國立臺北科技大學
National Taipei University of Technology

資料結構

Data Structure

Lab 02

姓名：曾致嘉

學號：113AB0014

Lab01-Ex1. 1356

Example: Find the number of the smaller numbers(LeetCode 1365)

To-do: (1) Count operations. And (2) Comparison of Method 1 and Method 2, including time complexity or other.

Code

```
class Solution1 {
    public:
    vector<int> smallerNumbersThanCurrent(vector<int>& nums) {
        int counter = 0;//複雜度計算

        int count[101]={0};//計數陣列,範圍 0-100
        int n= nums.size();//陣列長度
        vector<int> result(n);//vector<int>是動態陣列,在執行分配記憶體時,可以根據 num.size()自動調整大小
        counter += 2;//宣告與指派*2 +2

        //計算每個數字的出現次數,此迴圈遍歷 nums,並統計每個數字 num 出現的次數
        for(int num:nums){
            counter++; //歷遍所有的數字 +n
            count[num]++;
        }

        //計算比當前數字小的數量
        for(int i=1; i<101; i++){
            count[i]+=count[i-1];
        }
        counter += 101; //統計所有數字 +101

        //計算 result 陣列
        for(int i=0; i<n; i++){
            counter += 8; // 進入 result*1，進入 nums*2，進入 count*1，三元運算式*1，指派*1，迴圈判斷*1、i 遞增*1 = +8n
            result[i]=(nums[i]==0)?0:count[nums[i]-1];
            //count[nums[i]-1]代表比 nums[i]小的數的數量,nums[i]==0 時則直接返回 0
        }
```

```
            counter++; //返回結果 +1
            return result;
            //f(n) = 2 + n + 101 + 8n + 1 = 9n + 104 => O(n)
      }

};
```

```
class Solution2 {
      public:
      vector<int> smallerNumbersThanCurrent(vector<int>& nums) {
            int counter = 0;//複雜度計算
            vector<int> result; // 存結果
            int count; // 計數器

            // 外層迴圈遍歷 nums 陣列中的每個元素
            for (int i = 0; i < nums.size(); i++) {
                  counter += 2; // 迴圈判斷*1、i 遞增*1 = +2n
                  count = 0; // 每次重新計算當前元素的較小數量
                  counter += 1; // count 指派 +n

                  // 內層迴圈再次遍歷 nums 陣列,統計比 nums[i] 小的元素數量
                  for (int j = 0; j < nums.size(); j++) {
                        counter += 6; // 迴圈判斷*1、j 遞增*1、進入 nums*2，count
遞增＊1 = +6n^2
                        if (nums[j] < nums[i]) { count++; } // 如果 nums[j] 比 nums[i]
小,則計數器 +1
                  }

                  //
                  counter += 1; // 推入 result*1 +n
                  result.push_back(count); // 將計算出的數量存入 result
```

```
        }
        // 返回結果 +1
        counter += 1;
        return result;
        // f(n)   = 2n + n + 6n^2 + n + 1 = 6n^2 + 4n + 1 => O(n^2)
    }
};
```

Discussion Section

2. The function f(n) for Method 1 is 9n+104 with a complexity of O(n), while for Method 2, it is 6n^2 + 4n + 1 with a complexity of O(n^2). Therefore, in terms of time complexity, Method 1 is better.

Lab01-Ex2. 1480
Add clear comments to the program to describe its actions and functions effectively.

Code

```
class Solution {
    public:
    vector<int> runningSum(vector<int>& nums) {
        int n = nums.size(); // 取得陣列長度
        vector<int> result(n);  // 建立一個大小為 n 的 vector
        result[0]=nums[0]; // 第一個元素直接存入

        // 第二個元素之後，為前項累計
        for(int i=1; i<n; i++){
            result[i]=result[i-1]+nums[i];
        }

        // 回傳答案
        return result;
    }
};
```

Lab01-Q1. 1394

the largest number where the digit appears as many times as its value.

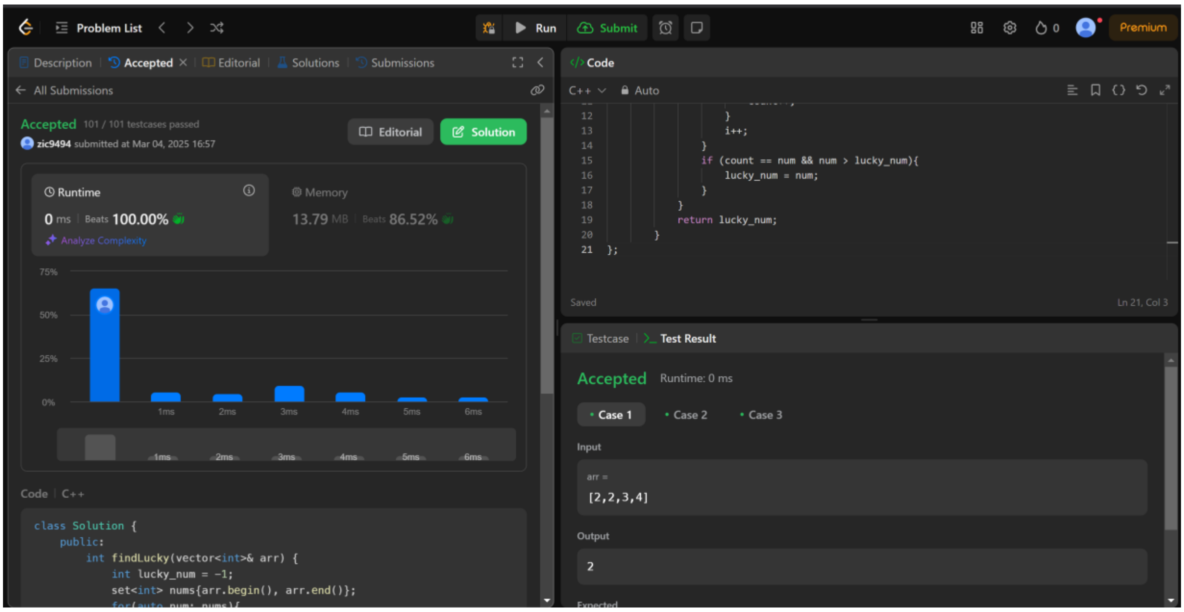
Code

```
class Solution {
    public:
        int findLucky(vector<int>& arr) {
            int lucky_num = -1; // 設定初始值 1
            set<int> nums{arr.begin(), arr.end()}; // 將 arr 轉換為 set，一次插入 set 是 logn= +nlogn

            for(auto num: nums){
                // 指派變數*2 = +2n
                int i = 0;
                int count =0;
                while(i<arr.size()){
                    // 迴圈判斷*1、i 遞增*2 = 3n^2
                    if(arr[i] == num){
                        count++;
                    }
                    i++;
                }

                //判斷*3、指派*1 = +4n
                if (count == num && num > lucky_num){
                    lucky_num = num;
                }
            }
            //返回結果*1 = +1
            return lucky_num;
            //f(n) = nlogn + 2n + 3n^2 + 4n + 1 = 3n^2 + 4n + nlogn + 2 => O(n^2)
        }
};
```

Result



Discussion
3. The time complexity is known to be O(n^2) from the comments in the code.