Assignment 6 Solution -2D Arrays | DSA

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]

Solution Code:

```
package in.ineuron.pptAssignment06;
import java.util.*;
public class PermutationReconstruction {
		public int[] findPermutation(String s) {
		int n = s.length();
		int[] perm = new int[n + 1];
		int left = 0;
		int right = 0;

		for (int i = 0; i < n; i++) {
			if (s.charAt(i) == 'I') {
				perm[i] = left++;
			} else {
				perm[i] = right--;
			}
		}

		perm[n] = left;

		for (int i = 0; i < n + 1; i++) {
			perm[i] -= right;
		}

		return perm;
	}

	public static void main(String[] args) {
		PermutationReconstruction pr = new PermutationReconstruction();
		String s = "IDID";
		int[] permutation = pr.findPermutation(s);
		System.out.println(Arrays.toString(permutation));
	}
}
```

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:

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

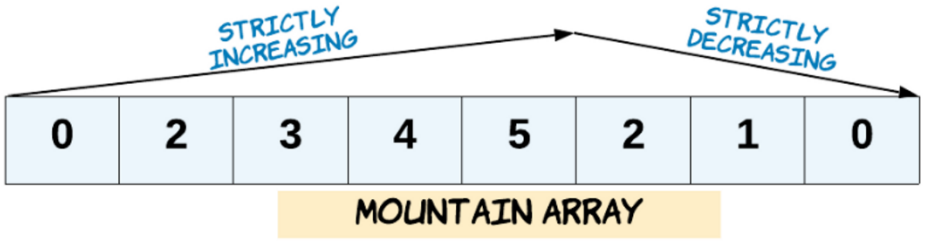
Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

Solution Code:

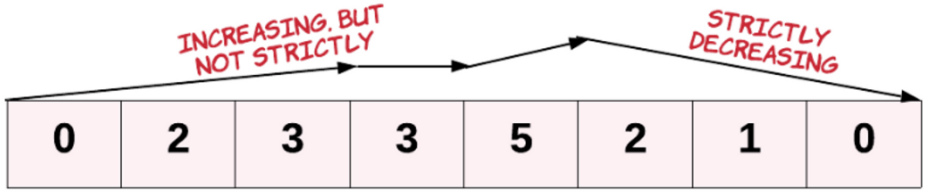
```
package in.ineuron.pptAssignment06;
public class SearchMatrix {
	public static void main(String[] args) {
		int[][] matrix = { { 1, 3, 5, 7 }, { 10, 11, 16, 20 }, { 23, 30, 34, 60 } };
		int target = 3;
		System.out.println(searchMatrix(matrix, target));
	}
	public static boolean searchMatrix(int[][] matrix, int target) {
		int rows = matrix.length;
		int cols = matrix[0].length;

		int left = 0;
		int right = rows * cols - 1;

		while (left <= right) {
			int mid = left + (right - left) / 2;
			int midValue = matrix[mid / cols][mid % cols];

			if (midValue == target) {
				return true;
			} else if (midValue < 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]



MOUNTAIN ARRAY



NOT A MOUNTAIN ARRAY

Example 1:
Input: arr = [2,1]
Output: false

Solution Code:

```
package in.ineuron.pptAssignment06;

public class MountainArray {
	public static boolean validMountainArray(int[] arr) {
		if (arr.length < 3) {
			return false;
		}

		int i = 0;
		while (i < arr.length - 1 && arr[i] < arr[i + 1]) {
			i++;
		}

		if (i == 0 || i == arr.length - 1) {
			return false;
		}

		while (i < arr.length - 1 && arr[i] > arr[i + 1]) {
			i++;
		}
```

```
            return i == arr.length - 1;
        }

        public static void main(String[] args) {
            int[] arr = { 2, 1 };
            boolean result = validMountainArray(arr);
            System.out.println(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.

Solution Code :

```
package in.ineuron.pptAssignment06;

import java.util.HashMap;
import java.util.Map;

public class ContiguousSubarray {

    public static int findMaxLength(int[] nums) {
        // Create a map to store the cumulative count and its corresponding index
        Map<Integer, Integer> countMap = new HashMap<>();
        countMap.put(0, -1); // Initialize the map with count 0 at index -1
        int maxLen = 0;
        int count = 0;

        for (int i = 0; i < nums.length; i++) {
            // Increment count by 1 for 1 and decrement by 1 for 0
            count += nums[i] == 1 ? 1 : -1;

            // If the count is already present in the map, update the maximum length
            if (countMap.containsKey(count)) {
                int prevIndex = countMap.get(count);
                maxLen = Math.max(maxLen, i - prevIndex);
            } else {
```

```
                        // If the count is not present, add it to the map
                        countMap.put(count, i);
                    }
                }

                return maxLen;
            }

            public static void main(String[] args) {
                int[] nums = { 0, 1 };
                int maxLength = findMaxLength(nums);
                System.out.println("Maximum length of contiguous subarray: " + maxLength);
            }
        }
```

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 15 + 22 + 33 + 41 = 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 34 + 52 + 42 + 25 = 40.

Solution Code:

```
package in.ineuron.pptAssignment06;
import java.util.Arrays;
public class MinimumProductSum {
    public static int minProductSum(int[] nums1, int[] nums2) {
        Arrays.sort(nums1); // Sort nums1 in ascending order
        Arrays.sort(nums2); // Sort nums2 in ascending order

        int n = nums1.length;
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += nums1[i] * nums2[n - i - 1];
    // Multiply the smallest number in nums1 with the largest number in nums2, andso on
        }
        return sum;
    }
```

```
        public static void main(String[] args) {
            int[] nums1 = { 5, 3, 4, 2 };
            int[] nums2 = { 4, 2, 2, 5 };
            int minProduct = minProductSum(nums1, nums2);
            System.out.println("Minimum product sum: " + minProduct);
        }
    }
```

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].

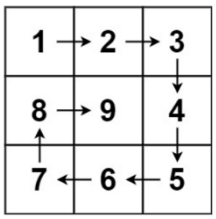
Solution Code:

```
package in.ineuron.pptAssignment06;

import java.util.*;
public class DoubledArray {

   public static int[] findOriginalArray(int[] changed) {
      if (changed.length % 2 != 0) {
         // If the length of changed is odd, it cannot be a doubled array
         return new int[0];
      }

      Map<Integer, Integer> frequency = new HashMap<>();
      for (int num : changed) {
         frequency.put(num, frequency.getOrDefault(num, 0) + 1);
      }

      List<Integer> original = new ArrayList<>();
      Arrays.sort(changed);

      for (int num : changed) {
         if (frequency.containsKey(num) && frequency.containsKey(num * 2)) {
            original.add(num);
```

```
                int count = frequency.get(num);
                int doubleCount = frequency.get(num * 2);

                if (count == 1) {
                    frequency.remove(num);
                } else {
                    frequency.put(num, count - 1);
                }

                if (doubleCount == 1) {
                    frequency.remove(num * 2);
                } else {
                    frequency.put(num * 2, doubleCount - 1);
                }
            }
        }

        if (original.size() == changed.length / 2) {
            int[] result = new int[original.size()];
            for (int i = 0; i < original.size(); i++) {
                result[i] = original.get(i);
            }
            return result;
        } else {
            return new int[0];
        }
    }

    public static void main(String[] args) {
        int[] changed = {1, 3, 4, 2, 6, 8};
        int[] original = findOriginalArray(changed);
        System.out.println(Arrays.toString(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]]

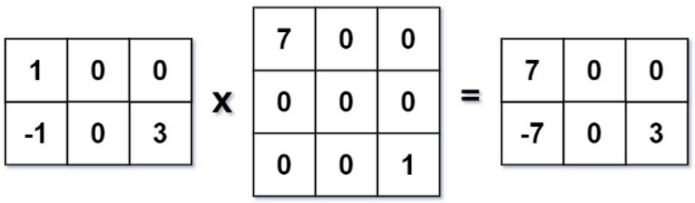
Solution Code:

```
package in.ineuron.pptAssignment06;

public class SpiralMatrixGenerator {

	public static int[][] generateMatrix(int n) {
		int[][] matrix = new int[n][n];
		int num = 1; // Starting number
		int rowStart = 0, rowEnd = n - 1;
		int colStart = 0, colEnd = n - 1;

		while (rowStart <= rowEnd && colStart <= colEnd) {
			// Fill the top row
			for (int col = colStart; col <= colEnd; col++) {
				matrix[rowStart][col] = num++;
			}
			rowStart++;

			// Fill the right column
			for (int row = rowStart; row <= rowEnd; row++) {
				matrix[row][colEnd] = num++;
			}
			colEnd--;

			// Fill the bottom row
			if (rowStart <= rowEnd) {
				for (int col = colEnd; col >= colStart; col--) {
					matrix[rowEnd][col] = num++;
				}
				rowEnd--;
			}

			// Fill the left column
			if (colStart <= colEnd) {
				for (int row = rowEnd; row >= rowStart; row--) {
```

```
                                matrix[row][colStart] = num++;
                            }
                            colStart++;
                        }
                    }

                    return matrix;
                }

                public static void printMatrix(int[][] matrix) {
                    for (int[] row : matrix) {
                        for (int num : row) {
                            System.out.print(num + " ");
                        }
                        System.out.println();
                    }
                }

                public static void main(String[] args) {
                    int n = 3;
                    int[][] matrix = generateMatrix(n);
                    printMatrix(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]]

Solution Code:

```
package in.ineuron.pptAssignment06;

import java.util.*;
public class SparseMatrixMultiplication {

	public static void main(String[] args) {
		int[][] mat1 = { { 1, 0, 0 }, { -1, 0, 3 } };
		int[][] mat2 = { { 7, 0, 0 }, { 0, 0, 0 }, { 0, 0, 1 } };

		SparseMatrixMultiplication matrixMultiplication = new
		SparseMatrixMultiplication();

		int[][] result = matrixMultiplication.multiply(mat1, mat2);

		// Print the result
		for (int[] row : result) {
			for (int element : row) {
				System.out.print(element + " ");
			}
			System.out.println();
		}
	}

	public int[][] multiply(int[][] mat1, int[][] mat2) {
		int m = mat1.length;
		int k = mat1[0].length;
		int n = mat2[0].length;

		int[][] result = new int[m][n];

		// Create a map to store the non-zero elements of mat2
		Map<Integer, Map<Integer, Integer>> sparseMat2 = new HashMap<>();
		for (int j = 0; j < k; j++) {
```

```
                for (int i = 0; i < n; i++) {
                    if (mat2[j][i] != 0) {
                    sparseMat2.computeIfAbsent(j, HashMap::new).put(i, mat2[j][i]);
                    }
                }
            }

            // Perform matrix multiplication
            for (int i = 0; i < m; i++) {
                for (int j : sparseMat2.keySet()) {
                    Map<Integer, Integer> colMap = sparseMat2.get(j);
                    for (int l : colMap.keySet()) {
                        result[i][l] += mat1[i][j] * colMap.get(l);
                    }
                }
            }
            return result;
        }
    }
```