1. Given an integer array nums sorted in non-decreasing order, remove the duplicates in-place such that each unique element appears only once. The relative order of the elements should be kept the same. Then return the number of unique elements in nums.

Input: nums = [0,0,1,1,1,2,2,3,3,4]

Output: 5, nums = [0,1,2,3,4,\_,\_,\_,\_,\_]

Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

**Solution:**

public int removeDuplicates(int[] nums) {

int i = 0;

for (int n : nums) {

if (i == 0 || n > nums[i-1])

nums[i++] = n;

}

return i;

}

TC: O(n)

SC: O(1)

1. You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store.

Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.





**Solution:**

public int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int max = 0;

while(left < right){

int w = right - left;

int h = Math.min(height[left], height[right]);

int area = h \* w;

max = Math.max(max, area);

if(height[left] < height[right]) left++;

else if(height[left] > height[right]) right--;

else {

left++;

right--;

}

}

return max;

}

TC: O(N), SC: O(1)

1. Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

Input: nums = [1,2,3,4]

Output: [24,12,8,6]

**Solution:**

public int[] productExceptSelf(int[] nums) {

int n = nums.length;

int[] prefix = new int[n];

int[] suffix = new int[n];

int[] result = new int[n];

prefix[0] = 1;

for (int i = 1; i < n; i++) {

prefix[i] = prefix[i-1] \* nums[i-1];

}

suffix[n-1] = 1;

for (int i = n-2; i >= 0; i--) {

suffix[i] = suffix[i+1] \* nums[i+1];

}

for (int i = 0; i < n; i++) {

result[i] = prefix[i] \* suffix[i];

}

return result;

}

TC: O(n)

SC: O(n)

1. Given an array of meeting time intervals intervals where intervals[i] = [starti, endi], return the minimum number of conference rooms required.

Input: intervals = [[0,30],[5,10],[15,20]]

Output: 2



**Solution:**

public int minMeetingRooms(List<List<Integer>> a) {

Map<Integer, Integer> m = new TreeMap<>();

for (int i = 0; i < a.size(); i++) {

m.put(a.get(i).get(0), m.getOrDefault(a.get(i).get(0), 0) + 1);//start time

m.put(a.get(i).get(1), m.getOrDefault(a.get(i).get(1), 0) - 1);//end time

}

int c = 0;

int maxi = 0;

for (Map.Entry<Integer, Integer> entry : m.entrySet()) {

c += entry.getValue();

if (c > maxi) {

maxi = c;

}

}

return maxi;

}

TC: O(n)

SC: O(n)

1. Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears once or twice, return an array of all the integers that appears twice. You must write an algorithm that runs in O(n) time and uses only constant extra space.

Input: nums = [4,3,2,7,8,2,3,1]

Output: [2,3]

**Solution:**

public List<Integer> findDuplicates(int[] nums) {

List<Integer> duplicates = new ArrayList<>();

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

int index = Math.abs(nums[i]) - 1;

if (nums[index] < 0) {

duplicates.add(index + 1);

} else {

nums[index] = -nums[index];

}

}

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

nums[i] = Math.abs(nums[i]);

}

return duplicates;

}

TC: O(n)

SC: O(1)

1. There are n buildings in a line. You are given an integer array heights of size n that represents the heights of the buildings in the line. The ocean is to the right of the buildings. A building has an ocean view if the building can see the ocean without obstructions. Formally, a building has an ocean view if all the buildings to its right have a smaller height. Return a list of indices (0-indexed) of buildings that have an ocean view, sorted in increasing order.



Input: heights = [4,2,3,1]

Output: [0,2,3]

Explanation: Building 1 (0-indexed) does not have an ocean view because building 2 is taller.

**Solution:**

public List<Integer> findBuildings(int[] a) {

int n = a.length;

List<Integer> ans = new ArrayList<>();

int maxi = a[n - 1];

ans.add(n - 1);

for (int i = n - 2; i >= 0; i--) {

if (a[i] > maxi) {

maxi = a[i];

ans.add(i);

}

}

Collections.reverse(ans);

return ans;

}

TC: O(n)

SC: O(n)