**Assignment 4 Questions - 2D Arrays | DSA**

**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]

**------------**

**Solution :** class Solution(object):

**------------** def arraysIntersection(self, arr1: List[int], arr2: List[int], arr3: List[int]) -> List[int]:

def find(arr, val):

left, right = 0, len(arr) - 1

while left < right:

mid = (left + right) >> 1

if arr[mid] >= val:

right = mid

else:

left = mid + 1

return arr[left] == val

res = []

for num in arr1:

if find(arr2, num) and find(arr3, num):

res.append(num)

return res

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(N), where N is the maximum size among the sizes of the arrays. In the worst case we traverse till the array having the maximum size is evaluated.
* **Space Complexity :** O(1). We only use some variables that occupy constant extra space.

====================================x===================================

**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]]

**------------**

**Solution :** class Solution():

**------------** def findDifference(self, nums1: List[int], nums2: List[int]) -> List[List[int]]:

duplicated = []

results = [[], []]

sorted(nums1)

sorted(nums2)

# Drop duplicated

nums1 = set(nums1)

nums2 = set(nums2)

# Find difference

for n1 in nums1:

for n2 in nums2:

if n1 == n2:

duplicated.append(n1)

break

for n1 in nums1:

if n1 not in duplicated:

results[0].append(n1)

for n2 in nums2:

if n2 not in duplicated:

results[1].append(n2)

return results

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(N∗M). In the first call to getElementsOnlyInFirstList, we iterate over the first list and, for each element, iterate over the second, which costs us N∗M operations. Then again, doing it for the other pair (nums2, nums1), the total operations will be M∗N. Hence the total time complexity would be O(N∗M).
* Space complexity : O(1).

The only space required (two lists of size N and M) is to store the output list that is not considered part of the space complexity. Hence, the total space complexity would be constant.

====================================x===================================

**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 :** class Solution(object):

**------------** def transpose(self, A):

R, C = len(A), len(A[0])

ans = [[None] \* R for \_ in xrange(C)]

for r, row in enumerate(A):

for c, val in enumerate(row):

ans[c][r] = val

return ans

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(R∗C), where RRR and CCC are the number of rows and columns in the given matrix A.
* Space Complexity: O(R∗C)O(R \* C)O(R∗C), the space used by the answer.

====================================x===================================

**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

**------------**

**Solution :** class Solution():

**------------** def arrayPairSum(self, nums: List[int]) -> int:

nums.sort()

r=0

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

r+=min(nums[i],nums[i+1])

return(r)

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(nlogn). Sorting will take O(nlogn) complexity.
* Space Complexity : O(1).

====================================x===================================

**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

**------------**

**Solution :** class Solution():

**------------**  def arrangeCoins(self, n: int) -> int:

left, right = 0, n

while left <= right:

k = (right + left) // 2

curr = k \* (k + 1) // 2

if curr == n:

return k

if n < curr:

right = k - 1

else:

left = k + 1

return right

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(log N).
* Space Complexity : O(1).

====================================x===================================

**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]

**------------**

**Solution :** class Solution():

**------------** def sortedSquares(self, A: List[int]) -> List[int]:

for i in range(len(A)):

A[i] \*= A[i]

A.sort()

return A

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(N log N), where N is the length of A.
* Space Complexity : O(N).

====================================x===================================

**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

**------------**

**Solution :** class Solution():

**------------** def maxCount(self, m: int, n: int, ops: List[List[int]]) -> int:

length = len(ops)

if length == 0:

return m\*n

result = [ops[0][0] , ops[0][1]]

for i in range(1,length):

result[0] = min(result[0] , ops[i][0])

result[1] = min(result[1] , ops[i][1])

return result[0]\*result[1]

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(N).
* Space Complexity : O(1).

====================================x===================================

**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]

**------------**

**Solution :** class Solution():

**------------** def shuffle(self, nums: List[int], n: int) -> List[int]:

result = [0] \* (2 \* n)

for i in range(n):

result[2 \* i] = nums[i]

result[2 \* i + 1] = nums[n + i]

return result

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(N).
* We iterate on nnn elements of the nums array, which takes us O(n)O(n)O(n) time.
* Initializing the result array will take O(2n)O(2n)O(2n) time.
* Thus, overall we take O(n+2n)=O(n)O(n + 2n) = O(n)O(n+2n)=O(n) time.
* Space Complexity : O(1).
* We are not using any additional space other than the output array.

====================================x===================================