QUESTION:

Two cats and a mouse are at various positions on a line. You will be given their starting positions. Your task is to determine which cat will reach the mouse first, assuming the mouse doesn't move and the cats travel at equal speed. If the cats arrive at the same time, the mouse will be allowed to move and it will escape while they fight. You are given q queries in the form of , x , y and z representing the respective positions for cats A and B , and for mouse C . Complete the function cats and a mouse to return the appropriate answer to each query, which will be printed on a new line.

- If cat A catches the mouse first, print Cat A.
- If cat B catches the mouse first, print Cat B.
- If both cats reach the mouse at the same time, print Mouse C as the two cats fight and mouse escapes.

Input Format

The first line contains a single integer, q, denoting the number of queries. Each of the q subsequent lines contains three space-separated integers describing the respective values of x(cat A's location),y(cat B's location), and z (mouse C's location).

Output Format

For each query, return Cat A if cat A catches the mouse first, Cat B if cat B catches the mouse first, or Mouse C if the mouse escapes.

Sample Input 0

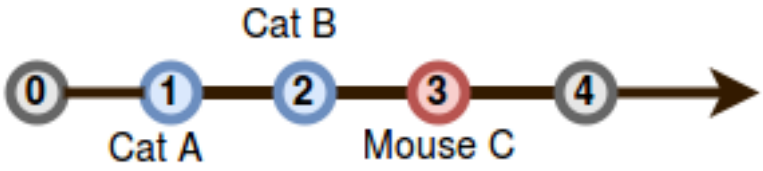
2
1 2 3
1 3 2

Sample Output 0

Cat B
Mouse C

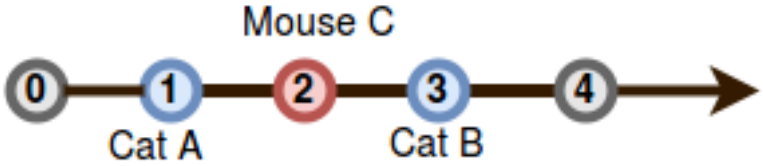
Explanation 0

*Query 0:*The positions of the cats and mouse are shown below:



Cat B will catch the mouse first, so we print Cat Bon a new line.

Query 1: In this query, cats A and Breach mouse Cat the exact same time:



Because the mouse escapes, we print Mouse Con a new line.

Ex. No: 1

Determination of length using absolute difference

Date:

AIM:

To determine which cat will reach the mouse first, assuming the mouse doesn't move and the cats travel at equal speed

PSEUDOCODE:

```
//Program: To determine whether mouse will escapes or not.
//Input: Position of the Cat A, Cat B, Mouse
//Output: Mouse caught or not. If it is the which cat caught it.
 BEGIN

  x=absolute difference(c-a)
  y= absolute difference (c-b)
  IF(x<y)
        PRINT  'Cat A'
  ELSE IF(x==y)
        PRINT  'Mouse'
  ELSE
        PRINT  'Cat B'

 END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
main()
{
int a,b,c,x,y;
printf("Position of catA\n");
scanf("%d",&a);
printf("Position of catB\n");
scanf("%d",&b);
printf("Position of Mouse\n");
scanf("%d",&c);
x=abs(c-a);
y=abs(c-b);
if(x<y)
        printf("cat A");
else if(x==y)
        printf("Mouse");
```

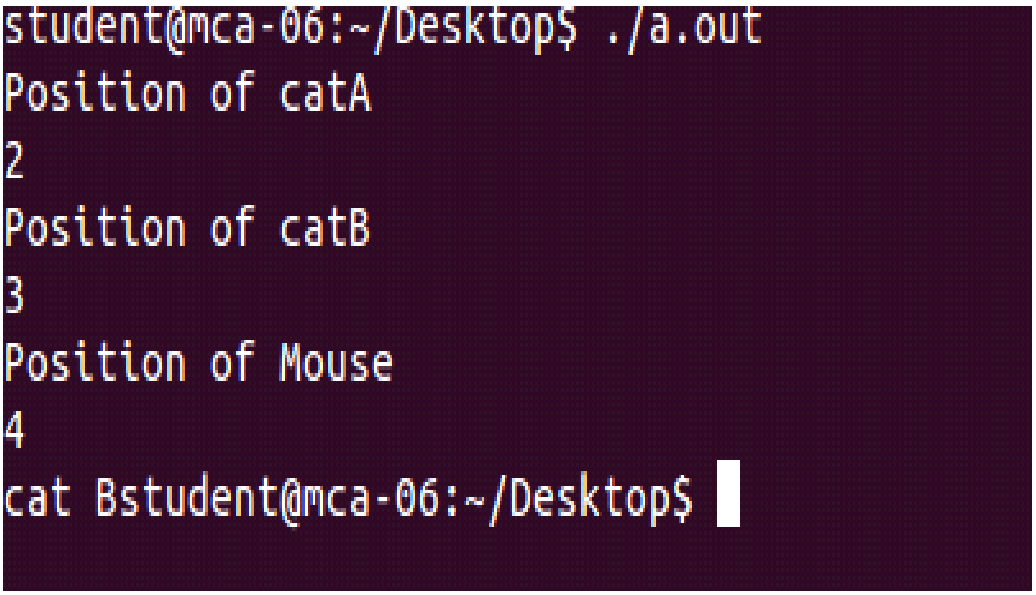
```
else
  printf("cat B");
}
```

DATA STRUCTURE USED: None

TIME COMPLEXITY: O(1)

ROUTINE: None

OUTPUT:



RESULT:

Thus the program that determines whether the mouse caught or escaped, if it is then who caught it .

QUESTION:

Given an array of integers, find and print the maximum number of integers you can select from the array such that the absolute difference between any two of the chosen integers is less than or equal to 1. For example, if your array is a=[1,1,2,2,4,4,5,5,5] you can create
two sub arrays meeting the criterion:[1,1,2,2]and[4,4,5,5,5]. The maximum length sub array has5elements.

Input Format

The first line contains a single integer n, the size of the array a .The second line contains n space-separated integers a[i].

Output Format

A single integer denoting the maximum number of integers you can choose from the array such tsshat the absolute difference between any two of the chosen integers is<=1.

Sample Input 0

6
4 6 5 3 3 1

Sample Output 0

3

Explanation 0

We choose the following multi set of integers from the array:{4,3,3}. Each pair in the multi set has an absolute difference< =1(i.e|4-3|=1and |3-3|=0), so we print the number of chosen integers,3, as our answer.

Sample Input 1

6
1 2 2 3 1 2

Sample Output1

5

Explanation 1

We choose the following multi set of integers from the array:{1,2,2,1,2}. Each pair in the multi set has an absolute difference <=1(i.e., |1-2| = 0, |1-1| = 0and |2-2|=0), so we print the number of chosen integers, 5, as our answer.

Ex. No: 2

Determine the maximum number of integers

Date:

AIM:

To find and print the maximum number of integers you select from the array such that the absolute difference between any two of the chosen integers is less than or equal to 1.

PSEUDOCODE:

```
//Program: To check whether the given number is present or not in the n*n matrix or not.
//Input: Size n, Array of size n
//Output: Maximum of the count
  BEGIN

   FOR  i←0 to  i<n
         h[arr[i]]+=1;
   FOR i←0 to i<9
         IF max<(h[i]+h[i+1])
                max=(h[i]+h[i+1])
   PRINT max

   END
```

SOURCE CODE:

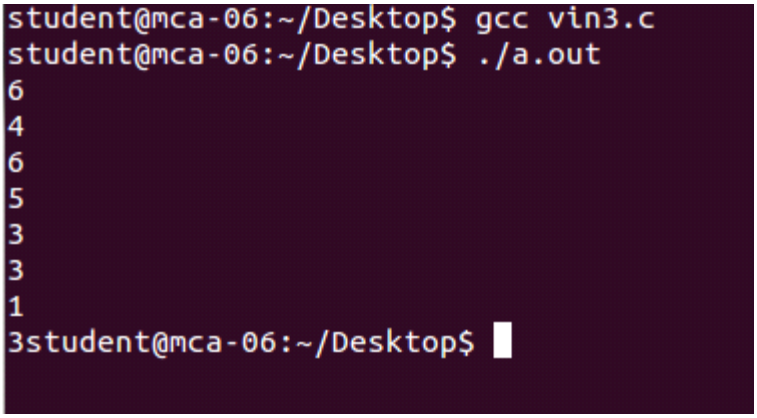
```
#include<stdio.h>
#include<stdlib.h>
main()
{
int h[10]={};
int n,i,max=0;
scanf("%d",&n);
int arr[n];
for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
for(i=0;i<n;i++)
        h[arr[i]]+=1;
for(i=0;i<9;i++)
      {
        if(max<(h[i]+h[i+1]))
                max=(h[i]+h[i+1]);
}
printf("%d",max);
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: $O(n^2)$

ROUTINE: Iterative

OUTPUT:



RESULT:

Thus the program to determine and print the maximum number of integers was successfully executed and verified.

QUESTION:

Numeros the Artist had two lists that were permutations of one another. He was very proud. Unfortunately, while transporting them from one exhibition to another, some numbers were lost out of the first list. Can you find the missing numbers?

Notes

- If a number occurs multiple times in the lists, you must ensure that the frequency of that number in both lists is the same. If that is not the case, then it is also a missing number.
- You have to print all the missing numbers in ascending order.
- Print each missing number once, even if it is missing multiple times.
- The difference between maximum and minimum number in the second list is less than or equal to 100.

Input Format

There will be four lines of input:

n - the size of the first list, arr
The next line contains n space-separated integers arr_i
m - the size of the second list, brr
The next line contains m space-separated integers brr_i

Constraints

- $1 \leq n, m \leq 2 \times 10^5$
- $n \leq m$
- $1 \leq x \leq 10^4, x \in B$
- $Xmax - Xmin < 101$

Output Format

Output the missing numbers in ascending order.

Sample Input

```
10
203 204 205 206 207 208 203 204 205 206
13
203 204 204 205 206 207 205 208 203 206 205 206 204
```

Ex. No: 3

Find the missing number in the given lists

Date:

AIM:

To write a c program to find the missing number in the given two lists.

PSEUDOCODE:

```
//Program: To find the missing number in the two list.
//Input: Two arrays
//Output: missing number in the given list.
  BEGIN

    FOR i←0 to i<n1
          FOR j←0 to j<n1
                IF(arr1[i]==arr1[j])
                      count1+=1
          FOR j←0  to j<n2
                IF(arr1[i]==arr2[j])
                      count2+=1
          IF(count1!=count2)
                IF(temp[t++]<arr1[i])
                      temp[k++]=arr1[i]
                ELSE
                      FOR i←0 to i<10
                            temp[i+1]=temp[i]
          count1=0
          count2=0
     PRINT  temp[0]
     FOR i←1 totemp[0]!=temp[i]
                PRINT temp[i]

  END
```

SOURCE CODE:

```
#include<stdio.h>
main()
{
 int n1,n2,i,j,count1=0,count2=0,k=0,t=0;
 scanf("%d%d",&n1,&n2);
 int arr1[n1],arr2[n2];
 int temp[10]={0};
 for(i=0;i<n1;i++)
          scanf("%d",&arr1[i]);
```

```
for(i=0;i<n2;i++)
        scanf("%d",&arr2[i]);
 for(i=0;i<n1;i++)
 {
          for(j=0;j<n1;j++)
          {
                  if(arr1[i]==arr1[j])
                          count1+=1;
          }
          for(j=0;j<n2;j++)
          {
                  if(arr1[i]==arr2[j])
                          count2+=1;
          }
          if(count1!=count2)
          {
                   if(temp[t++]<arr1[i])
                          temp[k++]=arr1[i];
                  else
                  {
                           for(i=0;i<10;i++)
                                   temp[i+1]=temp[i];
                  }
          }1
          count1=0;
          count2=0;
          }
          printf("%d ",temp[0]);
           for(i=1;temp[0]!=temp[i];i++)
                   printf("%d ",temp[i]);
          }
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: $O(n^3)$

ROUTINE: Iterative

OUTPUT:

RESULT:

Thus the program that to find the missing number in the given two lists was is successfully executed and verified.

QUESTION:

You have been given an integer array *A* and a number *K*. Now, you need to find out whether any two different elements of the array *A* sum to the number *K*. Two elements are considered to be different if they lie at different positions in the array. If there exists such a pair of numbers, print "**YES**" (without quotes), else print "**NO**" without quotes.

Input Format:

The first line consists of two integers *N*, denoting the size of array *A* and *K*. The next line consists of *N* space separated integers denoting the elements of the array *A*.

Output Format:

Print the required answer on a single line.

Constraints:

$1 \leq N \leq 106$

$1 \leq K \leq 2 * 106$

$1 \leq A[i] \leq 106$

SAMPLE INPUT

5 9

1 2 3 4 5

SAMPLE OUTPUT

YES

Ex. No: 4

Sum of elements of the array is the number or not

Date:

AIM:

To write a c program to find out whether any two different elements of the array , sum to the number.

PSEUDOCODE:

```
//Program: To find the two different elements of the array, sum to the number.
//Input: Array size, array and the number
//Output: YES or NO
  BEGIN

   FOR i←0 to i<n
            FOR j←i+1 to j<n
                    if(arr[i]+arr[j]==k)
                            count++
    IF(count>0)
          Print "YES"
    ELSE
          Printf  "NO"

   END
```

SOURCE CODE:

```
#include<stdio.h>
main()
{
 int n,k,i,j,count=0;
 printf("Enter the array size : ");
 scanf("%d",&n);
 printf("Enter the sum value : ");
 scanf("%d",&k);
 int arr[n];
 printf("Enter the array elements : ");
 for(i=0;i<n;i++)
          scanf("%d",&arr[i]);
 for(i=0;i<n;i++)
 {
          for(j=i+1;j<n;j++)
          {
                  if(arr[i]+arr[j]==k)
                          count++;
```

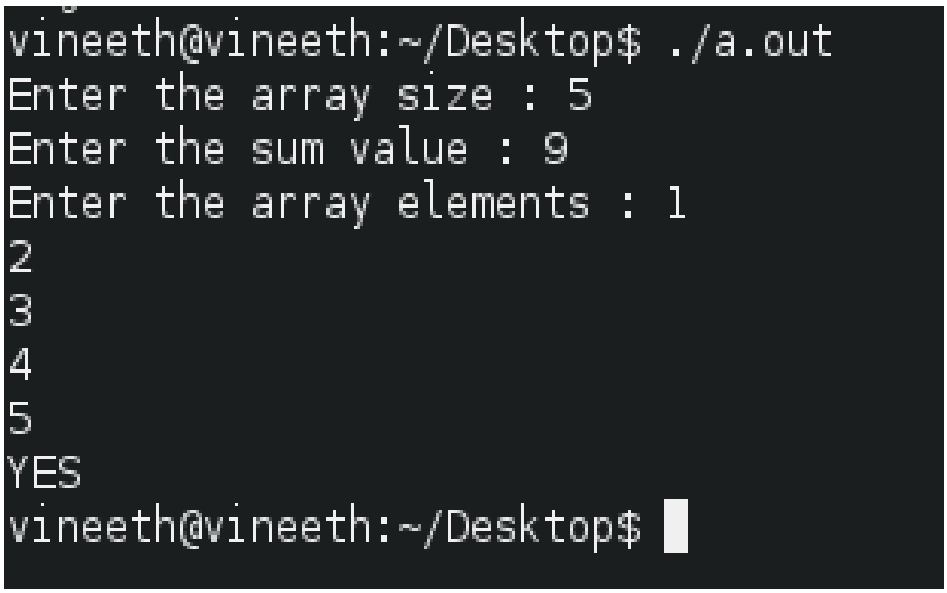
```
            }
        }
        if(count>0)
            printf("YES\n");
        else
            printf("NO\n");
    }
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: $O(n^2)$

ROUTINE: Iterative

OUTPUT:



RESULT:

Thus the program that to find out whether any two different elements of the array , sum to the number was successfully executed and verified.

QUESTION:

There is a class consisting of **'N'** students . There can be many students with the same name.

Now, you have to print the names of the students followed by there frequency as shown in the sample explanation given below.

Output the names in the **lexicographical order.**

Input :

First line contains an integer **'N'**, i.e the no. of students in the class.

Next 'N' lines contains the names of the students.

Output:

Each line consists of the name of student space and separated its frequency.

Constraints:

1<=N<=1000

string length<=100

string consists of lowercase letters

Note : For practicing use Map technique only .

SAMPLE INPUT

5 sumit
ambuj
himanshu
ambuj
ambuj

SAMPLE OUTPUT

ambuj 3
himanshu 1
sumit 1

Ex. No: 5

Printing names in the lexicographical order

Date:

AIM:

To write a c program to print the given names in lexicographical order along with its count.

PSEUDOCODE:

```
//Program: To print the given names in lexicographical order and its count.
//Input: No of names to be given.
//Output: Printing names in an order.
  BEGIN

        Check(a,b)
        if(a<b)
                return 1
        else if(a>b)
                return 0
        else
                return 99
        Main()
                Read n
                FOR(i←0 to i<n)
                        Read(a[i])
                FOR(i←0 to i←n-1)
                        FOR(j←i+1 to j<n)
                                        O←check(a[i][z],a[j][z]);
                                        While(1)
                                                IF(o==0)
                                                        t←a[i]
                                                        a[i]←a[j]
                                                        a[j]←t
                                                        break
                                                ELSE IF(o==99)
                                                        z←z+1
                                                        o←check(a[i][z],a[j][z])
                                                ELSE
                                                        Break
                                        z=0
        FOR(i←0 to i←n)
                        b[i]←a[i]
        FOR(i←0 to i<n)
                FOR(j←0 to j<n)
                        IF(a[i]==b[j])
```

```
                    b[j][0]←'*'
                    u←u+1
                IF(u>0)
                    Display a[i],u
                    u=0

    END
```

SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
int check(char a,char b)
{
if(a<b)
    return 1;
else if(a>b)
    return 0;
else
    return 99;
}
void main()
{
    int n,i,k,j,z=0,u=0,o;
    printf("ENTER THE NO OF NAMES:");
    scanf("%d",&n);
    char a[n][20],b[n][20],t[20];
    for(i=0;i<n;i++)
        {
            scanf("%s",a[i]);
        }
    for(i=0;i<n-1;i++)
        {
            for(j=i+1;j<n;j++)
            {
                o=check(a[i][z],a[j][z]);
                while(1)
                {
                    if(o==0)
                    {
                        strcpy(t,a[i]);
                        strcpy(a[i],a[j]);
                        strcpy(a[j],t);
                        break;
                    }
                    else if(o==99)
                    {
```

```
                                        z=z+1;
                                        o=check(a[i][z],a[j][z]);
                                }
                                else
                                break;
                        }
                        z=0;
                }
        }
    for(i=0;i<n;i++)
                strcpy(b[i],a[i]);
    printf("The count of names in lexicographical order is:\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
                if(strcmp(a[i],b[j])==0)
                {
                        b[j][0]='*';
                        u+=1;
                }
        }
        if(u>0)
        {
                printf("%d-",u);
                puts(a[i]);
                printf("\n");
                u=0;
        }
    }
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: $O(n^2)$

ROUTINE: Iterative

OUTPUT:

```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 1.3a.c
vineeth@vineeth:~/Desktop$ ./a.out
ENTER THE NO OF NAMES:5
sumit
ambuj
himanushu
ambuj
ambuj
The count of names in lexicographical order is:
3-ambuj

1-himanushu

1-sumit

vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program that to print the names in lexicographical order is was successfully executed and verified.

QUESTION:
After Governor's attack on prison, Rick found himself surrounded by walkers. They are coming towards him from all sides. Now, suppose Rick have infinite number of bullets with him. Suppose Rick need 1 bullet to kill each walker (yeah he is good in killing walkers. They need to be shot at head. See, how good he is). **Now as soon as he kills 1 walker rest of the walkers move forward by 1 m**. There are n walkers each at some distance from Rick. If any walker is able to reach Rick, he dies. Now, you need to tell if he survives or not. If he survives print "**Rick now go and save Carl and Judas**" else print "**Goodbye Rick**" followed by no of walkers he was able to kill before he died in next line. One more thing **Rick's gun can fire 6 shots without reload. It takes him 1 sec to reload and during this time walkers move 1 m forward.**

[**Input**]
First line contains an integer t indicating number of test cases.
Next line contains an integer n denoting no.of walkers followed by n space separated integers denoting the distance of walkers from him.

[**Output**]
For each test case output one line denoting the answer as explained above.

[**Constraints**]
1<=t<=100
1<=n<=100000
1<=dis[i]<=50000

SAMPLE INPUT
2
5
2 4 2 5 6
4
2 2 2 2

SAMPLE OUTPUT
Rick now go and save Carl and Judas
Goodbye Rick

Ex. No: 6

Checking the scenario that Rick died or not

Date:

AIM:

To write a c program to check the scenario that Rick died or not.

PSEUDOCODE:

```
//Program: To check the scenario that Rick died or not.
//Input: No of walkers and distance of the walkers from Rick.
//Output: Died or not
 BEGIN

        Read n
        C←n
        FOR(i←0 to i<n)
                Read(a[i])
        FOR(i←0 to i<n-1)
                 FOR(j←i+1 to j<n)
                        IF(a[i]>a[j])
                                k←a[i]
                                a[i]←a[j]
                                a[j]←k
        FOR(i←0 to i<n)
                IF(a[0]==0)
                        Print("Goodbye Rick\n")
                        y←y+1
                        break
                ELSE
                        FOR(j←0 to j<c)
                                a[j]←a[j+1]-1
                                c←c-1
        IF(y==0)
                Print("Rick now go and save Carl and Judas\n")
 END
```

SOURCE CODE:

```
#include<stdio.h>
void main()
{
int n,i,j,k,y=0;
printf("Enter the no of walkers:");
scanf("%d",&n);
int a[n],c=n;
printf("Enter the distance of each walkers\n");
```

```
for(i=0;i<n;i++)
        scanf("%d",&a[i]);
for(i=0;i<n-1;i++)
{
         for(j=i+1;j<n;j++)
        {
                if(a[i]>a[j])
                {
                        k=a[i];
                        a[i]=a[j];
                        a[j]=k;
                }
        }
}
for(i=0;i<n;i++)
{
        if(a[0]==0)
        {
                printf("Goodbye Rick\n");
                y+=1;
                break;
        }
        else
        {
        for(j=0;j<c;j++)
                a[j]=a[j+1]-1;
        c-=1;
        }
}
if(y==0)
printf("Rick now go and save Carl and Judas\n");
}
```

DATA STRUCTURE USED: Array

TIME COMPLEXITY: $O(n^2)$

ROUTINE: Iterative

OUTPUT:

```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 1.3b.c
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of walkers:5
Enter the distance of each walkers
2
4
2
5
6
Rick now go and save Carl and Judas
vineeth@vineeth:~/Desktop$ ./a.out
Enter the no of walkers:4
Enter the distance of each walkers
2
2
2
2
Goodbye Rick
vineeth@vineeth:~/Desktop$
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program for the scenario that to find whether Rick died or not was successfully executed and verified.

QUESTION:
Monk is standing at the door of his classroom. There are currently **N** students in the class, **i**'th student got **Ai** candies.
There are still **M** more students to come. At every instant, a student enters the class and wishes to be seated with a student who has **exactly** the same number of candies. For each student, Monk shouts YES if such a student is found, NO otherwise. Help Monk in this problem using threaded binary search tree.
Input:
First line contains an integer **T**. **T** test cases follow.
First line of each case contains two space-separated integers **N** and **M**.
Second line contains **N** + **M** space-separated integers, the candies of the students.
Output:
For each test case, output **M** new line, Monk's answer to the **M** students.
Print "YES" (without the quotes) or "NO" (without the quotes) pertaining to the Monk's answer.
Constraints:
$1 \leq \mathbf{T} \leq 10$
$1 \leq \mathbf{N}, \mathbf{M} \leq 105$
$0 \leq \mathbf{Ai} \leq 1012$
SAMPLE INPUT
1
2 3
3 2 9 11 2
SAMPLE OUTPUT
NO
NO
YES
Explanation
Initially students with 3 and 2 candies are in the class.
A student with 9 candies enters, No student with 9 candies in class. Hence, "NO"
A student with 11 candies enters, No student with 11 candies in class. Hence, "NO"
A student with 2 candies enters, Student with 2 candies found in class. Hence, "YES"

Ex. No: 7

Checking whether the element is already there in a tree or not

Date:

AIM:

To write a c program to check whether the element is there in the tree or not.

PSEUDOCODE:

```
//Program: To to check whether the element is there in the tree or not.
//Input: No of initial students, and no of students to come and their no of candies.
//Output: YES or NO
 BEGIN

Insert (tree, elem)
        nn->lf←nn->rf←1
        nn->data←elem
        IF(tree==NULL)
                nn->lc←nn->rc←NULL;
                tree←n
                IF(w>n)
                        Print "YES"
        ELSE
                temp←tree
                While(temp!=NULL)
                        If(elem<temp->data)
                                If(temp->lf==0)
                                        temp←temp->lc;
                                Else
                                        nn->lc←NULL
                                        nn->rc←emp
                                        temp->lf←0
                                        temp->lc←nn
                                        if(w>n)
                                                Print "YES"
                                        break
                        Else if(elem>temp->data)
                                If(temp->rf==0)
                                        temp←temp->rc;
                                Else
                                        nn->lc←NULL
                                        nn->rc←temp->rc
                                        temp->rf←0
                                        temp->rc←nn
                                        if(w>n)
                                                Print "YES"
```

```
                                        break
                Else
                        If(w>n)
                                Print "NO"
                                break
        return tree;
main()
        root←NULL
        Read n
        Read m
        FOR(i←0 to i<m+n)
                Read elem[i]
        FOR(i←0 to i<m+n)
                root=insert(root,elem[i])
                w+=1

  END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int n,w=0;
typedef struct node TBST;
struct node
{
        int data,lf,rf;
        struct node *lc,*rc;
};
TBST* insert(TBST *tree,int elem)
{
TBST *nn,*temp;
nn=(TBST*)malloc(sizeof(TBST));
nn->lf=nn->rf=1;
nn->data=elem;
if(tree==NULL)
{
        nn->lc=nn->rc=NULL;
        tree=nn;
        if(w>n)
                printf("YES\n");
}
else
{
        temp=tree;
        while(temp!=NULL)
        {
```

```
                if(elem<temp->data)
                {
                        if(temp->lf==0)
                                temp=temp->lc;
                        else
                        {
                                nn->lc=NULL;
                                nn->rc=temp;
                                temp->lf=0;
                                temp->lc=nn;
                                if(w>n)
                                        printf("YES\n");
                                break;
                        }
                }
                else if(elem>temp->data)
                {
                        if(temp->rf==0)
                                temp=temp->rc;
                        else
                        {
                                nn->lc=NULL;
                                nn->rc=temp->rc;
                                temp->rf=0;
                                temp->rc=nn;
                                if(w>n)
                                        printf("YES\n");
                                break;
                        }
                }
                else
                {
                        if(w>n)
                                printf("NO\n");
                        break;
                }
        }
}
return tree;
}
void main()
{
TBST *root=NULL;
int i,m,x;
printf("No students in the class room intially\n");
scanf("%d",&n);
printf("Enter the no of students to come\n");
```

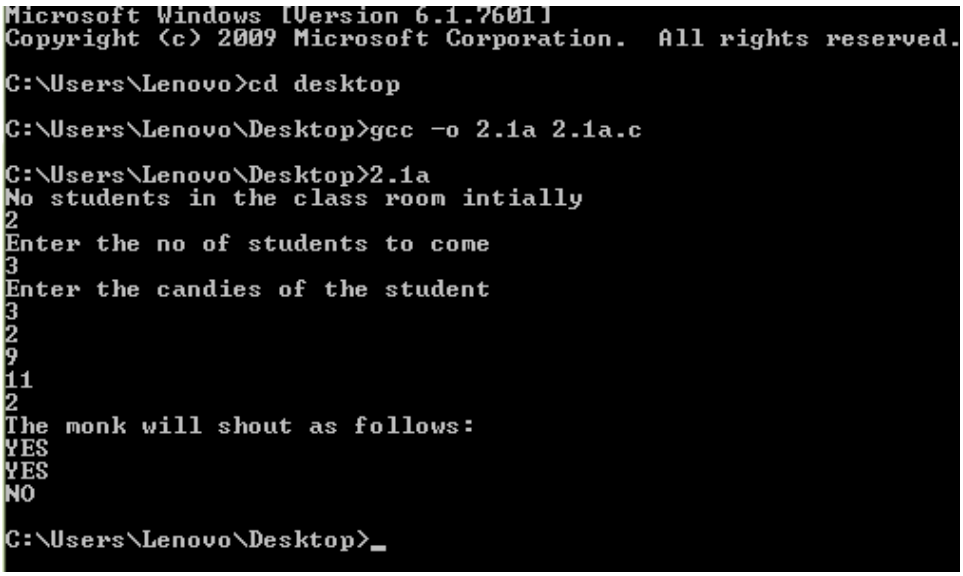
```
scanf("%d",&m);
int elem[m+n];
printf("Enter the candies of the student\n");
for(i=0;i<m+n;i++)
        scanf("%d",&elem[i]);
printf("The monk will shout as follows:\n");
for(i=0;i<m+n;i++)
{
        root=insert(root,elem[i]);
        w+=1;
}
}
```

DATA STRUCTURE USED: Threaded binary tree

TIME COMPLEXITY: O(log n)

ROUTINE: Recursive

OUTPUT:



RESULT:

Thus the program that displaying what monk said is successfully executed and verified.

QUESTION:
The height of a binary tree is the number of edges between the tree's root and its furthest leaf. For example, the following binary tree is of height 2:

Function Description

Complete the *get Height* or *height* function in the editor. It must return the height of a binary tree as an integer.

Get Height or height has the following parameter(s):

 root: a reference to the root of a binary tree.

Note -The Height of binary tree with single node is taken as zero.

Input Format

The first line contains an integer n, the number of nodes in the tree.

Next line contains n space separated integer where ith integer denotes node[i].data.

Note: Node values are inserted into a binary search tree before a reference to the tree's root node is passed to your function. In a binary search tree, all nodes on the left branch of a node are less than the node value. All values on the right branch are greater than the node value.

Constraints

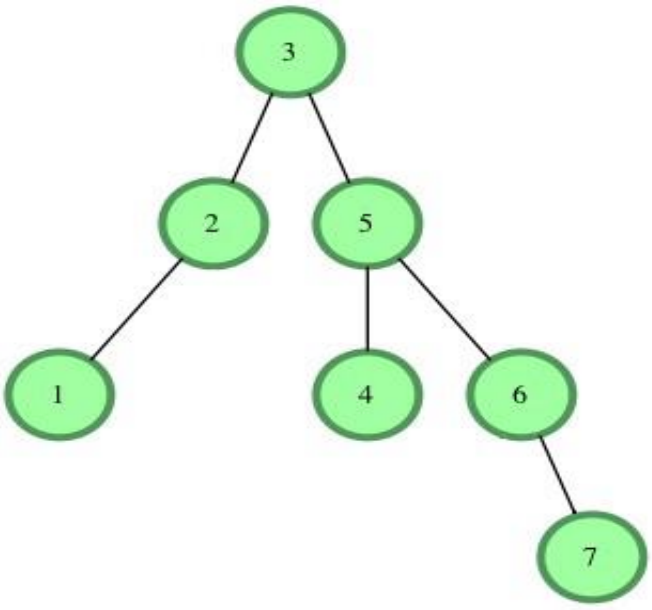
1<=node.data[i]<=20

1<=n<=20

Output Format

Your function should return a single integer denoting the height of the binary tree.

Sample Input



Sample Output

3

Explanation

The longest root-to-leaf path is shown below:

There are 4 nodes in this path that are connected by 3 edges, meaning our binary tree's height = 3.

Ex. No: 8

Finding the height of the tree

Date:

AIM:

To write a c program to find height of the tree.

PSEUDOCODE:

```
//Program: To find the height of the tree.
//Input: No of nodes in the tree, elements in the tree.
//Output: height of the tree
BEGIN

Height(temp)
        If(temp==NULL)
                return -1
        Else
                lh←height(temp->left)
                rh←height(temp->right)
                If(lh>rh)
                        return (lh+1)
                Else
                        return (rh+1)
Insert(root, ele)
        If(root==NULL)
                root←(struct node*)malloc(sizeof(struct node))
                root->data←ele
                root->left←NULL
                root->right←NULL
        Else if(ele<root->data)
                root->left←insert(root->left,ele)
        Else if(ele>root->data)
                root->right←insert(root->right,ele)
        return (root)
 main()
        Read n
        FOR(i←0 to i<n)
                Read ele[i]
                root←insert(root,ele[i])
        Print root


 END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
int height(struct node *temp)
{
int lh,rh;
if(temp==NULL)
        return -1;
else
{
        lh=height(temp->left);
        rh=height(temp->right);
        if(lh>rh)
                return (lh+1);
        else
                return (rh+1);
}
}
struct node *insert(struct node *root,int ele)
{
if(root==NULL)
{
        root=(struct node*)malloc(sizeof(struct node));
        root->data=ele;
        root->left=NULL;
        root->right=NULL;
}
else if(ele<root->data)
        root->left=insert(root->left,ele);
else if(ele>root->data)
        root->right=insert(root->right,ele);
return (root);
}
void main()
{
int i,n,k;
printf("ENTER THE NUMBER OF ELEMENTS\n");
scanf("%d",&n);
int ele[n];
printf("ENTER THE ELEMENTS\n");
for(i=0;i<n;i++)
```

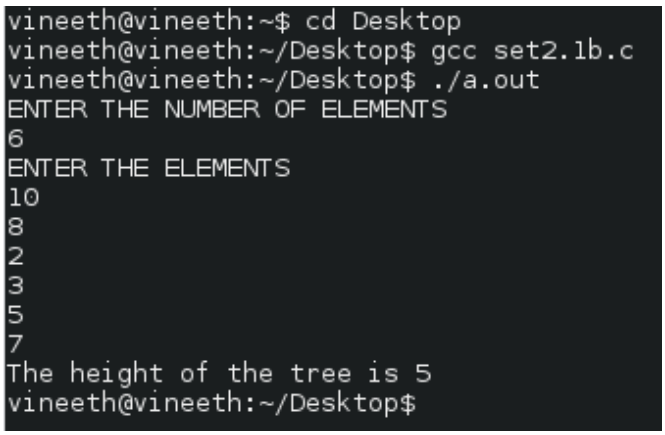
```
{
        scanf("%d",&ele[i]);
        root=insert(root,ele[i]);
}
printf("The height of the tree is %d\n",height(root));
}
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

OUTPUT:



RESULT:

Thus the program that to find the height of the tree was successfully executed and verified.

QUESTION:

After a hectic week at work, Mancunian and Liverbird decide to go on a fun weekend camping trip. As they were passing through a forest, they stumbled upon a unique tree of *N* nodes. Vertices are numbered from *1* to *N*.

Each node of the tree is assigned a value x, which is the difference in height of left sub tree and right sub tree. Being bored, they decide to work together (for a change) and test their reasoning skills. The tree is rooted at vertex *1*. For any particular node, they want to find its closest ancestor.

Input format

The first line contains two integers *N* denoting the number of vertices in the tree The second line contains N integers. The integer denotes the value of each vertex.

The third line contains a vertex for which the closest ancestor is to be found.

Output format

Print the closest ancestor of the given vertex.

SAMPLE INPUT

5

10 6 11 4 7

7

SAMPLE OUTPUT

6

Ex. No: 9

Finding the ancestor

Date:

AIM:

To write a c program to find ancestor of the given node.

PSEUDOCODE:

```
//Program: To find the ancestor.
//Input: No of nodes in the tree, elements in the tree.
//Output: least common ancestor.
BEGIN

Insert(root, ele)
        If(root==NULL)
                root->data←ele
                root->left←NULL
                root->right←NULL
        Else If(ele<root->data)
                root->left←Insert(root->left,ele)
        Else If(ele>root->data)
                root->right←Insert(root->right,ele);
        return root
Search(temp, ele)
        If(temp==NULL)
                return temp
        Else
                If(ele<temp->data)
                        return Search(temp->left,ele)
                Else if(ele>temp->data)
                        return Search(temp->right,ele)
                Else
                        return temp
Main()
        Read n
        For(i←0 to i<n)
                Read ele[i]
        root=Insert(root,ele[i])
        temp←root
        Read num
        addr←Search(root,num)
        While(1)
                If(addr==temp->right  || addr==temp->left)
                        break
                Else if(num>temp->data)
```

```
                    temp←temp->right
                Else if(num<temp->data)
                    temp=temp->left
        Display temp->data, num


    END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int ele)
{
    if(root==NULL)
    {
        root=(struct node*)malloc(sizeof(struct node));
        root->data=ele;
        root->left=NULL;
        root->right=NULL;
    }
    else if(ele<root->data)
        root->left=insert(root->left,ele);
    else if(ele>root->data)
        root->right=insert(root->right,ele);
    return root;
}
struct node *search(struct node *temp,int ele)
{
    if(temp==NULL)
        return temp;
    else
    {
        if(ele<temp->data)
            return search(temp->left,ele);
        else if(ele>temp->data)
            return search(temp->right,ele);
        else
            return temp;
    }
}
void main()
```

```
{
        int i,n,k,num;
        printf("ENTER THE NUMBER OF ELEMENTS\n");
        scanf("%d",&n);
        int ele[n];
        struct node *addr,*temp;
        for(i=0;i<n;i++)
        {
                printf("ENTER THE ELEMENT\n");
                scanf("%d",&ele[i]);
                root=insert(root,ele[i]);
        }
        temp=root;
        printf("ENTER THE ELEMENT TO BE SEARCHED\n");
        scanf("%d",&num);
        addr=search(root,num);
        while(1)
        {
                if(addr==temp->right  || addr==temp->left)
                        break;
                else if(num>temp->data)
                        temp=temp->right;
                else if(num<temp->data)
                        temp=temp->left;
        }
        printf("%d is before vertex of  %d\n",temp->data,num);
}
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

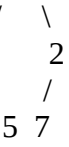
OUTPUT:

```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc ses2.2a.c
vineeth@vineeth:~/Desktop$ ./a.out
ENTER THE NUMBER OF ELEMENTS
5
ENTER THE ELEMENT
10
ENTER THE ELEMENT
6
ENTER THE ELEMENT
11
ENTER THE ELEMENT
4
ENTER THE ELEMENT
7
ENTER THE ELEMENT TO BE SEARCHED
7
6 is before vertex of  7
vineeth@vineeth:~/Desktop$
```

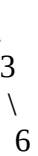
RESULT:

Thus the program that to find the ancestor of the node was successfully executed and verified.

QUESTION:
Given a binary tree, check if each non-leaf node has only one child. i.e print each non-leaf nodes which has either left or right children as non-empty.
Examples:

Input :


Output : 2

Input :


Output : 2 3

Ex. No: 10

Printing the node with single child

Date:

AIM:

To write a c program to print the node with child.

PSEUDOCODE:

```
//Program: To print the node with single child.
//Input: No of nodes in the tree, elements in the tree.
//Output: Node with single child.
BEGIN

Insert(root, ele)
        If(root==NULL)
                root->data←ele
                root->left←NULL
                root->right←NULL
        Else If(ele<root->data)
                root->left←Insert(root->left,ele)
        Else If(ele>root->data)
                root->right←Insert(root->right,ele);
        return root
Oneleafchild(temp)
        If(temp!=NULL)
                If((temp->left==NULL&&temp->right!=NULL)||(temp->
                                        left!=NULL&&temp->right==NULL))
                        Print temp->data
                        Oneleafchild(temp->left)
                        Oneleafchild(temp->right)
Main()
        Read n
        For(i←0 to i<n)
                ele[i]
                root=Insert(root,ele[i])
        Oneleafchild(root)

END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
```

```
        int data;
        struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int ele)
{
        if(root==NULL)
        {
                root=(struct node*)malloc(sizeof(struct node));
                root->data=ele;
                root->left=NULL;
                root->right=NULL;
        }
        else if(ele<root->data)
                root->left=insert(root->left,ele);
        else if(ele>root->data)
                root->right=insert(root->right,ele);
        return root;
}
void oneleafchild(struct node *temp)
{
        if(temp!=NULL)
        {
                 if((temp->left==NULL&&temp->right!=NULL)||(temp->
                                                left!=NULL&&temp->right==NULL))
                        printf("%d\n",temp->data);
                oneleafchild(temp->left);
                oneleafchild(temp->right);
        }
}
void main()
{
        int ele[40],i,n,k;
        printf("ENTER THE NUMBER OF ELEMENTS\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("ENTER THE ELEMENT\n");
                scanf("%d",&ele[i]);
                root=insert(root,ele[i]);
        }
        printf("The nodes with one child are:\n");
        oneleafchild(root);
}
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

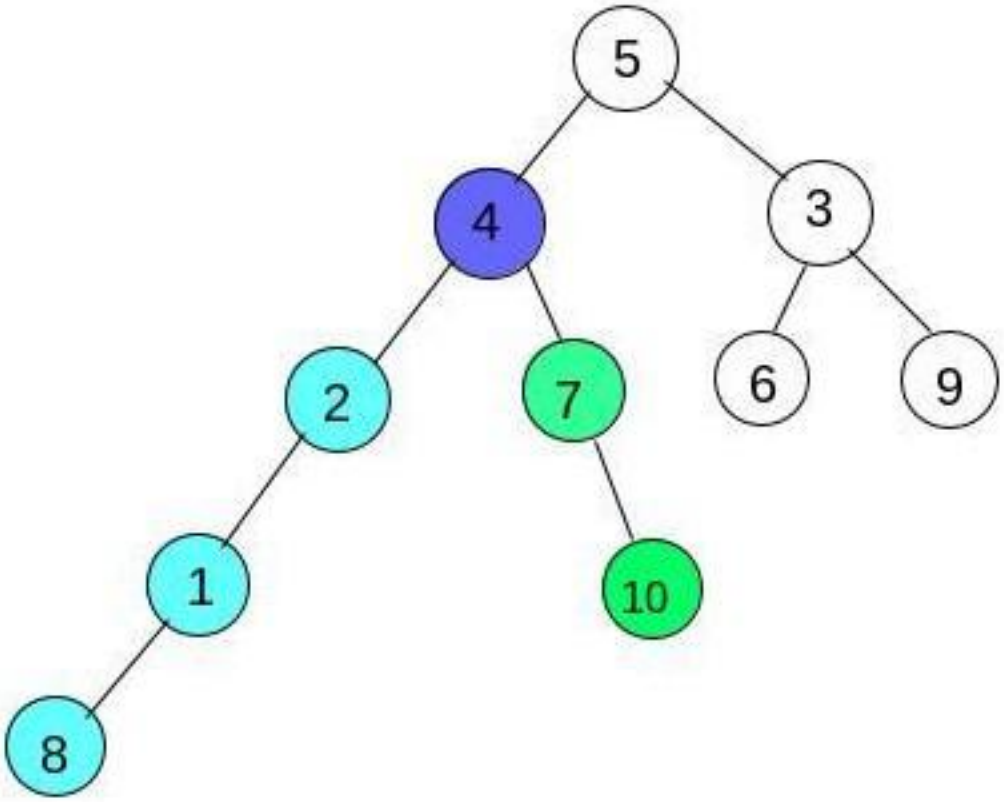
OUTPUT:

```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 2.2b.c
vineeth@vineeth:~/Desktop$ ./a.out
ENTER THE NUMBER OF ELEMENTS
9
ENTER THE ELEMENT
10
ENTER THE ELEMENT
9
ENTER THE ELEMENT
11
ENTER THE ELEMENT
6
ENTER THE ELEMENT
15
ENTER THE ELEMENT
5
ENTER THE ELEMENT
7
ENTER THE ELEMENT
12
ENTER THE ELEMENT
16
The nodes with one child are:
9
11
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program that to print the node with single child was successfully executed and verified.

QUESTION:
The problem is you have to find a path between two nodes in an undirected tree. And the approach I came up with is find the lca(lowest common ancestor) of node 'A' and 'B' and then print the path from node 'A' to lca and then from the next node after lca to node 'B'.



For example consider the above picture, you can see that node A = 8, node B = 10,
lca(A, B) = 4.
I can find lca(A, B) using sparse table but the problem is how can i find path from node 'A' to lca and the path from node 'B' to lca. I know i can do a dfs for finding the path but thats brute force i need a more efficient solution. What i want to say is that i need an algorithm that will only check path 8->1->2->->4->5 and not 4->7->10 if node A = 8, node B = 9 & lca(A, B) = 5.

Ex. No: 11

Finding the least common ancestor

Date:

AIM:

To write a c program to find the least common ancestor.

PSEUDOCODE:

```
//Program: To find the least common ancestor.
//Input: No of nodes in the tree, elements in the tree.
//Output: least common ancestor.
BEGIN

Insert(root, ele)
        If(root==NULL)
                root->data←ele
                root->left←NULL
                root->right←NULL
        Else If(ele<root->data)
                root->left←Insert(root->left,ele)
        Else If(ele>root->data)
                root->right←Insert(root->right,ele);
        return root
Search(temp, element, k[])
         y←0
        If(temp==NULL)
                return 0
        Else
                If(element<temp->data)
                        k[y++]←temp->data
                        return search(temp->left,element,k)
                Else if(element>temp->data)
                        k[y++]←temp->data
                        return search(temp->right,element,k)
                Else
                        return 0
        return 0
Main()
         Read n
        For(i←0 to i<n)
                Read arr[i]
        For(i←0 to i<n)
                element←arr[i]
                root←Insert(root,element)
        Read A,B
```

```
        Search(root,A,a);
        Search(root,B,b)
        For(i←0 to i<n)
                For(j←0 to j<n)
                        if(a[i]==b[j])
                                c[t++]=a[i]
        For(i←0 to i<n)
                For(j←i+1 to j<n)
                        If(a[i]>a[j])
                                nt=a[i]
                                a[i]=a[j]
                                a[j]=nt
        Print  c[0]


END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int element,y=0;
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int element)
{
        if(root==NULL)
          {
                root=(struct node*)malloc(sizeof(struct node));
                root->data=element;
                root->left=NULL;
                root->right=NULL;
          }
         else if(element<root->data)
                root->left=insert(root->left,element);
         else if(element>root->data)
                root->right=insert(root->right,element);
         return root;
}
int search(struct node *temp,int element,int k[])
{
        int y=0;
        if(temp==NULL)
                return 0 ;
```

```
        else
        {
                if(element<temp->data)
                {
                        k[y++]=temp->data;
                        return search(temp->left,element,k);
                }
                else if(element>temp->data)
                {
                        k[y++]=temp->data;
                        return search(temp->right,element,k);
                }
                else
                        return 0;
        }
        return 0;
}
void main()
{
        int n,inum,A,B;
        printf("ENTER SIZE...\n");
        scanf("%d",&n);
        int a[n],b[n],c[n];
        int i,arr[n],j,t=0,nt;
        printf("\nENTER element...\n");
        for(i=0;i<n;i++)
                scanf("%d",&arr[i]);
        for(i=0;i<n;i++)
        {
                element=arr[i];
                root=insert(root,element);
        }
        int elem;
        printf("ENTER THE SEARCH ELEMENTS...\n");
        scanf("%d%d",&A,&B);
        search(root,A,a);
        search(root,B,b);
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        if(a[i]==b[j])
                                c[t++]=a[i];
                }
        }
        for(i=0;i<n;i++)
        {
```

```
                for(j=i+1;j<n;j++)
                {
                        if(a[i]>a[j])
                        {
                                nt=a[i];
                                a[i]=a[j];
                                a[j]=nt;
                        }
                }
        }
        printf("THE LEAST COMMON ANCESTOR IS : %d\n",c[0]);
}
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

OUTPUT:

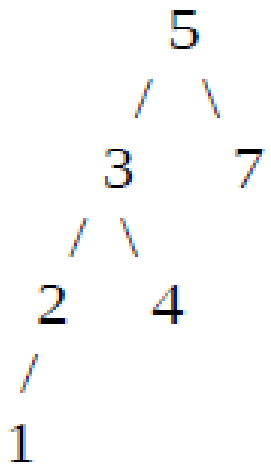
```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 2.3a.c
vineeth@vineeth:~/Desktop$ ./a.out
ENTER SIZE...
9

ENTER element...
10
9
11
6
15
5
7
12
16
ENTER THE SEARCH ELEMENTS...
5
7
THE LEAST COMMON ANCESTOR IS : 6
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program that to print the least common ancestor was successfully executed and verified.

QUESTION:
Given a Binary Search Tree and a key, write a function that prints all the ancestors of the key in the given binary search tree.
For example, if the given tree is following Binary Search Tree and key is 1, then your function should print 2, 3 and 5



Ex. No: 12

Printing the path in reverse order

Date:

AIM:

To write a c program to print the path of the given node in reverse order.

PSEUDOCODE:

```
//Program: To print the path in reverse order.
//Input: No of nodes in the tree, elements in the tree.
//Output: path of the node.
BEGIN

  Insert(root, ele)
        If(root==NULL)
              root->data←ele
              root->left←NULL
              root->right←NULL
        Else If(ele<root->data)
              root->left←Insert(root->left,ele)
        Else If(ele>root->data)
              root->right←Insert(root->right,ele);
        return root

  Ancestor(temp, element)
        If(temp->data==element)
              return
        Else
              If(element<temp->data)
                    Ancestor(temp->left,element)
                    Print temp->data
              Else
                    Ancestor(temp->right,element)
                    Print temp->data
  Main()
        Read n
        For(i←0 to i<n)
              Read arr[i]
              element←arr[i]
              root←insert(root,element)
        Read A
        Ancestor(root,A)

END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int element;
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int element)
{
        if(root==NULL)
          {
                root=(struct node*)malloc(sizeof(struct node));
                root->data=element;
                root->left=NULL;
                root->right=NULL;
          }
         else if(element<root->data)
                root->left=insert(root->left,element);
         else if(element>root->data)
                root->right=insert(root->right,element);
         return root;
}
void ancestor(struct node *temp,int element)
{
        if(temp->data==element)
                return ;
        else
        {
                if(element<temp->data)
                {
                        ancestor(temp->left,element);
                        printf(" -> ");
                        printf("%d",temp->data);
                }
                else
                {
                        ancestor(temp->right,element);
                        printf(" -> ");
                        printf("%d",temp->data);
                }
        }
}
void main()
{
```

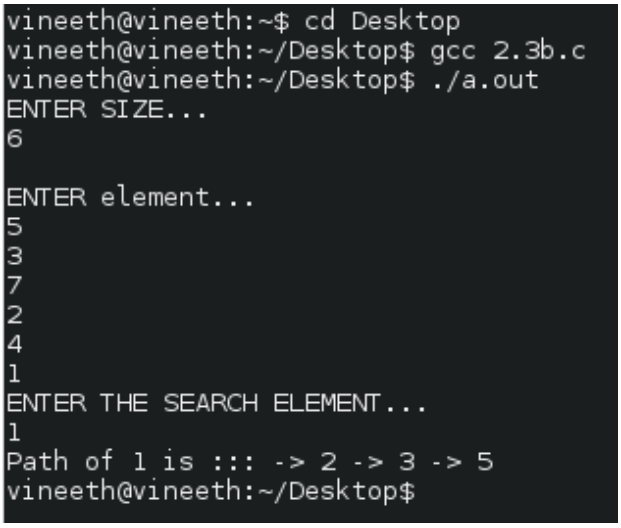
```
        int n,A,i;
        printf("ENTER SIZE...\n");
        scanf("%d",&n);
        int arr[n];
        printf("\nENTER element...\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
                element=arr[i];
                root=insert(root,element);
        }
        printf("ENTER THE SEARCH ELEMENT...\n");
        scanf("%d",&A);
        printf("Path of %d is :::",A);
        ancestor(root,A);
        printf("\n");
}
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

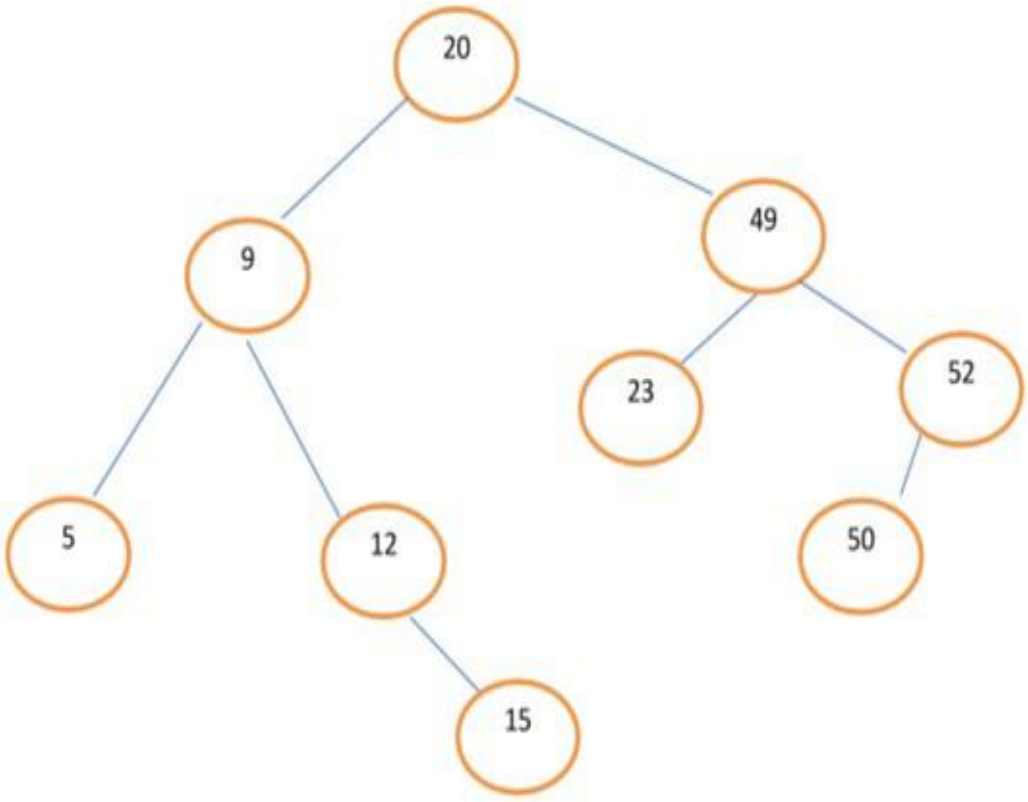
OUTPUT:



RESULT:

Thus the program to print path of the node in reverse order was successfully executed and verified.

QUESTION:
Given a Binary Tree, find sum of all left leaves in it. For example, sum of all left leaves in below Binary Tree is 5+23+50 = 78.



The idea is to traverse the tree, starting from root. For every node, check if its left subtree is a leaf. If it is, then add it to the result.

Ex. No: 13

Sum of left leaf nodes of a tree

Date:

AIM:

To write a c program to print sum of the left leaf nodes.

PSEUDOCODE:

```
//Program: To print sum of the left leaf nodes.
//Input: No of nodes in the tree, elements in the tree.
//Output: sum of the left leaf nodes.
BEGIN

  Insert(root, ele)
        If(root==NULL)
              root->data←ele
              root->left←NULL
              root->right←NULL
        Else If(ele<root->data)
              root->left←Insert(root->left,ele)
        Else If(ele>root->data)
              root->right←Insert(root->right,ele);
        return root

  Lef(i,root)
        If(root!=NULL)
              If(root->left==NULL && root->right==NULL && i->left==root)
              k←k+root->data
              lef(root,root->left)
              lef(root,root->right)
  Main()
        Read n
        For(i←0 to i<n)
              Read element
              root←Insert(root,element)
        Lef(NULL,root)
        Print k

END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int k=0;
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int element)
{
         if(root==NULL)
           {
                root=(struct node*)malloc(sizeof(struct node));
                root->data=element;
                root->left=NULL;
                root->right=NULL;
           }
          else if(element<root->data)
                root->left=insert(root->left,element);
          else if(element>root->data)
                root->right=insert(root->right,element);
          return root;
}
void lef(struct node *i,struct node *root)
{
        if(root!=NULL)
          {
              if(root->left==NULL && root->right==NULL && i->left==root)
                        k=k+root->data;
              lef(root,root->left);
              lef(root,root->right);
          }
}
void main()
{
        printf("ENTER SIZE...\n");
        int n;
        scanf("%d",&n);
        int i,element;
        printf("\nENTER element...\n");
        for(i=0;i<n;i++)
        {
              scanf("%d",&element);
              root=insert(root,element);
        }
```

```
        lef(NULL,root);
        printf("sum of all the left leaf nodes is %d\n",k);
    }
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

OUTPUT:

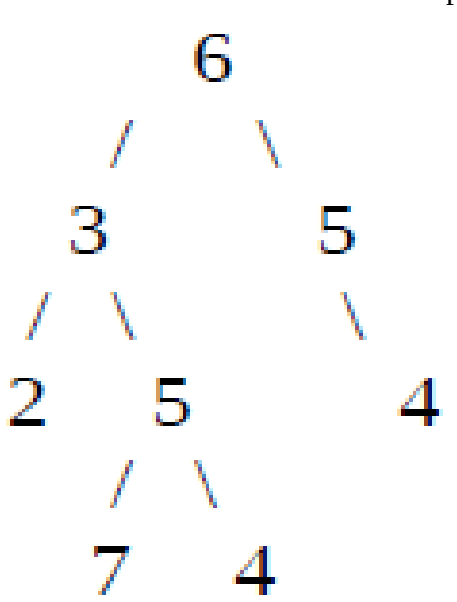
```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 2.4a.c
vineeth@vineeth:~/Desktop$ ./a.out
ENTER SIZE...
9

ENTER element...
20
9
5
12
15
49
23
52
50
sum of all the left leaf nodes is 78
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program to print sum of the left leaf nodes was successfully executed and verified.

QUESTION:
Given a binary tree, where every node value is a Digit from 1-9 .Find the sum of all the numbers which are formed from root to leaf paths.For example consider the following Binary Tree.



There are 4 leaves, hence 4 root to leaf paths:
Path Number
6->3->2 632
6->3->5->7 6357
6->3->5->4 6354
6->5>4 654
Answer = 632 + 6357 + 6354 + 654 = 13997

Ex. No: 14

Sum of all the path to the leaf node

Date:

AIM:

To write a c program to print sum of all the path to the leaf nodes.

PSEUDOCODE:

```
//Program: To print sum of all the path to the leaf nodes.
//Input: No of nodes in the tree, elements in the tree.
//Output: sum of  all the path to the leaf nodes.
BEGIN

  Insert(root, ele)
        If(root==NULL)
              root->data←ele
              root->left←NULL
              root->right←NULL
        Else If(ele<root->data)
              root->left←Insert(root->left,ele)
        Else If(ele>root->data)
              root->right←Insert(root->right,ele)
        return root

  Pathleaf(root, val)
        If(root!=NULL)
              If((root->left==NULL)&&(root->right==NULL))
                    arr[x]←val*10+root->data
                    x++
        else
              val←(val*10)+root->data
              pathleaf(root->left,val)
              pathleaf(root->right,val)
  Main()
        Read n
        For(i←0 to i<n)
               Read element
               root←insert(root,element)
        pathleaf(root,0)
        For(i←0 to arr[i]!='\0')
              k+=arr[i]
              Print arr[i]
        Print k

END
```

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
int arr[100],x=0;
struct node
{
        int data;
        struct node *left,*right;
}*root=NULL;
struct node *insert(struct node *root,int element)
{
         if(root==NULL)
          {
                root=(struct node*)malloc(sizeof(struct node));
                root->data=element;
                root->left=NULL;
                root->right=NULL;
          }
          else if(element<root->data)
                root->left=insert(root->left,element);
          else if(element>root->data)
                root->right=insert(root->right,element);
          return root;
}
void pathleaf(struct node *root,int val)
{
          if(root!=NULL)
           {
              if((root->left==NULL)&&(root->right==NULL))
              {
                      arr[x]=val*10+root->data;
                      x++;
              }
              else
                      val=(val*10)+root->data;
              pathleaf(root->left,val);
              pathleaf(root->right,val);
           }
}
void main()
{
        printf("ENTER SIZE...\n");
        int n,k=0;
        scanf("%d",&n);
        int i,element;
        printf("\nENTER element...\n");
```

```
        for(i=0;i<n;i++)
        {
                scanf("%d",&element);
                root=insert(root,element);
        }
        pathleaf(root,0);
        printf("The paths to leaf nodes are \n");
        for(i=0;arr[i]!='\0';i++)
        {
                k+=arr[i];
                printf("%d\n",arr[i]);
        }
        printf("Sum of all the path to the leaf node is %d\n",k);
}
```

DATA STRUCTURE USED: Binary search tree

TIME COMPLEXITY: O(log n)

ROUTINE: recursive

OUTPUT:

```
vineeth@vineeth:~$ cd Desktop
vineeth@vineeth:~/Desktop$ gcc 2.4b.c
vineeth@vineeth:~/Desktop$ ./a.out
ENTER SIZE...
9

ENTER element...
20
9
5
12
15
49
23
52
50
The paths to leaf nodes are
2095
21035
2513
25470
Sum of all the path to the leaf node is 51113
vineeth@vineeth:~/Desktop$
```

RESULT:

Thus the program to print sum of path to the leaf nodes was successfully executed and verified.