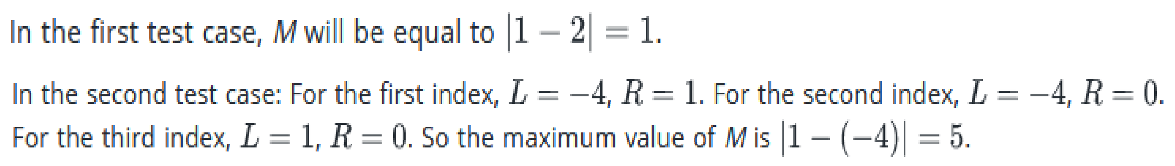
QUESTION:

Zulu is good in maths . He loves to play with numbers. One day while browsing through a book, he encountered a nice problem. In the problem, he was given an array *A* of *N* numbers. For each index *i* in the array we define two quantities. Let *r* be the maximum index such that r>=i and sub-array from *i* to *r* (inclusive) is either non-decreasing or non-increasing. Similarly, let *l* be the minimum index such that l<=i and sub-array from *l* to *i* (inclusive) is either non-decreasing or non-increasing. Now, we define points of an index *i* to be equal to max(|Ai-Al|,|Ai-Ar|). Note that *l* and *r* can be different for each index *i*. The task of the problem is to find the index of the array *A* which has the maximum points. Since the problem seems a bit harder, Zulu is struck. Can you solve this problem for Zulu?
Input: In the first line of input, TEST will be given which is the number of test cases. For each of the test case, a single integer *N* will be given in the first line. In the second line, *N* space separated integers will be provided.
Output:
For each of the test cases, output the points of the index which have maximum points in the array in a separate line.
SAMPLE INPUT
2
2
1 2
3
-4 1 0
SAMPLE OUTPUT
1
5
Explanation


In the first test case, *M* will be equal to $|1-2|=1$.

In the second test case: For the first index, $L=-4$, $R=1$. For the second index, $L=-4$, $R=0$. For the third index, $L=1$, $R=0$. So the maximum value of *M* is $|1-(-4)|=5$.

EX NO:1.1(a)	**FINDING THE INDEX**
DATE:15.11.2018	

AIM:

To write a program to find the index of the array *A* which has the maximum points. Since the problem seems a bit harder, Zulu is struck.

PSEUDOCODE:

```
//Program: To find the index of the array
//Input: Array
//Output: Index of the element with maximum points
For i←0 to i<n1
  r ←i
  l ←i
  If(i>=0)
          k←i-1
   If(arr[i]<arr[k])
               while(arr[i]<arr[k]&&k>=0)
                            k--
                           k++
   Else
              While(arr[i]>arr[k]&&k>=0)
                            k--
                           k++
   l ←k
  If(i<=n1-1)
              k=i+1
   If(arr[i]<arr[k])
               While(arr[i]<arr[k]&&k<=n1-1)
                            k++
               k--
   Else
               While(arr[i]>arr[k]&&k<=n1-1)
                            k++
               k--
 r ←k
 max2=max(abs(arr[i]-arr[l]),abs(arr[i]-arr[r]))
 If(max2>max1)
              max1=max2
Print max1
```

SOURCE CODE:

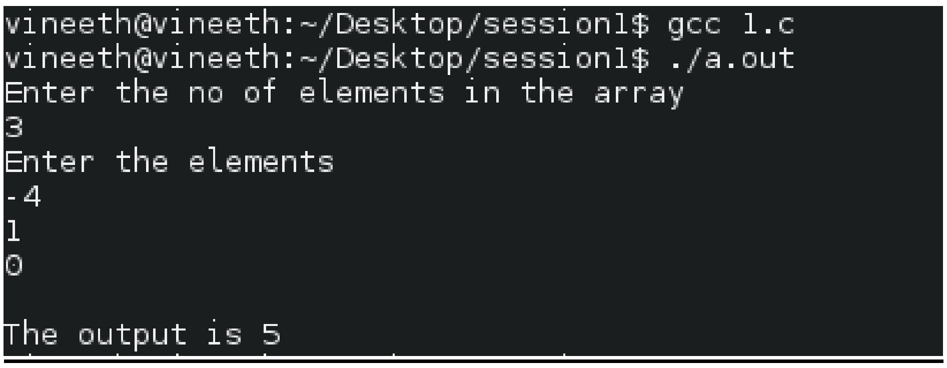
```
        #include<stdio.h>
        int max(int a,int b)
        {
                return((a<b)?b:a);
        }
        void main()
        {
                int n1,i,j,k,l,r,max1=0,max2=0;
                printf("Enter the no of elements in the array\n");
                scanf("%d",&n1);
                int arr[n1];
                 printf("Enter the elements\n");
                 for(i=0;i<n1;i++)
                         scanf("%d",&arr[i]);
                for(i=0;i<n1;i++)
                 {
                         r=i;
                         l=i;
                         if(i>=0)
                         {
                                k=i-1;
                                  if(arr[i]<arr[k])
                                  {
                                          while(arr[i]<arr[k]&&k>=0)
                                                  k--;
                                          k++;
                                  }
                                  else
                                  {
                                          while(arr[i]>arr[k]&&k>=0)
                                                  k--;
                                          k++;
                                  }
                        }
                        l=k;
                          if(i<=n1-1)
                          {
                                  k=i+1;
                                  if(arr[i]<arr[k])
                                  {
                                          while(arr[i]<arr[k]&&k<=n1-1)
                                                  k++;
                                          k--;
                                  }
                                  else
                                  {
```

```
                                    while(arr[i]>arr[k]&&k<=n1-1)
                                              k++;
                                    k--;
                          }
                 }
                r=k;
                max2=max(abs(arr[i]-arr[l]),abs(arr[i]-arr[r]));
                if(max2>max1)
                          max1=max2;
               printf("\n");
        }
        printf("\nThe output is %d\n",max1);

}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n)

OUTPUT:



```
vineeth@vineeth:~/Desktop/session1$ gcc 1.c
vineeth@vineeth:~/Desktop/session1$ ./a.out
Enter the no of elements in the array
3
Enter the elements
-4
1
0

The output is 5
```

RESULT:

Thus a program to find the index of the array *A* which has the maximum points was successfully executed and verified.

QUESTION:

Write a C program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.

- Write a Brute force approach and state its complexity
- Suggest an alternate solution that shows the improvement in the running time.

EX NO:1.1(b)	FINDING THE EXACT ELEMENTS IN THE ARRAY
DATE:15.11.2018	

AIM:

To Write a C program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x.

PSEUDOCODE-1(Brute force):

```
//Program: To check the existence of two elements.
//Input: Array
//Output: Elements exist or not.
 Find(n , a[], x)
        For i←0 to i<n-1
                 For j←i+1 to j<n
                          If((a[i]+a[j])==x)
                                   return 1
        return 0
```

PSEUDOCODE-2(O(n)):

```
//Program: To check the existence of two elements.
//Input: Array
//Output: Elements exist or not.
For i←0 to i<n+1
        If(h[i]>0&&(tar-i)<n+1&&h[tar-i]>0)
                If(i!=(tar-i))
                        Print(tar-i,tar)
                        break
                Else
                        If(h[i]>1)
                                print(i,tar-i,tar)
                h[tar-i]-=1
```

SOURCE CODE-1(Brute force):

```
#include<stdio.h>
int i,j;
int find(int n,int a[],int x)
{
        for(i=0;i<n-1;i++)
        {
                for(j=i+1;j<n;j++)
                {
                        if((a[i]+a[j])==x)
                        {
                                return 1;
                        }
                }
        }
```

```
        return 0;
    }
    void main()
    {
         int n;
         printf("Enter the number of terms : ");
         scanf("%d",&n);
         int a[n];
         printf("Enter the array elements : ");
         for(i=0;i<n;i++)
              scanf("%d",&a[i]);
         int x;
         printf("Enter the element :\n");
         scanf("%d",&x);
         if(find(n,a,x))
         {
              printf("The sum of %d and %d in the array results in the given element
             %d\n",a[i],a[j],x);
         }
         else
         {
              printf("The sum of the elements in the array doesn't leads to the given
             elements\n");
         }
    }
```

SOURCE CODE-2(O(n)):

```
    #include<stdio.h>
    void main()
    {
        int i,n,tar;
        printf("Enter the size of the array\n");
        scanf("%d",&n);
        int h[n+1],arr[n];
        for(i=0;i<n+1;i++)
            h[i]=0;
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        {
            scanf("%d",&arr[i]);
            h[arr[i]]+=1;
        }
        printf("Enter the target element\n");
        scanf("%d",&tar);
        for(i=0;i<n+1;i++)
        {
            if(h[i]>0&&(tar-i)<n+1&&h[tar-i]>0)
            {
```

```
                    if(i!=(tar-i))
                         printf("%d and %d are the elements whose sum is %d\n",i,tar-
                         i,tar);
                    else
                    {
                         if(h[i]>1)
                         {
                              printf("%d and %d are the elements whose sum is
                         %d\n",i,tar-i,tar);
                              break;
                         }

                    }
                    h[tar-i]-=1;
               }
          }
     }
```

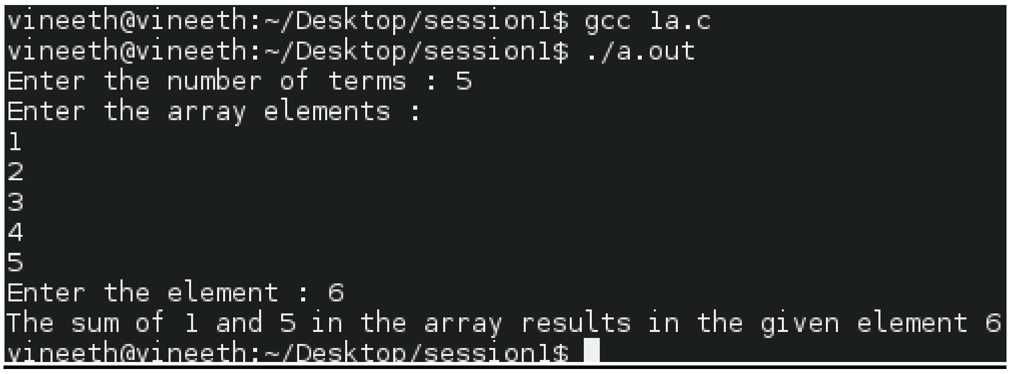
DATA STRUCTURE USED-1(Brute force): Array,Hashing

TIME COMPLEXITY: O(n^2)

DATA STRUCTURE USED-2(O(n)): Array

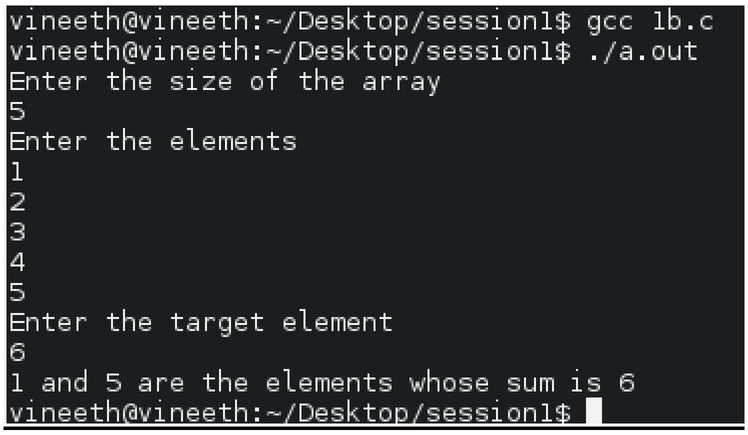
TIME COMPLEXITY: O(n)

OUTPUT-1(Brute force):



```
vineeth@vineeth:~/Desktop/session1$ gcc 1a.c
vineeth@vineeth:~/Desktop/session1$ ./a.out
Enter the number of terms : 5
Enter the array elements :
1
2
3
4
5
Enter the element : 6
The sum of 1 and 5 in the array results in the given element 6
vineeth@vineeth:~/Desktop/session1$
```

OUTPUT-2(O(n)):



```
vineeth@vineeth:~/Desktop/session1$ gcc 1b.c
vineeth@vineeth:~/Desktop/session1$ ./a.out
Enter the size of the array
5
Enter the elements
1
2
3
4
5
Enter the target element
6
1 and 5 are the elements whose sum is 6
vineeth@vineeth:~/Desktop/session1$ 
```

RESULT:

Thus a C program that, given an array A[] of n numbers and another number x, determines whether or not there exist two elements in S whose sum is exactly x was successfully executed and verified.

QUESTION:

Sheldon loves trains. One day he decided to take a trip. He has a map of the train routes. He wants to find the total travel duration between any two given stations. A train traveling from station a to b will stop at all intermediate stations for a particular amount of time. The map has the following information:

- The travel times between two consecutive stations.
- The waiting times for all intermediate stations.
- He wants to find the total travel duration between any two stations subject to the following 2 conditions:

1. Considering the waiting times of all intermediate stations without any changes (Condition - a)
2. Considering the waiting times of all intermediate stations to be the smallest of all waiting times of stations between the starting station and the ending station (Condition - b)

See Explanation Section for better understanding.

Input

- First line has an integer **T**, the number of test cases. **T** test cases follow.
- First line of each test case has **N**, the number of stations.
- Second line of each test case has **time$_i$**, the time for traveling from station$_i$ to station$_{i+1}$.
- Third line of each test case has **wait$_i$**, the waiting times for the intermediate stations between the first and last stations
- Fourth line has **Q**, the number of queries, **Q** lines follow
- Each query has **K** , **A** and **B** - the type of query, the starting and ending stations, respectively.

Output

- For each query, print one line.
- If query type is 0, the line should contain the total duration by applying condition-a.
- If query type is 1, the line should contain the total duration by applying condition-b.

Constraints

All values can be stored in a 32-bit integer variable.

Example

Input:

```
1
6
20 30 40 50 60
2 3 4 5
4
0 2 5
1 2 5
0 6 1
1 6 1
```

Output:

```
127
126
214
208
```

EX NO:1.2(A)	FINDING THE TIME INTERVAL
DATE:22.11.2018	

AIM:

To Write a C program to find the total travel duration between any two given stations.

PSEUDOCODE:

```
//Program: To find the time interval
//Input: Array
//Output: time interval
For  j←y-1 to j<z-1
        sum+←stn[j];
        If(x==0)
                For(j=y;j<z-1;j++)
                        sum+←ti[j]
        Else
                min=ti[y]
                For j←y to j<z-1
                        If(min>ti[j])
                                min=ti[k]
                                sum+=2*min
        out[k++]=sum
For i←0 to i<q
        Print out[i]
```

SOURCE CODE:

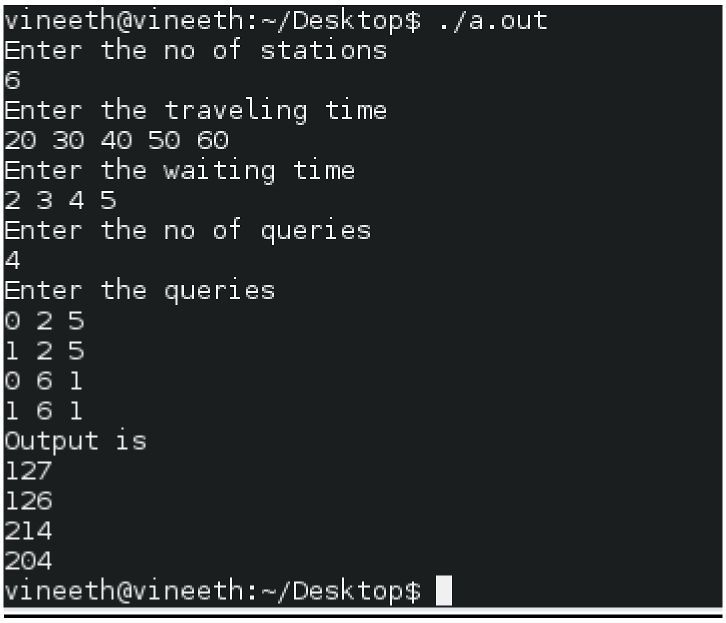
```
#include<stdio.h>
void main()
{
        int i,j,n,q,x,y,z,sum,min,t;
        printf("Enter the no of stations\n");
        scanf("%d",&n);
        int stn[n-1],ti[n];
        printf("Enter the traveling time\n");
        for(i=0;i<n-1;i++)
                scanf("%d",&stn[i]);
        ti[0]=0;
        printf("Enter the waiting time \n");
        for(i=1;i<n-1;i++)
                scanf("%d",&ti[i]);
        ti[n-1]=0;
        printf("Enter the no of queries\n");
        scanf("%d",&q);
        int out[q],k=0;
        printf("Enter the queries\n");
```

```
        for(i=0;i<q;i++)
        {
                sum=0;
                scanf("%d",&x);
                scanf("%d",&y);
                scanf("%d",&z);
                if(z<y)
                {
                        t=z;
                        z=y;
                        y=t;
                }
                for(j=y-1;j<z-1;j++)
                        sum+=stn[j];
                if(x==0)
                {
                        for(j=y;j<z-1;j++)
                                sum+=ti[j];
                }
                else
                {
                        min=ti[y];
                        for(j=y;j<z-1;j++)
                        {
                                if(min>ti[j])
                                        min=ti[k];
                        }
                        sum+=2*min;
                }
                out[k++]=sum;
        }
        printf("Output is \n");
        for(i=0;i<q;i++)
                printf("%d\n",out[i]);
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n^2)

OUTPUT:



```
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of stations
6
Enter the traveling time
20 30 40 50 60
Enter the waiting time
2 3 4 5
Enter the no of queries
4
Enter the queries
0 2 5
1 2 5
0 6 1
1 6 1
Output is
127
126
214
204
vineeth@vineeth:~/Desktop$ 
```

RESULT:

Thus a C program to find the total travel duration between any two given stations was successfully executed and verified.

QUESTION:

Given a matrix where every row is sorted in increasing order. Write an efficient algorithm that finds and returns a common element in all rows. If there is no common element, then returns -1.

```
Input: mat[4][5] = { {1, 2, 3, 4, 5},
                     {2, 4, 5, 8, 10},
                     {3, 5, 7, 9, 11},
                     {1, 3, 5, 7, 9},
                   };
Output: 5
```

EX NO:1.2(B)	FINDING THE COMMON ELEMENT
DATE:22.11.2018	

AIM:

To Write a C program to find the common element in the rows of the matrix.

PSEUDOCODE:

```
//Program: To find the common elements in the matrix
//Input: Array
//Output: xommon element
For i←0 to i<n
    flag[0]←1
    sum←0
    For j←1 to j<m
        flag[j]←0
    For k←0 to k<n
        If(arr[0][i]==arr[j][k])
            flag[j]←1
    For j←0;j<m;j++)
        sum+←flag[j]
        If(sum==m)
            Print arr[0][i]
```

SOURCE CODE:

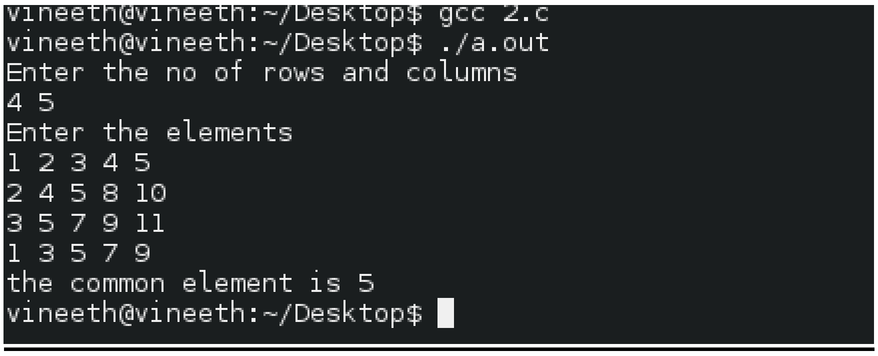
```
#include<stdio.h>
void main()
{
    int m,n,i,j,k;
    printf("Enter the no of rows and columns\n");
    scanf("%d%d",&m,&n);
    int arr[m][n];
    printf("Enter the elements\n");
    for(i=0;i<m;i++)
    {
        for(j=0;j<n;j++)
            scanf("%d",&arr[i][j]);
    }
    int flag[m],sum;
    for(i=0;i<n;i++)
    {
        flag[0]=1;
        sum=0;
        for(j=1;j<m;j++)
        {
            flag[j]=0;
```

```
                        for(k=0;k<n;k++)
                        {
                                if(arr[0][i]==arr[j][k])
                                        flag[j]=1;
                        }
                }
                for(j=0;j<m;j++)
                        sum+=flag[j];
                if(sum==m)
                {
                        printf("the common element is ");
                        printf("%d\n",arr[0][i]);
                }
        }
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n^2)

OUTPUT:



```
vineeth@vineeth:~/Desktop$ gcc 2.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of rows and columns
4 5
Enter the elements
1 2 3 4 5
2 4 5 8 10
3 5 7 9 11
1 3 5 7 9
the common element is 5
vineeth@vineeth:~/Desktop$ 
```

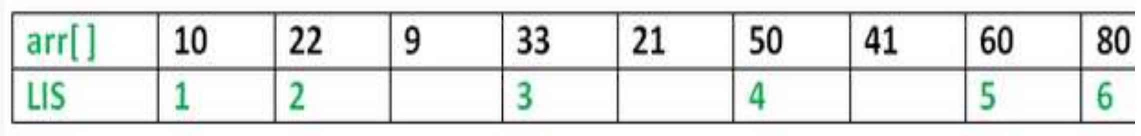
RESULT:

Thus a C program to find the common element in the rows of the matrix was successfully executed and verified.

QUESTION:

Longest Increasing Subsequence

The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.



arr[ ]	10	22	9	33	21	50	41	60	80
LIS	1	2		3		4		5	6

More Examples:
Input : arr[] = {3, 10, 2, 1, 20}
Output : Length of LIS = 3
The longest increasing subsequence is 3, 10, 20
Input : arr[] = {3, 2}
Output : Length of LIS = 1
The longest increasing subsequences are {3} and {2}
Input : arr[] = {50, 3, 10, 7, 40, 80}
Output : Length of LIS = 4
The longest increasing subsequence is {3, 7, 40, 80}

EX NO:1.3(A)	LONGEST INCREASING ELEMENT
DATE:29.11.2018	

AIM:

To Write a C program to find the count longest increasing element.

PSEUDOCODE:

```
//Program: To find the count of longest increasing element
//Input: Array
//Output:count
For i←0 to i<n-1
    c←1
    k←a[i]
For j←i+1 to j<n
    if(a[j]>k)
        c+=1
        k←a[j]
If(max<c)
    max=c
Print max
```

SOURCE CODE:

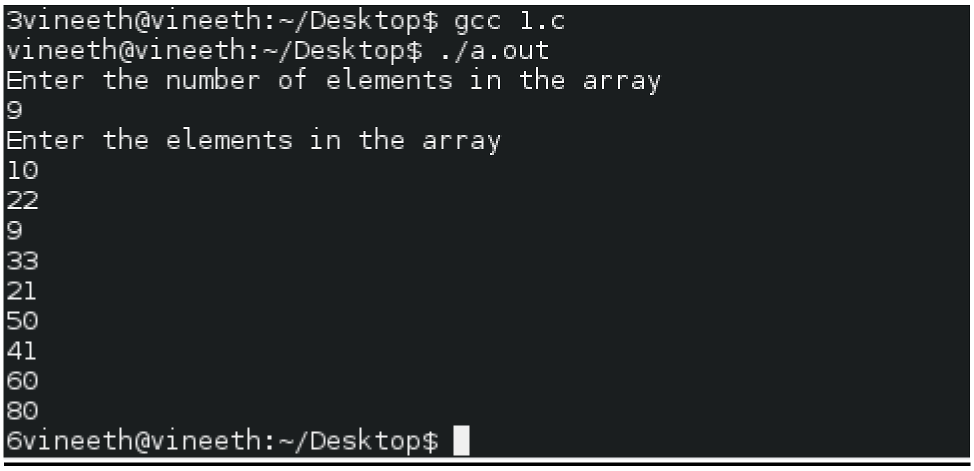
```
#include<stdio.h>
void main()
{
    int i,j,max=0,n,c,k;
    printf("Enter the number of elements in the array\n");
    scanf("%d",&n);
    int a[n];
    printf("Enter the elements in the array\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    for(i=0;i<n-1;i++)
    {
        c=1;
        k=a[i];
        for(j=i+1;j<n;j++)
        {
            if(a[j]>k)
            {
                c+=1;
                k=a[j];
            }
        }
        if(max<c)
            max=c;
```

```
            }
            printf("%d",max);
      }
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: $O(n^2)$

OUTPUT:



```
3vineeth@vineeth:~/Desktop$ gcc l.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the number of elements in the array
9
Enter the elements in the array
10
22
9
33
21
50
41
60
80
6vineeth@vineeth:~/Desktop$ 
```

RESULT:

Thus a C program to find the count longest increasing element was successfully executed and verified.

QUESTION:

Raju always like to collect coins. Raju has already managed to collect **n** different types of coins **a1**,**a2**....**an** . One day he went into the sale and find out that the sale consists of **109** types of coins where **i-th** type of coin costs **i** dollars. He decided to buy some more different types of coins given that he does not have that type of coin already. But he has only **k** dollars to spend. So help him to choose the type of coins.

Input:

The first line will contain test cases T. Then T Test cases follow**.** Each of the test cases will contain two integers **n, k** the number of types of coins that Raju already has and the money he has respectively. The next line contains **n** distinct integers **a1**,**a2**...**an** the types of coins that Raju already has. **Output:**

Print a single integer denoting the number of different types of coins that Raju can buy so that the number of different coins in his collection is **maximum**. Cost should not exceed **k**.

Input

```
 2
 4 14
 4 6 12 8
 3 7
1 3 4
```

Output

```
 4
 2
```

Explanation: For the second test case Raju should buy two coins: one coin of the 2-nd type and one coin of the 5-th type. For any other purchase for 7 dollars(assuming that the coins of types 1, 3 and 4 have already been bought), it is impossible to buy more than two coins.

EX NO:1.3(B)	TYPE OF COINS
DATE:29.11.2018	

AIM:

To Write a C program to find type of coins.

PSEUDOCODE:

```
//Program: To find the type of coins.
//Input: Array
//Output: number of types
For i←0 to i<n
        t←a[i]
For j←i-1 to j>=0 and t<a[j]
         a[j+1]←a[j]
        a[j+1]←t
For i←1 to i<k
        If(i!=a[i-1])
                  if(i<k)
                           k-=i
                           c++

                  else
                        break
Print c
```

SOURCE CODE:

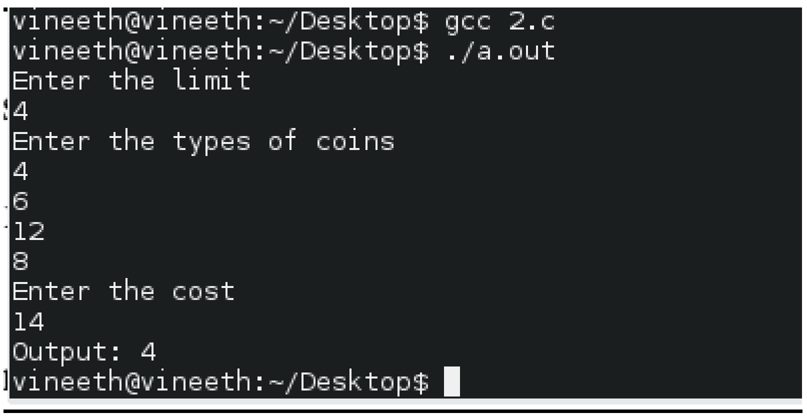
```
#include<stdio.h>
void main()
{
        int n,i,j,k,t=0,c=0;
        printf("Enter the limit\n");
        scanf("%d",&n);
        int a[n];
        printf("Enter the types of coins\n");
        for(i=0;i<n;i++)
                 scanf("%d",&a[i]);
        printf("Enter the cost\n");
        scanf("%d",&k);
        for(i=0;i<n;i++)
        {
                t=a[i];
                for(j=i-1;j>=0&&t<a[j];j--)
                         a[j+1]=a[j];
                a[j+1]=t;
        }
        for(i=1;i<k;i++)
```

```
            {
                    if(i!=a[i-1])
                      {
                            if(i<k)
                             {
                                    k-=i;
                                    c++;
                             }
                            else
                                    break;
                      }
            }
            printf("Output: %d\n",c);
    }
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n)

OUTPUT:



```
vineeth@vineeth:~/Desktop$ gcc 2.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the limit
4
Enter the types of coins
4
6
12
8
Enter the cost
14
Output: 4
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus a C program to find the count of type of coins was successfully executed and verified.

QUESTION:

Once Monk was watching a fight between an array and a tree, of being better. Tree got frustrated and converted that array into a Binary Search Tree by inserting the elements as nodes in BST, processing elements in the given order in the array. Now Monk wants to know the height of the created Binary Search Tree. Help Monk for the same.

Note:

1) In Binary Search Tree, the left sub-tree contains only nodes with values less than or equal to the parent node; the right sub-tree contains only nodes with values greater than the parent node.

2) Binary Search Tree with one node, has height equal to 1.

Input Format :

The first line will consist of *1* integer *N*, denoting the number of elements in the array. In next line, there will be *N* space separated integers, A[i] where 1≤i≤N, denoting the elements of array.

Output Format: Printthe height of the created Binary Search Tree.

SAMPLE INPUT	**SAMPLE OUTPUT**
4 2 1 3 4	3

Explanation

N=4.
Insert *2*. It becomes root of the tree.
Insert *1*. It becomes left child of *2*.
Insert *3*. It becomes right child of *2*.
Insert *4*. It becomes right child of *3*.
Final height of tree = *3*.

Source: hackerearth

EX NO:2.1(A)	**HEIGHT OF TREE**
DATE:06.12.2018	

AIM:

To Write a C program to find height of the tree.

PSEUDOCODE:

```
//Program: To find the height of the tree.
//Input: Node values
//Output: Height
For i←0 to i<n
        t←a[i]
For j←i-1 to j>=0 and t<a[j]
         a[j+1]←a[j]
        a[j+1]←t
For i←1 to i<k
        If(i!=a[i-1])
                  if(i<k)
                           k-=i
                           c++

                  else
                        break
Print c
```

SOURCE CODE:

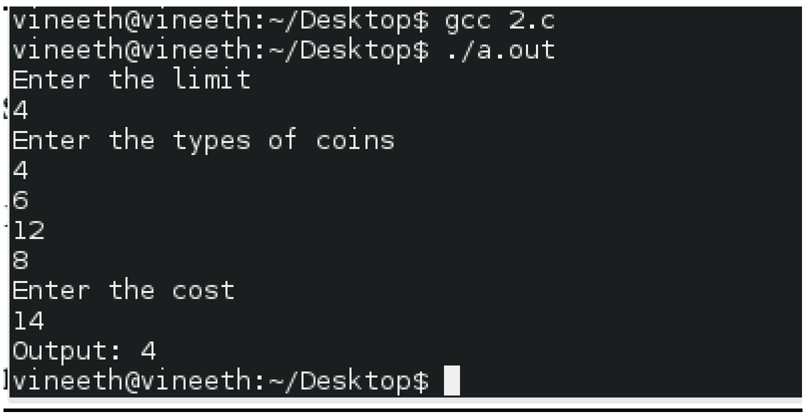
```
#include<stdio.h>
void main()
{
        int n,i,j,k,t=0,c=0;
        printf("Enter the limit\n");
        scanf("%d",&n);
        int a[n];
        printf("Enter the types of coins\n");
        for(i=0;i<n;i++)
                 scanf("%d",&a[i]);
        printf("Enter the cost\n");
        scanf("%d",&k);
        for(i=0;i<n;i++)
        {
                t=a[i];
                for(j=i-1;j>=0&&t<a[j];j--)
                         a[j+1]=a[j];
                a[j+1]=t;
        }
        for(i=1;i<k;i++)
```

```
        {
                if(i!=a[i-1])
                 {
                        if(i<k)
                         {
                                k-=i;
                                c++;
                         }
                        else
                                break;
                 }
        }
        printf("Output: %d\n",c);
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n)

OUTPUT:

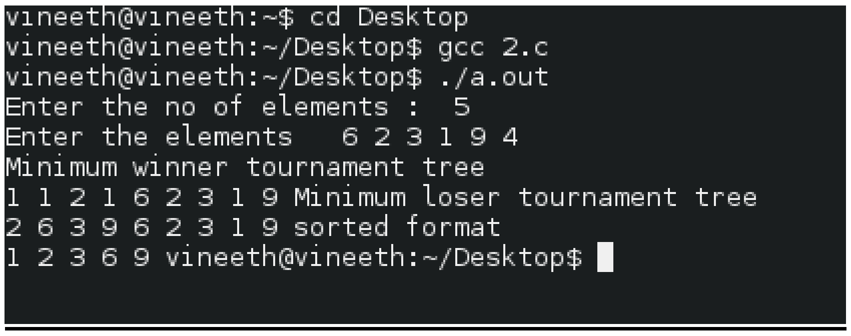


```
vineeth@vineeth:~/Desktop$ gcc 2.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the limit
4
Enter the types of coins
4
6
12
8
Enter the cost
14
Output: 4
vineeth@vineeth:~/Desktop$ 
```

RESULT:

Thus a C program to find the count of type of coins was successfully executed and verified.

OUTPUT:



```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 2.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of elements :  5
Enter the elements   6 2 3 1 9 4
Minimum winner tournament tree
1 1 2 1 6 2 3 1 9 Minimum loser tournament tree
2 6 3 9 6 2 3 1 9 sorted format
1 2 3 6 9 vineeth@vineeth:~/Desktop$ 
```

RESULT:

Thus the C program to implement tournament tree was successfully executed and verified.

QUESTION:

You have to sort 'n' numbers using heapsort.
Input Format:
First line contains 't'(t<=10) which denotes the number of testcases.
Then follow 't' lines, each having two lines .
First line of a testcase has 'n'(n<=100000) which is the number of elements in the array.
Second line are 'n' numbers (a[i]<=10^9), elements of the array.
Output Format
'T' lines each denoting 'n' numbers each denoting the sorted array.
Sample Input

```
2
5
1 2 3 5 4
6
6 5 3 4 1 2
```

Sample Output

```
1 2 3 4 5
1 2 3 4 5 6
```

EX NO:2.4(A)	**HEAP SORT**
DATE:20.12.2018	

AIM:

To Write a program to sort using binary heap technique.

PSEUDOCODE:

```
//Program: To heapify.
//Input:  values
//Output: sorted ouput
heapsort(n,a)
   For i in range(n//2,0,-1)
      k=i
      While 2*k<=n
         j=2*k
         If j<n
            If a[j]<a[j+1]
               j+=1
         If a[k]<a[j]
            a[k],a[j]=a[j],a[k]
            k=j
         Else
            break
```

SOURCE CODE:

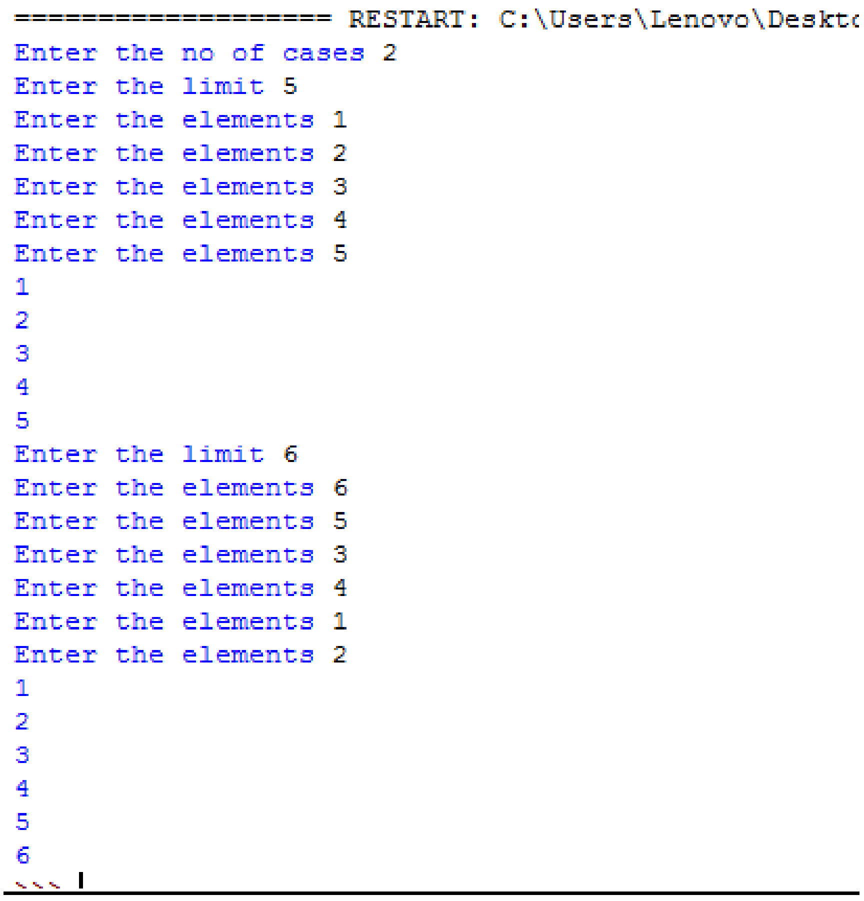
```
def heapsort(n,a):
   for i in range(n//2,0,-1):
      k=i
      while 2*k<=n:
         j=2*k
         if j<n:
            if a[j]<a[j+1]:
               j+=1
         if a[k]<a[j]:
            a[k],a[j]=a[j],a[k]
            k=j
         else:
            break
def delete(n,a):
   for i in range(n,1,-1):
      a[1],a[i]=a[i],a[1]
      heapsort(i-1,a)
for z in range(int(input("Enter the no of cases "))):
   a=[]
   n=int(input("Enter the limit "))
   a.append(-1)
```

```
        for i in range(1,n+1):
            a.append(int(input("Enter the elements ")))
        heapsort(n,a)
        delete(n,a)
        for i in range(1,n+1):
            print(a[i])
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n)

OUTPUT:



```
=================== RESTART: C:\Users\Lenovo\Deskto
Enter the no of cases 2
Enter the limit 5
Enter the elements 1
Enter the elements 2
Enter the elements 3
Enter the elements 4
Enter the elements 5
1
2
3
4
5
Enter the limit 6
Enter the elements 6
Enter the elements 5
Enter the elements 3
Enter the elements 4
Enter the elements 1
Enter the elements 2
1
2
3
4
5
6
>>> 
```

RESULT:

Thus a program to sort the input using heap technique was successfully executed and verified.

QUESTION:

In this problem you are given two type of query
1. Insert an integer to the list.
2. Given an integer **x**, you're about to find an integer **k** which represent x's index if the list is sorted in ascending order. Note that in this problem we will use 1-based indexing.

As the problem title suggest, this problem intended to be solved using Balanced Binary Search Tree, one of its example is AVL Tree.

Input

The first line contains an integer Q, which denotes how many queries that follows.
The next Q lines will be one of the type queries which follow this format: 1 x means insert x to the list 2 x means find x's index if the list is sorted in ascending order.

Output

For each query type 2, print a line containing an integer as the answer or print "Data tidak ada" no quotes if the requested number does not exist in the current lis.

Example

Input:

```
10
10
1 100
1 74
2 100
2 70
1 152
1 21
1 33
2 100
2 21
2 1
```

Output:

```
2
Data tidak ada
4
1
Data tidak ada
```

Explanation

Until the third query, the current list is {74, 100}. Therefore you must print 2 as 100 is on the first index.
Arriving at the fourth query we haven't add any other number so the list still consists of {74, 100}. Since 70 is not in the list you must print "Data tidak ada" remember no quotes.
For the last three queries the list looks like this {21, 33, 74, 100, 152} So the answer for the eighth, ninth, and tenth query respectively are 4, 1, and "Data tidak ada".

Constraints

$1 \leq Q \leq 200000$
$1 \leq x \leq 106$
It is guaranteed that all integer that inserted in the list will be distinct.

EX NO:2.4(B)	**AVL TREE**
DATE:20.12.2018	

AIM:

To Write a C program to design the avl tree for the given criteria.

PSEUDOCODE:

```
//Program: To design.
//Input:  values
//Output: required output
While(i<=k)
        Read b
        Switch(b)
        case 1:get ele
                 root=insert(root,ele)
                   break
        case 2:
                get ele
                if(search(root,ele))
                        print no of nodes in the tree
                else
                        print "Data tidak ada
        i++
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int v=0,c;
typedef struct node avl;
struct node
{
        struct node *lchild,*rchild;
        int data;
};
int height(avl *tree)
{
        if(tree==NULL)
                return -1;
        else
        {
                int lh=height(tree->lchild);
                int rh=height(tree->rchild);
                if(lh>rh)
                        return lh+1;
                else
                        return rh+1;
```

```
	}
}
int inorder(avl *temp,int ele)
{
	if(temp!=NULL)
	{
		inorder(temp->lchild,ele);
		if(ele==temp->data)
		{
			v=c;
			c=0;
			return v+1;
		}
		else
			c+=1;
		inorder(temp->rchild,ele);
	}
}
avl* LL(avl *tree)
{
	avl *k2=tree,*k1=k2->lchild;
	k2->lchild=k1->rchild;
	k1->rchild=k2;
	return k1;
}
int search(struct node *temp,int ele)
{
	if(temp==NULL)
		return 0;
	else
	{
		if(ele<temp->data)
			return search(temp->lchild,ele);
		else if(ele>temp->data)
			return search(temp->rchild,ele);
		else
			return 1;
	}
	return 0;
}
avl* RR(avl *tree)
{
	avl *k1=tree,*k2=k1->rchild;
	k1->rchild=k2->lchild;
	k2->lchild=k1;
	return k1;
}
```

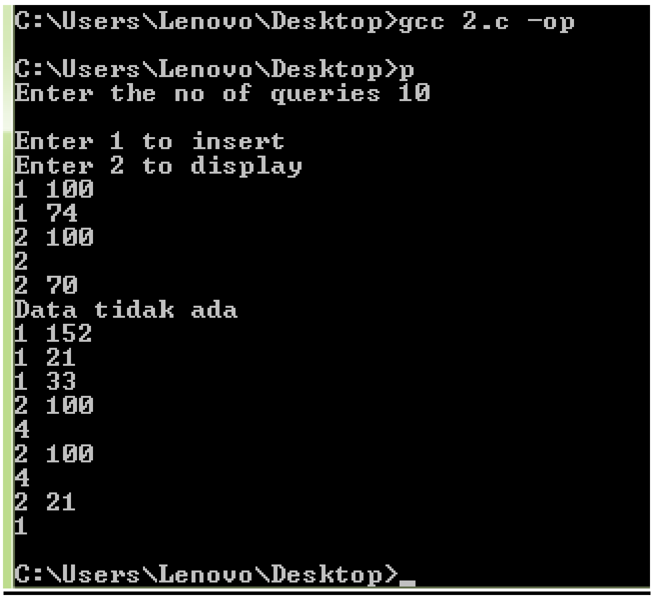
```
avl* LR(avl *tree)
{
        avl *k3=tree,*k1=k3->lchild,*k2=k1->rchild;
        k1->rchild=k2->lchild;
        k3->lchild=k2->rchild;
        k2->lchild=k1;
        k2->rchild=k3;
        return k2;
}
avl *RL(avl *tree)
{
        avl *k1=tree,*k3=k1->rchild,*k2=k2->lchild;
        k1->rchild=k2->lchild;
        k3->lchild=k2->rchild;
        k2->lchild=k1;
        k2->rchild=k2;
        return k2;
}
avl* balance(avl* tree)
{
        int heightdiff=height(tree->lchild)-height(tree->rchild);
        if(heightdiff==2)
        {
                int x=height(tree->lchild->rchild)-height(tree->lchild->rchild);
                if(x==1||x==0)
                        tree=LL(tree);
                else
                        tree=LR(tree);
        }
        else if(heightdiff==-2)
        {
                int y=height(tree->rchild->lchild)-height(tree->rchild->lchild);
                if(y==-1||y==0)
                        tree=RR(tree);
                else
                        tree=RL(tree);
        }
        return tree;
}
avl* insert(avl *tree,int ele)
{
        if(tree==NULL)
        {
                tree=(avl*)malloc(sizeof(avl));
                tree->data=ele;
                tree->rchild=tree->lchild=NULL;
        }
```

```
		else
		{
			if(ele<tree->data)
			{
				tree->lchild=insert(tree->lchild,ele);
				tree=balance(tree);
			}
			else if(ele>tree->data)
			{
				tree->rchild=insert(tree->rchild,ele);
				tree=balance(tree);
			}
		}
		return tree;
	}
	int main()
	{
		avl *root=NULL;
		int ele,i=1,k,b;
		printf("Enter the no of queries ");
		scanf("%d",&k);
		printf("\nEnter 1 to insert\nEnter 2 to display\n");
		while(i<=k)
		{
			scanf("%d",&b);
			switch(b)
			{
				case 1:     scanf("%d",&ele);
					root=insert(root,ele);
				   break;
				case 2:
					scanf("%d",&ele);
					if(search(root,ele))
						printf("%d\n",inorder(root,ele));
					else
						printf("Data tidak ada \n");
			}
			i++;
		}
		return 0;
	}
```

DATA STRUCTURE USED: AVL tree

TIME COMPLEXITY: O(log n)

OUTPUT:



```
C:\Users\Lenovo\Desktop>gcc 2.c -op

C:\Users\Lenovo\Desktop>p
Enter the no of queries 10

Enter 1 to insert
Enter 2 to display
1 100
1 74
2 100
2
2 70
Data tidak ada
1 152
1 21
1 33
2 100
4
2 100
4
2 21
1

C:\Users\Lenovo\Desktop>_
```

RESULT:

Thus a program to design the given problem was successfully executed and verified.

QUESTION:

Write an efficient algorithm to convert the given max heap into a min heap.

```
        10
       /   \
     5      6
    /  \   /  \
  2     1 4    3
```

EX NO:2.5(A)	**MAX HEAP TO MIN HEAP**
DATE:3.01.2019	

AIM:

To Write a program to convert max heap to min heap

PSEUDOCODE:

```
//Program: To heapify.
//Input:  values
//Output: Min heap ouput
heapsort(n,a)
   For i in range(n//2,0,-1)
      k=i
      While 2*k<=n
         j=2*k
         If j<n
            If a[j]>a[j+1]
               j+=1
         If a[k]>a[j]
            a[k],a[j]=a[j],a[k]
            k=j
         Else
            break
```

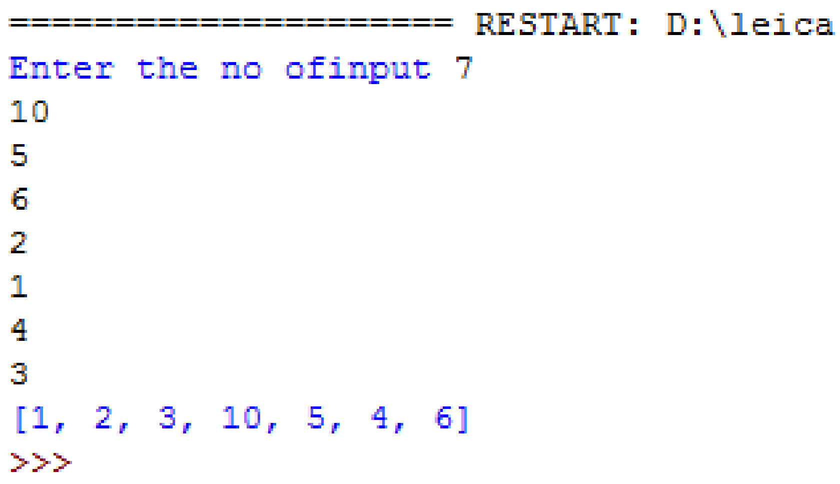
SOURCE CODE:

```
def heapify(n,a):
   for i in range(n//2,0,-1):
      k=i
      while((2*k)<=n):
         j=2*k
         if(j<n):
            if(a[j]>a[j+1]):
               j+=1
         if(a[k]>a[j]):
            a[k],a[j]=a[j],a[k]
            k=j
         else:
            break

li=[-1]
n=int(input("Enter the no ofinput "))
for i in range(n):
   li.append(int(input()))
heapify(n,li)
li.pop(0)
print(li)
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n)

OUTPUT:



```
====================== RESTART: D:\leica
Enter the no ofinput 7
10
5
6
2
1
4
3
[1, 2, 3, 10, 5, 4, 6]
>>>
```

RESULT:
Thus a program to covert min heap using heap technique was successfully executed and verified

QUESTION:

Given a dictionary of words and an input string, find out the longest pref
input string which is also present in the given dictionary.
For example, if dictionary consists of following words - **“word”, “cat”, “**
“name”
And if the input string is 'camera', then output should be 'cam'. If the inpu
'cataract', output should be cat.

EX NO:2.5(B)	TRIES
DATE:3.01.2019	

AIM:

To Write a program to implement tries.

PSEUDOCODE:

```
//Program: To find the string.
//Input:  values
//Output: string found or not
For i←0  to i<n
    Read str
    If(search(root,str))
        Print FOUND
    Else
        Print NOT FOUND
```

SOURCE CODE:

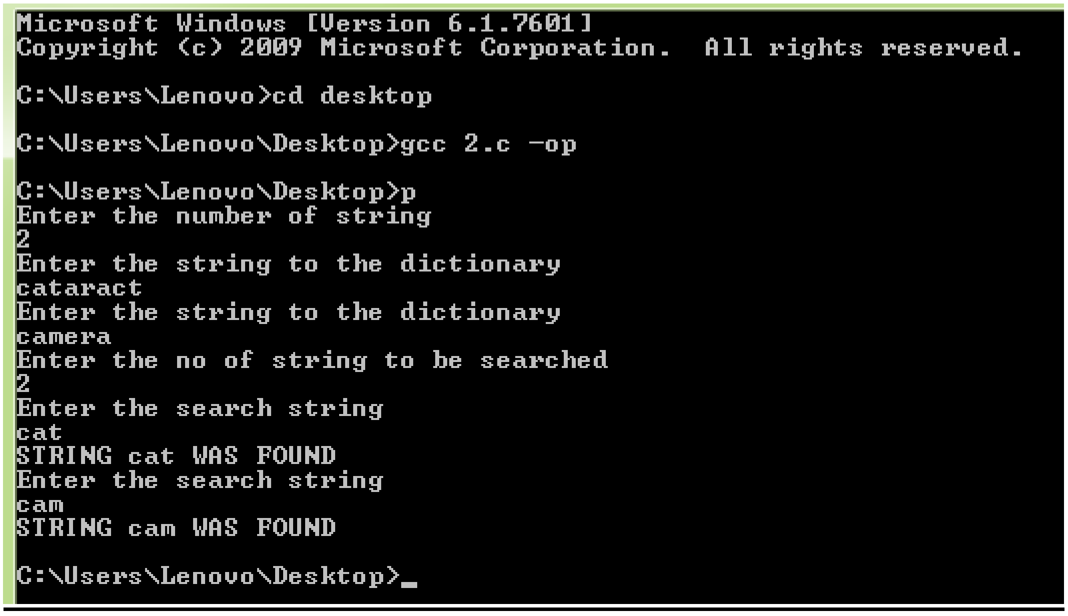
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct standardtries
{
    int flag;
    char data;
    struct standardtries *addr[26];
};
typedef struct standardtries tries;
tries* insert(tries *tree,char str[])
{
    tries *temp=tree;
    int len=strlen(str),i,j,index;
    for(i=0;i<len;i++)
    {
        index=str[i]-'a';
        if(temp->addr[index]==NULL)
        {
            temp->addr[index]=(tries*)malloc(sizeof(tries));
            temp=temp->addr[index];
            temp->data=str[i];
            temp->flag=0;
            for(j=0;j<26;j++)
                temp->addr[j]=NULL;
        }
        else
            temp=temp->addr[index];
    }
```

```
        temp->flag=1;
        return tree;
}
int search(tries *tree,char str[])
{
        tries *temp=tree;
        int len=strlen(str),i,index;
        for(i=0;i<len;i++)
        {
                index=str[i]-'a';
                if(temp->addr[index]==NULL)
                        return 0;
                else
                        temp=temp->addr[index];
        }
        return 1;
}
void main()
{
        tries *root=(tries*)malloc(sizeof(tries));
        root->flag=0;
        int i;
        for(i=0;i<26;i++)
                root->addr[i]=NULL;
        int n;
        printf("Enter the number of string\n");
        scanf("%d",&n);
        char str[15];
        for(i=0;i<n;i++)
        {
                printf("Enter the string to the dictionary\n");
                scanf("%s",str);
                root=insert(root,str);
        }
        printf("Enter the no of string to be searched\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter the search string\n");
                scanf("%s",str);
                if(search(root,str))
                        printf("STRING %s WAS FOUND\n",str);
                else
                        printf("STRING %s WAS NOT FOUND\n",str);
        }
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(nm)

OUTPUT:



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Lenovo>cd desktop

C:\Users\Lenovo\Desktop>gcc 2.c -op

C:\Users\Lenovo\Desktop>p
Enter the number of string
2
Enter the string to the dictionary
cataract
Enter the string to the dictionary
camera
Enter the no of string to be searched
2
Enter the search string
cat
STRING cat WAS FOUND
Enter the search string
cam
STRING cam WAS FOUND

C:\Users\Lenovo\Desktop>_
```

RESULT:

Thus a program to implement tries technique was successfully executed and verified

QUESTION:

Given a tree represented as undirected graph. Count the number of nodes at given level l. It may be assumed that vertex 0 is root of the tree.

Examples:

```
Input : 7
0 1
0 2
1 3
1 4
1 5
2 6
2
Output : 4
Input : 6
0 1
0 2
1 3
2 4
2 5
2
Output : 3
```

BFS is a traversing algorithm which start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly connected to source node). Then, move towards the next-level neighbour nodes.

As the name BFS suggests, traverse the graph breadth wise as follows:

1. First move horizontally and visit all the nodes of the current layer.

2. Move to the next layer.

In this code, while visiting each node, the level of that node is set with an increment in the level of its parent node i.e., level[child] = level[parent] + 1. This is how the level of each node is determined. The root node lies at level zero in the tree.

Explanation :

```
    0        Level 0
   /  \
  1   2      Level 1
 / |\   \|
3 4 5   6   Level 2
```

Given a tree with 7 nodes and 6 edges in which node 0 lies at 0 level. Level of 1 can be updated as : level[1] = level[0] +1 as 0 is the parent node of 1. Similarly, the level of other nodes can be updated by adding 1 to the level of their parent.

level[2] = level[0] + 1, i.e level[2] = 0 + 1 = 1.
level[3] = level[1] + 1, i.e level[3] = 1 + 1 = 2.
level[4] = level[1] + 1, i.e level[4] = 1 + 1 = 2.
level[5] = level[1] + 1, i.e level[5] = 1 + 1 = 2.
level[6] = level[2] + 1, i.e level[6] = 1 + 1 = 2.

Then, count of number of nodes which are at level l(i.e, l=2) is 4 (node:- 3, 4, 5, 6)

EX NO:3.1(A)	COUNTING THE ELEMENTS IN THE GRAPH
DATE:24.01.2019	

AIM:

To Write a program to count the no of elements in desired level.

PSEUDOCODE:

```
//Program: To count the no of elements in desired level.
//Input:  Tree in the form of graph
//Output: count of the elements
While(front<=rear)
    k=Dequeue()
    topsort[j++]=k
    For i←1 to n
        If(adj[k][i]==1)
            adj[k][i]=0
            indeg[i]=indeg[i]-1
            If(indeg[i]==0)
                Enqueue(i)
                c2[ia]=c2[ia]+1
    If(c1==1)
        c1=c2[ia++]
    Else
         c1-=1
```

SOURCE CODE:

```
#include<stdio.h>
# include<stdlib.h>
#define MAX 50
int n,adj[MAX][MAX];
int front = -1,rear = -1,queue[MAX];
void create_graph()
{
    int i,j,v1,v2,ch=0;
    printf("\nEnter the no of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            adj[i][j]=0;
    do{
        printf("Enter the edge(v1,v2):");
        scanf("%d%d",&v1,&v2);
        adj[v1+1][v2+1]=1;
        printf("\nAdd more(0/1):");
        scanf("%d",&ch);
    }while(ch==1);
}
```

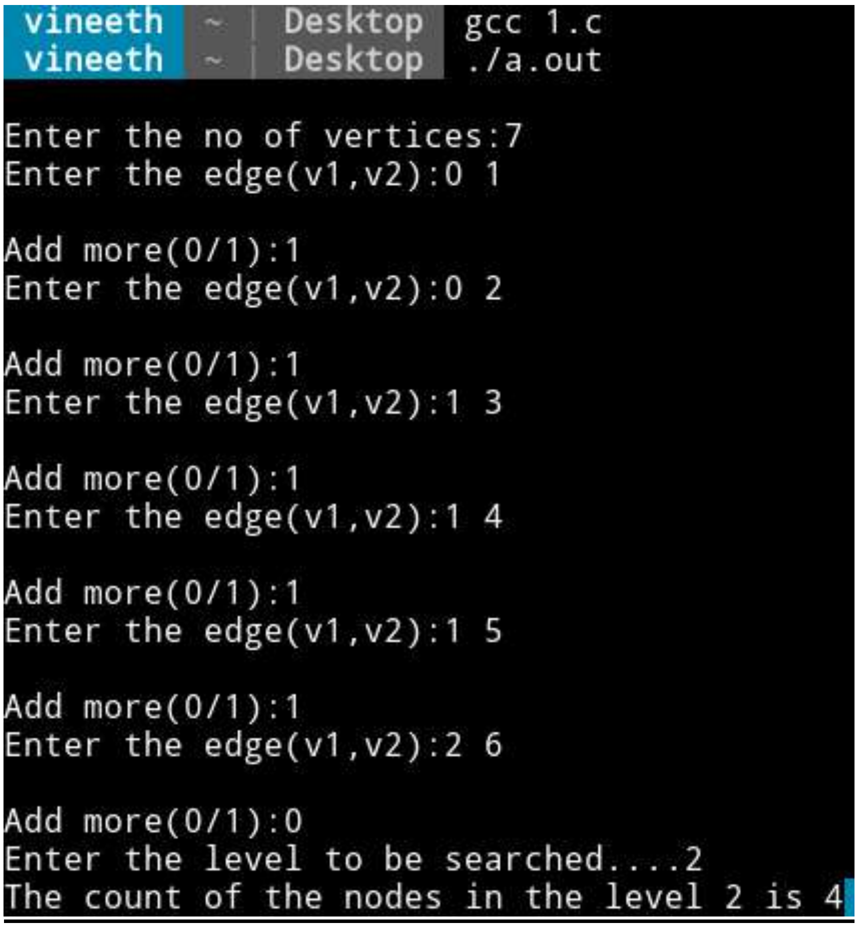
```
int indegree(int node)
{
        int i,count=0;
        for(i=1;i<=n;i++)
                if(adj[i][node]==1)
                        count++;
        return count;
}

void enqueue(int i)
{        int a;
        if(front==-1)
                front++;
        rear++;
        queue[rear]=i;
}
int dequeue()
{      int i;
        i=queue[front];
        front++;
        return i;
}
void main()
{
        int i,j=1,k,gv;
        int topsort[MAX],indeg[MAX];
        create_graph();
        int c2[n],ia=1,c1=1;
        for(i=1;i<=n;i++)
           c2[i]=0;
        for(i=1;i<=n;i++)
        {
                indeg[i]=indegree(i);
                if(indeg[i]==0)
                enqueue(i);
         }
        while(front<=rear)
        {
                k=dequeue();
                topsort[j++]=k;
                for(i=1;i<=n;i++)
                {
                        if(adj[k][i]==1)
                        {
                                adj[k][i]=0;
                                indeg[i]=indeg[i]-1;
                                if(indeg[i]==0)
```

```
                                        {
                                                enqueue(i);
                                                c2[ia]=c2[ia]+1;
                                        }
                                }
                        }
                if(c1==1)
                        c1=c2[ia++];
                else
                        c1-=1;
        }
        printf("Enter the level to be searched....");
        scanf("%d",&i);
        if(i==0)
                printf("The count of the nodes in the level 0 is 1");
        else
                printf("The count of the nodes in the level %d is %d",i,c2[i]);
        }
```

DATA STRUCTURE USED: Two dimensional Array

TIME COMPLEXITY: O(n^2)

OUTPUT:

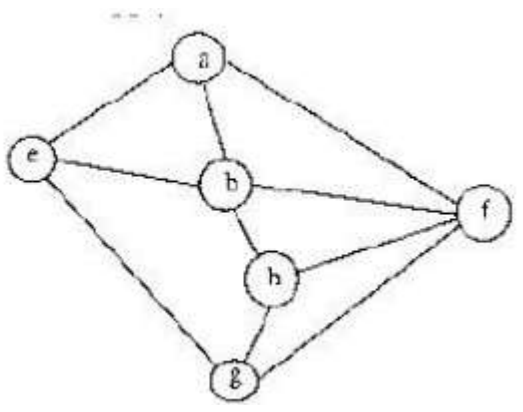


```
vineeth  ~ | Desktop  gcc 1.c
vineeth  ~ | Desktop  ./a.out

Enter the no of vertices:7
Enter the edge(v1,v2):0 1

Add more(0/1):1
Enter the edge(v1,v2):0 2

Add more(0/1):1
Enter the edge(v1,v2):1 3

Add more(0/1):1
Enter the edge(v1,v2):1 4

Add more(0/1):1
Enter the edge(v1,v2):1 5

Add more(0/1):1
Enter the edge(v1,v2):2 6

Add more(0/1):0
Enter the level to be searched....2
The count of the nodes in the level 2 is 4
```

RESULT:

Thus a program to count the elements in the level was successfully executed and verified

QUESTION:
Implement Depth First Traversal for the following graph which is represented using Adjacency matrix.



EX NO:3.1(B)	IMPLEMENTATION OF DFS
DATE:24.01.2019	

AIM:

To Write a program to implement DFS

PSEUDOCODE:

```
//Program: To implement dfs
//Input:  Adjacency matrix representing graph
//Output: DFS
void dfs(v)
        visit[v]=1;
        for w←1 to n
                if(adj[v][w]==1)
                        if(visit[w]==0)
                                dfs(w)
```

SOURCE CODE:

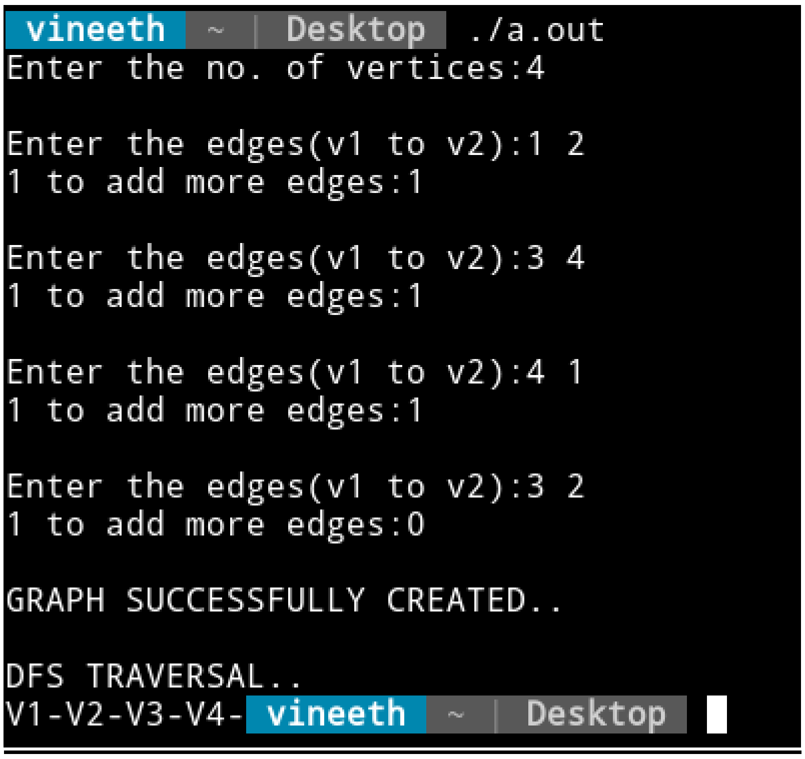
```
#include<stdio.h>
#define MAX 15
int n,adj[MAX][MAX],visit[MAX],dfsorder[MAX];
int i,j,v1,v2,count=1;
void create_graph()
{
        int ch=1;
        printf("Enter the no. of vertices:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        adj[i][j]=0;
        do
        {
                printf("\nEnter the edges(v1 to v2):");
                scanf("%d%d",&v1,&v2);
                adj[v1][v2]=adj[v2][v1]=1;
                printf("1 to add more edges:");
                scanf("%d",&ch);
        }while(ch==1);
}
void display()
{
        printf("\nAdjacacency Matrix");
        printf("\nVERTEX");
        for(i=1;i<=n;i++)
                printf("\tV%d",i);
        for(i=1;i<=n;i++)
```

```
        {
                printf("\nV%d",i);
                for(j=1;j<=n;j++)
                {
                        printf("\t%d",adj[i][j]);
                }
        }
}
void dfs(int v)
{
        int w;
        printf("V%d-",v);
        visit[v]=1;

        for(w=1;w<=n;w++)
        {
                if(adj[v][w]==1)
                {
                        if(visit[w]==0)
                                dfs(w);
                }
        }
}
int main()
{
        create_graph();
        printf("\nGRAPH SUCCESSFULLY CREATED..\n");
        display();
        for(i=1;i<=n;i++)
        {
                visit[i]=0;
        }
        printf("\nDFS TRAVERSAL..\n");
        for(i=1;i<=n;i++)
        {
                if(visit[i]==0)
                        dfs(i);

        }
        return 0;
}
```

DATA STRUCTURE USED: Two dimensional Array

TIME COMPLEXITY: O(n^2)

OUTPUT:

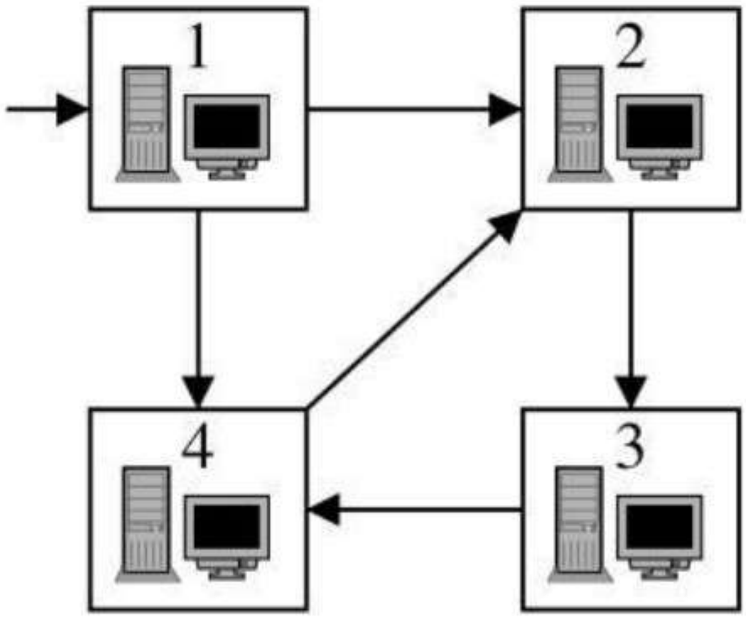


```
 vineeth  ~ | Desktop  ./a.out
Enter the no. of vertices:4

Enter the edges(v1 to v2):1 2
1 to add more edges:1

Enter the edges(v1 to v2):3 4
1 to add more edges:1

Enter the edges(v1 to v2):4 1
1 to add more edges:1

Enter the edges(v1 to v2):3 2
1 to add more edges:0

GRAPH SUCCESSFULLY CREATED..

DFS TRAVERSAL..
V1-V2-V3-V4- vineeth  ~ | Desktop
```

RESULT:
Thus a program to implement DFS was successfully executed and verified.

QUESTION:

Bytelandian Information Agency (BIA) uses a net of n computers. The computers are numbered from 1 to n, and the computer number 1 is a server. The computers are connected by one-way information channels. Every channel connects a pair of computers. The whole network is organised in such a way that one can send information from the server to any other computer either directly or indirectly.
When BIA acquires new information, the information is put on the server and propagated in the net. The chief of BIA considers what would happen if one computer stopped working (was blown away by terrorists for example). It could happen that some other computers would stop receiving information from the server, because the broken computer was a necessary transmitter. We will call such computers critical.
Solve it using Kruskal's algorithm.For example in the situation in the picture below the critical computers are 1 and 2. 1 is the server and all information sent from the server to 3 has to go through 2.



Task
Write a program which
- reads a description of the net from standard input,
- finds all critical computers.
- writes the numbers of critical computers to standard output.

EX NO:3.2(A)	FINDING THE VERTEX
DATE:02.02.2019	

AIM:

To Write a program to find the following vertex

PSEUDOCODE:

```
//Program: To find the vertex
//Input: Adjacency matrix representing graph
//Output: vertices
For w←1 to w<=n
    For l←1 to l<=n
        copy[w][l]=0
        copy[l][w]=0
    For k←1 to k<=n
            If(indegree(k)==0&&k!=w&&k!=p)
                printf(w)
    copy1()
```

SOURCE CODE:

```
#include<stdio.h>
#define MAX 15
int n,adj[MAX][MAX],copy[MAX][MAX],visit[MAX],dfsorder[MAX];
int i,j,v1,v2,count=1;
void create_graph();
void display();
void dfs(int v);
void create_graph()
{
    int ch=1;
    printf("Enter the no. of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            adj[i][j]=0;
    do
    {
        printf("\nEnter the edges(v1 to v2):");
        scanf("%d%d",&v1,&v2);
        adj[v1][v2]=1;
        printf("1 to add more edges:");
        scanf("%d",&ch);
    }while(ch==1);
}
void copy1()
{
```

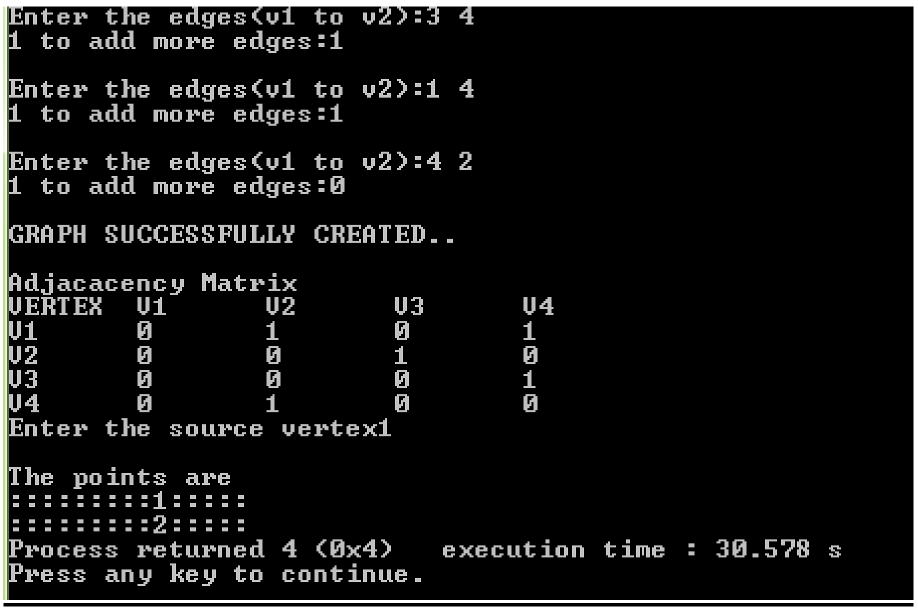
```
for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
                copy[i][j]=adj[i][j];
}
void display()
{
        printf("\nAdjacacency Matrix");
        printf("\nVERTEX");
        for(i=1;i<=n;i++)
                printf("\tV%d",i);
        for(i=1;i<=n;i++)
        {
                printf("\nV%d",i);
                for(j=1;j<=n;j++)
                {
                        printf("\t%d",adj[i][j]);
                }
        }
}
int indegree(int node)
{
        int i,count=0;
        for(i=1;i<=n;i++)
                if(copy[i][node]==1)
                        count++;
        return count;
}

void main()
{
        int k,w,l,p;
        create_graph();
        printf("\nGRAPH SUCCESSFULLY CREATED..\n");
        display();
        copy1();
        printf("\nEnter the source vertex");
        scanf("%d",&p);
        printf("\nThe points are ");
        printf("\n::::::::%d:::::",p);
        for(w=1;w<=n;w++)
        {
                for(l=1;l<=n;l++)
                {       copy[w][l]=0;
                        copy[l][w]=0;
                }
                for(k=1;k<=n;k++)
                {
```

```
                                        if(indegree(k)==0&&k!=w&&k!=p)
                                                printf("\n:::::::::%d:::::",w);
                        }
                        copy1();
                }

        }
```

DATA STRUCTURE USED: Two dimensional Array

TIME COMPLEXITY: O(n^2)

OUTPUT:



```
Enter the edges(v1 to v2):3 4
1 to add more edges:1

Enter the edges(v1 to v2):1 4
1 to add more edges:1

Enter the edges(v1 to v2):4 2
1 to add more edges:0

GRAPH SUCCESSFULLY CREATED..

Adjacacency Matrix
VERTEX  V1      V2      V3      V4
V1      0       1       0       1
V2      0       0       1       0
V3      0       0       0       1
V4      0       1       0       0
Enter the source vertex1

The points are
:::::::::1:::::
:::::::::2:::::
Process returned 4 (0x4)   execution time : 30.578 s
Press any key to continue.
```

RESULT:
Thus a program to find the vertex was successfully executed and verified.

QUESTION:

Petya live in a city named Mayapur. As the in the morning, everybody likes to drink hot tea in bed. So the citizens needs milk to produce tea. For this purpose, they want that they should be able to go to at least some milkmen using bi-directional roads.
There are m roads in the city but all of them are currently unrepaired, hence not a state of use. As Petya cares about his city a lot. He got a project about repairing these roads. So he has to select some roads to repair such that every citizen is connected to at least one milkman. For repairing each road he needs to pay a cost. As he does not want to spend a lot of money in it, He wants to minimize the cost needed in this project. Note that a milkmen does not need to go to some other milkmen for milk as he can take milk from his own home.
Help Petya in finding out minimum cost needed to repair the roads in the above given way. If it is not possible for a citizen to connect to any of the milkmen, output "impossible" (without quotes). Solve it using Prim’s algorithm.

Input

First line contains T : number of test cases. (1 <= T <= 100)
For each test case, First line contains two space seperated number n, m: number of citizens in Mayapur and m denotes number of unrepaired roads.
Next line contains n space integers either 0 or 1 which indices that citizen is milkmen or not[1 means he is a milkman]. (1 <= n <= 10^5)
Then For each of the next m lines, each line contains three space seperated integers u, v and c, denoting that there exists a unrepaired road between u and v such that cost of repairing of road is c. (1 <= u, v <= n and u != v)
(1 <= m <= min (n * (n - 1) / 2, 2 * 10^5). 1 <= c <= 10^9.)

Output

For each test case output the minimum cost as required.

Example

Input:

```
1
5 72
0 1 0 1 0
1 2 11
1 3 1
1 5 17
2 3 1
3 5 18
4 5 3
2 4 5
```

Output:

```
22
```

EX NO:3.2(B)	**MILK DISTRIBUTION**
DATE:02.02.2019	

AIM:

To Write a program solve the distribution of Milk problem

PSEUDOCODE:

```
//Program: To find low cost
//Input:  Adjacency matrix representing graph
//Output: cost
For i🡐1 to i<n
        If(visited[i]==0&&distance[i]<min_distance)
            If(!(lop[i]==1&&lop[from[i]]==1))
                v=i
                min_distance=distance[i]
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;
 int lop[MAX];
int prims();
int main()
{
    int i,j,total_cost,w,k=1;
    printf("Enter no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the values\n");
    for(i=0;i<n;i++)
        scanf("%d",&lop[i]);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            G[i][j]=0;
    printf("Enter the position a , b and the weight ");
    do
    {
        scanf("%d%d%d",&i,&j,&w);
        i=i-1;
        j=j-1;
        G[i][j]=G[j][i]=w;
        printf("Enter 1 to add more edges or 0  :  ");
        scanf("%d",&k);
```

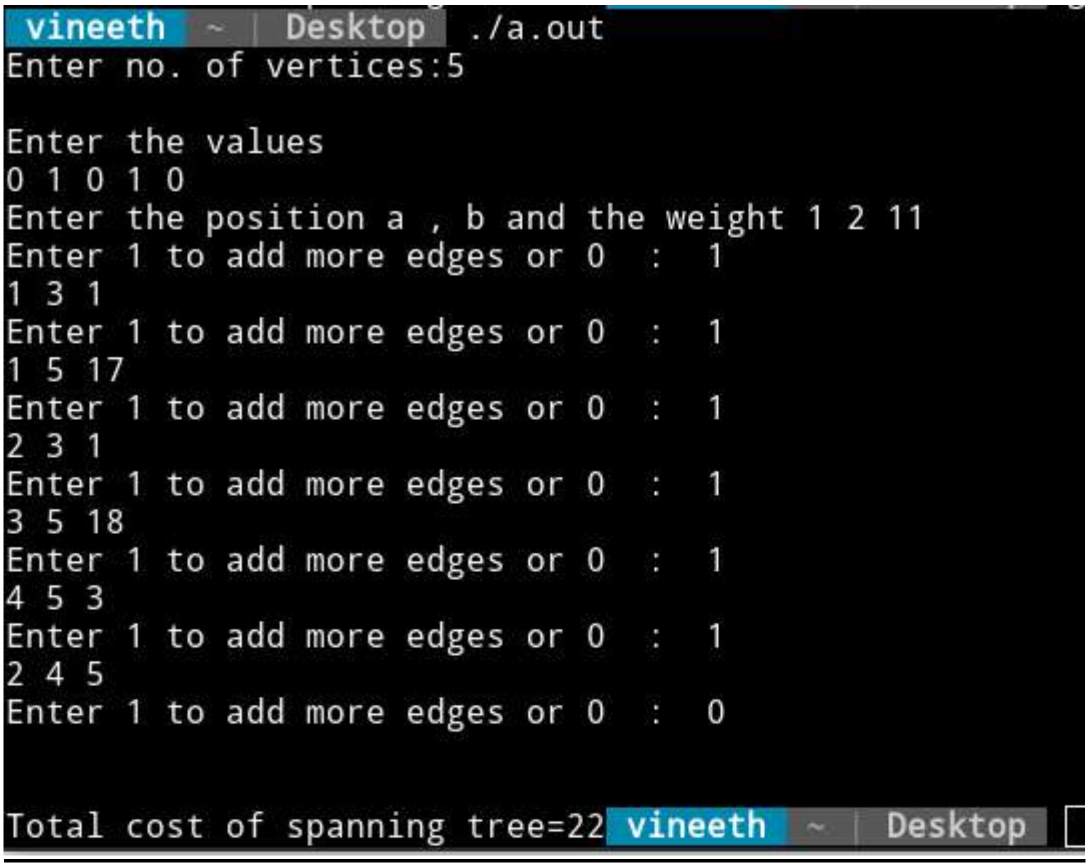
```
	}while(k);
	total_cost=prims();
	printf("\n\nTotal cost of spanning tree=%d",total_cost);
	return 0;
}

int prims()
{
	int cost[MAX][MAX];
	int u,v,min_distance,distance[MAX],from[MAX];
	int visited[MAX],no_of_edges,i,min_cost,j;

	//create cost[][] matrix,spanning[][]
	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
		{
			if(G[i][j]==0)
				cost[i][j]=infinity;
			else
				cost[i][j]=G[i][j];
				spanning[i][j]=0;
		}

	distance[0]=0;
	visited[0]=1;

	for(i=1;i<n;i++)
	{
		distance[i]=cost[0][i];
		from[i]=0;
		visited[i]=0;
	}

	min_cost=0;
	no_of_edges=n-1;

	while(no_of_edges>0)
	{
		min_distance=infinity;
		for(i=1;i<n;i++)
			if(visited[i]==0&&distance[i]<min_distance)
			{
				if(!(lop[i]==1&&lop[from[i]]==1))
				{
					v=i;
					min_distance=distance[i];
				}
```

```
                        }

                    u=from[v];
                    spanning[u][v]=distance[v];
                    spanning[v][u]=distance[v];
                    no_of_edges--;
                    visited[v]=1;

                    for(i=1;i<n;i++)
                        if(visited[i]==0&&cost[i][v]<distance[i])
                        {
                            distance[i]=cost[i][v];
                            from[i]=v;
                        }

                    min_cost=min_cost+cost[u][v];
                }

                return(min_cost);
        }
```

DATA STRUCTURE USED: Two dimensional Array

TIME COMPLEXITY: O(n^2)

OUTPUT:

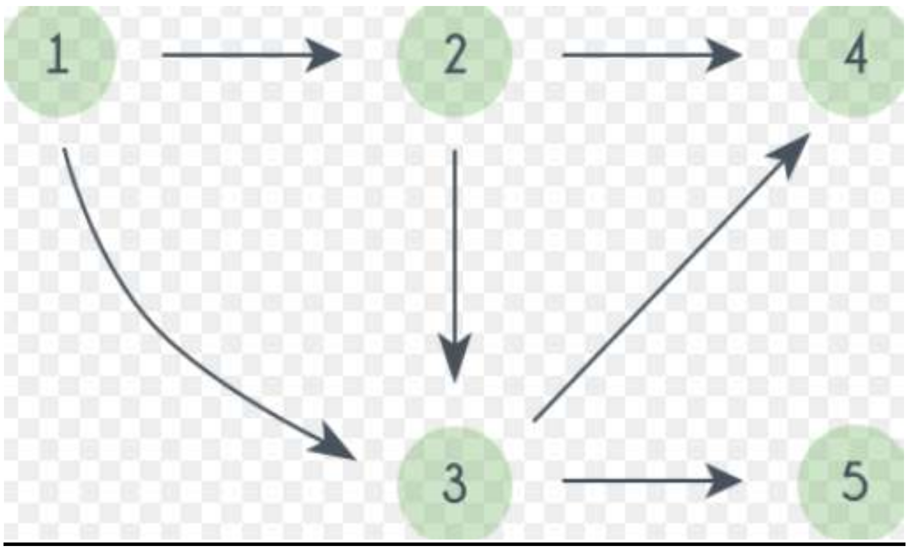


```
vineeth ~ | Desktop  ./a.out
Enter no. of vertices:5

Enter the values
0 1 0 1 0
Enter the position a , b and the weight 1 2 11
Enter 1 to add more edges or 0  :  1
1 3 1
Enter 1 to add more edges or 0  :  1
1 5 17
Enter 1 to add more edges or 0  :  1
2 3 1
Enter 1 to add more edges or 0  :  1
3 5 18
Enter 1 to add more edges or 0  :  1
4 5 3
Enter 1 to add more edges or 0  :  1
2 4 5
Enter 1 to add more edges or 0  :  0


Total cost of spanning tree=22 vineeth ~ | Desktop
```

RESULT:
Thus a program to solve the distribution of milk was successfully executed and verified.

QUESTION:

Consider the DAG with Consider V = {1, 2, 3, 4, 5}, shown below. Give the topological ordering for the graph using appropriate data structure.


$$\begin{tikzcd}
1 \arrow[r] \arrow[dr, bend right] & 2 \arrow[r] \arrow[d] & 4 \\
& 3 \arrow[ur] \arrow[r] & 5
\end{tikzcd}$$

EX NO:3.3(A)	TOPOLOGICAL SORT
DATE:07.03.2019	

AIM:

To write a program to implement topological sort

PSEUDOCODE:

```
//Program: To implement topological sort
//Input:  Adjacency matrix representing graph
//Output: topological order
While(front<=rear)
        k←dequeue()
        topsort[j++]←k
        For i←1 to i<=n
                If(G[k][i]==1)
                        G[k][i]=0
                        indeg[i]=indeg[i]-1
                If(indeg[i]==0)
                        enqueue(i)
```

SOURCE CODE:

```
#include<stdio.h>
# include<stdlib.h>
#define MAX 50
int n,G[MAX][MAX];
int front = -1,rear = -1,queue[MAX];
void create_graph();
void enqueue(int i);
int dequeue();
int indegree(int node);
void create_graph()
{
        int e,v1,v2,i,j;
        printf("\nEnter The No Of Vertices : ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        G[i][j]=0;
        printf("\nEnter the No of Edges : ");
      scanf("%d",&e);
        printf("Enter The Edges :\n");
        printf("FROM\tTO\n");
      for(i=1;i<=e;i++)
        {
                scanf("%d%d",&v1,&v2);
                G[v1][v2]=1;
        }
```

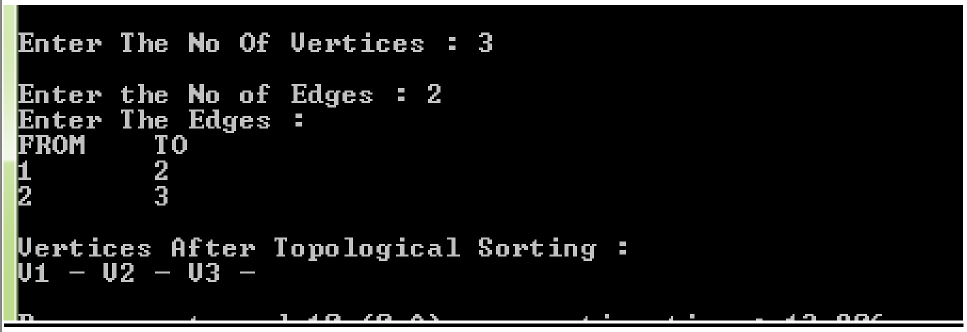
```
}

int indegree(int node)
{
        int i,count=0;
        for(i=1;i<=n;i++)
                if(G[i][node]==1)
                        count++;
        return count;
}
void enqueue(int i)
{
        if(front==-1)
                front++;
        rear++;
        queue[rear]=i;
}
int dequeue()
{
        int i;
        i=queue[front];
        front++;
        return i;
}

void main()
{
int i,j=1,k;
int topsort[MAX],indeg[MAX];
create_graph();
for(i=1;i<=n;i++)
{
        indeg[i]=indegree(i);
        if(indeg[i]==0)
                enqueue(i);
}
while(front<=rear)
{
        k=dequeue();
        topsort[j++]=k;
        for(i=1;i<=n;i++)
        {
                if(G[k][i]==1)
                {
                G[k][i]=0;
                indeg[i]=indeg[i]-1;
                if(indeg[i]==0)
```

```
                        enqueue(i);
                    }
                }
        }
        printf("\nVertices After Topological Sorting : \n");
        for(i=1;i<=n;i++)
                printf("V%d - ",topsort[i]);
        printf("\n");
        }
```

DATA STRUCTURE USED: Two dimensional Array

TIME COMPLEXITY: O(n^2)

OUTPUT:

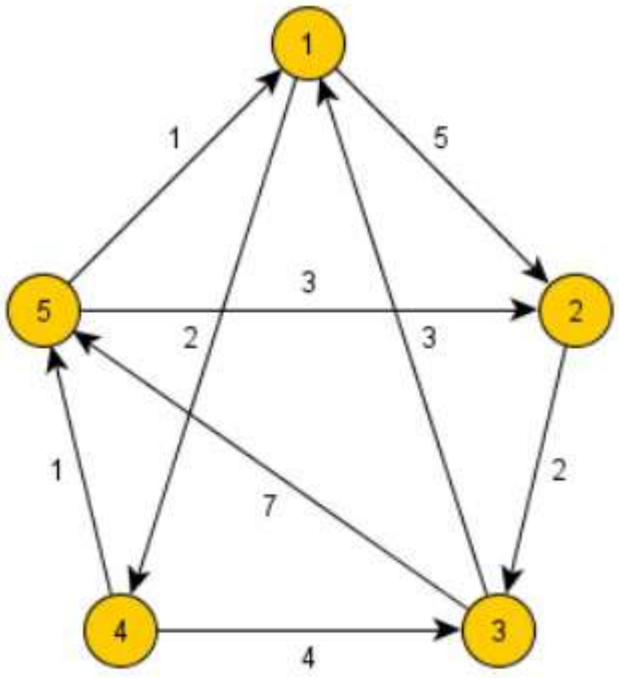


```
Enter The No Of Vertices : 3

Enter the No of Edges : 2
Enter The Edges :
FROM    TO
1       2
2       3

Vertices After Topological Sorting :
V1 - V2 - V3 -
```

RESULT:

Thus a program to implement topological sort was successfully executed and verified.

QUESTION:
Consider the following graph:



The numbers next to the edges denote the length of the edge. Execute the shortest paths algorithm between all pairs of nodes.

EX NO:3.3(B)	SHORTEST PATH ALGORITHM
DATE:07.03.2019	

AIM:

To Write a program for shortest path algorithm

PSEUDOCODE:

```
//Program: To implement shortest path algorithm
//Input:  Adjacency matrix representing graph
//Output: All Paths
For k ← 0 to  k < n
  For  i = 0 to  i < n
   For  j = 0 to j < n
    If (dist[i][k] + dist[k][j] < dist[i][j])
     dist[i][j] ← dist[i][k] + dist[k][j]
```

SOURCE CODE:

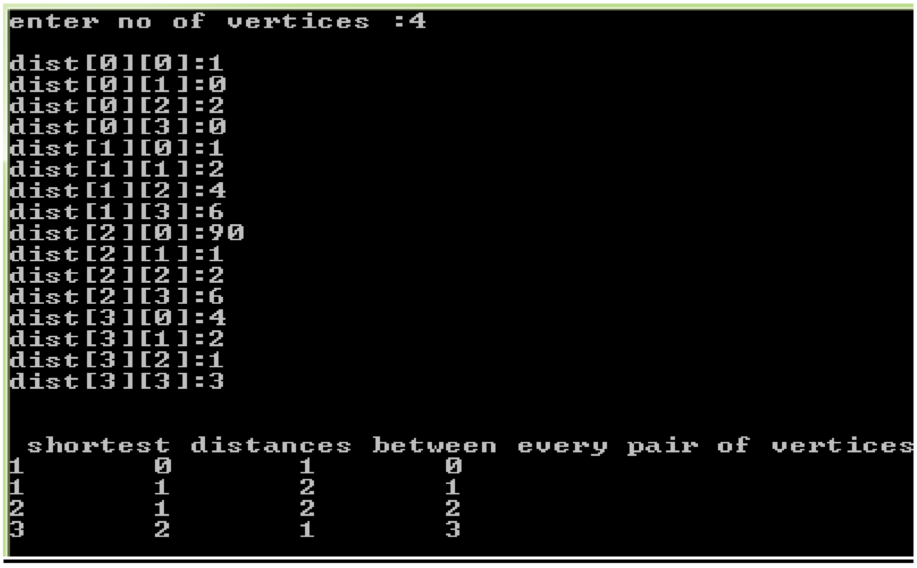
```
#include<stdio.h>
int i, j, k,n,dist[50][50];
void floydWarshell ()
{
 for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
   for (j = 0; j < n; j++)
    if (dist[i][k] + dist[k][j] < dist[i][j])
     dist[i][j] = dist[i][k] + dist[k][j];
}
void main()
{
  int i,j;
  printf("enter no of vertices :");
  scanf("%d",&n);
  printf("\n");
  for(i=0;i<n;i++)
  for(j=0;j<n;j++)
   {
    printf("dist[%d][%d]:",i,j);
    scanf("%d",&dist[i][j]);
   }
 floydWarshell();
 printf (" \n\n shortest distances between every pair of vertices \n");
 for (i = 0; i < n; i++)
 {
  for (j = 0; j < n; j++)
   printf ("%d\t", dist[i][j]);
  printf("\n");
 }
```

}

DATA STRUCTURE USED: Two dimensional Array

TIME COMPLEXITY: O(n^2)

OUTPUT:



```
enter no of vertices :4

dist[0][0]:1
dist[0][1]:0
dist[0][2]:2
dist[0][3]:0
dist[1][0]:1
dist[1][1]:2
dist[1][2]:4
dist[1][3]:6
dist[2][0]:90
dist[2][1]:1
dist[2][2]:2
dist[2][3]:6
dist[3][0]:4
dist[3][1]:2
dist[3][2]:1
dist[3][3]:3


 shortest distances between every pair of vertices
1       0       1       0
1       1       2       1
2       1       2       2
3       2       1       3
```

RESULT:

Thus a program to implement Floyd warshell’s algorithm was successfully executed and verified.

QUESTION:
By making use of a pseudo-random number generator to simulate random choices of the pivot element, we can make QuickSort behave as if whatever input it receives is actually an average case. Execute an algorithm for randomized quick sort.

EX NO:3.4(A)	**RANDOMISED QUICK SORT**
DATE:07.03.2019	

AIM:

To Write a program for randomized quick sort

PSEUDOCODE:

```
//Program: To implement randomised quick sort
//Input:  unsorted array
//Output: sorted array
While(i < j)
    While(a[i] < piv)
        i++
    While(a[j] > piv)
        j--
    temp←a[i]
    a[i]←a[j]
    a[j]←temp
```

SOURCE CODE:

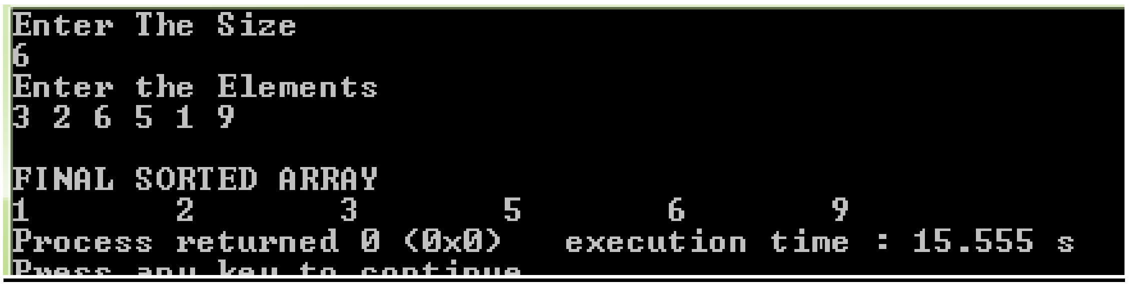
```
#include<stdio.h>
#include <stdlib.h>
int Randoms(int lower, int upper)
{
      int num = (rand() % (upper - lower + 1)) + lower;
    return num;
 }
void quick(int a[],int low,int high)
{
 int i,j,temp,piv;
  if(low < high)
   {
    i=low;
    j=high;
    srand(time(0));
    piv=a[Randoms(low,high)];
    while(i < j)
     {
      while(a[i] < piv)
       {
        i++;
       }
      while(a[j] > piv)
       {
        j--;
       }
```

```
          temp=a[i];
          a[i]=a[j];
          a[j]=temp;
         }
       quick(a,low,j-1);
       quick(a,j+1,high);
      }
    }
   main()
    {
      int n;
      printf("Enter The Size\n");
      scanf("%d",&n);
      int b[n],i;
      printf("Enter the Elements\n");
       for(i=0;i<n;i++)
          scanf("%d",&b[i]);
       quick(b,0,n-1);
       printf("\nFINAL SORTED ARRAY\n");
       for(i=0;i<n;i++)
        {
          printf("%d\t",b[i]);
         }
     }
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: O(n log n)

OUTPUT:



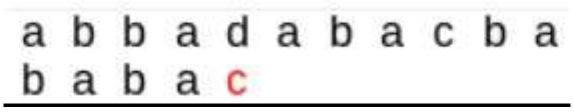
```
Enter The Size
6
Enter the Elements
3 2 6 5 1 9

FINAL SORTED ARRAY
1       2       3       5       6       9
Process returned 0 (0x0)   execution time : 15.555 s
Press any key to continue
```

RESULT:

Thus a program to implement randomised quick sort algorithm was successfully executed and verified.

QUESTION:

Using KMP algorithm find whether the second string(pattern) is present in first string(text).



EX NO:3.4(B)	KMP ALGORITHM
DATE:07.03.2019	

AIM:

To Write a program to implement KMP algorithm

PSEUDOCODE:

```
//Program: To implement KMP
//Input:  enter the text
//Output: return whether string found or not
while(i < N)
  if(pat[j] == txt[i])
        j++
        i++
  if (j == M)
        j = lps[j-1]
  else if(pat[j] != txt[i])
    if(j != 0)
        j = lps[j-1]
    else
        i = i+1
```

SOURCE CODE:

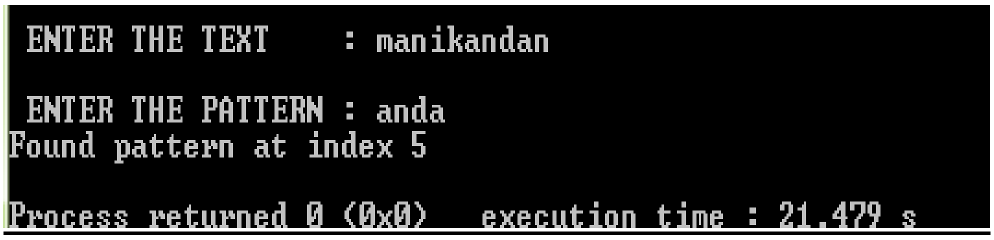
```
#include<stdio.h>
#include<string.h>
char txt[100],pat[100];
int M ,N ,lps[100],j=0,i=0;
void computeLPSArray()
{
  int len = 0, i;
  lps[0] = 0;
  i = 1;
  while(i < M)
  {
        if(pat[i] == pat[len])
        {
                len++;
                lps[i] = len;
                i++;
        }
        else
        {
                if( len != 0 )
                        len = lps[len-1];
                else
                {
                        lps[i] = 0;
```

```
                        i++;
                    }
                }
  }
}
void KMPSearch()
{
  int j=0,i=0;
  M = strlen(pat);
  N = strlen(txt);
  computeLPSArray();
  while(i < N)
  {
        if(pat[j] == txt[i])
        {
              j++;
              i++;
        }

        if (j == M)
        {
              printf("Found pattern at index %d \n", i-j);
              j = lps[j-1];
        }
        else if(pat[j] != txt[i])
        {
          if(j != 0)
              j = lps[j-1];
          else
              i = i+1;
        }
  }
}

int main()
{
 printf("\n ENTER THE TEXT    : ");
 gets(txt);
 printf("\n ENTER THE PATTERN : ");
 gets(pat);
 KMPSearch();
 return 0;
}
```

DATA STRUCTURE USED: one dimensional Array

TIME COMPLEXITY: O(n)

OUTPUT:



```
 ENTER THE TEXT    : manikandan

 ENTER THE PATTERN : anda
Found pattern at index 5

Process returned 0 (0x0)   execution time : 21.479 s
```

RESULT:

Thus a program to implement KMP algorithm was successfully executed and verified.

QUESTION:

We are given a sorted array A[] of n elements. We need to find if x is present in A or not. In binary search we always used middle element, here we will randomly pick one element in given range. Execute an algorithm for randomized binary search.

EX NO:3.4(C)	RANDOMISED BINARY SEARCH
DATE:07.03.2019	

AIM:

To Write a program to implement randomized binary search

PSEUDOCODE:

```
//Program: To implement randomised binary search
//Input:  sorted array
//Output: element found or not
If(key == A[mid])
        return 1
Else If(key < A[mid])
        return binarysearch(A,low,mid-1,key)
Else
        return binarysearch(A,mid+1,high,key)
```

SOURCE CODE:

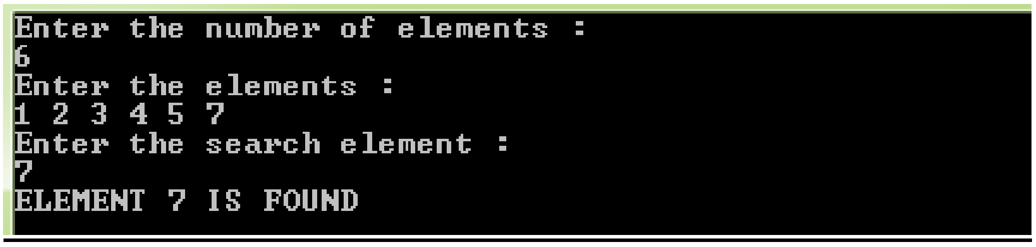
```
#include<stdio.h>
#include <stdlib.h>
int Randoms(int lower, int upper)
{
    int num = (rand() % (upper - lower + 1)) + lower;
   return num;
 }
 int main()
  {
   int binarysearch (int A[],int low,int high,int key);
    int n;
    printf("Enter the number of elements : \n");
    scanf("%d",&n);
    int A[n],i,key;
   printf("Enter the elements : \n");
    for(i=0;i<=n-1;i++)
           scanf("%d",&A[i]);
    printf("Enter the search element : \n");
    scanf("%d",&key);
    if(binarysearch(A,0,n-1,key))
     printf("ELEMENT %d IS FOUND\n",key);
   else
     printf("ELEMENT %d IS NOT FOUND\n",key);
  return 0;
  }
int binarysearch (int A[],int low,int high,int key)
  {
   int mid;
```

```
        srand(time(0));
        if(low <= high)
          {
              mid=Randoms(low,high);
              if(key == A[mid])
                  return 1;
              else if(key < A[mid])
                  return binarysearch(A,low,mid-1,key);
              else
                  return binarysearch(A,mid+1,high,key);
          }
      return 0;
    }
```

DATA STRUCTURE USED: sorted array

TIME COMPLEXITY: O(n)

OUTPUT:



```
Enter the number of elements :
6
Enter the elements :
1 2 3 4 5 7
Enter the search element :
7
ELEMENT 7 IS FOUND
```

RESULT:

Thus a program to implement randomised binary search algorithm was successfully executed and verified.