Leetcode Scratching Recorder

Author: Zhong-Liang Xiang
Start from: August 7th, 2017

1. Two Sum

```
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].
```

人家媳妇：
time $O(n)$, space $O(n)$
方法中巧妙使用map，一次遍历完成任务．
遍历数组时，若未在map中找到想要的元素，则把数组中当前元素投入map．(注key=数组元素值，val=该元素indx)

```
vector<int> twoSum(vector<int> &numbers, int target) {
    //Key is the number and value is its index in the vector.
    map<int, int> hash; //原来人家用的是unordered_map
    vector<int> result;
    for (int unsigned i = 0; i < numbers.size(); i++) {
        int numberToFind = target - numbers[i];

        //if numberToFind is found in map, return them
        if (hash.find(numberToFind) != hash.end()) {
      //map 中若找到
            //+1 because indices are NOT zero based
            result.push_back(hash[numberToFind] + 1);
            result.push_back(i + 1);
            return result;
        }

        //number was not found. Put it in the map.
        hash[numbers[i]] = i;
    }
    return result;
}
```

自家媳妇 :)
time $O(n^2)$, space $O(1)$
Brute Force

```
class Solution {
public:
    vector<int> twoSum(vector<int> nums, int target) {
        vector<int> result;
        for (int i = 0; i < nums.size(); i++)
            for (int j = i + 1; j < nums.size(); j++)
                if (nums[j] == target - nums[i]) {
                    result.push_back(i);
                    result.push_back(j);
                }
        return result;
    }
};
```

27. Remove Element

Given input array `nums = [3,2,2,3]`, `val = 3`

Your function should return length = 2, with the first two elements of nums being 2.
就是把不是3的元素往前移动至队首，再返回不是 3 的元素个数．
$O(n)$ time, $O(0)$ space.

```
int removeElement(vector<int> nums, int val){
    int i = 0, k = 0;
    while(i < nums.size()){
        if(nums[i] != val)
            nums[k++] = nums[i];
        i++;
    }
    return k;
}
```

26. Remove Duplicates from Sorted Array

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,
Given input array `nums = [1,1,2]`, 变成 `[1,2]`

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

自家：
$O(n)$ time, $O(1)$ space.

```
int removeDuplicates(vector<int> &nums) {
    if (nums.size() == 1) return 1;
    if (nums.size() == 0) return 0;
    int i = 1, k = 1;
    while (i < nums.size()) {
        if (nums[i] != nums[i - 1])
            nums[k++] = nums[i];
        i++;
    }
    return k;
}
```

人家:
$O(n)$ time, $O(1)$ space.

```
int removeDuplicates(vector<int> &A) {
    int count = 0;
    int n = A.size();
    for (int i = 1; i < n; i++) {
        if (A[i] == A[i - 1]) count++;
        else A[i - count] = A[i];
    }
    return n - count;
}
```

35. Search Insert Position

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.

```
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0
```

自家媳妇：
$O(n)$ time, $O(1)$ space. 忘记使用 $BinarySearch$ 了．
请记住一件事，有序数组操作，必用$BinarySearch$.

```
int searchInsert(vector<int>& A, int target) {
    if (A.size() == 0) return 0;
    if (A[A.size() - 1] < target) return A.size();
    if (A[0] >= target) return 0;
    int i = 1;
    while (i < A.size()) {
        if (A[i] == target) return i;
        if (A[i - 1] < target && A[i] > target) return i;
        i++;
    }
```

自己的二分查找.
$O(logn)$ time, $O(1)$ space.
经验：

- 1. 设定条件 `while(lo < hi)`
- 2. 退出 `while` 后，`lo` 一定等于 `hi`.
- 3. 在循环外，应再次判断`A[lo]`所指向的值， 因为这个值前面没读取过．

```
int searchInsert(vector<int>& A, int target) {
    int n = A.size();
    if (n == 0) return 0;
    if (A[n - 1] < target) return n;
    if (A[0] >= target) return 0;
    int lo = 0, hi = n - 1, mid;

    while (lo < hi) {
        mid = (lo + hi) / 2;
        if (A[mid] == target) return mid;
        if (A[mid] < target)
            lo = mid + 1;
        if (A[mid] > target)
            hi = mid - 1;
    }
    // lo == hi
    if (A[lo] < target) return lo + 1;
    else return lo;
}
```

人家媳妇：
$O(logn)$ time, $O(1)$ space. 用了二分查找．

```
//TODO
int searchInsert(vector<int>& nums, int target) {
    int n = nums.size();
    if (n == 0) return 0;
    int i = 0, j = n - 1;
    while (i < j - 1) {
        int mid = i + (j - i) / 2;
        if (nums[mid] == target) return mid;
        if (nums[mid] > target) j = mid;
        else i = mid;
    }
    if (target <= nums[i]) return i;
    if (target <= nums[j]) return j;
    return j + 1;
}
```

53. Maximum Subarray

同121题.
Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array `[-2,1,-3,4,-1,2,1,-5,4]`,
the contiguous subarray `[4,-1,2,1]` has the largest sum = `6`.

人家媳妇：
$O(n)$ time, $O(1)$ space.
想法：子序列若小于 0 ，还不如重新设定子序列，既，通过 `max(f+A[i],A[i])` 就可以搞定.

```
int maxSubArray(vector<int>& A) {
    int n = A.size();
    if (n == 0) return 0;

    int re = A[0], sum = 0, i;
    for (i = 0; i < n; i++) {
        sum += A[i];
        re = sum > re ? sum : re; //re 始终存着最大子序列的和
        sum = sum > 0 ? sum : 0; //重新设定子序列
    }
    return re;
}
```

这帮变态还应用了动态规划方法($DP\ approach$).

66. Plus One

Given a non-negative integer represented as a non-empty array of digits, plus one to the integer.

该题目要求：将一整数按位存储在vector中，对其实现+1操作，返回结果.
对存储序列vector中元素，倒序遍历，末位+1，若<10可直接返回，否则，保存进位加之到下一位，循环至最高位.
若此时，进位依然为1，则新建长度增一的vector首位为1，其余为0，返回即可.

$O(n)$ time, $O(1)$ space.
他人思路, 自己的代码:

```
vector<int> plusOne(vector<int>& A) {
    int n = A.size();
    if (A[n - 1] + 1 < 10) {
        A[n - 1] += 1;
        return A;
    } else A[n - 1] = 0;

    //最后一位有为10
    int i;
    for (i = n - 2; i >= 0; i--) {
        A[i] += 1; //倒数第二位+1
        if (A[i] == 10) A[i] = 0;
        else return A;
    }

    // 若循环执行完，仍没return，
    // 则另建一个vector，长度为n+1，首位为 1 ，其余n为为 0 ，并返回
    vector<int> B;
    B.push_back(1);
    for (i = 0; i < n; i++)
        B.push_back(0);
    return B;
}
```

人家更牛逼的代码:

```
vector<int> plusOne(vector<int> &digits) {
    int n = digits.size();
    for (int i = n - 1; i >= 0; --i) {
        if (digits[i] == 9) digits[i] = 0;
        else {
            digits[i]++;
            return digits;
        }
    }

    // 很牛逼的样子
    digits[0] = 1;
    digits.push_back(0);
    return digits;
}
```

88. Merge Sorted Array

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:
You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

默认了升序.
重点是在数组尾部插入元素: `int i = m - 1, j = n - 1, k = m + n - 1;`
还有就是A中元素若比B多,就不管了,A中元素们都在正确位置,但若B元素多需赋值到A中,这点处理需注意.

$O(n)$ time, $O(1)$ space.
照葫芦画瓢代码:

```
void merge(vector<int>& A, int m, vector<int>& B, int n) {
    int i = m - 1, j = n - 1, k = m + n - 1;
    while (i >= 0 & j >= 0)
        A[k--] = A[i] > B[j] ? A[i--] : B[j--];

    while (j >= 0) //A多不管,元素在正确位置,B多得管
        A[k--] = B[j--];
}
```

118. Pascal's Triangle

Given numRows, generate the first numRows of Pascal's triangle.

For example, given `numRows = 5`,
Return

```
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

在每一行后面+1, 再更新该行.
另外学会向`vector`容器中放`vector`元素.
人家代码,自己理解:

```
vector<vector<int>> generate(int numRows) {
    vector<vector<int>> result;
    vector<int> row;

    // i = 0, 该行有1元素, j = -1, 但合并条件是j > 0
    // i = 1, 该行有2元素, j = 0, 但合并条件是j > 0
    // i = 2, 该行有3元素, j 指向倒数第二个元素
    // i = 3, 该行有4元素, j 指向倒数第二个元素
    for (int i = 0; i < numRows; i++) {
        row.push_back(1); // 在尾部先+1

        for (int j = i - 1; j > 0; j--)
            //和j的前一个元素合并至j位置
            row[j] = row[j - 1] + row[j];
        result.push_back(row);
    }
    return result;
}
```

119. Pascal's Triangle II

Given an index k, return the k^{th} row of the Pascal's triangle.

For example, given $k = 3$,
Return `[1,3,3,1]`.

直接生成由0组成的数组,设定`a[0] = 1`,迭代生成最后结果.
纯人家代码,不易理解:

```
vector<int> getRow(int rowIndex) {
    vector<int> A(rowIndex + 1, 0);
    A[0] = 1;
    for (int i = 1; i < rowIndex + 1; i++)
        for (int j = i; j >= 1; j--)
            A[j] += A[j - 1];
    return A;
}
```

121. Best Time to Buy and Sell Stock

Example 1:

```
Input: [7, 1, 5, 3, 6, 4]
Output: 5

max. difference = 6-1 = 5 (not 7-1 = 6, as selling price needs to be larg
er than buying price)
```

Example 2:

```
Input: [7, 6, 4, 3, 1]
Output: 0
```

同53题.

The maximum (minimum) subarray problem. 用到DP.

这题是重点:

就是序列从A[0]开始计算, 不符合条件就清0, 从断处继续计算. 好处是 $O(n)$ time 就可以处理, 牛逼的不行不行的.

$O(n)$ time, $O(1)$ space.

照葫芦画瓢代码:

```
//是个 最大子序列问题 相对应的 有最小子序列问题
//Kadane's algorithm uses the dynamic programming approach
//to find the maximum (minimum) subarray
int maxProfit(vector<int>& A) {
    int i = 1, profit = 0, temp = 0, p = 0;
    while (i < A.size()) {
        temp = A[i] - A[i - 1];
        p = max(0, p += temp); //难点在这,p要和temp累积,小于0则清0
        profit = max(p, profit); //profit记录累积过程中的最大收益值
        i++;
    }
    return profit;
}

int max(int a, int b) {
    return a > b ? a : b;
}
```

更牛逼的代码:
//TODO 还没看呢

```
int maxProfit(vector<int>& prices) {
    int mini = INT_MAX;
    int pro = 0;

    for (int i = 0; i < prices.size(); i++) {
        mini = min(prices[i], mini);
        pro = max(pro, prices[i] - mini);
    }
    return pro;
}
```

122. Best Time to Buy and Sell Stock II

为获得最高收益而多次买卖,但只能像这样买卖: `针对 [7, 1, 5, 3, 6, 4], 我们 buy 1, sell 5; buy 3, sell 6.`

$O(n)$ time, $O(1)$ space.

自家代码:

```
//这代码就简单了
//  7   1   5   3   6   4
//      B   S   B   S
int maxProfit(vector<int>& A) {
    int i, temp, p = 0, profit = 0;
    for (i = 1; i < A.size(); i++) {
        temp = A[i] - A[i - 1];
        p = 0 > temp ? 0 : temp;
        profit += p;
    }
    return profit;
}
```

167. Two Sum II - Input array is sorted

Given an array of integers that is already ***sorted in ascending order***, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have *exactly* one solution and you may not use the *same* element twice.

```
Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2
```

相当于第1题又重做了一遍.注意`map<int, int> hash;`定义方式.
$O(n)$ time, $O(1)$ space.
自己代码:

```
vector<int> twoSum(vector<int>& A, int ta) {
    vector<int> result;
    map<int, int> hash;
    int i, temp, max, min;

    for (i = 0; i < A.size(); i++) {
        temp = ta - A[i];
        if (hash.find(temp) != hash.end()) { //找到了
            max = hash[temp] > i ? hash[temp] : i;
            min = hash[temp] > i ? i : hash[temp];
            result.push_back(min + 1); // bcz base = 1
            result.push_back(max + 1);
            return result;
            //下面代码,就是没找到的话,将对应<k, v>存入hash
        }
        hash[A[i]] = i;
    }
    return result;
}
```

169. Majority Element

Given an array of size n, find the majority element. The majority element is the element that appears **more than** `⌊ n/2 ⌋` times.

You may assume that the array is non-empty and the majority element always exist in the array.

常规办法: 使用 `map<int, int> hash;` 然后顺序扫描并投票. 这办法 robust, but $O(n)$ time, $O(n/2)$ space, bcz hash.

这题前提很重要,就是 **the majority element always exist**. 否则下解出错,如 `3 3 5 6 7`, 会 `return 7`.

重点是下面精巧的逻辑:

```
初始值 major = A[0], count = 1;
从i = 1开始扫数组:
若count==0, count++; major = A[i];
否则若有相等的, count++;
否则(就是不等,且count未归0,那就count-1) count--;
```

$O(n)$ time, $O(1)$ extra space
人家牛逼代码 without hash:

```cpp
int majorityElement(vector<int>& A) {
    int major = A[0], count = 1;
    // 只判断一半是不行的,如 1 2 2 || 3 3 3
    for (int i = 1; i < A.size(); i++) {
        if (count == 0) {
            count++;
            major = A[i];
        } else if (major == A[i]) count++;
        else count--;
    }
    return major;
}
```

189. Rotate Array

Rotate an array of n elements to the right by k steps.

For example, with $n = 7$ and $k = 3$, the array `[1,2,3,4,5,6,7]` is rotated to `[5,6,7,1,2,3,4]`.

Related problem: Reverse Words in a String II

限制为 $O(1)$ extra space 的话, 相对有难度.
人家很帅屁的方法:
`[1,2,3,4,5,6,7]` Original
`[7,6,5,4,3,2,1]` A[0..n-1] reverse
`[5,6,7,4,3,2,1]` A[0..k-1] reverse
`[5,6,7,1,2,3,4]` A[k..n-1] reverse

按人家思路,自己写的代码:

```
void rotate(vector<int>& A, int k) {
    int n = A.size();
    k %= n;
    if (n == 0 || n == 1 || k >= n) return;
    reverse(A, 0, n - 1);
    reverse(A, 0, k - 1);
    reverse(A, k, n - 1);
}

void reverse(vector<int>& A, int start, int end) {
    int temp;
    //代码简洁明了
    while (start < end) { //注意条件
        temp = A[start];
        A[start] = A[end];
        A[end] = temp;
        start++;
        end--;
    }
    // start and end are the index based on 0
    // 自己写的这个有些复杂并且还错了!!!
    // int temp, n = end - start + 1;
    // for (int i = start; i <= (n-1)/2; i++){
    //     temp = A[i];
    //     A[i] = A[n - (i + 1)];
    //     A[n - (i + 1)] = temp;
    // }
}
```

217. Contains Duplicate

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

本题虽然简单, 看下老外的总结: This problem is trivial but **quite nice example of space-time tradeoff**.

直接想到的是用 `set<int> set`:
扫数组, 问是否在set中找到,若有, 则返回 `false`;
若未找到, 将该元素入set;
若顺利扫完数组, 返回 `true`;

学习set声明和简单用法:

```
set<int> set;
set.insert(A[i]);

//表找到元素, set.end()表最后一个元素后面的位置
//find()若找到, 返索引, 否则返回最后一个元素后面的位置
set.find(A[i]) != set.end() //表找到
```

方法1: 简单的双重循环, $O(n^2)$ time, $O(1)$ extra space.
方法2: 先排序,再线性查找, $O(nlogn)$ time, $O(1)$ extra space.

```
bool containsDuplicate(vector<int>& A) {
    int n = A.size();
    if (n == 0 || n == 1) return false;

    sort(A.begin(), A.end()); // 注意: vector中sort的应用
    for (int i = 1; i < n; i++) {
        if (A[i] == A[i - 1]) return true;
    }
    return false;
}
```

方法3: 用set, 想用hash_set,可能更快. $O(n)$ time, $O(n)$ extra space.

```
//用set
bool containsDuplicate(vector<int>& A) {
    int n = A.size();
    set<int> set;
    for (int i = 0; i < n; i++) {//注意下面如何表示找到
        if (set.find(A[i]) != set.end()) return true;
        set.insert(A[i]);
    }
    return false;
}
```

219. Contains Duplicate II

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that **nums[i] = nums[j]** and the **absolute** difference between i and j is at most k.

首先想到,用 `map<int, int>, 其中key = A[i], value = i;`
还想到, 同一个key可以挂多个value. 所以就有了下面的代码.
自己代码:

```
bool containsNearbyDuplicate(vector<int>& A, int k) {
    map<int, int> map;
    for (int i = 0; i < A.size(); i++) {
        if (map.find(A[i]) != map.end() && (i - map[A[i]]) <= k)
            return true;
        map[A[i]] = i;
    }
    return false;
}
```

人家代码:

```
人家代码, 用set
核心是: 维护一个set,其中只包含i之前的k个元素,有重复的也会被set干掉.
如k=3, 当前i=4, 下面indx=0的元素应从set中删除!
0       1 2 3      4
```

```
bool containsNearbyDuplicate(vector<int>& A, int k) {
    unordered_set<int> set; //基于hash实现,比set查询速度快
    if (k < 0) return false;
    for (int i = 0; i < A.size(); i++) {
        if (i > k) set.erase(A[i - k - 1]);
        if (set.find(A[i]) != set.end()) return true;
        set.insert(A[i]);
    }
    return false;
}
```

268. Missing Number

Given an array containing n distinct numbers taken from `0, 1, 2, ..., n`, find the one that is missing from the array.

For example,
Given `nums = [0, 1, 3]` return `2`.

Note:
Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

题意: 从0..n中随机选n个不同的数入数组, 注意:数组乱序. 找出缺失的那个数.
三种方法, XOR, sum, 二分查找(要求数组sorted).

人家牛逼想法:XOR
XOR 相同为0不同为1. `0 ^ 1 ^ 2 ^ 3 ^ 2 ^ 1 ^ 0 = 3`

$O(n)$ time, $O(1)$ extra space. 比下面sum方法快(bit运算).

```
// xor 位运算
int missingNumber(vector<int>& A) {
    int re = 0;
    for (int i = 1; i <= A.size(); i++)
        re = re ^ i ^ A[i - 1];
    return re;
}
```

SUM方法:
$O(n)$ time, $O(1)$ extra space.

```
// 0..size()和 - 数组和
int missingNumber(vector<int>& A) {
    int sum = 0, n = A.size();
    for (int i = 0; i < n; i++)
        sum += A[i];
    int re = (n * (n + 1)) / 2 - sum;
    return re;
}
```

283. Move Zeroes

Given an array `nums`, write a function to move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

For example, given `nums = [0, 1, 0, 3, 12]`, after calling your function, `nums` should be `[1, 3, 12, 0, 0]`.

Note:

1. You must do this in-place without making a copy of the array;
2. Minimize the total number of operations.

人家方法:
扫描数组, 遇非0元素, 送入position的位置. 最后,依据position修改数组末端元素们为0.
$O(n)$ time, $O(1)$ extra space.

```cpp
void moveZeroes(vector<int>& A) {
    int posi = 0, n = A.size();
    for (int i = 0; i < n; i++)
        if (A[i] != 0) A[posi++] = A[i];

    while (posi < n)
        A[posi++] = 0;
}
```

414. Third Maximum Number

Given a **non-empty** array of integers, return the **third** maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in $O(n)$.

Example 1:

```
Input: [3, 2, 1]

Output: 1

Explanation: The third maximum is 1.
```

Example 2:

```
Input: [1, 2]

Output: 2

Explanation: The third maximum does not exist, so the maximum (2) is retu
rned instead.
```

Example 3:

```
Input: [2, 2, 3, 1]

Output: 1

Explanation: Note that the third maximum here means the third maximum dis
tinct number.
Both numbers with value 2 are both considered as second maximum.
```

自己的笨方法, 粗暴, 易理解, 且符合题目要求:

```
使用了set.

扫描三遍数组:
第一遍: 找出max, 并依次把每个元素送入set, 若 set.size() < 3, 返回max;
第二遍找出sec, 第三遍找出third;
返回 third.
```

$O(n)$ time, $O(n)$ extra space.

```
int thirdMax(vector<int>& A) {
    int i, n = A.size();
    int max = INT_MIN, sec = INT_MIN, third = INT_MIN;
    set<int> set;

    for (i = 0; i < n; i++) {
        set.insert(A[i]);
        max = max > A[i] ? max : A[i];
    }

    if (set.size() < 3) return max;

    for (i = 0; i < n; i++)
        if (A[i] != max)
            sec = sec > A[i] ? sec : A[i];

    for (i = 0; i < n; i++)
        if (A[i] != max && A[i] != sec)
            third = third > A[i] ? third : A[i];

    return third;
}
```

人家的精巧方法:
用到 `set, set.erase(), set.begin(), set.rbegin()`
set 默认升序. `set.rbegin()` 表示最后一个元素的迭代器(指针).
$O(n)$ time, $O(n)$ extra space.

```
int thirdMax(vector<int>& A) {
    set<int> top3;
    for (int i = 0; i < A.size(); i++) {
        top3.insert(A[i]);
        // set 默认按 key 升序排序
        if (top3.size() > 3) top3.erase(top3.begin());
    }
    return top3.size() == 3 ? *top3.begin() : *top3.rbegin();
}
```

448. Find All Numbers Disappeared in an Array

Given an array of integers where `1 ≤ a[i] ≤ n` (n = size of array), some elements appear twice and others appear once.

Find all the elements of `[1, n]` inclusive that do not appear in this array.

Could you do it without extra space and in $O(n)$ runtime? You may assume the returned list does not count as extra space.

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[5,6]
```

别人的妙想,实在佩服:
$O(n)$ time, $O(1)$ extra space.
重点: 扫数组,做下面的动作
`nums[nums[i] -1] = -nums[nums[i]-1]`
e.g.

```
 4       3       2       7      8       2        3       1
-4      -3      -2      -7      8       2       -3      -1
```

```
vector<int> findDisappearedNumbers(vector<int>& A) {
    vector<int> re;
    int p, i;
    for (i = 0; i < A.size(); i++) {
        p = abs(A[i]) - 1;
        if (A[p] > 0) A[p] = -A[p];
    }
    for (i = 0; i < A.size(); i++)
        if (A[i] > 0) re.push_back(i + 1);
    return re;
}
```

485. Max Consecutive Ones

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1:

```
Input: [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutiv
e 1s.
The maximum number of consecutive 1s is 3.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

咱家代码:
`count`计数,遇到1则+1并有条件的更新到`max`中(`max = max > co ? max : co;`), 遇到0则`count`清0;
$O(n)$ time, $O(1)$ extra space.

```
int findMaxConsecutiveOnes(vector<int>& A) {
    int co = 0, max = 0, i;
    for (i = 0; i < A.size(); i++) {
        if (A[i] == 1) {
            co++;
            max = max > co ? max : co;
        }
        else co = 0;
    }
    return max;
}
```

532. K-diff Pairs in an Array

Given an array of integers and an integer **k**, you need to find the number of **unique** k-diff pairs in the array. Here a **k-diff** pair is defined as an integer pair (i, j), where **i** and **j** are both numbers in the array and their absolute difference($|i - j|$) is **k**.

Example 1:

```
Input: [3, 1, 4, 1, 5], k = 2
Output: 2
Explanation: There are two 2-diff pairs in the array, (1, 3) and (3, 5).
Although we have two 1s in the input, we should only return the number o
f unique pairs.
```

Example 2:

```
Input:[1, 2, 3, 4, 5], k = 1
Output: 4
Explanation: There are four 1-diff pairs in the arra
y, (1, 2), (2, 3), (3, 4) and (4, 5).
```

Example 3:

```
Input: [1, 3, 1, 5, 4], k = 0
Output: 1
Explanation: There is one 0-diff pair in the array, (1, 1).
```

Note:

1. The pairs (i, j) and (j, i) count as the same pair.
2. The length of the array won't exceed 10,000.
3. All the integers in the given input belong to the range: `[-1e7, 1e7]`.

自己代码不对.

```
//TODO 结果不对,需修改
int findPairs(vector<int>& A, int k) {
    if (k < 0 || A.size() == 0) return 0;
    map<int, int> big;
    map<int, int> small;
    int key1, key2, count = 0;
    for (int i = 0; i < A.size(); i++) {

        key1 = A[i] + k;
        key2 = A[i] - k;

        if (big.find(A[i]) == big.end()) big[A[i]] = A[i] + (k + 1);
        if (small.find(A[i]) == small.end()) small[A[i]] = A[i] - (k + 1);

        if (big.find(key1) != big.end()) { //big 和 samll 都找到了 key 1
            if (A[i] >= key1 && big[key1] == key1 + (k + 1)) {
                big[key1] == A[i];
                count++;
            }
            if (A[i] < key1 && small[key1] == key1 - (k + 1)) {
                small[key1] == A[i];
                count++;
            }
        }

        if (big.find(key2) != big.end()) { //big 和 samll 都找到了 key 2
            if (A[i] >= key2 && big[key2] == key2 + (k + 1)) {
                big[key2] == A[i];
                count++;
            }
            if (A[i] < key2 && small[key2] == key2 - (k + 1)) {
                small[key2] == A[i];
                count++;
            }
        }
    }
    return count;
}
```

561. Array Partition I

Given an array of **2n** integers, your task is to group these integers into **n** pairs of integer, say $(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)$ which makes sum of $min(a_i, b_i)$ for all i from 1 to n as large as possible.

Example 1:

```
Input: [1,4,3,2]

Output: 4
Explanation: n is 2, and the maximum sum of pairs is 4 = min(1, 2) + min
(3, 4).
```

Note:

1. **n** is a positive integer, which is in the range of `[1, 10000]`.
2. All the integers in the array will be in the range of `[-10000, 10000]`.

自家代码:
先排序, 索引为偶数元素求和.
副产品为: vector的排序方式 `sort(A.begin(), A.end())`.
$O(nlogn)$ time, $O(1)$ extra space.

```
int arrayPairSum(vector<int>& A) {
    sort(A.begin(), A.end());
    int sum = 0;
    for (int i = 0; i < A.size(); i += 2) {
        sum += A[i];
    }
    return sum;
}
```

566. Reshape the Matrix

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.
Example 1:

```
Input:
nums =
[[1,2],
 [3,4]]
r = 1, c = 4
Output:
[[1,2,3,4]]
Explanation:
The row-traversing of nums is [1,2,3,4]. The new reshaped matrix i
s a 1 * 4 matrix, fill it row by row by using the previous list.
```

Example 2:

```
Input:
nums =
[[1,2],
 [3,4]]
r = 2, c = 4
Output:
[[1,2],
 [3,4]]
Explanation:
There is no way to reshape a 2 * 2 matrix to a 2 * 4 matrix. So output th
e original matrix.
```

Note:

1. The height and width of the given matrix is in range `[1, 100]`.
2. The given r and c are all **positive**.

我的照葫芦画瓢代码:
精巧想法是: `res[i / c][i % c] = A[i / cc][i % cc];` 老服气了!
副产品: `vector<vector<int>>vec(m,vector<int>(n,0));`
m*n的二维vector，所有元素为0

```
vector<vector<int>> matrixReshape(vector<vector<int>>& A, int r, int c) {
    int rr = A.size(), cc = A[0].size();
    if (rr * cc != r * c) return A;

    vector<vector<int>> res(r, vector<int>(c, 0));
    for (int i = 0; i < r * c; i++)
        res[i / c][i % c] = A[i / cc][i % cc];
    return res;
}
```

581. Shortest Unsorted Continuous Subarray (TODO)

Given an integer array, you need to find one **continuous subarray** that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order, too.

You need to find the **shortest** such subarray and output its length.

Example 1:

```
Input: [2, 6, 4, 8, 10, 9, 15]
Output: 5
Explanation: You need to sort [6, 4, 8, 10, 9] in ascending order to mak
e the whole array sorted in ascending order.
```

Note:

1. Then length of the input array is in range `[1, 10,000]`.
2. The input array may contain duplicates, so ascending order here means `<=`.

643. Maximum Average Subarray

Given an array consisting of n integers, find the contiguous subarray of given length k that has the maximum average value. And you need to output the maximum average value.

Example 1:

```
Input: [1,12,-5,-6,50,3], k = 4
Output: 12.75
Explanation: Maximum average is (12-5-6+50)/4 = 51/4 = 12.75
```

Note:

1. `1 <= k <= n <= 30,000`.
2. Elements of the given array will be in the range `[-10,000, 10,000]`.

自己方法1 Brute Force:
窗口 `k = 4`, 从数组左到右滑动, 代码省略.
$O(n * k)$ time, $O(1)$ extra space.

自个方法2 改进版的 Brute Force:
想法如下:

```
k = 4 的情形

1   12  -5  -6  50  3
^           ^

1   12  -5  -6  50  3
    ^           ^
明显的, 第一行的 sum  + 50  -  1 就是第二行的 sum
                    A[i]   A[i-k]
```

$O(n)$ time, $O(1)$ extra space.

```
double findMaxAverage(vector<int>& A, int k) {
    int i, sum = 0, max = INT_MIN;
    for (i = 0; i < k; i++) sum += A[i];
    max = max > sum ? max : sum;
    for (i = k; i < A.size(); i++) {
        sum = sum + A[i] - A[i - k];
        max = max > sum ? max : sum;
    }
    return 1.0 * max / k;
}
```

628. Maximum Product of Three Numbers

Given an integer array, find three numbers whose product is maximum and output the maximum product.

Example 1:

```
Input: [1,2,3]
Output: 6
```

Example 2:

```
Input: [1,2,3,4]
Output: 24
```

Note:

1. The length of the given array will be in range [3,104] and all elements are in the range [-1000, 1000].
2. Multiplication of any three numbers in the input won't exceed the range of 32-bit signed integer.

理很明白, 逻辑没弄清.
想法是:找出 top3 最大, top2 最小, 一趟找出, 关键是逻辑.
形象一些讲就是: **孩子们长身体,老大穿不了的衣服传给老二,老二穿不了的传给老三,老三找到合适衣服直接穿上**.

人家逻辑:
$O(n)$ time, $O(1)$ extra space.

```
int maximumProduct(vector<int>& A) {
    int m1 = INT_MIN, m2 = INT_MIN;
    int m3 = INT_MIN, s1 = INT_MAX, s2 = INT_MAX, res, i, v;

    //孩子们长身体,老大穿不了的衣服传给老二,老二穿不了的传给老三.
    for (i = 0; i < A.size(); i++) {
        v = A[i];
        if (v > m1) {m3 = m2; m2 = m1; m1 = v;}
        else if (v > m2) {m3 = m2; m2 = v;}
        else if (v > m3) {m3 = v;}

        if (v < s1) {s2 = s1; s1 = v;}
        else if (v < s2) {s2 = v;}
    }
    res = m1 * m2 * m3 > m1 * s1 * s2 ? m1 * m2 * m3 : m1 * s1 * s2;
    return res;
}
```

605. Can Place Flowers

Suppose you have a long flowerbed in which some of the plots are planted and some are not. However, flowers cannot be planted in adjacent plots - they would compete for water and both would die.

Given a flowerbed (represented as an array containing 0 and 1, where 0 means empty and 1 means not empty), and a number n, return if n new flowers can be planted in it without violating the no-adjacent-flowers rule.

Example 1:

```
Input: flowerbed = [1,0,0,0,1], n = 1
Output: True
```

Example 2:

```
Input: flowerbed = [1,0,0,0,1], n = 2
Output: False
```

Note:

1. The input array won't violate no-adjacent-flowers rule.
2. The input array size is in the range of [1, 20000].
3. n is a non-negative integer which won't exceed the input array size.

自己代码:
$n \leq 3$ 时, 情况都列出来;
$n > 3$ 时, 那点若为首点 `i == 0 && A[i] == 0 && A[1] == 0`,则count+1,且将A[0]=1,否则那点若为尾点`i == (n - 1) && A[i] == 0 && A[i - 1] == 0`,则count+1,且将A[n-1]=1,否则的话,这个点既不是首也不是尾巴,就可以放心的看这个点和前一个点以及后一个点是否全为0,`A[i] == 0 && A[i - 1] == 0 && A[i + 1] == 0`,若是则count+1,且A[i]=1.
我这个逻辑稍显复杂,较难控制.
编写这个用了多久怎么能和你说呢!
我用了足足1分钟! ^^
$O(n)$ time, $O(1)$ extra space.

```
bool canPlaceFlowers(vector<int>& A, int k) {
    int i, count = 0, n = A.size();
    if (n <= 3) {
        if (n == 1 && A[0] == 0) count = 1;
        if (n == 1 && A[0] == 1) count = 0;

        if (n == 2 && A[0] == 0 && A[1] == 0) count = 1;
        if ((n == 2 && A[0] == 1 && A[1] == 0) || (n == 2 && A[0] == 0 && A
] == 1)) count = 0;

        if (n == 3 && A[0] == 0 && A[1] == 0 && A[2] == 0) count = 2;
        if (n == 3 && A[0] == 0 && A[1] == 0 && A[2] == 1) count = 1;
        if (n == 3 && A[0] == 1 && A[1] == 0 && A[2] == 0) count = 1;

        return k <= count ? true : false;
    }
    else {
        for (i = 0; i < n; i++) {
            if (i == 0 && A[i] == 0 && A[1] == 0) {A[i] = 1; count++;}
            else if (i == (n - 1) && A[i] == 0 && A[i - 1] == 0) {A[i] = 1;
ount++;}
            else if (A[i] == 0 && A[i - 1] == 0 && A[i + 1] == 0) {A[i] = 1
; count++;}
        }
        return k <= count ? true : false;
    }
}
```

再看看人家的,妈的过分吧,是人脑袋吗,还有点正常人的思维吗,想杀人!
$O(n)$ time, $O(1)$ extra space.

```
bool canPlaceFlowers(vector<int>& A, int n) {
    A.insert(A.begin(), 0);
    A.push_back(0);
    for (int i = 1; i < A.size() - 1; ++i){
        if (A[i - 1] + A[i] + A[i + 1] == 0){
            --n;
            ++i;
        }
    }
    return n <= 0;
}
```

153. Find Minimum in Rotated Sorted Array

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

Find the minimum element.

You may assume no duplicate exists in the array.

代码如下:
二分查找变形.

> 重要思考: `5 6 7 0 1 2 3`, 为什么当 A[mid] <= A[hi] 时, 是 hi = mid,而不是 hi = mid - 1 ?
> 1 < 3, 1有可能是要找的值, 故而 hi = mid
> 7 > 3, 7必然不是要找的, 而mid +1 处才有可能是要找的,故而 lo = mid + 1

$O(logn)$ time, $O(1)$ extra space.

```
// Binary search
int findMin(vector<int>& A) {
    int n = A.size(), lo = 0, hi = n - 1, mid;

    while (lo < hi) {
        if (A[0] < A[n - 1]) return A[0];

        mid = (lo + hi) / 2;
        if (A[mid] > A[hi]) lo = mid + 1;
        else hi = mid;
        // 重要思考
    }
    return A[lo];
}
```

154. Find Minimum in Rotated Sorted Array II

> Follow up for "153. Find Minimum in Rotated Sorted Array":
> What if **duplicates** are allowed?
> Would this affect the run-time complexity? How and why?

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

Example 1:

`0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`.

Example 2:

`1 3 3 3` might become `3 3 1 3`.

Find the minimum element.

解题重点:
仍然考虑 `5 6 7 0 1 2 3`, 再结合 `3 3 1 3` 或 `3 1 3 3` 来思考.

> 重要思考: `5 6 7 0 1 2 3`, 为什么当 A[mid] <= A[hi] 时, 是 hi = mid,而不是 hi = mid - 1 ?
> 1 < 3, 1有可能是要找的值, 故而 hi = mid
> 7 > 3, 7必然不是要找的, 而mid +1 处才有可能是要找的,故而 lo = mid + 1
> 本题再结合 `3 3 1 3` 或 `3 1 3 3` 来思考.
> 当 A[mid] == A[hi] 时候, hi 所指的 3 就没用了, 往前挪挪, hi--

与 153题想比, 当A[mid] == A[hi] 时候, hi--就行了.
$O(logn)$ time, $O(1)$ extra space.

```
//思考时最好仍基于 5 6 7 0 1 2 3
int findMin(vector<int>& A) {
    int lo = 0, hi = A.size() - 1, mid;
    while (lo < hi) {
        mid = (lo + hi) / 2;
        if (A[mid] > A[hi]) lo = mid + 1;
        else if (A[mid] < A[hi]) hi = mid;
        else hi--; // 此时 A[mid] == A[hi]
    }
    return A[lo]; //此时 lo == hi
}
```

4. Median of Two Sorted Arrays(不会,ＴＯＤＯ)

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be $O(log(m+n))$.

Example 1:

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

11. Container With Most Water

给一个非负数组A,元素们是A[i], i=0,...,n-1. 索引i代表x坐标,值A[i]表示高度.索引i与A[i],既(i, A[i])表示了垂直于x轴的线段. 请找出两条线段,与x轴组成的桶最多能装多少水?
人家想法:
先从底最宽搜起．两个边找最矮的移动，计算容积，保留最大值，循环条件 left < right.
自己代码:
$O(n)$ time, $O(1)$ space.

```
int maxArea(vector<int>& A) {
    int l = 0, r = A.size() - 1, water = 0, h;
    while (l < r) {
        h = min(A[l], A[r]);
        water = max(water, h * (r - l));
        while (A[l] <= h && l < r)
            l++;
        while (A[r] <= h && l < r)
            r--;
    }
    return water;
}
```