冒泡排序刷题路线

力扣

数据量 $< 10^3$：冒泡排序 $O(n^2)$ 随便用

冒泡排序核心代码：

```
void bubbleSort(int a[], int n) {
    // 外层循环控制排序轮数，共需要n-1轮
    for (int i = 0; i < n - 1; i++) {
        // swapped标志位，用于优化：如果一轮中没有交换，说明已完全有序
        bool swapped = false;

        // 内层循环进行相邻元素比较交换
        // 注意：每轮排序后，最后的i+1个元素已有序，所以比较范围逐渐减小
        for (int j = 0; j < n - 1 - i; j++) {
            // 如果前一个元素大于后一个元素，则交换（升序排序）
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
                swapped = true;  // 标记发生了交换
            }
        }

        // 优化：如果本轮没有发生交换，说明数组已完全有序，提前结束排序
        if (!swapped) break;
    }
}
```

排序数组

给你一个整数数组 `nums`，请你将该数组升序排列。

你必须在 不使用任何内置函数 的情况下解决问题，时间复杂度为 $O(n \log n)$，并且空间复杂度尽可能小。

示例 1：

- 输入：`nums = [5, 2, 3, 1]`
- 输出：`[1, 2, 3, 5]`
- 解释：数组排序后，某些数字的位置没有改变（例如，2 和 3），而其他数字的位置发生了改变（例如，1 和 5）。

示例 2：

- 输入：`nums = [5, 1, 1, 2, 0, 0]`
- 输出：`[0, 0, 1, 1, 2, 5]`
- 解释：请注意，`nums` 的值不一定唯一。

提示：

- $1 \leq \text{nums.length} \leq 5 \times 10^4$
- $-5 \times 10^4 \leq \text{nums[i]} \leq 5 \times 10^4$

常规写法：

```cpp
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        int n = nums.size();

        for(int i =0;i<n-1;i++){
            bool swapped = false;
            for(int j = 0;j<n-1-i;j++){
                if(nums[j]>nums[j+1]){
                    swap(nums[j],nums[j+1]);
                    swapped = true;
                }
            }
            if(!swapped){
                break;
            }
        }
        return nums;
    }
};
```

> 但超时了

代码实现：

- 快速排序

移动零

题目描述：

给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

请注意，必须在不复制数组的情况下原地对数组进行操作。

示例 1：

- 输入：`nums = [0, 1, 0, 3, 12]`
- 输出：`[1, 3, 12, 0, 0]`

示例 2：

- 输入：`nums = [0]`
- 输出：`[0]`

提示：

- $1 \leq \text{nums.length} \leq 10^4$
- $-2^{31} \leq \text{nums[i]} \leq 2^{31} - 1$

```cpp
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int n = nums.size();

        for(int i =0;i<n-1;i++){
            bool swapped = false;
            for(int j=0;j<n-1-i;j++){
                if(nums[j]==0&&nums[j+1]!=0){
                    swap(nums[j],nums[j+1]);
                    swapped = true;
```

```
                }
            }
            if(!swapped){break;}
        }
    }
};
```

非零元素的先后顺序不能变

不能创建新的数组

多数元素

题目描述：

给定一个大小为 n 的数组 `nums` ，返回其中的多数元素。多数元素是指在数组中出现次数 大于 $\lfloor n/2 \rfloor$ 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

示例 1：

- 输入：`nums = [3, 2, 3]`
- 输出：`3`

示例 2：

- 输入：`nums = [2, 2, 1, 1, 1, 2, 2]`
- 输出：`2`

提示：

- $n == \text{nums.length}$
- $1 \leq n \leq 5 \times 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

```cpp
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int n =nums.size();

        for(int i=0;i<n-1;i++){
            bool swapped = false;

            for(int j=0;j<n-1-i;j++){
                if(nums[j]>nums[j+1]){
                    swap(nums[j],nums[j+1]);
                    swapped = true;
                }
            }
            if(!swapped){break;}
        }
        return nums[n/2];
    }
};
```

将数组排好序后，由于“多数元素”出现次数超过一半，它必然会占据数组的正中间位置

> 多数元素只有一个

> 但超时了

解决方法：

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        //使用sort内置排序
        sort(nums.begin(), nums.end());
        return nums[nums.size() / 2];
    }
};
```

分发饼干

题目描述：

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。

对每个孩子 `i` ，都有一个胃口值 `g[i]` ，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 `j` ，都有一个尺寸 `s[j]` 。如果 `s[j] >=`

`g[i]` ，我们可以将这个饼干 `j` 分配给孩子 `i` ，这个孩子会得到满足。你的目标是满足尽可能多的孩子，并输出这个最大数值。

示例 1：

- 输入： `g = [1,2,3], s = [1,1]`
- 输出： `1`
- 解释： 你有三个孩子和两块小饼干，3 个孩子的胃口值分别是：1, 2, 3。虽然你有两块小饼干，由于他们的尺寸都是 1，你只能让胃口值是 1 的孩子满足。所以你应该输出 1。

提示：

- $1 \leq g.length \leq 3 \times 10^4$
- $0 \leq s.length \leq 3 \times 10^4$
- $1 \leq g[i], s[j] \leq 2^{31} - 1$

```
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
       int n_g = g.size();

       for(int i=0;i<n_g-1;i++){
            bool swapped = false;
            for(int j=0;j<n_g-1-i;j++){
                if(g[j]>g[j+1]){
                    swap(g[j],g[j+1]);
                    swapped = true;
                }
            }
            if(!swapped){break;}
       }

       int n_s = s.size();
       for(int i=0;i<n_s-1;i++){
            bool swapped = false;
```

```
            for(int j=0;j<n_s-1-i;j++){
                if(s[j]>s[j+1]){
                    swap(s[j],s[j+1]);
                    swapped = true;
                }
            }
            if(!swapped){break;}
        }

        int child = 0;
        int cookie = 0;
        while(child < g.size()&& cookie< s.size()){
            if(s[cookie]>=g[child]){
                child++;
            }
            cookie++;
        }

        return child;
    }
};
```