670. Maximum Swap

You are given an integer `num`. You can swap two digits at most once to get the maximum valued number.

Return *the maximum valued number you can get*.

Example 1:

```
Input: num = 2736
Output: 7236
Explanation: Swap the number 2 and the number 7.
```

Example 2:

```
Input: num = 9973
Output: 9973
Explanation: No swap.
```

Constraints:

- $0 <= num <= 10^8$

670. Maximum Swap

/*
    Brute force
    Time complexity: $O\left(n^2\right)$
    Space complexity: $O\left(n\right)$
    $n=\log_{10}\left(num\right)$
*/

```
class Solution {
public:
    int maximumSwap(int num){
        std::string s=std::to_string(num);
        int n=s.size();

        // For every digit s[i]
        for(int i=0;i<n-1;++i){
            // Find the last maximum from i+1 to n-1
            char mx='0'; // Last encountered max
            int mx_pos=0; // with its position (will be used for teh swap)
            for(int j=i+1;j<n;++j){
                if(mx<=s[j]){
                    mx=s[j];
                    mx_pos=j;
                }
            }

            // If the current digit s[i] is less than the maximum,
            // perform the swap
            if(s[i]<mx){
                std::swap(s[i],s[mx_pos]);
                return std::stoi(s);
            }
        }

        // If no swap is performed
        return std::stoi(s);
    }
};
```

670. Maximum Swap

```
/*
    Max suffixes
    Time complexity: $O(n+n)=O(n)$
    Space complexity: $O(n+n)=O(n)$
    $n=\log_{10}(num)$
*/
class Solution {
    public:
        int maximumSwap(int num){
            std::string s=std::to_string(num);
            int n=s.size();

            // Preprocess all maximums from the right,
            // store the first encountered max (from the right) and its index (will be used for the swap)
            std::vector<std::pair<char,int>> max_suffix(n);
            max_suffix[n-1]={s[n-1],n-1};
            for(int i=n-2;i>=0;--i){
                if(s[i]>max_suffix[i+1].first) max_suffix[i]={s[i],i};
                else max_suffix[i]=max_suffix[i+1];
            }

            // For every digit s[i]
            for(int i=0;i<n-1;++i){
                // If the current digit s[i] is less than the maximum,
                // perform the swap
                if(s[i]<max_suffix[i].first){
                    std::swap(s[i],s[max_suffix[i].second]);
                    return stoi(s);
                }
            }

            // If no swap is performed
            return stoi(s);
        }
};
```