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]

Soln:

```
def reconstruct_permutation(s):
    n = len(s)
    perm = []
    start = 0
    end = n

    for c in s:
        if c == 'I':
            perm.append(start)
            start += 1
        elif c == 'D':
            perm.append(end)
            end -= 1

    perm.append(start)

    return 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.

Soln:

```
def searchMatrix(matrix, target):
    m = len(matrix)
    n = len(matrix[0]) if m > 0 else 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
```

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]

Soln:

```
def validMountainArray(arr):
    n = len(arr)
    if n < 3:
        return False

    left = 0
    right = n - 1

    while left < right:
        if arr[left] < arr[left + 1]:
            left += 1
        elif arr[right] < arr[right - 1]:
            right -= 1
        else:
            break

    return left == right and left != 0 and right != n - 1
```

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.

Soln:

```
def findMaxLength(nums):
    max_len = 0
    count = 0
    count_map = {0: -1}

    for i in range(len(nums)):
        count += 1 if nums[i] == 1 else -1

        if count in count_map:
            curr_len = i - count_map[count]
            max_len = max(max_len, curr_len)
        else:
            count_map[count] = i

    return max_len
```

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.

Soln:

```
def minProductSum(nums1, nums2):
    nums1.sort()
    nums2.sort(reverse=True)
    min_product_sum = 0

    for i in range(len(nums1)):
        min_product_sum += nums1[i] * nums2[i]

    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].

Soln:

```
def findOriginalArray(changed):
    freqMap = {}
    original = []

    for num in changed:
        freqMap[num] = freqMap.get(num, 0) + 1

    for num in changed:
        if freqMap.get(num, 0) == 0:
            continue
        if freqMap.get(num * 2, 0) == 0:
            return []
        original.append(num)
        freqMap[num] -= 1
        freqMap[num * 2] -= 1

    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.

Soln:

```
def generateMatrix(n):
    rowStart = 0
    rowEnd = n - 1
    colStart = 0
    colEnd = n - 1
    num = 1
    matrix = [[0] * n for _ in range(n)]


    while num <= n * n:
        # Traverse top row
        for col in range(colStart, colEnd + 1):
            matrix[rowStart][col] = num
            num += 1
        rowStart += 1


        # Traverse right column
        for row in range(rowStart, rowEnd + 1):
            matrix[row][colEnd] = num
            num += 1
        colEnd -= 1
```

```
        # Traverse bottom row
        for col in range(colEnd, colStart - 1, -1):
            matrix[rowEnd][col] = num
            num += 1
        rowEnd -= 1

        # Traverse left column
        for row in range(rowEnd, rowStart - 1, -1):
            matrix[row][colStart] = num
            num += 1
        colStart += 1

    return matrix
```

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.

Soln:

```
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
```