**Question 1**

**Convert 1D Array Into 2D Array**

**You are given a 0-indexed 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with m rows and n columns using all the elements from original.**

**The elements from indices 0 to n - 1 (inclusive) of original should form the first row of the constructed 2D array, the elements from indices n to 2 \* n - 1 (inclusive) should form the second row of the constructed 2D array, and so on.**

**Return *an* m x n *2D array constructed according to the above procedure, or an empty 2D array if it is impossible*.**

**Example 1:**

**Input: original = [1,2,3,4], m = 2, n = 2**

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

**Prgm:**

class convert1D\_2D {

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

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

return new int[][] {};

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

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

ans[i / n][i % n] = original[i];

return ans;

}

}

**Question 2**

**You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase may be incomplete.**

**Given the integer n, return *the number of complete rows of the staircase you will build*.**

**Example 1:**

**Input: n = 5**

**Output: 2**

**Prgm:**

class stairCase{

public static int arrangeCoins(int n) {

if(n==0){

return 0;

}

int start = 1;

int end = n;

int mid=0;

int ans = 0;

while(start<=end){

mid = start + (end-start)/2;

if((mid\*(mid+1))/2 == n){

return mid;

}

else if((mid\*(mid+1))/2 < n){

start = mid+1;

ans = mid;

}

else{

end = mid-1;

}

}

return ans;

}

public static void main(String[] args) {

int n = 5;

System.out.println(arrangeCoins(n));

}

}

**Question 3**

**Given an integer array nums sorted in non-decreasing order, return *an array of the squares of each number sorted in non-decreasing order*.**

**Example 1:**

**Input: nums = [-4,-1,0,3,10]**

**Output: [0,1,9,16,100]**

**Prgm:**

class Squ\_sortedArray{

public static int[] sortedSquares(int[] nums) {

int[]res = new int[nums.length];

int i = 0;

int j = nums.length-1;

int index= nums.length-1;

while(i <= j){

int val1 = nums[1] \* nums[i];

int val2 = nums[j] \* nums[j];

if(val1 > val2){

res[index] = val1;

i++;

}else{

res[index] = val2;

j--;

}

index--;

}

return res;

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int nums[] = new int[n];

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

nums[i] = scn.nextInt();

int[] res = sortedSquares(nums);

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

System.out.print(res[i] + " ");

}

}

}

**Question 4**

**Given two 0-indexed integer arrays nums1 and nums2, return *a list* answer *of size* 2 *where:***

* **answer[0] *is a list of all distinct integers in* nums1 *which are not present in* nums2\*.\***
* **answer[1] *is a list of all distinct integers in* nums2 *which are not present in* nums1.**

**Note that the integers in the lists may be returned in any order.**

**Example 1:**

**Input: nums1 = [1,2,3], nums2 = [2,4,6]**

**Output: [[1,3],[4,6]]**

**Prgm:**

class Solution{

public static List<List<Integer>> findDifference(int[] nums1, int[] nums2) {

int i=0,j=0;

Arrays.sort(nums1);

Arrays.sort(nums2);

int n=nums1.length;

int m=nums2.length;

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

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

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

while(i<n && j<m){

if(nums1[i]<nums2[j]){

if(a.size()==0 || nums1[i]!=a.get(a.size()-1))

a.add(nums1[i]);

i++;

}

else if(nums1[i]==nums2[j]){

int aa=nums1[i];

while(i<n && nums1[i]==aa)

i++;

while(j<m && nums2[j]==aa)

j++;

}

else{

if(b.size()==0 || nums2[j]!=b.get(b.size()-1))

b.add(nums2[j]);

//Avoiding the duplicates

j++;

}

}

while(i<n){

if(a.size()==0 || nums1[i]!=a.get(a.size()-1))

a.add(nums1[i]);

i++;

}

while(j<m){

if(b.size()==0 || nums2[j]!=b.get(b.size()-1))

b.add(nums2[j]);

j++;

}

ans.add(a);

ans.add(b);

return ans;

}

public static void main(String[] args) {

int nums1[] = {1,2,3};

int nums2[] = {2,4,6};

System.out.println(findDifference(nums1, nums2));

}

}

**Question 5**

**Given two integer arrays arr1 and arr2, and the integer d, *return the distance value between the two arrays*.**

**The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.**

**Example 1:**

**Input: arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2**

**Output: 2**

**Prgm:**

class Distance\_Value{

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

Arrays.sort(arr2);

int ans = 0;

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

int a = Arrays.binarySearch(arr2, 0, arr2.length, arr1[i]);

if (a < 0) a = -(a+1);

boolean flag = false;

if(a<arr2.length && Math.abs(arr2[a] - arr1[i]) <= d)flag = true;

if(a != 0 && Math.abs(arr2[a-1] - arr1[i]) <= d)flag = true;

if(!flag)

ans++;

}

return ans;

}

public static void main(String[] args) {

int[] arr1 = {4,5,8};

int[] arr2 = {10,9,1,8};

int d = 2;

System.out.println(findTheDistanceValue(arr1, arr2, d));

}

}

**Question 6**

**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.**

**Example 1:**

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

**Output:**

**[2,3]**

**Prgm:**

class Duplicates{

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

Arrays.sort(nums);

int val = nums[0];

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

for(int i = 1 ; i < nums.length ;i++)

{

if(nums[i] == val)

{

res.add(val);

}

val = nums[i];

}

return res;

}

public static void main(String[] args) {

int nums[] = {4,3,2,7,8,2,3,1};

System.out.println(FindDuplicates(nums));

}

}

**Question 7**

**Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:**

* **[4,5,6,7,0,1,2] if it was rotated 4 times.**
* **[0,1,2,4,5,6,7] if it was rotated 7 times.**

**Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].**

**Given the sorted rotated array nums of unique elements, return *the minimum element of this array*.**

**You must write an algorithm that runs in O(log n) time.**

**Example 1:**

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

**Output: 1**

**Prgm:**

class Rotated\_sortedArray{

public static int findMin(int[] nums) {

int low = 0;

int high = nums.length - 1;

while(low < high){

int mid = (low+high)/2;

if(nums[mid] < nums[high]){

high = mid;

}

else if(nums[mid] > nums[high]){

low = mid+1;

}

else{

high -= 1;

}

}

return nums[low];

}

public static void main(String[] args) {

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

System.out.println(findMin(nums));

}

}

**Question 8**

**An integer array original is transformed into a doubled array changed by appending twice the value of every element in original, and then randomly shuffling the resulting array.**

**Given an array changed, return original *if* changed *is a doubled array. If* changed *is not a doubled array, return an empty array. The elements in* original *may be returned in any order*.**

**Example 1:**

**Input: changed = [1,3,4,2,6,8]**

**Output: [1,3,4]**

**Prgm:**

class originalArray{

public static List<Integer>findOriginal(int[] arr){

Map<Integer, Integer> numFreq = new HashMap<>();

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

numFreq.put(

arr[i],

numFreq.getOrDefault(arr[i], 0)

+ 1);

}

Arrays.sort(arr);

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

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

int freq = numFreq.get(arr[i]);

if (freq > 0) {

res.add(arr[i]);

numFreq.put(arr[i], freq - 1);

int twice = 2 \* arr[i];

numFreq.put(

twice,

numFreq.get(twice) - 1);

}

}

return res;

}

public static void main(String[] args){

List<Integer> res = findOriginal(

new int[] {1,3,4,2,6,8 });

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

System.out.println(

res.get(i) + " ");

}

}

}