Q1. Given a linked list and a key 'X' in, the task is to check if X is present in the linked list or not.

Examples:

Input: 14->21->11->30->10, X = 14

Output: Yes

Explanation: 14 is present in the linked list.

Ans:

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

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

class LinkedList {
    Node head;

    // Method to insert a new node at the end of the linked list
    void insert(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }

    // Method to check if a key 'X' is present in the linked list
    boolean search(int key) {
        Node current = head;
        while (current != null) {
            if (current.data == key) {
                return true; // Key found
            }
            current = current.next;
        }
```

```
        return false; // Key not found
    }
}

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();

        // Inserting elements into the linked list
        linkedList.insert(1);
        linkedList.insert(2);
        linkedList.insert(3);
        linkedList.insert(4);
        linkedList.insert(5);

        int keyToSearch = 3;

        // Checking if the key is present in the linked list
        if (linkedList.search(keyToSearch)) {
            System.out.println(keyToSearch + " is present in the linked list.");
        } else {
            System.out.println(keyToSearch + " is not present in the linked list.");
        }
    }
}
```

Q2. 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.

Input : LL = 1 -> 2 -> 4 -> 5 -> 6 pointer = 2 value = 3.
Output : 1 -> 2 -> 3 -> 4 -> 5 -> 6

Ans:

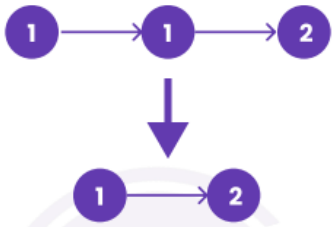
```
class Node {
    int data;
    Node next;

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

```
class LinkedList {
    Node head;

    // Method to insert a new node after a given node
    void insertAfter(Node prevNode, int newData) {
        if (prevNode == null) {
            System.out.println("Previous node cannot be null.");
            return;
        }

        Node newNode = new Node(newData);
        newNode.next = prevNode.next;
        prevNode.next = newNode;
    }

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

public class Main {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();

        // Inserting elements into the linked list
        linkedList.insert(1);
        linkedList.insert(2);
        linkedList.insert(4);

        System.out.println("Linked List before insertion:");
        linkedList.printList();

        // Get a pointer to a node (let's say the node with data 2)
        Node prevNode = linkedList.head.next;

        // Inserting a new node after the given node
        linkedList.insertAfter(prevNode, 3);
```

```
        System.out.println("Linked List after insertion:");
        linkedList.printList();
    }
}
```

Q3. 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.

Example 1:



Input: head = [1,1,2]
Output: [1,2]

Ans:

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

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

public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode current = head;

        while (current != null && current.next != null) {
            if (current.val == current.next.val) {
                current.next = current.next.next; // Skip the duplicate node
            } else {
                current = current.next; // Move to the next distinct element
            }
        }
```

```
        return head;
    }

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

    public static void main(String[] args) {
        // Example usage
        ListNode head = new ListNode(1);
        head.next = new ListNode(1);
        head.next.next = new ListNode(2);
        head.next.next.next = new ListNode(3);
        head.next.next.next.next = new ListNode(3);

        System.out.println("Linked List before removing duplicates:");
        printList(head);

        Solution solution = new Solution();
        ListNode result = solution.deleteDuplicates(head);

        System.out.println("Linked List after removing duplicates:");
        printList(result);
    }
}
```

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

Example 1:

Input: head = [1,2,2,1]

Output: true

Ans:

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

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

public class Solution {
    public boolean isPalindrome(ListNode head) {
        if (head == null || head.next == null) {
            return true; // An empty list or a list with a single node is a palindrome
        }

        // Step 1: Find the middle of the linked list
        ListNode slow = head;
        ListNode 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
        ListNode reversedSecondHalf = reverseList(slow);

        // Step 3: Compare the reversed second half with the first half
        while (reversedSecondHalf != null) {
            if (head.val != reversedSecondHalf.val) {
                return false; // Not a palindrome
            }
            head = head.next;
            reversedSecondHalf = reversedSecondHalf.next;
        }

        return true; // It is a palindrome
    }

    // Helper method to reverse a linked list
    private ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode current = head;
```

```
        while (current != null) {
            ListNode nextNode = current.next;
            current.next = prev;
            prev = current;
            current = nextNode;
        }

        return prev;
    }

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

        Solution solution = new Solution();
        boolean result = solution.isPalindrome(head);

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

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.

Example:

Input:

List1: 5->6->3 // represents number 563

List2: 8->4->2 // represents number 842

Output:

Resultant list: 1->4->0->5 // represents number 1405

Ans:

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

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

```
    }
}

public class Solution {
    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        ListNode dummyHead = new ListNode(0);
        ListNode current = dummyHead;
        int carry = 0;

        while (l1 != null || l2 != null || carry != 0) {
            int sum = carry;

            if (l1 != null) {
                sum += l1.val;
                l1 = l1.next;
            }

            if (l2 != null) {
                sum += l2.val;
                l2 = l2.next;
            }

            current.next = new ListNode(sum % 10);
            carry = sum / 10;

            current = current.next;
        }

        return dummyHead.next;
    }

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

    public static void main(String[] args) {
        // Example usage
        ListNode l1 = new ListNode(2);
```

```
        l1.next = new ListNode(4);
        l1.next.next = new ListNode(3);

        ListNode l2 = new ListNode(5);
        l2.next = new ListNode(6);
        l2.next.next = new ListNode(4);

        Solution solution = new Solution();
        ListNode result = solution.addTwoNumbers(l1, l2);

        System.out.println("Sum of the two numbers:");
        printList(result);
    }
}
```