**[287. Find the Duplicate Number](https://leetcode.com/problems/find-the-duplicate-number/)**

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

There is only **one repeated number** in nums, return *this repeated number*.

You must solve the problem **without** modifying the array nums and uses only constant extra space.

**Example 1:**

**Input:** nums = [1,3,4,2,2]

**Output:** 2

**Example 2:**

**Input:** nums = [3,1,3,4,2]

**Output:** 3

**Example 3:**

**Input:** nums = [3,3,3,3,3]

**Output:** 3

**Constraints:**

* 1 <= n <= 105
* nums.length == n + 1
* 1 <= nums[i] <= n
* All the integers in nums appear only **once** except for **precisely one integer** which appears **two or more** times.

**Follow up:**

* How can we prove that at least one duplicate number must exist in nums?
* Can you solve the problem in linear runtime complexity?
* public class Solution {
* public int findDuplicate(int[] nums) {
* int slow = nums[0];
* int fast = nums[0];
* do {
* slow = nums[slow];
* fast = nums[nums[fast]];
* } while (slow != fast);
* slow = nums[0];
* while (slow != fast) {
* slow = nums[slow];
* fast = nums[fast];
* }
* return slow;
* }
* }

[**75. Sort Colors**](https://leetcode.com/problems/sort-colors/)

Given an array nums with n objects colored red, white, or blue, sort them [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm)so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

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

You must solve this problem without using the library's sort function.

**Example 1:**

**Input:** nums = [2,0,2,1,1,0]

**Output:** [0,0,1,1,2,2]

**Example 2:**

**Input:** nums = [2,0,1]

**Output:** [0,1,2]

**Constraints:**

* n == nums.length
* 1 <= n <= 300
* nums[i] is either 0, 1, or 2.

**Follow up:** Could you come up with a one-pass algorithm using only constant extra space?

class Solution {

    public void sortColors(int[] nums) {

        int left= 0;

        int right= nums.length-1;

        int ptr;

        int zero=0;

        int one=0;

        int two=0;

        for(int i=0;i<nums.length;i++){

            if(nums[i]==0) zero++;

            else if(nums[i]==1) one++;

            else two++;

        }

        ptr=zero;

        while(ptr<=right){

            if(nums[ptr]==0){

                int temp=nums[ptr];

                nums[ptr]=nums[left];

                nums[left]=temp;;

                left++;

            }

            else if(nums[ptr]==2){

                int temp=nums[ptr];

                nums[ptr]=nums[right];

                nums[right]=temp;

                right--;

            }

            else{

                ptr++;

            }

        }

    }

}

[**26. Remove Duplicates from Sorted Array**](https://leetcode.com/problems/remove-duplicates-from-sorted-array/)

Given an integer array nums sorted in **non-decreasing order**, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in*nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

* Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.
* Return k.

**Custom Judge:**

The judge will test your solution with the following code:

int[] nums = [...]; // Input array

int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;

for (int i = 0; i < k; i++) {

assert nums[i] == expectedNums[i];

}

If all assertions pass, then your solution will be **accepted**.

**Example 1:**

**Input:** nums = [1,1,2]

**Output:** 2, nums = [1,2,\_]

**Explanation:** Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Example 2:**

**Input:** nums = [0,0,1,1,1,2,2,3,3,4]

**Output:** 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

**Explanation:** Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Constraints:**

* 1 <= nums.length <= 3 \* 104
* -100 <= nums[i] <= 100
* nums is sorted in **non-decreasing** order.
* class Solution {
* public int removeDuplicates(int[] nums) {
* int j=1;
* for(int i=1;i<nums.length;i++){
* if(nums[i]!=nums[i-1]){
* nums[j]=nums[i];
* j++;
* }
* }
* return j;
* }

}