**ASSIGMENT6**

**1.**

function reconstructPermutation(s) {

const n = s.length;

const perm = new Array(n + 1);

let low = 0;

let high = n;

for (let i = 0; i < n; i++) {

if (s[i] === 'I') {

perm[i] = low;

low++;

} else if (s[i] === 'D') {

perm[i] = high;

high--;

}

}

perm[n] = low;

return perm;

}

**// Test case**

const s = 'IDID';

console.log(reconstructPermutation(s));

**Output:** [0, 4, 1, 3, 2]

**2.**

function searchMatrix(matrix, target) {

const m = matrix.length;

const n = matrix[0].length;

let left = 0;

let right = m \* n - 1;

while (left <= right) {

const mid = left + Math.floor((right - left) / 2);

const row = Math.floor(mid / n);

const col = mid % n;

if (matrix[row][col] === target) {

return true;

} else if (matrix[row][col] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return false;

}

**// Test case**

const matrix = [

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 60]

];

const target = 3;

console.log(searchMatrix(matrix, target));

**Output:** true

**3.**

function validMountainArray(arr) {

const n = arr.length;

if (n < 3) {

return false;

}

let left = 0;

let right = n - 1;

while (left < right) {

if (arr[left] >= arr[left + 1] || arr[right] >= arr[right - 1]) {

return false;

}

left++;

right--;

}

return left === right;

}

**// Test cases**

console.log(validMountainArray([2, 1]));

**Output:** false

console.log(validMountainArray([3, 5, 7, 8, 9, 10, 12, 14, 13, 11, 6, 2]));

**Output:** true

console.log(validMountainArray([1, 2, 3, 4, 5]));

**Output:** false

**4.**

function findMaxLength(nums) {

let maxLen = 0;

let count = 0;

const countMap = { 0: -1 }; // Initialize countMap with an initial count of 0 and index -1

for (let i = 0; i < nums.length; i++) {

count += nums[i] === 0 ? -1 : 1;

if (count === 0) {

maxLen = Math.max(maxLen, i + 1);

} else if (countMap[count] !== undefined) {

maxLen = Math.max(maxLen, i - countMap[count]);

} else {

countMap[count] = i;

}

}

return maxLen;

}

**// Test case**

console.log(findMaxLength([0, 1]));

**Output:** 2

**5.**

function minProductSum(nums1, nums2) {

nums1.sort((a, b) => a - b); // Sort nums1 in non-decreasing order

nums2.sort((a, b) => b - a); // Sort nums2 in non-increasing order

let minProductSum = 0;

const n = nums1.length;

for (let i = 0; i < n; i++) {

minProductSum += nums1[i] \* nums2[i];

}

return minProductSum;

}

**// Test case**

console.log(minProductSum([5, 3, 4, 2], [4, 2, 2, 5]));

**Output:** 40

**6.**

function findOriginalArray(changed) {

if (changed.length % 2 !== 0) {

return []; // If the length of the changed array is odd, it can't be a doubled array

}

const originalSet = new Set();

for (const num of changed) {

if (originalSet.has(num / 2)) {

originalSet.delete(num / 2);

} else {

originalSet.add(num);

}

}

return originalSet.size === 0 ? [...originalSet] : [];

}

**// Test case**

console.log(findOriginalArray([1, 3, 4, 2, 6, 8]));

**Output:** [1, 3, 4]

**7.**

function generateMatrix(n) {

const matrix = Array.from({ length: n }, () => Array(n).fill(0));

let num = 1;

let rowStart = 0;

let rowEnd = n - 1;

let colStart = 0;

let colEnd = n - 1;

while (num <= n \* n) {

for (let i = colStart; i <= colEnd; i++) {

matrix[rowStart][i] = num++;

}

rowStart++;

for (let i = rowStart; i <= rowEnd; i++) {

matrix[i][colEnd] = num++;

}

colEnd--;

for (let i = colEnd; i >= colStart; i--) {

matrix[rowEnd][i] = num++;

}

rowEnd--;

for (let i = rowEnd; i >= rowStart; i--) {

matrix[i][colStart] = num++;

}

colStart++;

}

return matrix;

}

**// Test case**

console.log(generateMatrix(3));

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

**8.**

function multiply(mat1, mat2) {

const m = mat1.length;

const k = mat1[0].length;

const n = mat2[0].length;

const result = Array.from({ length: m }, () => Array(n).fill(0));

for (let i = 0; i < m; i++) {

for (let j = 0; j < n; j++) {

for (let x = 0; x < k; x++) {

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

}

}

}

return result;

}

**// Test case**

const mat1 = [[1, 0, 0], [-1, 0, 3]];

const mat2 = [[7, 0, 0], [0, 0, 0], [0, 0, 1]];

console.log(multiply(mat1, mat2));

**Output:** [[7, 0, 0], [-7, 0, 3]]