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.

Soln:

```
def find_common_elements(arr1, arr2, arr3):
    i, j, k = 0, 0, 0  # Pointers for arr1, arr2, and arr3
    result = []  # Array to store the common elements

    while i < len(arr1) and j < len(arr2) and k < len(arr3):
        if arr1[i] == arr2[j] == arr3[k]:
            result.append(arr1[i])
            i += 1
            j += 1
            k += 1
        elif arr1[i] < arr2[j]:
            i += 1
        elif arr2[j] < arr3[k]:
            j += 1
        else:
            k += 1

    return result
```

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]

Soln:

```
def find_disjoint_elements(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)

    disjoint_nums1 = list(set1 - set2)
    disjoint_nums2 = list(set2 - set1)

    return [disjoint_nums1, disjoint_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]]

Soln:

```
def transpose(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    # Creating a new matrix with swapped rows and columns
    transposed = [[0 for _ in range(rows)] for _ in range(cols)]
    for i in range(rows):
        for j in range(cols):
            transposed[j][i] = matrix[i][j]

    return transposed
```

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.

Soln:

```
def array_pair_sum(nums):
    nums.sort()
    total_sum = 0

    for i in range(0, len(nums), 2):
        total_sum += nums[i]

    return total_sum
```

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*.

```
def arrange_coins(n):
    left, right = 1, n

    while left <= right:
        mid = left + (right - left) // 2
        coins_needed = (mid * (mid + 1)) // 2

        if coins_needed == n:
            return mid

        if coins_needed < n:
            left = mid + 1
        else:
            right = mid - 1

    return right
```

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]

Soln:

```
def sorted_squares(nums):
    n = len(nums)
    result = [0] * n
    left = 0
    right = n - 1
    index = n - 1

    while left <= right:
        left_square = nums[left] ** 2
        right_square = nums[right] ** 2

        if left_square > right_square:
            result[index] = left_square
            left += 1
        else:
            result[index] = right_square
            right -= 1

        index -= 1

    return result
```

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*

Soln:

```
def max_count(m, n, ops):

    min_a = m

    min_b = n



    for op in ops:

        min_a = min(min_a, op[0])

        min_b = min(min_b, op[1])



    return min_a * min_b
```

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].

Soln:

```
def shuffle(nums, n):

    result = []



    for i in range(n):

        result.append(nums[i])

        result.append(nums[n + i])



    return result
```