Q.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.

Ans: To find the integers that appeared in all three arrays, we can use the following approach:

1. Initialize three pointers, one for each array: `ptr1` for `arr1`, `ptr2` for `arr2`, and `ptr3` for `arr3`. Set them to the starting index of each array.

2. Initialize an empty result array to store the common integers.

3. Iterate while `ptr1`, `ptr2`, and `ptr3` are within the bounds of their respective arrays:

- If `arr1[ptr1]`, `arr2[ptr2]`, and `arr3[ptr3]` are equal, we have found a common integer. Add it to the result array and increment all three pointers.

- Otherwise, if any of the elements are smaller, we increment the corresponding pointer.

4. Return the result array.

Here's the implementation in Python:

```python

def find\_common\_elements(arr1, arr2, arr3):

ptr1 = ptr2 = ptr3 = 0

result = []

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

```

Let's test the function with the given example:

```python

arr1 = [1, 2, 3, 4, 5]

arr2 = [1, 2, 5, 7, 9]

arr3 = [1, 3, 4, 5, 8]

result = find\_common\_elements(arr1, arr2, arr3)

print(result)

```

Output:

```

[1, 5]

```

The function returns the expected output `[1, 5]`, which are the integers that appeared in all three arrays.

Q. 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].

Ans: To find the distinct integers that are present in one array but not in the other, we can use the following approach:

1. Create two sets: `set1` to store the unique integers in `nums1` and `set2` to store the unique integers in `nums2`.

2. Initialize two empty lists: `not\_in\_nums2` to store the distinct integers in `nums1` that are not in `nums2`, and `not\_in\_nums1` to store the distinct integers in `nums2` that are not in `nums1`.

3. Iterate over each element `num` in `nums1`:

- If `num` is not in `set2`, add it to `not\_in\_nums2`.

4. Iterate over each element `num` in `nums2`:

- If `num` is not in `set1`, add it to `not\_in\_nums1`.

5. Return a list containing `not\_in\_nums1` and `not\_in\_nums2`.

Here's the implementation in Python:

```python

def find\_disjoint(nums1, nums2):

set1 = set(nums1)

set2 = set(nums2)

not\_in\_nums2 = [num for num in nums1 if num not in set2]

not\_in\_nums1 = [num for num in nums2 if num not in set1]

return [not\_in\_nums1, not\_in\_nums2]

```

Let's test the function with the given example:

```python

nums1 = [1, 2, 3]

nums2 = [2, 4, 6]

result = find\_disjoint(nums1, nums2)

print(result)

```

Output:

```

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

```

Q. 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]]

Ans: Certainly! Here's the code to find the transpose of a matrix in Python:

```python

def transpose(matrix):

rows = len(matrix)

cols = len(matrix[0])

transposed = [[0] \* rows for \_ in range(cols)]

for i in range(rows):

for j in range(cols):

transposed[j][i] = matrix[i][j]

return transposed

# Example usage

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

result = transpose(matrix)

print(result)

```

Output:

```

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

```

The code correctly finds the transpose of the matrix `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]` and returns `[[1, 4, 7], [2, 5, 8], [3, 6, 9]]`.

Q. 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.

Ans: To maximize the sum of the minimum values in pairs, we can sort the array in ascending order and then pair the consecutive elements. Since we want to maximize the sum, we pair the smallest numbers together and continue with the larger numbers. The sum of the smaller elements will contribute more to the overall sum.

Here's the step-by-step approach:

1. Sort the given array `nums` in ascending order.

2. Initialize a variable `max\_sum` to 0 to store the maximum sum.

3. Iterate over the sorted array starting from index 0 with a step size of 2:

- Add the element at the current index to `max\_sum`.

4. Return `max\_sum`.

Here's the implementation in Python:

```python

def array\_pair\_sum(nums):

nums.sort()

max\_sum = 0

for i in range(0, len(nums), 2):

max\_sum += nums[i]

return max\_sum

```

Let's test the function with the given example:

```python

nums = [1, 4, 3, 2]

result = array\_pair\_sum(nums)

print(result)

```

Output:

```

4

```

The function returns the expected output `4`, which is the maximized sum of the minimum values in pairs `[1, 2]` and `[3, 4]`.

Q. 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

**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.

Ans: To determine the number of maximum integers in the matrix after performing all the operations, we can observe that the maximum integers will be present in the top-left corner of the matrix. The number of maximum integers will be determined by the minimum values of `ai` and `bi` among all the operations.

Here's the step-by-step approach:

1. Initialize `min\_row` and `min\_col` variables to `m` and `n`, respectively, to track the minimum values of `ai` and `bi`.

2. Iterate over each operation `[ai, bi]` in the `ops` array:

- Update `min\_row` with the minimum value of `ai` and `min\_col` with the minimum value of `bi`.

3. Return the product of `min\_row` and `min\_col`.

Here's the implementation in Python:

```python

def max\_count(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

```

Let's test the function with the given example:

```python

m = 3

n = 3

ops = [[2, 2], [3, 3]]

result = max\_count(m, n, ops)

print(result)

```

Output:

```

4

```

The function returns the expected output `4`, which is the number of maximum integers in the matrix after performing all the operations.