**Python 1 Week Preparation :**

**Day 1:**

**Qn 1: Plus Minus**

Given an array of integers, calculate the ratios of its elements that are *positive*, *negative*, and *zero*. Print the decimal value of each fraction on a new line with  places after the decimal.

**Note:** This challenge introduces precision problems. The test cases are scaled to six decimal places, though answers with absolute error of up to  are acceptable.

**Example**

There are  elements, two positive, two negative and one zero. Their ratios are ,  and . Results are printed as:

0.400000

0.400000

0.200000

**Function Description**

Complete the *plusMinus* function in the editor below.

plusMinus has the following parameter(s):

* *int arr[n]*: an array of integers

**Print**  
Print the ratios of positive, negative and zero values in the array. Each value should be printed on a separate line with  digits after the decimal. The function should not return a value.

**Input Format**

The first line contains an integer, , the size of the array.  
The second line contains  space-separated integers that describe .

**Constraints**

**Output Format**

**Print** the following  lines, each to  decimals:

1. proportion of positive values
2. proportion of negative values
3. proportion of zeros

**Sample Input**

STDIN Function

----- --------

6 arr[] size n = 6

-4 3 -9 0 4 1 arr = [-4, 3, -9, 0, 4, 1]

**Sample Output**

0.500000

0.333333

0.166667

**Explanation**

There are  positive numbers,  negative numbers, and  zero in the array.  
The proportions of occurrence are positive: , negative:  and zeros:

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'plusMinus' function below.

#

# The function accepts INTEGER\_ARRAY arr as parameter.

#

def plusMinus(arr):

    n = len(arr)

    # Initialize counters for positive, negative, and zero values

    positive\_count = 0

    negative\_count = 0

    zero\_count = 0

    # Loop through the array and classify each element

    for num in arr:

        if num > 0:

            positive\_count += 1

        elif num < 0:

            negative\_count += 1

        else:

            zero\_count += 1

    # Calculate the ratios

    positive\_ratio = positive\_count / n

    negative\_ratio = negative\_count / n

    zero\_ratio = zero\_count / n

    # Print the ratios with 6 decimal places

    print(f"{positive\_ratio:.6f}")

    print(f"{negative\_ratio:.6f}")

    print(f"{zero\_ratio:.6f}")

if \_\_name\_\_ == '\_\_main\_\_':

    n = int(input().strip())

    arr = list(map(int, input().rstrip().split()))

    plusMinus(arr)

**Qn 2: Min-Max Sum**

Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers. Then print the respective minimum and maximum values as a single line of two space-separated long integers.

**Example**

The minimum sum is  and the maximum sum is . The function prints

16 24

**Function Description**

Complete the *miniMaxSum* function in the editor below.

miniMaxSum has the following parameter(s):

* *arr*: an array of  integers

**Print**

Print two space-separated integers on one line: the minimum sum and the maximum sum of  of  elements.

**Input Format**

A single line of five space-separated integers.

**Constraints**

**Output Format**

Print two space-separated long integers denoting the respective minimum and maximum values that can be calculated by summing exactly *four* of the five integers. (The output can be greater than a 32 bit integer.)

**Sample Input**

1 2 3 4 5

**Sample Output**

10 14

**Explanation**

The numbers are , , , , and . Calculate the following sums using four of the five integers:

1. Sum everything except , the sum is .
2. Sum everything except , the sum is .
3. Sum everything except , the sum is .
4. Sum everything except , the sum is .
5. Sum everything except , the sum is .

**Hints:** Beware of integer overflow! Use 64-bit Integer.

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'miniMaxSum' function below.

#

# The function accepts INTEGER\_ARRAY arr as parameter.

#

def miniMaxSum(arr):

    # Sort the array

    arr.sort()

    # Calculate the minimum and maximum sums

    min\_sum = sum(arr[:4])  # Sum of the smallest four numbers

    max\_sum = sum(arr[1:])  # Sum of the largest four numbers

    # Print the results

    print(min\_sum, max\_sum)

if \_\_name\_\_ == '\_\_main\_\_':

    arr = list(map(int, input().rstrip().split()))

    miniMaxSum(arr)

**Qn 3: Time Conversion**

Given a time in [-hour AM/PM format](https://en.wikipedia.org/wiki/12-hour_clock), convert it to military (24-hour) time.

Note: - 12:00:00AM on a 12-hour clock is 00:00:00 on a 24-hour clock.  
- 12:00:00PM on a 12-hour clock is 12:00:00 on a 24-hour clock.

**Example**

* Return '12:01:00'.
* Return '00:01:00'.

**Function Description**

Complete the *timeConversion* function in the editor below. It should return a new string representing the input time in 24 hour format.

timeConversion has the following parameter(s):

* *string s*: a time in  hour format

**Returns**

* *string*: the time in  hour format

**Input Format**

A single string  that represents a time in -hour clock format (i.e.:  or ).

**Constraints**

* All input times are valid

**Sample Input**

07:05:45PM

**Sample Output**

19:05:45

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'timeConversion' function below.

#

# The function is expected to return a STRING.

# The function accepts STRING s as parameter.

#

def timeConversion(s):

    # Extract the hour, minute, and second

    hour = int(s[:2])

    minute\_second = s[2:8]

    period = s[-2:]  # AM or PM

    # Convert hour based on AM/PM

    if period == "AM":

        if hour == 12:

            hour = 0  # Midnight case

    elif period == "PM":

        if hour != 12:

            hour += 12  # Convert PM hour to 24-hour format

    # Format the hour to ensure two digits and return the result

    return f"{hour:02}{minute\_second}"

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    s = input()

    result = timeConversion(s)

    fptr.write(result + '\n')

    fptr.close()

**Mock Test 1 :**

[Click here](https://drive.google.com/file/d/1PL_RqdJMhoW6DuYA2NP_Mg2N-GHeuSDQ/view?usp=drivesdk) to view my Test Report

**Day 2 :**

**Qn 1 : Lonely Integer**

Given an array of integers, where all elements but one occur twice, find the unique element.

**Example**

The unique element is .

**Function Description**

Complete the *lonelyinteger* function in the editor below.

lonelyinteger has the following parameter(s):

* *int a[n]*: an array of integers

**Returns**

* *int:* the element that occurs only once

**Input Format**

The first line contains a single integer, , the number of integers in the array.  
The second line contains  space-separated integers that describe the values in .

**Constraints**

* It is guaranteed that  is an odd number and that there is one unique element.
* , where .

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'lonelyinteger' function below.

#

# The function is expected to return an INTEGER.

# The function accepts INTEGER\_ARRAY a as parameter.

#

def lonelyinteger(a):

    unique = 0

    for num in a:

        unique ^= num

    return unique

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())

    a = list(map(int, input().rstrip().split()))

    result = lonelyinteger(a)

    fptr.write(str(result) + '\n')

    fptr.close()

**Qn 2 : Diagonal Difference**

Given a square matrix, calculate the absolute difference between the sums of its diagonals.

For example, the square matrix  is shown below:

1 2 3

4 5 6

9 8 9

The left-to-right diagonal = . The right to left diagonal = . Their absolute difference is .

**Function description**

Complete the  function in the editor below.

diagonalDifference takes the following parameter:

* *int arr[n][m]*: an array of integers

**Return**

* *int*: the absolute diagonal difference

**Input Format**

The first line contains a single integer, , the number of rows and columns in the square matrix .  
Each of the next  lines describes a row, , and consists of  space-separated integers .

**Constraints**

**Output Format**

Return the absolute difference between the sums of the matrix's two diagonals as a single integer.

**Sample Input**

3

11 2 4

4 5 6

10 8 -12

**Sample Output**

15

**Explanation**

The primary diagonal is:

11

5

-12

Sum across the primary diagonal: 11 + 5 - 12 = 4

The secondary diagonal is:

4

5

10

Sum across the secondary diagonal: 4 + 5 + 10 = 19  
Difference: |4 - 19| = 15

**Note:** |x| is the [absolute value](https://www.mathsisfun.com/numbers/absolute-value.html) of x

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'diagonalDifference' function below.

#

# The function is expected to return an INTEGER.

# The function accepts 2D\_INTEGER\_ARRAY arr as parameter.

#

def diagonalDifference(arr):

    n = len(arr)

    primary\_sum = 0

    secondary\_sum = 0

    for i in range(n):

        primary\_sum += arr[i][i]

        secondary\_sum += arr[i][n-1-i]

    return abs(primary\_sum - secondary\_sum)

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())

    arr = []

    for \_ in range(n):

        arr.append(list(map(int, input().rstrip().split())))

    result = diagonalDifference(arr)

    fptr.write(str(result) + '\n')

    fptr.close()

**Qn 3 : Counting sort 1**

**Comparison Sorting**  
Quicksort usually has a running time of , but is there an algorithm that can sort even faster? In general, this is not possible. Most sorting algorithms are *comparison sorts*, i.e. they sort a list just by comparing the elements to one another. A comparison sort algorithm cannot beat  (worst-case) running time, since  represents the minimum number of comparisons needed to know where to place each element. For more details, you can see [these notes](http://www.cs.cmu.edu/~avrim/451f11/lectures/lect0913.pdf) (PDF).

**Alternative Sorting**  
Another sorting method, the *counting sort*, does not require comparison. Instead, you create an integer array whose index range covers the entire range of values in your array to sort. Each time a value occurs in the original array, you increment the counter at that index. At the end, run through your counting array, printing the value of each non-zero valued index that number of times.

**Example**

All of the values are in the range , so create an array of zeros, . The results of each iteration follow:

i arr[i] result

0 1 [0, 1, 0, 0]

1 1 [0, 2, 0, 0]

2 3 [0, 2, 0, 1]

3 2 [0, 2, 1, 1]

4 1 [0, 3, 1, 1]

The frequency array is . These values can be used to create the sorted array as well: .

**Note**  
For this exercise, always return a frequency array with 100 elements. The example above shows only the first 4 elements, the remainder being zeros.

**Challenge**  
Given a list of integers, count and return the number of times each value appears as an array of integers.

**Function Description**

Complete the *countingSort* function in the editor below.

countingSort has the following parameter(s):

* *arr[n]:* an array of integers

**Returns**

* *int[100]:* a frequency array

**Input Format**

The first line contains an integer , the number of items in .  
Each of the next  lines contains an integer  where .

**Constraints**

**Sample Input**

100

63 25 73 1 98 73 56 84 86 57 16 83 8 25 81 56 9 53 98 67 99 12 83 89 80 91 39 86 76 85 74 39 25 90 59 10 94 32 44 3 89 30 27 79 46 96 27 32 18 21 92 69 81 40 40 34 68 78 24 87 42 69 23 41 78 22 6 90 99 89 50 30 20 1 43 3 70 95 33 46 44 9 69 48 33 60 65 16 82 67 61 32 21 79 75 75 13 87 70 33

**Sample Output**

0 2 0 2 0 0 1 0 1 2 1 0 1 1 0 0 2 0 1 0 1 2 1 1 1 3 0 2 0 0 2 0 3 3 1 0 0 0 0 2 2 1 1 1 2 0 2 0 1 0 1 0 0 1 0 0 2 1 0 1 1 1 0 1 0 1 0 2 1 3 2 0 0 2 1 2 1 0 2 2 1 2 1 2 1 1 2 2 0 3 2 1 1 0 1 1 1 0 2 2

**Explanation**

Each of the resulting values  represents the number of times  appeared in .

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'countingSort' function below.

#

# The function is expected to return an INTEGER\_ARRAY.

# The function accepts INTEGER\_ARRAY arr as parameter.

#

def countingSort(arr):

    # Initialize a list of 100 zeros to store the frequency of each number

    count = [0] \* 100

    # Loop through the input array and update the count for each number

    for num in arr:

        count[num] += 1

    # Return the frequency array

    return count

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())  # Read the number of elements in the array

    arr = list(map(int, input().rstrip().split()))  # Read the array of integers

    result = countingSort(arr)  # Call the countingSort function

    fptr.write(' '.join(map(str, result)))  # Write the result to the output file

    fptr.write('\n')

    fptr.close()  # Close the output file

**Mock Test 2 : Flipping The Matrix**

#!/bin/python3

def flippingMatrix(matrix):

    n = len(matrix) // 2  # Calculate n from the 2n size matrix

    max\_sum = 0

    # Iterate over the nxn submatrix in the upper-left quadrant

    for i in range(n):

        for j in range(n):

            # Calculate the maximum value obtainable from the 4 corresponding quadrants

            max\_value = max(

                matrix[i][j],  # Top-left

                matrix[i][2 \* n - j - 1],  # Top-right

                matrix[2 \* n - i - 1][j],  # Bottom-left

                matrix[2 \* n - i - 1][2 \* n - j - 1]  # Bottom-right

            )

            max\_sum += max\_value

    return max\_sum

if \_\_name\_\_ == '\_\_main\_\_':

    q = int(input().strip())  # Number of queries

    for \_ in range(q):

        n = int(input().strip())  # Dimension of the submatrix

        matrix = []

        for \_ in range(2 \* n):

            matrix.append(list(map(int, input().rstrip().split())))

        result = flippingMatrix(matrix)

        print(result)

**Day 3 :**

**Qn 1 : Zig Zag Sequence**

In this challenge, the task is to debug the existing code to successfully execute all provided test files.

Given an array of  distinct integers, transform the array into a zig zag sequence by permuting the array elements. A sequence will be called a zig zag sequence if the first  elements in the sequence are in increasing order and the last  elements are in decreasing order, where . You need to find the *lexicographically smallest* zig zag sequence of the given array.

**Example**.

Now if we permute the array as , the result is a zig zag sequence.

Debug the given function findZigZagSequence to return the appropriate zig zag sequence for the given input array.

**Note:** You can modify at most *three* lines in the given code. You cannot add or remove lines of code.

*To restore the original code, click on the icon to the right of the language selector.*

**Input Format**

The first line contains  the number of test cases. The first line of each test case contains an integer , denoting the number of array elements. The next line of the test case contains  elements of array .

**Constraints**

 (*is always odd*)

**Output Format**

For each test cases, print the elements of the transformed zig zag sequence in a single line.

**My Answer :**

def findZigZagSequence(a, n):

    a.sort()  # Sort the array to get the lexicographically smallest order

    # Find the middle element (pivot) for the zig-zag

    mid = (n - 1) // 2

    # Swap the middle element with the last element

    a[mid], a[n-1] = a[n-1], a[mid]

    # Reverse the second half of the array

    st = mid + 1

    ed = n - 2  # The second half starts from mid+1 and ends at n-2

    while st <= ed:

        a[st], a[ed] = a[ed], a[st]

        st += 1

        ed -= 1

    # Output the result

    print(\*a)

# Input and execution logic

test\_cases = int(input())  # Read number of test cases

for cs in range(test\_cases):

    n = int(input())  # Read number of elements in the array

    a = list(map(int, input().split()))  # Read the array

    findZigZagSequence(a, n)  # Call the function for each test case

**Qn 2 : Tower Breakers**

Two players are playing a game of Tower Breakers! Player  always moves first, and both players always play optimally.The rules of the game are as follows:

* Initially there are  towers.
* Each tower is of height .
* The players move in alternating turns.
* In each turn, a player can choose a tower of height  and reduce its height to , where  and  [evenly divides](https://en.wiktionary.org/wiki/evenly_divisible) .
* If the current player is unable to make a move, they lose the game.

Given the values of  and , determine which player will win. If the first player wins, return . Otherwise, return .

**Example**. 

There are  towers, each  units tall. Player  has a choice of two moves:  
- remove  pieces from a tower to leave  as   
- remove  pieces to leave

Let Player  remove . Now the towers are  and  units tall.

Player  matches the move. Now the towers are both  units tall.

Now Player  has only one move.

Player  removes  pieces leaving . Towers are  and  units tall.  
Player  matches again. Towers are both  unit tall.

Player  has no move and loses. Return .

**Function Description**

Complete the *towerBreakers* function in the editor below.

towerBreakers has the following paramter(s):

* *int n:* the number of towers
* *int m:* the height of each tower

**Returns**

* *int:* the winner of the game

**Input Format**

The first line contains a single integer , the number of test cases.  
Each of the next  lines describes a test case in the form of  space-separated integers,  and .

**Constraints**

**Sample Input**

STDIN Function

----- --------

2 t = 2

2 2 n = 2, m = 2

1 4 n = 1, m = 4

**Sample Output**

2

1

**Explanation**

We'll refer to player  as  and player  as

In the first test case,  chooses one of the two towers and reduces it to . Then  reduces the remaining tower to a height of . As both towers now have height ,  cannot make a move so  is the winner.

In the second test case, there is only one tower of height .  can reduce it to a height of either  or .  chooses  as both players always choose optimally. Because  has no possible move,  wins.

**My Answer :**

#!/bin/python3

import os

# Complete the 'towerBreakers' function below.

#

# The function is expected to return an INTEGER.

# The function accepts following parameters:

#  1. INTEGER n - number of towers

#  2. INTEGER m - height of each tower

#

def towerBreakers(n, m):

    # If the height of the towers is 1, Player 1 has no valid moves and loses.

    if m == 1:

        return 2  # Player 2 wins if m = 1

    # If n is odd, Player 1 wins. If n is even, Player 2 wins.

    if n % 2 == 1:

        return 1  # Player 1 wins if the number of towers is odd

    else:

        return 2  # Player 2 wins if the number of towers is even

if \_\_name\_\_ == '\_\_main\_\_':

    # Open the file to write the output

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    # Read number of test cases

    t = int(input().strip())

    for t\_itr in range(t):

        # Read the input for each test case

        first\_multiple\_input = input().rstrip().split()

        n = int(first\_multiple\_input[0])  # Number of towers

        m = int(first\_multiple\_input[1])  # Height of each tower

        # Call the function to determine the winner

        result = towerBreakers(n, m)

        # Write the result to the output file

        fptr.write(str(result) + '\n')

    # Close the output file

    fptr.close()

**Qn 3 : Caesar Cipher**

Julius Caesar protected his confidential information by encrypting it using a cipher. [Caesar's cipher](https://en.wikipedia.org/wiki/Caesar_cipher) shifts each letter by a number of letters. If the shift takes you past the end of the alphabet, just rotate back to the front of the alphabet. In the case of a rotation by 3, w, x, y and z would map to z, a, b and c.

Original alphabet: abcdefghijklmnopqrstuvwxyz

Alphabet rotated +3: defghijklmnopqrstuvwxyzabc

**Example**

The alphabet is rotated by , matching the mapping above. The encrypted string is .

**Note:** The cipher *only* encrypts letters; symbols, such as -, remain unencrypted.

**Function Description**

Complete the *caesarCipher* function in the editor below.

caesarCipher has the following parameter(s):

* *string s*: cleartext
* *int k*: the alphabet rotation factor

**Returns**

* *string:* the encrypted string

**Input Format**

The first line contains the integer, , the length of the unencrypted string.  
The second line contains the unencrypted string, .  
The third line contains , the number of letters to rotate the alphabet by.

**Constraints**

 is a valid ASCII string without any spaces.

**Sample Input**

11

middle-Outz

2

**Sample Output**

okffng-Qwvb

**Explanation**

Original alphabet: abcdefghijklmnopqrstuvwxyz

Alphabet rotated +2: cdefghijklmnopqrstuvwxyzab

m -> o

i -> k

d -> f

d -> f

l -> n

e -> g

- -

O -> Q

u -> w

t -> v

z -> b

**My Answer :**

#!/bin/python3

import os

# Complete the 'caesarCipher' function below.

# The function is expected to return a STRING.

# The function accepts following parameters:

#  1. STRING s

#  2. INTEGER k

def caesarCipher(s, k):

    # Initialize an empty list to store the result

    result = []

    # Normalize k to be within 0-25 (since the alphabet has 26 letters)

    k = k % 26

    # Iterate over each character in the string

    for char in s:

        # If the character is lowercase

        if 'a' <= char <= 'z':

            # Shift the character and ensure it wraps around within 'a' to 'z'

            shifted\_char = chr(((ord(char) - ord('a') + k) % 26) + ord('a'))

            result.append(shifted\_char)

        # If the character is uppercase

        elif 'A' <= char <= 'Z':

            # Shift the character and ensure it wraps around within 'A' to 'Z'

            shifted\_char = chr(((ord(char) - ord('A') + k) % 26) + ord('A'))

            result.append(shifted\_char)

        else:

            # If it's not a letter, append the character as it is

            result.append(char)

    # Join the list into a string and return it

    return ''.join(result)

if \_\_name\_\_ == '\_\_main\_\_':

    # Open the file to write the result

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    # Read input

    n = int(input().strip())  # length of the string (not needed for the function)

    s = input()  # the unencrypted string

    k = int(input().strip())  # the shift factor

    # Call the caesarCipher function and get the result

    result = caesarCipher(s, k)

    # Write the result to the output file

    fptr.write(result + '\n')

    # Close the file

    fptr.close()

**Mock Test 3 : Palindrome Index**

def palindromeIndex(s):

    # Helper function to check if a string is a palindrome

    def is\_palindrome(string):

        return string == string[::-1]

    n = len(s)

    # Check from both ends of the string

    for i in range(n // 2):

        if s[i] != s[n - i - 1]:

            # If characters don't match, check by removing either character

            if is\_palindrome(s[i + 1:n - i]):

                return i

            elif is\_palindrome(s[i:n - i - 1]):

                return n - i - 1

            else:

                return -1

    return -1  # The string is already a palindrome

if \_\_name\_\_ == '\_\_main\_\_':

    q = int(input().strip())  # Number of queries

    for \_ in range(q):

        s = input().strip()

        print(palindromeIndex(s))

**Day 4 :**

**Qn 1 : Grid Challenge**

Given a square grid of characters in the range ascii[a-z], rearrange elements of each row alphabetically, ascending. Determine if the columns are also in ascending alphabetical order, top to bottom. Return YES if they are or NO if they are not.

**Example**

The grid is illustrated below.

a b c

a d e

e f g

The rows are already in alphabetical order. The columns a a e, b d f and c e g are also in alphabetical order, so the answer would be YES. Only elements within the same row can be rearranged. They cannot be moved to a different row.

**Function Description**

Complete the *gridChallenge* function in the editor below.

gridChallenge has the following parameter(s):

* *string grid[n]:* an array of strings

**Returns**

* *string:* either YES or NO

**Input Format**

The first line contains , the number of testcases.

Each of the next  sets of lines are described as follows:  
- The first line contains , the number of rows and columns in the grid.  
- The next  lines contains a string of length

**Constraints**

*Each string consists of lowercase letters in the range ascii[a-z]*

**Output Format**

For each test case, on a separate line print YES if it is possible to rearrange the grid alphabetically ascending in both its rows and columns, or NO otherwise.

**Sample Input**

STDIN Function

----- --------

1 t = 1

5 n = 5

ebacd grid = ['ebacd', 'fghij', 'olmkn', 'trpqs', 'xywuv']

fghij

olmkn

trpqs

xywuv

**Sample Output**

YES

**Explanation**

The x grid in the  test case can be reordered to

abcde

fghij

klmno

pqrst

uvwxy

This fulfills the condition since the rows 1, 2, ..., 5 and the columns 1, 2, ..., 5 are all alphabetically sorted.

**My Answer :**

import os

def gridChallenge(grid):

    # Sort each row individually

    sorted\_grid = [sorted(row) for row in grid]

    # Check if columns are sorted

    n = len(sorted\_grid)

    # Check each column for order

    for col in range(n):

        for row in range(1, n):

            if sorted\_grid[row][col] < sorted\_grid[row - 1][col]:

                return "NO"

    return "YES"

if \_\_name\_\_ == '\_\_main\_\_':

    # Open the output file to write results

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    # Read number of test cases

    t = int(input().strip())

    for t\_itr in range(t):

        # Read the size of the grid (n x n)

        n = int(input().strip())

        grid = []

        for \_ in range(n):

            # Read each row of the grid

            grid\_item = input().strip()

            grid.append(grid\_item)

        # Call the gridChallenge function to get the result

        result = gridChallenge(grid)

        # Write the result to the output file

        fptr.write(result + '\n')

    # Close the output file

    fptr.close()

**Qn 2 : Recursive Digit Sum**

We define super digit of an integer  using the following rules:

Given an integer, we need to find the *super digit* of the integer.

* If  has only  digit, then its super digit is .
* Otherwise, the super digit of  is equal to the super digit of the sum of the digits of .

For example, the super digit of  will be calculated as:

super\_digit(9875) 9+8+7+5 = 29

super\_digit(29) 2 + 9 = 11

super\_digit(11) 1 + 1 = 2

super\_digit(2) = 2

**Example**

The number  is created by concatenating the string   times so the initial .

superDigit(p) = superDigit(9875987598759875)

9+8+7+5+9+8+7+5+9+8+7+5+9+8+7+5 = 116

superDigit(p) = superDigit(116)

1+1+6 = 8

superDigit(p) = superDigit(8)

All of the digits of  sum to . The digits of  sum to .  is only one digit, so it is the super digit.

**Function Description**

Complete the function *superDigit* in the editor below. It must return the calculated super digit as an integer.

superDigit has the following parameter(s):

* *string n:* a string representation of an integer
* *int k:* the times to concatenate  to make

**Returns**

* *int:* the super digit of  repeated  times

**Input Format**

The first line contains two space separated integers,  and .

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'superDigit' function below.

#

# The function is expected to return an INTEGER.

# The function accepts following parameters:

#  1. STRING n

#  2. INTEGER k

#

def superDigit(n, k):

    # Helper function to calculate the super digit of a number

    def calculate\_super\_digit(num):

        # If the number has only one digit, return it

        if num < 10:

            return num

        else:

            # Otherwise, calculate the sum of digits and call recursively

            return calculate\_super\_digit(sum(int(digit) for digit in str(num)))

    # Step 1: Calculate the sum of digits of n

    initial\_sum = sum(int(digit) for digit in n)

    # Step 2: Multiply the sum by k (since the number is formed by repeating the string k times)

    total = initial\_sum \* k

    # Step 3: Find the super digit of the resulting total

    return calculate\_super\_digit(total)

if \_\_name\_\_ == '\_\_main\_\_':

    # File pointer to write the result

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    # Read input

    first\_multiple\_input = input().rstrip().split()

    # n is the string representation of the number, k is how many times it's repeated

    n = first\_multiple\_input[0]

    k = int(first\_multiple\_input[1])

    # Get the result from the superDigit function

    result = superDigit(n, k)

    # Write the result to the file

    fptr.write(str(result) + '\n')

    # Close the file pointer

    fptr.close()

**Qn 3 : New Year Chaos**

It is New Year's Day and people are in line for the Wonderland rollercoaster ride. Each person wears a sticker indicating their *initial* position in the queue from  to . Any person can bribe the person *directly in front* of them to swap positions, but they still wear their original sticker. One person can bribe *at most two others*.

Determine the minimum number of bribes that took place to get to a given queue order. Print the number of bribes, or, if anyone has bribed more than two people, print Too chaotic.

**Example**

If person  bribes person , the queue will look like this: . Only  bribe is required. Print 1.

Person  had to bribe  people to get to the current position. Print Too chaotic.

**Function Description**

Complete the function *minimumBribes* in the editor below.

minimumBribes has the following parameter(s):

* *int q[n]*: the positions of the people after all bribes

**Returns**

* No value is returned. Print the minimum number of bribes necessary or Too chaotic if someone has bribed more than  people.

**Input Format**

The first line contains an integer , the number of test cases.

Each of the next  pairs of lines are as follows:  
- The first line contains an integer , the number of people in the queue  
- The second line has  space-separated integers describing the final state of the queue.

**Constraints**

**Subtasks**

For  score   
For  score

**Sample Input**

STDIN Function

----- --------

2 t = 2

5 n = 5

2 1 5 3 4 q = [2, 1, 5, 3, 4]

5 n = 5

2 5 1 3 4 q = [2, 5, 1, 3, 4]

**Sample Output**

3

Too chaotic

**Explanation**

**Test Case 1**

The initial state:



After person  moves one position ahead by bribing person :



Now person  moves another position ahead by bribing person :



And person  moves one position ahead by bribing person :



So the final state is  after three bribing operations.

**Test Case 2**

No person can bribe more than two people, yet it appears person  has done so. It is not possible to achieve the input state.

**My Answer :**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'minimumBribes' function below.

#

# The function accepts INTEGER\_ARRAY q as parameter.

#

def minimumBribes(q):

    bribes = 0

    # Loop through each person in the queue

    for i in range(len(q)-1, -1, -1):

        # Check if this person is more than two positions ahead of their original position

        if q[i] - (i + 1) > 2:

            print("Too chaotic")

            return

        # Count how many times a person has been overtaken by others

        # We only need to check from max(0, q[i]-2) to i-1, since a person can move at most two positions ahead

        for j in range(max(0, q[i] - 2), i):

            if q[j] > q[i]:

                bribes += 1

    print(bribes)

if \_\_name\_\_ == '\_\_main\_\_':

    t = int(input().strip())

    for t\_itr in range(t):

        n = int(input().strip())

        q = list(map(int, input().rstrip().split()))

        minimumBribes(q)

**Mock Test 4 : Truck Tour**

#!/bin/python3

import math

import os

import random

import re

import sys

#

# Complete the 'truckTour' function below.

#

# The function is expected to return an INTEGER.

# The function accepts 2D\_INTEGER\_ARRAY petrolpumps as parameter.

#

def truckTour(petrolpumps):

    total\_surplus = 0  # Total surplus petrol

    current\_balance = 0  # Petrol balance while traversing

    start\_index = 0  # Starting petrol pump index

    for i in range(len(petrolpumps)):

        petrol, distance = petrolpumps[i]

        total\_surplus += petrol - distance

        current\_balance += petrol - distance

        # If current balance is negative, reset the starting point

        if current\_balance < 0:

            start\_index = i + 1

            current\_balance = 0

    # If total surplus is negative, a solution is not possible

    return start\_index if total\_surplus >= 0 else -1

if \_\_name\_\_ == '\_\_main\_\_':

    n = int(input().strip())

    petrolpumps = []

    for \_ in range(n):

        petrolpumps.append(list(map(int, input().rstrip().split())))

    result = truckTour(petrolpumps)

    print(result)