Q1.Convert 1D Array Into 2D Array

You are given a \*0-indexed\* 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with  m rows and n columns using \*all\* the elements from original.

The elements from indices 0 to n - 1 (\*inclusive) of original should form the first row of the constructed 2D array, the elements from indices n to 2 \* n - 1 (inclusive\*) should form the second row of the constructed 2D array, and so on.

Return an m x n 2D array constructed according to the above procedure, or an empty 2D array if it is impossible.

class Solution:

def construct2DArray(self, original: List[int], m: int, n: int) -> List[List[int]]:

if m \* n != len(original):

return []

ans = [[0] \* n for \_ in range(m)]

idx = 0

for i in range(m):

for j in range(n):

ans[i][j] = original[idx]

idx += 1

return ans

Q2. 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\*.

class Solution:

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

return int((-1+ math.sqrt(1+8\*n))//2)

Q3. Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

class Solution:

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

# Two Pointer Approach

n = len(nums)

l, r = 0, n - 1

k = n - 1

ans = [0] \* n

while k >= 0:

if abs(nums[l]) > nums[r]:

ans[k] = nums[l] \* nums[l]

l += 1

else:

ans[k] = nums[r] \* nums[r]

r -= 1

k -= 1

return ans

Q4. 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.

class Solution:

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

n1=set(nums1)

n2=set(nums2)

r1=list(set(x for x in nums1 if x not in n2))

r2=list(set(x for x in nums2 if x not in n1))

return [r1,r2]

Q5. Given two integer arrays arr1 and arr2, and the integer d, return the distance value between the two arrays.

The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

class Solution:

def findTheDistanceValue(self, arr1: List[int], arr2: List[int], d: int) -> int:

arr2.sort()

distance = len(arr1)

for num in arr1:

start = 0

end = len(arr2) - 1

while start <= end:

mid = (start+end)//2

if abs(num- arr2[mid]) <= d:

distance -= 1

break

elif arr2[mid] > num :

end = mid-1

elif arr2[mid] < num :

start = mid+1

return distance

Q6. Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears \*once\* or \*twice, return \*an array of all the integers that appears \*\*twice\*\*.You must write an algorithm that runs in O(n) time and uses only constant extra space.

class Solution:

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

ans = []

for i in range(len(nums)):

if nums[abs(nums[i])-1]>0:

nums[abs(nums[i])-1] = -nums[abs(nums[i])-1]

else:

ans.append(abs(nums[i]))

return ans

Q7. Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:

class Solution:

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

low=0

high=len(nums)-1

res=nums[0]

while low<=high:

if nums[low]<nums[high]:

res=min(res,nums[low])

break

mid=(low+high)//2

res=min(res,nums[mid])

if nums[mid]>=nums[low]:

low=mid+1

else:

high=mid-1

return res

Q8. An integer array original is transformed into a \*doubled\* array changed by appending \*twice the value\* of every element in original, and then randomly \*shuffling\* the resulting array.

Given an array changed, return original if changed is a \*\*doubled\* array. If\* changed is not a \*\*doubled\* array, return an empty array. The elements in\* original may be returned in \*\*any\* order\*.

class Solution:

def findOriginalArray(self, changed):

cnt, ans = Counter(changed), []

if len(changed) % 2: return []

for x in sorted(cnt.keys()):

if cnt[x] > cnt[x \* 2]: return []

if x == 0:

if cnt[x] % 2:

return []

else:

ans += [0] \* (cnt[x] // 2)

else:

ans += [x] \* cnt[x]

cnt[2 \* x] -= cnt[x]

return ans