DAA-SKILL-2

**1.Marc's Cakewalk**

Marc loves cupcakes, but he also likes to stay fit. Each cupcake has a calorie count, and Marc can walk a distance to expend those calories. If Marc has eaten  cupcakes so far, after eating a cupcake with  calories he must walk *at least*  miles to maintain his weight.

**Example**

If he eats the cupcakes in the order shown, the miles he will need to walk are . This is not the minimum, though, so we need to test other orders of consumption. In this case, our minimum miles is calculated as .

Given the individual calorie counts for each of the cupcakes, determine the minimum number of miles Marc must walk to maintain his weight. Note that he can eat the cupcakes *in any order*.

**Function Description**

Complete the *marcsCakewalk* function in the editor below.

marcsCakewalk has the following parameter(s):

* *int calorie[n]:* the calorie counts for each cupcake

**Returns**

* *long:* the minimum miles necessary

**Input Format**

The first line contains an integer , the number of cupcakes in .  
The second line contains  space-separated integers, .

**Constraints**

**Sample Input 0**

3

1 3 2

**Sample Output 0**

11

**Explanation 0**

Let's say the number of miles Marc must walk to maintain his weight is . He can minimize  by eating the  cupcakes in the following order:

1. Eat the cupcake with  calories, so .
2. Eat the cupcake with  calories, so .
3. Eat the cupcake with  calories, so .

We then print the final value of , which is , as our answer.

**Sample Input 1**

4

7 4 9 6

**Sample Output 1**

79

**Explanation 1**

CODE:

import sys

n = int(input().strip())

calories = list(map(int, input().strip().split(' ')))

calories.sort()

toplam = 0

count = 0

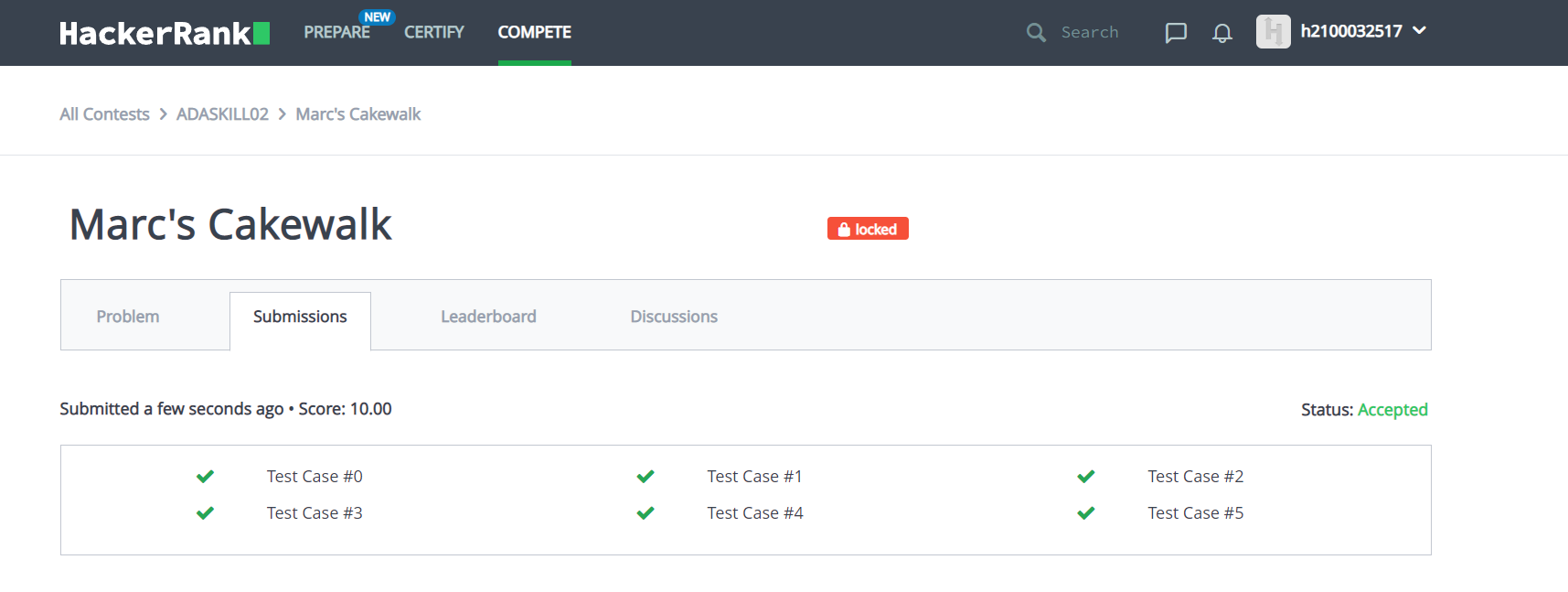
for kalori in calories[::-1]:

toplam += kalori \* (2 \*\* count)

count += 1

print(toplam)

OUTPUT:



2.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.

CODE:

def gridChallenge(arr):

for j in range(len(arr[0])):

for i in range(1,len(arr)):

if arr[i][j]<arr[i-1][j]:

return "NO"

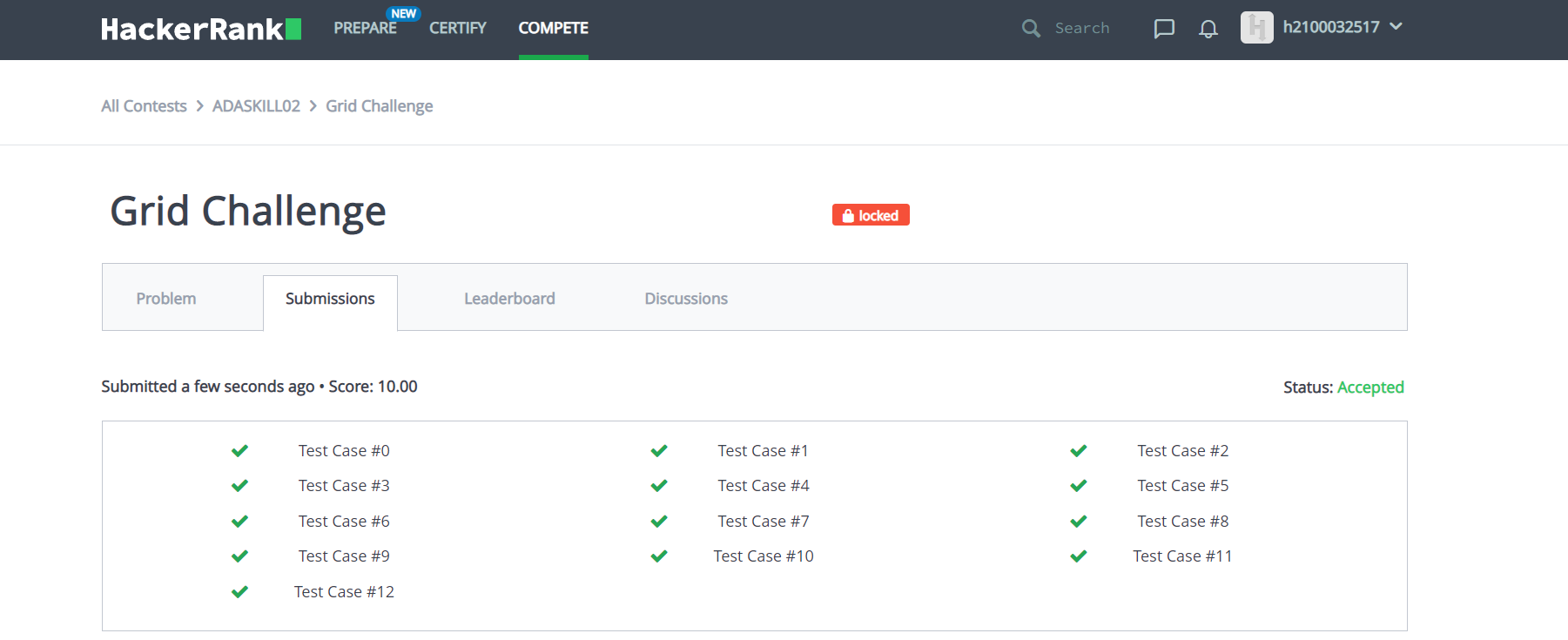
return "YES"

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

arr = [sorted(input()) for i in range(int(input()))]

print(gridChallenge(arr))

OUTPUT:



3.Luck Balance

Lena is preparing for an important coding competition that is preceded by a number of sequential preliminary contests. Initially, her luck balance is 0. She believes in "saving luck", and wants to check her theory. Each contest is described by two integers,  and :

* is the amount of luck associated with a contest. If Lena *wins* the contest, her luck balance will *decrease* by ; if she *loses* it, her luck balance will *increase* by .
* denotes the contest's *importance rating*. It's equal to  if the contest is *important*, and it's equal to  if it's *unimportant*.

If Lena loses no more than  *important* contests, what is the maximum amount of luck she can have after competing in all the preliminary contests? This value *may* be negative.

**Example**

Contest L[i] T[i]

1 5 1

2 1 1

3 4 0

If Lena loses all of the contests, her will be . Since she is allowed to lose  important contests, and there are only  important contests, she can lose all three contests to maximize her luck at .

If , she has to win at least  of the  important contests. She would choose to win the lowest value important contest worth . Her final luck will be .

**Function Description**

Complete the *luckBalance* function in the editor below.

luckBalance has the following parameter(s):

* *int k*: the number of important contests Lena can lose
* *int contests[n][2]:* a 2D array of integers where each  contains two integers that represent the luck balance and importance of the  contest

**Returns**

* *int:* the maximum luck balance achievable

**Input Format**

The first line contains two space-separated integers  and , the number of preliminary contests and the maximum number of important contests Lena can lose.  
Each of the next  lines contains two space-separated integers,  and , the contest's luck balance and its importance rating.

**Constraints**

**Sample Input**

STDIN Function

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

6 3 n = 6, k = 3

5 1 contests = [[5, 1], [2, 1], [1, 1], [8, 1], [10, 0], [5, 0]]

2 1

1 1

8 1

10 0

5 0

**Sample Output**

29

**Explanation**

There are  contests. Of these contests,  are important and she cannot lose more than  of them. Lena maximizes her luck if she wins the  important contest (where ) and loses all of the other five contests for a total luck balance of 29.

CODE:

N, K = map(int, input().strip().split())

luck = 0

important = []

for i in range(N):

L, T = list(map(int, input().strip().split()))

if T == 0:

luck += L

else:

important.append(L)

for i in sorted(important, reverse=True):

if K > 0:

luck += i

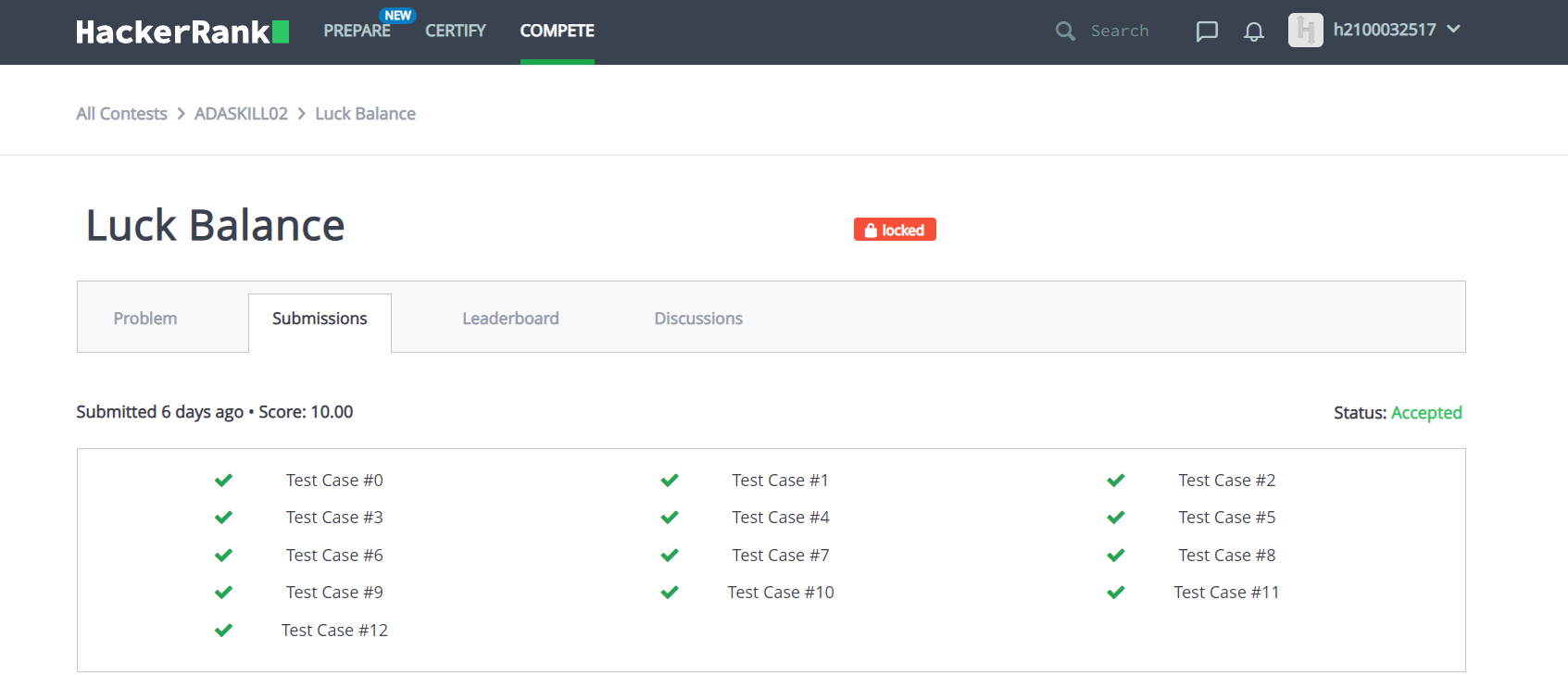
K -= 1

else:

luck -= i

print(luck)

OUTPUT:



4.Maximum Perimeter Triangle

Given an array of stick lengths, use  of them to construct a [non-degenerate triangle](https://www.hackerrank.com/external_redirect?to=https://en.wikipedia.org/wiki/Degeneracy_(mathematics)#Triangle) with the maximum possible perimeter. Return an array of the lengths of its sides as  integers in non-decreasing order.

If there are several valid triangles having the maximum perimeter:

1. Choose the one with the *longest maximum side*.
2. If more than one has that maximum, choose from them the one with the *longest minimum side*.
3. If more than one has that maximum as well, print any one them.

If no non-degenerate triangle exists, return .

**Example**

The triplet  will not form a triangle. Neither will  or , so the problem is reduced to  and . The longer perimeter is .

**Function Description**

Complete the *maximumPerimeterTriangle* function in the editor below.

maximumPerimeterTriangle has the following parameter(s):

* *int sticks[n]:* the lengths of sticks available

**Returns**

* *int[3] or int[1]:* the side lengths of the chosen triangle in non-decreasing order or -1

**Input Format**

The first line contains single integer , the size of array .  
The second line contains  space-separated integers , each a stick length.

**Constraints**

**Sample Input 0**

5

1 1 1 3 3

**Sample Output 0**

1 3 3

**Explanation 0**

There are  possible unique triangles:

The second triangle has the largest perimeter, so we print its side lengths on a new line in non-decreasing order.

**Sample Input 1**

3

1 2 3

**Sample Output 1**

-1

**Explanation 1**

The triangle  is degenerate and thus can't be constructed, so we print -1 on a new line.

**Sample Input 2**

6

1 1 1 2 3 5

**Sample Output 2**

1 1 1

**Explanation 2**

The triangle (1,1,1) is the only valid triangle.

CODE:

tri=int(input())

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

a=sorted(a)

for i in range(0,tri-2):

k=a[tri-1-i]

l=a[tri-2-i]

m=a[tri-3-i]

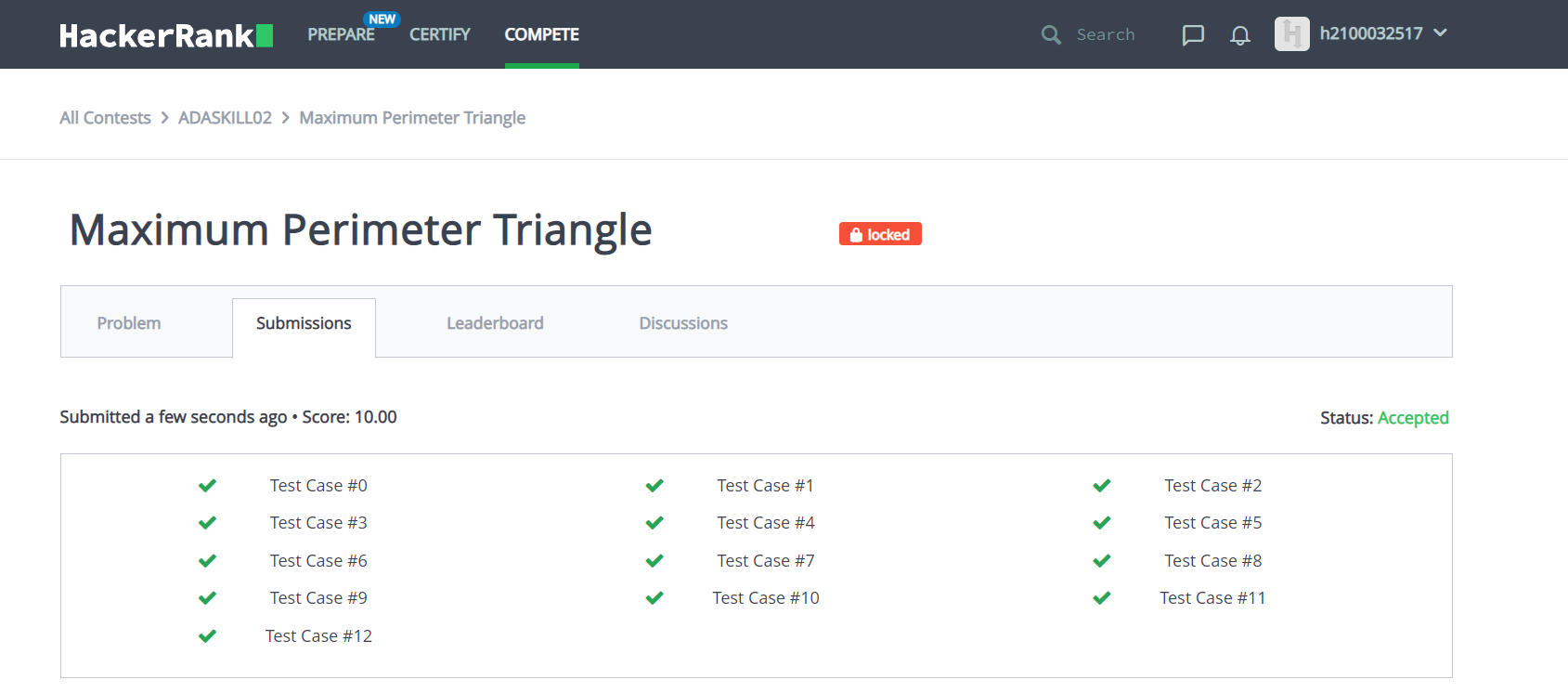
if l+m>k:

print(str(m)+ " "+str(l)+" "+str(k))

exit()

print(-1)

OUTPUT:



5.Beautiful Pairs

You are given two arrays,  and , both containing  integers.

A pair of indices  is *beautiful* if the  element of array  is equal to the  element of array . In other words, pair  is *beautiful* if and only if . A set containing beautiful pairs is called a *beautiful set*.

A beautiful set is called *pairwise disjoint* if for every pair  belonging to the set there is no repetition of either  or  values. For instance, if  and  the beautiful set  is not pairwise disjoint as there is a repetition of , that is .

Your task is to change **exactly** element in  so that the size of the pairwise disjoint beautiful set is maximum.

**Function Description**

Complete the *beautifulPairs* function in the editor below. It should return an integer that represents the maximum number of pairwise disjoint beautiful pairs that can be formed.

beautifulPairs has the following parameters:

* *A*: an array of integers
* *B*: an array of integers

**Input Format**

The first line contains a single integer , the number of elements in  and .  
The second line contains  space-separated integers .  
The third line contains  space-separated integers .

**Constraints**

**Output Format**

Determine and print the maximum possible number of pairwise disjoint beautiful pairs.

**Note:** You must first change  element in , and your choice of element must be optimal.

**Sample Input 0**

4

1 2 3 4

1 2 3 3

**Sample Output 0**

4

**Explanation 0**

You are given  and .  
The beautiful set is  and maximum sized pairwise disjoint beautiful set is either  or .  
We can do better. We change the  element of array  from  to . Now new B array is:  and the pairwise disjoint beautiful set is . So, the answer is 4.  
Note that we could have also selected index 3 instead of index 2 but it would have yeilded the same result. Any other choice of index is not optimal.

**Sample Input 1**

6

3 5 7 11 5 8

5 7 11 10 5 8

**Sample Output 1**

6

CODE:

ap=int(input())

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

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

res=0

for i in range(ap):

for j in range(ap):

if x[i]==y[j] and x[i]>0:

res+= 1

x[i]=0

y[j]=0

if res<ap:

res +=1

else:

res -=1

print(res)

OUTPUT:

