<aside>

💡 **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].

</aside>

```
class Solution {
    // Returns the elements in the first arg nums1 that don't exist in the second arg nums2.
    List<Integer> getElementsOnlyInFirstList(int[] nums1, int[] nums2) {
        Set<Integer> onlyInNums1 = new HashSet<> ();
        // Store nums2 elements in an unordered set.
        Set<Integer> existsInNums2 = new HashSet<> ();
        for (int num : nums2) {
            existsInNums2.add(num);
        }

        // Iterate over each element in the list nums1.
        for (int num : nums1) {
            if (!existsInNums2.contains(num)) {
                onlyInNums1.add(num);
            }
        }

        // Convert to vector.
        return new ArrayList<>(onlyInNums1);
    }

    public List<List<Integer>> findDifference(int[] nums1, int[] nums2) {
        return Arrays.asList(getElementsOnlyInFirstList(nums1, nums2),
getElementsOnlyInFirstList(nums2, nums1));
    }
}
```

<aside>

💡 **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]]

</aside>

```
class Solution {
  public int[][] transpose(int[][] A) {
    int[][] ans = new int[A[0].length][A.length];

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

    return ans;
  }
}
```

<aside>

💡 **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) {
    Arrays.sort(nums);
    int len = nums.length;
    int result = 0;
    for (int i = 0; i < len - 1; i += 2) {
      result += nums[i];
    }
    return result;
  }
}
```

<aside>

💡 **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*.

</aside>

```
class Solution {
public int arrangeCoins(int n) {

    long start=1;
    long sum=1;
    while( sum <= n){
        sum+= ++start;
    }

    return (int) start-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[] A) {
    for (int i = 0; i < A.length; i++)
      A[i] = A[i] * A[i];
    Arrays.sort(A);
    return A;
  }
}
```

<aside>

💡 **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*

</aside>

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

<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];
        }

        for(int j = 1; j < n+1; j++) {
            result[(j*2)-1] = nums[n+j-1];
        }


        return result;
    }
}
```