

Arrays

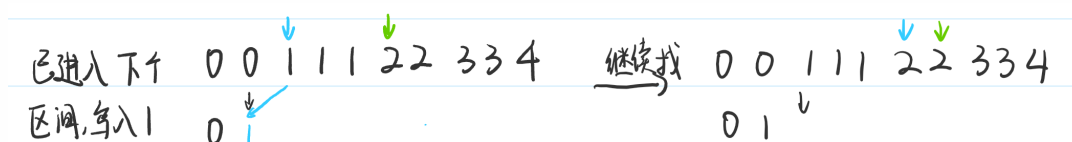
Two Pointers:

26. Remove Duplicates From Sorted Array

Two Pointers; In-place Modification

解法一: Three Pointers

一个pointer从 $0 \sim \text{nums.length}$ 进行写入，两个pointer在前方不断寻找一个数字区间，当找到新区间或走到终点时让第一个pointer将旧区间的值写入。



解法二: Two Pointers

可将此解法化简为两个pointers，一个读入一个写入。因为负责写入的pointer已经存储了数字，读入的pointer可以通过这个存储确定是否已经进入了新的数字区间，从而让第一个pointer继续写入。

27. Remove Element

Two Pointers; In-place Modification

解法一: 死算

用循环进行数字移位

解法二: Two Pointers

时间复杂度: $O(n)$; 空间复杂度: $O(1)$

读入pointer碰到val则跳过，否则让写入pointer写入该值。

88. Merge Sorted Array

Two Pointers

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

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

解法一：死算

用循环进行数字移位

解法二：Three Pointers

参考

2个pointer分别在两个数组的前方**读取**，一个pointer在num1里进行**写入**，每次写入时判断，先写小的那个。

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n)
    {
        int cur1 = m - 1;
        int cur2 = n - 1;
        int curf = m + n - 1;
        while (cur1 >= 0 && cur2 >= 0)
        {
            if (nums1[cur1] < nums2[cur2])
            {
                nums1[curf] = nums2[cur2];
                curf--;
                cur2--;
            }
            else
            {
                nums1[curf] = nums1[cur1];
                curf--;
                cur1--;
            }
        }
        while (cur1 >= 0)
            nums1[curf--] = nums1[cur1--];
        while (cur2 >= 0)
```

```
        nums1[curf--] = nums2[cur2--];  
    }  
}
```

283. Move Zeros

Two Pointers

解法一： Two Pointers

两个Pointers分别进行读和写，当读到非零数则让写入Pointer写入。最后将后面所有数字填补为0即可。

905. Sort Array by Parity

Two Pointers; Sort

解法一： Two Pointers & Swap

两个指针，一个读奇数一个读偶数。奇数指针尽可能地保持在偶数指针之前。奇数指针找到最近的奇数，偶数指针找到最近的在奇数指针之后的偶数。如果偶数指针已经超过了数组边界，则说明不需要再对数组做任何改动，返回当前数组即可。否则将奇数和偶数指针所指的两个数交换，并继续重复以上步骤。

解法二： Customized Sort

参考

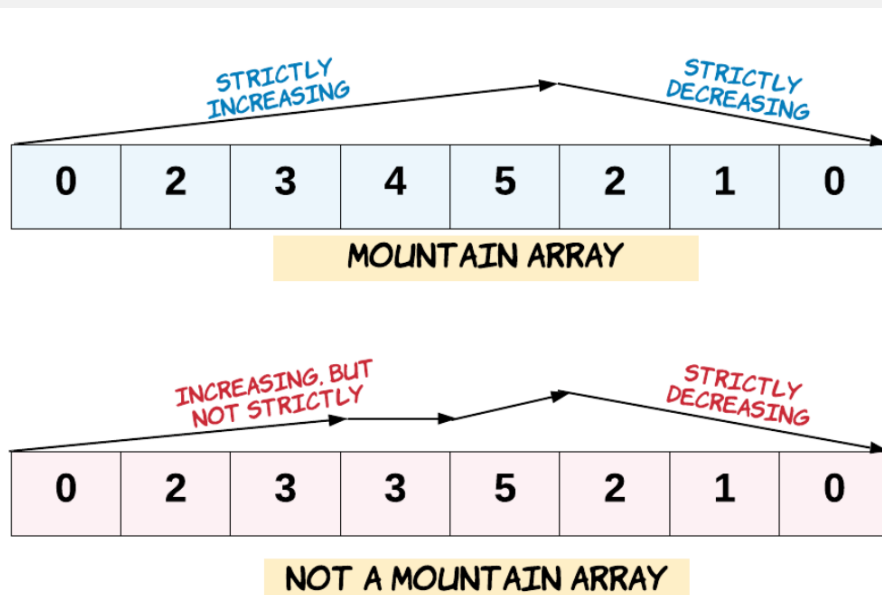
建立Integer数组，使用Java自带Customized Sort完成排序，最后赋值回A，返回A。

941. Valid Mountain Array

Two Pointers

Given an array A of integers, return true if and only if it is a valid mountain array. Recall that A is a mountain array if and only if:

1. $A.length \geq 3$
2. There exists some i with $0 < i < A.length - 1$ such that: $A[0] < A[1] < \dots < A[i-1] < A[i] > A[i+1] > \dots > A[A.length - 1]$



解法一：循环遍历

直接循环遍历整个数组，记录上升或下降的改变。当且仅当数组先升再降且改变一次升降情况时数组符合条件。比较简单，要判断的情况有点多。

解法二：Two Pointers

参考

设置两个pointer，一个从后往前，到下一位数开始变小时停止，一个从前往后，到下一位数开始变小时停止。如果两个指针相遇则说明数组符合条件

```
class Solution
{
    public boolean validMountainArray(int A[])
    {
        int j = A.length - 1;
        int i = 0;
        while (j - 1 >= 0 && A[j - 1] > A[j])
            j--;
        while (i + 1 < A.length && A[i + 1] > A[i])
            i++;
        if (i == j && i != 0 && i != A.length - 1)
            return true;
    }
}
```

```

        return false;
    }
}

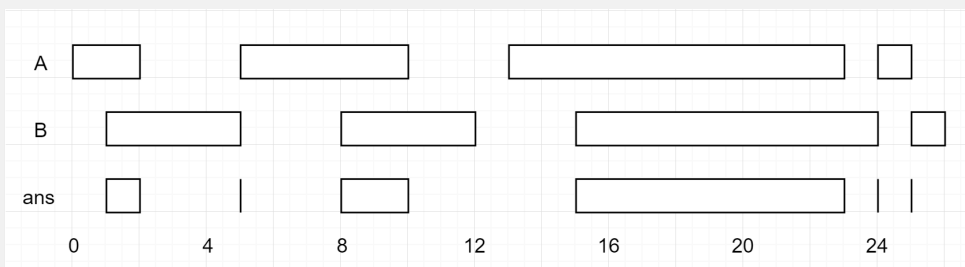
```

986. Interval List Intersections

Two Pointers

Given two lists of closed intervals, each list of intervals is pairwise disjoint and in sorted order.

Return the intersection of these two interval lists.



解法一：Two Pointers 分类判断

时间复杂度: $O(m + n)$; 空间复杂度: $O(m + n)$

两个指针分别在A数组和B数组的某个Interval上，首先判断A指针在前还是B指针在前。其次判断两个Interval是否有交集。在前的Interval的末端需要比在后的Interval的前端靠后才能有交集。有交集之后还需要判断靠后的Interval的末端在前还是靠前的Interval末端在前。判断完成之后把对应的交集放进答案集里，然后把靠前的指针后移。



解法二：Two Pointers 精简判断

时间复杂度: $O(m + n)$; 空间复杂度: $O(m + n)$

参考

判断的过程可以简化，首先取得两个Interval的**前端中靠后的一个**，再取得两个Interval**后端之中靠前的一个**，判断两者是否相交。如果相交前端结果应该小于等于后端结果，而这个前后端就是交集的前后端。

之后把靠前的指针后移。

```
class Solution {
public:
    vector<vector<int>>
    intervalIntersection(vector<vector<int>>& A,
        vector<vector<int>>& B) {
        int i = 0;
        int j = 0;
        vector<vector<int>> ans;
        while (i < A.size() && j < B.size())
        {
            int st = max(A[i][0], B[j][0]);
            int ed = min(A[i][1], B[j][1]);
            if (st <= ed)
                ans.push_back(vector<int> {st, ed});
            if (A[i][1] <= B[j][1])
                i++;
            else
                j++;
        }
        return ans;
    }
};
```

1089. Duplicate Zeros

Two Pointers

解法一：死算

用循环进行数字移位

解法二：Two Pointers

参考

通过遍历查找数组中0的数量来确定数字需要移动多少位。**从后往前进行写入覆盖。**

假定数组可以无限长，则写入pointer从数组的理论最长位置（加上复制的0之后）开始，读入pointer从真实长度位置开始，一旦遇到0则让写入pointer多写一位。

写入pointer一直在移动，但是**仅当进入数组真实长度位置后才开始正式写入。**

```
class Solution {
public:
    void duplicateZeros(vector<int>& A) {
        int n = A.size(), j = n + count(A.begin(), A.end(), 0);
        for (int i = n - 1; i >= 0; --i) {
            if (--j < n)
                A[j] = A[i];
            if (A[i] == 0 && --j < n)
                A[j] = 0;
        }
    }
};
```

1471. The k Strongest Values in an Array

Two Pointers; Sort

解法一：排序 + Two Pointers处理

首先将数组排序，找到中位数（注意题目定义的中位数和正常的定义不同），然后从两头开始的数字一定是和中位数差值最大的，让两个Pointers从两头分别开始，像Merge Sorted Arrays一样进行比较，取得前k个差值最大的数字。当差值一样的时候，因为数组已经经过排序，所以肯定是使用从尾端开始的Pointer指向的数字。

简单遍历:

1. Two Sums

HashMap

解法一：二重循环遍历查找

时间复杂度: $O(n^2)$; 空间复杂度: $O(1)$

解法二：HashMap保存Complement值

时间复杂度: $O(n)$; 空间复杂度: $O(n)$

448. Find All Numbers Disappeared in an Array

HashMap

解法一：用加法加上不可能出现的数做标记

时间复杂度: $O(n)$; 空间复杂度: $O(1)$

一道很有意思的题目，对时间和空间复杂度都有要求。因为题目给定了数字的范围，因此可以用特殊的方法来做**标记**。因为每个数都小于等于 n ，所以每次遇到一个数，就给数组对应这个数下标的数加上 n 。最后再遍历一遍数组，对每个数 $\text{div } n$ ，如果得到0或者这个数是 n 就说明这个数没有在数组里出现过。

解法二：用负数做标记

参考 时间复杂度: $O(n)$; 空间复杂度: $O(1)$

遇到一个数就把数组对应这个数 - 1 的为下表的数变成负数来做标记，最后遍历，那些没有变成负数的下标 + 1 即为没有出现过的数。

1299. Replace Elements with Greatest Element on Right Side

解法一：从后往前循环遍历

参考

时间复杂度: $O(n)$; 空间复杂度: $O(1)$

一边记录遇到的最大值一边进行替换，因此仅需单重循环。因为每次需要的信息都只包括这个数字之后的数字，且不需要这个数字之前的数字，所以可以从后往前进行处理，也不需要回来再看之前的数据。

Sort:

75. Sort Colors

Sort; Three Pointers

Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

解法一：Three Pointers 交换排序

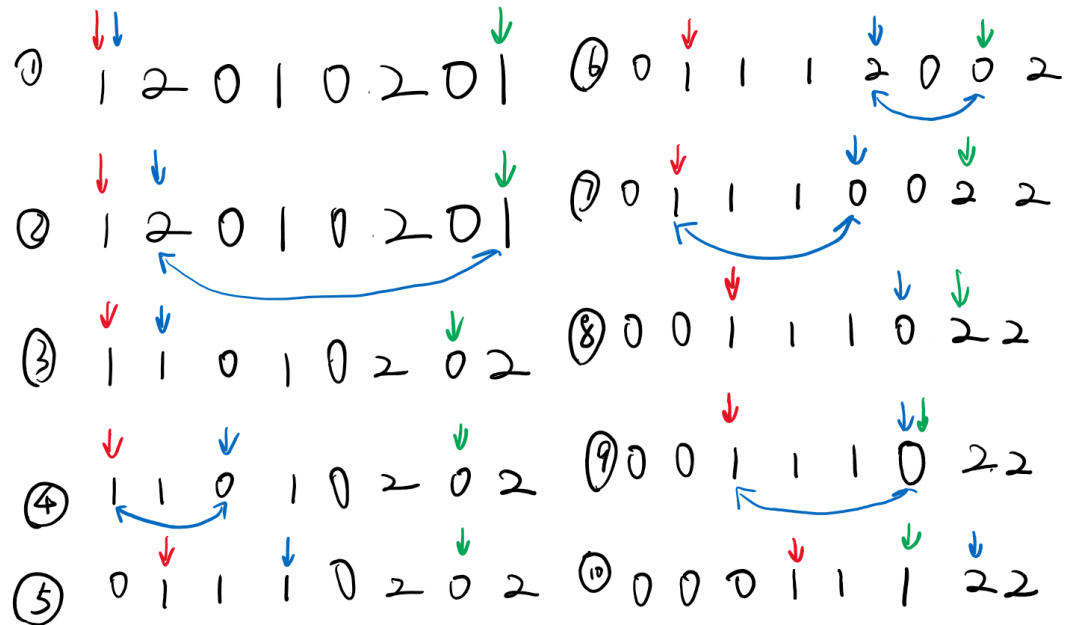
参考

设置三个Pointers, low, mid和high, low和mid从头开始, high从尾开始。mid遇到的数字可能有三种情况:

1. 遇到2, 则和high交换, 这个时候high--, high之后的所有数字都是2 (如第二步)
2. 遇到0, 则和low交换, 这个时候low++, low之前所有的数字都是0, 同时mid++, 这是因为我们确定mid交换过来的数字只可能是1, 因为2**已经在mid来的路上被处理过了** (如第四步)
3. 遇到1, 直接mid++

这样最后当mid超过high的时候数列就处理完了, 注意当mid和high相遇的时候还需要一步操作, 因为我们只知道high之后的数字都是2, 而不知道当前

数字的情况，所以还需要一次判断。



```
class Solution {
    public void sortColors(int[] nums) {
        int low = 0;
        int mid = 0;
        int high = nums.length - 1;
        while (mid <= high)
        {
            if (nums[mid] == 0)
            {
                nums[mid] = nums[low];
                nums[low] = 0;
                low++;
                mid++;
            }
            else if (nums[mid] == 2)
            {
                nums[mid] = nums[high];
                nums[high] = 2;
                high--;
            }
            else
                mid++;
        }
    }
}
```

136. Single Number

Sort; HashMap

169. Majority Element

Sort; HashMap

973. K Closest Points to Origin

Sort; Map

解法一：Map排序

将每个点距离原点的距离计算后作为Map的Key，对应的数值是距离为这个Key的所有点的index组成的vector。因为Map是自动排序的，最后只要按顺序直接输出前K个点即可。

解法二：Distance排序后输出

计算所有的Distance之后排序得到第K位的点的距离。之后再循环遍历所有的点把Distance小于等于这个距离的所有点输出。

977. Squares of a Sorted Array

Sort; Two Pointers

解法一：直接平方后重新排序

时间复杂度： $O(n\log n)$ ； 空间复杂度： $O(1)$

解法二：Two Pointers，存储进新数组

参考

时间复杂度： $O(n)$ ； 空间复杂度： $O(n)$

1051. Height Checker

Sort

1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

Sort; Find biggest num

解法一：排序+找最大值

首先排序，然后对于横向和纵向数组，分别找最大的间距，包括和0以及和最大边界的间距。最后两个最大值相乘。**相乘时注意分别对1000000007取余再对结果取余**，否则会溢出。

1481. Least Number of Unique Integers after K Removals

Sort; HashMap; Greedy

解法一：Bucket Sort从较低出现频率的数字开始去除

因为需要最后Unique的Number最少，所以可以从出现频率最低的数字开始一个一个去除，直到不能再去除位置。可以用HashMap来保存每个数字出现的频率，再把所有频率排序（用C++的Map自动排序），然后从频率低到高的一个一个减少数字，最后看还剩多少特殊的数字即可。

解法二：直接排序

参考

直接在记录频率的过程中把某出现频率对应的数字存进一个数组中，将数组排序，最后从低到高进行处理即可。

其他:

9. Palindrome Number

344. Reverse String

387. First Unique Character in a String

HashMap

414. Third Manimum Number

485. Max Consecutive Ones

Find Biggest Num

1295. Find Numbers with Even Number of Digits

Easy Counting

1346. Check if N and its Duplicate Exists

HashMap

1464. Maximum Product of Two Elements in an Array

1470. Shuffle the Array

1480. Running Sum of 1d Array