2948. Make Lexicographically Smallest Array by Swapping Elements

You are given a **0-indexed** array of **positive** integers `nums` and a **positive** integer `limit`.

In one operation, you can choose any two indices `i` and `j` and swap `nums[i]` and `nums[j]` **if** `|nums[i] - nums[j]| <= limit`.

Return *the* ***lexicographically smallest array*** *that can be obtained by performing the operation any number of times*.

An array `a` is lexicographically smaller than an array `b` if in the first position where `a` and `b` differ, array `a` has an element that is less than the corresponding element in `b`. For example, the array `[2,10,3]` is lexicographically smaller than the array `[10,2,3]` because they differ at index `0` and `2 < 10`.

Example 1:

```
Input: nums = [1,5,3,9,8], limit = 2
Output: [1,3,5,8,9]
Explanation: Apply the operation 2 times:
- Swap nums[1] with nums[2]. The array becomes [1,3,5,9,8]
- Swap nums[3] with nums[4]. The array becomes [1,3,5,8,9]
We cannot obtain a lexicographically smaller array by applying any more operations.
Note that it may be possible to get the same result by doing different operations.
```

Example 2:

```
Input: nums = [1,7,6,18,2,1], limit = 3
Output: [1,6,7,18,1,2]
Explanation: Apply the operation 3 times:
- Swap nums[1] with nums[2]. The array becomes [1,6,7,18,2,1]
- Swap nums[0] with nums[4]. The array becomes [2,6,7,18,1,1]
- Swap nums[0] with nums[5]. The array becomes [1,6,7,18,1,2]
We cannot obtain a lexicographically smaller array by applying any more operations.
```

Example 3:

```
Input: nums = [1,7,28,19,10], limit = 3
Output: [1,7,28,19,10]
Explanation: [1,7,28,19,10] is the lexicographically smallest array we can obtain
because we cannot apply the operation on any two indices.
```

Constraints:

- `1 <= nums.length <=` 10^5
- `1 <= nums[i] <=` 10^9
- `1 <= limit <=` 10^9

Overview

We are given an array `nums` in which we can swap any two elements `nums[i]` and `nums[j]` if their absolute difference is less than or equal to `limit`. We want to find the lexicographically smallest possible array we can make by applying this swap operation an unlimited number of times on `nums`.

Lexicographical order compares arrays element by element, starting from the leftmost index. For two arrays, the comparison stops as soon as we find an index where the elements differ:

- The array with the smaller element at this differing index is considered smaller.
- If all elements are the same up to the shorter array's length, the shorter array is considered smaller.

For the example arrays given in the problem description, `[2,10,3]` and `[10,2,3]`, the first elements don't match: `2 < 10`, so `[2,10,3]` is lexicographically smaller.

Let's look at another example: For `[1, 1, 3, 5]` and `[1, 1, 2, 7, 9]`, the first occurrence in which the elements do not match is at index 2: `2 < 3` so `[1, 1, 2, 7, 9]` is lexicographically smaller.

2948. Make Lexicographically Smallest Array by Swapping Elements

```
/*
   Sorting+Grouping
   Time complexity: O(nlogn)
   Space complexity: O(n)
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;
class Solution {
   public:
      vi lexicographicallySmallestArray(vi& nums, int limit){
         int n=nums.size();

         // Store values and their indexes
         vii val_idx;
         for(int i=0;i<n;++i) val_idx.push_back({nums[i],i});

         // Sort thel based on values
         std::sort(val_idx.begin(),val_idx.end());

         // Create groups
         vvi groups={{val_idx[0].second}};
         for(int i=1;i<n;++i){
            // nums[i] val_idx[i-1].first
            // nums[j] is val_idx[i].first
            // If a pair (nums[i],nums[j]) statify the rule, add nums[j] to the same group of nums[i]
            if(val_idx[i].first-val_idx[i-1].first<=limit) groups.back().push_back(val_idx[i].second);
            else groups.push_back({val_idx[i].second}); // Otherwise, create a new group for nums[j]
         }
```

```
        // Build the answer
        // For each group
        vi ans(n);
        for(auto& group: groups){
            // Generate the sorted values bases on the indexes in the group
            vi sorted_group_vals;
            for(auto& i: group) sorted_group_vals.push_back(nums[i]);

            // Sort group indexes, this help us to match each value with its
            // correct index in the answer
            std::sort(group.begin(),group.end());

            // Match each value with its correct index in the answer
            // i: Pointer represents the indexes in the answer
            // j: Pointer to iterate over the group indexes
            int j=0;
            for(auto& i: group){
                ans[i]=sorted_group_vals[j++];
            }
        }
        return ans;
    }
};
```