Code Jam 2021 - Round 3

Analysis: Build-A-Pair

Test Set 1

With the total number of digits up to 8 we can iterate through all possible pairs of numbers we can construct from these digits and choose the optimal one. One way to do this is to iterate through all possible permutations of the given digits and then split every permutation into two in all possible ways as if it were a string. For every such a split we can construct the numbers and calculate the answer. Be careful though, as numbers should not start with 0, every split where 0 is the first digit in a number is not valid.

If we denote the length of D as N, then this takes $O(N! \times N^2)$ time.

Test Set 2

First we can calculate the lengths of each resulting number. These should be as close as possible, which means in case N is even, each number must have exactly $N/2$ digits, and if N is odd, then the larger number must have $\lceil N/2 \rceil$ digits and the smaller number must have $\lfloor N/2 \rfloor$ digits.

Further we will refer to the larger number as A and to the smaller number as B.

Now, let's deal with the case when N is odd. In such a case, we can iterate through all non-zero digits and try them as the first digit of A and B. Now we would want to make A as small as possible (as it is already larger than B), and B as large as possible. Luckily, these two goals are complementary: we can use the $\lfloor N/2 \rfloor$ smallest remaining digits to construct the rest of A (just take them in non-decreasing order), and use the rest to construct B (by taking them in non-increasing order). Notice that choosing the leading digits greedily is also possible, but care must be taken to not choose zero. This way of simply trying everything only takes $O(b^2 \times N)$ time anyway, where b is the base (10), which is fast enough for the limits of the problem.

The case when N is even is trickier, as just choosing the first digits does not give us a unique way to construct the rest of A and B, because there is no guarantee that A will be larger. However, we can use similar technique as part of the solution.

Note that if we have a prefix of length i $A_1 A_2 \ldots A_i$ of A and prefix $B_1 B_2 \ldots B_i$ of B, we can guarantee that A will be larger than B no matter what digits we use further if and only if there is a $k \leq i$ such that $A_j = B_j$ for all $1 \leq j < k$ and $A_k > B_k$. This gives us the following algorithm: choose all possible ways to construct $A_1 A_2 \ldots A_{k-1}$ and $B_1 B_2 \ldots B_{k-1}$, then iterate through all possible A_k and B_k, then construct the rest of A and B so that the former is as small as possible and the latter is as large as possible, using the same technique as above. For each of these possibilities, update the answer with the difference between the constructed numbers. Note, as before, that we need to make sure that the numbers we construct do not start with 0.

The time complexity of such an algorithm is $O(2^{N/2})$ for selecting $A_1 A_2 \ldots A_{k-1}$ and $B_1 B_2 \ldots B_{k-1}$ (these are pairs of equal digits, so we cannot have more than $N/2$ of them, and the order among them is irrelevant, as long as we do not start with zero), $O(b^2)$ for selecting A_k and B_k, and $O(N)$ for constructing the numbers once we got the prefixes. Overall this gives us $O(2^{N/2} \times b^2 \times N$, which is really fast in practice with a good implementation.

The algorithm can be optimized further, but it is not required for the given constraints. There are ways to use the fact that number of digits is limited to reduce $2^{N/2}$ to something with b (the base) instead of N in the exponent. There is also a way to greedily solve the full problem in linear time. Moreover, if the input and output are given in run-length encoding, it can be solved in linear time in that encoding of the input, which is $O(\log N)$.