题目：

Given an unsorted array of integers, find the length of longest increasing subsequence.

For example,

Given `[10, 9, 2, 5, 3, 7, 101, 18]`,

The longest increasing subsequence is `[2, 3, 7, 101]`, therefore the length is `4`. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Your algorithm should run in O(n^2) complexity.

Follow up: Could you improve it to O(*n* log *n*) time complexity?

1.时间：O（N^2）;空间：O（1）   -->>理解错题意

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        if (nums.size() < 2) return nums.size();
        const int len = nums.size();
        int maxIncreLen = 0;
        for (int i = 0; i < len; ++i){
            int count = 0;
            for (int k = i + 1; k < len; ++k){
                if (nums[k] > nums[i]){
                    count++;
                }
            }
            maxIncreLen = std::max(maxIncreLen, count);
        }
        return maxIncreLen;
    }
};
```

2.时间：O（N^2）;空间：O（1）   -->>N^2 复杂度，不高效

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        if (nums.size() < 2) return nums.size();
        std::vector<int> dp(nums.size(), 0);
        int maxLen = 0;
        for (int i = 0; i < nums.size(); ++i){
            dp[i] = 1;
            for (int k = 0; k < i; ++k){
                if (nums[i] > nums[k]){
                    dp[i] = std::max(dp[i], dp[k] + 1);
                }
            }
            maxLen = std::max(maxLen, dp[i]);
        }
        return maxLen;
    }
};
```

3.时间：O（LOGN）；空间：O（N）

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        if (nums.size() < 2) return nums.size();
        std::vector<int> dp(nums.size(), 0);
        std::vector<int> ends(nums.size(), 0);
        int rIndex = 0; /* ends 有效区[0~rIndex] */

        int maxLen = 1;
        dp[0] = 1;
        ends[0] = nums[0];
        for (int i = 1; i < nums.size(); ++i){
            /* lower_bound：找到第一个可以插入的位置 */
            auto fIndex = std::lower_bound(ends.begin(), ends.begin() + rIndex + 1, nums[i]) - ends.begin();
            if (fIndex > rIndex){   /* ends 上未有大于等于 nums[i]的数 */
                ends[++rIndex] = nums[i];
                dp[i] = rIndex + 1;
            } else{
                dp[i] = fIndex + 1;
                ends[fIndex] = nums[i];
            }
            maxLen = std::max(maxLen, dp[i]);
        }
        return maxLen;
    }
};
```