原地修改或者是O(n)复杂度解数组整数题

1.Leetcode007整数反转

给你一个 32 位的有符号整数 `x` ，返回将 `x` 中的数字部分反转后的结果。

如果反转后整数超过 32 位的有符号整数的范围 `[−231, 231 − 1]` ，就返回 0。

假设环境不允许存储 64 位整数（有符号或无符号）。

示例 1：

```
输入：x = 123
输出：321
```

示例 2：

```
输入：x = -123
输出：-321
```

示例 3：

```
输入：x = 120
输出：21
```

> 对其反转

```
class Solution {
    public int reverse(int x) {
        int res = 0;

        while(x!=0){
            // 判断是否溢出res
            if(((res*10)/10)!=res){
                res = 0;
                return res;
            }
            res = res*10+x%10;
            x = x/10;
        }
        return res;
    }
}
```

2.Leecode136只出现一次的数字

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗?

示例 1:

输入: [2,2,1]
输出: 1
示例 2:

输入: [4,1,2,1,2]
输出: 4

> 只出现一次，用异或运算符

```
class Solution {
    public int singleNumber(int[] nums) {
        int res = 0;
        for(int num:nums){
            res ^= num;
        }
        return res;
    }
}
```

3.Leetcode_Offer03数组中重复的数字

找出数组中重复的数字。

在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

示例 1：

输入：
[2, 3, 1, 0, 2, 5, 3]
输出：2 或 3

> 修改数组本身

```
class Solution {
    public int findRepeatNumber(int[] nums) {
        // 重复的数字
        int len = nums.length;
        for(int i=0;i<len;i++){
            while(i!=nums[i]){
                if(nums[i]==nums[nums[i]]){
                    return nums[i];
                }
                swap(nums,i,nums[i]);
            }
        }
        return -1;
    }
    // 交换
    public void swap(int[] nums,int i,int j){
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
```

```
}
```

4.Leetcode169多数元素

给定一个大小为 n 的数组，找到其中的多数元素。多数元素是指在数组中出现次数 大于 ⌊ n/2 ⌋ 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

示例 1：

输入：[3,2,3]
输出：3
示例 2：

输入：[2,2,1,1,1,2,2]
输出：2

> 诸侯争霸，即投票消耗法

```
class Solution {
    public int majorityElement(int[] nums) {
        int count = 1;
        int cancadiate = nums[0];
        for(int i=1;i<nums.length;i++){
            if(count==0){
                cancadiate = nums[i];
            }
            count += nums[i]==cancadiate?1:-1;
        }
        return cancadiate;
    }
}
```

5.Leetcode448找到所有数组中消失的数字

给定一个范围在 1 ≤ a[i] ≤ n (n = 数组大小) 的 整型数组，数组中的元素一些出现了两次，另一些只出现一次。

找到所有在 [1, n] 范围之间没有出现在数组中的数字。

您能在不使用额外空间且时间复杂度为O(n)的情况下完成这个任务吗? 你可以假定返回的数组不算在额外空间内。

示例:

输入:
[4,3,2,7,8,2,3,1]

输出:
[5,6]

> 1-n之间

```
class Solution {
    public List<Integer> findDisappearedNumbers(int[] nums) {
        // 对其排序
```

```
        for(int i=0;i<nums.length;i++){
            // 值不等则交换值
            while(nums[i]!=nums[nums[i]-1]){
                swap(nums,i,nums[i]-1);
            }
        }
        // 排序好了再次遍历
        List<Integer> res = new ArrayList<>();
        for(int i=0;i<nums.length;i++){
            if(i!=nums[i]-1){
                res.add(i+1);
            }
        }
        return res;
    }

    // 交换
    public void swap(int[] nums,int i,int j){
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```

6.Leetcode041缺失的第一个正数

给你一个未排序的整数数组 nums ， 请你找出其中没有出现的最小的正整数。

进阶：你可以实现时间复杂度为 O(n) 并且只使用常数级别额外空间的解决方案吗?

示例 1：

输入：nums = [1,2,0]
输出：3

示例 2：

输入：nums = [3,4,-1,1]
输出：2

示例 3：

输入：nums = [7,8,9,11,12]
输出：1

> 可用hashSet来辅助解题，之后对1-n进行遍历，看其在hashSet中是否存在

```
class Solution {
    public int firstMissingPositive(int[] nums) {
        // 最小正整数是1-n的
        int len = nums.length;
        //对其遍历
        for(int i=0;i<len;i++){
            // 比较值 // 但可能有的值不符合条件 就放着不动了
            while(nums[i]>0&&nums[i]<=len&&nums[i]!=nums[nums[i]-1]){
                // 交换
```

```
                swap(nums,i,nums[i]-1);
            }
        }

        // 对其再次遍历
        for(int i=0;i<len;i++){
            if(i!=nums[i]-1){
                return i+1;
            }
        }
        return len+1;
    }

    // 交换
    public void swap(int[] nums,int i,int j){
        int temp = nums[i];
        nums[i]  = nums[j];
        nums[j]  = temp;
    }
}
```