Permute Digits Problem

Problem Link:

https://codeforces.com/contest/915/problem/C

level of the problem: ***1700**

First Solution: Wrong Approach

The code tries to find the lexicographically smallest string that can be obtained by swapping any two digits of 'a', such that the resulting string is smaller than 'b'.

To do this, it first checks if the length of string 'b' is greater than that of 'a'. If this is true, it means that no matter what swaps we do, we cannot make 'a' smaller than 'b'. So the code sorts the string 'a' in descending order and prints it, as it is already the lexicographically smallest possible string.

If the length of 'b' is not greater than that of 'a', it uses a nested loop to generate all possible permutations of the string 'a' by swapping any two digits. For each permutation, the code checks if the resulting string is smaller than 'b'. If it is, the code updates 'a' to the lexicographically smallest string found so far.

The code uses 'swap' function to swap the digits and 'abs' function to calculate the absolute difference between the integers represented by the two strings. Finally, it prints the lexicographically smallest string 'a' found.

However ,the code will give a runtime error due to the large number of permutations generated by the nested loop.

The number of permutations of a string of length n is n! (n factorial), which can be very large for even moderately sized strings. In the worst case, the code has to generate and compare all n! permutations of string 'a', which can quickly become computationally expensive and lead to a runtime error.

For example, if the length of string 'a' is 10, the number of permutations is 10! = 3,628,800. This means the code has to generate and compare almost 3.6 million permutations, which can take a long time and cause a runtime error if the input is too large.

Link of the Solution:

https://www.ideone.com/Tt6C6s

Second Solution: Efficient Solution

The algorithm works by first sorting the string s1 in non-decreasing order, then looping over all pairs of indices (i,j) in s1, where i is less than j. For each pair of indices, the algorithm swaps the characters at indices (i,j) in s1, sorts the remaining characters in s1 in non-decreasing order, and checks if the resulting permutation of s1 is less than or equal to s2. If it is, the algorithm stops and outputs the resulting permutation of s1. Otherwise, the algorithm restores s1 to its previous state (before the swap) and continues with the next pair of indices.

Here is a step-by-step breakdown of the code:

1. The code starts by including the necessary headers and defining some macros.
2. The main() function reads in the input strings s1 and s2 using the cin function.
3. The length of s1 and s2 are calculated using the length() function.
4. The string s1 is sorted in non-decreasing order using the sort() function.
5. If the length of s1 is lessthan the length of s2, the code reverses s1 using the reverse() function and outputs it, since there is no possible permutation of s1 that is less than or equal to s2 in this case.

6. Otherwise, the code loops over all pairs of indices (i,j) in s1, where i is less than j.
7. For each pair of indices, the code swaps the characters at indices (i,j) in s1 using the swap() function.
8. The substring of s1 starting from index i+1 is then sorted in non-decreasing order using the sort() function, to ensure that the remaining characters in s1 are in non-decreasing order.
9. The resulting permutation of s1 is then checked to see if it is less than or equal to s2. If it is, the algorithm stops and outputs the resulting permutation of s1. Otherwise, the code restores s1 to its previous state (before the swap) and continues with the next pair of indices.
10. Finally, the resulting permutation of s1 is output using the cout function.

Link of the solution:

https://www.ideone.com/6Vew4N