**Permuting Two Arrays**

def revisador(n,A,B,k):

    bandera = True

    for i in range(n):

        if A[i] + B[i] < k:

            bandera = False

            return bandera

    return bandera

def permutador(n,k,A,B):

    A.sort(reverse = True)

    B.sort()

    if revisador(n,A,B,k):

        bandera = True

        return bandera

    else:

        bandera = False

        return bandera

def main():

    queries = int(input())

    cont = 0

    answers = []

    while cont < queries:

        cont += 1

        n,k = input().split(" ")

        n,k = int(n),int(k)

        A = list(map(int,input().split(" ")))

        B = list(map(int,input().split(" ")))

        answers.append(permutador(n,k,A,B))

    for i in range(queries):

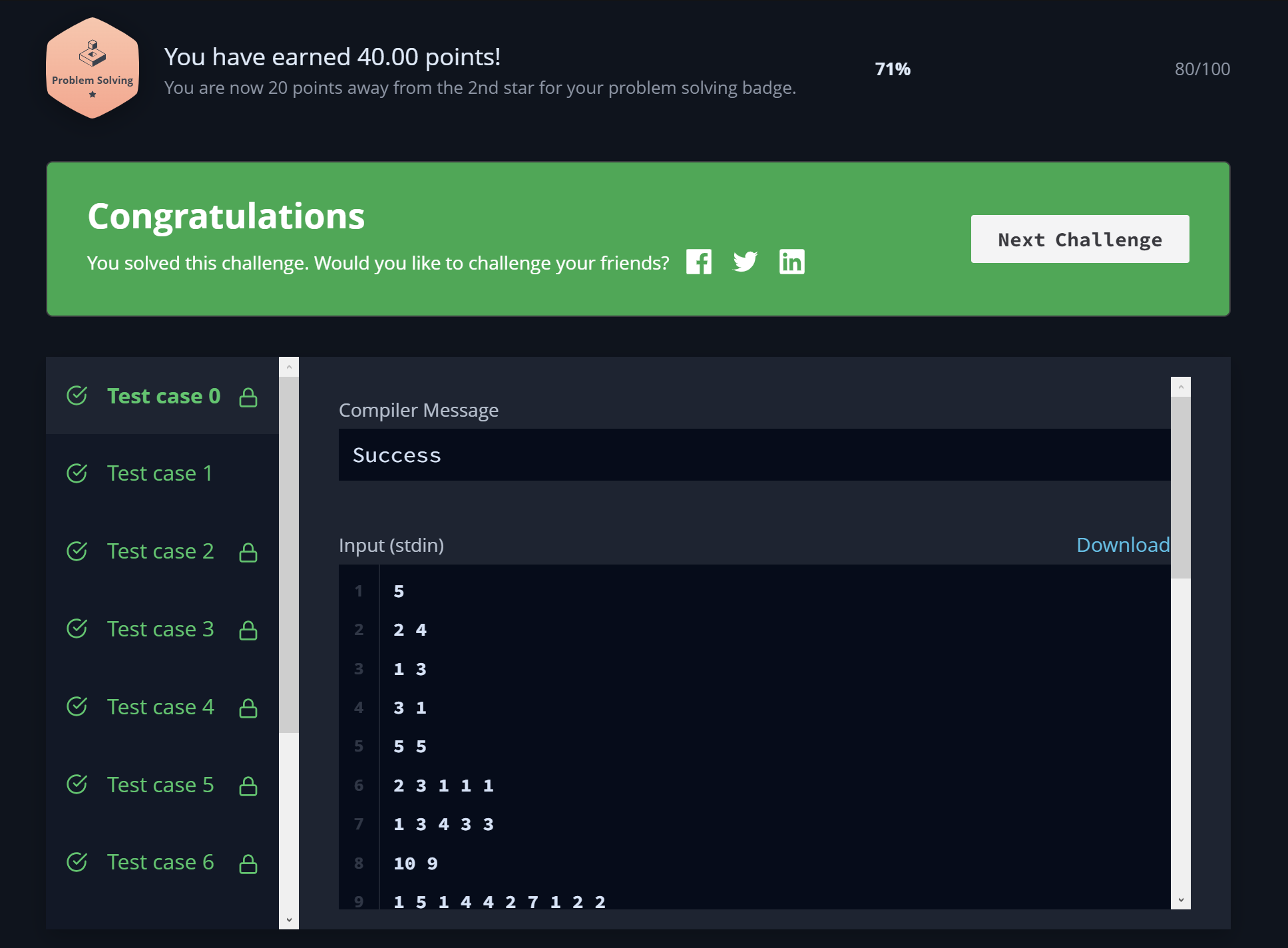
        if answers[i]:

            print("YES")

        else:

            print("NO")

main()



El invariante es el ciclo que recorre las 2 listas ordenadas con la condición dada

# Minimum Absolute Difference in an Array

def minimumAbsoluteDifference(lista, longitud):

    lista.sort()

    menor = abs(lista[0] - lista[1])

    for i in range(1, longitud):

        diferencia = abs(lista[i] - lista[i - 1])

        if diferencia < menor:

            menor = diferencia

    return menor

def main():

    n = int(input().strip())

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

    result = minimumAbsoluteDifference(arr,n)

    print(str(result) + '\n')

main()

Captura de pantalla de un celular

Descripción generada automáticamente

El ciclo es el invariante, pero con un cambio en el intervalo, ya que debe hacer comparaciones con el número siguiente

# Sumar and the Floating Rocks

def encontrar\_mcd(a, b):

    while b:

        a, b = b, a % b

    return a

def solve(x1,y1,x2,y2):

    contador = 0

    a,b = (y1 - y2),(x1 - x2)

    mcd = encontrar\_mcd(a, b)

    Y,X = a // mcd,b // mcd

    while True:

        if Y < 0 or X < 0:

            if y1 < y2:

                x1 -= abs(X)

                y1 += abs(Y)

            if y1 > y2:

                y1 -= abs(Y)

                x1 += abs(X)

        if Y > 0 and X > 0:

            if y1 < y2:

                x1 += abs(X)

                y1 += abs(Y)

            if y1 > y2:

                y1 -= abs(Y)

                x1 -= abs(X)

        if X == 0:

            if y1 < y2:

                y1 += abs(Y)

            if y1 > y2:

                y1 -= abs(Y)

        if Y == 0:

            if x1 < x2:

                x1 += abs(X)

            if x1 > x2:

                x1 -= abs(X)

        if y1 == y2 and x1 == x2:

            return contador

        contador += 1

def main():

    cantidad = int(input())

    cont = 0

    final = []

    while cont < cantidad:

        cont += 1

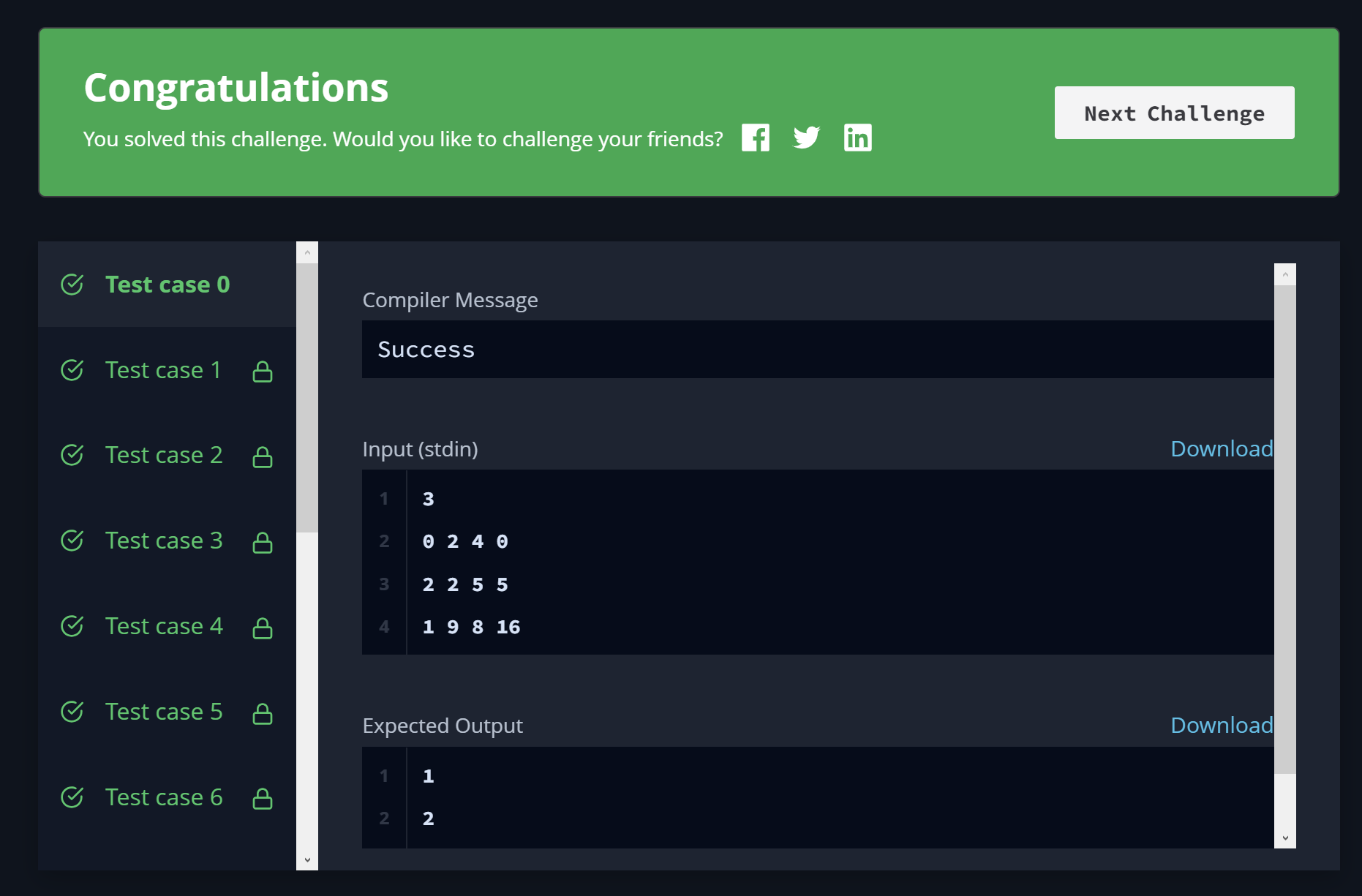
        x1,y1,x2,y2 = map(int,input().split(" "))

        final.append(solve(x1,y1,x2,y2))

    for i in range(cantidad):

        print(final[i])

main()



El invariante es la coordenada 2, ya que la coordenada 1 se altera hasta llegar a la coordenada 2

# Max Min

def maxMin(k,arr,n):

    arr.sort()

    menor = arr[k - 1] - arr[0]

    for i in range(1,(n - k) + 1):

        aux = arr[k + (i -1)] - arr[i]

        if menor > aux:

            menor = aux

    return menor

def main():

    n = int(input())

    k = int(input())

    cont = 0

    lista = []

    while cont < n:

        cont += 1

        lista.append(int(input()))

    print(maxMin(k,lista,n))

main()

Captura de pantalla de un video juego

Descripción generada automáticamente

El invariante es el ciclo que se recorre hasta el último subconjunto del tamaño k

# Priyanka and Toys

def toys(w,n):

    matriz = []

    w.sort()

    inicio = 0

    k = 0

    i = 0

    while i < n:

        if w[i] <= w[inicio] + 4:

            k += 1

            if i == n - 1:

                matriz.append(w[inicio:])

        else:

            matriz.append(w[inicio:inicio + k])

            inicio += k

            i = inicio - 1

            k = 0

        i += 1

    return matriz

def main():

    n = int(input())

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

    print(len(toys(w,n)))

main()

El invariante es el ciclo que se repite buscando hasta que número de peso puede ir el contenedor siguiente

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

# Divisible Sum Pairs

def divisibleSumPairs(n, k, ar):

    cant=0

    for i in range(n):

        for j in range(i):

            if (ar[i]+ar[j])%k==0:

                cant+=1

    return(cant)

def main():

    n,k = input().split(" ")

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

    print(divisibleSumPairs(n, k, ar))

main()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El invariante son los ciclos anidados que miran la divisibilidad de los elementos de la lista

# Extra Long Factorials

def extraLongFactorials(n):

    if n <= 1:

        return n

    else:

        return n \* extraLongFactorials(n - 1)

def main():

    n = int(input().strip())

    print(extraLongFactorials(n))

main()

Captura de pantalla de un celular

Descripción generada automáticamente

El invariante es el valor que entra, ya que por la definición de factorial, se multiplica el valor por el valor - 1

# Mark and Toys

def maximumToys(prices,k,n):

    prices.sort()

    suma = 0

    for i in range(n):

        suma += prices[i]

        if suma >= k:

            return prices[:i]

def main():

    n,k = map(int,input().split(" "))

    prices = list(map(int,input().split(" ")))

    print(len(maximumToys(prices,k,n)))

main()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El invariante es el ciclo que se recorre hasta que la suma de los valores de la lista, sea mayor al presupuesto

# Greedy Florist

def getMinimumCost(k,c,n):

    c.sort(reverse = True)

    suma = 0

    cont = 0

    for i in range(0,n,k):

        try:

            for j in range(i,i + k):

                suma += c[j] \* (cont + 1)

            cont += 1

        except:

            return suma

    return suma

def main():

    n,k = map(int,input().split(" "))

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

    print(getMinimumCost(k,c,n))

main()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El invariante es el ciclo que va recorriendo la lista saltando según la cantidad de amigos que compraron las flores

# Jim’s Orders

def ordenamiento(total,clientes):

  for i in range(1,len(total)):

    insertar = total[i]

    insertar2 = clientes[i]

    indice = i

    while indice > 0 and total[indice - 1] > insertar:

      total[indice] = total[indice - 1]

      clientes[indice] = clientes[indice - 1]

      indice -= 1

    total[indice] = insertar

    clientes[indice] = insertar2

  return clientes

def sumatorias(atencion,orden,clientes):

  total = []

  for i in range(len(clientes)):

    total.append(int(atencion[i]) + int(orden[i]))

  clientes = ordenamiento(total,clientes)

  return clientes

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

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

    cclientes = int(input().strip())

    clientes = []

    for i in range(cclientes):

      clientes.append(i + 1)

    contador = 0

    atencion = []

    orden = []

    while contador < cclientes:

        entrada = input().strip()

        atencion.append(entrada.split(" ")[0])

        orden.append(entrada.split(" ")[1])

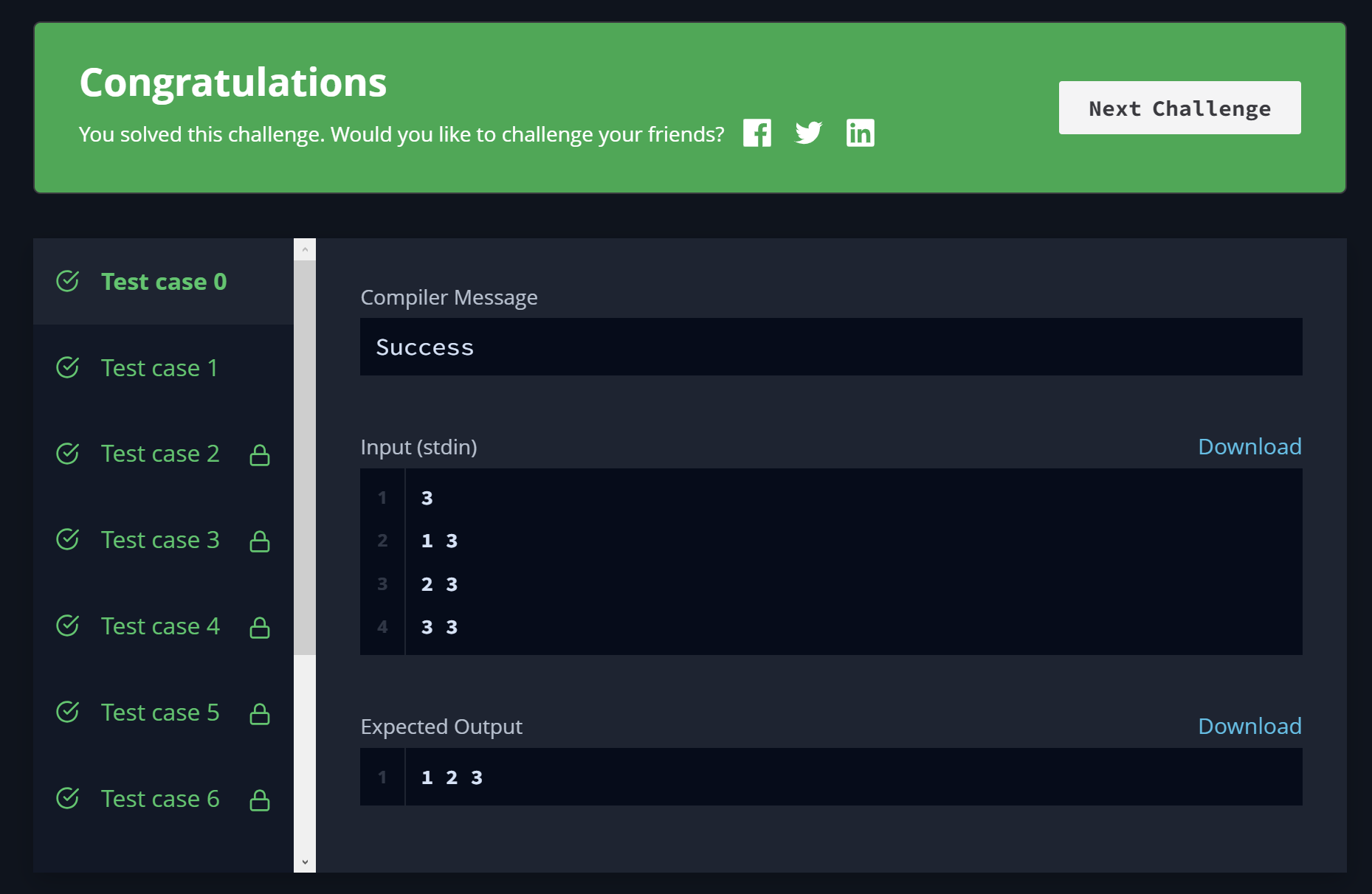
        contador += 1

    final = sumatorias(atencion,orden,clientes)

    fptr.write(' '.join(map(str, final)))

    fptr.write('\n')

fptr.close()



El invariante es la lista con lo que se demora en hacerse la orden del cliente, ya que es el valor que se suma para saber el orden de atención

# Recursive Staircase

def proceso(n):

  if n < 3:

    return n

  lista = [1,1,2]

  cont = 0

  while cont < n - 2:

    lista.append(lista[cont] + lista[cont + 1] + lista[cont + 2])

    cont += 1

  return lista[-1]

def main():

  escaleras = int(input())

  cont = 0

  lista = []

  while cont < escaleras:

      entrada = int(input())

      lista.append(entrada)

      cont += 1

  for i in range(len(lista)):

    final = proceso(lista[i])

    print(final)

main()

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

El invariante es la cantidad de escaleras, ya que la recurrencia es la cantidad de escaleras – 3 más la cantidad – 2 más la cantidad – 1, por lo tanto la cantidad de escaleras nunca cambia

# Eolymp 8609

def recursion(n):

    if n == 0:

        return n

    lista = [0]

    for i in range(1,n + 1):

        lista.append(lista[i - 1] + i)

    return lista[-1]

def main():

    n = int(input())

    print(recursion(n))

main()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El invariante es el valor que entra, ya que la recurrencia es el valor – 1 más el índice del valor

# Gym 201815C

def recursion(n,cont = 0):

    if n == cont:

        return

    print(cont + 1)

    return recursion(n,cont + 1)

def main():

    n = int(input())

    recursion(n)

main()

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

El invariante es el valor que entra

# SPOJ - HRECURS

x = int(input())

for i in range(x):

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

    sum1 = sum(arr[1:])

    y = "Case" + " " + str(i + 1) + ":" + " "+ str(sum1)

    print(y)

Tabla

Descripción generada automáticamente

El invariante es el número de veces que se debe imprimir el mensaje

# Gym-382700A

def recursion(n,cont = 0):

  if cont == n:

    return

  print("I love Recursion")

  recursion(n,cont + 1)

def main():

  n = int(input())

  recursion(n)

main()

Interfaz de usuario gráfica, Aplicación, Tabla

Descripción generada automáticamente

El invariante es el número de veces que se debe imprimir el mensaje

# SC and his recursion

def recursion(A,q):

    lista = [1]

    for i in range(1,q):

        lista.append((A \* lista[i - 1]) + i + 1)

    return lista[-1] + 1

def main():

    T = int(input())

    cont = 0

    while cont < T:

        cont += 1

        A,Q = map(int,input().split())

        cont1 = 0

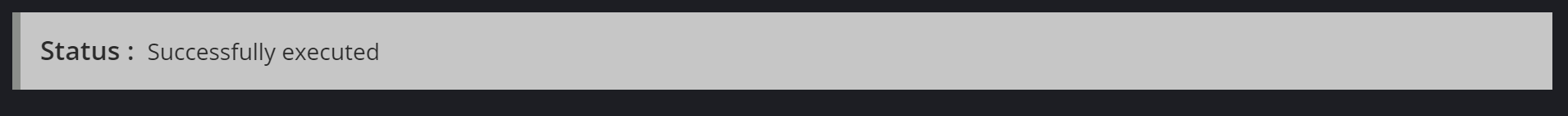
        while cont1 < Q:

            cont1 += 1

            q = int(input())

            print(recursion(A,q))

main()



El invariante es la q, que indica cuando termina el ciclo y el subíndice de la lista a retornar

# Largest Permutation

def largestPermutation(k,arr,n):

    index = {}

    for i in range(n):

        index[arr[i]] = i

    swaps = i = 0

    while swaps < k and i < n:

        if arr[i] < n - i:

            idx = index[n - i]

            arr[i],arr[idx] = arr[idx],arr[i]

            index[arr[idx]] = idx

            index[n - i] = i

            swaps += 1

        i += 1

    return arr

def main():

    n,k = map(int,input().split())

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

    print(' '.join(map(str, largestPermutation(k,arr,n))))

main()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El invariante es la cantidad de movimientos que se pueden hacer para ordernar la lista, ya que este define el límite del ciclo

# Luck Balance

def luckBalance(k, contests):

    a=[]

    s=0

    for i in contests:

        if(i[1]==1):

            a.append(i[0])

        else:

            s+=i[0]

    a.sort(reverse = True)

    for i in a[:k]:

        s+=i

    for i in a[k:]:

        s-=i

    return s

def main():

    n,k = map(int,input().split())

    contests = []

    for \_ in range(n):

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

    print(str(luckBalance(k,contests)) + '\n')

main()

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

El invariante es la cantidad de concursos que se harán, ya que es la clave para saber que suertes restar

# Forming a Magic Square

def formingMagicSquare(s):

    s = sum(s,[])

    magic = [[8, 1, 6, 3, 5, 7, 4, 9, 2], [6, 1, 8, 7, 5, 3, 2, 9, 4], [4, 3, 8, 9, 5, 1, 2, 7, 6], [2, 7, 6, 9, 5, 1, 4, 3, 8],  [2, 9, 4, 7, 5, 3, 6, 1, 8], [4, 9, 2, 3, 5, 7, 8, 1, 6], [6, 7, 2, 1, 5, 9, 8, 3, 4], [8, 3, 4, 1, 5, 9, 6, 7, 2]]

    minimumcost = 10000000000

    for mag in magic:

        diff = 0

        for i,j in zip(s,mag):

            diff += abs(i - j)

        minimumcost = min(minimumcost,diff)

    return minimumcost

def main():

    matriz = []

    matriz.append(input().split())

    matriz.append(input().split())

    matriz.append(input().split())

    matriz = list(map(int,formingMagicSquare(matriz)))

    print(formingMagicSquare(matriz))

main()

Captura de pantalla de un celular

Descripción generada automáticamente

El invariante es la matriz de entrada y la lista de magic, ya que se busca la diferencia entre la entrada y cada una de los valores de magic