Given a sorted array *nums*, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each element appear only *once* and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array**[**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Example 1:**

Given *nums* = **[1,1,2]**,

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 returned length.

**Example 2:**

Given *nums* = **[0,0,1,1,1,2,2,3,3,4]**,

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

It doesn't matter what values are set beyond the returned length.

重点：

这题不让你创建额外的一个array，所以要在输入array做变化

用的是快慢指针的思想，慢指针代表着当前最大的数，快指针代表重复的这个数

一开始他们都指向index0

然后如果相同，光移动快指针，//重复

如果不同，代表着快指针已经到了一个新的更大的数，慢指针加1，快指针所指的数赋值，快指针+1

class Solution {

public int removeDuplicates(int[] nums) {

int fast=0;

int slow=0;

if(nums.length==0)

return 0;

else{

while(fast<nums.length)

if(nums[fast]==nums[slow])

fast++;

else{

slow++;

nums[slow]=nums[fast];

fast++;

}

}

return slow+1;

}

}

FOR LOOP VERSION

class Solution {

public int removeDuplicates(int[] nums) {

int slow=0;

if(nums.length==0)

return 0;

else{

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

if (nums[fast]!=nums[slow]){

slow++;

nums[slow]=nums[fast];

}

}

}

return slow+1;

}

}