ASSIGNMENT-VI

**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]

**Ans:** def reconstruct\_permutation(s):

n = len(s)

result = []

available = list(range(n + 1))

for char in s:

if char == 'I':

result.append(available.pop(0))

elif char == 'D':

result.append(available.pop())

result.append(available[0])

return result

**Question 2**

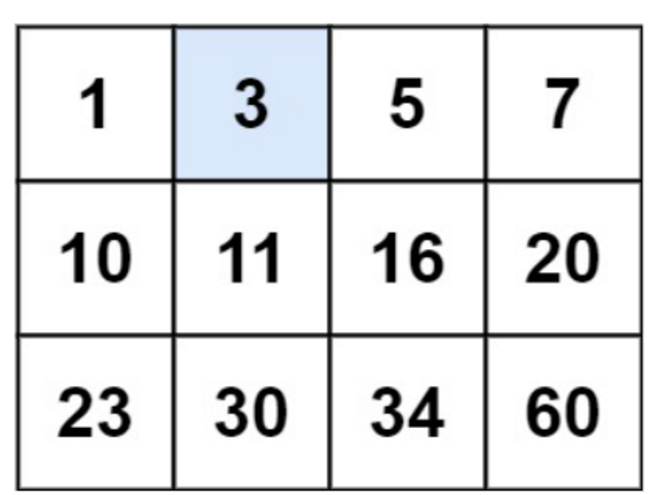
You are given an m x n integer matrix matrix with the following two properties:

* 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

**Ans:** def search\_matrix(matrix, target):

if not matrix or not matrix[0]:

return False

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

left, right = 0, m \* n - 1

while left <= right:

mid = (left + right) // 2

mid\_value = matrix[mid // n][mid % n]

if mid\_value == target:

return True

elif mid\_value < target:

left = mid + 1

else:

right = mid - 1

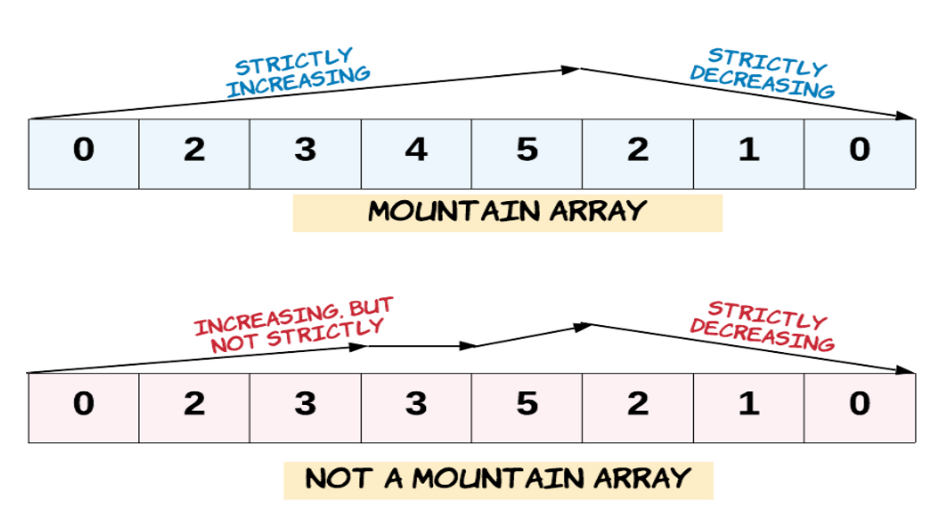
return False

**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

**Ans**: def valid\_mountain\_array(arr):

n = len(arr)

if n < 3:

return False

peak\_index = arr.index(max(arr))

if peak\_index == 0 or peak\_index == n - 1:

return False

for i in range(peak\_index):

if arr[i] >= arr[i + 1]:

return False

for i in range(peak\_index, n - 1):

if arr[i] <= arr[i + 1]:

return False

return True

**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.

**Ans**: def findMaxLength(nums):

n = len(nums)

count\_dict = {0: -1}

max\_length = 0

count = 0

for i in range(n):

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

**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.

**Ans**: def minProductSum(nums1, nums2):

nums1.sort()

nums2.sort(reverse=True)

min\_product\_sum = float('inf')

for i in range(len(nums1)):

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

min\_product\_sum = min(min\_product\_sum, product\_sum)

return min\_product\_sum

**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].

**Ans**: def reconstructOriginal(changed):

freq\_map = {}

for num in changed:

freq\_map[num] = freq\_map.get(num, 0) + 1

original = []

for num in changed:

half = num // 2

if half in freq\_map and freq\_map[half] > 0:

freq\_map[half] -= 1

original.append(half)

else:

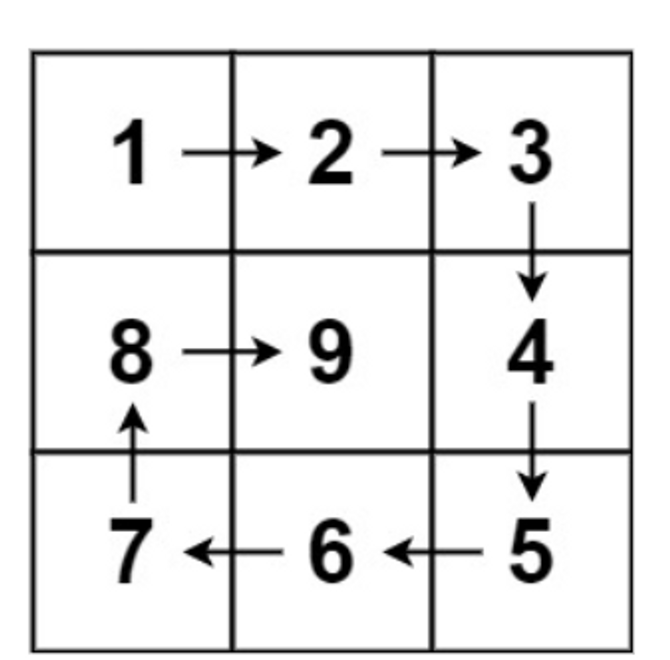
return []

return original

**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]]

**Ans:** def generateMatrix(n):

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

startRow, endRow = 0, n - 1

startCol, endCol = 0, n - 1

num = 1

while startRow <= endRow and startCol <= endCol:

for j in range(startCol, endCol + 1):

matrix[startRow][j] = num

num += 1

startRow += 1

for i in range(startRow, endRow + 1):

matrix[i][endCol] = num

num += 1

endCol -= 1

if startRow <= endRow:

for j in range(endCol, startCol - 1, -1):

matrix[endRow][j] = num

num += 1

endRow -= 1

if startCol <= endCol

for i in range(endRow, startRow - 1, -1):

matrix[i][startCol] = num

num += 1

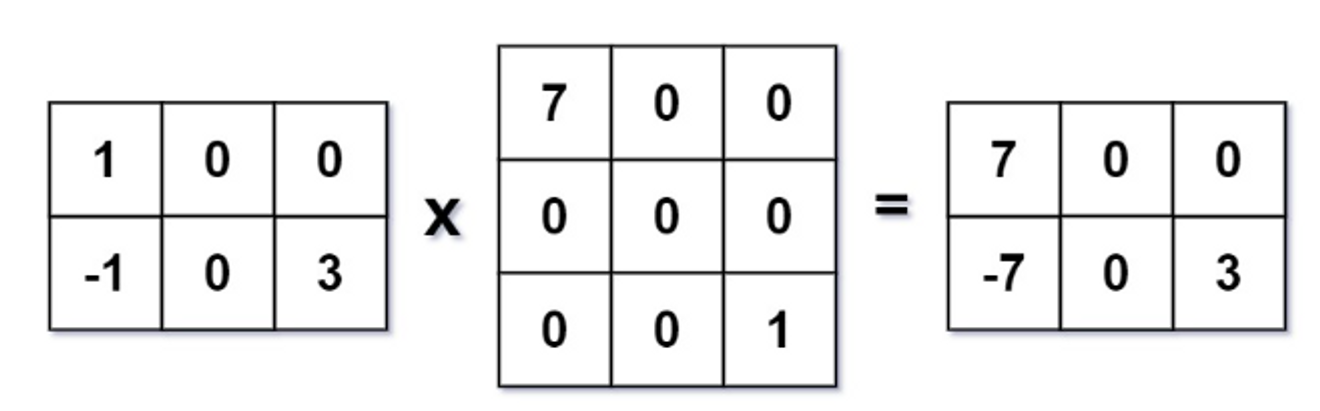
startCol += 1

return matrix

**Question 8**

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]]

**Ans:**

def multiply(mat1, mat2):

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

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

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

for i in range(m):

for j in range(k):

if mat1[i][j] != 0:

for col in range(n):

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

return result