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**.

```
def reconstructPermutation(s):

    n = len(s)

    perm = []

    low, high = 0, n

    for ch in s:

        if ch == 'I':

            perm.append(low)

            low += 1

        elif ch == 'D':

            perm.append(high)

            high -= 1

    perm.append(low if s[-1] == 'I' else high)

    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.

```
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, col = mid // n, 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]

```
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
```

Question 4

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

```
def findMaxLength(nums):
    max_length = 0
    count = 0
    prefix_sums = {0: -1}
    for i in range(len(nums)):
        if nums[i] == 1:
            count += 1
        else:
            count -= 1
```

```
        if count == 0:
            max_length = i + 1
        elif count in prefix_sums:
            max_length = max(max_length, i - prefix_sums[count])
        else:
            prefix_sums[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.

```
def minProductSum(nums1, nums2):
    nums1.sort()  # Sort nums1 in non-decreasing order
    nums2.sort(reverse=True)  # Sort nums2 in non-increasing order
    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*.

```
def findOriginalArray(changed):
    count = {}
    for num in changed:
        if num not in count:
            count[num] = 1
        else:
            count[num] += 1
    original = []
    for num, cnt in count.items():
        if cnt % 2 != 0:
            return []
        original.extend([num] * (cnt // 2))
    return original
```