2191. Sort the Jumbled Numbers

You are given a **0-indexed** integer array `mapping` which represents the mapping rule of a shuffled decimal system. `mapping[i] = j` means digit `i` should be mapped to digit `j` in this system.

The **mapped value** of an integer is the new integer obtained by replacing each occurrence of digit `i` in the integer with `mapping[i]` for all `0 <= i <= 9`.

You are also given another integer array `nums`. Return *the array* `nums` *sorted in* ***non-decreasing*** *order based on the* ***mapped values*** *of its elements.*

Notes:

- Elements with the same mapped values should appear in the **same relative order** as in the input.
- The elements of `nums` should only be sorted based on their mapped values and **not be replaced** by them.

Example 1:

```
Input: mapping = [8,9,4,0,2,1,3,5,7,6], nums = [991,338,38]
Output: [338,38,991]
Explanation:
Map the number 991 as follows:
1. mapping[9] = 6, so all occurrences of the digit 9 will become 6.
2. mapping[1] = 9, so all occurrences of the digit 1 will become 9.
Therefore, the mapped value of 991 is 669.
338 maps to 007, or 7 after removing the leading zeros.
38 maps to 07, which is also 7 after removing leading zeros.
Since 338 and 38 share the same mapped value, they should remain in the same
relative order, so 338 comes before 38.
Thus, the sorted array is [338,38,991].
```

Example 2:

```
Input: mapping = [0,1,2,3,4,5,6,7,8,9], nums = [789,456,123]
Output: [123,456,789]
Explanation: 789 maps to 789, 456 maps to 456, and 123 maps to 123. Thus, the
sorted array is [123,456,789].
```

Constraints:

- `mapping.length == 10`
- `0 <= mapping[i] <= 9`
- All the values of `mapping[i]` are **unique**.
- `1 <= nums.length <= 3 * 10`^{4}
- `0 <= nums[i] < 10`^{9}

2191. Sort the Jumbled Numbers

```
/*
   Map each nums[i] with its index in a new array: nums_with_indices[i]={nums{i},i}
   Map each nums[i] using the mapping array in a new array: nums_after_mapping
   Sort nums_with_indices based on nums_after_mapping
   Restore nums[i] from nums_with_indices

   Time complexity: O(nlogn+n+nlogn+n)=O(nlogn)
   Extra time compelxity: O(n+2n+logn)
*/
class Solution {
   public:
      int map_number(int number,std::vector<int>& mapping){
         int new_number=0,mul=1;
         do{
            int r=number%10;
            number/=10;
            new_number+=(mapping[r]*mul);
            mul*=10;
         }while(number!=0);
         return new_number;
      }
      std::vector<int> sortJumbled(std::vector<int>& mapping, std::vector<int>& nums){
         int n=nums.size();
         std::vector<int> nums_after_mapping(n);
         for(int i=0;i<n;++i){
            nums_after_mapping[i]=map_number(nums[i],mapping);
         }

         std::vector<std::pair<int,int>> nums_with_indices;
         for(int i=0;i<n;++i){
            nums_with_indices.push_back({nums[i],i});
         }

         std::sort(nums_with_indices.begin(),nums_with_indices.end(),[&](std::pair<int,int> a,
std::pair<int,int> b){return nums_after_mapping[a.second]<nums_after_mapping[b.second];});

         for(int i=0;i<n;++i) nums[i]=nums_with_indices[i].first;

         return nums;
      }
};
```

2191. Sort the Jumbled Numbers

```
/*
   Packing/Unpacking:
   Pack (nums[i]=nums[i]mapped nums[i])
   Sort based on mapped nums[i]
   Unpack nums[i]

   Time complexity: O(nlogn+n+nlogn+n)=O(nlogn)
   Extra time compelxity: O(n+logn)
*/

class Solution {
   public:
      int map_number(int number,std::vector<int>& mapping){
         int new_number=0,mul=1;
         do{
            int r=number%10;
            number/=10;
            new_number+=(mapping[r]*mul);
            mul*=10;
         }while(number!=0);
         return new_number;
      }

      std::vector<int> sortJumbled(std::vector<int>& mapping, std::vector<int>& nums){
         int n=nums.size();
         std::vector<uint64_t> nums_tmp(n);
         for(int i=0;i<n;++i){
            int x=map_number(nums[i],mapping);
            nums_tmp[i]=((uint64_t(nums[i]))<<34)|((uint64_t(x)));
         }

         std::sort(nums_tmp.begin(),nums_tmp.end(),[&](uint64_t a,uint64_t b){
            return (a&17179869183)<(b&17179869183);
         });

         for(int i=0;i<n;++i) nums[i]=int(nums_tmp[i]>>34);

         return nums;
      }
};
```