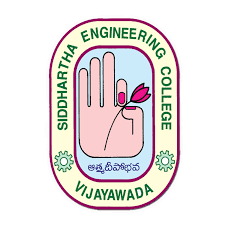
**Velagapudi Ramakrishna Siddhartha Engineering College**

**ADVANCE PROGRAMMING LAB – 2**

**Code : 20IT5352**

****

Contents

[**WEEK – 1** 3](#_Toc116238813)

[**WEEK – 2** 6](#_Toc116238814)

[**WEEK – 3** 10](#_Toc116238815)

[**WEEK – 4** 13](#_Toc116238816)

[**Task – 1:** 13](#_Toc116238817)

[**Task – 2** 16](#_Toc116238818)

[**Task – 3** 19](#_Toc116238819)

[**WEEK – 5** 22](#_Toc116238820)

[**Task – 1** 22](#_Toc116238821)

[**Task – 2** 25](#_Toc116238822)

[**Task – 3** 28](#_Toc116238823)

# **WEEK – 1**

**Aim :** Vlad enjoys listening to music. He lives in Sam's Town. A few days ago he had a birthday, so his parents gave him a gift: MP3-player! Vlad was the happiest man in the world! Now he can listen his favorite songs whenever he wants!

Vlad built up his own playlist. The playlist consists of N songs, each has a unique positive integer length. Vlad likes all the songs from his playlist, but there is a song, which he likes more than the others. It's named "Uncle Johny".

After creation of the playlist, Vlad decided to sort the songs in increasing order of their lengths. For example, if the lengths of the songs in playlist was {1, 3, 5, 2, 4} after sorting it becomes {1, 2, 3, 4, 5}. Before the sorting, "Uncle Johny" was on K-th position (1-indexing is assumed for the playlist) in the playlist.

Vlad needs your help! He gives you all the information of his playlist. Your task is to find the position of "Uncle Johny" in the sorted playlist.

**Sample Input :**

The first line of the input contains an integer **T** denoting the number of test cases. The description of **T** test cases follows.

The first line of each test case contains one integer **N** denoting the number of songs in Vlad's playlist. The second line contains **N** space-separated integers **A1**, **A2**, ..., **AN** denoting the lenghts of Vlad's songs. The third line contains the only integer **K** - the position of "Uncle Johny" in the initial playlist.

**Sample Output :**

For each test case, output a single line containing the position of "Uncle Johny" in the sorted playlist.

**Program :**

T = int(input())

while T > 0:

il = list(map(int, input().split()))

N = il[0]

M = il[-1]

if M >= N:

print("NO")

else:

if (N%M == 0) and (((N//M)\*M) == N):

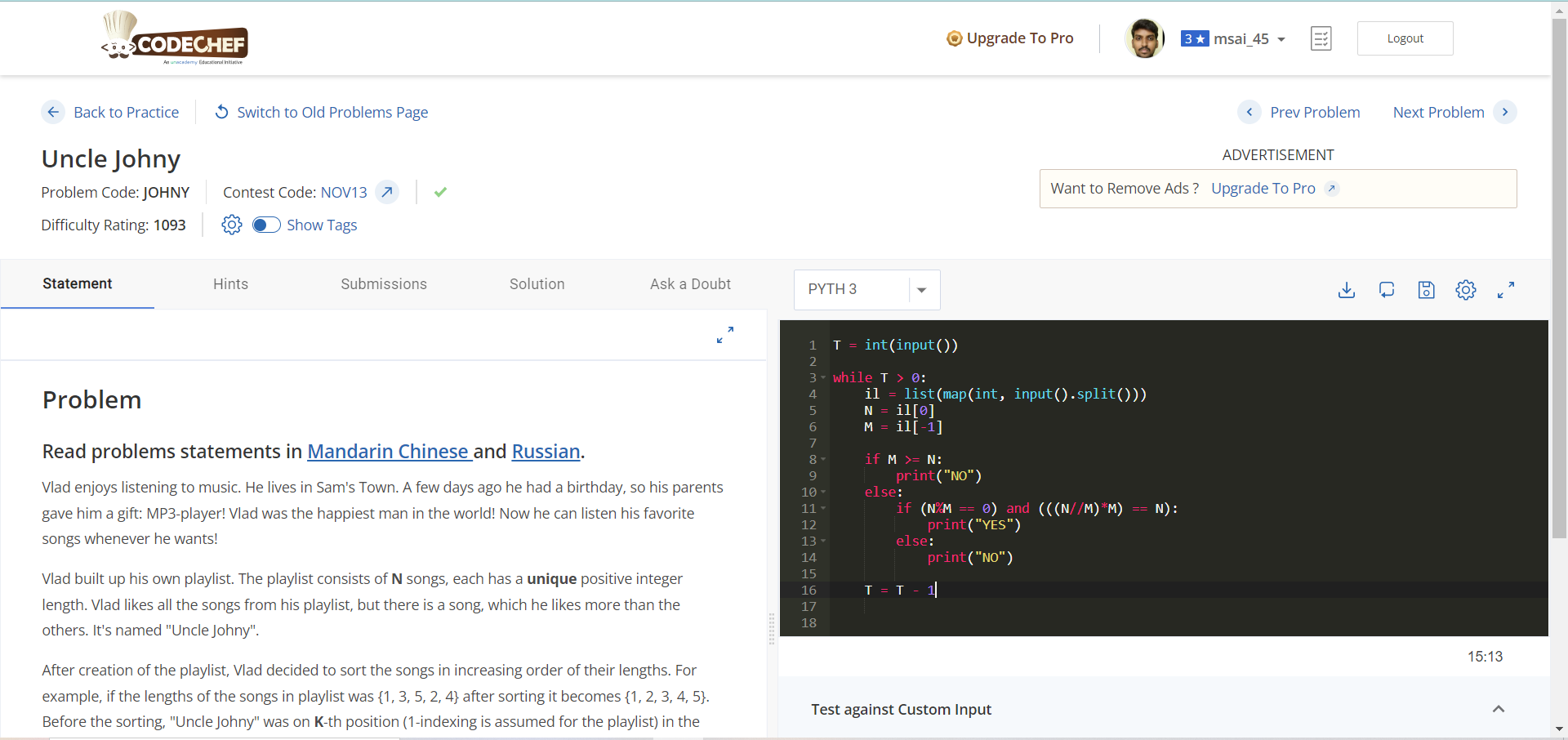
print("YES")

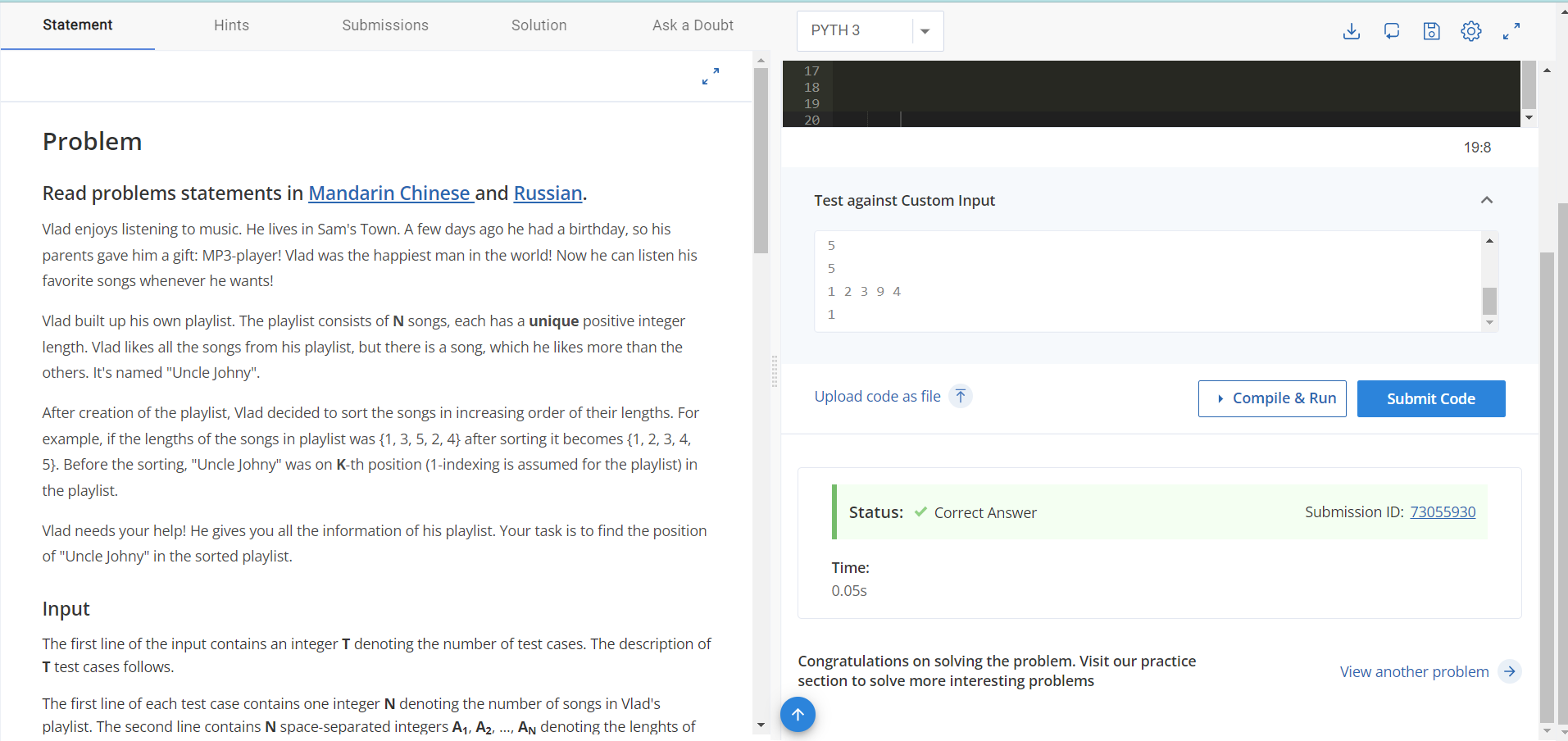
else:

print("NO")

T = T – 1

**Output :**





**Time Complexity :** O(n)

**Space Compexity :** O(n)

**Result :** Sucessfully Executed the Program.

# **WEEK – 2**

**Aim :** The game of billiards involves two players knocking 3 balls around on a green baize table. Well, there is more to it, but for our purposes this is sufficient.

The game consists of several rounds and in each round both players obtain a score, based on how well they played. Once all the rounds have been played, the total score of each player is determined by adding up the scores in all the rounds and the player with the higher total score is declared the winner.

The Siruseri Sports Club organises an annual billiards game where the top two players of Siruseri play against each other. The Manager of Siruseri Sports Club decided to add his own twist to the game by changing the rules for determining the winner. In his version, at the end of each round, the cumulative score for each player is calculated, and the leader and her current lead are found. Once all the rounds are over the player who had the maximum lead at the end of any round in the game is declared the winner.

**Sample Input :**

The first line of the input will contain a single integer N (N ≤ 10000) indicating the number of rounds in the game. Lines 2,3,...,N+1 describe the scores of the two players in the N rounds. Line i+1 contains two integer Si and Ti, the scores of the Player 1 and 2 respectively, in round i. You may assume that 1 ≤ Si ≤ 1000 and 1 ≤ Ti ≤ 1000.

**Sample Output :**

Your output must consist of a single line containing two integers W and L, where W is 1 or 2 and indicates the winner and L is the maximum lead attained by the winner.

**Program :**

import numpy as np

t = int(input())

temp = t

players = []

player1 = []

player2 = []

while t > 0:

score = list(map(int, input().split()[:2]))

player1.append(score[0])

player2.append(score[-1])

players.append(score)

t = t - 1

dict\_players = {}

dict\_players[1] = player1

dict\_players[2] = player2

p1 = dict\_players[1]

p2 = dict\_players[2]

p1 = np.array(p1)

p2 = np.array(p2)

player\_leads = list(abs(p1-p2))

leader\_round = {}

for i in range(1, temp+1):

if p1[i-1] > p2[i-1]:

value = player\_leads[i-1]

leader\_round[value] = 'p1'

else:

value = player\_leads[i-1]

leader\_round[value] = 'p2'

final\_result = sorted(leader\_round)

maxp = max(final\_result)

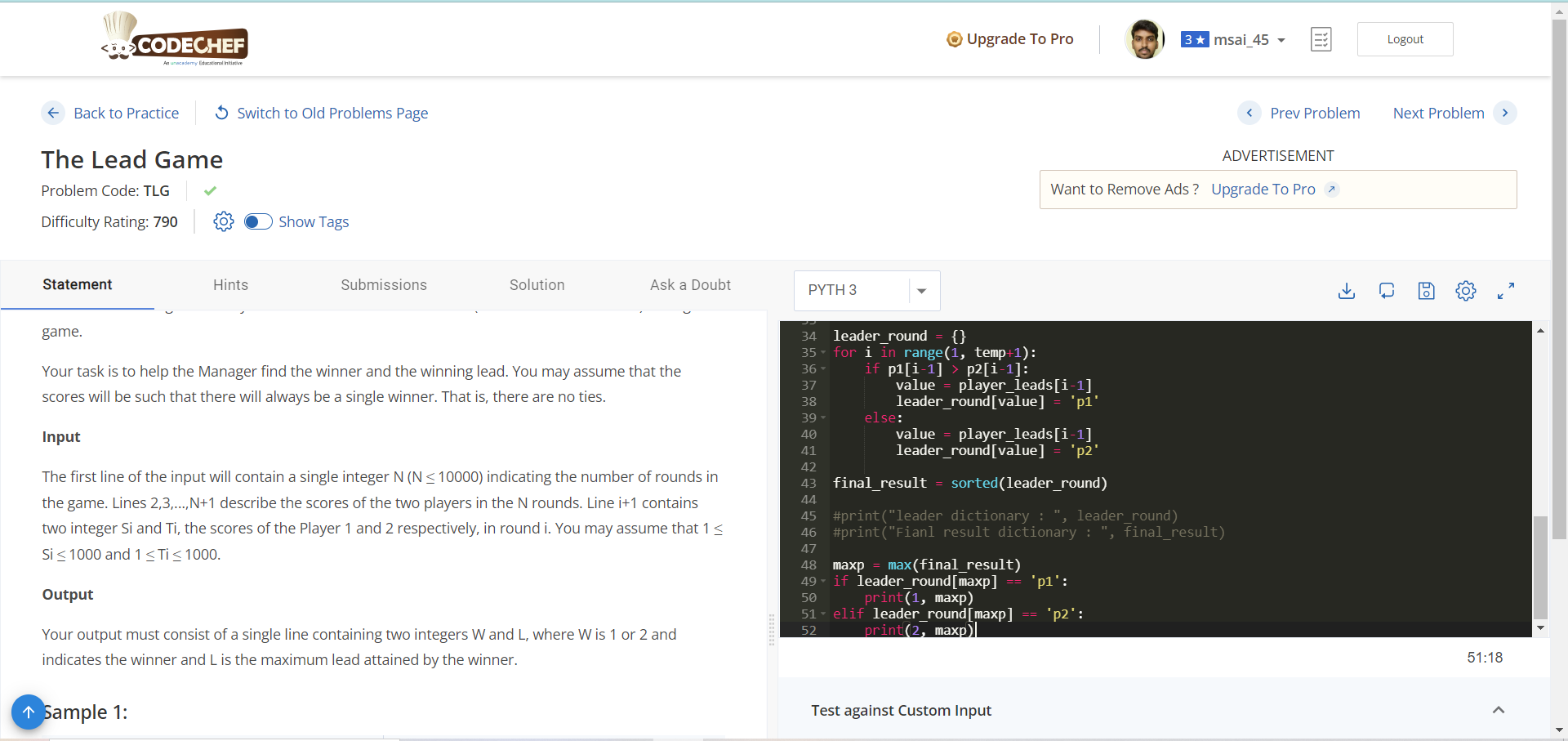
if leader\_round[maxp] == 'p1':

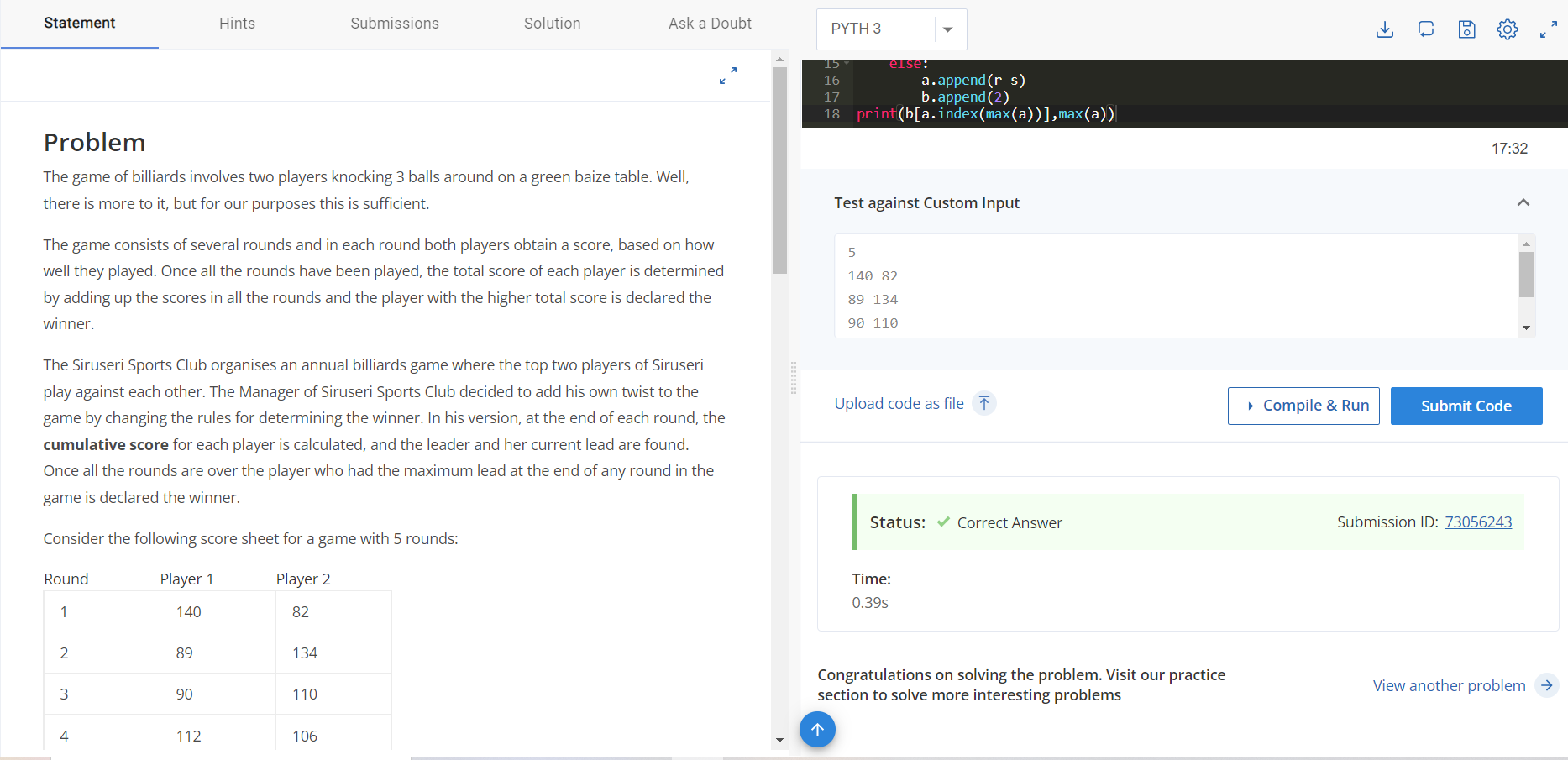
print(1, maxp)

elif leader\_round[maxp] == 'p2':

print(2, maxp)

**Output :**





**Time Complexity :** O(2n)

**Space Complexity :** O(n)

**Result** : Sucessfully Executed the program.

# **WEEK – 3**

**Aim :** Devu has n weird friends. Its his birthday today, so they thought that this is the best occasion for testing their friendship with him. They put up conditions before Devu that they will break the friendship unless he gives them a grand party on their chosen day. Formally, ith friend will break his friendship if he does not receive a grand party on dith day.

Devu despite being as rich as Gatsby, is quite frugal and can give at most one grand party daily. Also, he wants to invite only one person in a party. So he just wonders what is the maximum number of friendships he can save. Please help Devu in this tough task !!

**Sample Input :**

* The first line of the input contains an integer **T** denoting the number of test cases. The description of **T** test cases follows.
* First line will contain a single integer denoting **n**.
* Second line will contain **n** space separated integers where **i**th integer corresponds to the day **di**th as given in the problem.

**Sample Output :**

Print a single line corresponding to the answer of the problem.

**Program :**

test\_cases = int(input())

while test\_cases > 0:

no\_of\_friends = int(input())

friends\_list = list(map(int, input().split()[:no\_of\_friends]))

friends\_set = set(friends\_list)

if friends\_list == friends\_set:

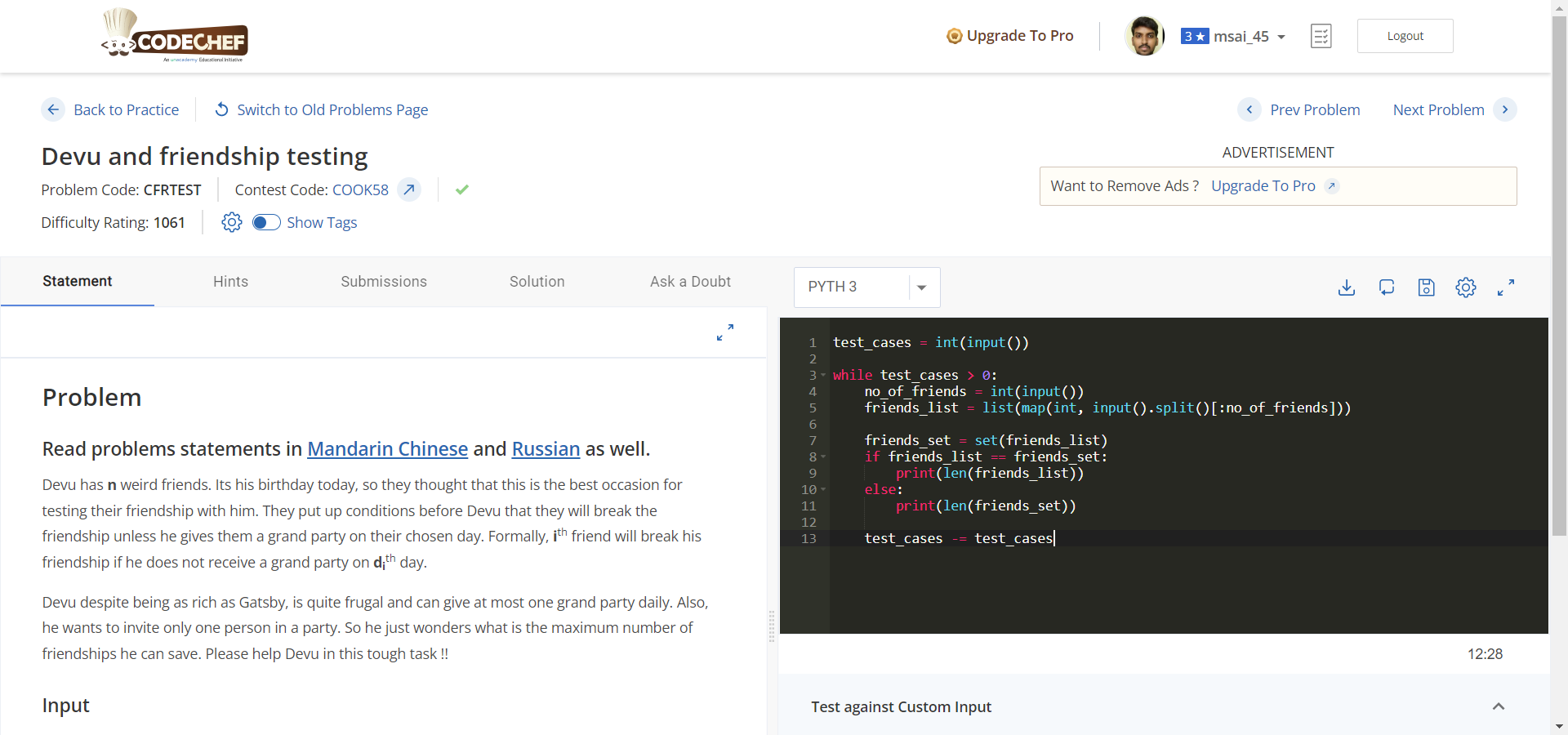
print(len(friends\_list))

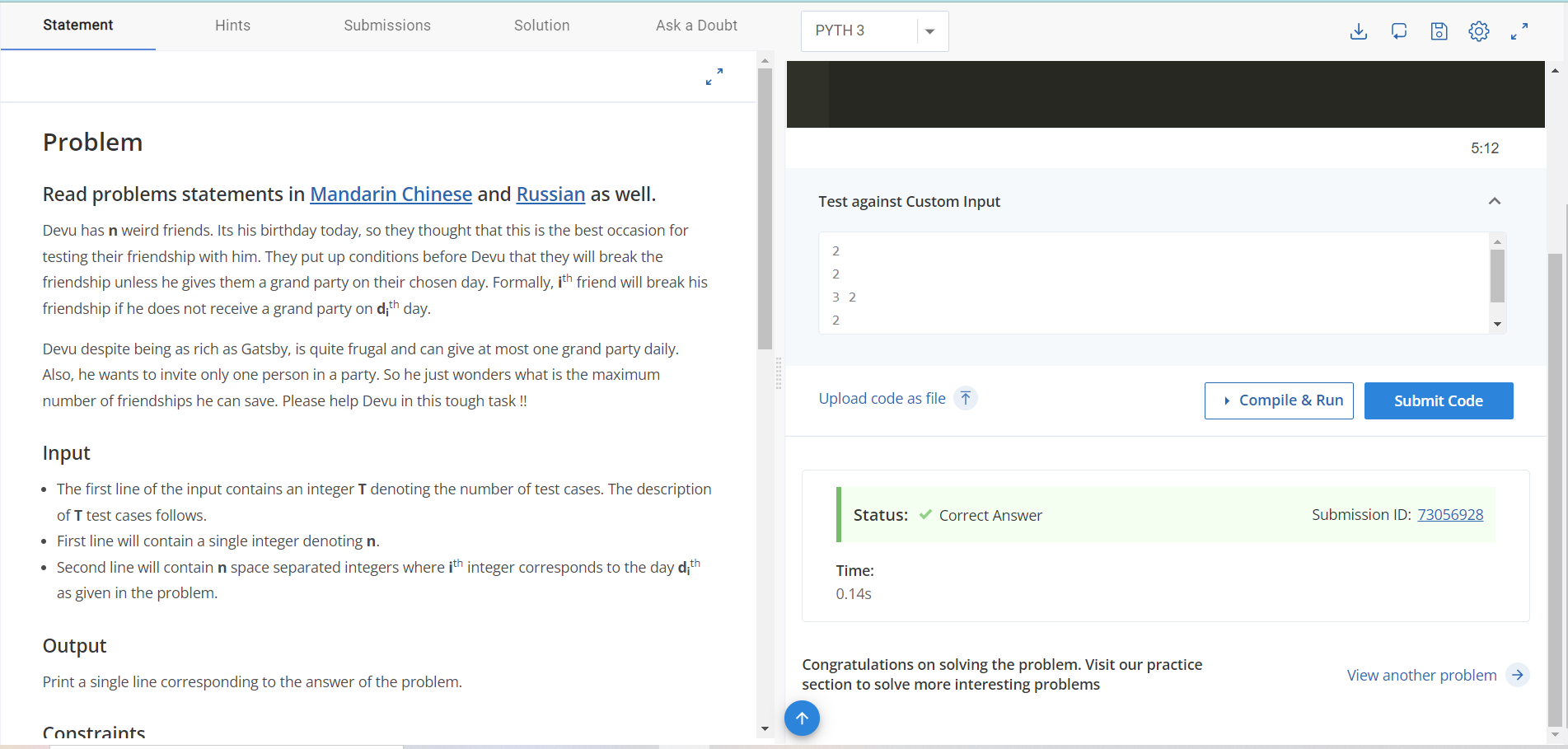
else:

print(len(friends\_set))

test\_cases -= test\_cases

**Output :**





**Time Complexity :** O(n)

**Space Complexity :** O(n)

**Result :** Sucessfully Executed the Program.

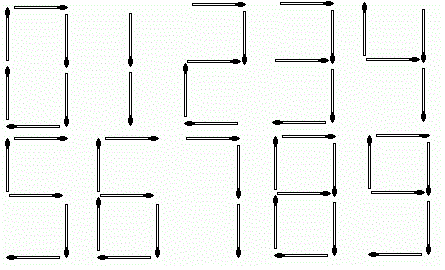
# **WEEK – 4**

## **Task – 1:**

**Aim :** Chef's son Chefu found some matches in the kitchen and he immediately starting playing with them.

The first thing Chefu wanted to do was to calculate the result of his homework — the sum of AA and BB, and write it using matches. Help Chefu and tell him the number of matches needed to write the result.

Digits are formed using matches in the following way:



**Sample Input :**

* The first line of the input contains a single integer T*T* denoting the number of test cases. The description of T*T* test cases follows.
* The first and only line of each test case contains two space-separated integers A*A* and B*B*.

**Sample Output :**

For each test case, print a single line containing one integer — the number of matches needed to write the result (A+B*A*+*B*).

**Program :**

test\_cases = int(input( ))

for i in range(test\_cases):

match = list(map(int, input( ).split( )))[:2]

mat\_res = match[0] + match[1]

number\_length = [6,2,5,5,4,5,6,3,7,6]

res\_str = str(mat\_res)

len\_res = len(res\_str)

min\_count = 0

for i in range(len\_res):

x = int(res\_str[i])

y = number\_length[x]

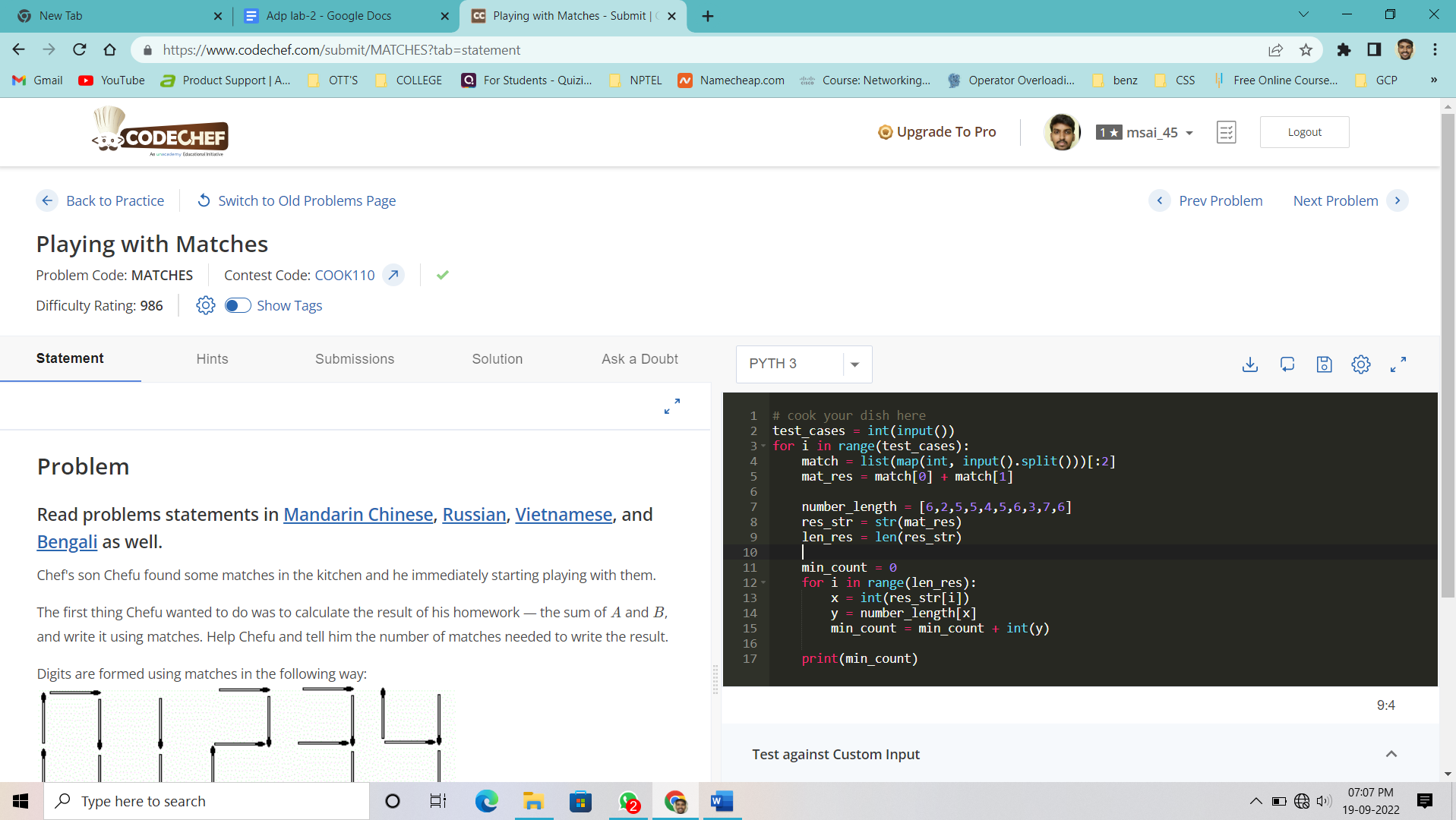
min\_count = min\_count + int(y)

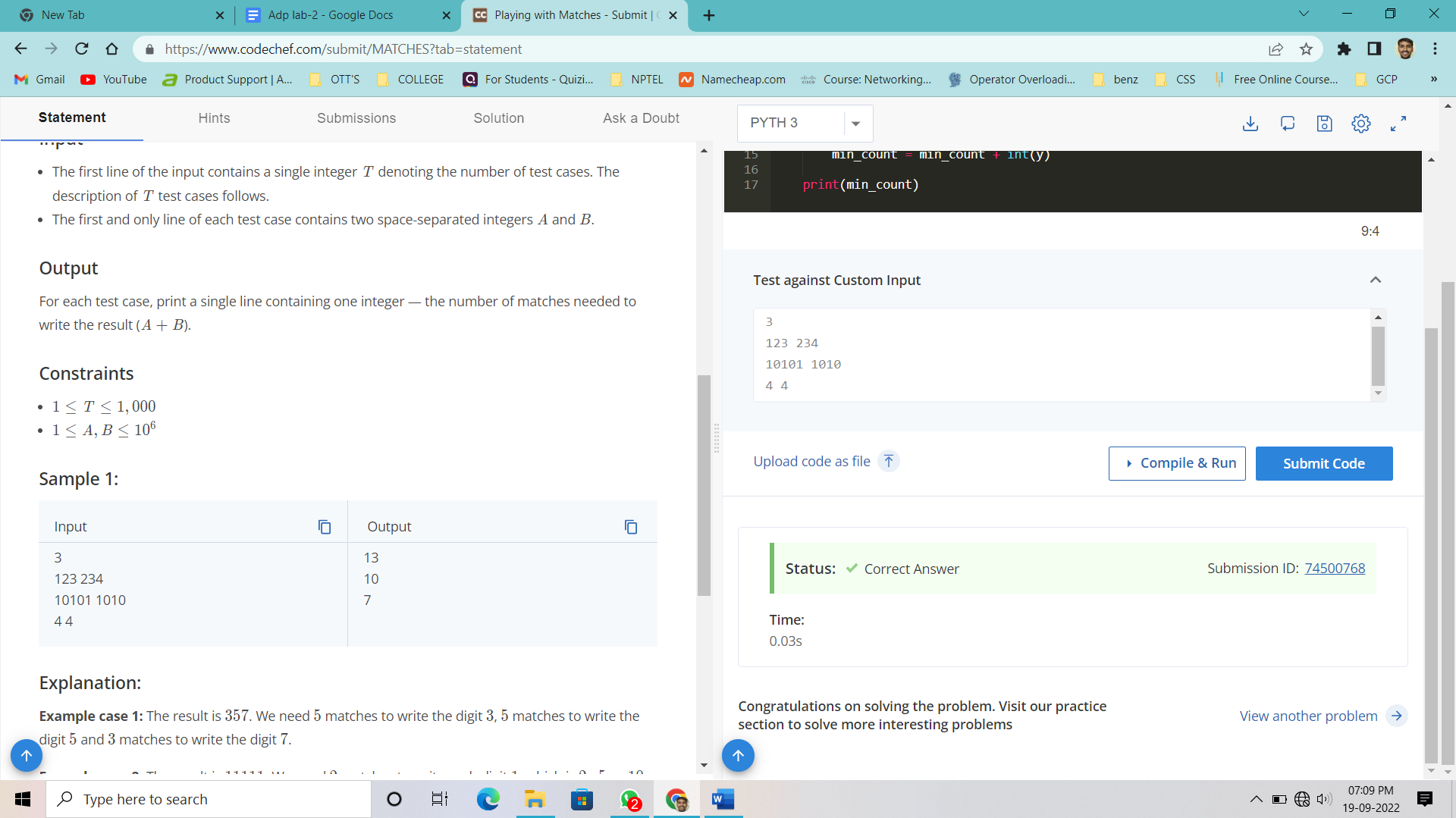
print(min\_count)

**Time Complexity :** O(n2)

**Space Complexity :** O(n)

**Output :**





**Result :** Sucessfully Executed the Program.

## **Task – 2**

**Aim :** You are given a binary string AA of length NN.

You can perform the following type of operation on the string A:

* Choose two different indices i and j (1 <= i, j <= N);
* Change Ai and Aj to Ai ⊕ Aj. Here ⊕ represents the bitwise XOR operation.

Find the minimum number of operations required to convert the given string into a palindrome.

**Sample Input :**

* First line of the input contains T*T*, the number of test cases. Then the test cases follow.
* First line of each test case contains an integer N*N* denoting the size of the string.
* Second line of each test case contains a binary string A*A* of length N*N* containing 00s and 11s only.

**Sample Output :**

For each test case, print the **minimum** number of operations required to make the string a palindrome.

**Program :**

import math

for \_ in range(int(input())):

length = int(input())

string = input()

h = length//2

if length%2 == 0:

half\_1 = string[:h]

half\_2 = string[h:]

half\_2 = half\_2[::-1]

else:

half\_1 = string[:h]

half\_2 = string[h+1:]

half\_2 = half\_2[::-1]

min\_count = 0

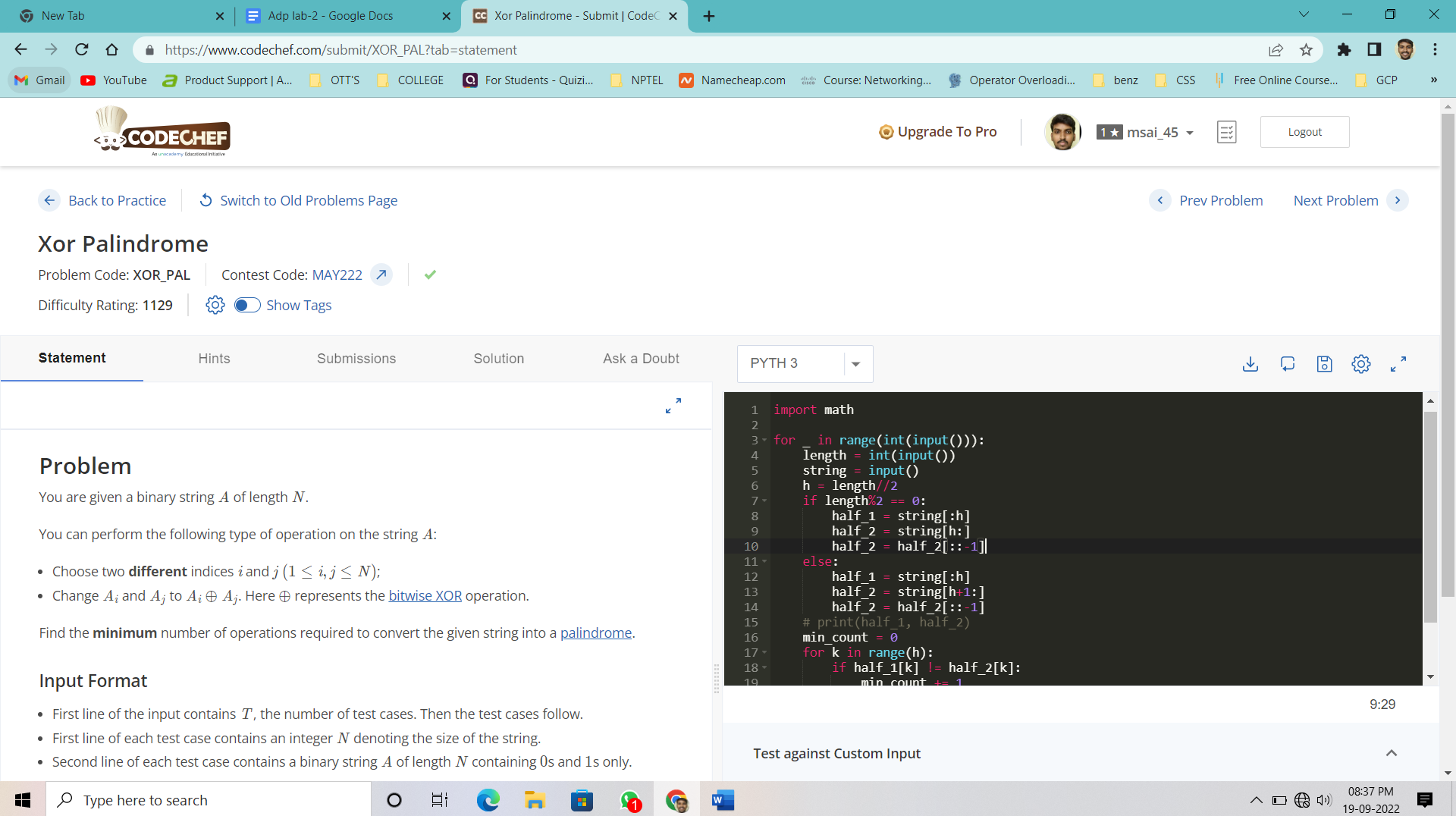
for k in range(h):

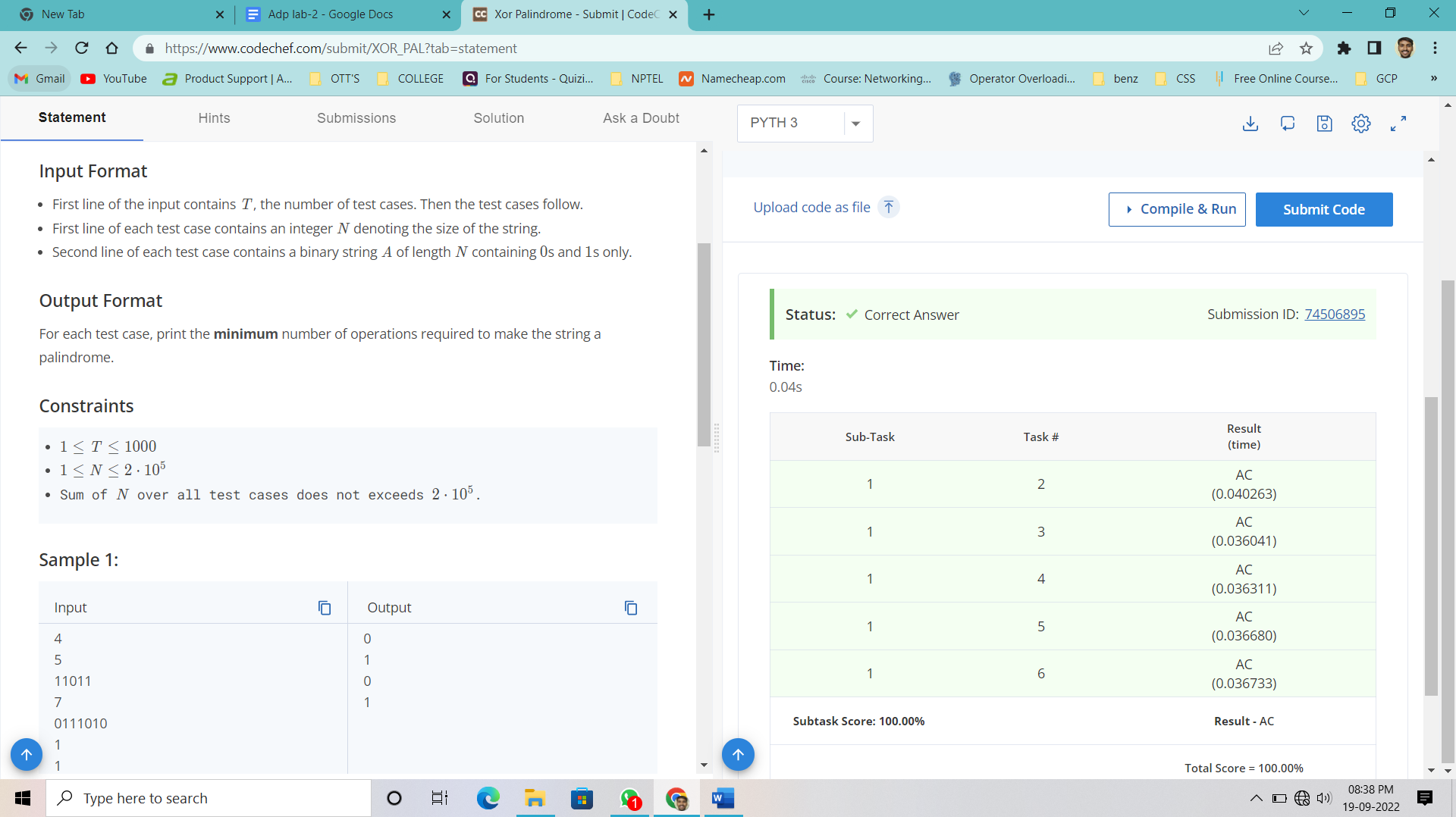
if half\_1[k] != half\_2[k]:

min\_count += 1

print(math.ceil(min\_count/2))

**Output :**





**Time Complexity :** O(n2)

**Space Complexity :** O(n)

**Result :** Sucessfully Executed the program.

## **Task – 3**

**Aim :** Chef likes to play table tennis. He found some statistics of matches which described who won the points in order. A game shall be won by the player first scoring 11 points except in the case when both players have 10 points each, then the game shall be won by the first player subsequently gaining a lead of 2 points. Could you please help the Chef find out who the winner was from the given statistics? (It is guaranteed that statistics represent always a valid, finished match.)

**Sample Input :**

The first line of the input contains an integer **T**, denoting the number of test cases. The description of **T** test cases follows. Each test case consist a binary string **S**, which describes a match. '0' means Chef lose a point, whereas '1' means he won the point.

**Sample Output :**

For each test case, output on a separate line a string describing who won the match. If Chef won then print "WIN" (without quotes), otherwise print "LOSE" (without quotes).

**Program :**

chef = int(input())

for \_ in range(chef):

score = input()

score = list(score)

one = score.count('1')

zero = score.count('0')

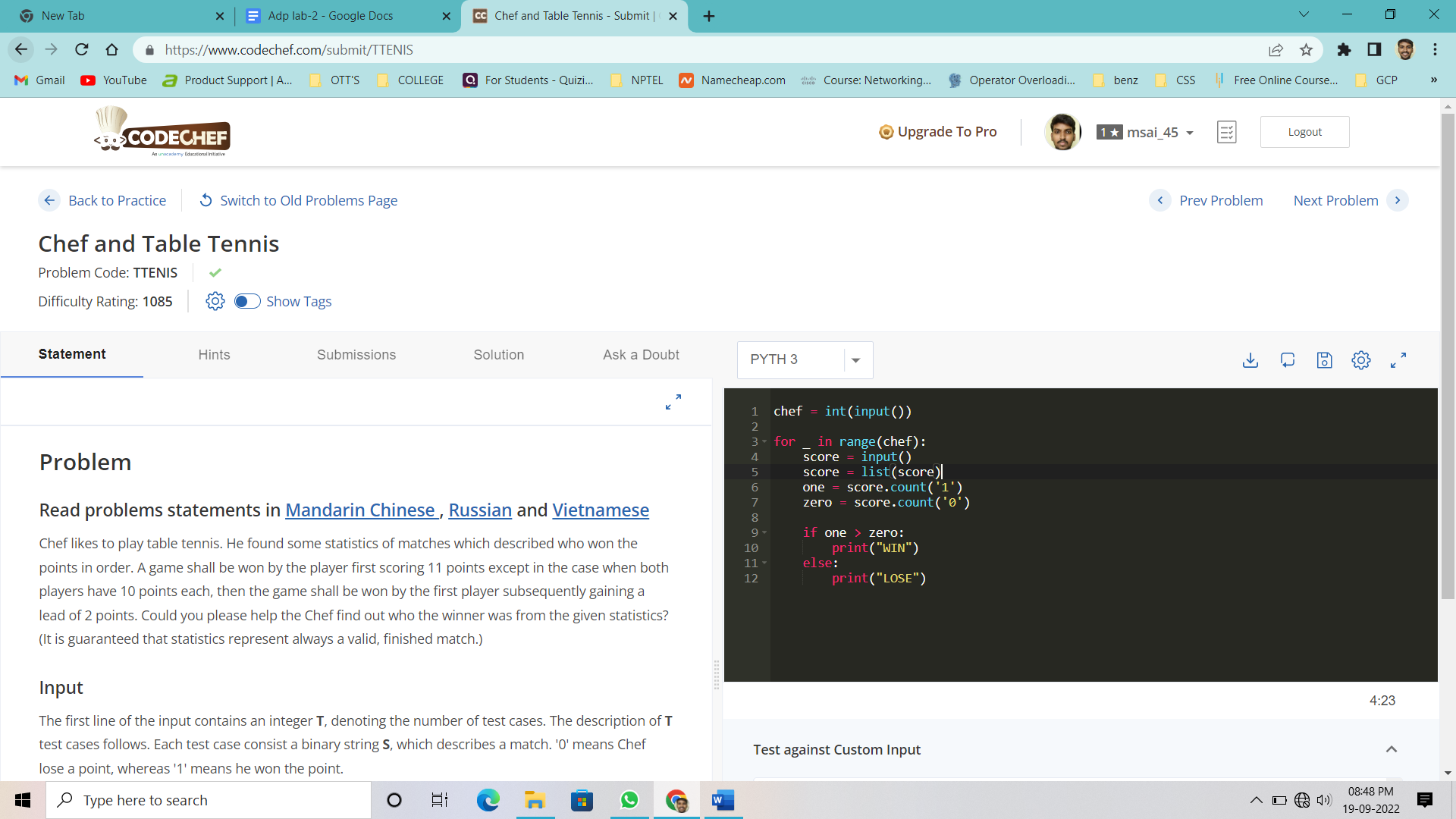
if one > zero:

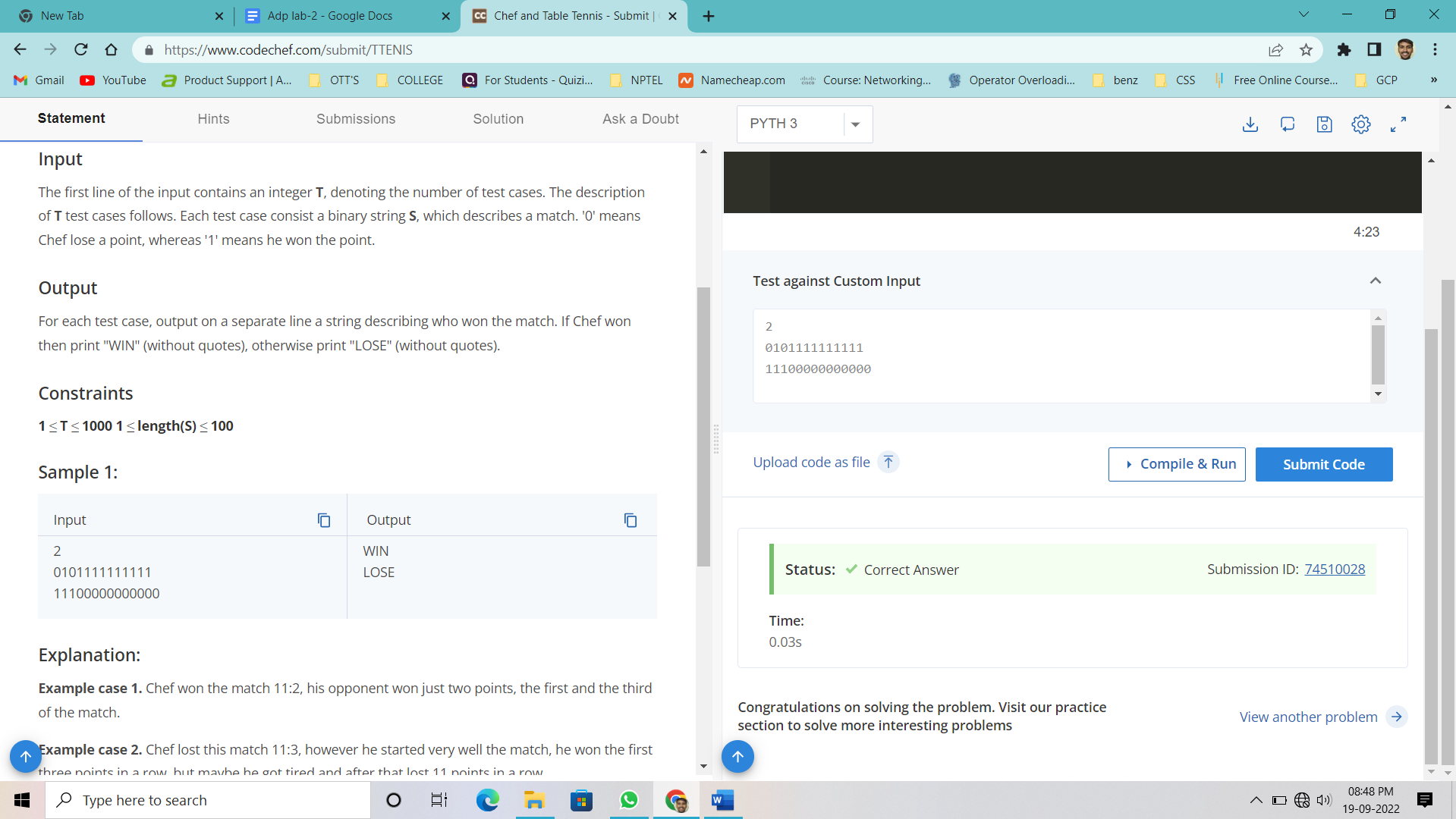
print("WIN")

else:

print("LOSE")

**Output :**





**Time Complexity :** O(n)

**Space Complexity :** O(n)

**Result :** Sucessfully Executed the program.

# **WEEK – 5**

## **Task – 1**

**Aim :** On a sunny day, Akbar and Birbal were taking a leisurely walk in palace gardens. Suddenly, Akbar noticed a bunch of sticks on the ground and decided to test Birbal's wits.

There are N*N* stick holders with negligible size (numbered 11 through N*N*) in a row on the ground. Akbar places all the sticks in them vertically; for each valid i*i*, the initial height of the stick in the i*i*-th holder is A\_i*Ai*​. Birbal has a stick cutter and his task is to completely cut all these sticks, i.e. reduce the heights of all sticks to 00. He may perform zero or more operations; in each operation, he should do the following:

* Choose an integer H*H* and fix the cutter at the height H*H* above the ground.
* The cutter moves from the 11-st to the N*N*-th stick holder. Whenever it encounters a stick whose current height is greater than H*H*, it cuts this stick down to height H*H* (i.e. for a stick with height h \gt H*h*>*H*, it removes its upper part with length h-H*h*−*H*).
* All the upper parts of sticks that are cut in one operation must have equal lengths. Otherwise, the operation may not be performed.

For example, if the heights of sticks are initially [5, 3, 5][5,3,5], then some valid values for H*H* in the first operation are 33 and 44 ― the cutter cuts the upper parts of two sticks and their lengths are [2, 2][2,2] and [1, 1][1,1] respectively. H = 2*H*=2 is an invalid choice because it would cut the upper parts of all three sticks with lengths [3, 1, 3][3,1,3], which are not all equal.

Akbar wants Birbal to completely cut all sticks in the minimum possible number of operations. If you want to be friends with Birbal, help him solve the problem.

**Sample Input :**

* The first line of the input contains a single integer T*T* denoting the number of test cases. The description of T*T* test cases follows.
* The first line of each test case contains a single integer N*N*.
* The second line contains N*N* space-separated integers A\_1, A\_2, \ldots, A\_N*A*1​,*A*2​,…,*AN*​.

**Sample Output :**

For each test case, print a single line containing one integer ― the minimum number of operations needed to completely cut all the sticks.

**Program :**

test = int(input())

for i in range(test):

number = int(input())

array = list(map(int, input().split()))

if 0 in array:

print(len(set(array)) - 1)

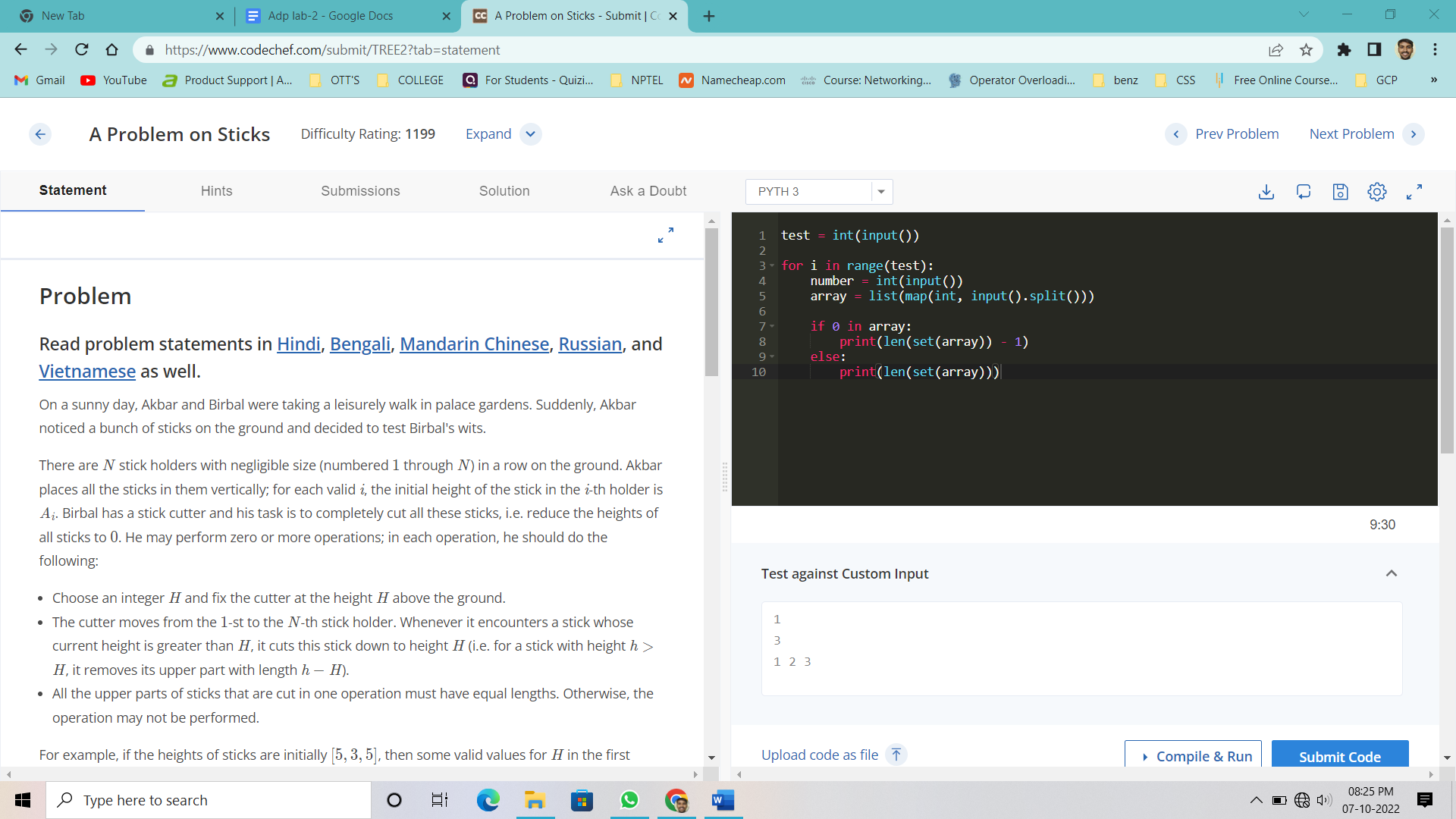
else:

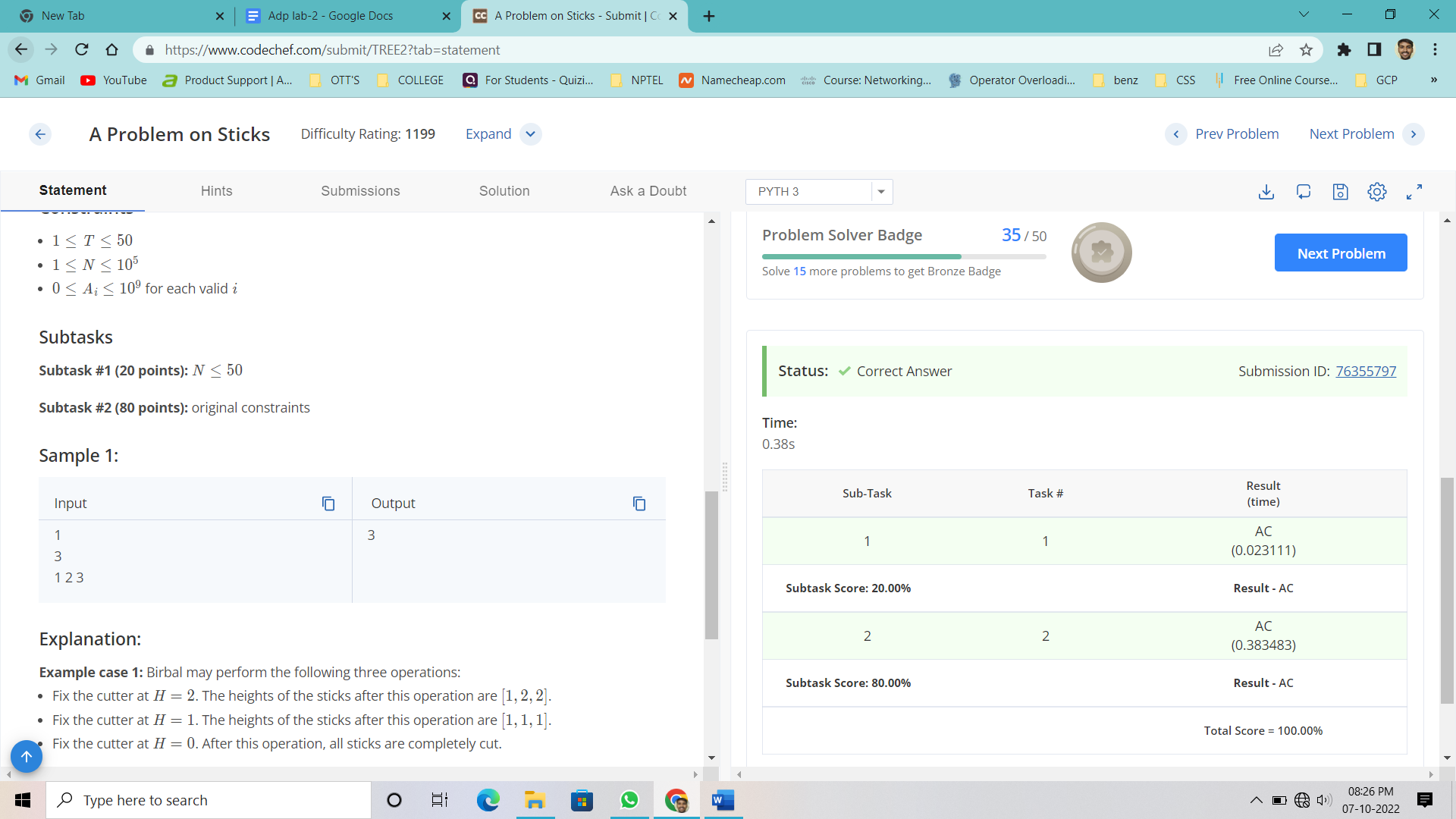
print(len(set(array)))

**Time Complexity :** O(n)

**Space Complexity :** O(n)

**Output :**





**Result :** Sucessully Executed the program.

## **Task – 2**

**Aim :** Chef is given a binary string A*A* of length N*N*. He can perform the following operation on A*A* any number of times:

* Choose L*L* and R*R* (1 \leq L \leq R \leq N)(1≤*L*≤*R*≤*N*), such that, in the [substring](https://en.wikipedia.org/wiki/Substring) A[L,R]*A*[*L*,*R*], the number of 11s is equal to the number of 00s and reverse the substring A[L,R]*A*[*L*,*R*].

Find the lexicographically smallest string that Chef can obtain after performing the above operation any (possibly zero) number of times on A*A*.

String X*X* is lexicographically smaller than string Y*Y*, if either of the following satisfies:

* X*X* is a prefix of Y*Y* and X \neq Y*X*=*Y*.
* There exists an index i*i* such that X\_i \lt Y\_i*Xi*​<*Yi*​ and X\_j = Y\_j, \forall j*Xj*​=*Yj*​,∀*j* such that 1 \leq j \lt i1≤*j*<*i*.

**Sample Input :**

* First line will contain T*T*, the number of test cases. Then the test cases follow. Each test case contains two lines.
* The first line contains the integer N*N*, the length of the binary string.
* The second line contains the binary string A*A*.

**Sample Output :**

For each test case, print the lexicographically smallest binary string that can be obtained after performing the operation any (possibly zero) number of times.

**Program :**

test = int(input())

for \_ in range(test):

length = int(input())

string\_array = input()

bin\_array = []

new\_binary = ''

for i in range(length):

bin\_array.append(int(string\_array[i]))

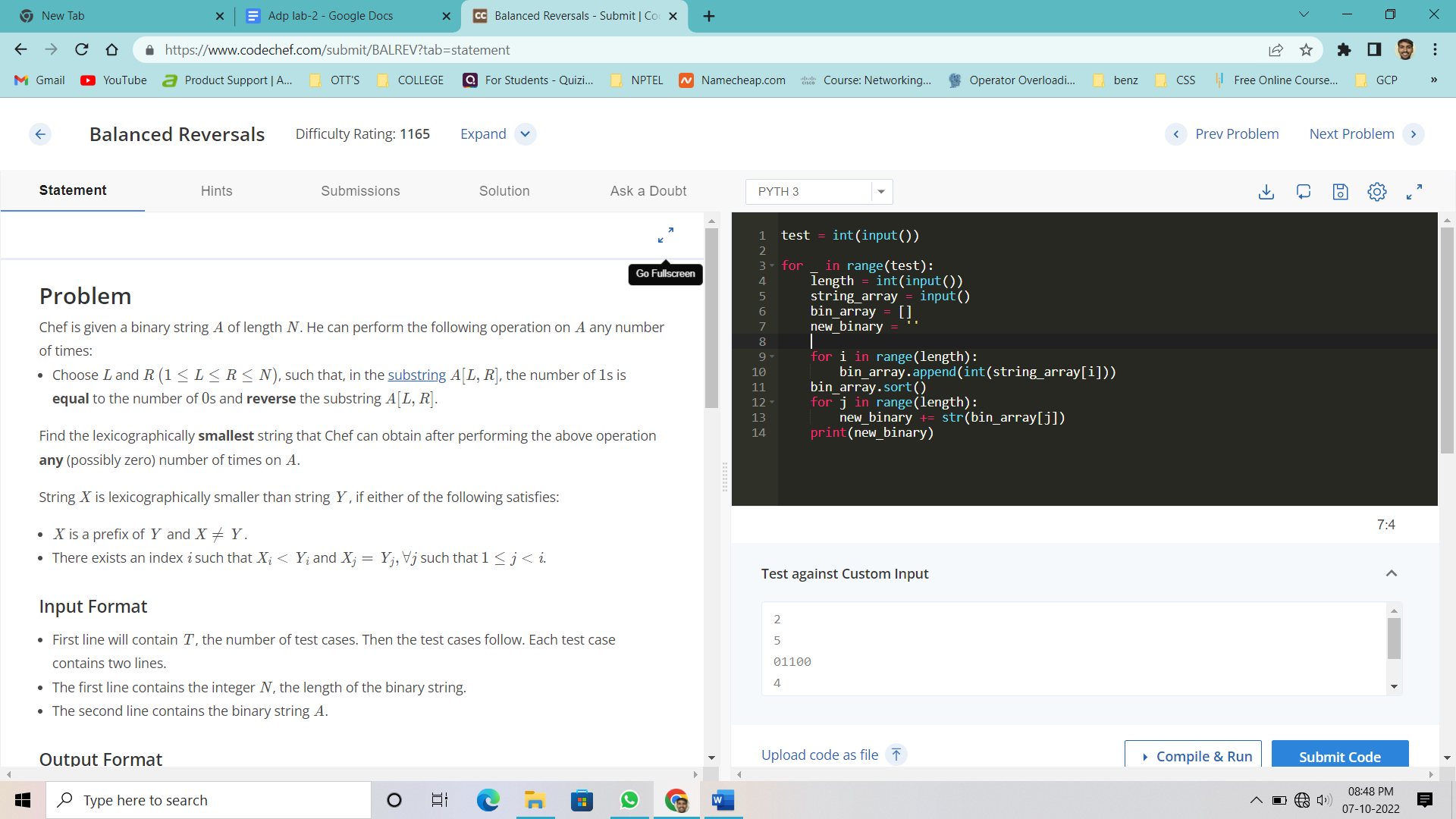
bin\_array.sort()

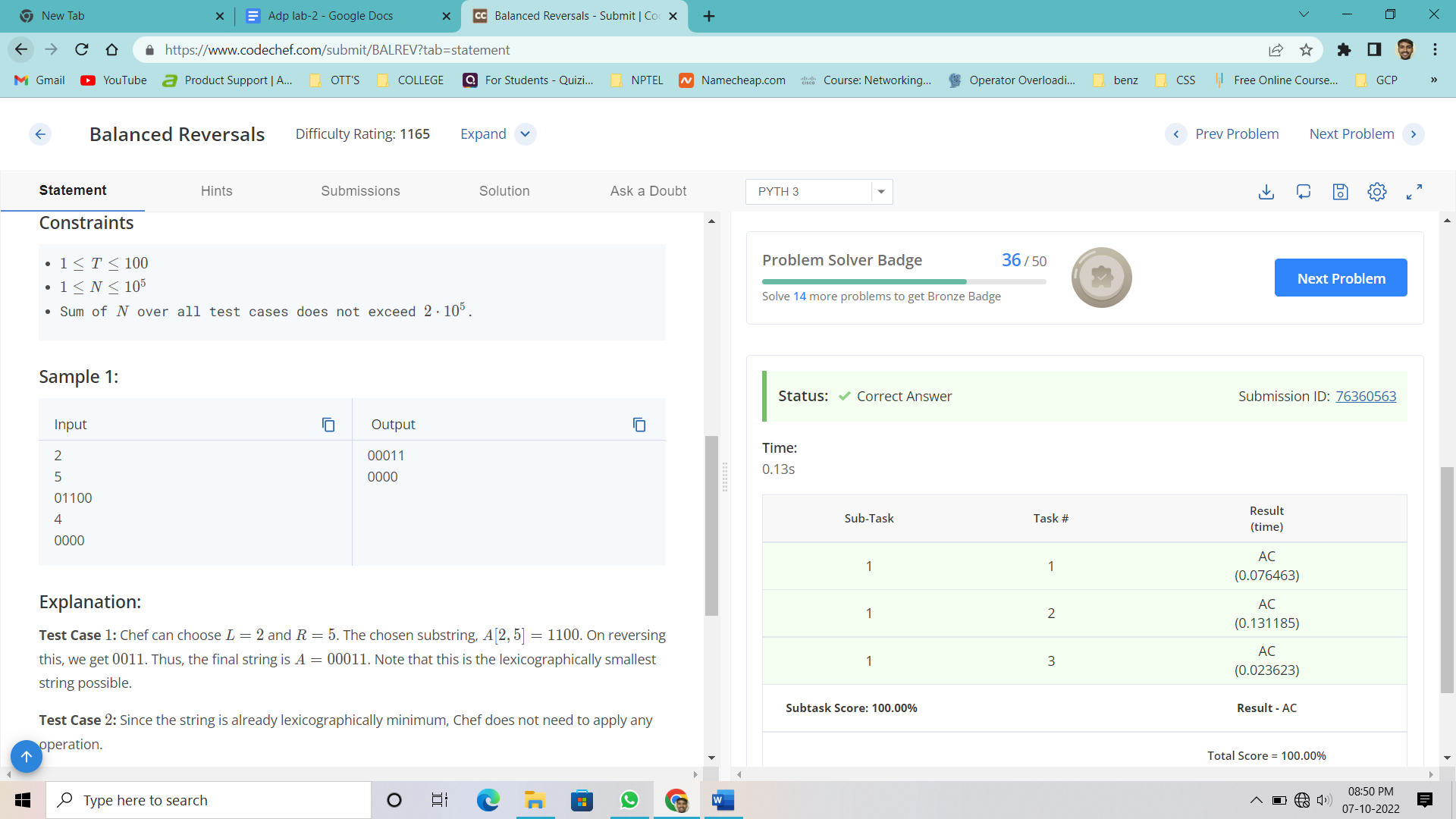
for j in range(length):

new\_binary += str(bin\_array[j])

print(new\_binary)

**Output :**





**Time Complexity :** O(n3)

**Space Complexity :** O(n)

**Result :** Sucessfully Executed the Program.

## **Task – 3**

**Aim :** A Little Elephant from the Zoo of Lviv likes *lucky strings*, i.e., the strings that consist only of the lucky digits 4 and 7.

The Little Elephant has K favorite lucky strings A1, A2, ..., AK. He thinks that the lucky string S is *good* if either |S| ≥ 47 or for some j from 1 to K we have that Aj is a substring of S.

The Little Elephant has found N lucky strings B1, B2, ..., BN under the pillow. Now he wants to know which of them are good. Help him and find for each i from 1 to N whether the string Bi is good or not.

Let S be some lucky string. Then

* |S| denotes the length of the string S;
* S[i] (1 ≤ i ≤ |S|) denotes the ith character of S (the numeration of characters starts from 1);
* The string T of the length M is called a *substring* of S if for some k from 0 to |S| - M we have  
  T[1] = S[k + 1], T[2] = S[k + 2], ..., T[M] = S[k + M].

**Sample Input :**

The first line of the input file contains two integers K and N, the number of favorite lucky strings of the Little Elephant and the number of strings he has found under the pillow. Each of the following K lines contains one favorite lucky string. Namely, jth line among these K lines contains the string Aj. Each of the following N lines contains one lucky string that was found under the pillow. Namely, ith line among these N lines contains the string Bi. The input file does not contain any whitespaces.

**Sample Output :**

For each of the N strings that were found under the pillow print Good if it is good, and Bad otherwise.

**Program :**

# cook your dish here

k, n = map(int,input().split())

a = [input() for i in range(k)]

b = [input() for i in range(n)]

for each in b:

count = 0

for j in a:

if j in each:

count = 1

if len(each) >= 47:

count = 1

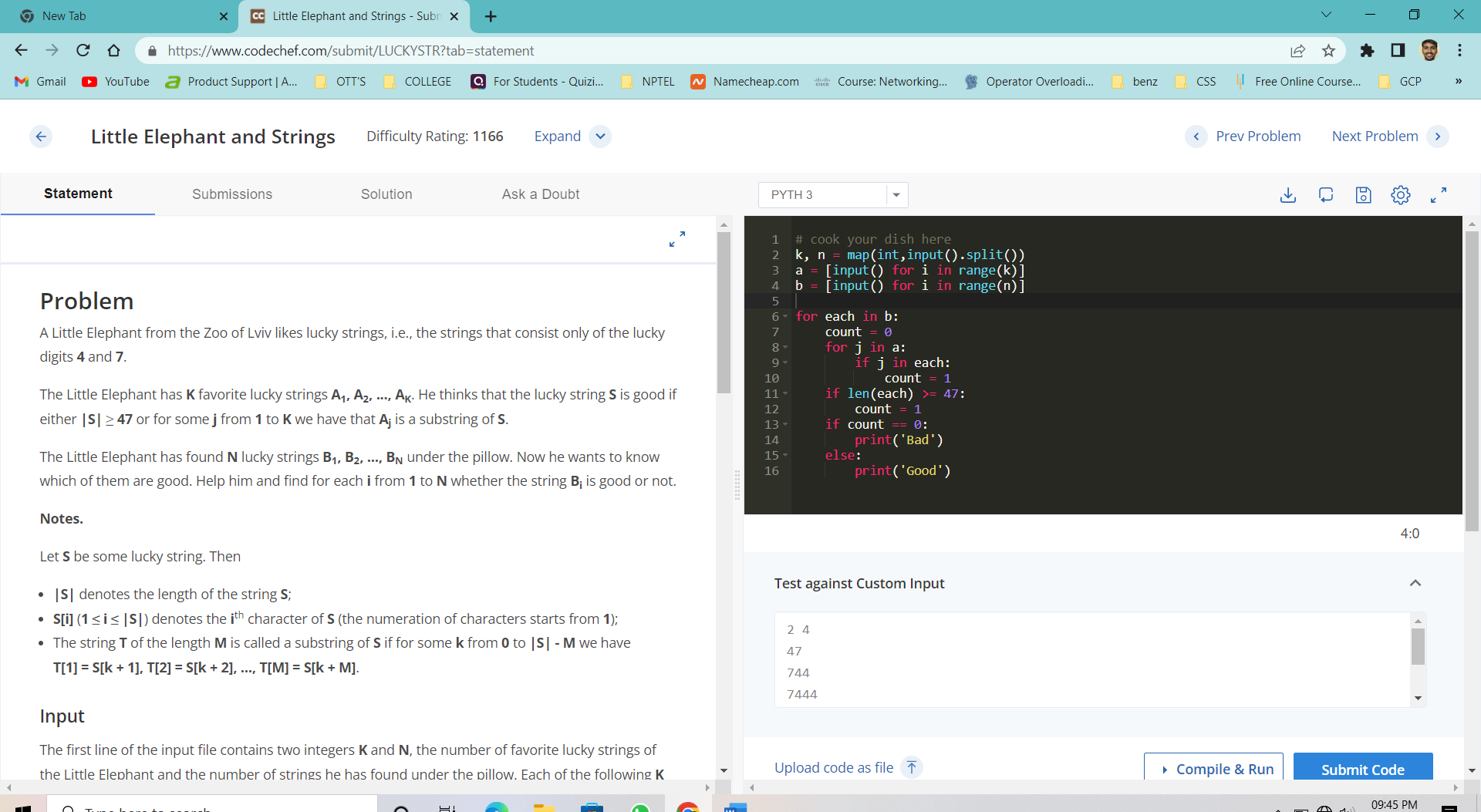
if count == 0:

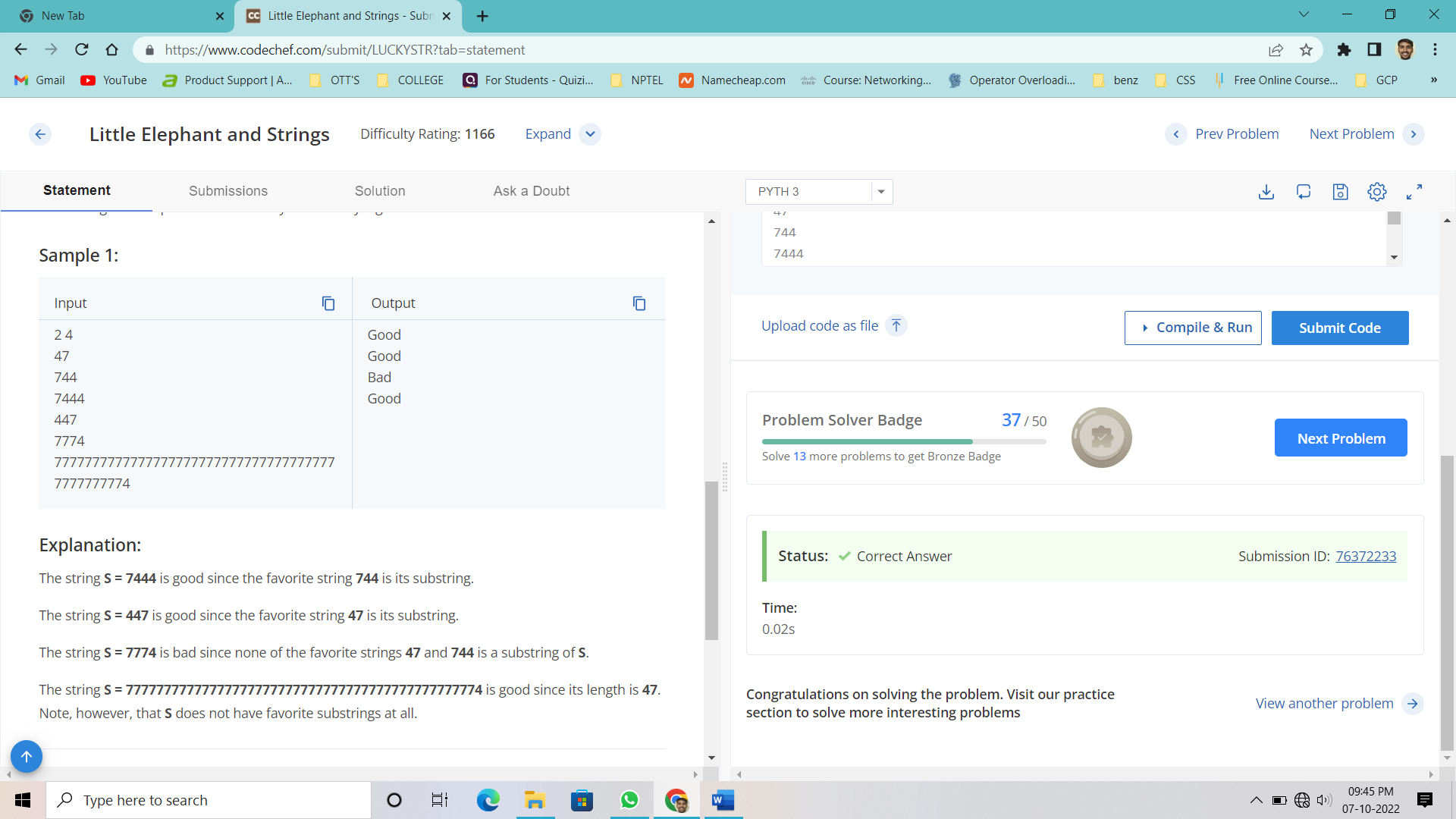
print('Bad')

else:

print('Good')

**Output :**





**Time Complexity :** O(n2)

**Space Complexity :** O(n)

**Result :** Sucessfully Executed the Program.

# **WEEK – 6**

**Task – 1 :**

**Aim :** You are given a binary string AA of length NN.

You can perform the following type of operation on the string A:

* Choose two different indices i and j (1 <= i, j <= N);
* Change Ai and Aj to Ai ⊕ Aj. Here ⊕ represents the bitwise XOR operation.

Find the minimum number of operations required to convert the given string into a palindrome.

**Sample Input :**

* First line of the input contains T*T*, the number of test cases. Then the test cases follow.
* First line of each test case contains an integer N*N* denoting the size of the string.
* Second line of each test case contains a binary string A*A* of length N*N* containing 00s and 11s only.

**Sample Output :**

For each test case, print the **minimum** number of operations required to make the string a palindrome.

**Program :**

import math

for \_ in range(int(input())):

length = int(input())

string = input()

h = length//2

if length%2 == 0:

half\_1 = string[:h]

half\_2 = string[h:]

half\_2 = half\_2[::-1]

else:

half\_1 = string[:h]

half\_2 = string[h+1:]

half\_2 = half\_2[::-1]

min\_count = 0

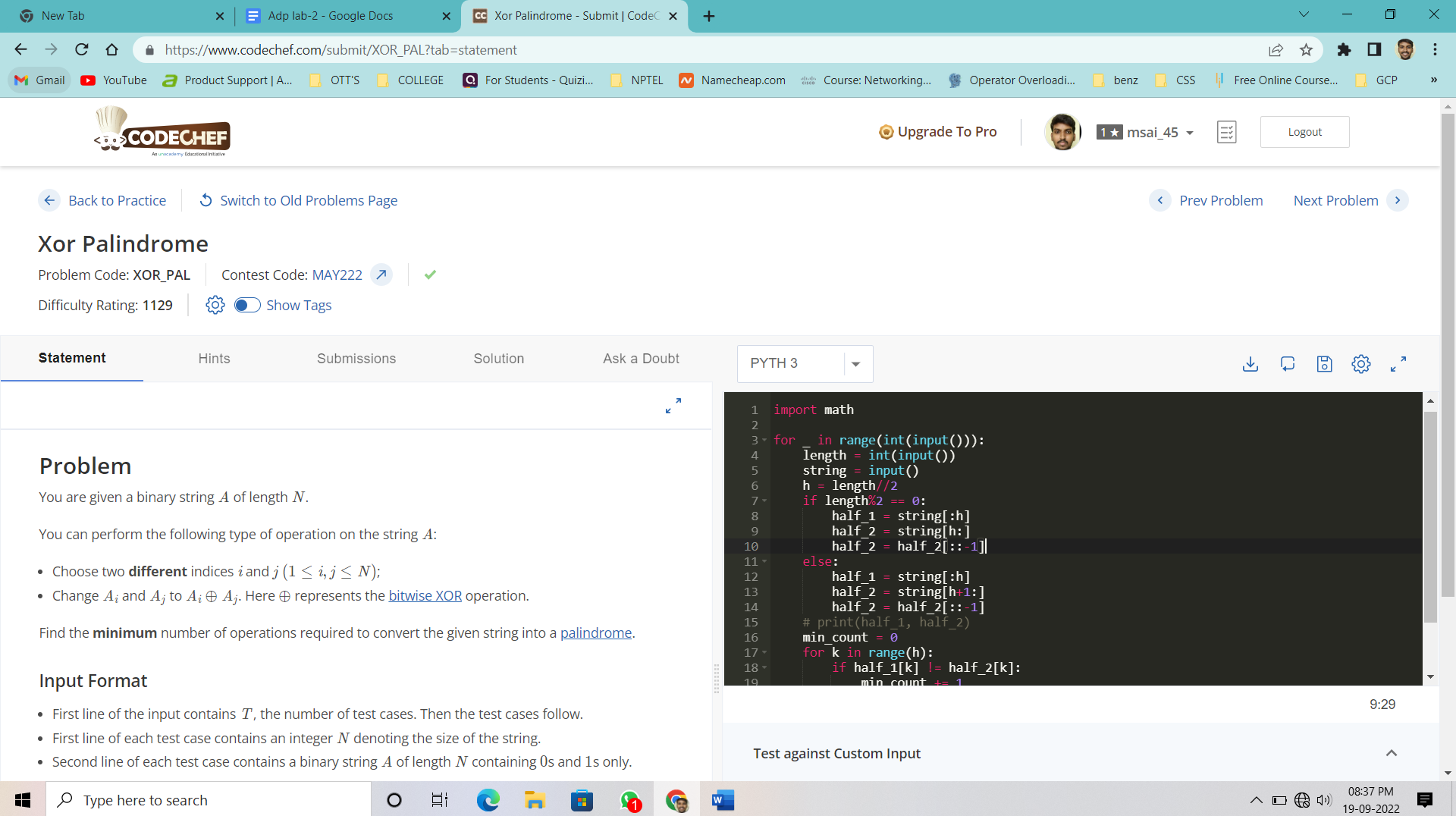
for k in range(h):

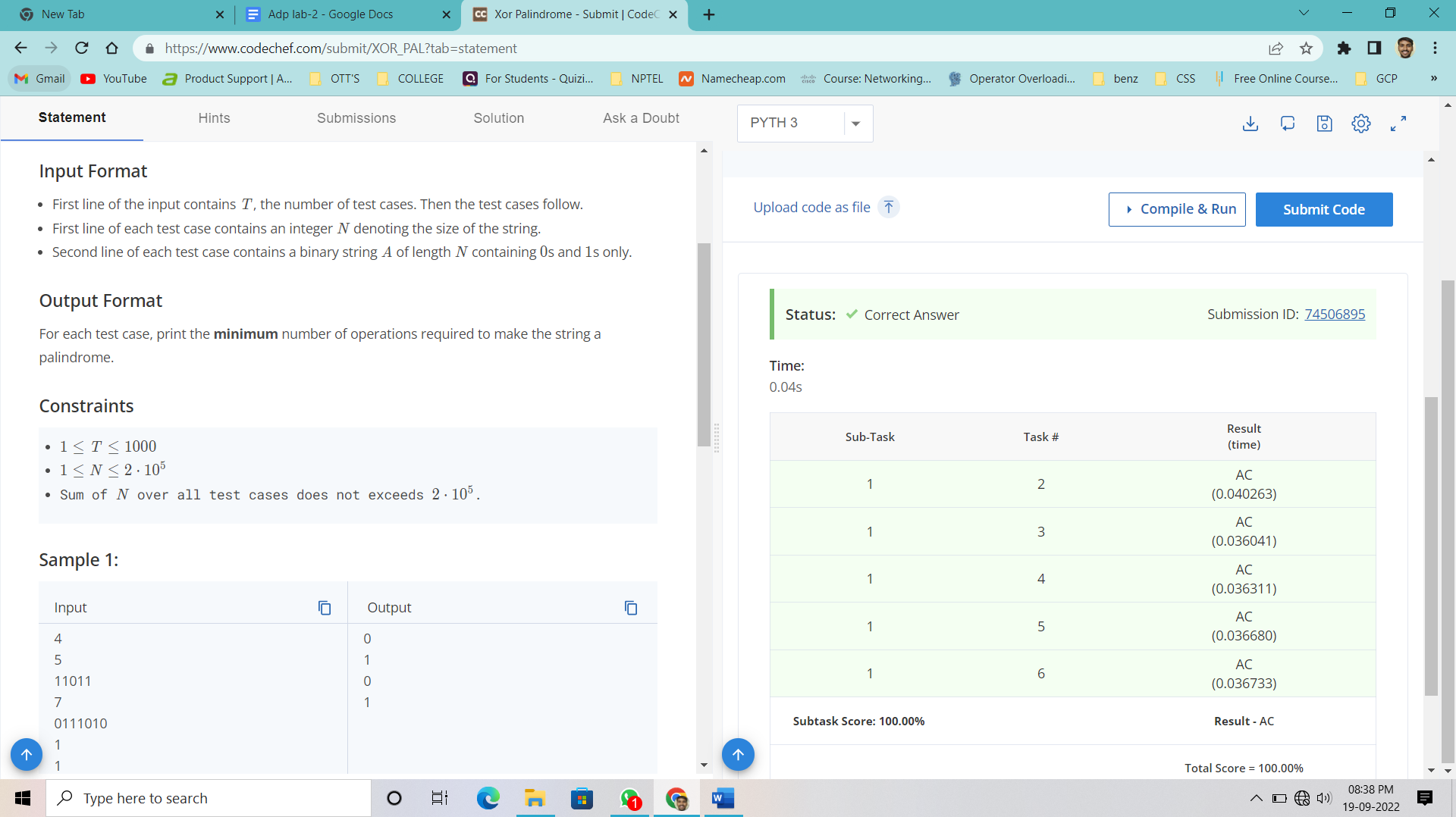
if half\_1[k] != half\_2[k]:

min\_count += 1

print(math.ceil(min\_count/2))

**Output :**





**Time Complexity :** O(n2)

**Space Complexity :** O(n)

**Result :** Sucessfully Executed the program.

**Task – 2:**

**Aim :**

Given an integer N*N*, determine the number of pairs (A, B)(*A*,*B*) such that:

* 1 <= A, B <= N1 ≤ *A*,*B* ≤ *N*;
* *A*+*B* is odd.

**Input Format**

* The first line of input will contain a single integer T, denoting the number of test cases.
* Each test case consists of a single integer N*N*.

**Output Format**

For each test case, output the number of required pairs.

**Program :**

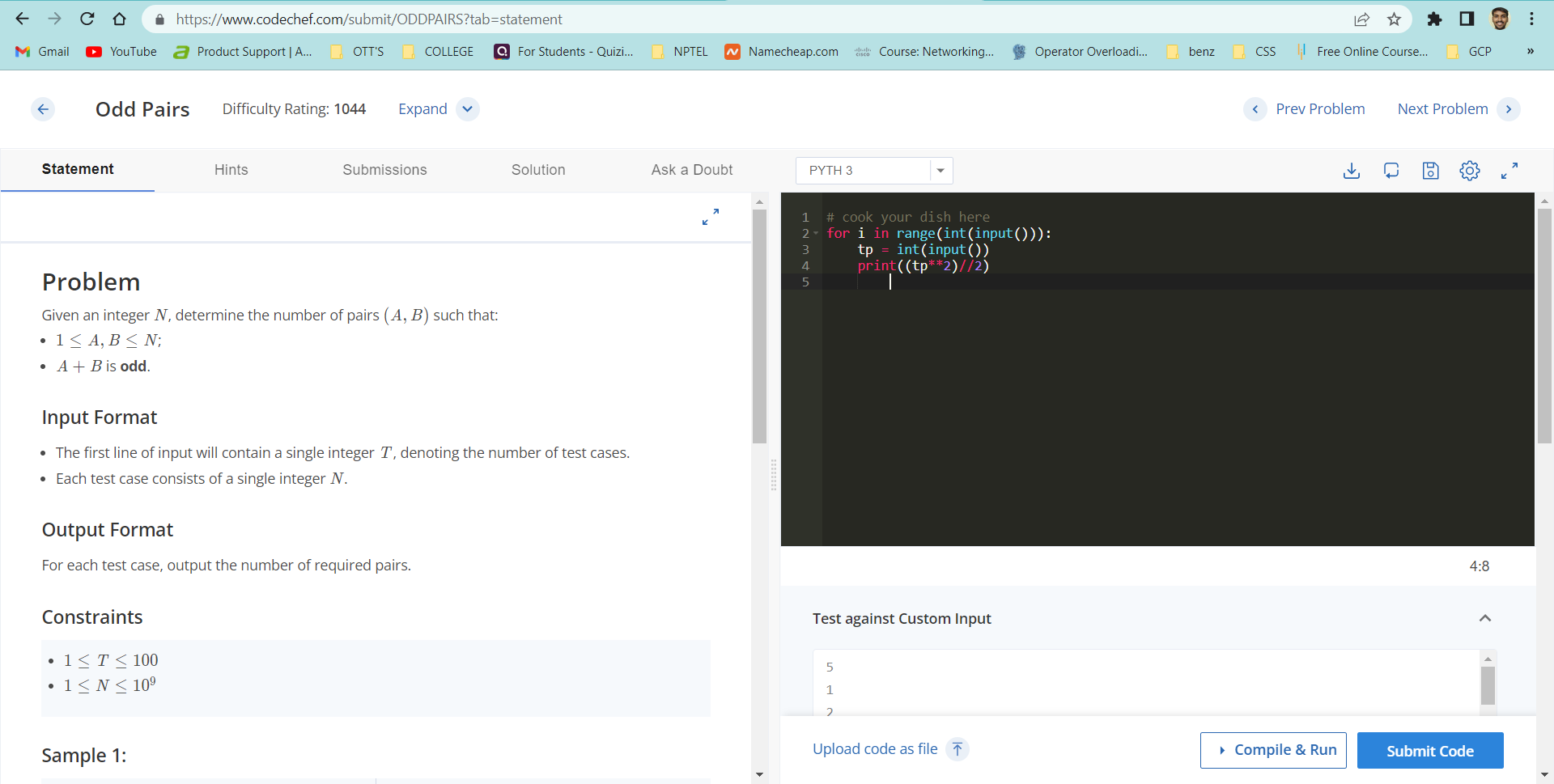
# cook your dish here

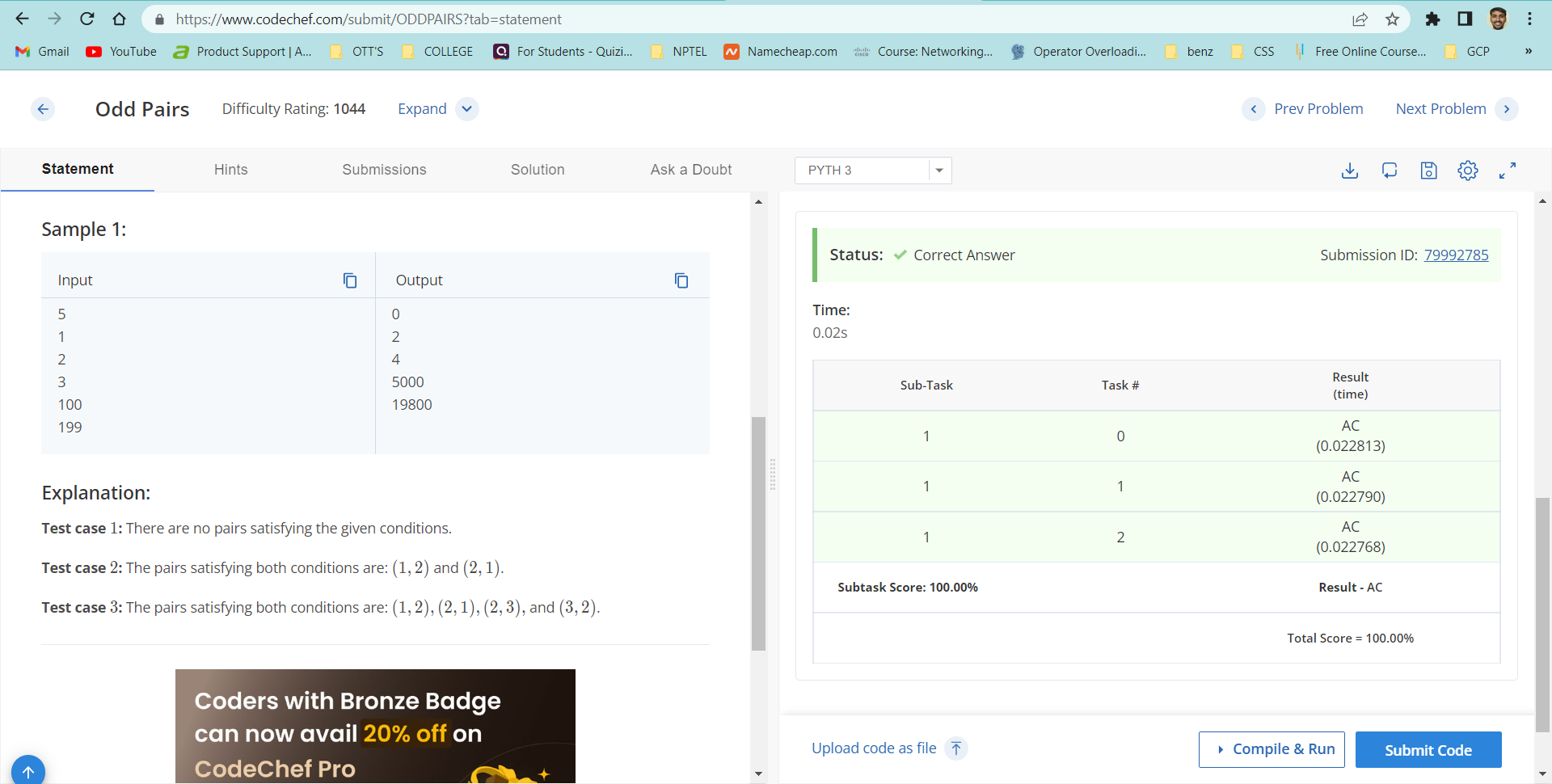
for i in range(int(input())):

tp = int(input())

print((tp\*\*2)//2)

**Output :**





**Time Complexity :** O(n)

**Space Complexity :** O(n)

**Result :** Sucessfully executed the program.

**Task – 3:**

**Aim :** The chef has a recipe he wishes to use for his guests, but the recipe will make far more food than he can serve to the guests. The chef therefore would like to make a reduced version of the recipe which has the same ratios of ingredients, but makes less food. The chef, however, does not like fractions. The original recipe contains only whole numbers of ingredients, and the chef wants the reduced recipe to only contain whole numbers of ingredients as well. Help the chef determine how much of each ingredient to use in order to make as little food as possible.

**Input**

Input will begin with an integer T, the number of test cases. Each test case consists of a single line. The line begins with a positive integer N, the number of ingredients. N integers follow, each indicating the quantity of a particular ingredient that is used.

**Output**

For each test case, output exactly N space-separated integers on a line, giving the quantity of each ingredient that the chef should use in order to make as little food as possible.

**Program :**

import math as mat

for i in range(int(input())):

narr = list(map(int,input().split()))

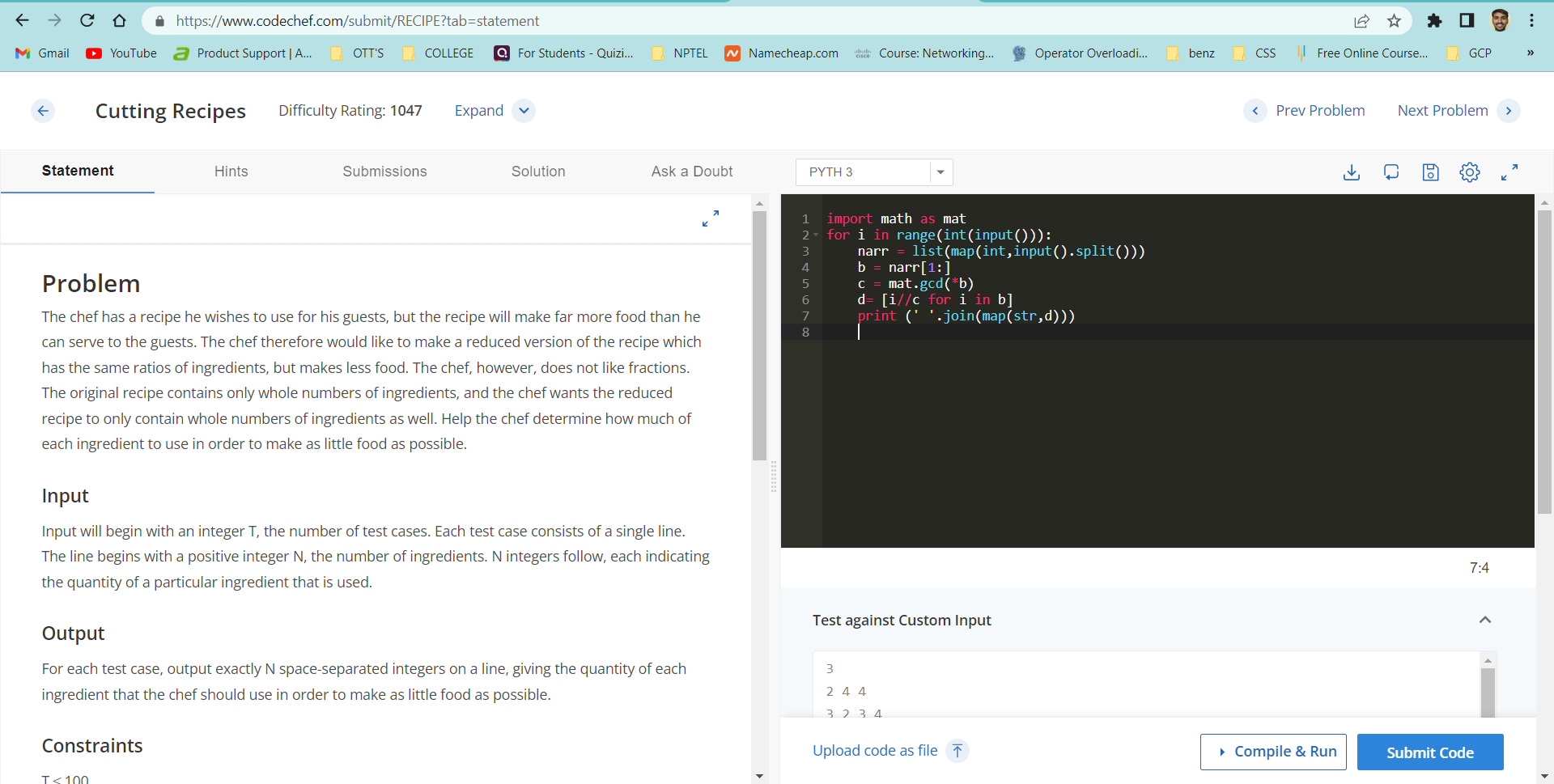
b = narr[1:]

c = mat.gcd(\*b)

d= [i//c for i in b]

print (' '.join(map(str,d)))

**Output :**





**Time Complexity :** O(n2)

**Space Complexity :** O(n2)

**Result :** Sucessfully Executed the program.