**Assignment – 6**

Q1. A permutation perm of n + 1 integers of all the integers in the range [0, n] can be represented as a string s of length n where:

* s[i] == 'I' if perm[i] < perm[i + 1], and
* s[i] == 'D' if perm[i] > perm[i + 1].

Given a string s, reconstruct the permutation perm and return it. If there are multiple valid permutations perm, return **any of them**.

**Example 1:**

**Input:** s = "IDID"

**Output:**

[0,4,1,3,2]

Sol1.

def findPermutation(s):

n = len(s)

perm = []

curr = 0

for ch in s:

if ch == 'I':

perm.append(curr)

curr += 1

elif ch == 'D':

perm.append(n)

n -= 1

perm.append(curr)

return perm

Q2. You are given an m x n integer matrix matrix with the following two properties:

* Each row is sorted in non-decreasing order.
* The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if* target *is in* matrix *or* false *otherwise*.

You must write a solution in O(log(m \* n)) time complexity.

**Example 1:**

**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

**Output:** true

Sol2.

def searchMatrix(matrix, target):

m = len(matrix)

n = len(matrix[0])

left = 0

right = m \* n - 1

while left <= right:

mid = (left + right) // 2

row = mid // n

col = mid % n

if matrix[row][col] == target:

return True

elif matrix[row][col] < target:

left = mid + 1

else:

right = mid - 1

return False

Q3. Given an array of integers arr, return *true if and only if it is a valid mountain array*.

Recall that arr is a mountain array if and only if:

* arr.length >= 3
* There exists some i with 0 < i < arr.length - 1 such that:
  + arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
  + arr[i] > arr[i + 1] > ... > arr[arr.length - 1]
* **xample 1:**
* **Input:** arr = [2,1]
* **Output:**
* false

Sol.

def validMountainArray(arr):

n = len(arr)

if n < 3:

return False

i = 1

while i < n and arr[i] > arr[i - 1]:

i += 1

if i == 1 or i == n:

return False

while i < n and arr[i] < arr[i - 1]:

i += 1

return i == n

Q4. Given a binary array nums, return *the maximum length of a contiguous subarray with an equal number of* 0 *and* 1.

**Example 1:**

**Input:** nums = [0,1]

**Output:** 2

**Explanation:**

[0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

Sol.

def findMaxLength(nums):

max\_length = 0

count = 0

count\_dict = {0: -1} # Initialize with a count of 0 at index -1

for i in range(len(nums)):

count += 1 if nums[i] == 1 else -1

if count in count\_dict:

max\_length = max(max\_length, i - count\_dict[count])

else:

count\_dict[count] = i

return max\_length

Q5. The **product sum** of two equal-length arrays a and b is equal to the sum of a[i] \* b[i] for all 0 <= i < a.length (**0-indexed**).

* For example, if a = [1,2,3,4] and b = [5,2,3,1], the **product sum** would be 1*5 + 2*2 + 3*3 + 4*1 = 22.

Given two arrays nums1 and nums2 of length n, return *the* ***minimum product sum*** *if you are allowed to* ***rearrange*** *the* ***order*** *of the elements in* nums1.

**Example 1:**

**Input:** nums1 = [5,3,4,2], nums2 = [4,2,2,5]

**Output:** 40

**Explanation:**

We can rearrange nums1 to become [3,5,4,2]. The product sum of [3,5,4,2] and [4,2,2,5] is 3*4 + 5*2 + 4*2 + 2*5 = 40.

Sol. def minProductSum(nums1, nums2):

nums1.sort()

nums2.sort(reverse=True)

product\_sum = 0

for i in range(len(nums1)):

product\_sum += nums1[i] \* nums2[i]

return product\_sum

Q6. 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*.

**Example 1:**

**Input:** changed = [1,3,4,2,6,8]

**Output:** [1,3,4]

**Explanation:** One possible original array could be [1,3,4]:

* Twice the value of 1 is 1 \* 2 = 2.
* Twice the value of 3 is 3 \* 2 = 6.
* Twice the value of 4 is 4 \* 2 = 8.

Other original arrays could be [4,3,1] or [3,1,4].

Sol.

def findOriginalArray(changed):

original = sorted(changed.copy())

for num in original:

if num % 2 == 0 and num // 2 in original:

original.remove(num // 2)

else:

return []

return original

Q7. Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

**Example 1:**

**nput:** n = 3

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

Sol.

def generateMatrix(n):

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

start\_row, end\_row = 0, n - 1

start\_col, end\_col = 0, n - 1

num = 1

while num <= n \* n:

for j in range(start\_col, end\_col + 1):

matrix[start\_row][j] = num

num += 1

start\_row += 1

for i in range(start\_row, end\_row + 1):

matrix[i][end\_col] = num

num += 1

end\_col -= 1

for j in range(end\_col, start\_col - 1, -1):

matrix[end\_row][j] = num

num += 1

end\_row -= 1

for i in range(end\_row, start\_row - 1, -1):

matrix[i][start\_col] = num

num += 1

start\_col += 1

return matrix

Q8. Given two [sparse matrices](https://en.wikipedia.org/wiki/Sparse_matrix) mat1 of size m x k and mat2 of size k x n, return the result of mat1 x mat2. You may assume that multiplication is always possible.

**Example 1:**

**Input:** mat1 = [[1,0,0],[-1,0,3]], mat2 = [[7,0,0],[0,0,0],[0,0,1]]

**Output:**

[[7,0,0],[-7,0,3]]

Sol. def multiply(mat1, mat2):

m, k, n = len(mat1), len(mat1[0]), len(mat2[0])

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

for i in range(m):

for val, col in enumerate(mat1[i]):

if val != 0:

for j in range(n):

result[i][j] += val \* mat2[col][j]

return result