ASSIGNMENT 6

1.

```
def reconstruct_permutation(s):
    n = len(s)
    perm = []
    start, end = 0, 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
s = "IDID"
result = reconstruct_permutation(s)
print(result)
```

2.

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

3.

```
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)
```

4.

```
def find_max_length(nums):
    max_length = 0
    count = 0
    prefix_sums = {0: -1}
    for i, num in enumerate(nums):
        count += 1 if num == 1 else -1

        if count == 0:
            max_length = i + 1
        elif count not in prefix_sums:
            prefix_sums[count] = i
        else:
            length = i - prefix_sums[count]
            if length > max_length:
                max_length = length

    return max_length
nums = [0, 1]
result = find_max_length(nums)
print(result)
nums = [0, 1]
result = find_max_length(nums)
print(result)
```

5.

```
def minimum_product_sum(nums1, nums2):
    nums1.sort()
    nums2.sort()

    min_product_sum = 0
```

```
    for i in range(len(nums1)):
        min_product_sum += nums1[i] * nums2[len(nums2) - i - 1]

    return min_product_sum
nums1 = [5, 3, 4, 2]
nums2 = [4, 2, 2, 5]
result = minimum_product_sum(nums1, nums2)
print(result)
```

6.

```
def find_original_array(changed):
    original = []
    for num in changed:
        original_value = num // 2
        if original_value in original:
            continue
        original.append(original_value)
    return original
changed = [1, 3, 4, 2, 6, 8]
result = find_original_array(changed)
print(result)
```

7.

```
def generate_spiral_matrix(n):
    left = 0
    right = n - 1
    top = 0
    bottom = n - 1

    result = [[0] * n for _ in range(n)]
    num = 1

    while num <= n * n:
```

```
        for i in range(left, right + 1):
            result[top][i] = num
            num += 1
        top += 1

        for i in range(top, bottom + 1):
            result[i][right] = num
            num += 1
        right -= 1

        for i in range(right, left - 1, -1):
            result[bottom][i] = num
            num += 1
        bottom -= 1

        for i in range(bottom, top - 1, -1):
            result[i][left] = num
            num += 1
        left += 1

    return result
n = 3
result = generate_spiral_matrix(n)
print(result)
```

8.

```
def multiply_sparse_matrices(mat1, mat2):
    result = {}
    mat1_dict = {}
    mat2_dict = {}
```

```
    for i in range(len(mat1)):
        for j in range(len(mat1[0])):
            if mat1[i][j] != 0:
                mat1_dict[(i, j)] = mat1[i][j]



    for i in range(len(mat2)):
        for j in range(len(mat2[0])):
            if mat2[i][j] != 0:
                mat2_dict[(i, j)] = mat2[i][j]

    for (row, col) in mat1_dict:
        for k in range(len(mat2[0])):
            if (col, k) in mat2_dict:
                product = mat1_dict[(row, col)] * mat2_dict[(col, k)]
                if (row, k) in result:
                    result[(row, k)] += product
                else:
                    result[(row, k)] = product



    rows = len(mat1)
    cols = len(mat2[0])
    matrix_result = [[0] * cols for _ in range(rows)]
    for (row, col), value in result.items():
        matrix_result[row][col] = value

    return matrix_result
mat1 = [[1, 0, 0], [-1, 0, 3]]
mat2 = [[7, 0, 0], [0, 0, 0], [0, 0, 1]]
result = multiply_sparse_matrices(mat1, mat2)
```

```
print(result)
```