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.

```
class Solution {

        public List<Integer> arraysIntersection(int[]
    arr1, int[] arr2, int[] arr3) {
            List<Integer> list = new
    ArrayList<Integer>();
            int length1 = arr1.length, length2 =
    arr2.length, length3 = arr3.length;
            int index1 = 0, index2 = 0, index3 = 0;
            while (index1 < length1 && index2 < length2
    && index3 < length3) {
                int num1 = arr1[index1], num2 =
    arr2[index2], num3 = arr3[index3];
                if (num1 == num2 && num1 == num3) {
                    list.add(num1);
                    index1++;
                    index2++;
                    index3++;
                } else {
                    int inc1 = 0, inc2 = 0, inc3 = 0;
                    if (num1 < num2 || num1 < num3)
                        inc1 = 1;
                    if (num2 < num1 || num2 < num3)
                        inc2 = 1;
                    if (num3 < num1 || num3 < num2)
                        inc = 1;
                    index1 += inc1;
                    index2 += inc2;
                    index3 += inc3;
                }
            }
            return list;
        }
    }
```

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].

```
class Solution {
  public List<List<Integer>> findDifference(int[] nums1, int[]
nums2) {
    List<List<Integer>> al= new ArrayList<>();
    HashSet<Integer> set1= new HashSet<>();
    HashSet<Integer> set2= new HashSet<>();
    List<Integer> list1= new ArrayList<>();
     List<Integer> list2= new ArrayList<>();
    for(int i: nums1) set1.add(i);

    for(int i: nums2) set2.add(i);

    for(int i: set1){

      if(set2.contains(i)) continue;
      else{
         list1.add(i);
```

```
      }

    }
    al.add(list1);
    for(int i: set2){

      if(set1.contains(i)) continue;
      else{

        list2.add(i);
      }
    }
    al.add(list2);
    return al;
  }
}
```

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]]

```
class Solution {
  public int[][] transpose(int[][] matrix) {
     int m= matrix.length;
    int n= matrix[0].length;

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

    for(int i=0; i< m; i++){
      for(int j=0; j< n; j++){
        tmax[j][i]= matrix[i][j];

      }
    }
    return tmax;

  }
```

}

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.

</aside>

```
class Solution {
  public int arrayPairSum(int[] nums) {
    int max= 0;
    Arrays.sort(nums);
    for(int i = 0; i< nums.length; i++)
    System.out.println(nums[i]);
    for(int i = 0; i< nums.length-1; i=i+2){
      max += Math.min(nums[i], nums[i+1]);
    }
    return max;

  }
}
```

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.

</aside>

```
class Solution {
  public int arrangeCoins(int n) {
    int coin = n;
    for(int i=1;i<=n;i++){
      coin=coin-i;
      if(coin<0){
        return i-1;
      }
    }

    return 1;
  }
}
```

<aside> 💡

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]

</aside>

```
class Solution {
  public int[] sortedSquares(int[] nums) {
    int n= nums.length;
    int index= n-1;
```

```
    int l=0, r= index;
    int[] res= new int[n]; // 2pointer approach
    while(l<=r){
      if(Math.abs(nums[l])< Math.abs(nums[r])){
        res[index--]= nums[r]*nums[r];
        r--;
      } else{
        res[index--]= nums[l]*nums[l];
        l++;

      }

    }
    return res;


  }
}
```

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.

</aside>

```
class Solution {
  public int maxCount(int m, int n, int[][] ops) {
    for (int[] op : ops) {
      m = Math.min(op[0], m);
      n = Math.min(op[1], n);
```

```
    }
    return m * n;
  }
}
```

<aside> 💡

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].

</aside>

```
class Solution {
  public int[] shuffle(int[] nums, int n) {


    int[] result = new int[nums.length];
    for (int i = 0; i < n; i++) {
      result[i * 2] = nums[i];
      result[i * 2 + 1] = nums[i + n];
    }
    return result;
  }
}
```