💡 \*\*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]

**def** find\_permutation(s):

n **=** len(s)

perm **=** [0] **\*** (n **+** 1)

small, large **=** 0, n

**for** i **in** range(n):

**if** s[i] **==** 'I':

perm[i] **=** small

small **+=** 1

**else**:

perm[i] **=** large

large **-=** 1

perm[n] **=** small

**return** perm

s **=** "IDID"

result **=** find\_permutation(s)

print(result)

*#OUTPUT:-* [0, 4, 1, 3, 2]

💡 \*\*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

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

**if** **not** matrix:

**return** **False**

rows, cols **=** len(matrix), len(matrix[0])

low, high **=** 0, rows **\*** cols **-** 1

**while** low **<=** high:

mid **=** (low **+** high) **//** 2

num **=** matrix[mid **//** cols][mid **%** cols]

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

**return** **True**

**elif** num **<** target:

low **=** mid **+** 1

**else**:

high **=** mid **-** 1

**return** **False**

matrix **=** [

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 60]

]

target **=** 3

result **=** search\_matrix(matrix, target)

print(result)

*#OUTPUT:-*True

💡 \*\*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

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

n **=** len(arr)

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

**return** **False**

i **=** 0

**while** i **<** n **-** 1 **and** arr[i] **<** arr[i **+** 1]:

i **+=** 1

**if** i **==** 0 **or** i **==** n **-** 1:

**return** **False**

**while** i **<** n **-** 1 **and** arr[i] **>** arr[i **+** 1]:

i **+=** 1

**return** i **==** n **-** 1

arr **=** [2, 1]

result **=** valid\_mountain\_array(arr)

print(result)

*#OUTPUT:-*False

💡 \*\*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.

**def** find\_max\_length(nums):

count **=** 0

max\_length **=** 0

counts **=** {0: **-**1}

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

**if** num **==** 1:

count **+=** 1

**else**:

count **-=** 1

**if** count **in** counts:

max\_length **=** max(max\_length, i **-** counts[count])

**else**:

counts[count] **=** i

**return** max\_length

nums **=** [0, 1]

result **=** find\_max\_length(nums)

print(result)

*#OUTPUT:-*2

💡 \*\*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.

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

nums1**.**sort()

nums2**.**sort(reverse**=True**)

**return** sum(nums1[i] **\*** nums2[i] **for** i **in** range(len(nums1)))

nums1 **=** [5, 3, 4, 2]

nums2 **=** [4, 2, 2, 5]

print(minProductSum(nums1, nums2))

*#OUTPUT:-*40

💡 \*\*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].

**def** findOriginalArray(changed):

**if** len(changed) **%** 2 **!=** 0:

**return** []

counter **=** {}

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

counter[num] **=** counter**.**get(num, 0) **+** 1

original **=** []

**for** num **in** sorted(changed):

**if** counter[num] **>** 0 **and** counter**.**get(num**\***2, 0) **>** 0:

original**.**append(num)

counter[num] **-=** 1

counter[num**\***2] **-=** 1

**if** sum(counter**.**values()) **==** 0:

**return** original

**else**:

**return** []

changed **=** [1, 3, 4, 2, 6, 8]

print(findOriginalArray(changed))

*#OUTPUT:-* [1, 3, 4]

💡 \*\*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]]

**def** generateMatrix(n):

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

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

num **=** 1

**while** num **<=** n **\*** n:

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

matrix[top][i] **=** num

num **+=** 1

top **+=** 1

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

matrix[i][right] **=** num

num **+=** 1

right **-=** 1

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

matrix[bottom][i] **=** num

num **+=** 1

bottom **-=** 1

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

matrix[i][left] **=** num

num **+=** 1

left **+=** 1

**return** matrix

n **=** 3

print(generateMatrix(n))

*#OUTPUT:-* [[1, 2, 3], [8, 9, 4], [7, 6, 5]]

💡 \*\*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]]

**def** multiply(mat1, mat2):

m, k, n **=** len(mat1), len(mat1[0]), len(mat2[0])

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

**for** i **in** range(m):

**for** j **in** range(n):

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

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

**return** result

mat1 **=** [[1, 0, 0], [**-**1, 0, 3]]

mat2 **=** [[7, 0, 0], [0, 0, 0], [0, 0, 1]]

print(multiply(mat1, mat2))

*#OUTPUT:-*[[7, 0, 0], [-7, 0, 3]]