**LeetCode Problems**

Easy

**1. Two Sum**

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

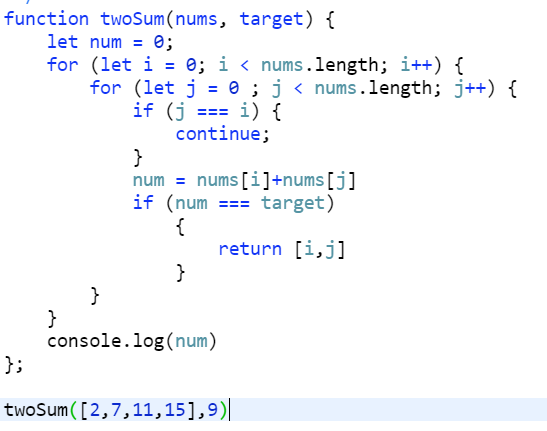
**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

**Solution:**



**2. Remove Element**

Given an array *nums* and a value *val*, remove all instances of that value [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) 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.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

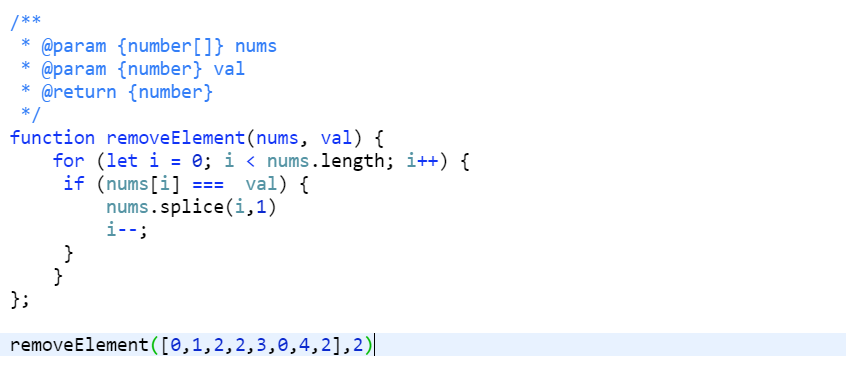
**Example 1:**

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

Your function should return length = **5**, with the first five elements of *nums* containing **0**, **1**, **3**, **0**, and **4**.

Note that the order of those five elements can be arbitrary.

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



**3. Remove Duplicates from Sorted Array**

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.

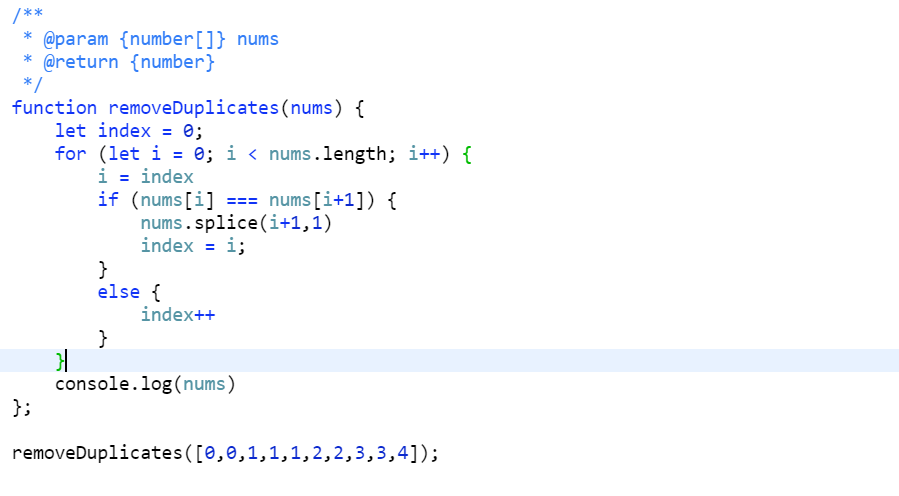
Do not allocate extra space for another array, you must do this by **modifying the input array**[**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Example 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.



**4. Search Insert Position**

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

**Example 1:**

**Input:** [1,3,5,6], 5

**Output:** 2

**Example 2:**

**Input:** [1,3,5,6], 2

**Output:** 1

**Example 3:**

**Input:** [1,3,5,6], 7

**Output:** 4

**Example 4:**

**Input:** [1,3,5,6], 0

**Output:** 0



**5. Plus One**

the question is treat an array as an integer. e.g. [1,2,3] is number 123 and plus one is 123+1 = 124, then convert this result number to a new array [1,2,4].  
The example should include **carry**. e.g. [2,9,9] plus one to result [3,0,0], or [9,9] plus one to [1,0,0].

**Example 1:**

**Input:** [1,2,3]

**Output:** [1,2,4]

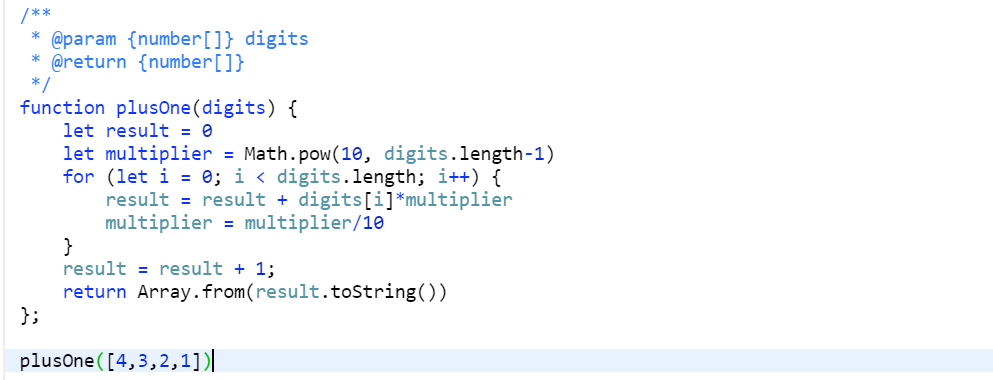
**Explanation:** The array represents the integer 123.

**Example 2:**

**Input:** [4,3,2,1]

**Output:** [4,3,2,2]

**Explanation:** The array represents the integer 4321.



**6. Maximum Subarray**

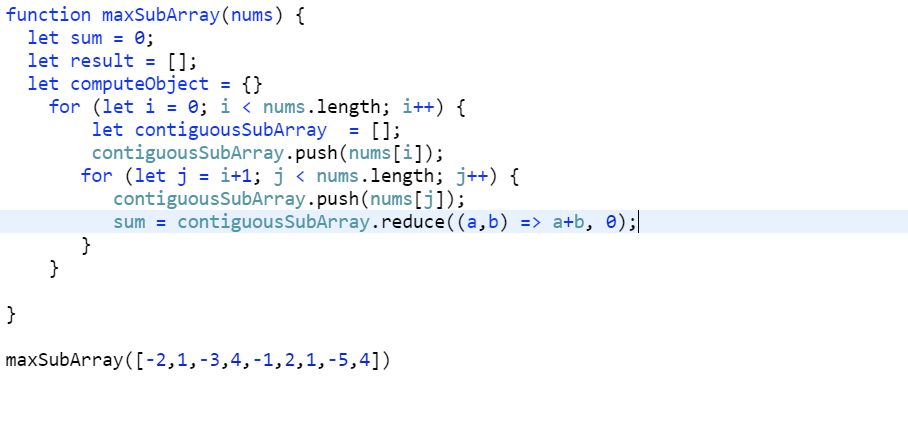
Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

**Example:**

**Input:** [-2,1,-3,4,-1,2,1,-5,4],

**Output:** 6

**Explanation:** [4,-1,2,1] has the largest sum = 6.



**7. Merge Sorted Array**

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

**Note:**

* The number of elements initialized in *nums1* and *nums2* are *m* and *n* respectively.
* You may assume that *nums1* has enough space (size that is **equal** to *m* + *n*) to hold additional elements from *nums2*.

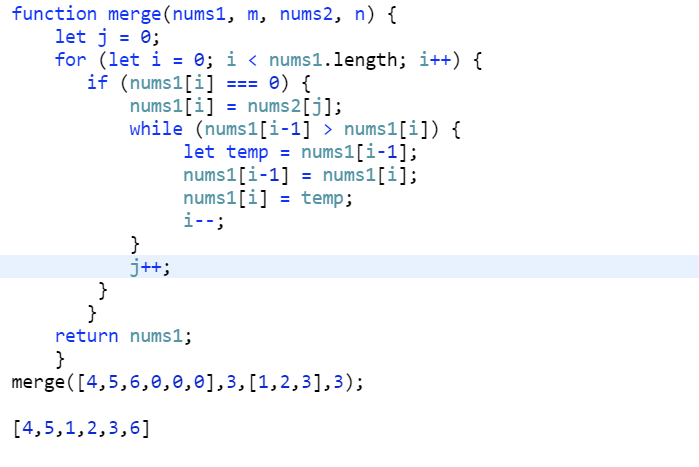
**Example:**

**Input:**

nums1 = [1,2,3,0,0,0], m = 3

nums2 = [2,5,6], n = 3

**Output:** [1,2,2,3,5,6]



**8. Pascal's Triangle**

Given a non-negative integer *numRows*, generate the first *numRows* of Pascal's triangle.

  
In Pascal's triangle, each number is the sum of the two numbers directly above it.

**Example:**

**Input:** 5

**Output:**

[

[1],

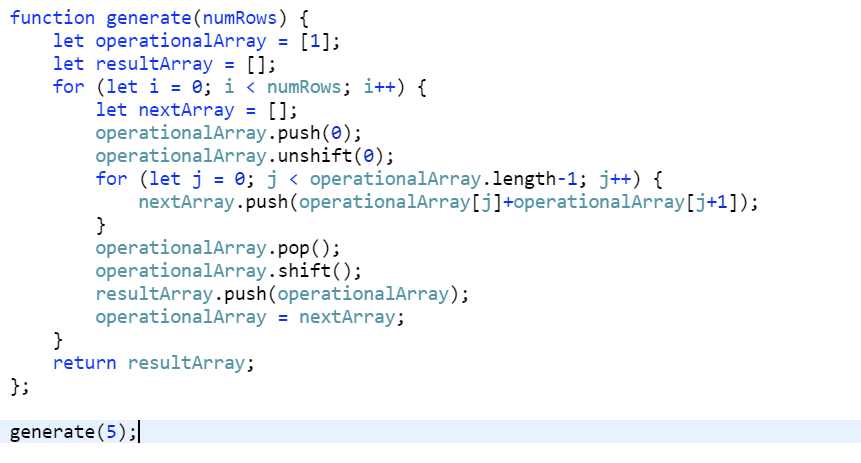
[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]



**9. Best Time to Buy and Sell Stock**

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

**Example 1:**

**Input:** [7,1,5,3,6,4]

**Output:** 5

**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

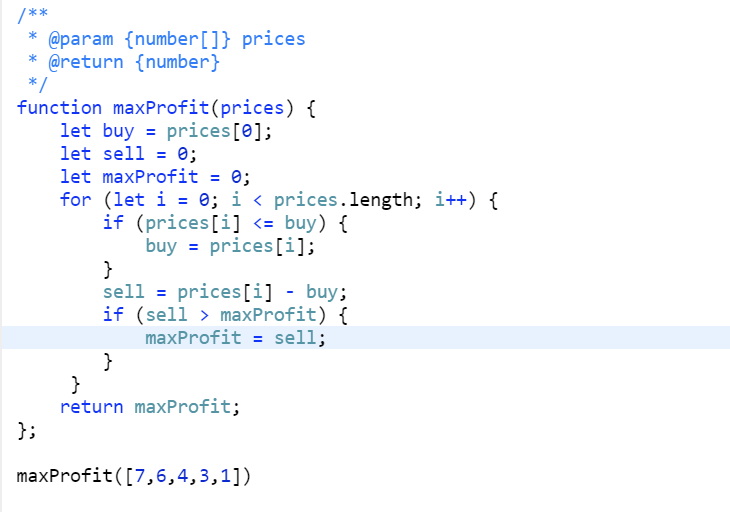
  Not 7-1 = 6, as selling price needs to be larger than buying price.

**Example 2:**

**Input:** [7,6,4,3,1]

**Output:** 0

**Explanation:** In this case, no transaction is done, i.e. max profit = 0.



**10. Two Sum II - Input array is sorted**

Given an array of integers that is already ***sorted in ascending order***, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

**Note:**

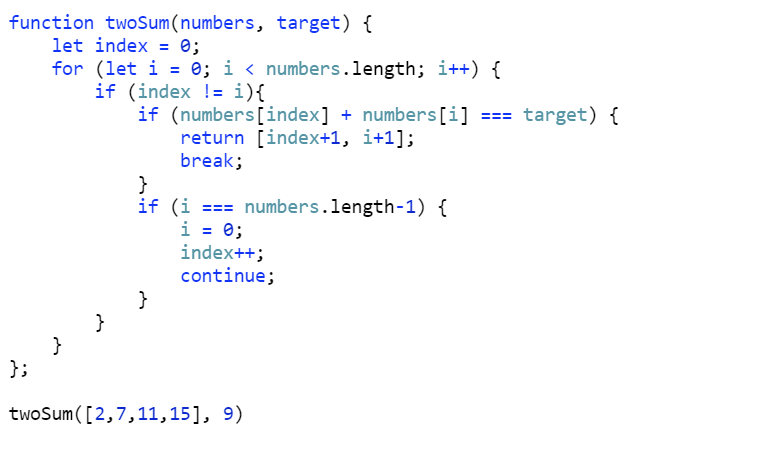
* Your returned answers (both index1 and index2) are not zero-based.
* You may assume that each input would have *exactly* one solution and you may not use the *same* element twice.

**Example:**

**Input:** numbers = [2,7,11,15], target = 9

**Output:** [1,2]

**Explanation:** The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.



**11. Majority Element**

Given an array of size *n*, find the majority element. The majority element is the element that appears **more than** ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

**Example 1:**

**Input:** [3,2,3]

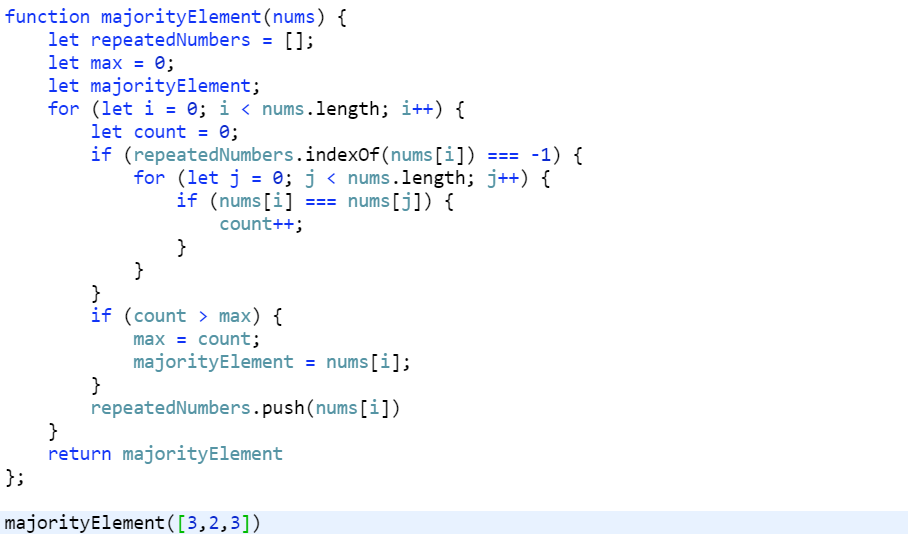
**Output:** 3

**Example 2:**

**Input:** [2,2,1,1,1,2,2]

**Output:** 2





**12. Rotate Array**

Given an array, rotate the array to the right by *k* steps, where *k* is non-negative.

**Follow up:**

* Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
* Could you do it in-place with O(1) extra space?

**Example 1:**

**Input:** nums = [1,2,3,4,5,6,7], k = 3

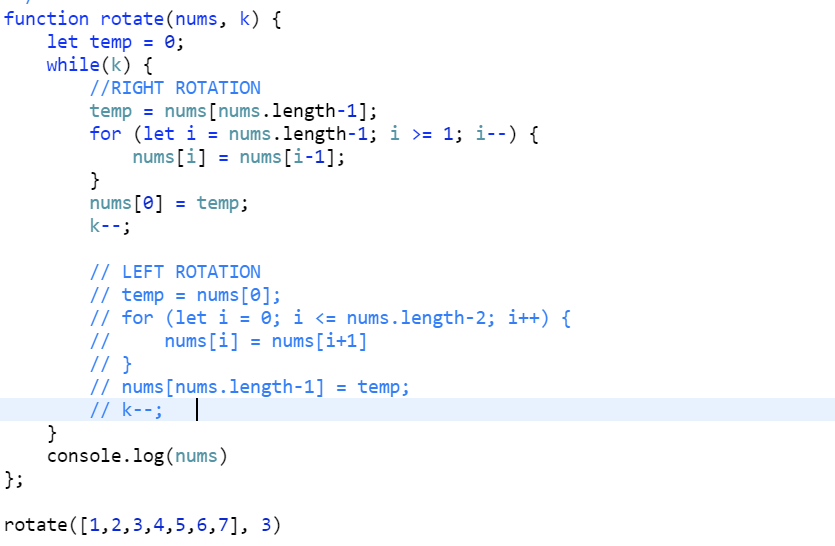
**Output:** [5,6,7,1,2,3,4]

**Explanation:**

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]



**13. Contains Duplicate:**

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

**Example 1:**

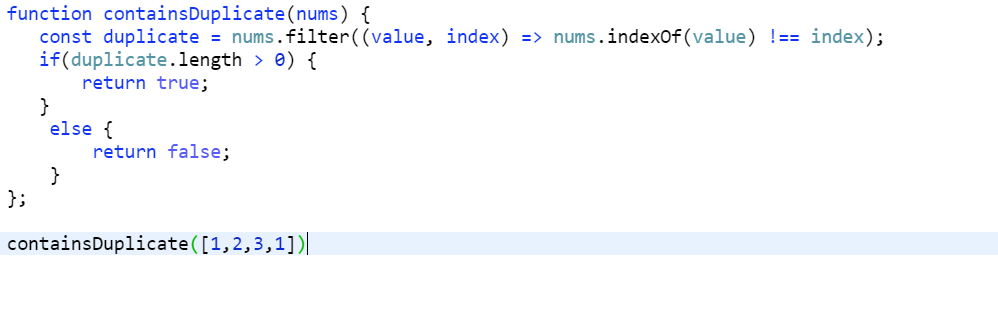
**Input:** [1,2,3,1]

**Output:** true

**Example 2:**

**Input:** [1,2,3,4]

**Output:** false



**14. Move Zeroes**

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

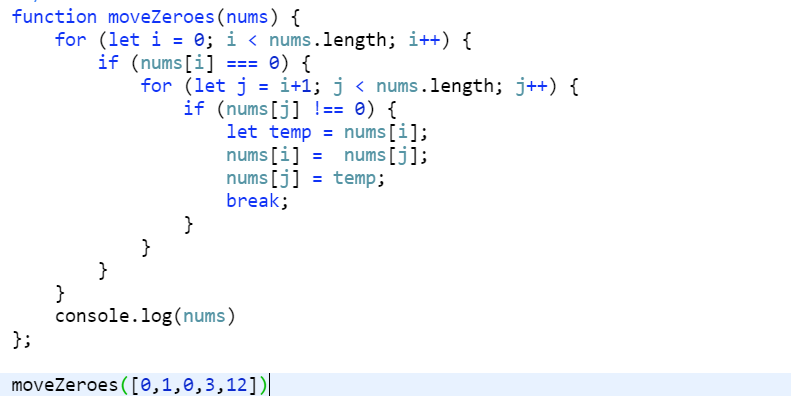
**Example:**

**Input:** [0,1,0,3,12]

**Output:** [1,3,12,0,0]

**Note**:

1. You must do this **in-place** without making a copy of the array.
2. Minimize the total number of operations.



**15. Third Maximum Number:**

Given a **non-empty** array of integers, return the **third**maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in O(n).

**Example 1:**

**Input:** [3, 2, 1]

**Output:** 1

**Explanation:** The third maximum is 1.

**Example 2:**

**Input:** [1, 2]

**Output:** 2

**Explanation:** The third maximum does not exist, so the maximum (2) is returned instead.

**Example 3:**

**Input:** [2, 2, 3, 1]

**Output:** 1

**Explanation:** Note that the third maximum here means the third maximum distinct number.

Both numbers with value 2 are both considered as second maximum.

