Q1

class Solution:

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

Q2

class Solution:

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

num1,num2 = set(nums1),set(nums2)

res1,res2 = set(), set()

for n in nums1:

if n not in num2:

res1.add(n)

for n in nums2:

if n not in num1:

res2.add(n)

return [list(res1),list(res2)]

Q3

def transpose(A):

R = len(A)

C = len(A[0])

ans = [[0 for \_ in range(R)] for \_ in range(C)]

for r in range(R):

for c in range(C):

ans[c][r] = A[r][c]

return ans

Q4

class Solution:

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

       nums.sort()

       max\_sum=0

       for i in range(0,len(nums),2):

            max\_sum+=nums[i]

       return max\_sum

Q5

class Solution:

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

# Initialize pointers to first and last possible row lengths

left, right = 1, n

while left <= right:

# Compute the midpoint between left and right

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

# Compute the total number of coins needed for mid complete rows

coins = (mid \* (mid + 1)) // 2

# If we have enough coins, look for a smaller number of rows

if coins <= n:

left = mid + 1

# Otherwise, look for a larger number of rows

else:

right = mid - 1

# Return the number of complete rows (i.e., right pointer)

return right

Q6

class Solution:

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

return sorted([num\*num for num in nums])

Q7

class Solution:

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

final\_m = m

final\_n = n

for i in ops:

current\_increment\_matrix = i

current\_m = i[0]

current\_n = i[1]

if current\_m < final\_m:

final\_m = current\_m

if current\_n < final\_n:

final\_n = current\_n

print(final\_m,final\_n)

return final\_m \* final\_n

Q8

class Solution:

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

res = []

for i in range(n):

res.append(nums[i])

res.append(nums[i+n])

return res