Degradion

Theo

Description

Let’s omit the meaningful story for now...

Given two multisets A & B,

Each operation f(x) -> reduce (one of) the **largest** element in A by x

Ask the shortest and lexicographical largest sequence of operation to reduce A to B.

Performance

The subtasks are hard, and some of the subtasks are not easier than the full solution :(

(I didn’t think about that, sorry :( )

Almost all contestant failed in one of more corner cases or approaches that do not work in special cases

Failing approaches

Output (i from 1 to n) (bi-ai)

Output number of 1s equal to the (sum of ai) - (sum of bi)

...

Sample test cases are really weak (intentionally), let’s take a look on this test case:

3

4 5 6

3 4 5

Observation 1

We will never reduce a same element twice (In some shortest sequence)

Reason:

If we reduce the same element by x1 and x2 with two operations, we can instead reduce this element by (x1+x2) the first time and erase the second time we’re reducing it.

(Why?)

Observation 2

For each element originally in A, we can mark which element it will become in B.

There are at most O(N!) of different markings

We can then construct the sequence consists of **at most** N elements (delete the zeros!!!)

wrong answer Integer parameter [name=magic] equals to 0, violates the range [1, 1000000000] Exited with error status 1

Then we can use some O(N^2) methods to check the validity of the sequence

40% score

Special cases

Sometimes it would help to investigate special instance of the problem first

Important instance: no two elements from A and B respectively are equal

Observation 3

For the special instance, we need to reduce all the element exactly once

(Implying the length of the shortest sequence is exactly **N**. Why?)

Observation 4

There exists a valid sequence for the special instance **iff** the smallest element in A is larger than the largest element in B

The “if” direction is easy, but how about the reverse direction?

Wait

But how to give a lexographical greatest solution?

We know that every “matching” from A to B works

Choose the largest! Obviously we want to match the largest element in A with the smallest element in B, second largest element in A with the second smallest element in B and so on.

Back to general case

In fact, the conclusion for special cases can be applied to the general case too.

Let’s try to “match” each element in A with an equal unmatched element in B whenever it’s possible. - Step 1

Afterwards, all the “unmatched” elements in A need to be reduced.

There exists a solution iff all the unmatched elements are larger than or equal to the largest element in B.

Solution

How can be do Step 1:

O(N^2) checking: 65%

Instead we can use two pointers, which will give O(N) performance and O(N). 100%