Almost Certainly

Subgroup 1.

For each prefix, let's iterate through pairs of numbers that will differ. After that, we will remove them from the arrays and count how many operations are needed to make the arrays equal. The number of operations is equal to the sum of elements in the first array minus the sum of elements in the second array. And they can only be equalized if for each i it holds that $a_i \geq b_i$. This solution works in $O(n^4)$.

Subgroup 2.

Let's improve the solution of the previous subgroup. We want to remove one element from both arrays so that the difference between them is as large as possible. Let's sort both prefixes. Notice that if we can remove a_i and b_j, then we can also remove a_{i-1} and b_j, as well as a_i and b_{j+1}. Then we don't need to iterate through $O(n^2)$ pairs for removal, but we can iterate through the elements of the first array, and find the element of the second array by moving a pointer. This solution works in $O(n^3)$.

Subgroup 3.

Let's further improve the solution of the previous subgroup. We need to check the condition $a_i \geq b_i$ faster. To do this, we need to notice that the b_i element is compared either with a_i or with a_{i+1}. Moreover, both prefixes will be divided into $O(1)$ segments, in which comparisons of one of two types need to be made. Let's precalculate with prefix sums whether the comparisons of each type are satisfied, after which the correctness check can be performed in $O(1)$. This solution works in $O(n^2 \log n)$.

Ideas for the complete solution.

Let's look at the two arrays as a set of segments $[b_i, a_i]$. Notice that it is never beneficial for the final answer to have $a_i < b_i$. Then let's see what the final answer will look like. Remove all indices for which $a_i = b_i$. Sort the remaining numbers in ascending order of a_i, then $a_1 \leq a_2 \leq \ldots \leq a_n$ and $b_1 < a_1, b_2 < a_2, \ldots b_n < a_n$. It is easy to understand that we need to remove a_n and b_1.

What conditions should the segments satisfy so that after removing a_n and b_1 everything is valid? $b_2 \leq a_1, b_3 \leq a_2, \ldots, b_n \leq a_{n-1}$. If we talk about this in terms of segments, this means that they all form a connected component. Then the answer is the sum of the lengths of the segments minus the length in coordinates of the longest component.

Subgroup 4-5.

Using this, it can be understood that the answer in subgroup 4 is equal to the sum of the lengths of the segments minus the length of the longest segment. For the fifth group, it is necessary to maintain the current component, and when possible, expand it and remember the longest component before that.

Complete solution.

In the implementation of the complete solution, let's maintain the current segment components. When moving to a new prefix, we need to be able to merge these components if they intersect with the new segment. All this can be easily maintained in std::set. The time complexity of the solution is $O(n \log n)$.