DSA ASSIGNMENT-4(2D-ARRAY)

Q1. 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.

Solution:

```
boolean solution(int[] arr, int[][] pieces) {
    HashMap<Integer,int[]> map = new HashMap<>();
    int sum=0;
    for(int[] p : pieces){
        map.put(p[0],p);
        sum+=p.length;
    }
    if(sum>arr.length) return false;
    int i=0;
    while(i<arr.length){
        if(!map.containsKey(arr[i])) return false;
        int tp=arr[i];
        int j=0;
        while(j<map.get(tp).length){
            if(map.get(tp)[j++] != arr[i++]) return false;
        }
    }
    return true;
}
```

Q2. 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].

Solution:

```
class Solution {
    public List<List<Integer>> findDifference(int[] nums1, int[]
nums2) {


     HashSet<Integer> set1=new HashSet<Integer>();
      HashSet<Integer> set2=new HashSet<Integer>();

     for(int ele: nums1){
         set1.add(ele);
     }

     for(int ele:nums2){
         set2.add(ele);
     }


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

      ArrayList<Integer> l1=new ArrayList<>();

      ArrayList<Integer> l2=new ArrayList<>();

     for(int ele:set2){

         if(set1.contains(ele)==false){
           l1.add(ele);
         }
     }
```

```

       for(int ele:set1){

         if(set2.contains(ele)==false){
           l2.add(ele);
         }
     }


       list.add(l2);
       list.add(l1);
       return list;
    }
}
```

Q3. 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]]

Solution:

```
class Solution {
    public int[][] transpose(int[][] matrix) {
        int r= matrix.length;
        int c=matrix[0].length;
        int[][] transpose= new int[c][r];
        for(int i=0; i<c; i++)
        {
            for(int j=0; j<r;j++)
            {
                transpose[i][j]=matrix[j][i];
            }
        }
        return transpose;}}
```

Q4. 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.

Solution:

```
class Solution {
    public int arrayPairSum(int[] nums) {
        Arrays.sort(nums);
        int len = nums.length;
        int result = 0;
        for (int i = 0; i < len ; i += 2) {
            result += nums[i];
        }
        return result;
    }
}
```

Q5. 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*.

Solution:

```
class Solution {
    public int arrangeCoins(int n) {
        int ans = 1;
    while(n > 0){
      ans++;
      n = n-ans;
    }
    return ans-1;
    }
}
```

Q6. 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]

```
Solution: class Solution {

    public int[] sortedSquares(int[] A) {
        int n = A.length;
        int[] result = new int[n];
        int i = 0, j = n - 1;
        for (int p = n - 1; p >= 0; p--) {
            if (Math.abs(A[i]) > Math.abs(A[j])) {
                result[p] = A[i] * A[i];
                i++;
            } else {
                result[p] = A[j] * A[j];
                j--;
            }
        }
        return result;
    }
}
```

Q7. 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*

Solution:

```
class Solution {
    public int maxCount(int m, int n, int[][] ops) {
        int k=ops.length;
        for (int i=0;i<k;i++)
        {
            int z=ops[i][0] ,x=ops[i][1];
            n=Math.min(n,x);
            m=Math.min(m,z);
        }
        return (m*n);
    }
}
```

Q8. 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].

Solution:

```
class Solution {
    public int[] shuffle(int[] nums, int n) {
        int temp = 2 * n;
        int [] arr = new int[temp];
        for(int i = 0; i < n; i++) {
            arr[2 * i] = nums[i];
            arr[(2 * i )+ 1] = nums[i + n];
        }
        return arr;
    }
}
```