Assignment Questions 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.

In [1]:
```
def output_a(arr1, arr2, arr3):
    result = []
    ptr1 = ptr2 = ptr3 = 0

    while ptr1 < len(arr1) and ptr2 < len(arr2) and ptr3 < len(arr3):
        if arr1[ptr1] == arr2[ptr2] == arr3[ptr3]:
            result.append(arr1[ptr1])
            ptr1 += 1
            ptr2 += 1
            ptr3 += 1
        elif arr1[ptr1] < arr2[ptr2]:
            ptr1 += 1
        elif arr2[ptr2] < arr3[ptr3]:
            ptr2 += 1
        else:
            ptr3 += 1

    return result
arr1 = [1, 2, 3, 4, 5]
arr2 = [1, 2, 5, 7, 9]
arr3 = [1, 3, 4, 5, 8]

result = output_a(arr1, arr2, arr3)
print(result)
```

```
[1, 5]
```

In [ ]:

💡 **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].

In [2]:
```
def output_b(nums1, nums2):
    set1 = set(nums1)
    set2 = set(nums2)

    result = []
    result.append(list(set1 - set2))
    result.append(list(set2 - set1))

    return result
nums1 = [1, 2, 3]
nums2 = [2, 4, 6]

result = output_b(nums1, nums2)
print(result)
```

```
[[1, 3], [4, 6]]
```

In [ ]:

💡 **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]]

In [3]:
```
def output_c(matrix):
    rows = len(matrix)
    cols = len(matrix[0])

    TM = [[matrix[j][i] for j in range(rows)] for i in range(cols)]

    return TM
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

result = output_c(matrix)
print(result)
```

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

In [ ]:

💡 **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.

In [5]:
```
def output_d(nums):
    nums.sort()
    n1 = 0
    for i in range(0, len(nums), 2):
        n1 += nums[i]

    return n1
nums = [1, 4, 3, 2]
result = output_d(nums)
print(result)
```

```
4
```

In [ ]:

💡 **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:

[]()

![v2.jpg](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4bd91cfa-d2b1-47b3-8197-a72e8dcfff4b/v2.jpg)

Input: n = 5

Output: 2

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

In [6]:
```
def output_e(n):
    left, right = 0, n

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

        if curr == n:
            return mid

        if curr > n:
            right = mid - 1
        else:
            left = mid + 1

    return right
n = 5
result = output_e(n)
print(result)
```

```
2
```

In [ ]:

💡 **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]

In [7]:
```
def output_f(nums):
    n = len(nums)
    result = [0] * n
    left = 0
    right = n - 1
    idx = n - 1

    while left <= right:
        if abs(nums[left]) > abs(nums[right]):
            result[idx] = nums[left] ** 2
            left += 1
        else:
            result[idx] = nums[right] ** 2
            right -= 1
        idx -= 1

    return result
nums = [-4, -1, 0, 3, 10]
result = output_f(nums)
print(result)
```

```
[0, 1, 9, 16, 100]
```

In [ ]:
```

```

💡 **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.

In [8]:
```
def output_g(m, n, ops):
    min_row = m
    min_col = n

    for op in ops:
        min_row = min(min_row, op[0])
        min_col = min(min_col, op[1])

    return min_row * min_col
m = 3
n = 3
ops = [[2, 2], [3, 3]]
result = output_g(m, n, ops)
print(result)
```

```
4
```

In [ ]:

💡 **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].

In [9]:
```
def output_h(nums, n):
    result = []
    i, j = 0, n

    while i < n and j < 2 * n:
        result.append(nums[i])
        result.append(nums[j])
        i += 1
        j += 1

    return result
nums = [2, 5, 1, 3, 4, 7]
n = 3
result = output_h(nums, n)
print(result)
```

```
[2, 3, 5, 4, 1, 7]
```

In [ ]:

In [ ]:

In [ ]: