Q1.public int[][] construct2DArray(int[] original, int m, int n) {

int[][] result = new int[m][n];

if (original.length != m \* n) {

return new int[0][0];

}

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

int row = i / n;

int col = i % n;

result[row][col] = original[i];

}

return result;

}

Q2.public int arrangeCoins(int n) {

int completeRows = 0;

int remainingCoins = n;

int currentRow = 1;

while (remainingCoins >= currentRow) {

completeRows++;

remainingCoins -= currentRow;

currentRow++;

}

return completeRows;

}

Q3.public int[] sortedSquares(int[] nums) {

int n = nums.length;

int[] result = new int[n];

int left = 0;

int right = n - 1;

int i = n - 1;

while (left <= right) {

int leftSquare = nums[left] \* nums[left];

int rightSquare = nums[right] \* nums[right];

if (leftSquare > rightSquare) {

result[i] = leftSquare;

left++;

} else {

result[i] = rightSquare;

right--;

}

i--;

}

return result;

}

Q4.def findDisappearedNumbers(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

distinct\_nums1 = list(set1 - set2)

distinct\_nums2 = list(set2 - set1)

return [distinct\_nums1, distinct\_nums2]

Q5.public int findTheDistanceValue(int[] arr1, int[] arr2, int d) {

int distanceValue = 0;

for (int i = 0; i < arr1.length; i++) {

boolean isDistanceValid = true;

for (int j = 0; j < arr2.length; j++) {

if (Math.abs(arr1[i] - arr2[j]) <= d) {

isDistanceValid = false;

break;

}

}

if (isDistanceValid) {

distanceValue++;

}

}

return distanceValue;

}

Q6.import java.util.ArrayList;

import java.util.List;

public class 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);

}

nums[index] = -nums[index];

}

return duplicates;

}

}

Q7.public class Solution {

public int findMin(int[] nums) {

int left = 0;

int right = nums.length - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (nums[mid] > nums[right]) {

// The minimum element is in the right half

left = mid + 1;

} else {

// The minimum element is in the left half or is the middle element

right = mid;

}

}

// At the end of the loop, left and right will be pointing to the minimum element

return nums[left];

}

}

Q8.import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Solution {

public int[] findOriginalArray(int[] changed) {

if (changed.length % 2 != 0) {

return new int[0];

}

Arrays.sort(changed);

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

for (int num : changed) {

if (num % 2 != 0) {

return new int[0];

}

int half = num / 2;

if (original.contains(half)) {

original.remove(Integer.valueOf(half));

} else {

original.add(num);

}

}

int[] result = new int[original.size()];

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

result[i] = original.get(i);

}

return result;

}

}