Typical Tasks

Inversions

A permutation p of n elements is given. You need to find for each element the number of inversions: the number of elements to the left, which are greater. It is solved quite simply. We construct an empty segment tree of size n with the operation sum. Now go through the permutation elements from left to right. A one in a leaf of a tree of segments will mean that such an element was already visited. When moving to the i-th element of $p[i]$, we will make a request to calculate the sum of $[p[i], n]$ in the segment tree: it will just count the number of elements to the left which are greater than $p[i]$. And finally, put one in position $p[i]$. The total time is $O(n \log n)$.

Inversions 2

Array of inversions a is given. You need to restore the permutation p. Let's build a segment tree of size n, which is filled with ones, and is able to respond to the request k-th one from the end. (finding the k-th one from the end is similar to the task of finding the k-th one from the beginning from step 2) Now, let's process the elements from right to left. A one in a leaf of a segment tree will mean that an element with this value is to the left. When moving to the i-th element, we will look for the $a[i]$-th one from the right: this just means that there will $a[i]$ greater elements to the left of the i-th position. Finally, put 0 in the found position, i.e. remove the element from consideration. The total time is $O(n \log n)$.

Nested Segments

We are given an array of length $2n$ where each number from 1 to n occurs exactly two times. Two occurrences of a number indicate the segment between two positions. Task: for each segment, it is necessary to calculate the number of nested segments. In other words, for each x count the number y such that two occurrences of y lie between two occurrences of x. We will process the elements from left to right. Note that for the segment x, the answer is the number of segments that have already been fully considered (both ends on the left), and those with the left end lying between x s.

The algorithm is as follows. Consider the number x. If we consider the left end of the segment, then we do nothing. Otherwise, we consider the number of nested segments between two x. Note that for this it is enough for us to store for each fully considered segment one in the tree of segments at the position of its left end. Then the number of nested segments between x will be equal to the sum on the corresponding segment. When the amount is calculated, put the one at the left end of the segment x. The total time is $O(n \log n)$.

Intersecting segments

Again we are given an array of length $2n$. Two occurrences of each number indicate the segment between two positions. Task: for each segment, it is necessary to calculate the number of segments intersecting with it. In other words, for each x, calculate the number y such that y is between two x and x is between two y. The line segments x and y can intersect in two ways: $x \ldots y \ldots x \ldots y$ and $y \ldots x \ldots y \ldots x$. Let's calculate for each x the number of y segments of the first type. Then we reverse the array and in the same way we calculate the number of segments y of the second type. The sum of these two values will be our answer. We will process the elements from left to right. Note that for the segment x the answer is the number of segments that are not completely considered (only the left end is considered), and those in which the left end lies between the x.

The algorithm is as follows. Consider the number x. If we consider the left end of the segment, then put the one in this position. Otherwise, we consider the sum between the x and set the zero to the left end of the segment x. The total time is $O(n \log n)$.