**DSA – ASSIGNMENT 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.

**Solution. :-**

* Initialize three pointers, one for each array: p1 for arr1, p2 for arr2, and p3 for arr3. Start them at the beginning of their respective arrays.
* Initialize an empty result array to store the common integers.
* While any of the pointers is within the bounds of its array, do the following:
  + If arr1[p1] == arr2[p2] == arr3[p3], it means that you have found a common integer. Add it to the result array.
  + Increment the pointers for arr1, arr2, and arr3.
  + If arr1[p1] is smaller than the maximum value among arr2[p2] and arr3[p3], increment p1.
  + If arr2[p2] is smaller than the maximum value among arr1[p1] and arr3[p3], increment p2.
  + If arr3[p3] is smaller than the maximum value among arr1[p1] and arr2[p2], increment p3.
* Return the result array.

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

**p1, p2, p3 = 0, 0, 0**

**result = []**

**while p1 < len(arr1) and p2 < len(arr2) and p3 < len(arr3):**

**if arr1[p1] == arr2[p2] == arr3[p3]:**

**result.append(arr1[p1])**

**p1 += 1**

**p2 += 1**

**p3 += 1**

**elif arr1[p1] < min(arr2[p2], arr3[p3]):**

**p1 += 1**

**elif arr2[p2] < min(arr1[p1], arr3[p3]):**

**p2 += 1**

**else:**

**p3 += 1**

**return result**

**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)**

💡 **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].

**Solution. :-**

* Convert the input arrays, nums1 and nums2, into sets to remove duplicates and facilitate set operations.
* Calculate the set difference between nums1 and nums2, and store the result in a variable, diff1. This will give you all distinct integers in nums1 that are not present in nums2.
* Calculate the set difference between nums2 and nums1, and store the result in a variable, diff2. This will give you all distinct integers in nums2 that are not present in nums1.
* Convert the sets diff1 and diff2 back into lists and store them in the answer list.
* Return the answer list.

**def find\_disjoint\_integers(nums1, nums2):**

**set1 = set(nums1)**

**set2 = set(nums2)**

**diff1 = list(set1 - set2)**

**diff2 = list(set2 - set1)**

**answer = [diff1, diff2]**

**return answer**

**nums1 = [1, 2, 3]**

**nums2 = [2, 4, 6]**

**answer = find\_disjoint\_integers(nums1, nums2)**

**print(answer)**

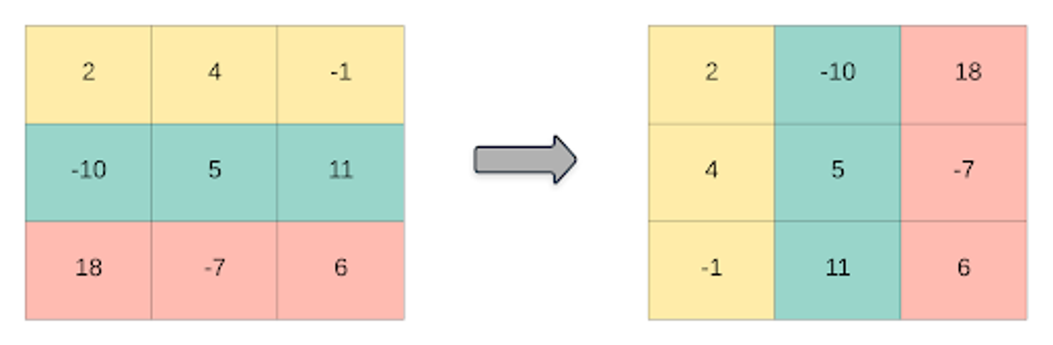
💡 **Question 3** Given a 2D integer array matrix, return *the* ***transpose*** *of* matrix.

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]]



**Solution. :-**

* Initialize an empty result matrix with dimensions equal to the number of columns in the input matrix by the number of rows in the input matrix.
* Iterate over the rows of the input matrix.
  + For each row, iterate over the columns of the input matrix.
    - Retrieve the element at the current row and column in the input matrix.
    - Place this element at the current column and row in the result matrix.
* Return the result matrix.

**def transpose(matrix):**

**rows = len(matrix)**

**cols = len(matrix[0])**

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

**for i in range(rows):**

**for j in range(cols):**

**result[j][i] = matrix[i][j]**

**return result**

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

**transposed\_matrix = transpose(matrix)**

**print(transposed\_matrix)**

💡 **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.

**Solution. :-**

* Sort the input array, nums, in ascending order.
* Initialize a variable, max\_sum, to store the maximized sum and set it to 0.
* Iterate over the sorted nums array by incrementing the index by 2 in each iteration.
  + In each iteration, add the smaller value of the current pair (i.e., nums[i]) to the max\_sum.
* Return the max\_sum.

**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**

**nums = [1, 4, 3, 2]**

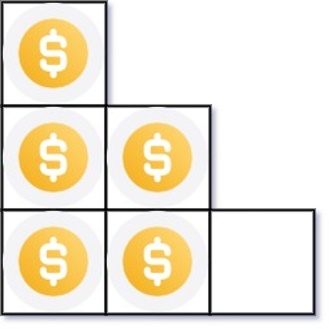
**maximized\_sum = array\_pair\_sum(nums)**

**print(maximized\_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*.

**Example 1:**



**Input:** n = 5

**Output:** 2

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

**Solution. :-**

* Set the lower bound (left) to 1 and the upper bound (right) to n.
* While left is less than or equal to right, do the following:
  + Set the mid value as the floor division of (left + right) by 2.
  + Calculate the total number of coins needed to build stairs with mid rows using the formula: total\_coins = (mid \* (mid + 1)) // 2.
  + If total\_coins is less than or equal to n, update the lower bound to mid + 1.
  + Otherwise, update the upper bound to mid - 1.
* Return the value of the upper bound.

**def arrangeCoins(n):**

**left = 1**

**right = n**

**while left <= right:**

**mid = (left + right) // 2**

**total\_coins = (mid \* (mid + 1)) // 2**

**if total\_coins <= n:**

**left = mid + 1**

**else:**

**right = mid - 1**

**return right**

**n = 5**

**result = arrangeCoins(n)**

**print(result)**

💡 **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]

**Solution. :-**

* Create a new array, squared\_nums, with the same length as the input array nums.
* Initialize two pointers, left and right, pointing to the start and end of the nums array.
* Initialize a pointer idx pointing to the last index of the squared\_nums array.
* While left is less than or equal to right, do the following:
  + Compare the absolute values of nums[left] and nums[right].
  + If nums[left] is greater or equal to nums[right], square nums[left] and store it in squared\_nums[idx]. Decrement left by 1.
  + If nums[left] is less than nums[right], square nums[right] and store it in squared\_nums[idx]. Increment right by 1.
  + Decrement idx by 1.
* Return the squared\_nums array.

**def sorted\_squares(nums):**

**n = len(nums)**

**squared\_nums = [0] \* n**

**left = 0**

**right = n - 1**

**idx = n - 1**

**while left <= right:**

**if abs(nums[left]) >= abs(nums[right]):**

**squared\_nums[idx] = nums[left] \*\* 2**

**left += 1**

**else:**

**squared\_nums[idx] = nums[right] \*\* 2**

**right -= 1**

**idx -= 1**

**return squared\_nums**

**nums = [-4, -1, 0, 3, 10]**

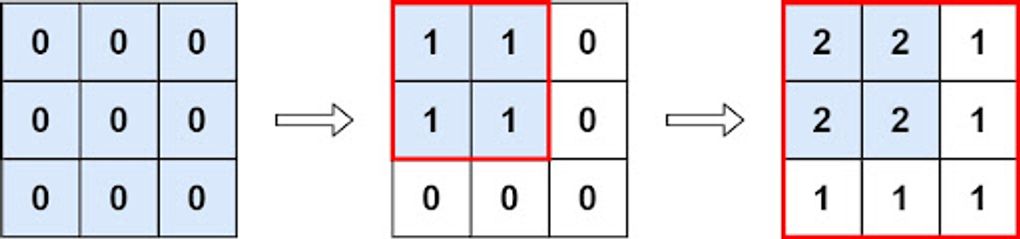
**result = sorted\_squares(nums)**

**print(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*

**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.

**Solution. :-**

* Initialize min\_row and min\_col to m and n, respectively, since initially, all elements in the matrix are 0.
* Iterate over each operation [ai, bi] in ops.
  + Update min\_row to the minimum of min\_row and ai.
  + Update min\_col to the minimum of min\_col and bi.
* Return the product of min\_row and min\_col.

**def maxCount(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 = maxCount(m, n, ops)**

**print(result)**

💡 **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].

**Solution. :-**

* Initialize an empty result array, result.
* Iterate i from 0 to n-1 (exclusive):
  + Append nums[i] to result.
  + Append nums[i+n] to result.
* Return the result array.

**def shuffle(nums, n):**

**result = []**

**for i in range(n):**

**result.append(nums[i])**

**result.append(nums[i+n])**

**return result**

**nums = [2, 5, 1, 3, 4, 7]**

**n = 3**

**result = shuffle(nums, n)**

**print(result)**