

Remove Element

Given an integer array nums and integer val remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val .

Consider the number of elements in nums which are not equal to val to be k , to get Accepted you need to do the following things:

change the array num such that the first k elements of nums contain the elements which are not equal to val . The remaining elements of nums are not important as well as the size of nums .

Return k .

Example:

Input: $\text{num} = [3, 2, 3]$ $\text{val} = 3$

O/p: $\text{num} = [2, 2, -, -]$

Explanation: your function should return $k = 2$ with the first two elements of num being 2. It does not matter what you leave beyond the returned k (hence they are underscores).

```

public int removeElement(int[] nums, int val) {
    int ind = 0;
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] != val) {
            nums[ind++] = nums[i];
        }
    }
    return ind;
}

```

33

nums = [3, 2, 2, 3] val = 3.

ind = 0.

i = 0 nums.length = 4.

0 < 4 ✓

nums[i] != val.

nums[0] = 3

ind = 0

3 = 3 ✗

i = 1 nums.length = 4

1 < 4 ✓

nums[i] != val

nums[1] = 2

2 = 3 ✗

nums[ind++] = nums[i];

ind = 1

i = 2

nums.length = 4

2 < 4 ✓

nums[i] != val

nums[2] = 2

2 = 2 ✓

nums[ind++] = nums[i];

ind = 2.

i = 3

3 < 4 ✓ nums.length = 4

nums[i] != val

nums[3] = 3

3 = 3 ✗

skip

At last ind
value is 2.

ind = 2


```
import java.util.Arrays;
```

```
public class RemoveElementExample{
```

```
    public static int removeElement(int[] nums,  
                                    int val){
```

```
        int index = 0;
```

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

```
            if (nums[i] != val){
```

```
                nums[index++] = nums[i];
```

```
            }
```

```
        }  
        return index;
```

```
    }  
    public static void main(String[] args){
```

```
        int[] nums = {3, 2, 2, 3};
```

```
        int val = 3;
```

```
        int newLength = removeElement(nums, val);
```

```
        System.out.println("new Length: " + newLength);
```

```
        System.out.println("Modified array: " +
```

```
            Arrays.toString(Arrays.copyOf(nums,  
                                           newLength)));
```

```
    }
```

Remove Duplicates from Sorted Array.

Given a sorted array `nums`, remove the duplicates in-place such that each element appears 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 with $O(1)$ extra memory.

Example 1:

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

Output: 2, `nums = [1,2]`

Explanation: 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.

Class Solution:

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

```
    if (nums.length == 0)
```

```
        return 0;
```

```
    int newIndex = 1;
```

```
    for (int p = 1; p < nums.length; p++) {
```

```
        if (nums[p-1] != nums[p]) {
```

```
            nums[newIndex] = nums[p];
```

```
            newIndex++;
```

```
        }
```

```
    }
```

```
    return newIndex;
```



```
public static void main (String[] args) {
```

```
    Solution sol = new Solution();
```

```
    int[] nums = {1, 1, 2, 2, 3, 3, 4, 5, 5};
```

use
[1, 1, 2]

```
    int newlength = sol.removeDuplicates(nums);
```

```
    System.out.println("Newlength: " + newlength);
```

```
    System.out.println("Modified array: ");
```

```
    for (int p = 0; p < newlength; p++) {
```

```
        System.out.print(num[p] + " ");
```

3
3
3

nums = [1, 1, 2]

nums.length = 3

0 == 0 X

newIndex = 1

p = 1 1 < nums.length

1 < 3 ✓

nums[p-1] != nums[p]

nums[0] != 1

1 == 1

1 != 1 X

p = 2

2 < 3 ✓

nums[2-1] != nums[2]

nums[1] != nums[2]

1 != 2 ✓

nums[newIndex] = nums[p];

→ [1]

2 at newIndex

$\boxed{\text{nextIndex} = 2} \rightarrow [1, 2]$

3 < 3X

o/p NewIndex: 2

$[1, 2]$

Given an integer array `nums` sorted in non-decreasing order, remove some duplicates in-place such that each unique element appears at most twice. The relative order of the elements should be kept the same.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array `nums`.

More formally, if there are k elements after removing the duplicates, then the first k elements of `nums` should hold the final result. It does not matter what you leave beyond the first k elements.

Return k after placing the final result in the first k slots of `nums`.

Example 1:

Input: num = [1, 1, 1, 2, 2, 3]

Output: 5, nums = [1, 1, 2, 2, 3, ...]

Explanation: your function should return $k=5$ with the first five elements of nums being 1, it does not matter what you leave beyond the returned k (hence they are underscores)

```
import java.util.Arrays;
```

```
class Solution {
```

```
    public static void main(String[] args) {
```

```
        int nums[] = {1, 1, 1, 2, 2, 3};
```

```
        int newLength = removeDuplicates(nums);
```

```
        System.out.println("New length" + newLength);
```

```
        System.out.println("Modified Array: " +
```

```
            Arrays.toString(Array.copyOf(nums, newLength)));
```

```
    }
```

```
    public static int removeDuplicates(int[] nums) {
```

```
        int p = 0;
```

```
        for (int n : nums) {
```

```
            if (p < 2 || n != nums[p-2]) {
```

```
                nums[p++] = n;
```

```
            }
```

```
        }
```

```
        return p;
```

```
    }
```

q=0

(Pnt n: num&)

n=1

0 < 2 || 1! = num[0-2]

~~1! = num[-2]~~

1! = 2 →

update

num[0++]

num[1] = n;

n → [1, -, -, -, -]

p=1

(Pnt n: num&)

n=2

1 < 2 || 1! || num[1-2]

~~2! || num[1]~~

2! || 1 →

update

num[p++]

num[1] = n;

n → [1, 1, -, -, -]

p=2

(Pnt n: num&)

n=1

2 < 2 || 1! || num[1-2]

~~1! || num[1]~~

1! || 1 →

update

num[p++]

skip this

q=2

(Pnt n: num&)

n=2

2 < 2 || 2! = num[2-2]

2! = 2

2! = 1 →

num[p++] = n

num[2++] = n

num[3] = n

2 = n

n → [1, 1, 2, -, -, -]

n=2

p=3

3 < 2 || 2! = num[3-2]

~~2! = num[1]~~

2! = 2

2! = 1 →

num[p++] = n

num[3++] = n

num[4] = n

n → [1, 1, 2, 2, -, -]

n=3

p=4

4 < 2 || 3! = num[4-2]

3! = num[2]

3! = 6

3! = 1 →

num[p++] = n

num[4++] = n

num[5] = n

n → [1, 1, 2, 2, 3, -]