选择排序刷题路线

选择排序是 $O(n^2)$

```
    // 选择排序主循环
    for (int i = 0; i < n - 1; i++) {        // i表示当前已排序部分的末尾位置
        int min_idx = i;                     // 假设当前位置i的元素是最小值

        // 在未排序部分中寻找真正的最小值
        for (int j = i + 1; j < n; j++) {    // j遍历未排序部分
            if (arr[j] < arr[min_idx]) {     // 发现更小的元素
                min_idx = j;                 // 更新最小值索引
            }
        }

        // 将找到的最小值与当前位置交换
        if (min_idx != i) {                  // 如果最小值不是当前位置
            swap(arr[i], arr[min_idx]);      // 交换两个位置的元素
        }
    }
```

力扣

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：

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

```
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0; i < n - 1; i++){
            int minIndex = i;
            for(int j = i + 1; j < n; j++){
                if(nums[j] < nums[minIndex]){
```

```
                    minIndex = j;
                }
            }
            if(minIndex != i){
                swap(nums[i], nums[minIndex]);
            }
        }
        return nums;
    }
};
```

但超时了

数组中的第K个最大元素

题目描述：

给定整数数组 `nums` 和整数 `k`，请返回数组中第 `k` 个最大的元素。

请注意，你需要找的是数组排序后的第 `k` 个最大的元素，而不是第 `k` 个不同的元素。

你必须设计并实现时间复杂度为 $O(n)$ 的算法解决此问题。

示例 1：

- 输入：`[3,2,1,5,6,4], k = 2`
- 输出：`5`

示例 2：

- 输入：`[3,2,3,1,2,4,5,5,6], k = 4`
- 输出：`4`

提示：

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

```
class Solution {
public:
    int findKthLargest(vector<int>& nums, int k) {
        int n = nums.size();
        for(int i=0;i<k;i++){
            int maxIndex = i;
            for(int j=i+1;j<n;j++){
                if(nums[j]>nums[maxIndex]){
                    maxIndex = j;
                }
            }

            swap(nums[i],nums[maxIndex]);
        }
        return nums[k-1];
    }
};
```

为了找“最大”元素，我们在内层循环中要寻找的是最大值的下标

> 每一轮都能确定一个极值的位置；第k轮时，刚好时[k-1]下标最大

> 但超时了