**26. Remove Duplicates from Sorted Array**

**Question**

Given a sorted array, remove the duplicates in place such that each element appear only *once* and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,  
Given input array *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 new length.

**Quick Navigation**

* [Solution](https://leetcode.com/articles/remove-duplicates-sorted-array/#solution)
  + [Approach #1 (Two Pointers) [Accepted]](https://leetcode.com/articles/remove-duplicates-sorted-array/#approach-1-two-pointers-accepted)

**Solution**

**Approach #1 (Two Pointers) [Accepted]**

**Algorithm**

Since the array is already sorted, we can keep two pointers iii and jjj, where iii is the slow-runner while jjj is the fast-runner. As long as nums[i]=nums[j]nums[i] = nums[j]nums[i]=nums[j], we increment jjj to skip the duplicate.

When we encounter nums[j]≠nums[i]nums[j] \neq nums[i]nums[j]≠nums[i], the duplicate run has ended so we must copy its value to nums[i+1]nums[i + 1]nums[i+1]. iii is then incremented and we repeat the same process again until jjj reaches the end of array.

**public** **int** **removeDuplicates(int[]** nums**)** **{**

**if** **(**nums**.**length **==** 0**)** **return** 0**;**

**int** i **=** 0**;**

**for** **(int** j **=** 1**;** j **<** nums**.**length**;** j**++)** **{**

**if** **(**nums**[**j**]** **!=** nums**[**i**])** **{**

i**++;**

nums**[**i**]** **=** nums**[**j**];**

**}**

**}**

**return** i **+** 1**;**

**}**

**Complexity analysis**

* Time complextiy : O(n)O(n)O(n). Assume that nnn is the length of array. Each of iii and jjj traverses at most nnn steps.
* Space complexity : O(1)O(1)O(1).