Coding Competitions Farewell Rounds - Round C

Analysis: Game Sort: Part 1

Test Set 1

In Test Set 1, the initial idea is to try all possible permutations of each part, but the limits are not small enough to try everything. We observe that it is always better to sort the first part to be the (lexicographically) smallest possible permutation, and to sort the last part to be the (lexicographically) largest possible permutation. Then we can try all permutations of the middle part (there are at most $8!$) and check if any of them makes all parts sorted in non-decreasing lexicographical order.

Test Set 2

For Test Set 2 we need a different approach, as there are too many permutations per part to try. We observe that, for each part, it is best to make it the smallest possible permutation that is not smaller than the previous part (or the smallest permutation in case of the first part). By doing so, we give the next part more opportunities (if any) to be a permutation that is not smaller than the current part.

We see that the best permutation of the current part is the one that shares the longest common prefix with the previous part, such that one of the following conditions holds:

- There is a character we can append to that prefix in the current part that is larger than the next character in the previous part; or
- The whole previous part is a prefix of the current part.

Then we can append the rest of the characters to the current part in sorted order.

One way to do this is to maintain a count of each character per part. When we are at a part, we need to find the longest common prefix with the previous part, subject to the condition above.

The solution above only requires a constant number of scans for each part. This makes the algorithm $O(\sum_i |\mathbf{S_i}|)$, which works for the problem constraints. However, less efficient implementations can also pass.