Ex. No: 1 **Date: 12.08.24**

Register No.: 230701374 **Name: Velan A**

Basic C Programming

1.a.

Aim: Given two numbers, write a C program to swap the given numbers.

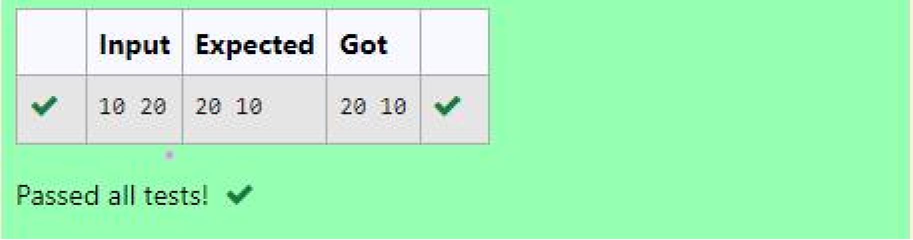
Algorithm:

```
DECLARE a, b, temp as INTEGER
READ a
READ b
// Swap values of a and b
temp = a
a = b
b = temp
PRINT a, b
```

Program:

```
#include<stdio.h>
int main(){
int a;
int b;
int temp;
scanf("%d",&a);
scanf("%d",&b);

temp=a;
a=b;
b=temp;
printf("%d %d",a,b);
}
```

Output:



	Input	Expected	Got	
✓	10 20	20 10	20 10	✓

Passed all tests! ✓

Ex. No: 2 **Date: 20.08.24**

Register No.: 230701374 **Name: Velan A**

Finding Time Complexity of Algorithms

2.a. Finding Complexity using Counter Method

Aim: Convert the following algorithm into a program and find its time complexity using the counter method.

```
void function (int n)
{
    int i= 1;    int s =1;

    while(s <= n)
    {
         i++;
         s += i;
     }
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

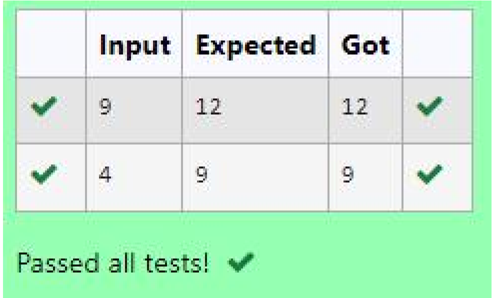
Input:
A positive Integer n
Output:
Print the value of the counter variable

Program:

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

```
void function(int n)
{
    int counter = 0;
    int i = 1;
    counter++;
    int s = 1;
    counter++;
    while(s<=n)
    {
        i++;
        s += i;
        counter++;
        counter++;
        counter++;
    }
    counter++;
    printf("%d",counter);
}

int main()
{
    int num;
    scanf("%d",&num);
    function(num);
}
```

Output:



	Input	Expected	Got	
✓	9	12	12	✓
✓	4	9	9	✓

Passed all tests! ✓

2.b. Finding Complexity using Counter Method

Aim: Convert the following algorithm into a program and find its time complexity using the counter method.

```
void func(int n)
{
    if(n==1)
    {
      printf("*");
    }
    else
    {
     for(int i=1; i<=n; i++)
     {
       for(int j=1; j<=n; j++)
       {
          printf("*");
          printf("*");
          break;
       }
     }
   }
 }
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

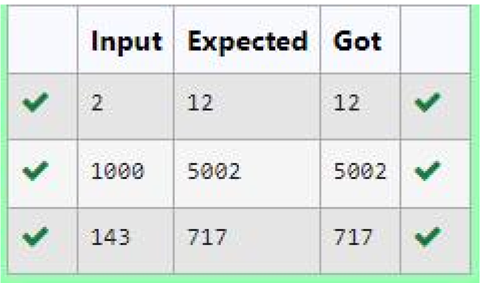
Output:

Print the value of the counter variable

Program:

```
#include <stdio.h>

void func(int n)

{

    int counter = 0;

    if(n==1)
    {
     //printf("*");
    }
    else
    {
        for(int i=1;i<=n;i++)
        {
            counter++;
            for(int j=1;j<=n;j++)
            {
                counter++;
                counter++;
                break;
                counter++;
            }
            counter++;
            counter++;
        }
        counter++;
    }
    counter++;
    printf("%d",counter);
}

int main()
{
    int num;
    scanf("%d",&num);

    func(num);

}
```

Output:



	Input	Expected	Got	
✓	2	12	12	✓
✓	1000	5002	5002	✓
✓	143	717	717	✓

2.c. Finding Complexity using Counter Method

Aim: Convert the following algorithm into a program and find its time complexity using counter method.

```
 Factor(num) {
 {
    for (i = 1; i <= num;++i)
    {
     if (num % i== 0)
        {
          printf("%d ", i);
        }
     }
  }
```

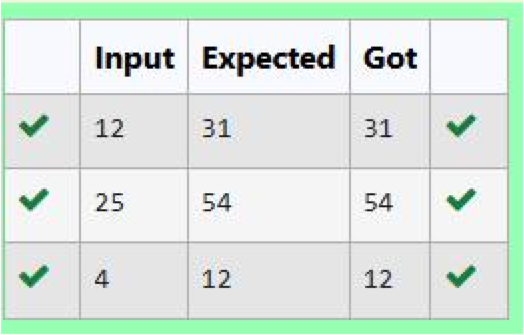
Note: No need of counter increment for declarations and scanf() and counter variable printf() statement.

Input:
A positive Integer n
Output:
Print the value of the counter variable

Program :

```
#include <stdio.h>

void Factor(int num)

{

    int counter = 0;

    for(int i = 1;i <=num;i++)

    {

        counter++;

        if(num%i==0)

        {
            counter++;

            //printf("%d",i);

        }

        counter++;

    }

    counter++;

    printf("%d",counter);

}

int main()

{

    int n;

    scanf("%d",&n);

    Factor(n);

}
```

Output:



	Input	Expected	Got	
✓	12	31	31	✓
✓	25	54	54	✓
✓	4	12	12	✓

2.d. Finding Complexity using Counter Method

Aim: Convert the following algorithm into a program and find its timecomplexity using counter method.

```
void function(int n)
{
    int c= 0;
    for(int i=n/2; i<n; i++)
        for(int j=1; j<n; j = 2 * j)
            for(int k=1; k<n; k = k * 2)
                c++;
}
```

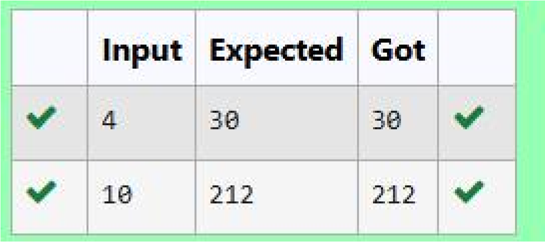
Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:
 A positive Integer n
Output:
Print the value of the counter variable

Program:

```
#include <stdio.h>

#include <stdlib.h>



void function(int n)

{

  int c=0;

  c++;

  for(int i=n/2; i<n;i++)

  {

    for(int j=1;j<n;j=2*j)

    {
            for(int k=1;k<n;k=k*2)
            {
                c++;
                c++;
            }
            c++;
            c++;
        }
        c++;
        c++;
    }
    c++;
    printf("%d",c);
}
int main()
{
    int num;
    scanf("%d",&num);
    function(num);
}
```

Output:



	Input	Expected	Got	
✓	4	30	30	✓
✓	10	212	212	✓

2.e. Finding Complexity using Counter Method

Aim: Convert the following algorithm into a program and find its time complexity using counter method.

```
void reverse(int n)
{
   int rev = 0, remainder;
   while (n != 0)
    {
        remainder = n % 10;
        rev = rev * 10 + remainder;
        n/= 10;


    }
print(rev);
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

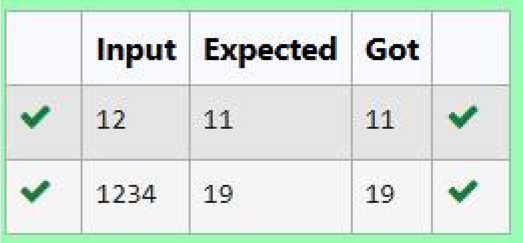
Input:
 A positive Integer n
Output:
Print the value of the counter variable

Program:

```
#include <stdio.h>

void reverse(int n)

{

   int c=0;

   int rev = 0,rem;

   c++;

   c++;

   while(n!=0)

   {

      rem = n%10;
```

```
        rev = rev * 10 + rem;

        n/=10;

        c++;

        c++;

        c++;

        c++;


    }

    c++;

    //printf("%d",rev);

    printf("%d",c);

}

int main()

{

    int num;

    scanf("%d",&num);

    reverse(num);

}
```

Output:



	Input	Expected	Got	
✓	12	11	11	✓
✓	1234	19	19	✓

Ex. No: 3 **Date: 26.08.24**

Register No.: 230701374 **Name: Velan A**

Greedy Algorithm

3.a. 1-G-Coin Problem

Aim: Write a program to take value V and we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change.

Input Format:

Take an integer from stdin.

Output Format:

print the integer which is change of the number.

Example Input :

64

Output:

4

Explanaton:

We need a 50 Rs note and a 10 Rs note and two 2 rupee coins.

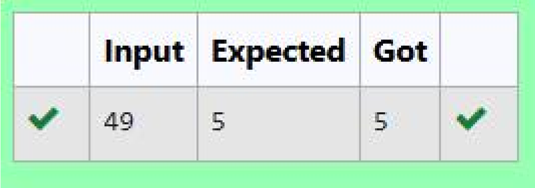
Program:

```
#include<stdio.h>


int main(){
    int n,change=0;
```

```
scanf("%d",&n);
if(n>=1000){
    change+=n/1000;
    n%=1000;
}
if(n>=500){
    change+=n/500;
    n%=500;
}
if(n>=100){
    change+=n/100;
    n%=100;
}
if(n>=50){
    change+=n/50;
    n%=50;
}
if(n>=20){
    change+=n/20;
    n%=20;
}
if(n>=10){
    change+=n/10;
    n%=10;
}
if(n>=5){
    change+=n/5;
```

```
        n%=5;
    }
    if(n>=2){
        change+=n/2;
        n%=2;
    }
    if(n==1){change=1;}
    printf("%d",change);
}
```

Output:



	Input	Expected	Got	
✓	49	5	5	✓

3.b. 2-G-Cookies Problem

Aim:

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie.

Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Example 1:

Input:

3

1 2 3

2

1 1

Output:

1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Constraints:

1 <= g.length <= 3 * 10^4

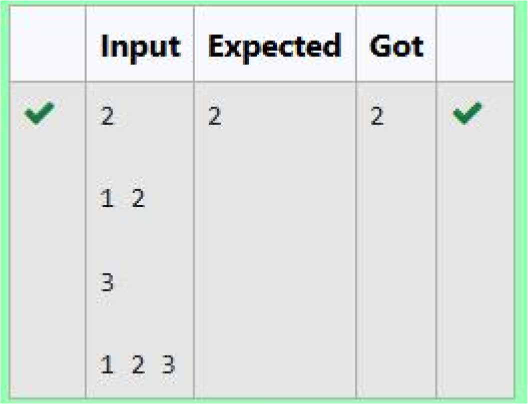
0 <= s.length <= 3 * 10^4

1 <= g[i], s[j] <= 2^31 - 1

Program:

```
#include<stdio.h>
int main(){
    int a,c=0;
    scanf("%d",&a);
    int g[a];
    for(int i=0;i<a;i++){
        scanf("%d",&g[i]);
    }
    int b;
    scanf("%d",&b);
    int s[b];
    for(int j=0;j<b;j++){
        scanf("%d",&s[j]);
    }
    for(int i=0;i<a;i++){
        for(int j=0;j<b;j++){
            if(g[i]<=s[j]){
                c++;
                s[j]=0;
                break;
            }
        }
    }
    printf("%d",c);
}
```

Output:



	Input	Expected	Got	
✓	2 1 2 3 1 2 3	2	2	✓

3.c. 3-G-Burger Problem

Aim:

A person needs to eat burgers. Each burger contains a count of calorie. After eating the burger, the person needs to run a distance to burn out his calories.
If he has eaten *i* burgers with c calories each, then he has to run at least $3^i * c$ kilometers to burn out the calories. For example, if he ate 3 burgers with the count of calorie in the order: [1, 3, 2], the kilometers he needs to run are (3^0 * 1) + (3^1 * 3) + (3^2 * 2) = 1 + 9 + 18 = 28.
But this is not the minimum, so need to try out other orders of consumption and choose the minimum value. Determine the minimum distance he needs to run. Note: He can eat burger in any order and use an efficient sorting algorithm.Apply greedy approach to solve the problem.

Input Format

First Line contains the number of burgers

Second line contains calories of each burger which is n space-separate integers

Output Format

Print: Minimum number of kilometers needed to run to burn out the calories

Sample Input

```
3
5 10 7
```

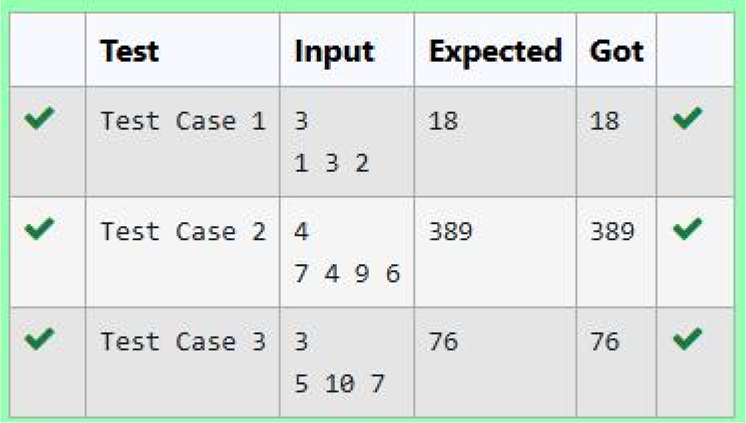
Sample Output

```
76
```

Program:

```
#include<stdio.h>

#include<math.h>


int main(){

    int n;
    scanf("%d",&n);
    int cal[n];
    for(int i=0;i<n;i++){
        scanf("%d ",&cal[i]);
    }
    int i, j, temp;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (cal[j] > cal[j+1]) {
                temp = cal[j];
                cal[j] = cal[j+1];
                cal[j+1] = temp;
            }
        }
    }
    int mulfact;
    int sum=0;
    int h=n-1;
    for(int i=0;i<n;i++)
    {
        mulfact=pow(n,i);
        sum+=mulfact*cal[h];
        h--;
    }
    printf("%d",sum);
}
```

Output:



	Test	Input	Expected	Got	
✓	Test Case 1	3 1 3 2	18	18	✓
✓	Test Case 2	4 7 4 9 6	389	389	✓
✓	Test Case 3	3 5 10 7	76	76	✓

3.d. 4-G-Array Sum Max Problem

Aim:

Given an array of N integer, we have to maximize the sum of arr[i] * i, where i is the index of the element (i = 0, 1, 2, ..., N).Write an algorithm based on Greedy technique with a Complexity O(nlogn).

Input Format:

First line specifies the number of elements-n

The next n lines contain the array elements.

Output Format:

Maximum Array Sum to be printed.

Sample Input:

5

2 5 3 4 0

Sample output:

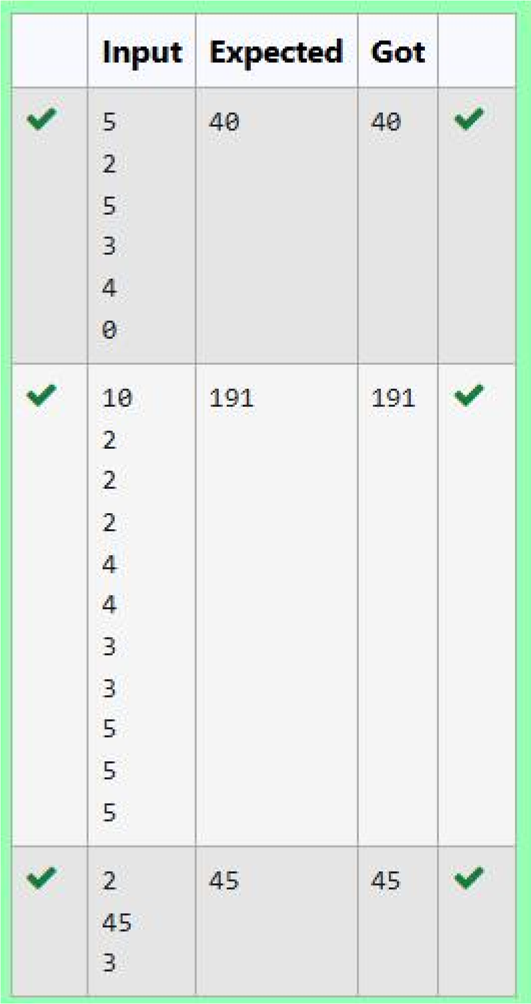
40

Program:

```
#include <stdio.h>

int main()

{

   int n,i,j;

   scanf("%d",&n);

   int arr[n];
```

```
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    int temp,sum = 0;
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(arr[i]>arr[j])
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
    for(i=0;i<n;i++)
    {
        sum+=arr[i]*i;
    }
    printf("%d",sum);
}
```

Output:



	Input	Expected	Got	
✓	5 2 5 3 4 0	40	40	✓
✓	10 2 2 2 4 4 3 3 5 5 5	191	191	✓
✓	2 45 3	45	45	✓

3.e. 5-G-Product of Array Elements-Minimum

Aim:

Given two arrays array_One[] and array_Two[] of same size N. We need to first rearrange the arrays such that the sum of the product of pairs(1 element from each) is minimum. That is SUM (A[i] * B[i]) for all i is minimum.

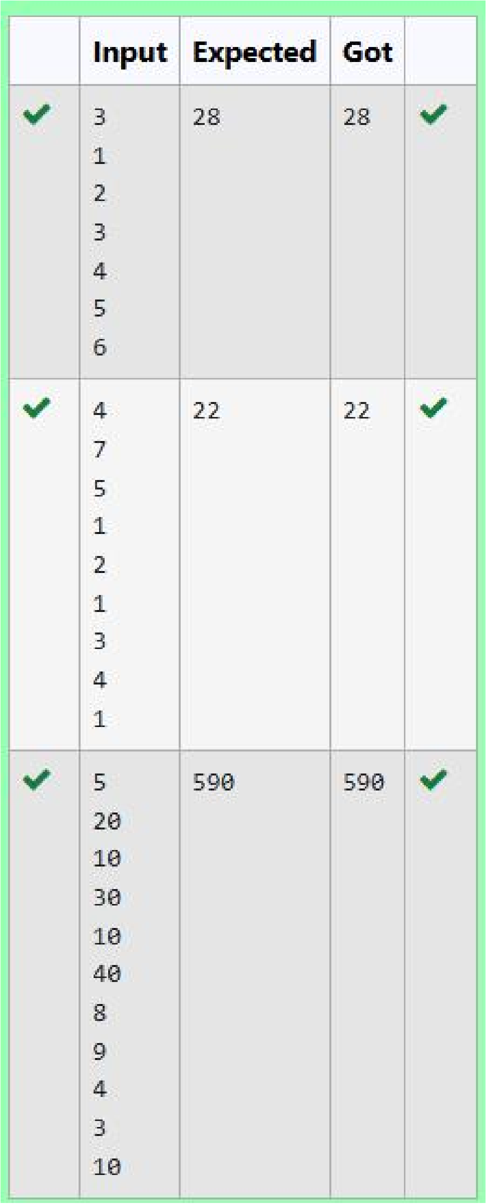
Program:

```
#include <stdio.h>


int main()

{

    int n,i,j;

    scanf("%d",&n);

    int arr_one[n];

    int arr_two[n];

    int temp_one,sum = 0;

    int temp_two;

    for(i=0;i<n;i++)

    {

        scanf("%d",&arr_one[i]);

    }

    for(i=0;i<n;i++)
```

```
{
    scanf("%d",&arr_two[i]);
}
for(i=0;i<n;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(arr_one[i]>arr_one[j])
        {
            temp_one = arr_one[i];
            arr_one[i] = arr_one[j];
            arr_one[j] = temp_one;
        }
    }
}
for(i=0;i<n;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(arr_two[j]>arr_two[i])
        {
            temp_two = arr_two[j];
```

```
                arr_two[j] = arr_two[i];

                arr_two[i] = temp_two;

            }

        }

    }

    for(i=0;i<n;i++)

    {

            sum+=arr_one[i]*arr_two[i];

    }

    printf("%d",sum);

}
```

Output:



	Input	Expected	Got	
✓	3 1 2 3 4 5 6	28	28	✓
✓	4 7 5 1 2 1 3 4 1	22	22	✓
✓	5 20 10 30 10 40 8 9 4 3 10	590	590	✓

Ex. No: 4 **Date: 03.09.24**

Register No.: 230701374 **Name: Velan A**

Divide and Conquer

4.a. Number of Zeros in a Given Array

Aim: Given an array of 1s and 0s this has all 1s first followed by all 0s. Aim is to find the number of 0s. Write a program using Divide and Conquer to Count the number of zeroes in the given array.

Input Format

First Line Contains Integer m – Size of array

Next m lines Contains m numbers – Elements of an array

Output Format

First Line Contains Integer – Number of zeroes present in the given array.

Algorithm:

```
function count(a, left, right) {
   // base case: if left index exceeds right index
   if left is greater than right {
      return 0
   }

   initialize mid as (left + right) / 2  // find the middle index

   // check if the middle element is 1
   if a[mid] is equal to 1 {
      // check if the next element is 0
```

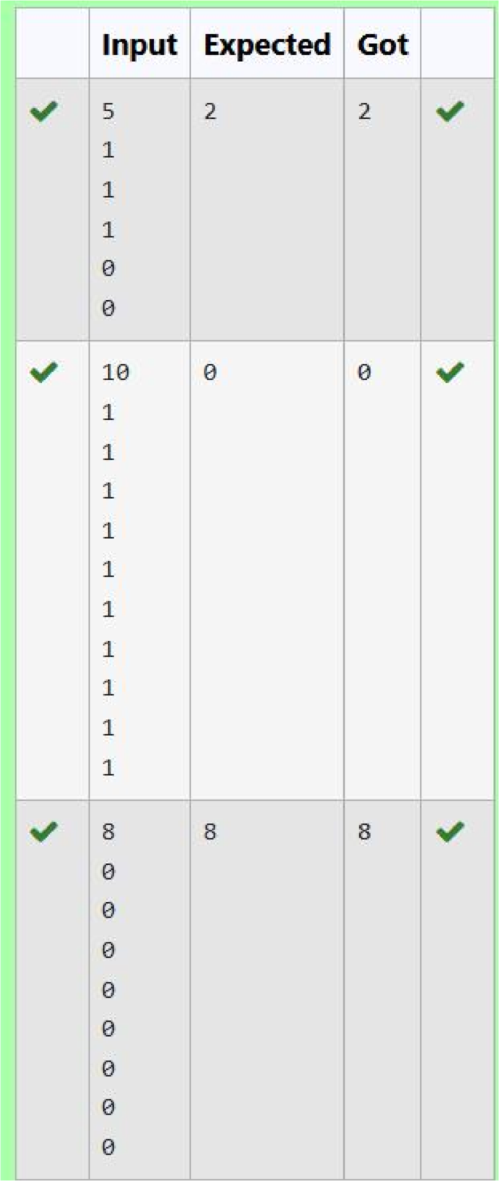
```
        if a[mid + 1] is equal to 0 {
            // count zeros from mid + 1 to right
            initialize c as (right - (mid + 1)) + 1
            return c
        } else {
            // search in the right half
            return count(a, mid + 1, right)
        }
    }
    // check if both ends are 0
    else if a[left] is equal to 0 and a[right] is equal to 0 {
        return right + 1  // return total count of elements
    }
    // search in the left half
    else {
        return count(a, left, mid - 1)
    }
}

function main() {
    initialize n  // number of elements
    read n from user

    initialize arr array of size n  // array to hold binary values

    // read values into the arr array
    for i from 0 to n - 1 {
        read arr[i] from user
    }

    initialize left as 0  // left index
    initialize right as n - 1  // ri
```

Program:

```
#include <stdio.h>
int count(int a[],int left,int right)
{
    if(left>right)
    {
        return 0;
    }
    int mid=(left+right)/2;
    if(a[mid]==1)
    {
        if(a[mid+1]==0)
        {
            int c= (right-(mid+1))+1;
            return c;
        }
        else{
            return count(a,mid+1,right);
        }
    }
    else if(a[left]==0 && a[right]==0)
    {
        return right+1;
    }
    else
    {
        return count(a,left,mid-1);
    }

}
int main()
{
    int n;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    int left=0;
    int right=n-1;
    int result=count(arr,left,right);
    printf("%d",result);
}

#include<stdio.h>

int divide(int arr[],int low,int high)
```

```
{
    if(arr[high]==1)
    {
        return 0;
    }
    if(arr[low]==0)
    {
        return high-low+1;
    }
    int mid=(low+high)/2;
    int left=divide(arr,low,mid);
    int right=divide(arr,mid+1,high);
    return left+right;
}
int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
    {
        scanf("%d",&arr[i]);
    }
    int count=divide(arr,0,size-1);
    printf("%d¥n",count);
}
```

Output:



	Input	Expected	Got	
✓	5 1 1 1 0 0	2	2	✓
✓	10 1 1 1 1 1 1 1 1 1 1	0	0	✓
✓	8 0 0 0 0 0 0 0 0	8	8	✓

4.b. Majority Element

Aim: Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than ⌊ n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

Constraints:

- n == nums.length
- 1 <= n <= 5 * 10^4
- -2^{31} <= nums[i] <= 2^{31} - 1

Algorithm:

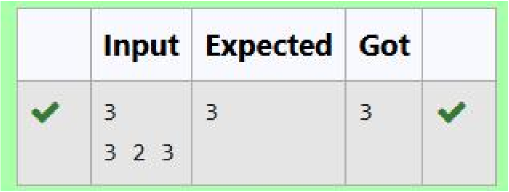
```
int divide(a, l, r, n) {

    // base case: if left index equals right index

    if l is equal to r {

        return a[l]  // return the only element

    }



    initialize mid as (l + r) / 2  // find the middle index



    // recursively divide the array

    initialize min as divide(a, l, mid, n)  // find min in left half

    initialize max as divide(a, mid + 1, r, n)  // find max in right half
    initialize leftc as 0  // counter for min occurrences
    initialize rightc as 0  // counter for max occurrences

    // count occurrences of min and max in the entire array
    for i from 0 to n - 1 {
        if a[i] is equal to min {
            increment leftc by 1  // count occurrences of min
        } else {
            increment rightc by 1  // count occurrences of max
        }
    }

    // check if min occurs more than n/2 times
    if leftc is greater than (n / 2) {
        return min  // return min if it is the majority element
    } else {
        return max  // return max otherwise
    }
}

int main() {
    initialize n  // number of elements
    read n from user

    initialize a array of size n  // array to hold input values
    // read values into the array
    for j from 0 to n - 1 {
        read a[j] from user
    }

    initialize l as 0  // left index
    initialize r as n - 1  // right index

    // call the divide function
    initialize result as divide(a, l, r, n)

    print result  // output the final majority element
}
```

Program:

```
#include<stdio.h>
int divide(int arr[],int l,int h,int s)
{
    if(l==h)
        return arr[l];
    int mid=(l+h)/2;
    int left=divide(arr,l,mid,s);
    int right=divide(arr,mid+1,h,s);
    if(left>s/2)
```

```
        return left;
    else
        return right;
}
int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
        scanf("%d",&arr[i]);
    int count=divide(arr,0,size-1,size);
    printf("%d",count);
}
```

Output:



	Input	Expected	Got	
✓	3 3 2 3	3	3	✓

4.c. Finding Floor Value

Aim: Given a sorted array and a value x, the floor of x is the largest element in array smaller than or equal to x. Write divide and conquer algorithm to find floor of x.

Input Format

First Line Contains Integer n – Size of array

Next n lines Contains n numbers – Elements of an array

Last Line Contains Integer x – Value for x

Output Format

First Line Contains Integer – Floor value for x

Algorithm:

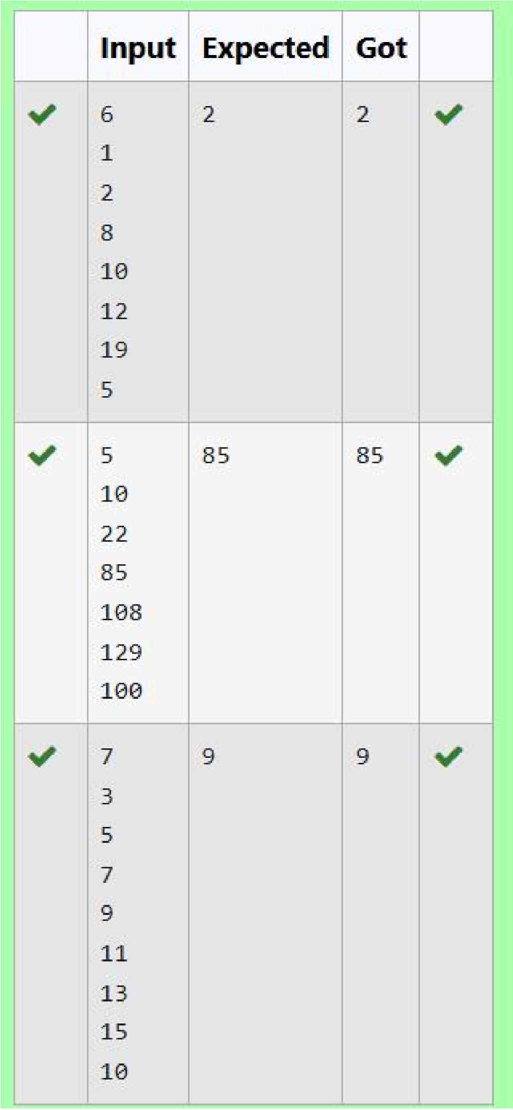
```
int large(arr, l, r, x){

    // Base case: if the range is invalid

    if r < l

        return 0  // return 0 when there is no valid element



    // Calculate the middle index

    mid = (l + r) / 2



    // Check if the middle element is equal to x

    if arr[mid] is equal to x

        return mid  // return the index of x if found



    // If the middle element is less than x

    else if arr[mid] < x

        // Recursively search in the right half
        floorIndex = large(arr, mid + 1, r, x)


        // Check if a valid floor index is found
        if floorIndex is not equal to 0
            return floorIndex  // return the found index
        else
            return mid  // return mid as the largest element less than x


    // If the middle element is greater than x, search in the left half
    else
        return large(arr, l, mid - 1, x)  // search in the left half
}
Int main()
    initialize n  // number of elements in the array
    read n from user


    initialize arr of size n  // array to hold input values


    // Read values into the array
    for i from 0 to n - 1
        read arr[i] from user


    initialize l as 0  // left index
    initialize r as n - 1  // right index
    initialize x  // the value for which we want to find the largest element less than or equal to x
    read x from user


    // Call the large function
    result = large(arr, l, r, x)


    // Check the result
    if result is equal to 0
        print x  // if no valid element, print x
    else
        print arr[result]  // print the largest element less than or equal to x
```

Program:

```
#include<stdio.h>
int divide(int arr[],int l,int h,int x)
{
    int mid;
    if(l==h)
        return mid=l;
    else
        mid=(l+h)/2;
    if(arr[mid]<=x)
        return arr[mid];
    int left=divide(arr,l,mid,x);
    int right=divide(arr,mid+1,h,x);
    if(left<=x)
```

```
        return left;
    else
        return right;
}
int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
        scanf("%d",&arr[i]);
    int x;
    scanf("%d",&x);
    int floor=divide(arr,0,size-1,x);
    printf("%d",floor);
}
```

Output:



	Input	Expected	Got	
✓	6 1 2 8 10 12 19 5	2	2	✓
✓	5 10 22 85 108 129 100	85	85	✓
✓	7 3 5 7 9 11 13 15 10	9	9	✓

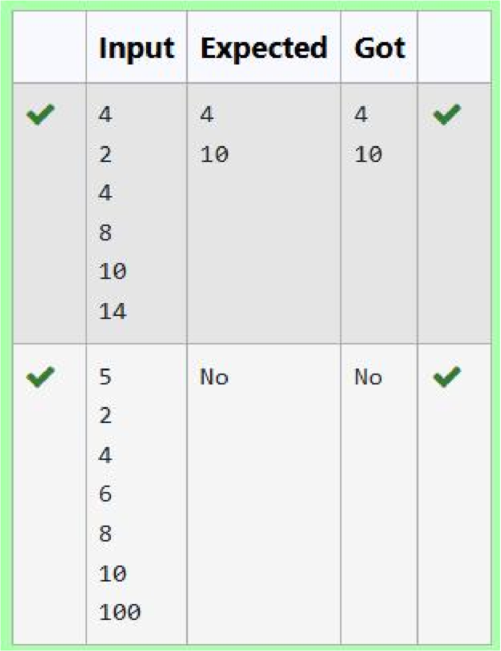
4.d. Two Elements Sum to X

Aim: Given a sorted array of integers say arr[] and a number x. Write a recursive program using divide and conquer strategy to check if there exist two elements in the array whose sum = x. If there exist such two elements then return the numbers, otherwise print as “No”.
Note: Write a Divide and Conquer Solution
Input Format
First Line Contains Integer n – Size of array
Next n lines Contains n numbers – Elements of an array
Last Line Contains Integer x – Sum Value
Output Format
First Line Contains Integer – Element1
Second Line Contains Integer – Element2 (Element 1 and Elements 2 together sums to value “x”)

Program:

```
#include <stdio.h>

void divide(int arr[],int l,int h,int x)

{

   int mid=(l+h)/2;

   if (l>=h)

   {

      printf("No");

      return;

   }

   int sum=arr[l]+arr[h];

   if(sum==x)

   {
        printf("%d¥n%d¥n",arr[l],arr[h]);
        return;
    }
    else if(sum<x)
    {
        divide(arr,mid+1,h,x);
    }
    else
    {
        divide(arr,mid-1,h,x);
    }
}
int main()
{
    int size,x;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
        scanf("%d",&arr[i]);
    scanf("%d",&x);
    divide(arr,0,size-1,x);
}
```

Output:



	Input	Expected	Got	
✓	4 2 4 8 10 14	4 10	4 10	✓
✓	5 2 4 6 8 10 100	No	No	✓

4.e. Implementation of Quick Sort

Aim: Write a Program to Implement the Quick Sort Algorithm

Input Format:
The first line contains the no of elements in the list-n
The next n lines contain the elements.

Output:
Sorted list of elements

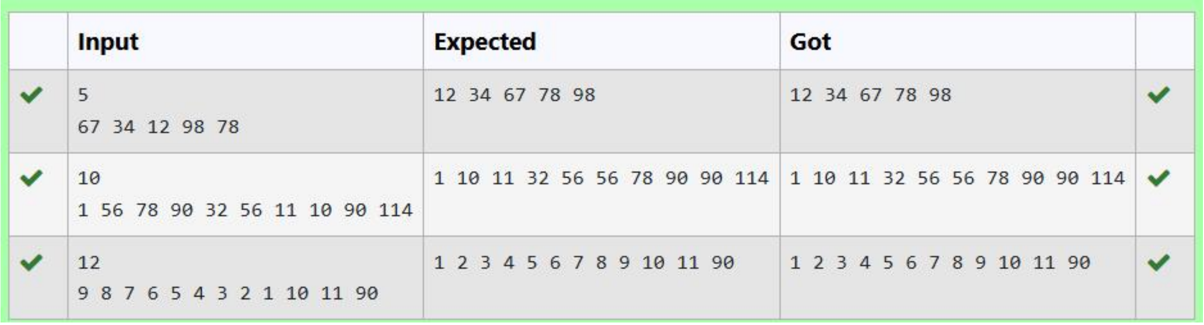
Algorithm:

```
int partition(a, left, right)
{
    pivot = right  // Choose the last element as pivot
    i = left - 1   // Index of smaller element


    for j from left to right - 1
    {
        if a[j] < a[pivot]
        {
            i++
            // Swap a[i] and a[j]
            temp = a[i]
            a[i] = a[j]
            a[j] = temp
        }
    }
    // Swap a[i + 1] and a[right]

    temp = a[i + 1]

    a[i + 1] = a[right]

    a[right] = temp

    return (i + 1)   // Return the partition index

}


function quick(a, left, right)

{

    if left < right

    {

        p = partition(a, left, right)  // Partition the array

        quick(a, left, p - 1)          // Recursively sort the left sub-array

        quick(a, p + 1, right)         // Recursively sort the right sub-array

    }

}


int main()

{

    initialize n  // number of elements

    read n from user


    initialize a of size n  // array to hold input values

    for i from 0 to n - 1

    {

        read a[i] from user
    }

    quick(a, 0, n - 1)  // Call the quicksort function

    // Print the sorted array
    for i from 0 to n - 1
    {
        print a[i]
    }
}
```

Program:

```
#include<stdio.h>
int divide(int arr[],int l,int h)
{
    int i=l,j=h;
    int pivot=arr[l];
    while(i<j)
    {
        while(arr[i]<=pivot && i<h)
            i++;
        while(arr[j]>pivot && j>l)
            j--;
        if(i<j)
        {
            int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
        }
    }
    int temp=arr[l];
    arr[l]=arr[j];
    arr[j]=temp;
    return j;
}
void quicksort(int arr[],int l,int h)
{
    if(l<h)
    {
        int div=divide(arr,l,h);
        quicksort(arr,l,div-1);
        quicksort(arr,div+1,h);
    }
}
int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i=0;i<size;i++)
    {
        scanf("%d",&arr[i]);
    }
    quicksort(arr,0,size-1);
    for(int i=0;i<size;i++)
        printf("%d ",arr[i]);
}
```

Output:



	Input	Expected	Got	
✓	5 67 34 12 98 78	12 34 67 78 98	12 34 67 78 98	✓
✓	10 1 56 78 90 32 56 11 10 90 114	1 10 11 32 56 56 78 90 90 114	1 10 11 32 56 56 78 90 90 114	✓
✓	12 9 8 7 6 5 4 3 2 1 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	1 2 3 4 5 6 7 8 9 10 11 90	✓

Ex. No: 5 **Date: 10.09.24**

Register No.: 230701374 **Name: Velan A**

Dynamic Programming

5.a. Playing with Numbers

Aim: Ram and Sita are playing with numbers by giving puzzles to each other. Now it was Ram term, so he gave Sita a positive integer 'n' and two numbers 1 and 3. He asked her to find the possible ways by which the number n can be represented using 1 and 3.Write any efficient algorithm to find the possible ways.

Example 1:

Input: 6
Output:6
Explanation: There are 6 ways to 6 represent number with 1 and 3

1+1+1+1+1+1
3+3
1+1+1+3
1+1+3+1
1+3+1+1
3+1+1+1

Input Format
First Line contains the number n

Output Format

Print: The number of possible ways 'n' can be represented using 1 and 3

Sample Input

6

Sample Output

6

Algorithm:

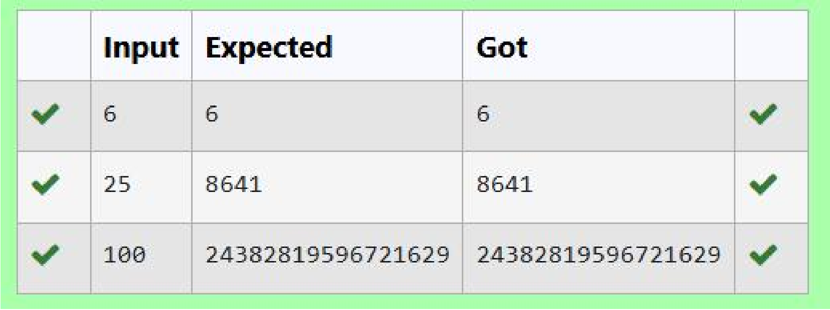
```
function countWays(n)
{
    initialize a of size n + 1  // Array to store the number of ways

    a[0] = 1  // Base case: 1 way to climb 0 stairs
    a[1] = 1  // Base case: 1 way to climb 1 stair

    if n >= 2
    {
        a[2] = 1  // Base case: 1 way to climb 2 stairs
    }

    if n >= 3
    {
        a[3] = 2  // Base case: 2 ways to climb 3 stairs
    }

    // Fill the array for all stairs from 4 to n
    for i from 4 to n
    {
        a[i] = a[i - 1] + a[i - 3]  // Total ways to climb i stairs
    }

    return a[n]  // Return the number of ways to climb n stairs
}
```

```
function main()
{
    initialize n  // Number of stairs
    read n from user

    result = countWays(n)  // Calculate the number of ways
    print result  // Print the result

    return 0
}
```

Program:

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    long arr[n+1];
    arr[0] = 1;
    for(int i = 1 ; i <= n ; i++)
    {
        arr[i] = arr[i -1 ];
        if(i>=3)
        {
            arr[i]+=arr[i-3];
        }
    }
    printf("%ld¥n",arr[n]);
}
```

¥***Output:***



	Input	Expected	Got	
✓	6	6	6	✓
✓	25	8641	8641	✓
✓	100	24382819596721629	24382819596721629	✓

5.b. Playing with chessboard

Aim: Ram is given with an n*n chessboard with each cell with a monetary value. Ram stands at the (0,0), that the position of the top left white rook. He is been given a task to reach the bottom right black rook position (n-1, n-1) constrained that he needs to reach the position by traveling the maximum monetary path under the condition that he can only travel one step right or one step down the board. Help ram to achieve it by providing an efficient DP algorithm.

Example:

Input

3

1 2 4

2 3 4

8 7 1

Output:

19

Explanation:

Totally there will be 6 paths among that the optimal is

Optimal path value:1+2+8+7+1=19

Input Format

First Line contains the integer n

The next n lines contain the n*n chessboard values

Output Format

Print Maximum monetary value of the path

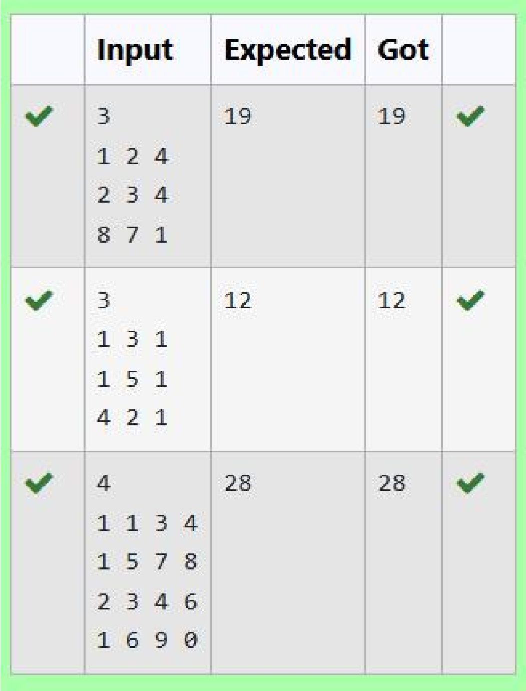
Algorithm:

```
function max(a, b)
{
    return (a > b) ? a : b  // Return the maximum of a and b
}

function maxMonetaryPath(n, board)
{
    initialize dp[n][n]  // Array to store maximum monetary path sums

    dp[0][0] = board[0][0]  // Starting point

    // Fill the first row
    for j from 1 to n - 1
    {
        dp[0][j] = dp[0][j - 1] + board[0][j]
    }

    // Fill the first column
    for i from 1 to n - 1
    {
        dp[i][0] = dp[i - 1][0] + board[i][0]
    }

    // Fill the rest of the dp table
    for i from 1 to n - 1
    {
        for j from 1 to n - 1
        {
            dp[i][j] = board[i][j] + max(dp[i - 1][j], dp[i][j - 1])
        }
    }

    return dp[n - 1][n - 1]  // Return the maximum monetary path to the bottom-right corner
}

function main()
{
    initialize n  // Size of the board
    read n from user

    initialize board[n][n]  // Create the board array
    for i from 0 to n - 1
    {
        for j from 0 to n - 1
        {
            read board[i][j] from user
        }
    }

    result = maxMonetaryPath(n, board)  // Calculate the maximum monetary path
    print result  // Print the result
}
```

Program:

```
#include <stdio.h>
#define max 100
int main()
{
    int n;
    scanf("%d",&n);
    int chess[max][max];
    for(int i = 0 ; i <n  ;i++)
    {
        for(int j = 0 ; j< n ; j++)
        {
            scanf("%d",&chess[i][j]);
        }
    }
    int dp[max][max];
    dp[0][0]=chess[0][0];
    for(int i = 1; i <n ; i++){
        dp[i][0] = dp[i-1][0] + chess[i][0];
        dp[0][i] = dp[0][i-1]+chess[0][i];
    }
    for(int i = 1 ; i < n ; i++)
    {
        for(int j =1 ; j < n; j++)
        {
            dp[i][j] = (dp[i-1][j]>dp[i][j-1] ? dp[i-1][j] : dp[i][j-1])+ chess[i][j];

        }
    }
    printf("%d",dp[n-1][n-1]);

}
```

Output:



	Input	Expected	Got	
✓	3 1 2 4 2 3 4 8 7 1	19	19	✓
✓	3 1 3 1 1 5 1 4 2 1	12	12	✓
✓	4 1 1 3 4 1 5 7 8 2 3 4 6 1 6 9 0	28	28	✓

5.c. Longest Common Subsequence

Aim: Given two strings find the length of the common longest subsequence(need not be contiguous) between the two.

Example:

s1: ggtabe

s2: tgatasb

s1	a	g	**g**	**t**	**a**	**b**	
s2	**g**	x	**t**	x	**a**	y	**b**

The length is 4

Solveing it using Dynamic Programming

For example:

Input	Result
aab azb	2

Algorithm:

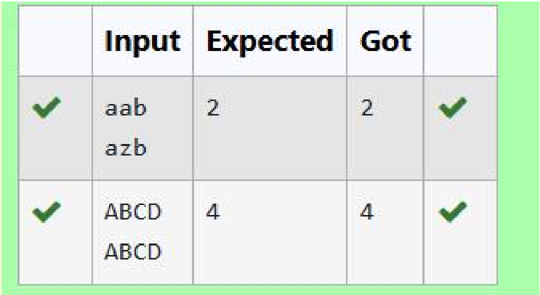
```
int longestCommonSubsequence(s1, s2)
{
    m = length of s1  // Length of first string
    n = length of s2  // Length of second string


    initialize dp[m + 1][n + 1]  // DP table
    // Initialize the DP table with base cases
    for i from 0 to m
    {
        for j from 0 to n
        {
            if i == 0 or j == 0
            {
                dp[i][j] = 0  // Base case: LCS of an empty string
            }
            else if s1[i - 1] == s2[j - 1]
            {
                dp[i][j] = dp[i - 1][j - 1] + 1  // Characters match
            }
            else
            {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])  // Characters do not match
            }
        }
    }

    return dp[m][n]  // Return length of LCS
}

function main()
{
    initialize s1[100], s2[100]  // Arrays to hold the strings

    read s1 from user
    read s2 from user
    result = longestCommonSubsequence(s1, s2)  // Calculate LCS
    print result  // Print the result
}
```

Program:

```
#include<stdio.h>
#include<string.h>
#define max 100
int main()
{
    char s1[max];
    char s2[max];
    scanf("%s",s1);
    scanf("%s",s2);
    int m=strlen(s1);
    int n=strlen(s2);
    int dp[m+1][n+1];
    for(int i=0;i<=m;i++)
        dp[i][0]=0;
    for(int j=0;j<=n;j++)
        dp[0][j]=0;
    for(int i=1;i<=m;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(s1[i-1]==s2[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
```

```
      else
        dp[i][j]=(dp[i-1][j]>dp[i][j-1])?dp[i-1][j]:dp[i][j-1];
    }
  }
  printf("%d¥n",dp[m][n]);
}
```

Output:



	Input	Expected	Got	
✓	aab azb	2	2	✓
✓	ABCD ABCD	4	4	✓

5.d. Longest non-decreasing Subsequence

Aim: Problem statement:

Find the length of the Longest Non-decreasing Subsequence in a given Sequence.

Eg:

Input:9

Sequence:[-1,3,4,5,2,2,2,2,3]

the subsequence is [-1,2,2,2,2,3]

Output:6

Algorithm:

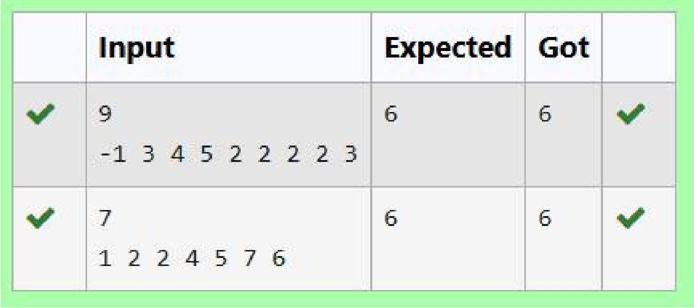
```
int longestNonDecreasingSubsequence(n, sequence)

{

    initialize dp[n]  // Array to hold the lengths of subsequences

    maxLength = 1  // Initialize the maximum length


    // Initialize dp array where each element is 1

    for i from 0 to n - 1

    {

        dp[i] = 1

    }


    // Calculate the length of the longest non-decreasing subsequence

    for i from 1 to n - 1

    {

        for j from 0 to i - 1

        {
            if sequence[j] <= sequence[i]
            {
                dp[i] = max(dp[i], dp[j] + 1)  // Update dp[i] if a longer subsequence is found
            }
        }

        maxLength = max(maxLength, dp[i])  // Update the maximum length found
    }

    return maxLength  // Return the length of the longest non-decreasing subsequence
}

function main()
{
    initialize n  // Number of elements in the sequence
    read n from user

    initialize sequence[n]  // Array to hold the sequence

    // Read values into the sequence
    for i from 0 to n - 1
    {
        read sequence[i] from user
    }

    result = longestNonDecreasingSubsequence(n, sequence)  // Calculate result
    print result  // Print the result
```

```
}
```

Program:

```
#include<stdio.h>
int max(int a,int b)
{
    return (a>b)?a:b;
}
int main()
{
    int n;
    scanf("%d",&n);
    int sequence[n];
    int arr[n];
    for(int i=0;i<n;i++)
    {
        scanf("%d",&sequence[i]);
        arr[i]=1;
    }
    for(int i=1;i<n;i++)
    {
        for(int j=0;j<i;j++)
        {
            if(sequence[i]>=sequence[j])
                arr[i]=max(arr[i],arr[j]+1);
        }
    }
    int result=arr[0];

    for(int i=1;i<n;i++)

    {

        if(arr[i]>result)

            result=arr[i];

    }

    printf("%d",result);

}
```

Output:



	Input	Expected	Got	
✓	9 -1 3 4 5 2 2 2 2 3	6	6	✓
✓	7 1 2 2 4 5 7 6	6	6	✓

Ex. No: 6 **Date: 17.09.24**

Register No.: 230701374 **Name: Velan A**

Competitive Programming

6.a. Finding Duplicates-O(n^2) Time Complexity (1) Space Complexity

Aim: Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

n Lines - n Elements

Output Format:

Element x - That is repeated

Algorithm:

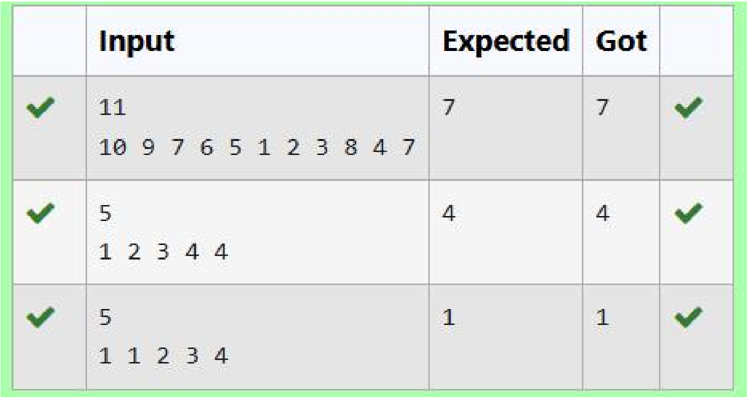
```
function main()
{
```

```
initialize n  // Number of elements in the array
read n from user

initialize arr[n]  // Array to hold input values

// Read values into the array
for i from 0 to n - 1
{
    read arr[i] from user
}

flag = 0  // Initialize a flag to indicate if a duplicate is found

// Search for the first duplicate element
for i from 0 to n - 1
{
    el1 = arr[i]  // Current element

    for j from 0 to n - 1
    {
        // Check for duplicates and ensure indices are different
        if el1 == arr[j] and i != j
        {
            print el1  // Print the duplicate element
            flag = 1  // Set flag to indicate a duplicate was found
            break  // Exit inner loop
        }
        }

        if flag
            break  // Exit outer loop if a duplicate was found
    }
}
```

Program:

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    int i,j;
    int arr[n];
    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])
            {
                printf("%d",arr[i]);
                break;
            }
        }
    }

    return 0;
}
```

Output:



	Input	Expected	Got	
✓	11 10 9 7 6 5 1 2 3 8 4 7	7	7	✓
✓	5 1 2 3 4 4	4	4	✓
✓	5 1 1 2 3 4	1	1	✓

6.b. Finding Duplicates-O(n) Time Complexity (1) Space Complexity

Aim: Find Duplicate in Array.

Given a read only array of n integers between 1 and n, find one number that repeats.

Input Format:

First Line - Number of elements

n Lines - n Elements

Output Format:

Element x - That is repeated

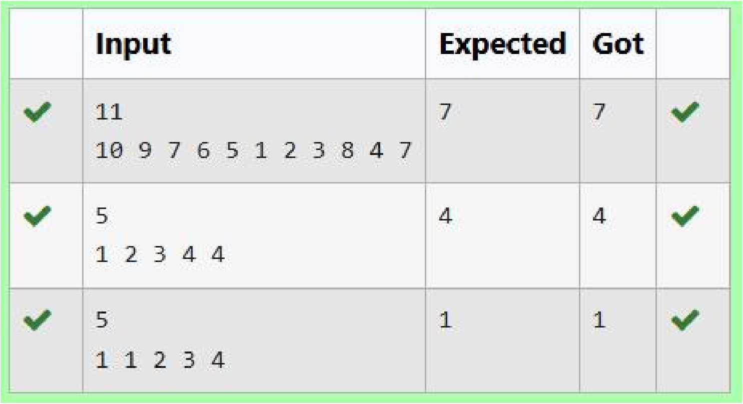
Algorithm:

```
function main()
{
    initialize n  // Number of elements in the array
    read n from user

    initialize a[n]  // Array to hold input values

    // Read values into the array
    for i from 0 to n - 1
    {
        read a[i] from user
    }

    initialize b[n]  // Array to keep track of seen elements
    for i from 0 to n - 1
    {
        b[i] = 0  // Initialize the tracking array
    }

    // Search for the first duplicate element
    for i from 0 to n - 1
    {
        // If the element is already present, i.e., b[a[i]] = 1
        if b[a[i]]
        {
            print a[i]  // Print the duplicate element
            break  // Exit the loop
        }
        else
        {
            b[a[i]] = 1  // Mark the element as seen
        }
    }
}
```

Program:

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d",&n);
    int i,j;

    int arr[n];

    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])

            {

                printf("%d",arr[i]);

                break;

            }

        }

    }

    return 0;

}
```

Output:



	Input	Expected	Got	
✓	11 10 9 7 6 5 1 2 3 8 4 7	7	7	✓
✓	5 1 2 3 4 4	4	4	✓
✓	5 1 1 2 3 4	1	1	✓

6.c. Print Intersection of 2 sorted arrays-O(m*n)Time Complexity,O(1) Space Complexity

Aim:

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

- The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

1

6 1 2 3 4 5 6

2 1 6

Output:

1 6

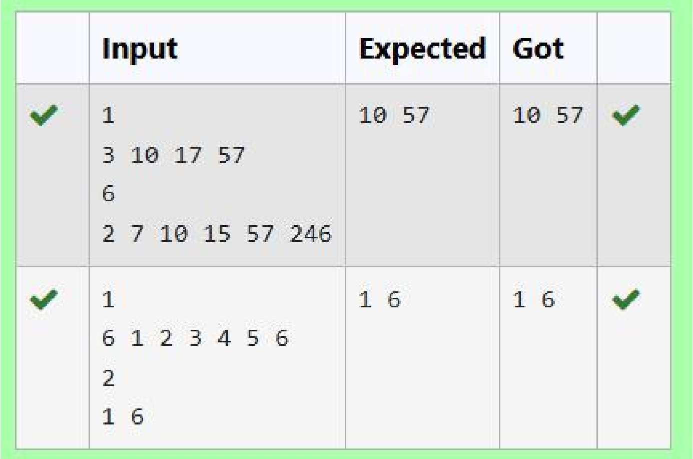
Algorithm:

```
function main()
{
    initialize n  // Number of test cases
    read n from user

    for i from 0 to n - 1
    {
        initialize n1  // Size of the first array
        read n1 from user

        initialize arr1[n1]  // First array

        // Read values into the first array
        for j from 0 to n1 - 1
        {
            read arr1[j] from user
        }

        initialize n2  // Size of the second array
        read n2 from user

        initialize arr2[n2]  // Second array

        // Read values into the second array
        for j from 0 to n2 - 1
        {
            read arr2[j] from user
        }

        // Check for common elements in both arrays
        for j from 0 to n1 - 1
        {
            for k from 0 to n2 - 1
            {
                if arr1[j] == arr2[k]
                {
                    print arr1[j]  // Print the common element
                }
            }
        }
    }
}
```

Program:

```
#include<stdio.h>
int main(){
    int t;
    scanf("%d",&t);
    int n,m;
    scanf("%d",&n);
    int arr1[n];
    for(int i=0;i<n;i++)
        scanf("%d",&arr1[i]);
    scanf("%d",&m);
    int arr2[m];
    for(int i=0;i<m;i++)
        scanf("%d",&arr2[i]);
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(arr1[i]==arr2[j])
            {
                printf("%d ",arr1[i]);
                break;
            }
        }
    }
}
```

Output:



	Input	Expected	Got	
✓	1 3 10 17 57 6 2 7 10 15 57 246	10 57	10 57	✓
✓	1 6 1 2 3 4 5 6 2 1 6	1 6	1 6	✓

6.d. Print Intersection of 2 sorted arrays-O(m+n)Time Complexity,O(1) Space Complexity

Aim:

Find the intersection of two sorted arrays.

OR in other words,

Given 2 sorted arrays, find all the elements which occur in both the arrays.

Input Format

- The first line contains T, the number of test cases. Following T lines contain:

1. Line 1 contains N1, followed by N1 integers of the first array
2. Line 2 contains N2, followed by N2 integers of the second array

Output Format

The intersection of the arrays in a single line

Example

Input:

1

3 10 17 57

6 2 7 10 15 57 246

Output:

10 57

Input:

1

6 1 2 3 4 5 6

2 1 6

Output:

1 6

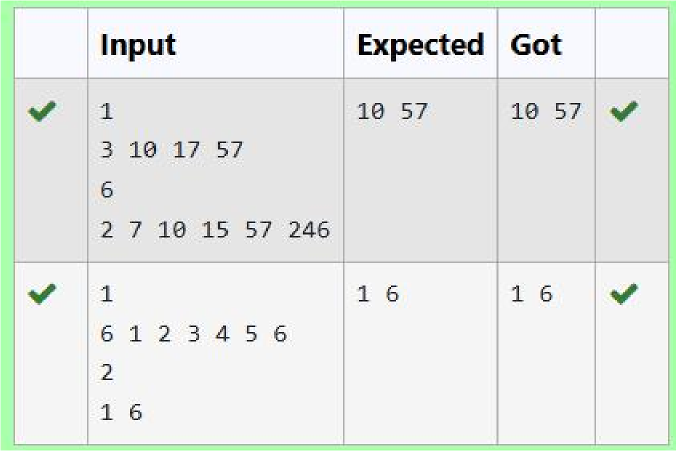
Algorithm:

```
function main()
{
    initialize T  // Number of test cases
    read T from user

    while T > 0
    {
        // Decrement the test case counter
        T--

        initialize n1, n2  // Sizes of the two arrays
        read n1 from user
        initialize arr1[n1]  // First array

        // Read values into the first array
        for i from 0 to n1 - 1
        {
            read arr1[i] from user
        }

        read n2 from user
        initialize arr2[n2]  // Second array

        // Read values into the second array
        for i from 0 to n2 - 1
        {
            read arr2[i] from user
        }

        initialize i = 0, j = 0  // Indices for both arrays

        // Iterate through both arrays to find common elements
        while i < n1 and j < n2
        {
            if arr1[i] < arr2[j]
            {
                i++  // Move to the next element in arr1
            }
            else if arr2[j] < arr1[i]
            {
                j++  // Move to the next element in arr2
            }
            else
            {
                print arr1[i]  // Print the common element
                i++  // Move to the next element in arr1
                j++  // Move to the next element in arr2
            }
        }

        print new line  // Move to the next line for output
    }
}
```

Program:

```
#include<stdio.h>
int main(){
    int t;
    scanf("%d",&t);
    int n,m;
    scanf("%d",&n);
    int arr1[n];
    for(int i=0;i<n;i++)
        scanf("%d",&arr1[i]);
    scanf("%d",&m);
    int arr2[m];
    for(int i=0;i<m;i++)
        scanf("%d",&arr2[i]);
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(arr1[i]==arr2[j])
            {
                printf("%d ",arr1[i]);

                break;

            }

        }

    }

}
```

Output:



	Input	Expected	Got	
✓	1 3 10 17 57 6 2 7 10 15 57 246	10 57	10 57	✓
✓	1 6 1 2 3 4 5 6 2 1 6	1 6	1 6	✓

6.e. Pair with Difference-O(n^2)Time Complexity,O(1) Space Complexity

Aim:

Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:

1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

YES as 5 - 1 = 4

So Return 1.

Algorithm:

```
function main()

{

    initialize n  // Number of elements in the array

    read n from user


    initialize arr[n]  // Array to hold input values


    // Read values into the array

    for i from 0 to n - 1
```

```
{
    read arr[i] from user
}

initialize t  // Target difference
read t from user

initialize flag = 0  // Flag to indicate if a pair is found

// Check for pairs with the specified difference
for i from 0 to n - 1
{
    for j from 0 to n - 1
    {
        if i != j and abs(arr[i] - arr[j]) == t
        {
            flag = 1  // Pair found
            break
        }
    }
    if flag
    {
        break
    }
}

// Output the result based on the flag
```

```
    if flag
    {
        print 1  // Pair found
    }
    else
    {
        print 0  // No pair found
    }

    return 0
}
```

Program:

```
#include <stdio.h>
int main()
{
    int size;
    scanf("%d",&size);
    int arr[size];
    for(int i = 0 ; i < size ; i++)
    {
        scanf("%d",&arr[i]);
    }
    int k;
    scanf("%d",&k);
    for(int i = 0 ; i < size ; i++)
    {
        for(int j = i+1 ; j < size ; j++)

        {

            if(arr[j] - arr[i] == k)

            {

                printf("%d",1);

                return 1;

            }

        }

    }

    printf("%d",0);

}
```

6.f. Pair with Difference -O(n) Time Complexity,O(1) Space Complexity

Aim: Given an array A of sorted integers and another non negative integer k, find if there exists 2 indices i and j such that A[j] - A[i] = k, i != j.

Input Format:

First Line n - Number of elements in an array

Next n Lines - N elements in the array

k - Non - Negative Integer

Output Format:

1 - If pair exists

0 - If no pair exists

Explanation for the given Sample Testcase:

YES as 5 - 1 = 4

So Return 1.

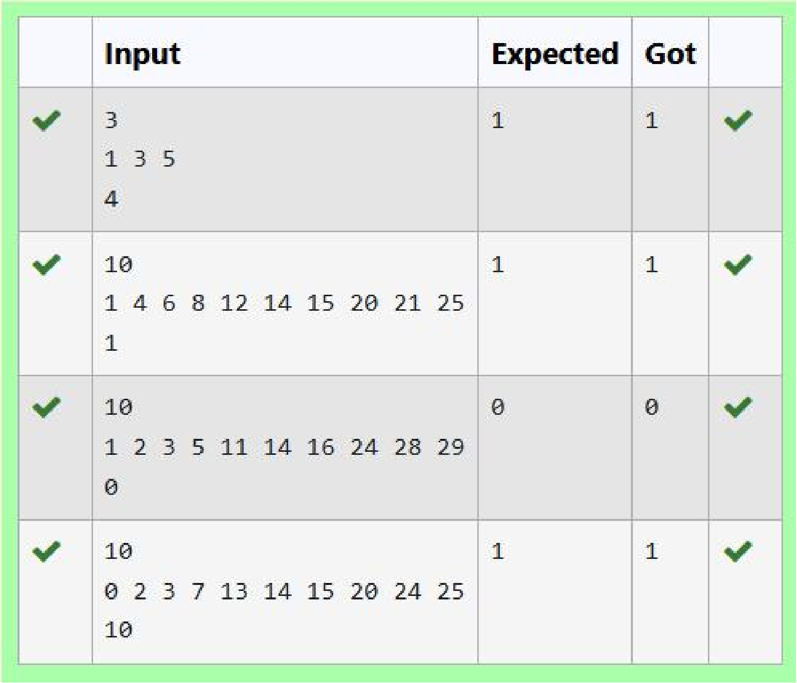
Algorithm:

```
function main()
{
   initialize n  // Number of elements in the array
   read n from user

   initialize arr[n]  // Array to hold input values

   // Read values into the array
   for i from 0 to n - 1
   {
    read arr[i] from user
}

initialize t  // Target difference
read t from user

initialize flag = 0  // Flag to indicate if a pair is found

initialize i = 0  // First index
initialize j = 1  // Second index

// Loop to find pairs with the specified difference
while i < n and j < n
{
    diff = abs(arr[i] - arr[j])  // Calculate the difference

    if i != j and diff == t
    {
        flag = 1  // Pair found
        break
    }
    else if diff < t
    {
        j++  // Increment second index
    }
    else
    {
            i++  // Increment first index
        }
    }

    // Output the result based on the flag
    if flag
    {
        print 1  // Pair found
    }
    else
    {
        print 0  // No pair found
    }

    return 0
}
```

Program:

```
#include<stdio.h>
int main(){
    int n,k;
    scanf("%d",&n);
    int arr[n];
    for(int i=0;i<n;i++)
        scanf("%d",&arr[i]);
    scanf("%d",&k);
    int i=0,j=1,found=0;
    while(j<n){
        int diff=arr[j]-arr[i];
        if(i!=j && diff==k){
            found=1;break;
        }
        else if(diff<k) j++;
        else{
            i++;
            if(i==j) j++;
        }
    }
    if(found) printf("1");
    else printf("0");
}
```

Output:



	Input	Expected	Got	
✓	3 1 3 5 4	1	1	✓
✓	10 1 4 6 8 12 14 15 20 21 25 1	1	1	✓
✓	10 1 2 3 5 11 14 16 24 28 29 0	0	0	✓
✓	10 0 2 3 7 13 14 15 20 24 25 10	1	1	✓