**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.** Solution from my leetcode : - <https://leetcode.com/gopsa2001/>

class Solution:

    def diStringMatch(self, s: str) -> List[int]:

        l=[]

        ln=len(s)

        k=0

        for i in range(ln):

            if s[i]=='I':

                 l.append(k)

                 k+=1

            else:

                l.append(ln)

                ln-=1

        l.append(k)

        return l

**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.** Solution from my leetcode : - <https://leetcode.com/gopsa2001/>

class Solution:

    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:

        for i in matrix:

            if target in i:

                return True

        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.** Solution from my leetcode : - <https://leetcode.com/gopsa2001/>

class Solution:

    def validMountainArray(self, arr: List[int]) -> bool:

        c=0

        if len(arr)==2 or len(arr)==1 :

            return False

        for i in range(len(arr)-1):

            if arr[i]<arr[i+1]:

                c+=1

            else:break

        for j  in range(c,len(arr)-1):

            if arr[j]<=arr[j+1]:

                 return False

        if c==len(arr)-1 or c==0:

            return False

        else: 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.** Solution from my leetcode : - <https://leetcode.com/gopsa2001/>

class Solution:

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

        m,c=0,0

        d={0:-1}

        for i in range(len(nums)):

            if nums[i]==0:

                c-=1

            else:

                c+=1

            if c in d:

                m=max(m,i-d[c])

            else:

                d[c]=i

        return m

**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 15 + 22 + 33 + 41 = 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 34 + 52 + 42 + 25 = 40.

**Ans.**

class Solution:

def minValue(self, a, b, n):

# Your code goes here

sm=0

a.sort()

b.sort(reverse=True)

for i in range (n):

sm=sm+a[i]\*b[I]

return sm

**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.** Solution from my leetcode : - <https://leetcode.com/gopsa2001/>

class Solution:

    def findOriginalArray(self, changed: List[int]) -> List[int]:

        c = Counter(changed)

        if c[0]%2:

            return []

        for x in sorted(c):

            if c[x] > c[2\*x]:

                return []

            c[2\*x] -=c[x] if x else c[x]//2

        return list(c.elements())

**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.** Solution from my leetcode : - <https://leetcode.com/gopsa2001/>

class Solution:

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

        r1 = 0

        r2 = n -1

        c1 = 0

        c2 = n -1

        res = [[0 for i in range(n)] for i in range(n)]

        print(res)

        k = 1

        while(r1<=r2 and c1 <= c2):

            for i in range(r1,r2+1):

                res[r1][i] = k

                k+=1

            for i in range(c1+1,c2+1):

                res[i][c2]=k

                k+=1

            for i in range(r2-1,r1-1,-1):

                res[c2][i] = k

                k+=1

            for i in range(c2-1,c1,-1):

                res[i][c1] = k

                k+=1

            r1+=1

            c1+=1

            c2-=1

            r2-=1

        return res

**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.**

class Solution:

  def multiply(self, mat1: List[List[int]], mat2: List[List[int]]) -> List[List[int]]:

    m = len(mat1)

    n = len(mat2)

    l = len(mat2[0])

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

    for i in range(m):

      for j in range(l):

        for k in range(n):

          ans[i][j] += mat1[i][k] \* mat2[k][j]

    return ans