**Assignment Questions and Answer 4**

💡 **Question 1** Given three integer arrays arr1, arr2 and arr3 **sorted** in **strictly increasing** order, return a sorted array of **only** the integers that appeared in **all** three arrays.

**Example 1:**

Input: arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]

Output: [1,5]

**Explanation:** Only 1 and 5 appeared in the three arrays.

import java.util.\*;

public class intersectionofThreeArray {

    static List<Integer> arraysIntersection(int[] arr1, int[] arr2, int[] arr3) {

        List<Integer> rst = new LinkedList<>();

        int i = 0, j = 0, k = 0;

        while (i < arr1.length && j < arr2.length && k < arr3.length) {

            if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {

                if (rst.isEmpty() || arr1[i] != rst.get(rst.size() - 1)) rst.add(arr1[i]);

                i++;

                j++;

                k++;

            } else if (arr1[i] < arr2[j]) i++;

            else if (arr2[j] < arr3[k]) j++;

            else k++;

        }

        return rst;

    }

    public static void main(String[] args) {

        int [] arr1 = {1,2,3,4,5}, arr2 = {1,2,5,7,9},

        arr3 = {1,3,4,5,8};

        System.out.println(arraysIntersection(arr1, arr2, arr3));

    }

}

💡 **Question 2**

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].

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));

    }

}

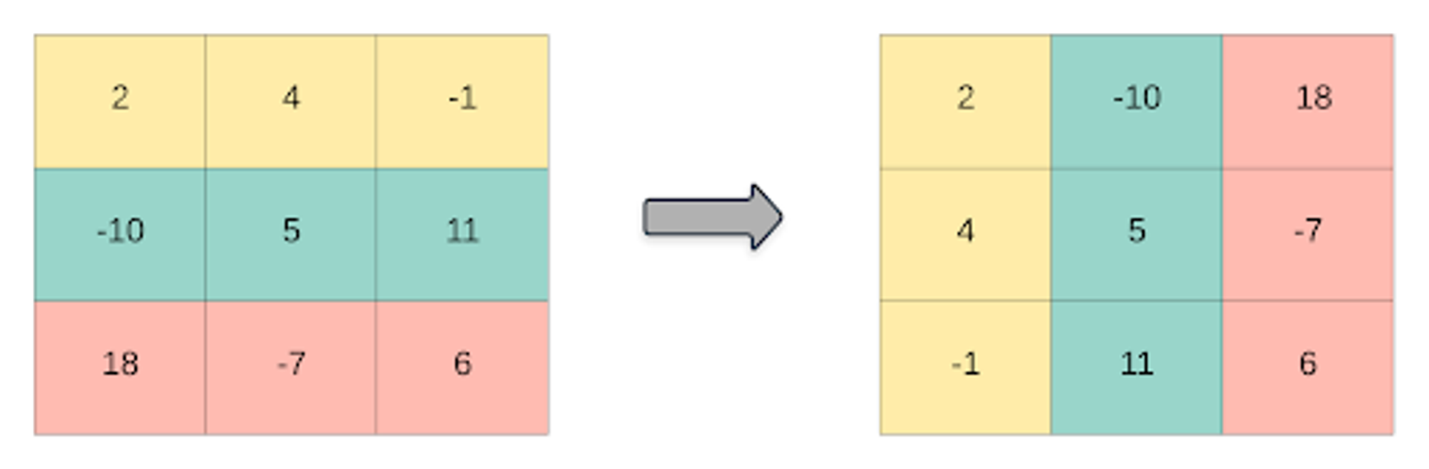
💡 **Question 3** Given a 2D integer array matrix, return *the* ***transpose*** *of* matrix.

The **transpose** of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

**Example 1:**

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[1,4,7],[2,5,8],[3,6,9]]



public class trasposeMatrix {

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

        int N = A.length;

        for (int i = 0; i < N; i++){

            for (int j = i + 1; j < N; j++) {

                int temp = A[i][j];

                A[i][j] = A[j][i];

                A[j][i] = temp;

            }

        }

        return A;

    }

    public static void main(String[] args) {

        int [][] matrix = {{1,2,3},{4,5,6},{7,8,9}};

        transpose(matrix);

        System.out.print("Modified matrix is \n");

        int N = matrix.length;

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < N; j++)

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

            System.out.print("\n");

        }

    }

}

💡 **Question 4** Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2), ..., (an, bn) such that the sum of min(ai, bi) for all i is **maximized**. Return *the maximized sum*.

**Example 1:**

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

Output: 4

**Explanation:** All possible pairings (ignoring the ordering of elements) are:

1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4

So the maximum possible sum is 4.

import java.util.Arrays;

public class arrayPartion {

    static int arrayPairSum(int[] nums) {

        int sum = 0;

        Arrays.sort(nums);

        for(int i = 1; i < nums.length; i+=2){

            sum += Math.min(nums[i], nums[i-1]);

        }

        return sum;

    }

    public static void main(String[] args) {

        int [] nums = {1,4,3,2};

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

    }

}

💡 **Question 5** 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.

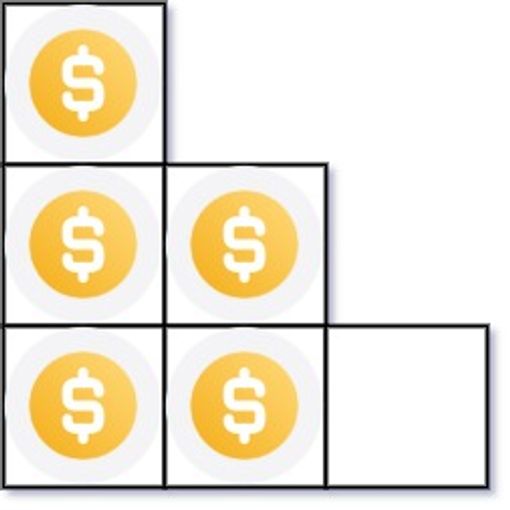
Given the integer n, return *the number of* ***complete rows*** *of the staircase you will build*.

**Example 1:**

**Input:** n = 5

**Output:** 2

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



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 6** 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]

import java.util.Arrays;

public class squareOfSortedArray {

    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 7** You are given an m x n matrix M initialized with all 0's and an array of operations ops, where ops[i] = [ai, bi] means M[x][y] should be incremented by one for all 0 <= x < ai and 0 <= y < bi.

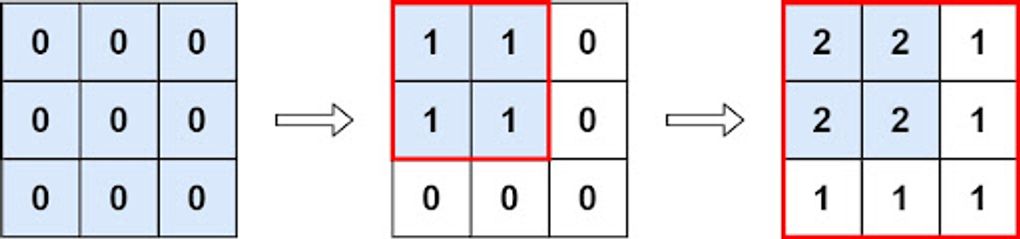
Count and return *the number of maximum integers in the matrix after performing all the operations*

**Example 1:**

**Input:** m = 3, n = 3, ops = [[2,2],[3,3]]

**Output:** 4

**Explanation:** The maximum integer in M is 2, and there are four of it in M. So return 4.



public class rangeAddition2 {

    static int maxCount(int m, int n, int[][] ops) {

        int leftMin = m, rightMin = n;

        for (int[] op : ops) {

            leftMin = Math.min(op[0], leftMin);

            rightMin = Math.min(op[1], rightMin);

        }

        return leftMin \* rightMin;

    }

    public static void main(String[] args) {

        int m = 3, n = 3;

        int [][] ops = {{2,2},{3,3}};

        System.out.println(maxCount(m, n, ops));

    }

}

💡 **Question 8**

Given the array nums consisting of 2n elements in the form [x1,x2,...,xn,y1,y2,...,yn].

*Return the array in the form* [x1,y1,x2,y2,...,xn,yn].

**Example 1:**

**Input:** nums = [2,5,1,3,4,7], n = 3

**Output:** [2,3,5,4,1,7]

**Explanation:** Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].

import java.util.Arrays;

public class shuffleTheArray {

    static int[] shuffle(int[] nums, int n) {

        int[] ans = new int[2\*n];

        int j=0;

        for(int i=0; i<2\*n; i=i+2){

            ans[i] = nums[j];

            ans[i+1] = nums[n+j];

            j++;

        }

        return ans;

    }

    public static void main(String[] args) {

        int [] nums = {2,5,1,3,4,7};

        int n = 3;

        System.out.println(Arrays.toString(shuffle(nums, n)));

    }

}