Solution

Approach 1: Two Pointers

**Algorithm**

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

When we encounter *nums*[*j*] = *nums*[*i*], the duplicate run has ended so we must copy its value to  *nums*[*i*+1]. i*i* is then incremented and we repeat the same process again until j*j* 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*). Assume that n*n* is the length of array. Each of i*i* and j*j* traverses at most *n* steps.
* Space complexity : *O*(1).