DVAPRAVCA

There exists an optimal choice of lines such that each of the two lines passes just between some red and some blue point. An $O(N^3 \log N)$ solution is to try each red-blue pair of points for one line, sort all points by their (signed) distance from that first line and then try to extend the region covered by the lines in both directions, as far as possible (until the first blue point).

The point-to-line distance can be calculated in a number of ways: drawing a perpendicular line from the point and finding where the lines intersect, using dot products or using cross products. Whichever way we use, the distance of point (x0, y0) from line Ax+By+C=0 is:

$$\frac{A \cdot x_0 + B \cdot y_0 + C}{\sqrt{A^2 + B^2}}$$

Because we did not take the absolute value of the numerator, this value of this expression can be negative, and it will be for all points in one half of the plane.

This algorithm can be improved to $O(N^3)$ if we notice that it is not necessary to sort the points by their distance. Instead it suffices to just calculate the distance, find the closest blue point on each side of the line and count how many red points are between the line and the blue point. Each of these steps is done in linear time.

For an even better solution, observe that for two lines of almost equal slope, the ordering of points sorted by distance from the lines is almost the same. If we gradually change the slope, occasionally some pairs of points adjacent in the ordering will swap places. Furthermore, if we start for example, from a horizontal line and increase the slope (rotate the line around the origin), for each pair of points we can exactly calculate when they will swap places in the ordering. Because there are $O(N^2)$ pairs of points (and swaps) and they need to be sorted, the complexity of the solution is at least $O(N^2 \log N)$.

We also need to, after each swap, calculate the largest consecutive group of red points in the ordering. Although it is possible to design a data structure to do this in O(1) time, a simple way is to use a complete binary tree. The complexity per swap is O(log N), which does not increase the overall complexity of the solution.

IGRACI

Let A be the source sequence (which needs to be found) and B be the sequence after the transformation described in the task (B is given as input). A key constraint is that $B_k < k$. Suppose that, instead of the sum of quotients as described, B_k is the number of elements between A_1 and A_{k-1} that are less than A_k. Note that with this new transformation the constraint $B_k < k$ will always hold.

Assuming the use of this modified transformation, from the sequence B it is possible to find a sequence A that generates B, consisting of any N distinct integers, for example 1 to N. The last element, A_N, will surely be B_N+1. The second to last, A_{N-1}, will be the $(B_{k-1}+1)$-th smallest of the remaining numbers etc. Using a data structure that supports the operation "find the m-th smallest number and remove it", this reconstruction can be done efficiently. Various binary trees support this kind of operation for a total time complexity of O(N log N).

Finally, note that, if instead of 1 through N we use, for example, the numbers N+1 to 2N, the transformation from the task description and the modified transformation give the same results so the reconstructed sequence is a valid solution for the original transformation.