Assignment 6 - 2D Arrays

💡 **Question 1**

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]

***Solution –***

def findPermutation(s):

    n = len(s)

    perm = []

    low, high = 0, n

    for c in s:

        if c == 'I':

            perm.append(low)

            low += 1

        elif c == 'D':

            perm.append(high)

            high -= 1

    # Append the remaining number

    perm.append(low)

    return perm

s = "IDID"

perm = findPermutation(s)

print(perm)

💡 **Question 2**

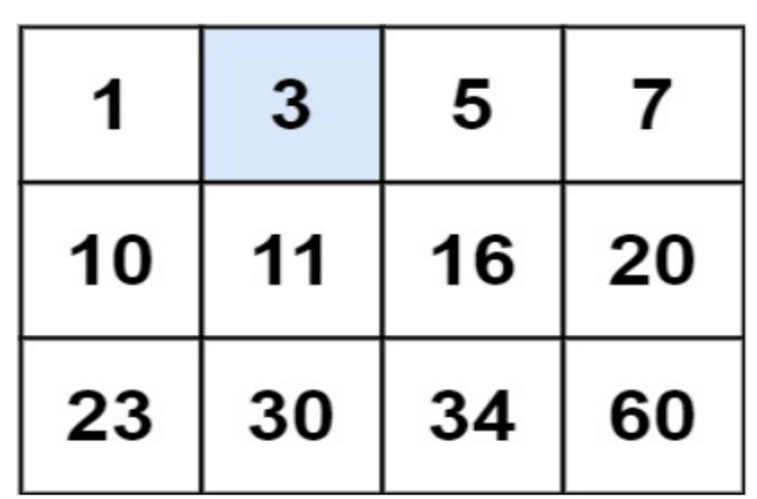
You are given an m x n integer matrix matrix with the following two properties:

* 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

***Solution –***

def searchMatrix(matrix, target):

    m, n = len(matrix), len(matrix[0])

    left, right = 0, 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

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

target = 3

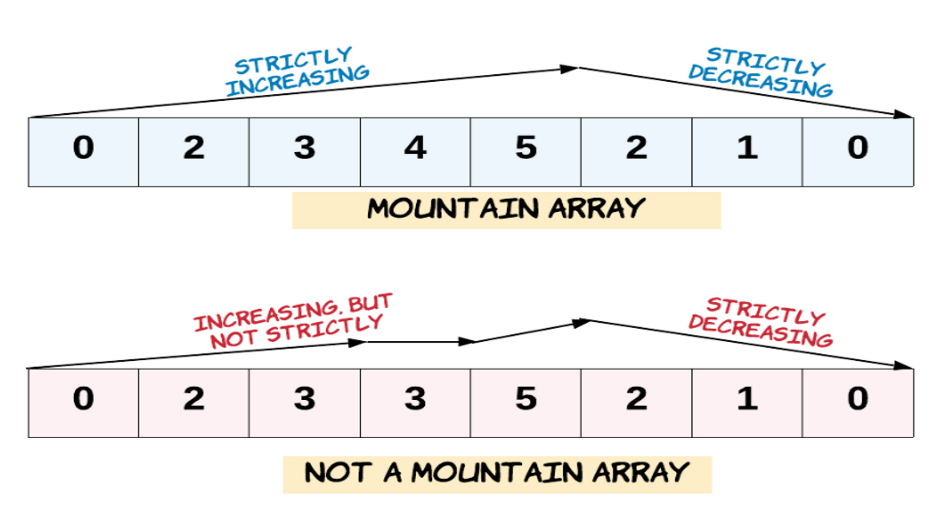
print(searchMatrix(matrix, target))

💡 **Question 3**

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] </aside>



**Example 1:**

**Input:** arr = [2,1]

**Output:**

False

***Solution –***

def validMountainArray(arr):

    if len(arr) < 3:

        return False

    i = 1

    while i < len(arr) and arr[i] > arr[i - 1]:

        i += 1

    if i == 1 or i == len(arr):

        return False

    while i < len(arr) and arr[i] < arr[i - 1]:

        i += 1

    return i == len(arr)

# Example test case

arr = [2, 1]

print(validMountainArray(arr))

False

💡 **Question 4**

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.

***Solution –***

def findMaxLength(nums):

    count = 0

    max\_length = 0

    count\_dict = {0: -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

nums = [0, 1]

print(findMaxLength(nums))

💡 **Question 5**

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.

***Solution –***

def minProductSum(nums1, nums2):

    nums1.sort()

    nums2.sort(reverse=True)

    min\_product\_sum = float('inf')

    for i in range(len(nums1)):

        min\_product\_sum = min(min\_product\_sum, nums1[i] \* nums2[i])

    return min\_product\_sum

# Example test case

nums1 = [5, 3, 4, 2]

nums2 = [4, 2, 2, 5]

print(minProductSum(nums1, nums2))

💡 **Question 6**

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].

***Solution –***

def findOriginalArray(changed):

    changed.sort()

    original = []

    for num in changed:

        if num / 2 in original:

            original.remove(num / 2)

        elif num % 2 == 0 and num / 2 not in original:

            original.append(num / 2)

        else:

            return []

    return original

# Example test case

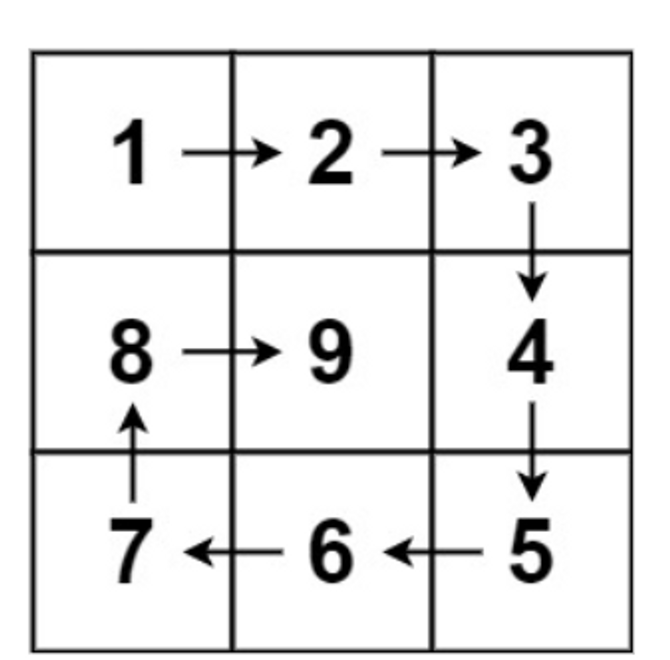
changed = [1, 3, 4, 2, 6, 8]

print(findOriginalArray(changed))

💡 **Question 7**

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

**Example 1:**



**Input:** n = 3

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

***Solution –***

def generateMatrix(n):

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

    left, right, top, bottom = 0, n - 1, 0, n - 1

    num = 1

    while num <= n \* n:

        # Traverse top row

        for i in range(left, right + 1):

            matrix[top][i] = num

            num += 1

        top += 1

        # Traverse right column

        for i in range(top, bottom + 1):

            matrix[i][right] = num

            num += 1

        right -= 1

        # Traverse bottom row

        for i in range(right, left - 1, -1):

            matrix[bottom][i] = num

            num += 1

        bottom -= 1

        # Traverse left column

        for i in range(bottom, top - 1, -1):

            matrix[i][left] = num

            num += 1

        left += 1

    return matrix

# Example test case

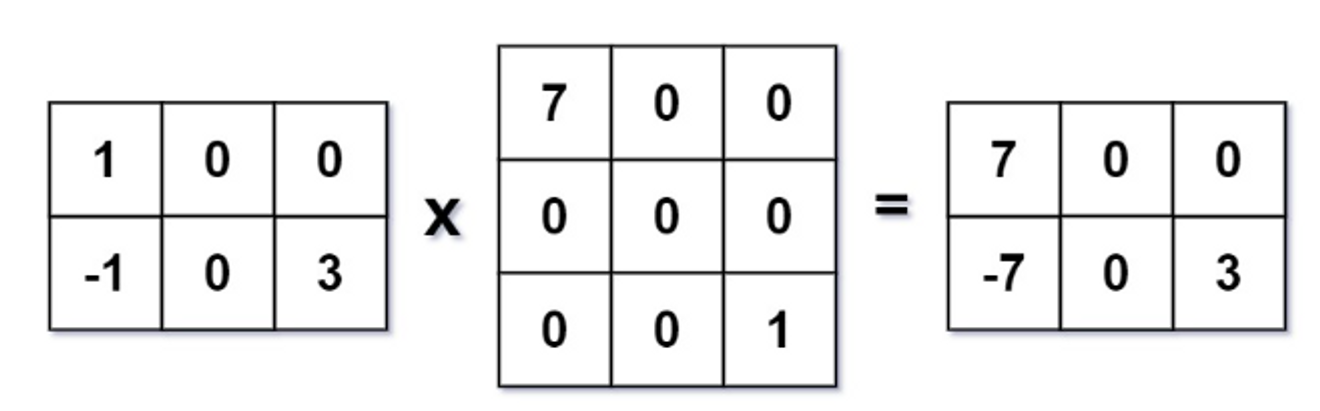
n = 3

print(generateMatrix(n))

💡 **Question 8**

Given two sparse matrices 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]]

***Solution –***

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 j in range(n):

            for x in range(k):

                result[i][j] += mat1[i][x] \* mat2[x][j]

    return result

# Example test case

mat1 = [[1, 0, 0], [-1, 0, 3]]

mat2 = [[7, 0, 0], [0, 0, 0], [0, 0, 1]]

print(multiply(mat1, mat2))