#1: Binary Search and Sorted Array

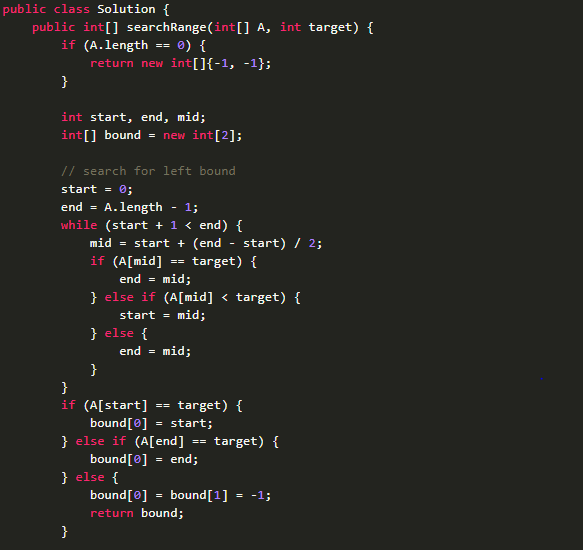
Problem and Answer

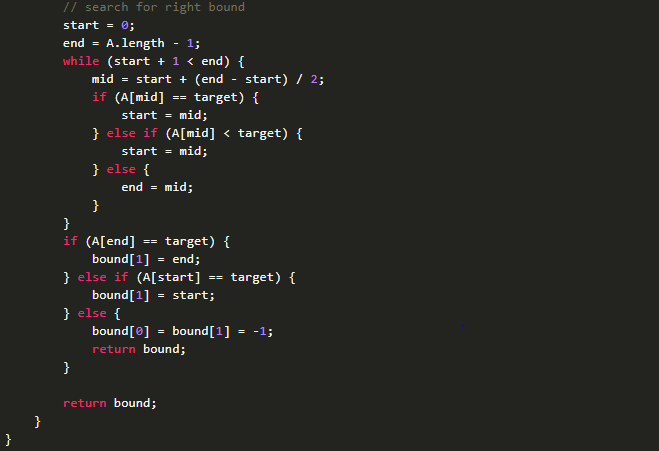
Search for a Range

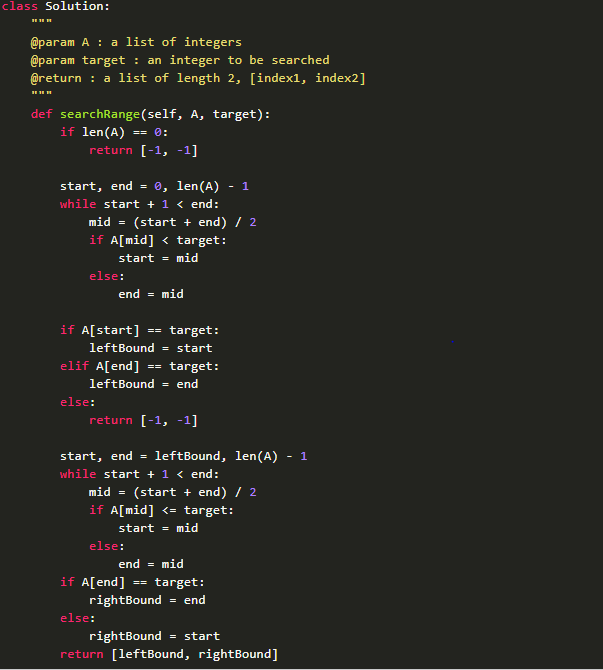
Given a sorted array of n integers, find the starting and ending position of a given target value.

If the target is not found in the array, return [-1, -1].

Given [5, 7, 7, 8, 8, 10] and target value 8, return [3, 4].







Search Insert Position

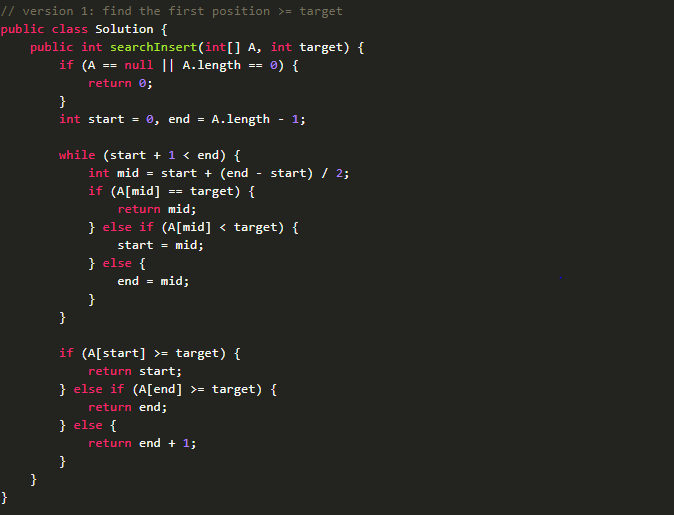
Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You may assume NO duplicates in the array.

[1,3,5,6], 5 → 2

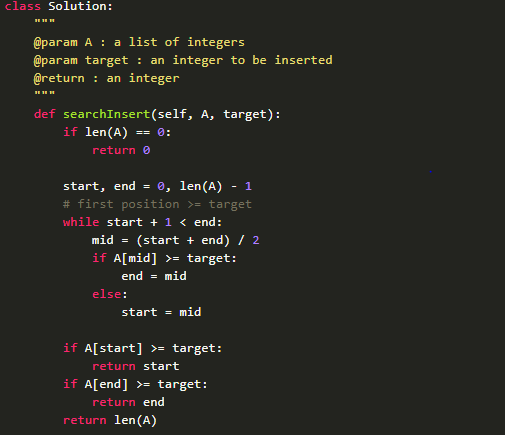
[1,3,5,6], 2 → 1

[1,3,5,6], 7 → 4

[1,3,5,6], 0 → 0







Search a 2D Matrix

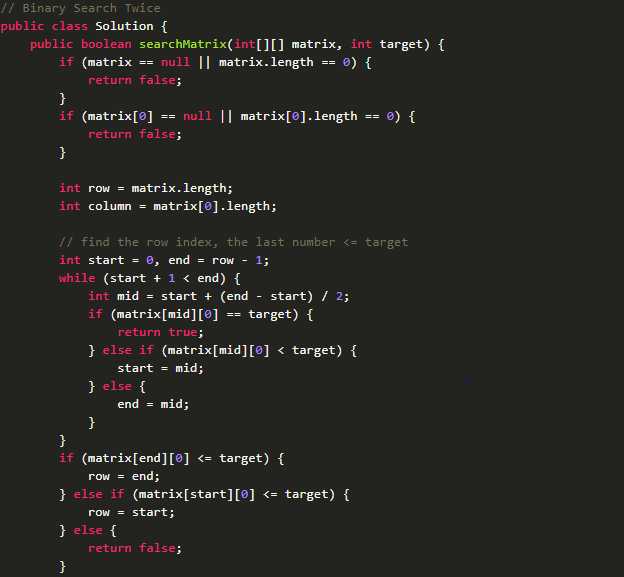
Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties: Integers in each row are sorted from left to right. The first integer of each row is greater than the last integer of the previous row.

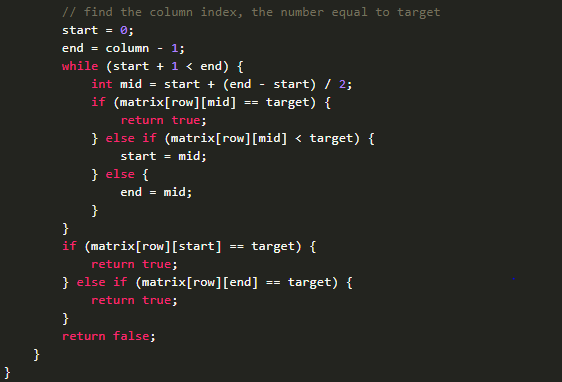
[ [1, 3, 5, 7],

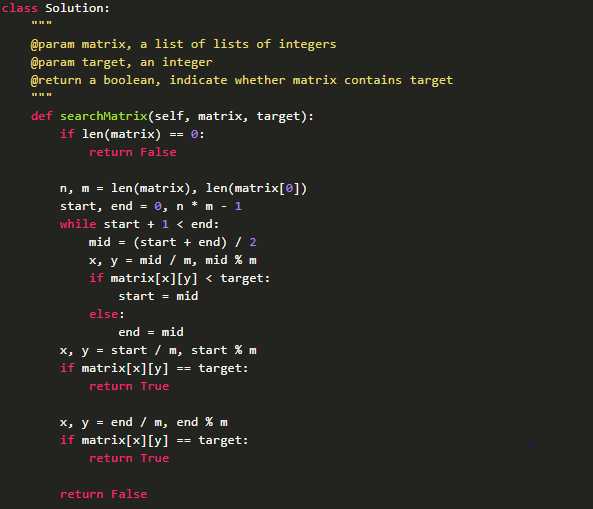
[10, 11, 16, 20],

[23, 30, 34, 50]]

Given target = 3, return true.



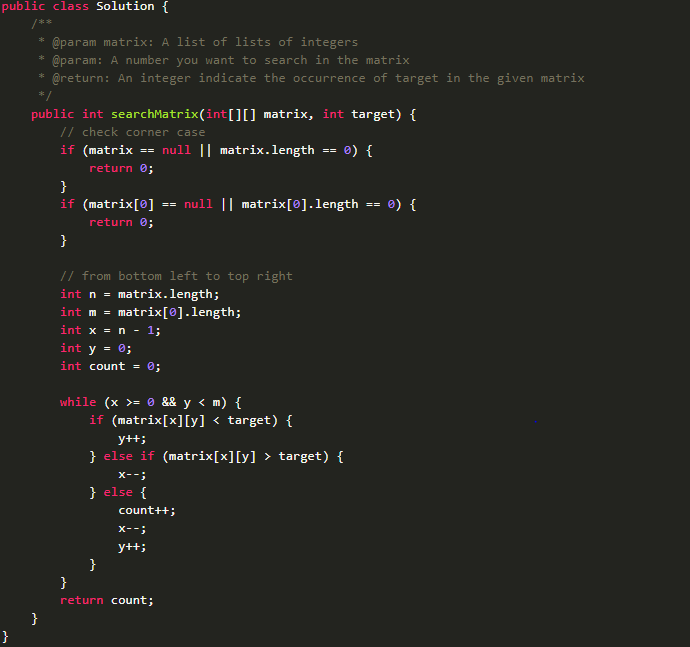


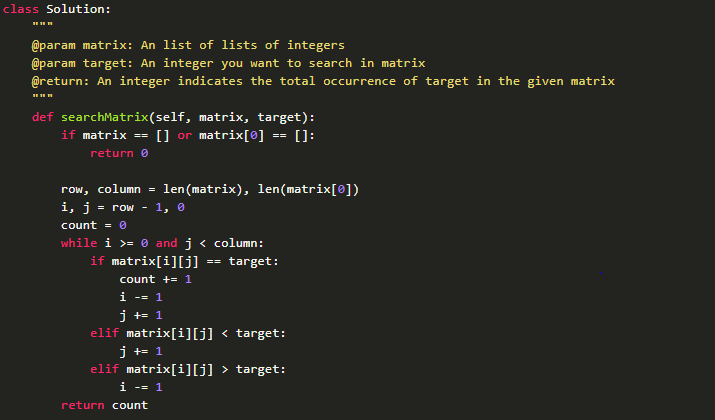


Search a 2D Matrix II

Write an efficient algorithm that searches for a value in an m x n matrix, return the occurrence of it.

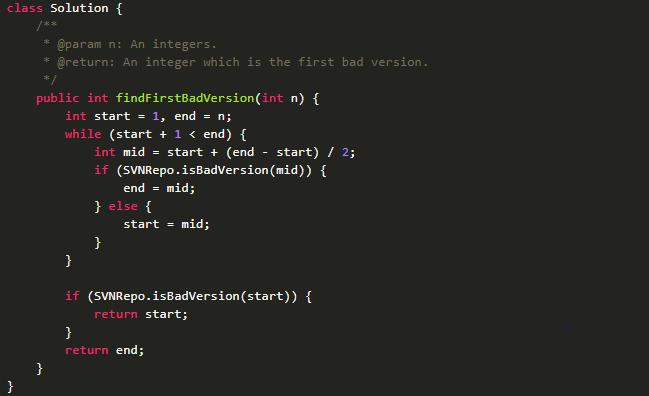
This matrix has the following properties: Integers in each row are sorted from left to right. Integers in each column are sorted from up to bottom. No duplicate integers in each row or column.

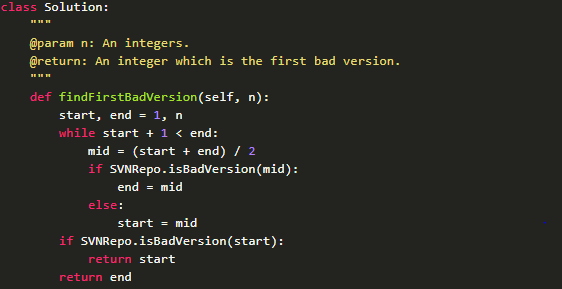




First Bad Version

The code base version is an integer start from 1 to n. One day, someone committed a bad version in the code case, so it caused this version and the following versions are all failed in the unit tests. Find the first bad version. You can call isBadVersion to help you determine which version is the first bad one. The details interface can be found in the code's annotation part.





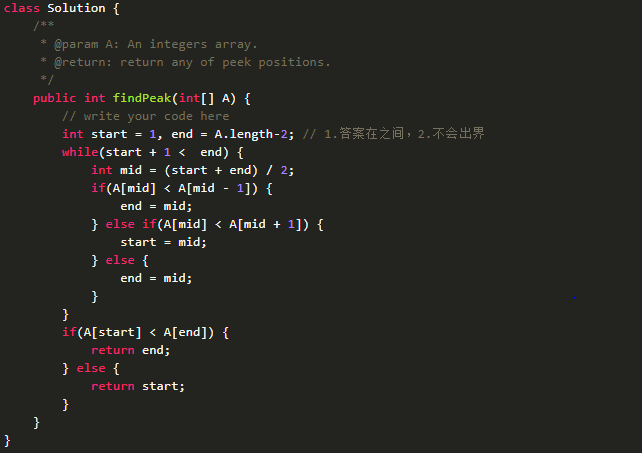
Find Peak Element

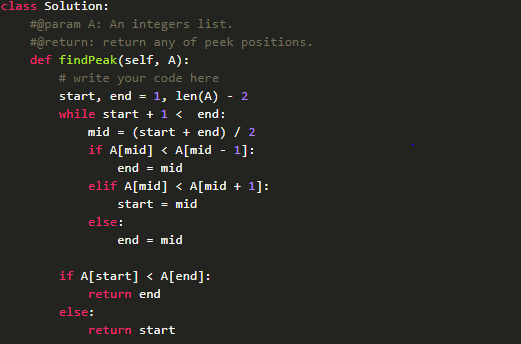
There is an integer array which has the following features:

The numbers in adjacent positions are different. A[0] < A[1] && A[A.length - 2] > A[A.length - 1].

We define a position P is a peak if: A[P] > A[P-1] && A[P] > A[P+1]

Find a peak element in this array. Return the index of the peak.





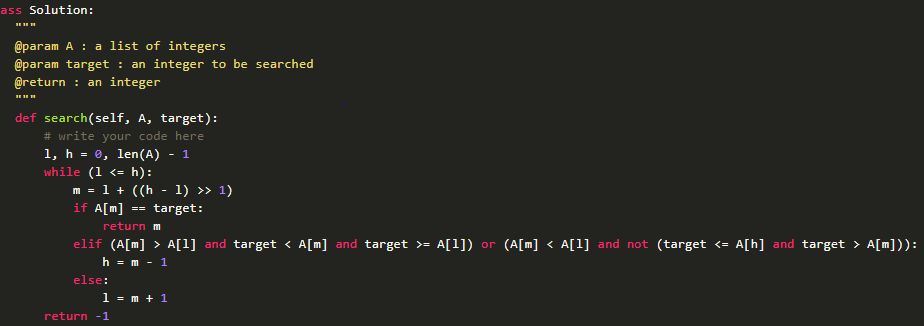
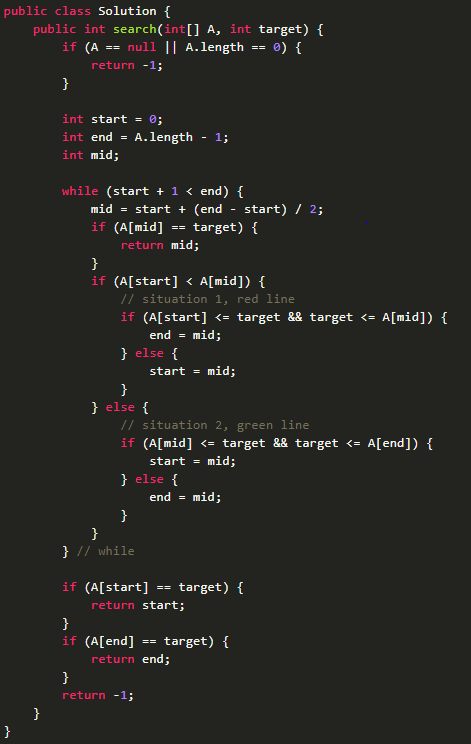
Search in Rotated Sorted Array

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.



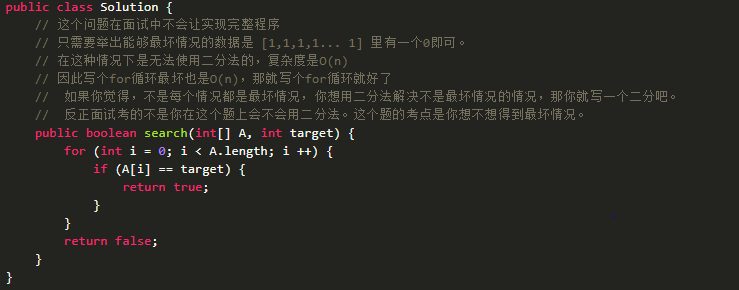
Search in Rotated Sorted Array II

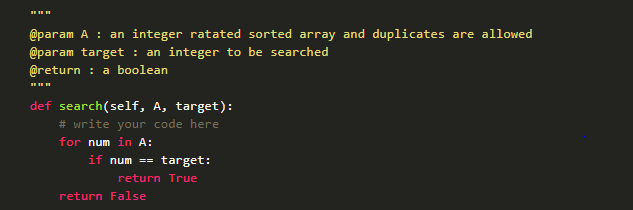
Follow up for Search in Rotated Sorted Array:

What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

Write a function to determine if a given target is in the array.

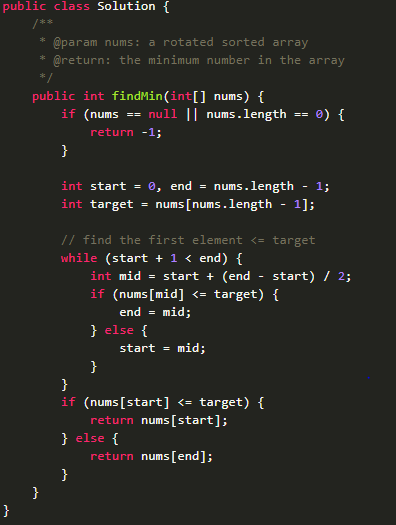


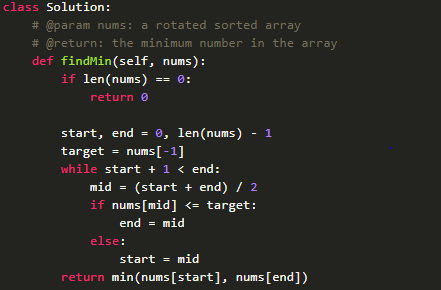


Find Minimum in Rotated Sorted Array

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2). Find the minimum element.

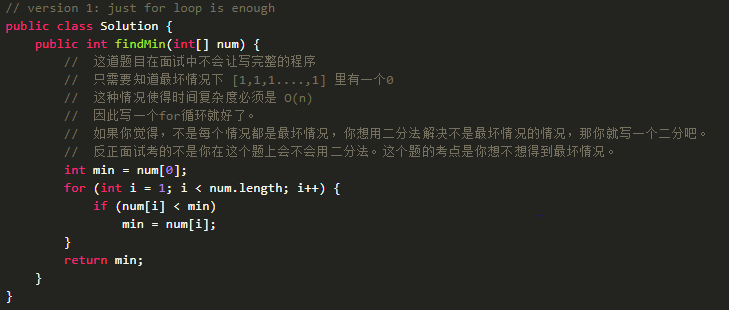


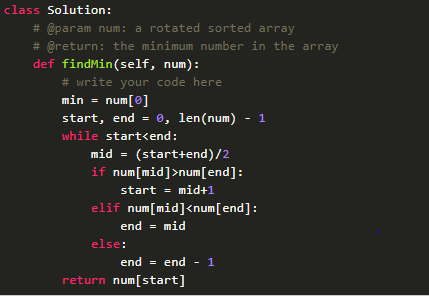


Find Minimum in Rotated Sorted Array II

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

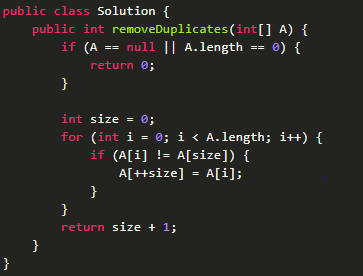
(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2). Find the minimum element.

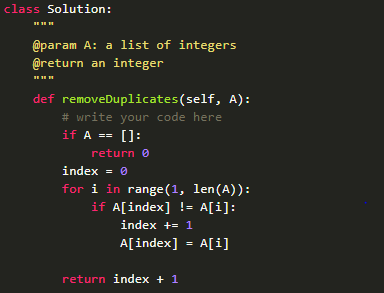




Remove Duplicates from Sorted Array I

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.

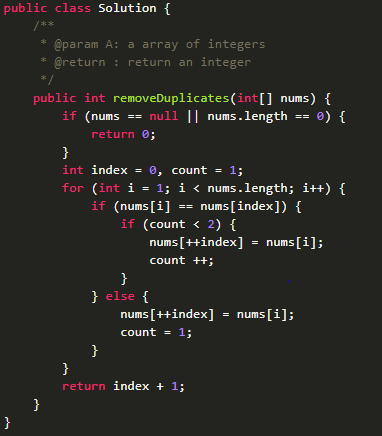


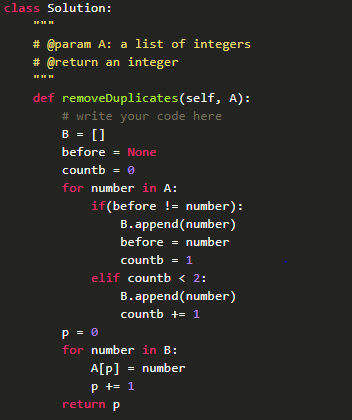


Remove Duplicates from Sorted Array II

Follow up for "Remove Duplicates": What if duplicates are allowed at most twice?

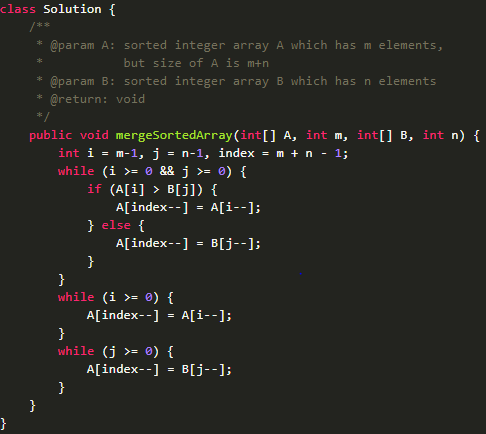
Given sorted array A = [1,1,1,2,2,3], return length = 5, and A is now [1,1,2,2,3].

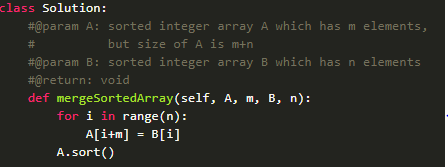




Merge Sorted Array

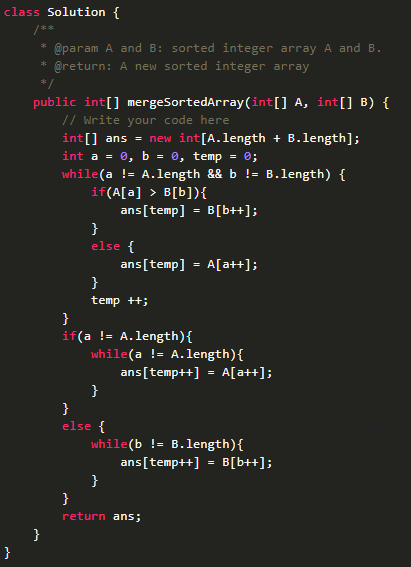
Given two sorted integer arrays A and B, merge B into A as one sorted array.

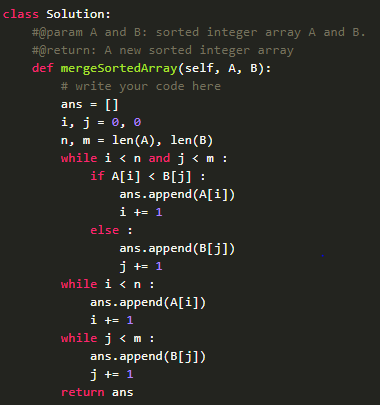




Merge Sorted Array II

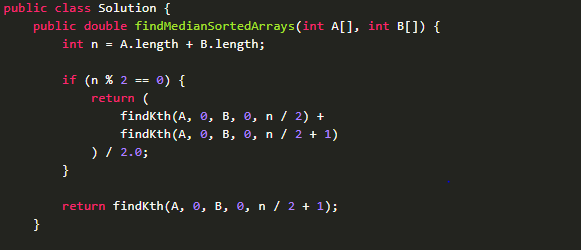
Merge two given sorted integer array A and B into a new sorted integer array.

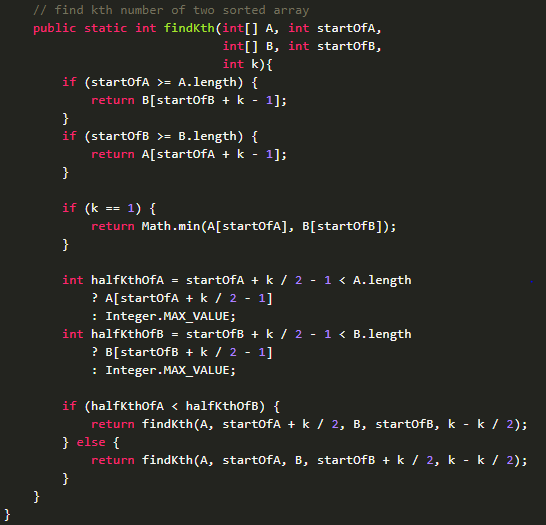


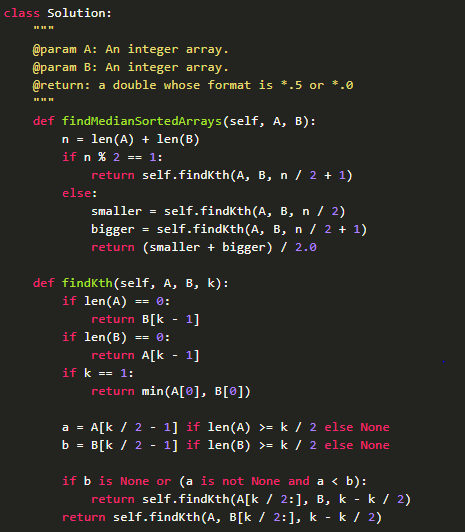


Median of Two Sorted Arrays

Two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays.





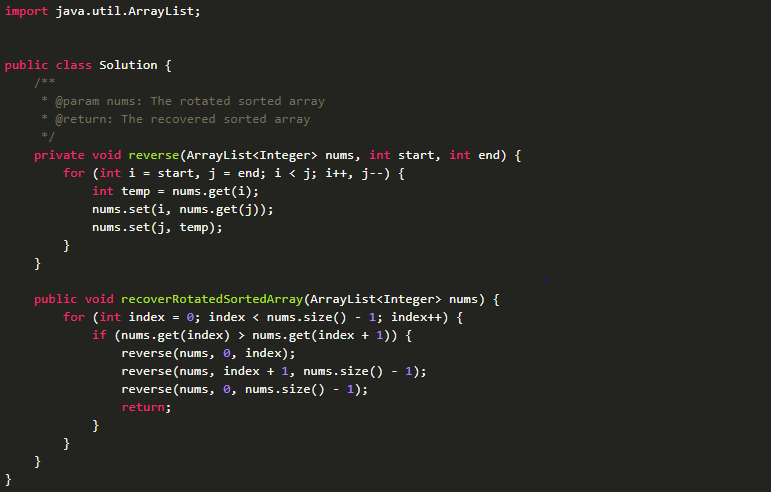


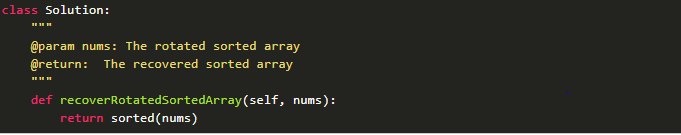
Recover rotated sorted array

Given a rotated sorted array, recover it to sorted array in-place.

[4, 5, 1, 2, 3] -> [1, 2, 3, 4, 5]

Challenge: In-place, O(1) extra space and O(n) time.





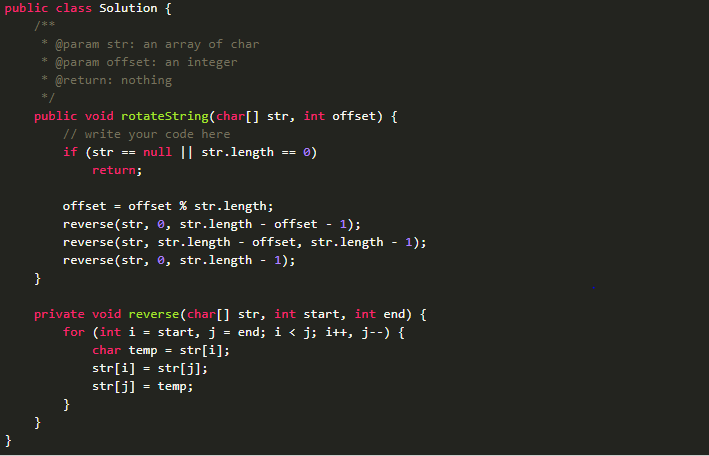
Rotate String

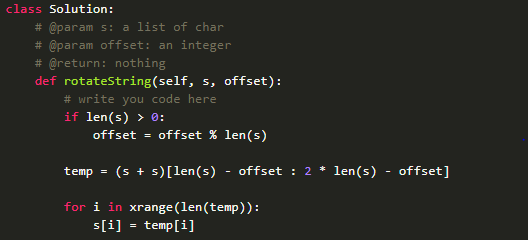
We are given two strings, A and B. A shift on A consists of taking string A and moving the leftmost character to the rightmost position. For example, if A = 'abcde', then it will be 'bcdea' after one shift on A. Return True if and only if A can become B after some number of shifts on A.

Input: A = 'abcde', B = 'cdeab' Output: true

Input: A = 'abcde', B = 'abced' Output: false

Note: A and B will have length at most 100.





Reverse Words in a String

Given an input string, reverse the string word by word.

Input: "the sky is blue", Output: "blue is sky the".

Note: A word is defined as a sequence of non-space characters. Input string may contain leading or trailing spaces. However, your reversed string should not contain leading or trailing spaces. You need to reduce multiple spaces between two words to a single space in the reversed string.

Follow up: For C programmers, try to solve it in-place in O(1) space.

