**Assignment Questions 6**

**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]

**CODE:**

**def** reconstructPermutation(s):

perm **=** []

start, end **=** 0, len(s)

**for** ch **in** s:

**if** ch **==** 'I':

perm**.**append(start)

start **+=** 1

**elif** ch **==** '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.

**Example 1:**

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

**Output:** true

**CODE:**

**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

num **=** matrix[row][col]

**if** num **==** target:

**return** **True**

**elif** num **<** 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]

**Example 1:**

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

**Output:**

False

**CODE:**

**def** validMountainArray(arr):

n **=** len(arr)

**if** n **<** 3:

**return** **False**

peak\_index **=** **-**1

**for** i **in** range(1, n **-** 1):

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

peak\_index **=** i

**break**

**elif** arr[i] **<=** arr[i**-**1]:

**return** **False**

**if** peak\_index **==** **-**1 **or** 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.

**CODE:**

**def** findMaxLength(nums):

max\_len **=** 0

sum\_dict **=** {0: **-**1}

\_sum **=** 0

**for** i, num **in** enumerate(nums):

\_sum **+=** 1 **if** num **==** 1 **else** **-**1

**if** \_sum **in** sum\_dict:

max\_len **=** max(max\_len, i **-** sum\_dict[\_sum])

**else**:

sum\_dict[\_sum] **=** 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.

**CODE:**

**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].

**CODE:**

**from** collections **import** defaultdict

**def** findOriginalArray(changed):

count\_dict **=** defaultdict(int)

**for** num **in** changed:

**if** count\_dict[num **/** 2] **>** 0:

count\_dict[num **/** 2] **-=** 1

**elif** num **\*** 2 **not** **in** count\_dict:

count\_dict[num **\*** 2] **=** 1

**else**:

**return** []

original **=** []

**for** num, count **in** count\_dict**.**items():

original**.**extend([num] **\*** count)

**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]]

**CODE:**

**def** generateMatrix(n):

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

value **=** 1

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

**while** value **<=** n **\*** n:

*# Fill top row*

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

matrix[top][i] **=** value

value **+=** 1

top **+=** 1

*# Fill right column*

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

matrix[i][right] **=** value

value **+=** 1

right **-=** 1

*# Fill bottom row*

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

matrix[bottom][i] **=** value

value **+=** 1

bottom **-=** 1

*# Fill left column*

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

matrix[i][left] **=** value

value **+=** 1

left **+=** 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]]

**CODE:**

**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(n):

**for** x **in** range(k):

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

**return** result