SOLUTION

Each segment is positioned at a particular **level** – a mast of height H has segments at levels 1 through H. Notice that the ordering of masts is irrelevant when counting the total inefficiency; the final result depends only on the number of sails at each level.

Let us assume that the masts are sorted in ascending order by height. Consider the following greedy algorithm:

- Process masts left to right keeping track of the total number of sails at each level.
- For each new mast of height H with K sails, do the following: out of the H current levels, choose K levels with the lowest number of sails and place the new sails there.
- Add up the values $S_X \cdot (S_X-1) / 2$ for each level X, where S_X is the number of sails placed at level X. Report the sum as the solution.

We will omit the full proof of correctness for the algorithm. Intuitively, our goal is to keep the number of sails at different levels as close as possible, therefore it makes sense to place new sails at levels with the lowest numbers of sails so far.

In order to obtain a formal proof, show that any other choice is no better than the one this algorithm makes. Assume that A and B are different levels such that the number of sails placed on level A so far is less than the number of sails placed on level B so far, and argue that any configuration C that places a sail on B but not on A can be transformed into a configuration C' that does the opposite and has lower or equal inefficiency.

Suboptimal solutions

Although the algorithm is conceptually simple, it is not easy to find an implementation efficient enough for the given input constraints.

The above algorithm can be directly simulated. We can explicitly maintain the total number of sails for each level and update it at each step. If we use an array, we obtain an algorithm of time complexity O(total_number_of_sails · max_height). Such solutions were awarded approximately 30 points.

Notice that, since the masts are sorted by height, we can only keep track of the sail counts, and ignore the particular levels where they are obtained (that is, we can keep the histogram sorted). Hence, we can use a priority queue to maintain the histogram and efficiently find the K lowest sail counts at each step. This approach can be implemented with the time complexity of O(total_number_of_sails · log(max_height)). Such solutions were awarded 40 points.

Keeping the histogram in a sorted array gives another suboptimal solution. At each step, we can find the K lowest sail counts and update the histogram in one sweep. We just need to ensure that, among the levels with same sail count, we pick and update earlier levels first (assuming the histogram is sorted by descending sail counts). This approach can be implemented with the time complexity O(N · max_height). Such solutions were awarded 50 points.

Model solution

The model solution also uses the idea that the histogram can be kept sorted, but, instead of keeping the sail counts, it only maintains the differences between two successive sail counts. For example if, in the current state, the height is 6 and the sail counts are 5, 3, 3, 3, 2 and 2, the difference array, **delta**, is 5, -2, 0, 0, -1, 0 and -2.

The essence of the model solution is a subroutine that transforms the difference array as we process new masts. When processing a new mast of height H with K sails, we first add new entries to the end of the difference array (corresponding to the highest levels that are currently empty), and then we add one sail to each of the lowest K levels.

In order to add one sail to each level in the interval [A, B], in most cases we can simply increase delta[A] and decrease delta[B]. When delta[A] is zero (i.e. when the left part of the interval is in the middle of a group of equal sail counts), then the procedure results in an array that is not sorted. However, whenever delta[A] is not zero the above described procedure results in a difference array corresponding to a sorted array.

Let **level group** be a group of levels with same sail count. Observe that level groups can be identified in the difference array as a sequence of zeros bounded by nonzero values. In other words, to identify the level group containing the level H, we need to find first nonzero value in delta sequence before and after H.

When updating the lowest K levels (interval [H−K+1, H]), the model solution works as follows:

If the left part of the interval (H−K+1) is in the middle of a level group [A, B] then update the intervals [A, A+B−H+K−1] and [B+1, H]. In order to keep the array sorted, we have moved the new sails from the right part to the left part of the level group.

Otherwise, update the interval [H−K+1, H].

In order to obtain an efficient solution we need to be able to quickly find the level group corresponding to a particular level. There are a number of solutions based on this or similar ideas that score between 70 and 100 points, depending on the efficiency of the data structure used to identify the level group.

The model solution uses the interval tree data structure to answer the needed queries. Processing one mast will take time O(log(H)), where H is the height of the mast. Therefore, the total time complexity of this solution is $O(N \cdot \log(\text{max\_height}))$.