**Assignment Questions 5**

<aside> 💡 **Question 1**

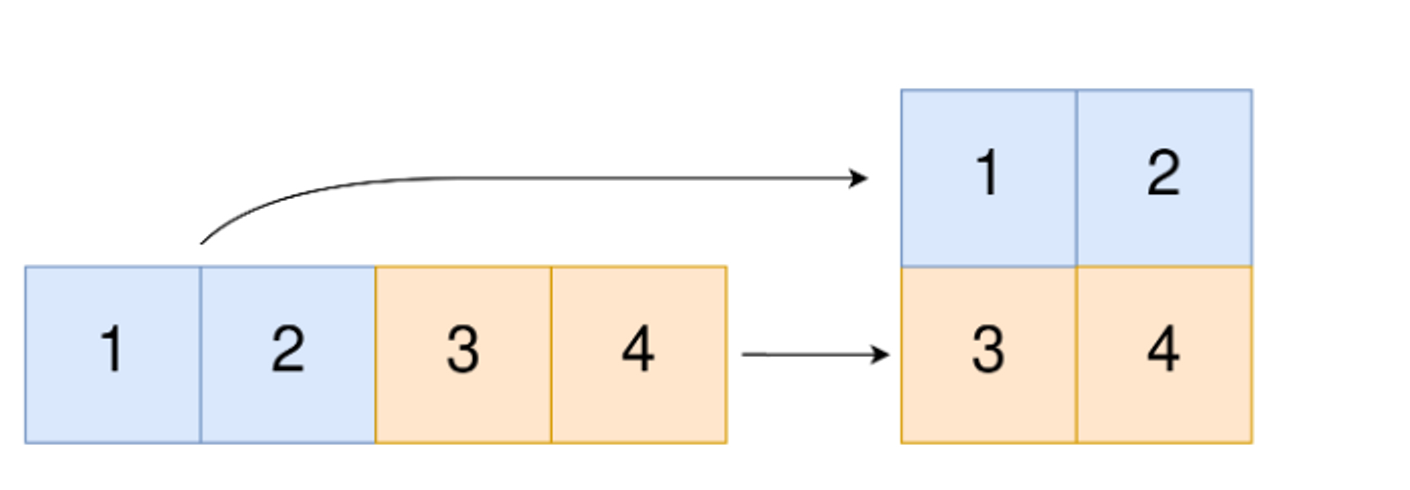
Convert 1D Array Into 2D Array

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]]

**Explanation:** The constructed 2D array should contain 2 rows and 2 columns.

The first group of n=2 elements in original, [1,2], becomes the first row in the constructed 2D array.

The second group of n=2 elements in original, [3,4], becomes the second row in the constructed 2D array.

public class convert1DArrayInto2DArray {

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

        int [][] arr=new int [m][n];

        if(original.length!=m\*n)return new int[0][0];

        int ix=0;

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

            for(int j=0;j<arr[0].length;j++){

                arr[i][j]=original[ix++];

            }

        }

        return arr;

    }

    public static void main(String[] args) {

        int [] original = {1,2,3,4};

        int m = 2, n = 2;

        int [] [] arr = construct2DArray(original, m, n);

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

            for (int j = 0; j < arr[i].length; j++)

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

            System.out.print("\n");

        }

    }

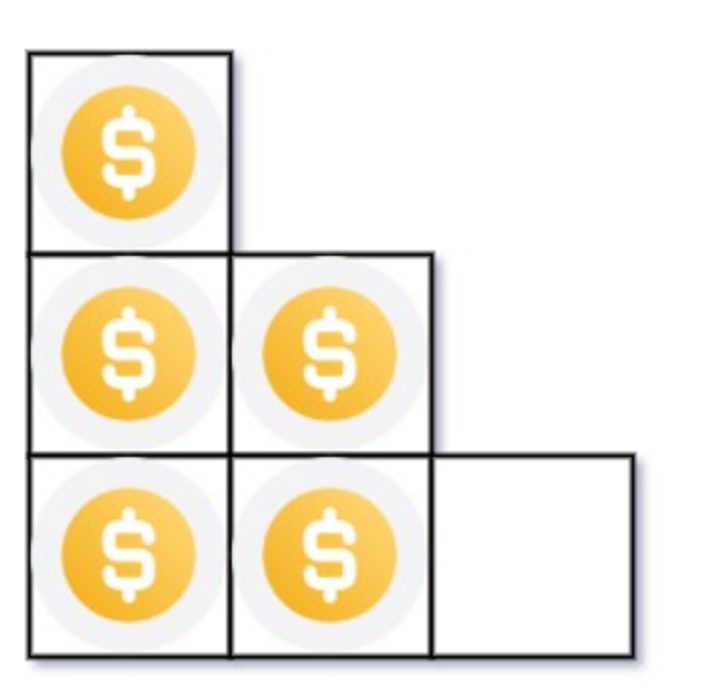
}

💡 **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

**Explanation:** Because the 3rd row is incomplete, we return 2.

package Assignment\_5;

public class arrangingCoins {

    static int arrangeCoins(int n) {

        long start=1;

        long sum=1;

        while( sum <= n){

            sum+= ++start;

        }

        return (int) start-1;

    }

    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]

**Explanation:** After squaring, the array becomes [16,1,0,9,100].

After sorting, it becomes [0,1,9,16,100].

package Assignment\_5;

import java.util.Arrays;

public class squaresOfSortedArray {

    static int[] sortedSquares(int[] A) {

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

            A[i] = A[i] \* A[i];

        }

            Arrays.sort(A);

            return A;

    }

    public static void main(String[] args) {

        int [] nums = {-4,-1,0,3,10};

        System.out.println(Arrays.toString(sortedSquares(nums)));

    }

}

💡 **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]]

**Explanation:**

For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].

package Assignment\_5;

import java.util.\*;

public class findTheDifferenceOfTwoArrays {

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

        HashMap<Integer,Integer> mp = new HashMap<>();

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

        for(int x:nums1){

            mp.put(x,1);

        }

        for(int x:nums2){

            if(mp.getOrDefault(x,0) == 1)

                mp.put(x,3);

            else if(mp.getOrDefault(x,0) == 0)

                mp.put(x,2);

        }

        List<Integer> f1 = new ArrayList<>();

        List<Integer> f2 = new ArrayList<>();

        for(int x:nums1){

            if(mp.getOrDefault(x,0) == 1){

                mp.put(x,0);

                f1.add(x);

            }

        }

        ans.add(f1);

        for(int x:nums2){

            if(mp.getOrDefault(x,0) == 2){

                mp.put(x,0);

                f2.add(x);

            }

        }

        ans.add(f2);

        return ans;

    }

    public static void main(String[] args) {

        int [] nums1 = {1,2,3}, 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

**Explanation:**

For arr1[0]=4 we have:

|4-10|=6 > d=2

|4-9|=5 > d=2

|4-1|=3 > d=2

|4-8|=4 > d=2

For arr1[1]=5 we have:

|5-10|=5 > d=2

|5-9|=4 > d=2

|5-1|=4 > d=2

|5-8|=3 > d=2

For arr1[2]=8 we have:

**|8-10|=2 <= d=2**

**|8-9|=1 <= d=2**

|8-1|=7 > d=2

**|8-8|=0 <= d=2**

package Assignment\_5;

import java.util.Arrays;

public class findTheDistanceValueBetweenTwoArrays {

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

        Arrays.sort(arr2);

        int ans = 0;

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

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

            if (it < 0) it = -(it+1);

            boolean isIt = false;

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

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

            if(!isIt)

                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]

package Assignment\_5;

import java.util.\*;

public class findAllDuplicatesInAnArray {

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

        ArrayList<Integer> al=new ArrayList<>();

         HashMap<Integer,Integer> map=new HashMap<>();

         if(nums.length==1){

             return al;

         }

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

             map.put(nums[i],map.getOrDefault(nums[i],0)+1);

         }

         for(int i:map.keySet()) {

             if(map.get(i)>1) {

                 al.add(i);

             }

         }

         Collections.sort(al);

         return al;

    }

    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

**Explanation:**

The original array was [1,2,3,4,5] rotated 3 times.

package Assignment\_5;

public class findMinimumInRotatedSortedArray {

    static int findMin(int[] nums) {

        int l = 0;

        int r = nums.length - 1;

        while (l < r) {

          final int m = (l + r) / 2;

          if (nums[m] < nums[r])

            r = m;

          else

            l = m + 1;

        }

        return nums[l];

      }

      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]

**Explanation:** One possible original array could be [1,3,4]:

* Twice the value of 1 is 1 \* 2 = 2.
* Twice the value of 3 is 3 \* 2 = 6.
* Twice the value of 4 is 4 \* 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4].

package Assignment\_5;

import java.util.\*;

public class findOriginalArrayFromDoubledArray {

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

        int[] temp = new int[changed.length/2];

        Queue<Integer> x = new LinkedList<>();

        Arrays.sort(changed);

        int i = 0;

        for(int num : changed){

            if(!x.isEmpty() && x.peek()==num)

                temp[i++] = x.poll()/2;

            else x.add(num\*2);

        }

        return x.size()>0 ? new int[]{}:temp;

    }

    public static void main(String[] args) {

        int [] changed = {1,3,4,2,6,8};

        System.out.println(Arrays.toString(findOriginalArray(changed)));

    }

}