Assignment 06 Solution

Q1. 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]

In [151…

```
def findPermutation(s):
    n = len(s)
    perm = []
    low, high = 0, n

    for c in s:
        if c == 'I':
            perm.append(low)
            low += 1
        else:
            perm.append(high)
            high -= 1

    perm.append(low)

    return perm
```

In [152…

```
s = "IDID"
perm = findPermutation(s)
print(perm)
```

```
[0, 4, 1, 3, 2]
```

Q2. 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.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.

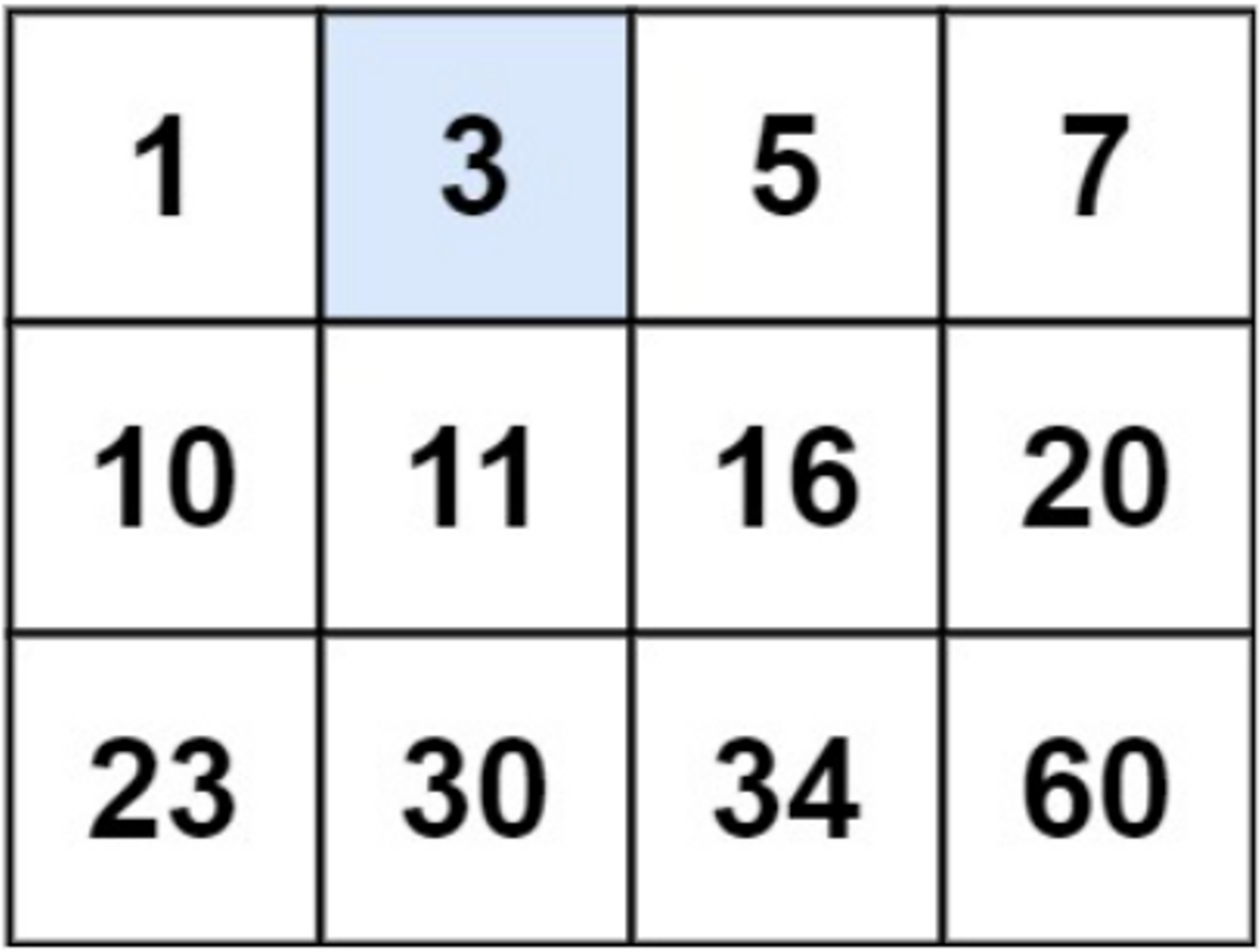
Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

Output: true

Example 1:

In [167…

Out[167]:



1	3	5	7
10	11	16	20
23	30	34	60

In [153…

```
def searchMatrix(matrix, target):
    m = len(matrix)
    n = len(matrix[0])
    low = 0
    high = m * n - 1

    while low <= high:
        mid = (low + high) // 2
        row = mid // n
        col = mid % n

        if matrix[row][col] == target:
            return True
        elif matrix[row][col] < target:
            low = mid + 1
        else:
            high = mid - 1

    return False
def searchMatrix(matrix, target):
    m = len(matrix)
    n = len(matrix[0])
    low = 0
    high = m * n - 1

    while low <= high:
        mid = (low + high) // 2
        row = mid // n
        col = mid % n

        if matrix[row][col] == target:
            return True
        elif matrix[row][col] < target:
            low = mid + 1
        else:
            high = mid - 1

    return False
```

In [154…

```
matrix = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 60]]
target = 3
print(searchMatrix(matrix, target))
```

```
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:

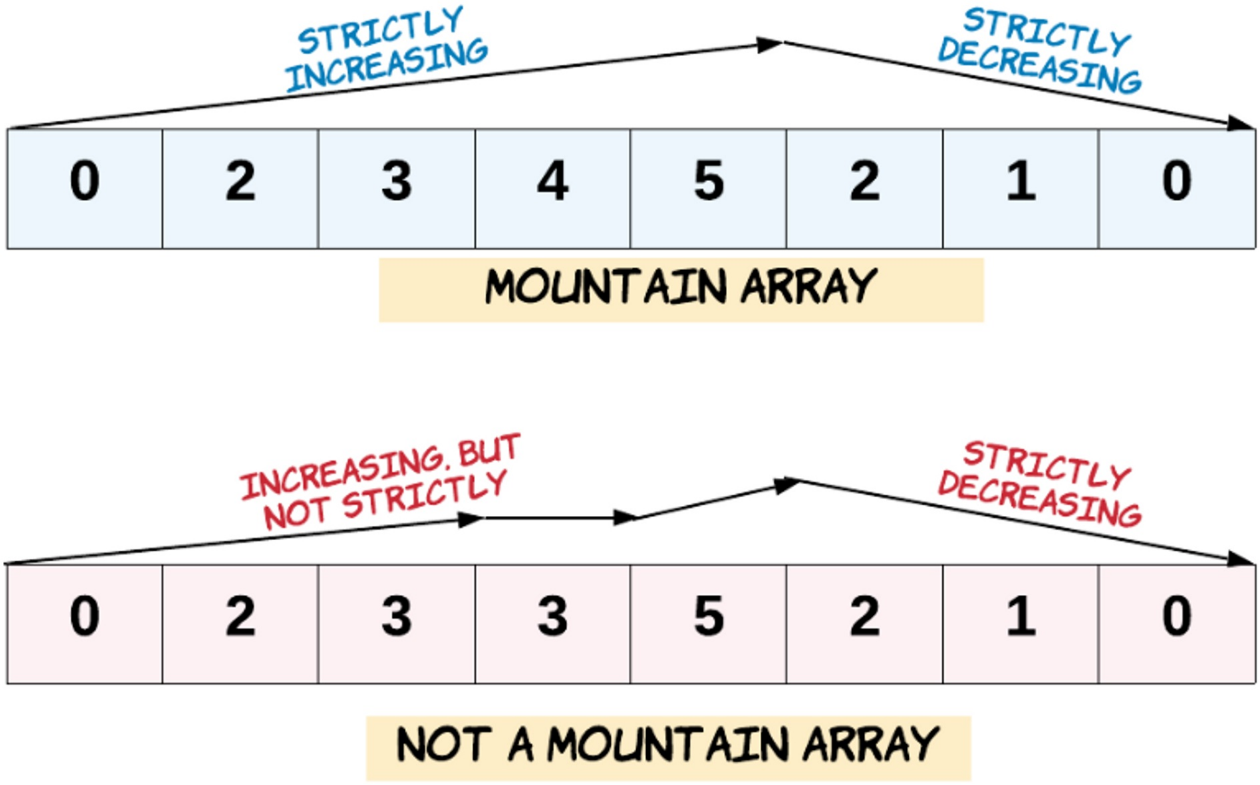
Input: arr = [2,1]

Output:

false

In [168…

Out[168]:



In [155…

```python
def validMountainArray(arr):
    n = len(arr)
    i = 0

    while i + 1 < n and arr[i] < arr[i + 1]:
        i += 1

    if i == 0 or i == n - 1:
        return False

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

    return i == n - 1
```

In [156…

```python
arr = [2, 1]
print(validMountainArray(arr))
```

```
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.

In [157…
```python
def findMaxLength(nums):
    count_map = {0: -1}
    count = 0
    max_length = 0

    for i, num in enumerate(nums):
        if num == 1:
            count += 1
        else:
            count -= 1

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

    return max_length
```

In [158…
```python
nums = [0, 1]
print(findMaxLength(nums))
```

```
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.

In [159…
```python
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
```

In [160…
```python
nums1 = [5, 3, 4, 2]
nums2 = [4, 2, 2, 5]
print(minProductSum(nums1, nums2))
```

```
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].

In [161…

```
from collections import Counter

def findOriginalArray(changed):
    original_count = Counter()
    changed_count = Counter(changed)

    for num in changed:
        if num == 0:
            if changed_count[num] % 2 == 1:
                return []
            continue

        if changed_count[num] > 0 and changed_count[num * 2] > 0:
            original_count[num * 2] += 1
            changed_count[num] -= 1
            changed_count[num * 2] -= 1
        else:
            return []

    original = []
    for num, count in original_count.items():
        original.extend([num] * count)

    return original
```

In [162…

```
changed = [1, 3, 4, 2, 6, 8]
result = findOriginalArray(changed)
print(result)
```

```
[]
```

Question 7

Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.

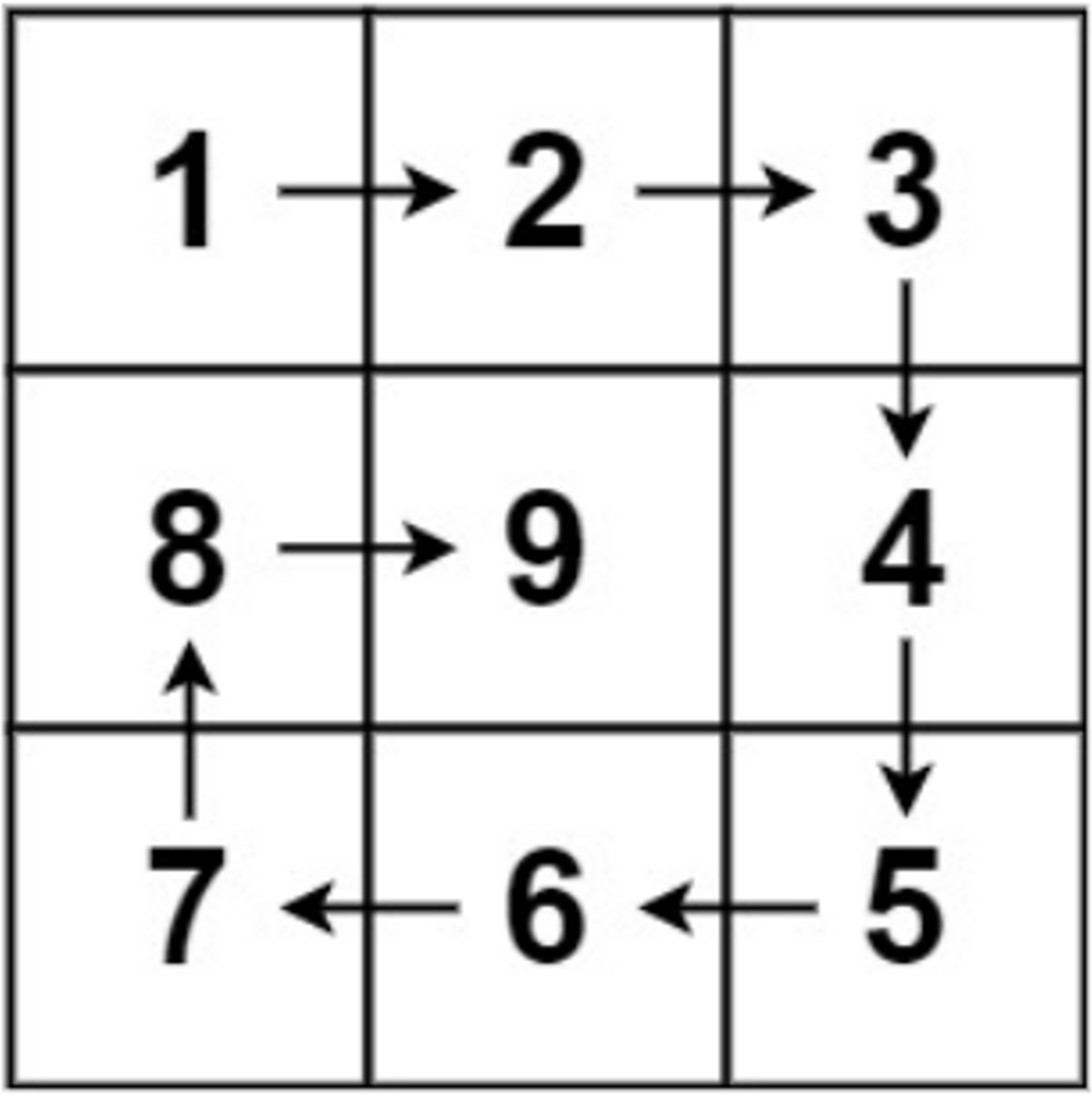
Input: n = 3

Output: [[1,2,3],[8,9,4],[7,6,5]]

Example 1:

In [169…

Out[169]:



In [163…

```
def generateMatrix(n):
    matrix = [[0] * n for _ in range(n)]
    row_start, row_end = 0, n - 1
    col_start, col_end = 0, n - 1
    num = 1

    while num <= n * n:
        # Fill top row
        for j in range(col_start, col_end + 1):
            matrix[row_start][j] = num
            num += 1
        row_start += 1

        # Fill right column
        for i in range(row_start, row_end + 1):
            matrix[i][col_end] = num
            num += 1
        col_end -= 1

        # Fill bottom row
        for j in range(col_end, col_start - 1, -1):
            matrix[row_end][j] = num
            num += 1
        row_end -= 1

        # Fill left column
        for i in range(row_end, row_start - 1, -1):
            matrix[i][col_start] = num
            num += 1
        col_start += 1

    return matrix
```

In [164…

```
n = 3
result = generateMatrix(n)
print(result)
```

```
[[1, 2, 3], [8, 9, 4], [7, 6, 5]]
```

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.

Input: mat1 = [[1,0,0],[-1,0,3]], mat2 = [[7,0,0],[0,0,0],[0,0,1]]

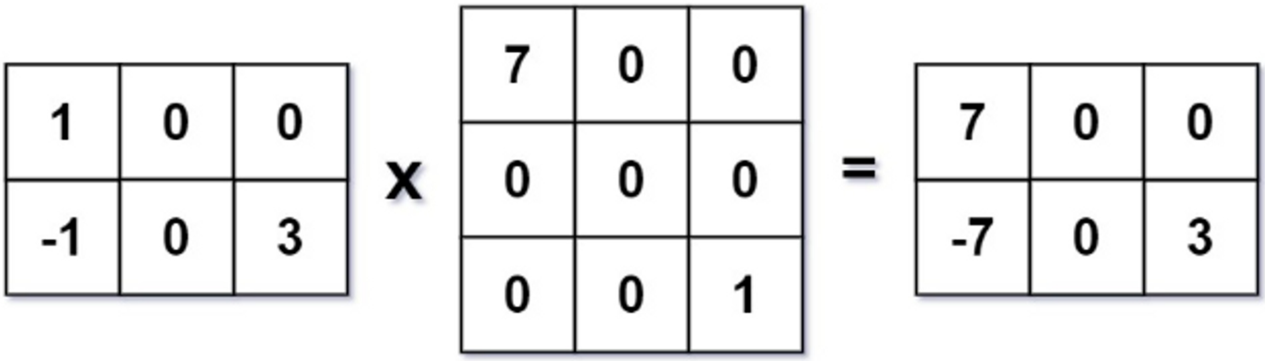
Output:

[[7,0,0],[-7,0,3]]

Example 1:

In [170…

Out[170]:



$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 3 \end{bmatrix} \times \begin{bmatrix} 7 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 7 & 0 & 0 \\ -7 & 0 & 3 \end{bmatrix}$$

In [165…

```
def multiply(mat1, mat2):
    m = len(mat1)
    n = len(mat2[0])
    k = len(mat2)
    result = [[0] * n for _ in range(m)]

    for i in range(m):
        for j in range(n):
            sum = 0
            for x in range(k):
                sum += mat1[i][x] * mat2[x][j]
            result[i][j] = sum

    return result
```

In [166…

```
mat1 = [[1, 0, 0], [-1, 0, 3]]
mat2 = [[7, 0, 0], [0, 0, 0], [0, 0, 1]]
result = multiply(mat1, mat2)
print(result)
```

```
[[7, 0, 0], [-7, 0, 3]]
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js