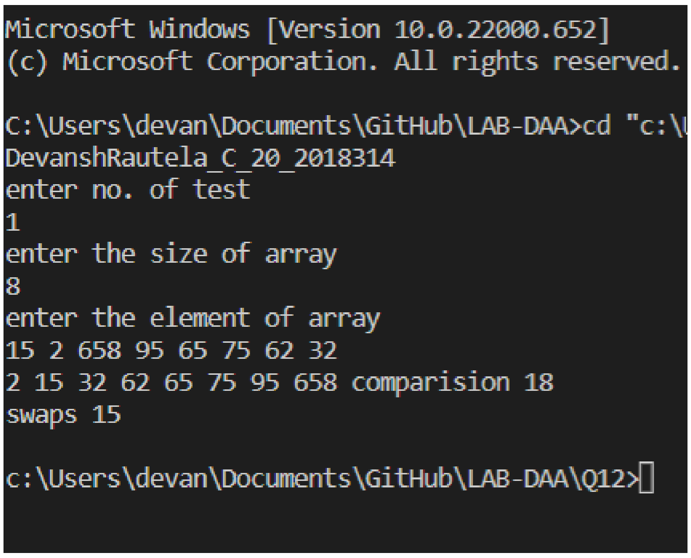
Worst Case= θ(n)

Best Case = Average case = θ(logn)

Source code:

```
#include<iostream>
#include<algorithm>
using namespace std;
int partition(int *arr,int l,int r)
{
    int pivot=arr[r];
    int i=l-1;
    for(int j=l;j<=r-1;j++)
    {
        if(arr[j]<pivot)
        {
            i++;
            swap(arr[j],arr[i]);

        }
    }
    swap(arr[i+1],arr[r]);
    return i+1;
}
int quickselect(int *arr,int l,int r,int k)
{
    if(k>0&&k<=r-l+1)
    {
        int p=partition(arr,l,r);
        if(p-l==k-1)
          return arr[p];
        if(p-l>k-1)
          return quickselect(arr,l,p-1,k);


          return quickselect(arr,p+1,r,k-p+l-1);
    }
return INT_MAX;
}
int main(){
 cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
 cout<<"enter no. of test"<<endl;
 cin>>t;
 while(t!=0)
 {
 int n;
 cout<<"enter the size of array"<<endl;
 cin>>n;
 int arr[n];
 int j;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
 }
 int k;
 cout<<"enter key"<<endl;
 cin>>k;
   int s=quickselect(arr,0,n-1,k);
   cout<<s<<endl;
   t--;
 }
}
```

OUTPUT



```
Microsoft Windows [Version 10.0.22000.652]
(c) Microsoft Corporation. All rights reserved.

C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
15 2 658 95 65 75 62 32
2 15 32 62 65 75 95 658 comparision 18
swaps 15

c:\Users\devan\Documents\GitHub\LAB-DAA\Q12>
```

PROGRAM 13

13. Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and print it. (Time Complexity = O(n)) (Hint: Use counting sort)

Algorithm:

```
countingSort(array, n)
max = find maximum element in the given array
create count array with size maximum + 1
Initialize count array with all 0's
for i = 0 to n
find the count of every unique element and
store that count at ith position in the count array
for j = 1 to max
Now, find the cumulative sum and store it in count array
for i = n to 1
Restore the array elements
Decrease the count of every restored element by 1
end countingSort
```

Complexity Analysis:

Time Complexity:

Worst Case: $T(n) = \theta(n+k)$

Best Case: $T(n)= \Omega(n+k)$

Average Case: $T(n) = \theta(n+k)$

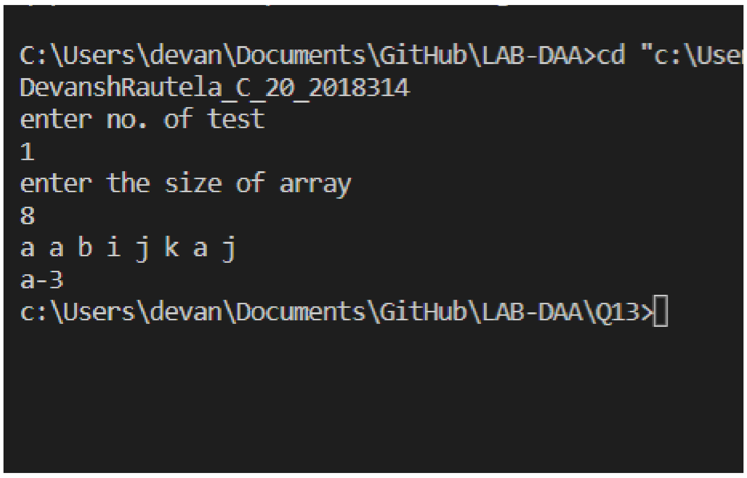
Space Complexity:

Worst Case=Best Case = Average case = $\theta(k)$

Source code :

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
    cout<<"enter no. of test"<<endl;
    cin>>t;
    while(t!=0)
    {
    int n;
    cout<<"enter the size of array"<<endl;
    cin>>n;
    char arr[n];
    for(int i=0;i<n;i++)
      cin>>arr[i];
    int count[26];
    memset(count,0,sizeof(count));
    for(int i=0;i<n;++i)
    {
        int k=arr[i];
        ++count[k-97];
    }
    int l=count[0],j=0;
    for(int i=1;i<26;i++)
    {
        if(l<count[i])
        {
          l=count[i];
          j=i;
        }
    }
    if(l>1)
    {
        char a='a'+j;
        cout<<a<<"-"<<l;
    }
    else
    {
      cout<<"no duplicate found"<<endl;
    }
    t--;
    }
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\Use
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
a a b i j k a j
a-3
c:\Users\devan\Documents\GitHub\LAB-DAA\Q13>
```

PROGRAM 14

14. Given an unsorted array of integers, design an algorithm and implement it using a program to find whether two elements exist such that their sum is equal to the given key element. (Time Complexity = O(n log n))

Algorithm:

Step 1 − divide the list recursively into two halves until it can no more be divided.

```
    MERGE_SORT(arr, beg, end)
      if beg < end
      set mid = (beg + end)/2
      MERGE_SORT(arr, beg, mid)
      MERGE_SORT(arr, mid + 1, end)
     MERGE (arr, beg, mid, end)
     end of if
  END MERGE_SORT
```

Step 2 − merge the smaller lists into new list in sorted order.

```
    while (i < n1 && j < n2)
        if (LeftArray[i] <= RightArray[j]) then
          a[k] = LeftArray[i];
          i++;
      else  then
        a[k] = RightArray[j];
        j++;   k++;
    end of while
```

Complexity Analysis:

Time Complexity:

Worst Case: T(n) = θ($n$logn)

Best Case: T(n)= Ω(nlogn)

Average Case:T(n) = θ($n$logn)

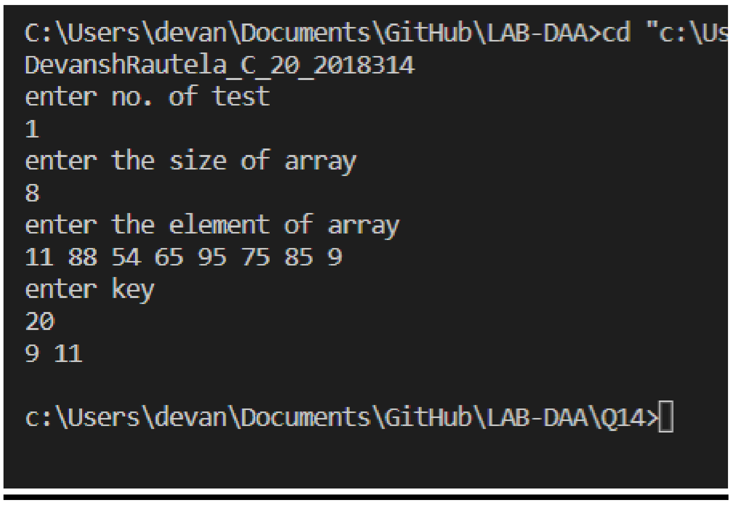
Space Complexity:

Worst Case=Best Case = Average case = θ(n)

Source Code :

```
#include <iostream>
#include <algorithm>
using namespace std;
void merge(int *arr,int l,int e,int mid)
{
    int i=0,j=0,k=l;
    int n1=mid-l+1;
    int n2=e-mid;
    int L[n1];
    int R[n2];
    for(int i=0;i<n1;i++)
    {
        L[i]=arr[l+i];
    }
    for(int i=0;i<n2;i++)
    {
        R[i]=arr[mid+1+i];
    }
    while(i<n1&&j<n2)
    {
        if(L[i]<=R[j])
        {
            arr[k]=L[i];
            i++;
        }
        else
        {
          arr[k]=R[j];
            j++;
        }
        k++;
    }
    while(i<n1)
    {
          arr[k]=L[i];
            i++;
        k++;
    }
    while(j<n2)
    {
          arr[k]=R[j];
            j++;
        k++;
    }
}
void mergesort(int *arr,int l,int e)
{
    if(l<e)
    {
        int mid=l+(e-l)/2;
        mergesort(arr,l,mid);
        mergesort(arr,mid+1,e);
        merge(arr,l,e,mid);
    }
}
int main(){
 cout<<”DevanshRautela_C_20_2018314”<<endl;
int t;
 cout<<"enter no. of test"<<endl;
 cin>>t;
 while(t!=0)
 {
 int n;
 cout<<"enter the size of array"<<endl;
 cin>>n;
 int arr[n];
 int j;
 cout<<"enter the element of array"<<endl;
 for(int i=0;i<n;i++)
 {
     cin>>arr[i];
 }
 int key;
 cout<<"enter key"<<endl;
 cin>>key;
  mergesort(arr,0,n-1);
  int l=0,r=n-1,f=0;
  while(l<r)
  {
      if(arr[l]+arr[r]==key)
      {
          cout<<arr[l]<<" "<<arr[r]<<endl;
          f=1;
          l++,r--;
      }
      else if(arr[l]+arr[r]<key)
        l++;
      else
        r--;
  }
  if(f==0)
    cout<<"no such pairs"<<endl;
  t--;
 }
}
```

OUTPUT



```
C:\Users\devan\Documents\GitHub\LAB-DAA>cd "c:\Us
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
11 88 54 65 95 75 85 9
enter key
20
9 11

c:\Users\devan\Documents\GitHub\LAB-DAA\Q14>
```

PROGRAM 15

15. You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find list of elements which are common to both. (Time Complexity = O(m+n))

<u>Algorithm:</u>

```
Function intersection(arr1,arr2,m,n)

Step 1 set i=0,j=0

Step 2  while(i<m&&j<n)
        {
            if(arr1[i]==arr2[j]&&(arr1[i-1]!=arr1[i]||arr2[j-1]!=arr2[j])) then
            {
              cout<<arr1[i]<<" ";
              i++;
              j++;
            }
            else if(arr1[i]>arr2[j])
              j++;
            else
              i++;
        }
Step 3 exit
```

<u>Complexity Analysis:</u>

Time Complexity:

Worst Case: T(n) = θ(n) [Linear]

Best Case: T(n)= Ω(1) [Constant]

Average Case:T(n) = θ(n) [Linear]

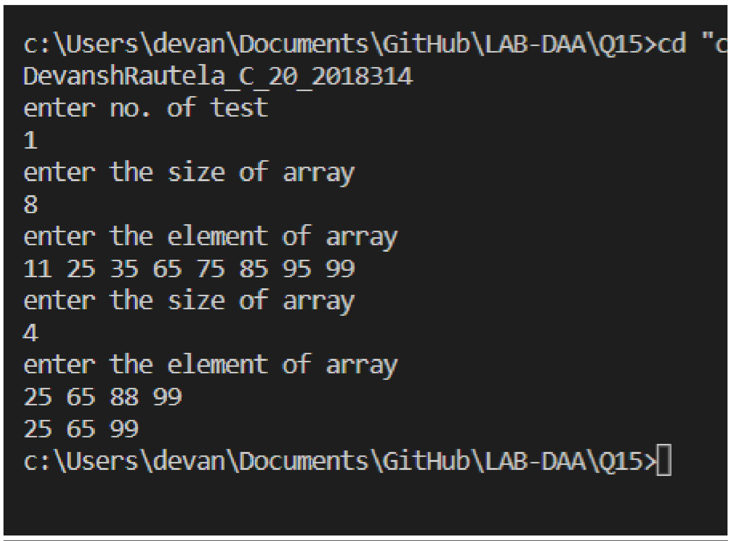
Space Complexity:

Worst Case=Best Case = Average case = θ(1) [Constant]

Source code :

```
#include<iostream>
using namespace std;
int main(){
    cout<<"DevanshRautela_C_20_2018314"<<endl;
    int t;
    cout<<"enter no. of test"<<endl;
    cin>>t;
    while(t!=0)
    {
        int m;
        cout<<"enter the size of array"<<endl;
        cin>>m;
        int arr1[m];
        cout<<"enter the element of array"<<endl;
        for(int i=0;i<m;i++)
        {
            cin>>arr1[i];
        }
        int n;
        cout<<"enter the size of array"<<endl;
        cin>>n;
        int arr2[n];
        cout<<"enter the element of array"<<endl;
        for(int i=0;i<n;i++)
        {
            cin>>arr2[i];
        }
        int i=0,j=0;
        while(i<m&&j<n)
        {
            if(arr1[i]==arr2[j]&&(arr1[i-1]!=arr1[i]||arr2[j-1]!=arr2[j]))
            {
              cout<<arr1[i]<<" ";
              i++;
              j++;
            }
            else if(arr1[i]>arr2[j])
              j++;
            else
              i++;
        }
        t--;
    }
}
```

OUTPUT



```
c:\Users\devan\Documents\GitHub\LAB-DAA\Q15>cd "c
DevanshRautela_C_20_2018314
enter no. of test
1
enter the size of array
8
enter the element of array
11 25 35 65 75 85 95 99
enter the size of array
4
enter the element of array
25 65 88 99
25 65 99
c:\Users\devan\Documents\GitHub\LAB-DAA\Q15>
```