#### CODILITY QUESTIONS:

#### 1)[BinaryGap](https://app.codility.com/programmers/lessons/1-iterations/binary_gap/):

**Find longest sequence of zeros in binary representation of an integer.A *binary gap* within a positive integer N is any maximal sequence of consecutive zeros that is surrounded by ones at both ends in the binary representation of N.**

using System;

// you can also use other imports, for example:

// using System.Collections.Generic;

// you can write to stdout for debugging purposes, e.g.

// Console.WriteLine("this is a debug message");

class Solution {

    public int solution(int N) {

int work = N;

while (work > 0 && (work & 1) == 0) {

work >>= 1;

}

work >>= 1;

int max = 0;

int zeros = 0;

while (work > 0) {

if ((work & 1) == 0) {

zeros++;

} else {

max = Math.Max(max, zeros);

zeros = 0;

}

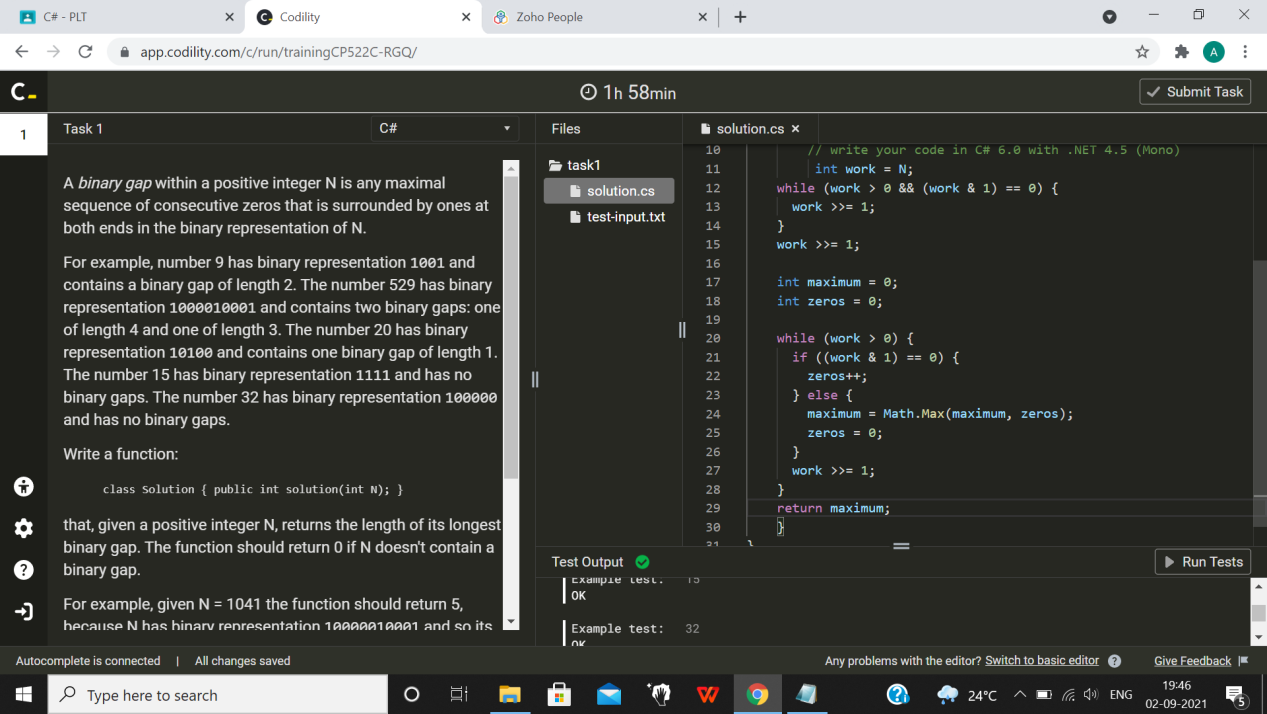
work >>= 1;

}

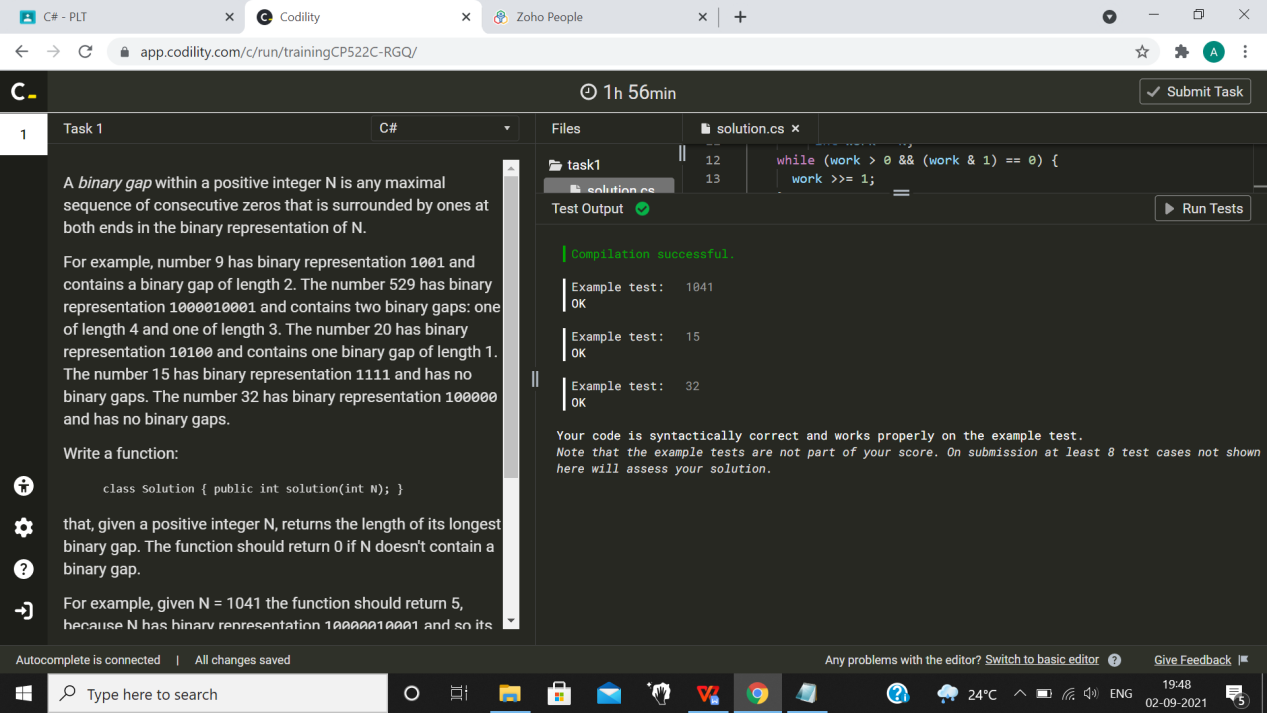
return max;

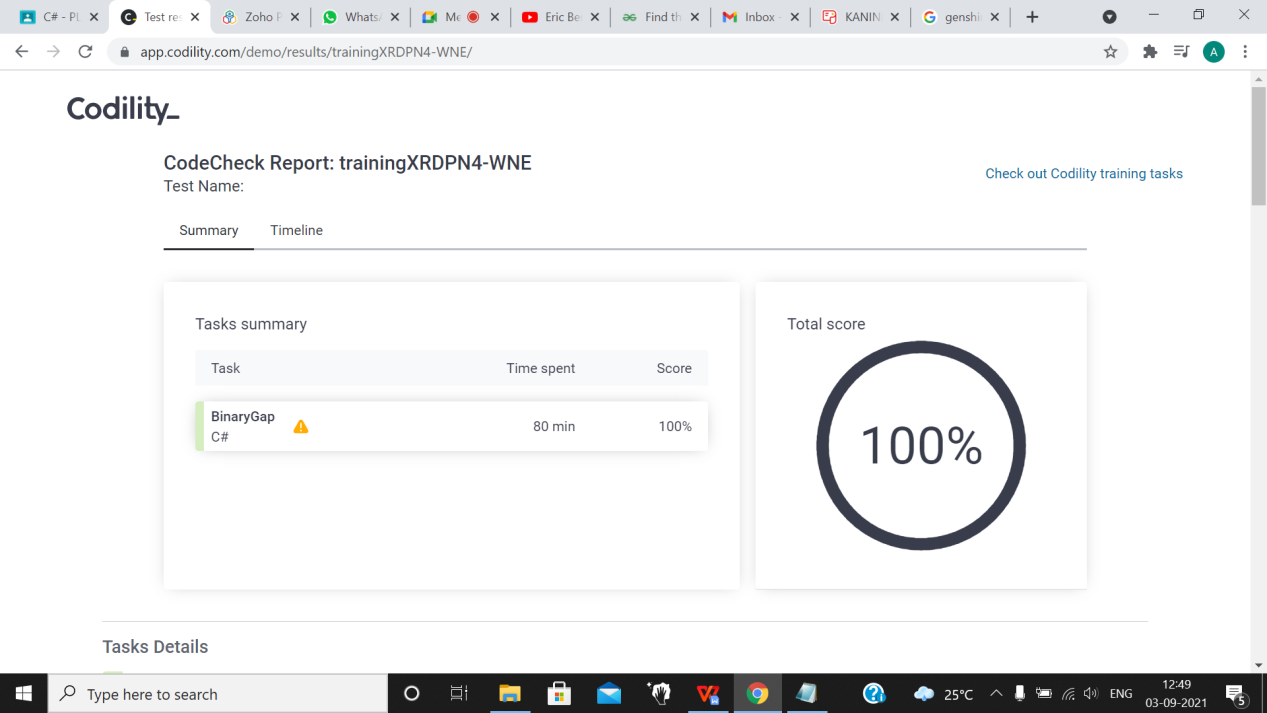
}

}



OUTPUT:





#### [CyclicRotation](https://app.codility.com/programmers/lessons/2-arrays/cyclic_rotation/):

**Rotate an array to the right by a given number of steps.An array A consisting of N integers is given. Rotation of the array means that each element is shifted right by one index, and the last element of the array is moved to the first place.**

using System;

// you can also use other imports, for example:

// using System.Collections.Generic;

// you can write to stdout for debugging purposes, e.g.

// Console.WriteLine("this is a debug message");

class Solution {

    public int[] solution(int[] A, int K) {

Int[] FinalArray=new int[A.Length];

for(int i = 0; i < K; i++)

{

int temp;

temp = A[A.Length- 1];

for(int j = A.Length- 1; j > 0; j--)

{

A[j] = A[j - 1];

}

A[0] = temp;

}

for (int i = 0; I<A.Length;i++)

{

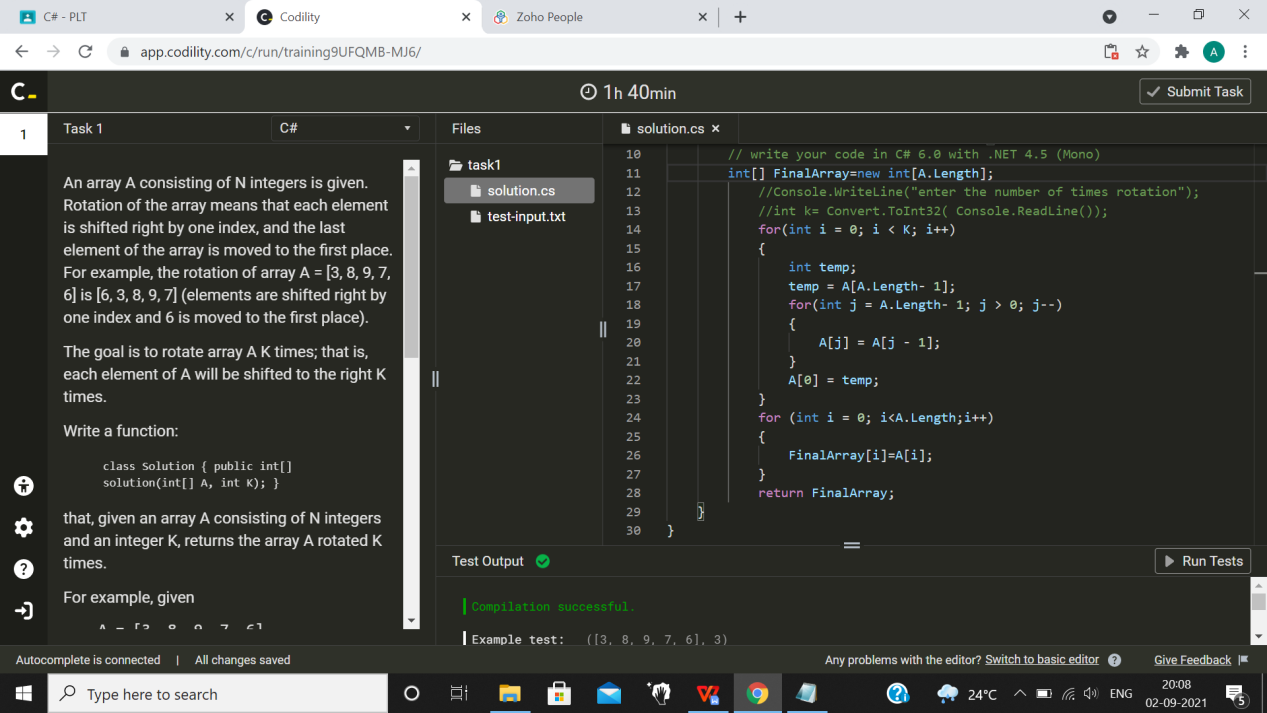
FinalArray[i]=A[i];

}

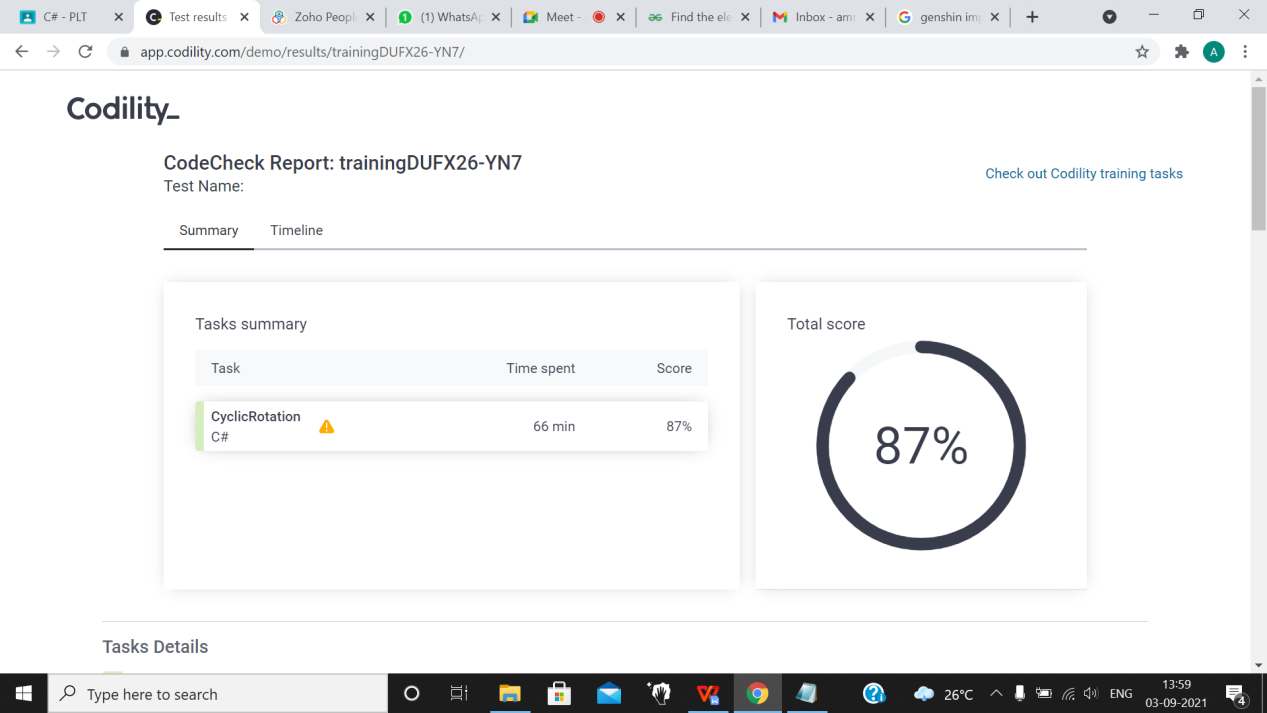
return FinalArray;

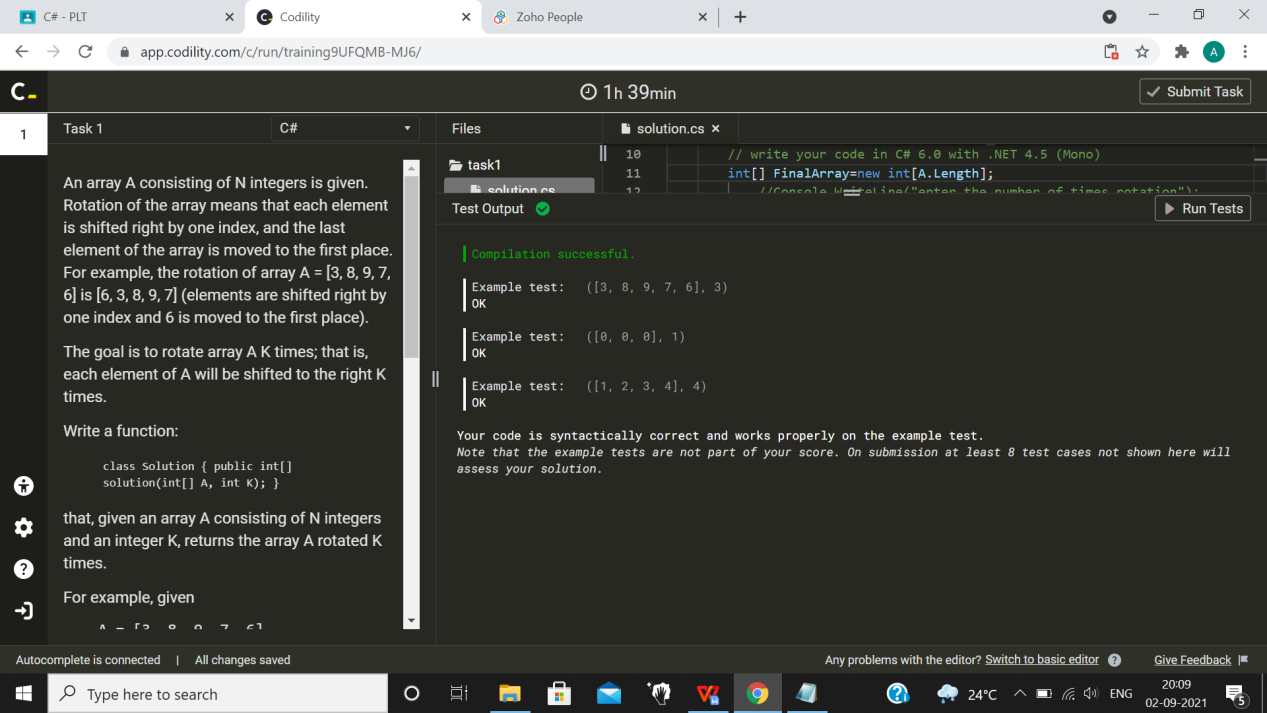
}

}



OUTPUT:





#### [OddOccurrencesInArray](https://app.codility.com/programmers/lessons/2-arrays/odd_occurrences_in_array/):

**Find value that occurs in odd number of elements.A non-empty array A consisting of N integers is given. The array contains an odd number of elements, and each element of the array can be paired with another element that has the same value, except for one element that is left unpaired.**

using System;

// you can also use other imports, for example:

// using System.Collections.Generic;

// you can write to stdout for debugging purposes, e.g.

// Console.WriteLine("this is a debug message");

class Solution {

    public int solution(int[] A) {

int res=A[0];

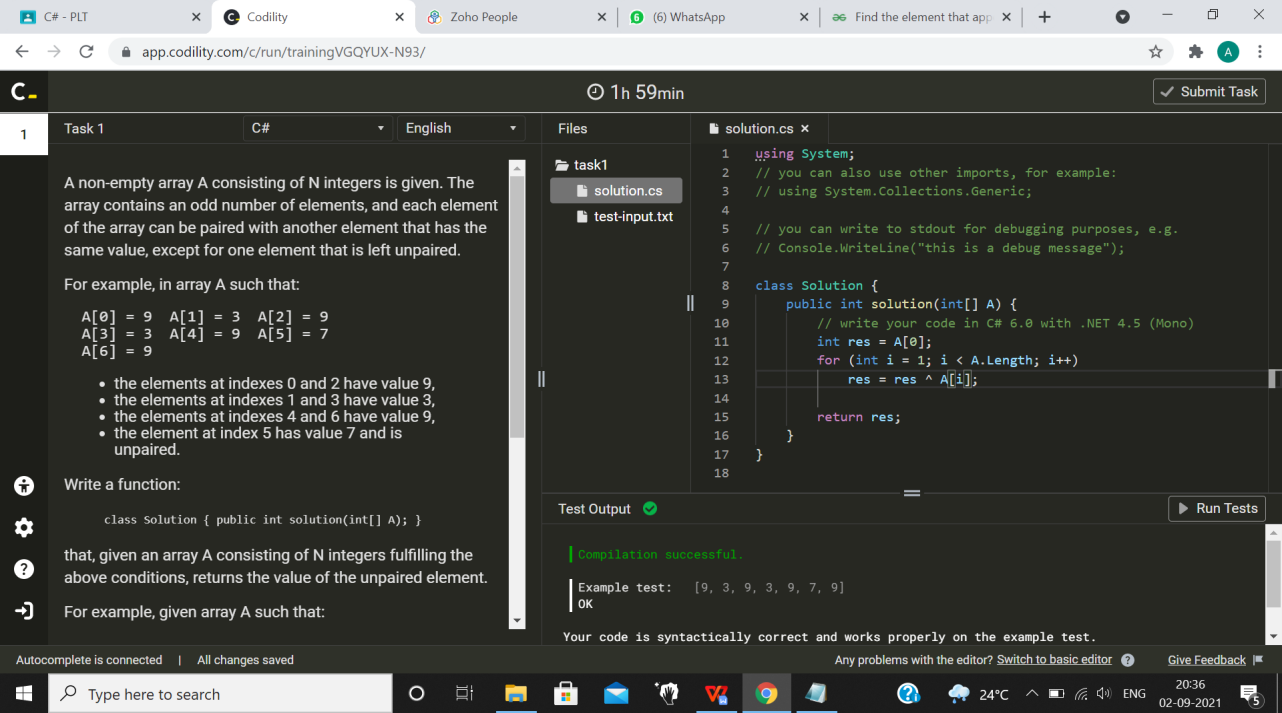
for(int i=1;i<A.Length;i++)

res=res^A[i];

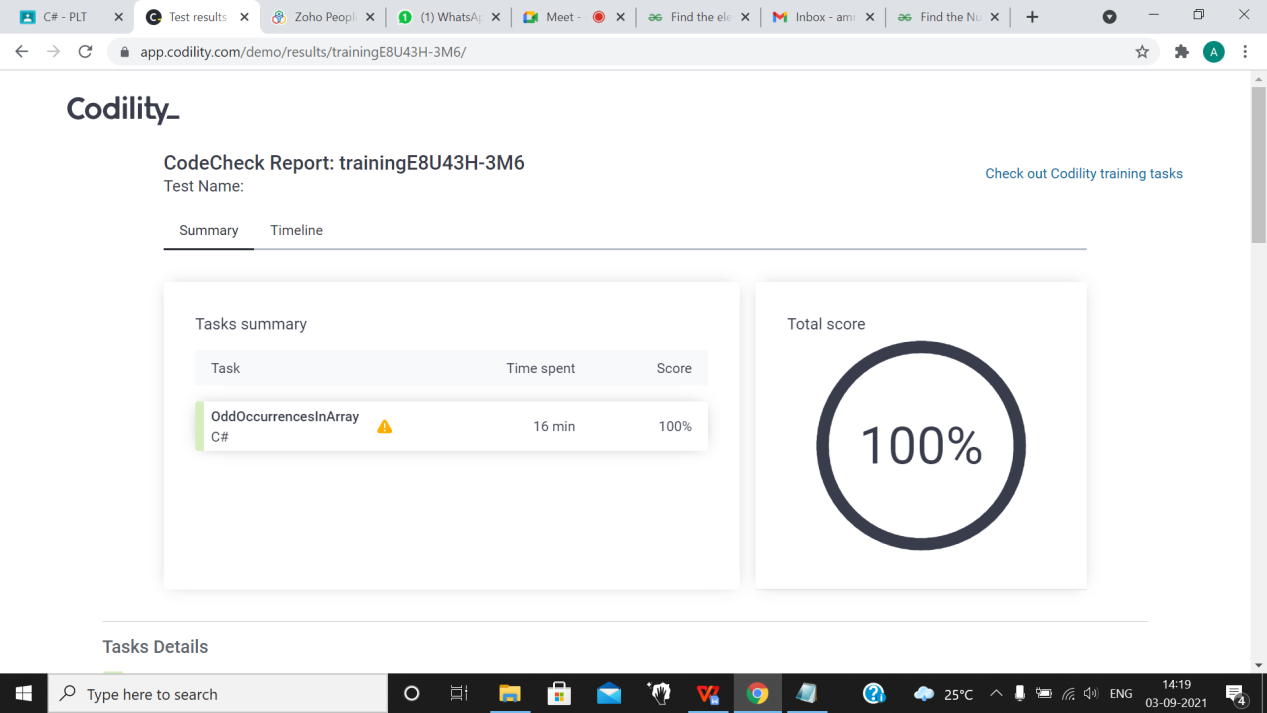
return res;

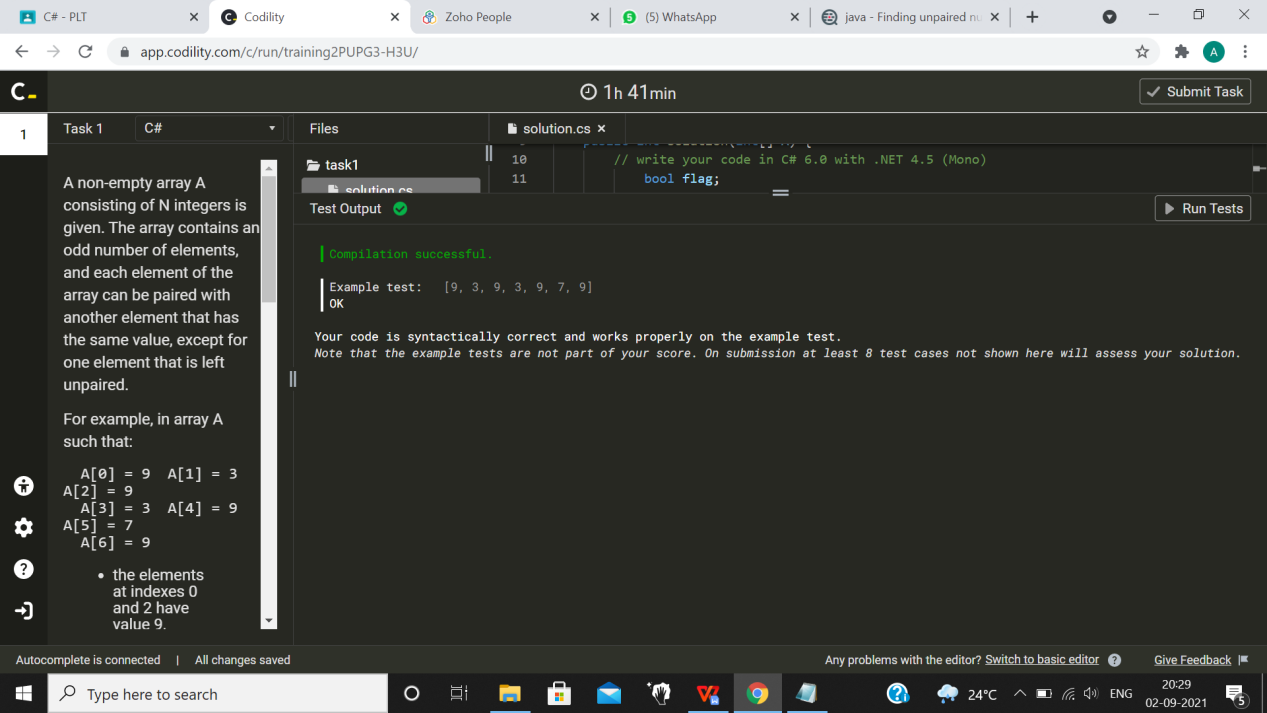
}

}



OUTPUT:





#### [PermMissingElem](https://app.codility.com/programmers/lessons/3-time_complexity/perm_missing_elem/):

**Find the missing element in a given permutation.An array A consisting of N different integers is given. The array contains integers in the range [1..(N + 1)], which means that exactly one element is missing.**

using System;

// you can also use other imports, for example:

// using System.Collections.Generic;

// you can write to stdout for debugging purposes, e.g.

// Console.WriteLine("this is a debug message");

class Solution {

    public int solution(int[] A) {

int result,sum=0;

for(int i=0;i<A.Length;i++)

{

sum = sum + A[i];

}

int n = A.Length + 1;

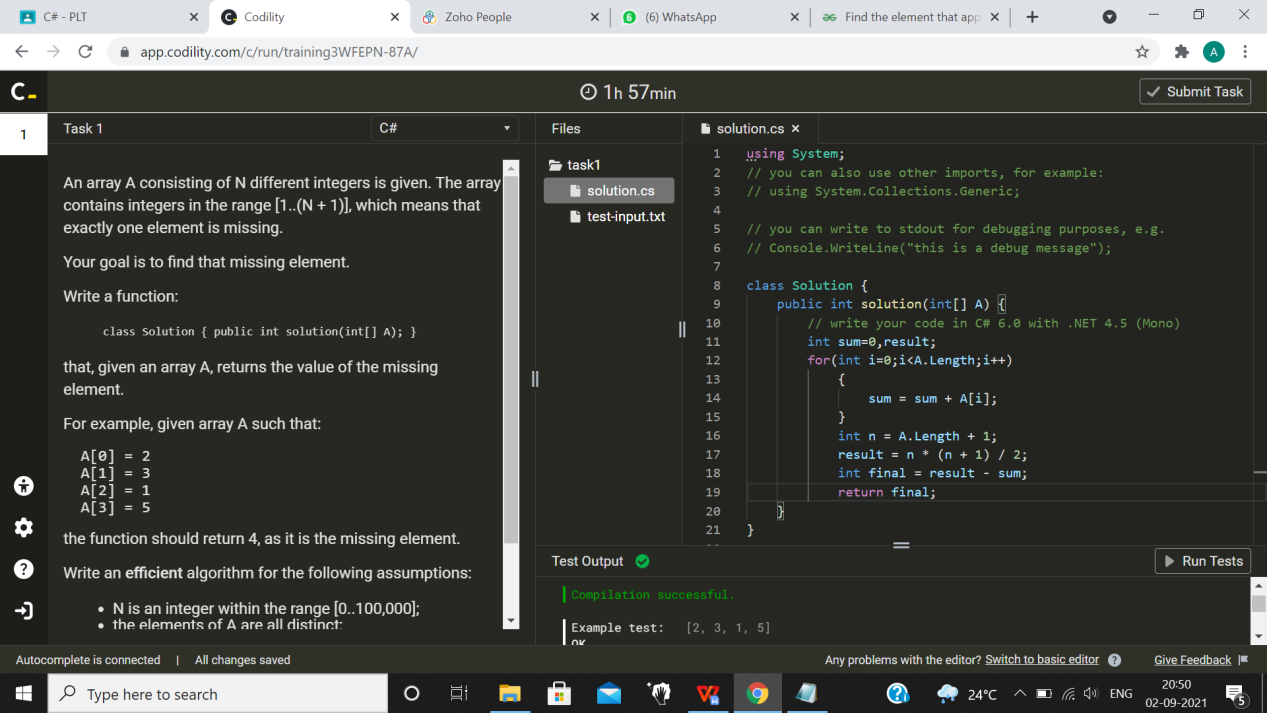
result = n \* (n + 1) / 2;

int final = result - sum;

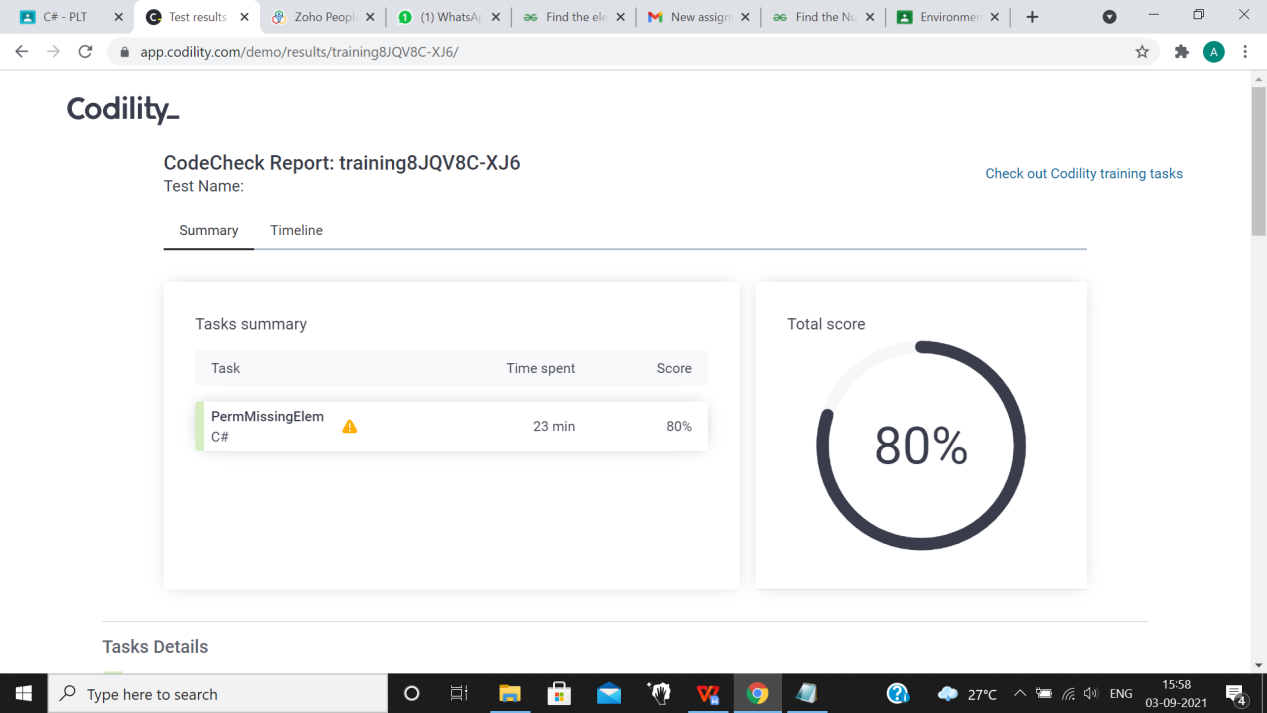
return final;

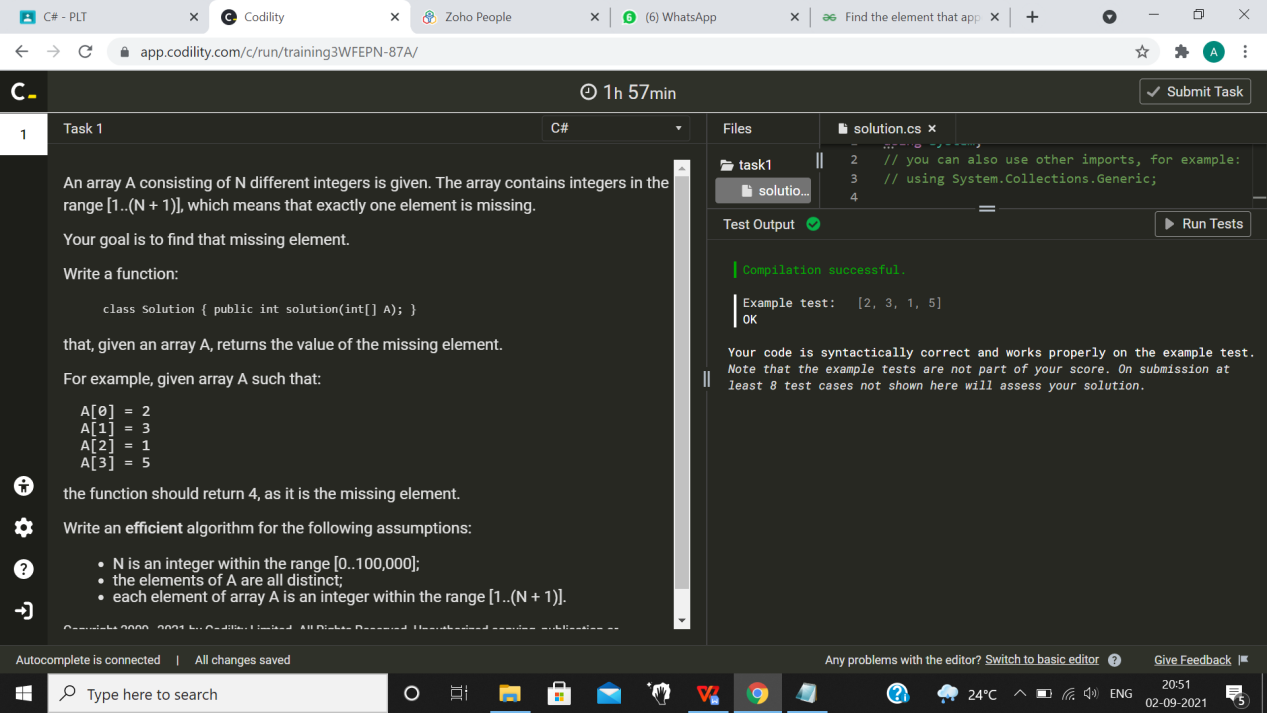
}

}



OUTPUT:





#### [TapeEquilibrium](https://app.codility.com/programmers/lessons/3-time_complexity/tape_equilibrium/):

**Minimize the value |(A[0] + ... + A[P-1]) - (A[P] + ... + A[N-1])|.A non-empty array A consisting of N integers is given. Array A represents numbers on a tape.**

**Any integer P, such that 0 < P < N, splits this tape into two non-empty parts: A[0], A[1], ..., A[P − 1] and A[P], A[P + 1], ..., A[N − 1].**

using System;

// you can also use other imports, for example:

// using System.Collections.Generic;

// you can write to stdout for debugging purposes, e.g.

// Console.WriteLine("this is a debug message");

class Solution {

    public int solution(int[] A) {

long sumMin = (int)A[0];

long sumMax = 0;

for (int i = 1; i < A.Length; i++) {

sumMax += (int)A[i];

}

int minDif = (int)Math.abs(sumMax - sumMin);

for (int i = 1; i < A.Length; i++) {

int dif = (int)Math.abs(sumMax - sumMin);

if (dif < minDif) {

minDif = dif;

}

sumMin += A[i];

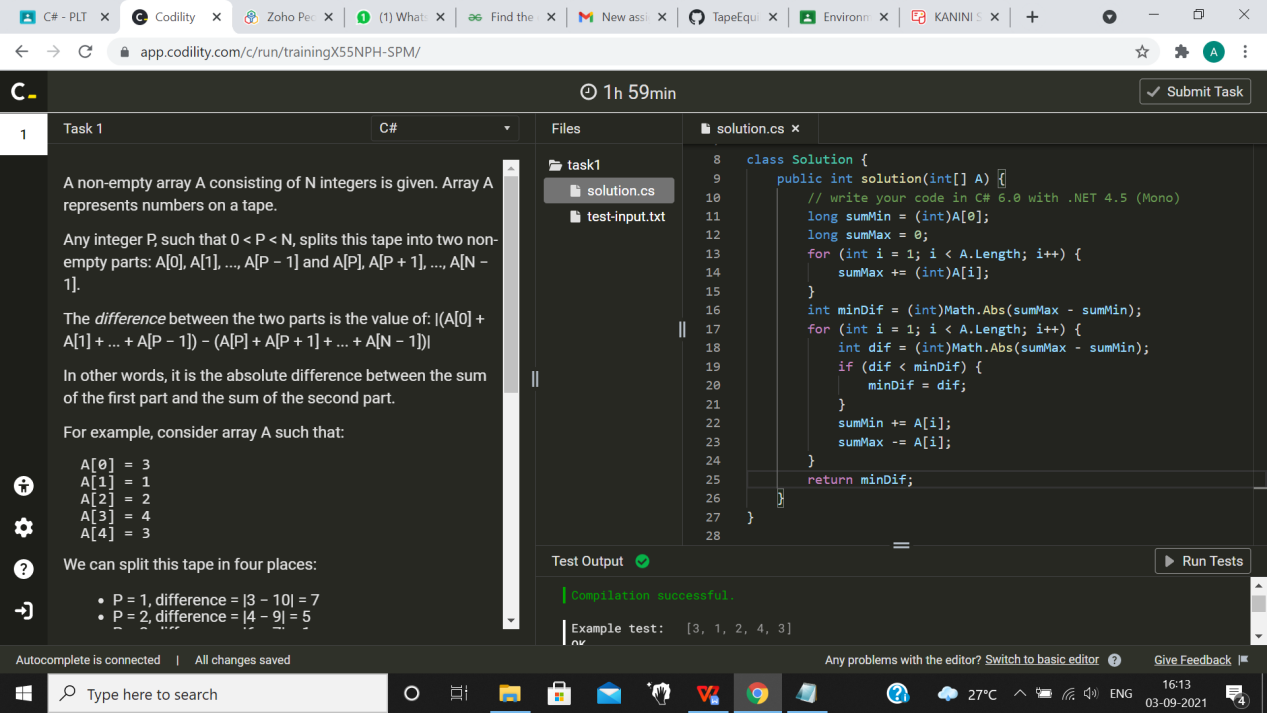
sumMax -= A[i];

}

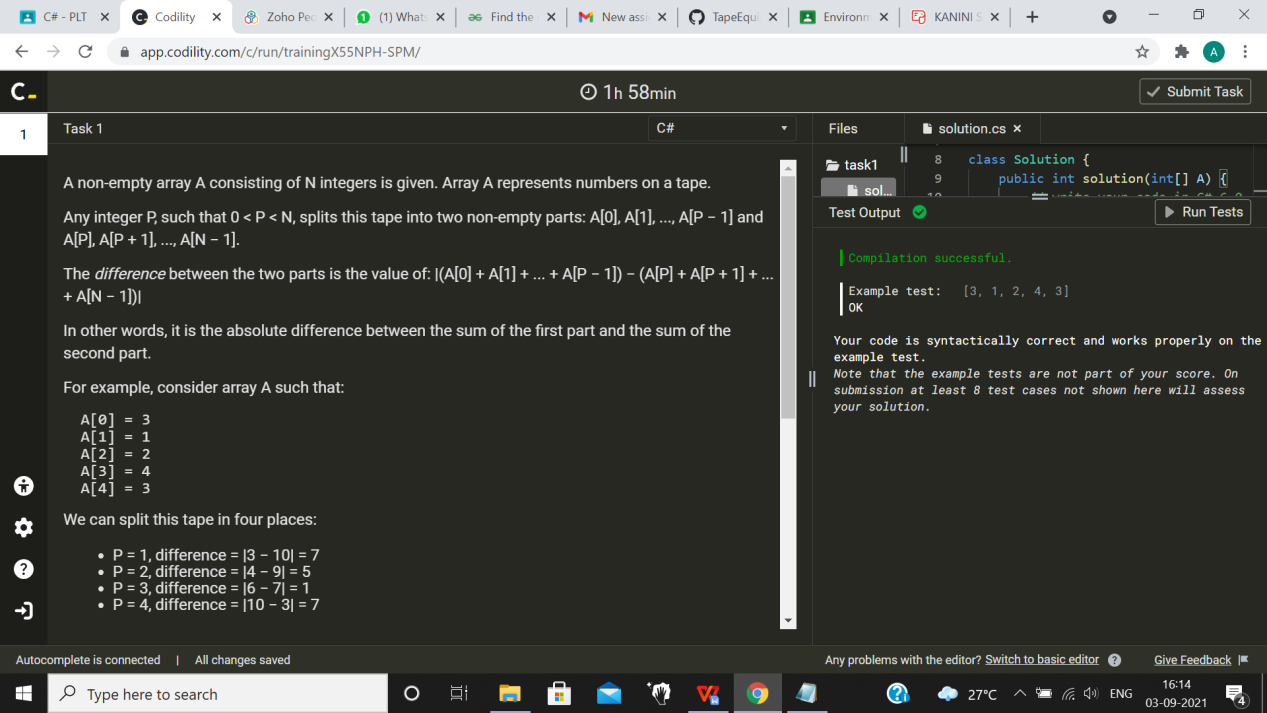
return minDif;

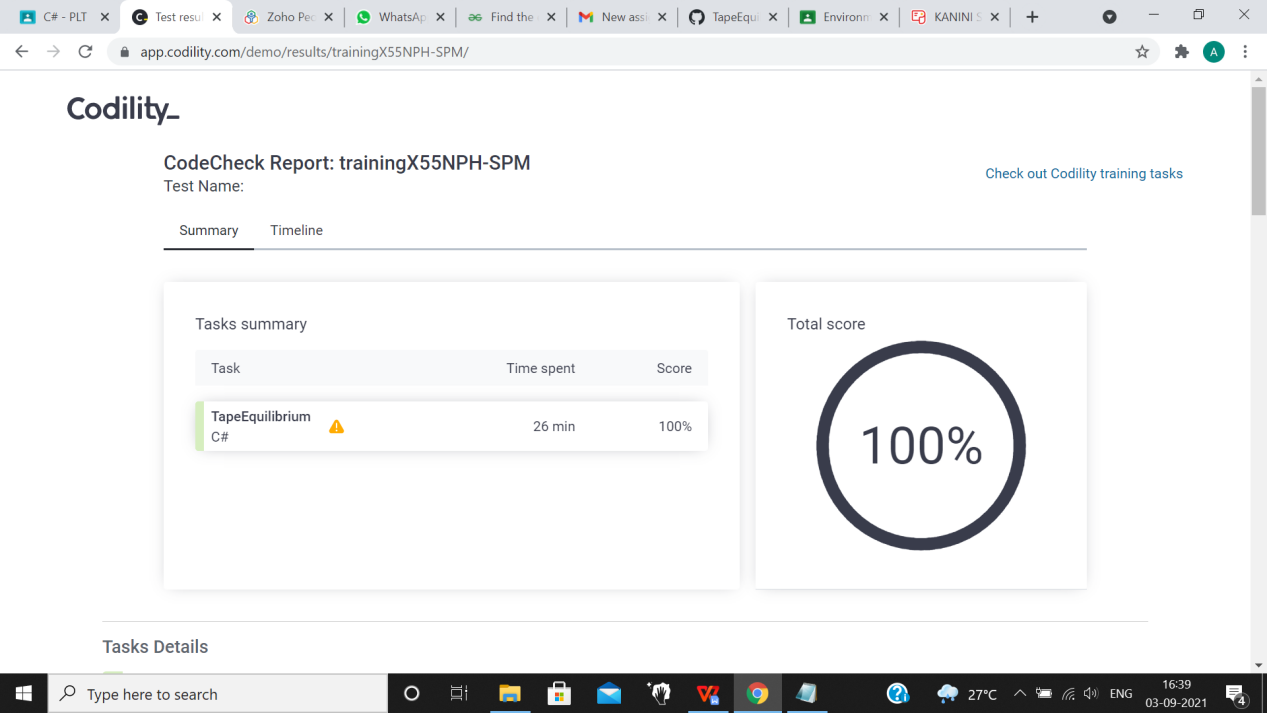
}

}



OUTPUT:





#### [FrogJmp](https://app.codility.com/programmers/lessons/3-time_complexity/frog_jmp/):

**Count minimal number of jumps from position X to Y.A small frog wants to get to the other side of the road. The frog is currently located at position X and wants to get to a position greater than or equal to Y. The small frog always jumps a fixed distance, D.**

**Count the minimal number of jumps that the small frog must perform to reach its target.**

using System;

// you can also use other imports, for example:

// using System.Collections.Generic;

// you can write to stdout for debugging purposes, e.g.

// Console.WriteLine("this is a debug message");

class Solution {

public int solution(int X, int Y, int D) {

// write your code in C# 6.0 with .NET 4.5 (Mono)

var distance = Y - X;

if (distance % D == 0)

{

return distance / D;

}

else

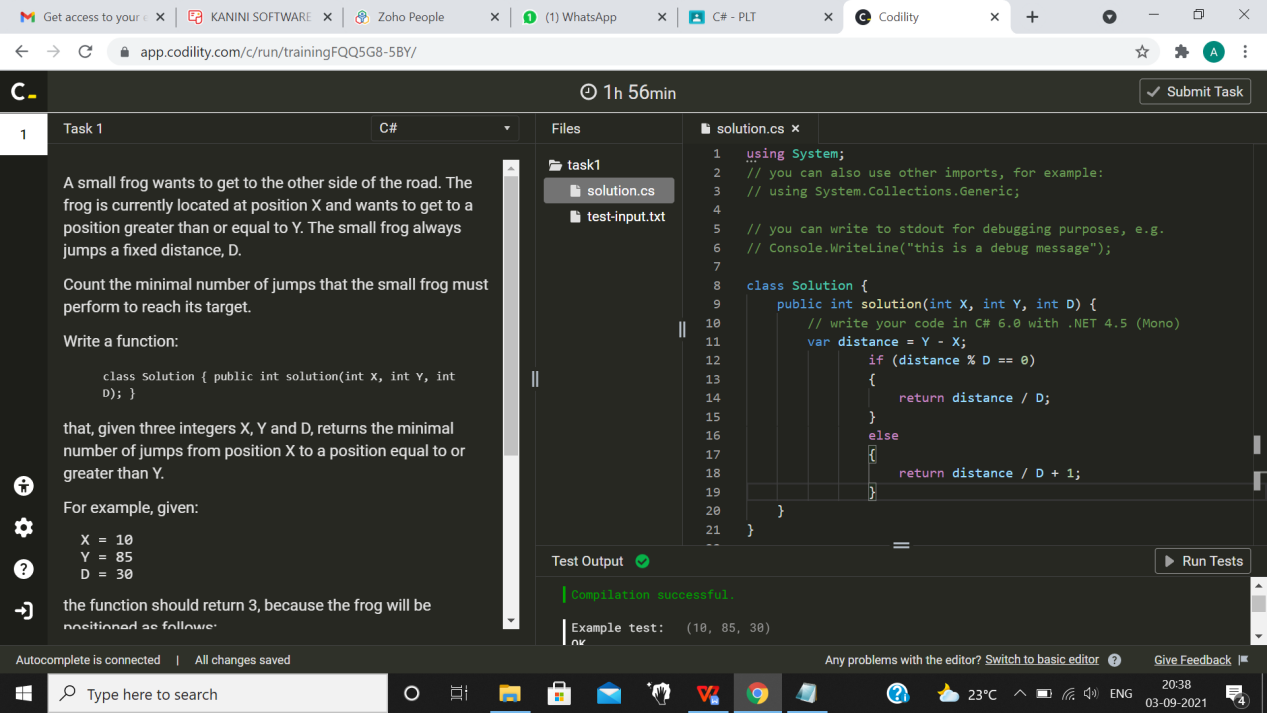
{

return distance / D + 1;

}

}

}



OUTPUT:

