Q1: Check if an Element is Present in a Linked List

Problem: Given a linked list and a key `X`, check if `X` is present in the linked list or not.

Solution:

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class LinkedListSearch {
    // Function to check if an element is present in the linked list
    public static boolean searchElement(Node head, int key) {
        // Traverse the linked list
        while (head != null) {
            // If the current node's data is equal to the key, return true
            if (head.data == key) {
                return true;
            }
            head = head.next;
        }
        // If the key is not found, return false
        return false;
    }

    public static void main(String[] args) {
        // Creating a linked list: 14 -> 21 -> 11 -> 30 -> 10
        Node head = new Node(14);
        head.next = new Node(21);
        head.next.next = new Node(11);
        head.next.next.next = new Node(30);
        head.next.next.next.next = new Node(10);

        int X1 = 14;
        int X2 = 13;

        System.out.println(searchElement(head, X1) ? "Yes" : "No");  // Output: Yes
        System.out.println(searchElement(head, X2) ? "Yes" : "No");  // Output: No
    }
```

}

Q2: Insert a Node at a Given Position in a Linked List

Problem: Insert a node at the given position in a linked list. We are given a pointer to a node, and the new node is inserted after the given node.

Solution:

```
class LinkedListInsertion {
    // Function to insert a new node after a given node
    public static void insertAfter(Node prevNode, int newData) {
        // Check if the previous node is null
        if (prevNode == null) {
            System.out.println("The given previous node cannot be null.");
            return;
        }

        // Create the new node and put data in it
        Node newNode = new Node(newData);
        // Make the next of the new node as next of previous node
        newNode.next = prevNode.next;
        // Make the next of previous node as new node
        prevNode.next = newNode;
    }

    public static void printList(Node head) {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Creating a linked list: 1 -> 2 -> 4 -> 5 -> 6
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(4);
        head.next.next.next = new Node(5);
        head.next.next.next.next = new Node(6);
```

```
        // Pointer to the node with value 2 and new value to be inserted is 3
        Node pointer = head.next;  // Node with value 2
        int value = 3;

        // Insert the new node
        insertAfter(pointer, value);

        // Print the updated list: 1 -> 2 -> 3 -> 4 -> 5 -> 6
        printList(head);  // Output: 1 2 3 4 5 6
    }
}
```

Q3: Delete All Duplicates in a Sorted Linked List

Problem: Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Solution:

```
class LinkedListRemoveDuplicates {
    // Function to remove duplicates from a sorted linked list
    public static Node removeDuplicates(Node head) {
        Node current = head;

        // Traverse the linked list
        while (current != null && current.next != null) {
            // If current node's data is the same as the next node's data
            if (current.data == current.next.data) {
                // Skip the next node
                current.next = current.next.next;
            } else {
                // Move to the next node
                current = current.next;
            }
        }
        return head;
    }

    public static void printList(Node head) {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
```

```
        }
        System.out.println();
    }

    public static void main(String[] args) {
        // Creating a sorted linked list: 1 -> 2 -> 2 -> 3 -> 4 -> 4 -> 5
        Node head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(2);
        head.next.next.next = new Node(3);
        head.next.next.next.next = new Node(4);
        head.next.next.next.next.next = new Node(4);
        head.next.next.next.next.next.next = new Node(5);

        // Remove duplicates
        head = removeDuplicates(head);

        // Print the updated list: 1 -> 2 -> 3 -> 4 -> 5
        printList(head);  // Output: 1 2 3 4 5
    }
}
```

Q4: Check if a Linked List is a Palindrome

Problem: Given the head of a singly linked list, return `true` if it is a palindrome or `false` otherwise.

Solution:

```
import java.util.Stack;

class LinkedListPalindrome {
    // Function to check if a linked list is a palindrome
    public static boolean isPalindrome(Node head) {
        Stack<Integer> stack = new Stack<>();
        Node current = head;

        // Push all elements of the list to the stack
        while (current != null) {
            stack.push(current.data);
            current = current.next;
        }
```

```
        // Reinitialize the current pointer to the head
        current = head;
        // Compare elements of the list with the elements in the stack
        while (current != null) {
            if (current.data != stack.pop()) {
                return false;  // If any element is not the same, it is not a palindrome
            }
            current = current.next;
        }

        return true;  // If all elements are the same, it is a palindrome
    }

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

        // Check if the linked list is a palindrome
        System.out.println(isPalindrome(head));  // Output: true
    }
}
```

Q4. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class LinkedListPalindromeCheck {

    // Function to check if a linked list is a palindrome
    public static boolean isPalindrome(Node head) {
        if (head == null || head.next == null) {
            return true;
        }
```

```
    // Step 1: Find the middle of the linked list
    Node 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 linked list
    Node secondHalf = reverseList(slow);

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

    return true;
}

// Function to reverse a linked list
public static Node reverseList(Node head) {
    Node prev = null;
    Node current = head;
    while (current != null) {
        Node next = current.next;
        current.next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

public static void main(String[] args) {
    // Example 1: Input: 1 -> 2 -> 2 -> 1, Output: true
    Node head1 = new Node(1);
    head1.next = new Node(2);
    head1.next.next = new Node(2);
    head1.next.next.next = new Node(1);
    System.out.println(isPalindrome(head1));  // Output: true
```

```
        // Example 2: Input: 1 -> 2, Output: false
        Node head2 = new Node(1);
        head2.next = new Node(2);
        System.out.println(isPalindrome(head2));  // Output: false
    }
}
```

Q5. Given two numbers represented by two lists, write a function that returns the sum list. The sum list is a list representation of the addition of two input numbers.

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class LinkedListSum {

    // Function to add two numbers represented by linked lists
    public static Node addTwoLists(Node l1, Node l2) {
        Node dummyHead = new Node(0);
        Node current = dummyHead;
        int carry = 0;

        // Traverse both linked lists
        while (l1 != null || l2 != null) {
            int x = (l1 != null) ? l1.data : 0;
            int y = (l2 != null) ? l2.data : 0;
            int sum = carry + x + y;

            // Update carry for the next calculation
            carry = sum / 10;
            // Create a new node with the sum of the current digits
            current.next = new Node(sum % 10);
            current = current.next;

            // Move to the next nodes in both linked lists
            if (l1 != null) l1 = l1.next;
            if (l2 != null) l2 = l2.next;
```

```
    }

    // If there is still a carry left, add a new node with the carry
    if (carry > 0) {
        current.next = new Node(carry);
    }

    return dummyHead.next;
}

// Function to print the linked list
public static void printList(Node head) {
    Node current = head;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    // Example 1: Input: List1: 5->6->3, List2: 8->4->2
    Node list1 = new Node(5);
    list1.next = new Node(6);
    list1.next.next = new Node(3);

    Node list2 = new Node(8);
    list2.next = new Node(4);
    list2.next.next = new Node(2);

    Node result = addTwoLists(list1, list2);
    System.out.print("Resultant list: ");
    printList(result);  // Output: 1->4->0->5

    // Example 2: Input: List1: 7->5->9->4->6, List2: 8->4
    Node list3 = new Node(7);
    list3.next = new Node(5);
    list3.next.next = new Node(9);
    list3.next.next.next = new Node(4);
    list3.next.next.next.next = new Node(6);

    Node list4 = new Node(8);
    list4.next = new Node(4);
```

```
        Node result2 = addTwoLists(list3, list4);
        System.out.print("Resultant list: ");
        printList(result2);  // Output: 7->6->0->3->0
    }
}
```