1. Palindrome Linked List

```java
import java.util.ArrayList;
import java.util.List;

class ListNode {
    int val;
    ListNode next;

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

public class isPalindromeLL {
    public static boolean isPalindrome(ListNode head) {
        List<Integer> values = new ArrayList<>();

        ListNode curr = head;
        while (curr != null) {
            values.add(curr.val);
            curr = curr.next;
        }

        if (values.size() == 1)
            return true;

        int mid = (int) values.size() / 2;
        for (int i = 0; i <= mid; i++) {
            if (values.get(i) != values.get(values.size() - 1 - i))
                return false;
        }
        return true;
    }

    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(2);
        head.next.next.next = new ListNode(1);

        boolean result = isPalindrome(head);

        System.out.println("Is the linked list a palindrome? " + result);
    }
}
```

```
PS D:\Professional\Programming>  & 'C:\
ssages' '-cp' 'C:\Users\venka\AppData\R
1906a\bin' 'isPalindromeLL'
Is the linked list a palindrome? true
PS D:\Professional\Programming>
```

2. Balanced Binary Tree

```java
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
    }
}

public class BalancedTree {
    private static boolean status = true;
    private static int traversal(TreeNode root, int level) {
        if (root == null) return level;
        int leftSide = traversal(root.left, level + 1);
        int rightSide = traversal(root.right, level + 1);
        if (Math.abs(leftSide - rightSide) > 1)
            status = false;
        return Math.max(leftSide, rightSide);
    }
    public static boolean isBalanced(TreeNode root) {
        int height = traversal(root, 0);
        System.out.println("Height of the tree: " + height);
        return status;
    }
    public static void main(String[] args) {
        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);
        boolean result = isBalanced(root);
        System.out.println("Is the binary tree balanced? " + result);
    }
}
```

```
PS D:\Professional\Programming>  & '(
ssages' '-cp' 'C:\Users\venka\AppDat
1906a\bin' 'BalancedTree'
Height of the tree: 3
Is the binary tree balanced? true
PS D:\Professional\Programming>
```

3. Floor of the Array

```java
public class FloorArray {
    static int findFloor(int[] arr, int k) {
        int index = -1;
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (int) (low + high) / 2;
            if (arr[mid] <= k) {
                index = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return index;
    }

    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5, };
        int k = 4;
        System.out.println(findFloor(arr, k));
    }
}
```

```
PS D:\Professional\Programming>  & 'C:\
ssages' '-cp' 'C:\Users\venka\AppData\R
1906a\bin' 'FloorArray'
3
PS D:\Professional\Programming>
```

4. Check Equal Arrays

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class CheckEqualArrays {
    public static boolean check(int[] arr1, int[] arr2) {
        Map<Integer, Integer> freq1 = new HashMap<>();
        Map<Integer, Integer> freq2 = new HashMap<>();
        for (int elt1 : arr1) {
            freq1.put(elt1, freq1.getOrDefault(elt1, 0) + 1);
        }
        for (int elt2 : arr2) {
            freq2.put(elt2, freq2.getOrDefault(elt2, 0) + 1);
        }
        for (int key : freq1.keySet()) {
            if (!freq2.containsKey(key)
                || freq1.get(key) != freq2.get(key)) {
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int length = scan.nextInt();
        int[] arr1 = new int[length];
        int[] arr2 = new int[length];
        for (int i = 0; i < length; i++) arr1[i] = scan.nextInt();
        for (int i = 0; i < length; i++) arr2[i] = scan.nextInt();
        }
        System.out.println(check(arr1, arr2));
    }
}
```

```
PS D:\Professional\Programming>  & 'C:\
ssages' '-cp' 'C:\Users\venka\AppData\R
1906a\bin' 'CheckEqualArrays'
5
1 2 3 4 5
2 5 4 3 1
true
PS D:\Professional\Programming>
```

5. Triplet Sum in an Array (3 Sum)

```java
import java.util.HashSet;
import java.util.Set;
import java.util.Scanner;

public class TreeSum {
    public static boolean find3Numbers(int arr[], int n, int x) {
        for (int i = 0; i < arr.length; i++) {
            Set<Integer> set = new HashSet<>();
            for (int j = i + 1; j < arr.length; j++) {
                int sum = x - arr[i];
                int elt = arr[j];
                int remain = sum - elt;
                if (set.contains(remain)) {
                    return true;
                } else {
                    set.add(arr[j]);
                }
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int length = scan.nextInt();
        int[] arr = new int[length];
        for (int i = 0; i < length; i++) {
            arr[i] = scan.nextInt();
        }
        int sum = scan.nextInt();
        System.out.println(find3Numbers(arr, length, sum));
        scan.close();
    }
}
```

```
PS D:\Professional\Programming>  & 'C:\
ssages' '-cp' 'C:\Users\venka\AppData\R
1906a\bin' 'TreeSum'
6
1 4 45 6 10 8
13
true
PS D:\Professional\Programming>
```

6. Knapsack Problem

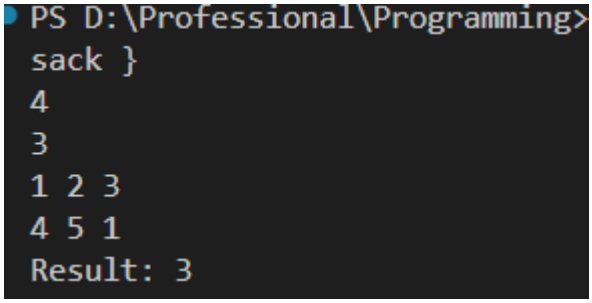
```java
import java.util.Scanner;

public class ZeroOneKnapsack {
    private static int maxProfit = 0;

    private static void backtracking(
         int[] wt, int[] val, int index, int profit, int capacity
    ) {
        if (index == wt.length) {
            maxProfit = Math.max(maxProfit, profit);
            return;
        }
        if (capacity - wt[index] >= 0) {
            maxProfit = Math.max(profit + val[index], maxProfit);
            int newProfit = profit + val[index];
            int newCap = capacity - wt[index];
            backtracking(wt, val, index + 1, newProfit, newCap);
        }
        backtracking(wt, val, index + 1, profit, capacity);
    }

    static int knapSack(int capacity, int val[], int wt[]) {
        maxProfit = 0;
        backtracking(wt, val, 0, 0, capacity);
        return maxProfit;
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int capacity = scan.nextInt();
        int length = scan.nextInt();
        int val[] = new int[length];
        int wt[] = new int[length];
        for (int i = 0; i < length; i++) {
            val[i] = scan.nextInt();
        }
        for (int i = 0; i < length; i++) {
            wt[i] = scan.nextInt();
        }
        System.out.println(knapSack(capacity, val, wt));
        scan.close();
    }
}
```


PS D:\Professional\Programming>
sack }
4
3
1 2 3
4 5 1
Result: 3