Arrays

简单遍历&HashMap：

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)

1229. Replace Elements with Greatest Element on Right Side

解法一：从后往前循环遍历

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

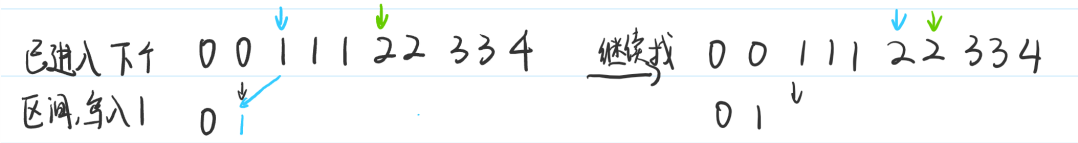
Two Pointers：

26. Remove Duplicates From Sorted Array

Two Pointers; In-place Modification

解法一：Three Pointers

一个pointer从 0 ~ nums.length 进行写入，两个pointer在前方不短寻找一个一个数字区间，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

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--];
    }
}
```

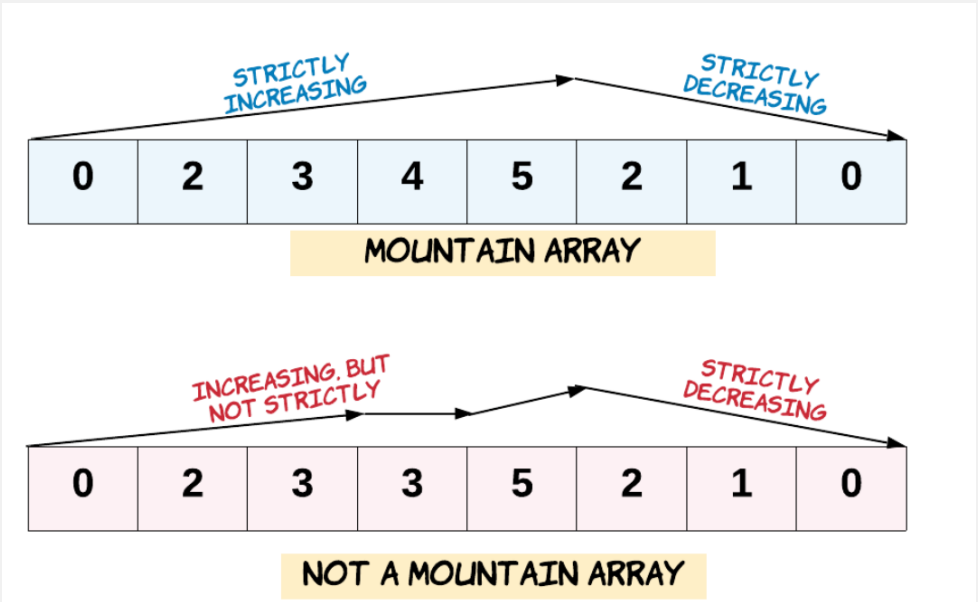
---

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 >= 3
2. There exists some i with 0 < i < A.length - 1 such that: A[0] < A[1] < ... A[i-1] < A[i] A[i] > A[i+1] > ... > 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;
    }
}
```

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;
        }
    }
};
```

其他

169. Majority Element

Sort; HashMap

387. First Unique Character in a String

HashMap

1346. Check if N and its Duplicate Exists

HashMap