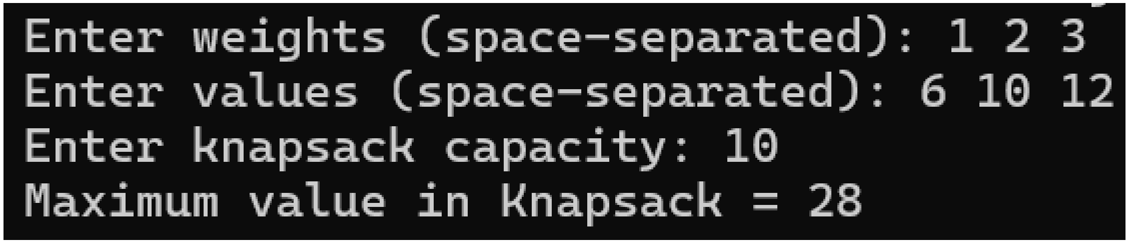
PRACTICE SET -2 12 – 11 - 2024

DSA

1.0-1 knapsack problem

```
import java.util.Scanner;

public class Knapsack {
  public static int knapsack(int[] weights, int[] values, int capacity) {
    int n = weights.length;
    int[][] dp = new int[n + 1][capacity + 1];

    for (int i = 1; i <= n; i++) {
      for (int w = 0; w <= capacity; w++) {
        if (weights[i - 1] <= w) {
          dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
        } else {
          dp[i][w] = dp[i - 1][w];
        }
      }
    }
    return dp[n][capacity];
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Get weights as a single line of space-separated integers
    System.out.print("Enter weights (space-separated): ");
    String[] weightsInput = scanner.nextLine().split(" ");
    int[] weights = new int[weightsInput.length];
    for (int i = 0; i < weightsInput.length; i++) {
      weights[i] = Integer.parseInt(weightsInput[i]);
    }

    // Get values as a single line of space-separated integers
    System.out.print("Enter values (space-separated): ");
    String[] valuesInput = scanner.nextLine().split(" ");
    int[] values = new int[valuesInput.length];
    for (int i = 0; i < valuesInput.length; i++) {
      values[i] = Integer.parseInt(valuesInput[i]);
    }
```

```
        // Get capacity as a single integer
        System.out.print("Enter knapsack capacity: ");
        int capacity = scanner.nextInt();

        // Calculate the maximum value for the knapsack
        int maxValue = knapsack(weights, values, capacity);
        System.out.println("Maximum value in Knapsack = " + maxValue);

        scanner.close();
    }
}
```

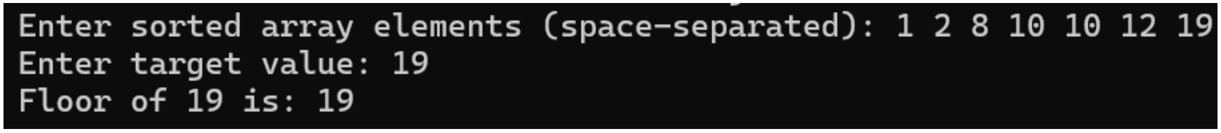


```
Enter weights (space-separated): 1 2 3
Enter values (space-separated): 6 10 12
Enter knapsack capacity: 10
Maximum value in Knapsack = 28
```

2.Floor in Sorted Array

```
import java.util.Scanner;

public class FloorInSortedArray {
    public static int findFloor(int[] arr, int target) {
        int left = 0, right = arr.length - 1;
        int floor = -1;

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

            if (arr[mid] == target) {
                return arr[mid];
            } else if (arr[mid] < target) {
                floor = arr[mid];
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return floor;
    }
```

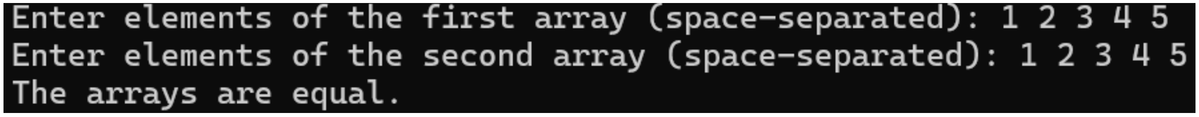
```
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Get array elements from user as a single line
    System.out.print("Enter sorted array elements (space-separated): ");
    String[] input = scanner.nextLine().split(" ");
    int[] arr = new int[input.length];
    for (int i = 0; i < input.length; i++) {
      arr[i] = Integer.parseInt(input[i]);
    }

    // Get target value
    System.out.print("Enter target value: ");
    int target = scanner.nextInt();

    // Find and print the floor of the target in the array
    System.out.println("Floor of " + target + " is: " + findFloor(arr, target));

    scanner.close();
  }
}
```



```
Enter sorted array elements (space-separated): 1 2 8 10 10 12 19
Enter target value: 19
Floor of 19 is: 19
```

3.Check equal arrays

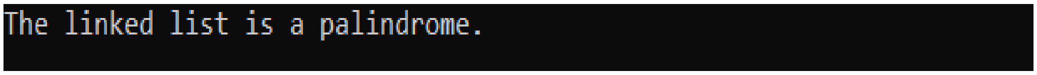
```
import java.util.Arrays;
import java.util.Scanner;

public class EqualArrays {
  public static boolean areArraysEqual(int[] arr1, int[] arr2) {
    // Check if both arrays are of the same length and contain the same elements
    return Arrays.equals(arr1, arr2);
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Get the first array from the user
    System.out.print("Enter elements of the first array (space-separated): ");
    String[] input1 = scanner.nextLine().split(" ");
    int[] arr1 = new int[input1.length];
    for (int i = 0; i < input1.length; i++) {
      arr1[i] = Integer.parseInt(input1[i]);
    }

    // Get the second array from the user
    System.out.print("Enter elements of the second array (space-separated): ");
    String[] input2 = scanner.nextLine().split(" ");
    int[] arr2 = new int[input2.length];
    for (int i = 0; i < input2.length; i++) {
      arr2[i] = Integer.parseInt(input2[i]);
    }

    // Check if the arrays are equal
    if (areArraysEqual(arr1, arr2)) {
      System.out.println("The arrays are equal.");
    } else {
      System.out.println("The arrays are not equal.");
    }

    scanner.close();
  }
}
```



```
Enter elements of the first array (space-separated): 1 2 3 4 5
Enter elements of the second array (space-separated): 1 2 3 4 5
The arrays are equal.
```

4.Palindrome linked list

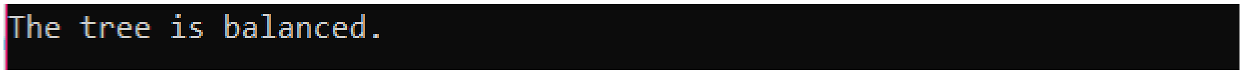
```
class ListNode {
  int val;
  ListNode next;

  ListNode(int val) {
    this.val = val;
    this.next = null;
  }
}

public class PalindromeLinkedList {
  public static boolean isPalindrome(ListNode head) {
    if (head == null || head.next == null) return true;
    // Step 1: Find the middle of the linked list using slow and fast pointers
    ListNode slow = head, fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    // Step 2: Reverse the second half of the list
    ListNode secondHalf = reverseList(slow);

    // Step 3: Compare the first half and the reversed second half
    ListNode firstHalf = head;
    while (secondHalf != null) {
        if (firstHalf.val != secondHalf.val) return false;
        firstHalf = firstHalf.next;
        secondHalf = secondHalf.next;
    }

    return true;
}

// Helper method to reverse a linked list
private static ListNode reverseList(ListNode head) {
    ListNode prev = null;
    while (head != null) {
        ListNode next = head.next;
        head.next = prev;
        prev = head;
        head = next;
    }
    return prev;
}

public static void main(String[] args) {
    // Example list: 1 -> 2 -> 2 -> 1
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(2);
    head.next.next.next = new ListNode(1);

    if (isPalindrome(head)) {
        System.out.println("The linked list is a palindrome.");
    } else {
        System.out.println("The linked list is not a palindrome.");
        }
    }
}
```



```
The linked list is a palindrome.
```

5.Balanced tree check

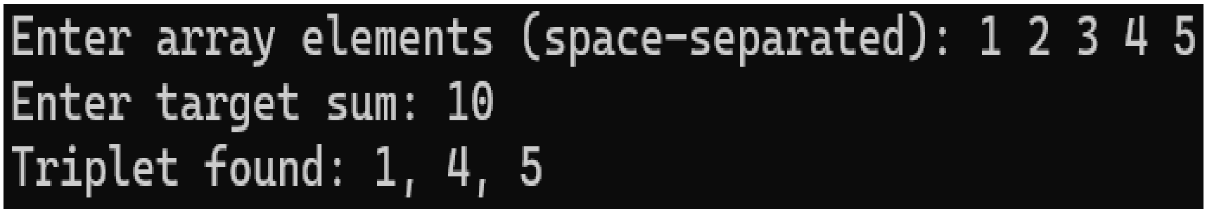
```
class TreeNode {
    int val;
    TreeNode left, right;

    TreeNode(int val) {
        this.val = val;
        this.left = this.right = null;
    }
}

public class BalancedTreeCheck {
    public static boolean isBalanced(TreeNode root) {
        return checkHeight(root) != -1;
    }

    // Helper method that returns the height of the tree if balanced, or -1 if unbalanced
    private static int checkHeight(TreeNode node) {
        if (node == null) return 0;

        int leftHeight = checkHeight(node.left);
        if (leftHeight == -1) return -1;

        int rightHeight = checkHeight(node.right);
        if (rightHeight == -1) return -1;

        // Check if the current node is balanced
        if (Math.abs(leftHeight - rightHeight) > 1) return -1;

        // Return the height of the current node
        return Math.max(leftHeight, rightHeight) + 1;
    }

    public static void main(String[] args) {
        // Example tree: Balanced tree
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.right.right = new TreeNode(6);

        if (isBalanced(root)) {
            System.out.println("The tree is balanced.");
        } else {
            System.out.println("The tree is not balanced.");
        }
    }
}
```



```
The tree is balanced.
```

6.Triplet sum in array

```
import java.util.Arrays;
import java.util.Scanner;

public class TripletSumInArray {

    public static boolean findTriplet(int[] arr, int target) {
        Arrays.sort(arr);  // Sort the array to use two-pointer technique

        for (int i = 0; i < arr.length - 2; i++) {
            int left = i + 1;
            int right = arr.length - 1;

            while (left < right) {
                int currentSum = arr[i] + arr[left] + arr[right];
                if (currentSum == target) {
                    System.out.println("Triplet found: " + arr[i] + ", " + arr[left] + ", " + arr[right]);
                    return true;
                } else if (currentSum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get array from user as a single line of space-separated integers
        System.out.print("Enter array elements (space-separated): ");
        String[] input = scanner.nextLine().split(" ");
        int[] arr = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            arr[i] = Integer.parseInt(input[i]);
        }

        // Get the target sum from user
        System.out.print("Enter target sum: ");
        int target = scanner.nextInt();

        // Check if a triplet exists
        if (!findTriplet(arr, target)) {
            System.out.println("No triplet found with the given sum.");
        }

        scanner.close();
    }
}
```



```
Enter array elements (space-separated): 1 2 3 4 5
Enter target sum: 10
Triplet found: 1, 4, 5
```