Assignment 5 Solution – 2D Arrays | DSA

Question 1

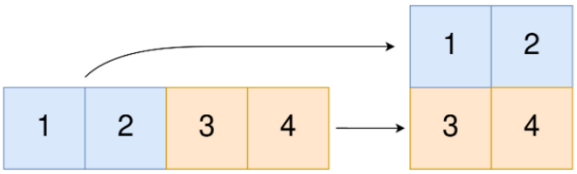
Convert 1D Array Into 2D Array

You are given a 0-indexed 1-dimensional (1D) integer array original, and two integers, m and n. You are tasked with creating a 2-dimensional (2D) array with m rows and n columns using all the elements from original.

The elements from indices 0 to n - 1 (inclusive) of original should form the first row of the constructed 2D array, the elements from indices n to 2 n - 1 (inclusive) should form the second row of the constructed 2D array, and so on.

Return an m x n 2D array constructed according to the above procedure, or an empty 2D array if it is impossible.

Example 1:



Input: original = [1,2,3,4], m = 2, n = 2

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

Explanation: The constructed 2D array should contain 2 rows and 2 columns.

The first group of n=2 elements in original**, [1,2],** becomes the first row in the constructed 2D array.

The second group of n=2 elements in original, **[3,4],** becomes the second row in the constructed 2D array

Solution Code:

```
package in.ineuron.pptAssignment05;

public class Construct2DArray {

	public static int[][] construct2DArray(int[] original, int m, int n) {
		int totalElements = original.length;
		if (totalElements != m * n) {
			return new int[0][0]; // Return an empty 2D array
		}

		int[][] newArray = new int[m][n];
		for (int i = 0; i < totalElements; i++) {
			int row = i / n;
			int column = i % n;
			newArray[row][column] = original[i];
		}

		return newArray;
	}
```

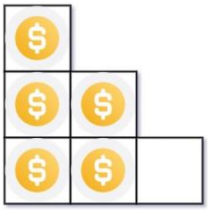
```
    public static void main(String[] args) {
        int[] original = { 1, 2, 3, 4 };
        int m = 2;
        int n = 2;
        int[][] result = construct2DArray(original, m, n);

        // Print the result
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Question 2

You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase may be incomplete.

Given the integer n, return the number of complete rows of the staircase you will build.
Example 1:



Input: n = 5
Output: 2
Explanation: Because the 3rd row is incomplete, we **return 2**

Solution Code:

```
package in.ineuron.pptAssignment05;
public class Staircase {

   public int countCompleteRows(int n) {
      int left = 1; // Starting with the first row
      int right = n; // Initially, the last row can be the complete row

      while (left <= right) {
         int mid = left + (right - left) / 2;
         int totalCoins = (mid * (mid + 1)) / 2;

         if (totalCoins == n)
            return mid; // Found a complete row

         if (totalCoins > n)
            right = mid - 1; // The current row is incomplete, check left side
         else
            left = mid + 1; // The current row is complete, check right side
      }

      return right; // Return the last complete row found
   }

   public static void main(String[] args) {
      Staircase staircase = new Staircase();
      int n = 5;
      int completeRows = staircase.countCompleteRows(n);
      System.out.println("Number of complete rows: " + completeRows);
   }
}
```

Question 3

Given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

Example 1:

Input: nums = [-4,-1,0,3,10]

Output: [0,1,9,16,100]

Explanation: After squaring, the array becomes [16,1,0,9,100].

After sorting, it becomes [0,1,9,16,100].

Solution Code:

```
package in.ineuron.pptAssignment05;

import java.util.Arrays;
public class SortedSquares {

	public static int[] sortedSquares(int[] nums) {
		int[] result = new int[nums.length];
		int left = 0;
		int right = nums.length - 1;
		int index = nums.length - 1;

		while (left <= right) {
			int leftSquare = nums[left] * nums[left];
			int rightSquare = nums[right] * nums[right];

			if (leftSquare > rightSquare) {
				result[index] = leftSquare;
				left++;
			} else {
				result[index] = rightSquare;
				right--;
			}
			index--;
		}

		return result;
	}

	public static void main(String[] args) {
		int[] nums = { -4, -1, 0, 3, 10 };
		int[] result = sortedSquares(nums);
		System.out.println(Arrays.toString(result));
	}
}
```

Question 4

Given two 0-indexed integer arrays nums1 and nums2, return a list answer of size 2 where:
- answer[0] is a list of all distinct integers in nums1 which are not present in nums2.
- answer[1] is a list of all distinct integers in nums2 which are not present in nums1.

Note that the integers in the lists may be returned in any order.

Example 1:
Input: nums1 = [1,2,3], nums2 = [2,4,6]
Output: [[1,3],[4,6]]

Explanation:
For nums1, nums1[1] = 2 is present at index 0 of nums2, whereas nums1[0] = 1 and nums1[2] = 3 are not present in nums2. Therefore, answer[0] = [1,3].

For nums2, nums2[0] = 2 is present at index 1 of nums1, whereas nums2[1] = 4 and nums2[2] = 6 are not present in nums2. Therefore, answer[1] = [4,6].

Solution Code :

```
package in.ineuron.pptAssignment05;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class ArrayDifference {
	public static List<List<Integer>> findArrayDifference(int[] nums1, int[] nums2) {
		List<Integer> diff1 = new ArrayList<>();
		List<Integer> diff2 = new ArrayList<>();

		Set<Integer> set1 = new HashSet<>();
		Set<Integer> set2 = new HashSet<>();

		// Add all elements of nums1 to set1
		for (int num : nums1) {
			set1.add(num);
		}

		// Add all elements of nums2 to set2
		for (int num : nums2) {
			set2.add(num);
		}

		// Find elements in nums1 that are not present in nums2
		for (int num : nums1) {
			if (!set2.contains(num)) {
				diff1.add(num);
			}
```

```
            }

            // Find elements in nums2 that are not present in nums1
            for (int num : nums2) {
                if (!set1.contains(num)) {
                    diff2.add(num);
                }
            }

            List<List<Integer>> answer = new ArrayList<>();
            answer.add(diff1);
            answer.add(diff2);

            return answer;
        }

        public static void main(String[] args) {
            int[] nums1 = { 1, 2, 3 };
            int[] nums2 = { 2, 4, 6 };

            List<List<Integer>> answer = findArrayDifference(nums1, nums2);
            System.out.println(answer);
        }
    }
```

Question 5

Given two integer arrays arr1 and arr2, and the integer d, return the distance value between the two arrays.
The distance value is defined as the number of elements arr1[i] such that there is not any element arr2[j] where |arr1[i]-arr2[j]| <= d.

Example 1:
Input: arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2
Output: 2

Explanation:

For arr1[0]=4 we have:
|4-10|=6 > d=2
|4-9|=5 > d=2
|4-1|=3 > d=2
|4-8|=4 > d=2
For arr1[1]=5 we have:
|5-10|=5 > d=2
|5-9|=4 > d=2
|5-1|=4 > d=2
|5-8|=3 > d=2
For arr1[2]=8 we have:
|8-10|=2 <= d=2
|8-9|=1 <= d=2
|8-1|=7 > d=2
|8-8|=0 <= d=2

Solution Code:

```
package in.ineuron.pptAssignment05;

public class DistanceValue {
	public static int distanceValue(int[] arr1, int[] arr2, int d) {
		int count = 0;

		for (int i = 0; i < arr1.length; i++) {
			boolean isValid = true;

			for (int j = 0; j < arr2.length; j++) {
				if (Math.abs(arr1[i] - arr2[j]) <= d) {
					isValid = false;
					break;
				}
			}

			if (isValid) {
				count++;
```

```
                }
            }
            return count;
        }

        public static void main(String[] args) {
            int[] arr1 = { 4, 5, 8 };
            int[] arr2 = { 10, 9, 1, 8 };
            int d = 2;

            int result = distanceValue(arr1, arr2, d);
            System.out.println("Distance value: " + result);
        }
    }
```

Question 6

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears once or twice, return an array of all the integers that appears twice. You must write an algorithm that runs in O(n) time and uses only constant extra space.

Example 1:
Input: nums = [4,3,2,7,8,2,3,1]
Output:
[2,3]

Solution Code:

```
package in.ineuron.pptAssignment05;
import java.util.ArrayList;
import java.util.List;

public class FindDuplicates {

    public static void main(String[] args) {
        int[] nums = { 4, 3, 2, 7, 8, 2, 3, 1 };
        System.out.println(findDuplicates(nums));
    }

    public static List<Integer> findDuplicates(int[] nums) {
        List<Integer> result = new ArrayList<>();

        for (int i = 0; i < nums.length; i++) {
            int index = Math.abs(nums[i]) - 1;

            if (nums[index] < 0) {
```

```
                        result.add(index + 1);
                    } else {
                        nums[index] = -nums[index];
                    }
                }

                for (int i = 0; i < nums.length; i++) {
                    nums[i] = Math.abs(nums[i]);
                }

                return result;
            }
        }
```

Question 7

Suppose an array of length n sorted in ascending order is rotated between 1 and n times. For example, the array nums = [0,1,2,4,5,6,7] might become:
- [4,5,6,7,0,1,2] if it was rotated 4 times.
- [0,1,2,4,5,6,7] if it was rotated 7 times.

Notice that rotating an array [a[0], a[1], a[2], ..., a[n-1]] 1 time results in the array [a[n-1], a[0], a[1], a[2], ..., a[n-2]].

Given the sorted rotated array nums of unique elements, return the minimum element of this array.

You must write an algorithm that runs in O(log n) time.

Example 1:
Input: nums = [3,4,5,1,2]
Output: 1
Explanation:
The original array was [1,2,3,4,5] rotated 3 times.

Solution Code:

```
package in.ineuron.pptAssignment05;

public class RotatedArrayMin {

	public static int findMin(int[] nums) {
		int left = 0;
		int right = nums.length - 1;

		while (left < right) {
			int mid = left + (right - left) / 2;

			if (nums[mid] > nums[right]) {
				left = mid + 1;
			} else {
				right = mid;
			}
		}

		return nums[left];
	}

	public static void main(String[] args) {
		int[] nums = { 3, 4, 5, 1, 2 };
		int min = findMin(nums);
		System.out.println("Minimum element: " + min);
	}
}
```

Question 8

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.pptAssignment05;

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));
  }
}
```