Remove Duplicates from Sorted Array

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.

Solution 1

```
class Solution {
    public:
    int removeDuplicates(int A[], int n) {
        if(n < 2) return n;
        int id = 1;
        for(int i = 1; i < n; ++i)
            if(A[i] != A[i-1]) A[id++] = A[i];
        return id;
    }
};
```

written by liyangguang1988 original link here

Solution 2

```
int count = 0;
for(int i = 1; i < n; i++){
    if(A[i] == A[i-1]) count++;
    else A[i-count] = A[i];
}
return n-count;
```

written by jasusy original link here

Solution 3

I don't understand why people insist on using old-style indexed looping. I much prefer the enhanced / range-based loops, they make things much cleaner.

C++

```
int removeDuplicates(vector<int>& nums) {
    int i = 0;
    for (int n : nums)
        if (!i || n > nums[i-1])
            nums[i++] = n;
    return i;
}
```

And to not need the `!i` check in the loop:

```
int removeDuplicates(vector<int>& nums) {
    int i = !nums.empty();
    for (int n : nums)
        if (n > nums[i-1])
            nums[i++] = n;
    return i;
}
```

Java

```
public int removeDuplicates(int[] nums) {
    int i = 0;
    for (int n : nums)
        if (i == 0 || n > nums[i-1])
            nums[i++] = n;
    return i;
}
```

And to not need the `i == 0` check in the loop:

```
public int removeDuplicates(int[] nums) {
    int i = nums.length > 0 ? 1 : 0;
    for (int n : nums)
        if (n > nums[i-1])
            nums[i++] = n;
    return i;
}
```

written by StefanPochmann original link here