Subject: Algorithm and Data Structure
Assignment 3

1. Implement a singly linked list with basic operations: insert, delete, search.

- **Test Case 1**:
  Input: Insert 3 → Insert 7 → Insert 5 → Delete 7 → Search 5
  Output: List = [3, 5], Found = True
- **Test Case 2**:
  Input: Insert 9 → Insert 4 → Delete 4 → Search 10
  Output: List = [9], Found = False

Code –

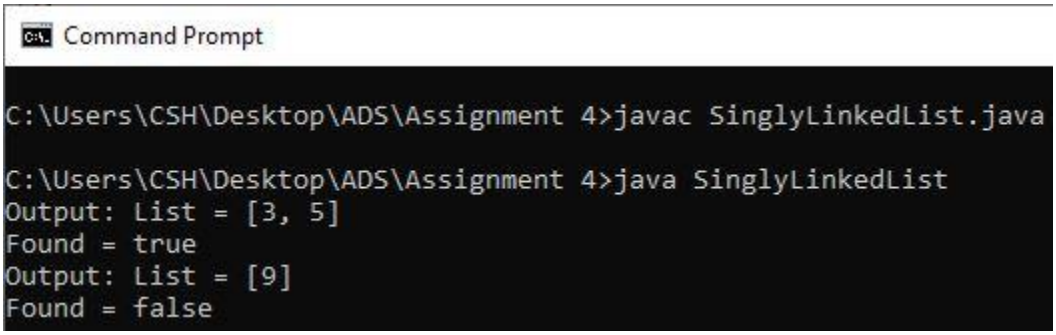
```
class SinglyLinkedList {


   private class Node {

      int data;

      Node next;


      public Node(int data) {

         this.data = data;

         this.next = null;

      }

   }


   private Node head;


   public 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;
    }
}

public void delete(int data) {
    if (head == null) {
        System.out.println("List is empty.");
        return;
    }

    if (head.data == data) {
        head = head.next;
        return;
    }

    Node temp = head;
    while (temp.next != null && temp.next.data != data) {
        temp = temp.next;
    }

    if (temp.next != null) {
        temp.next = temp.next.next;
    } else {
        System.out.println("Node with value " + data + " not found.");
    }
}

public boolean search(int data) {
```

```
    Node temp = head;
    while (temp != null) {
        if (temp.data == data) {
            return true;
        }
        temp = temp.next;
    }
    return false;
}

public void printList() {
    Node temp = head;
    System.out.print("Output: " + "List = [");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(", ");
        }
        temp = temp.next;
    }
    System.out.println("]");
}

public static void main(String[] args) {

    SinglyLinkedList list1 = new SinglyLinkedList();
    list1.insert(3);
    list1.insert(7);
    list1.insert(5);
```

```
        list1.delete(7);
        list1.printList();
        System.out.println("Found = " + list1.search(5));

        SinglyLinkedList list2 = new SinglyLinkedList();
        list2.insert(9);
        list2.insert(4);
        list2.delete(4);
        list2.printList();
        System.out.println("Found = " + list2.search(10));
    }
}
```

Output –



2. Reverse a singly linked list.

- **Test Case 1**:
  Input: List = [1, 2, 3, 4, 5]
  Output: List = [5, 4, 3, 2, 1]
- **Test Case 2**:
  Input: List = [10, 20, 30]
  Output: List = [30, 20, 10]

Code –

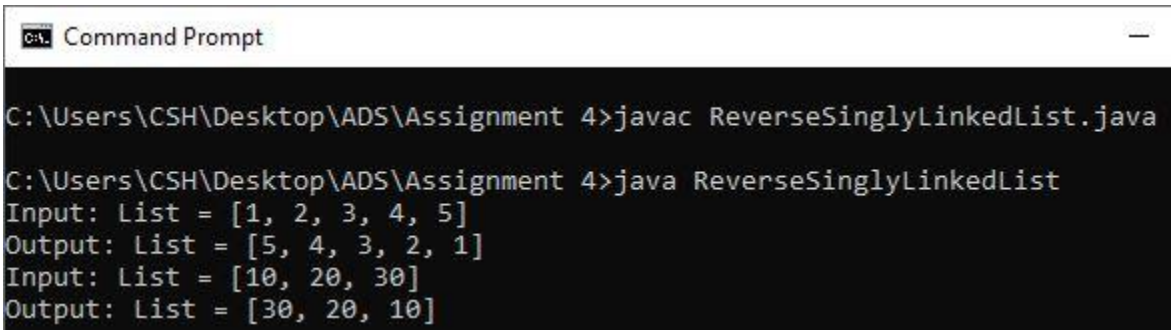
```
class ReverseSinglyLinkedList {

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

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

private Node head;

public 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;
    }
}

public void reverse() {
    Node prev = null;
    Node current = head;
    Node next = null;
    while (current != null) {
        next = current.next;
        current.next = prev;
```

```
        prev = current;
        current = next;
    }
    head = prev;
}

public void printList() {
    Node temp = head;
    System.out.print("List = [");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(", ");
        }
        temp = temp.next;
    }
    System.out.println("]");
}

public static void main(String[] args) {

    ReverseSinglyLinkedList list1 = new ReverseSinglyLinkedList();
    list1.insert(1);
    list1.insert(2);
    list1.insert(3);
    list1.insert(4);
    list1.insert(5);
    System.out.print("Input: ");
    list1.printList();
```

```
        list1.reverse();
        System.out.print("Output: ");
        list1.printList();

        ReverseSinglyLinkedList list2 = new ReverseSinglyLinkedList();
        list2.insert(10);
        list2.insert(20);
        list2.insert(30);
        System.out.print("Input: ");
        list2.printList();
        list2.reverse();
        System.out.print("Output: ");
        list2.printList();
    }
}
```

Output –



3. Detect a cycle in a linked list.

- **Test Case 1**:
  Input: List = [1 → 2 → 3 → 4 → 5 → 3 (cycle)]
  Output: Cycle Detected
- **Test Case 2**:
  Input: List = [6 → 7 → 8 → 9]
  Output: No Cycle

Code –

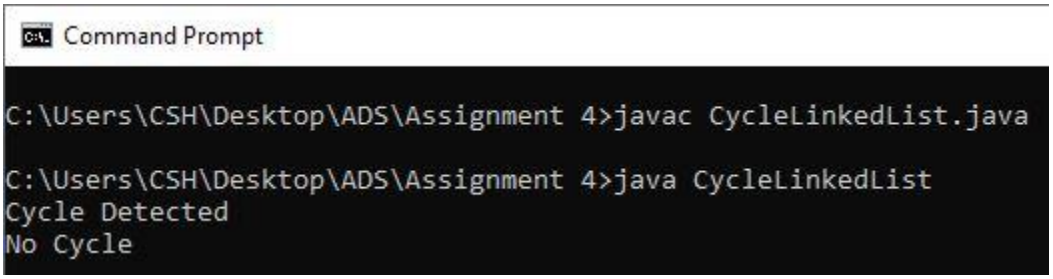
```
class CycleLinkedList {
```

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

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

private Node head;

public 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;
    }
}

public void createCycle(int pos) {
    if (head == null) return;

    Node temp = head;
```

```
    Node cycleNode = null;
    int count = 0;

    while (temp.next != null) {
        if (count == pos) {
            cycleNode = temp;
        }
        temp = temp.next;
        count++;
    }

    temp.next = cycleNode;
}

public boolean detectCycle() {
    Node slow = head;
    Node fast = head;

    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;

        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```

```
    public static void main(String[] args) {

        CycleLinkedList list1 = new CycleLinkedList();
        list1.insert(1);
        list1.insert(2);
        list1.insert(3);
        list1.insert(4);
        list1.insert(5);
        list1.createCycle(2);
        if (list1.detectCycle()) {
            System.out.println("Cycle Detected");
        } else {
            System.out.println("No Cycle");
        }

        CycleLinkedList list2 = new CycleLinkedList();
        list2.insert(6);
        list2.insert(7);
        list2.insert(8);
        list2.insert(9);
        if (list2.detectCycle()) {
            System.out.println("Cycle Detected");
        } else {
            System.out.println("No Cycle");
        }
    }
}
```

Output –



4. Merge two sorted linked lists.

- **Test Case 1**:
  Input: List1 = [1, 3, 5], List2 = [2, 4, 6]
  Output: Merged List = [1, 2, 3, 4, 5, 6]
- **Test Case 2**:
  Input: List1 = [10, 15, 20], List2 = [12, 18, 25]
  Output: Merged List = [10, 12, 15, 18, 20, 25]

Code –

```
class SortedLinkedList {


   private class Node {

      int data;

      Node next;


      public Node(int data) {

         this.data = data;

         this.next = null;

      }

   }


   private Node head;


   public void insert(int data) {

      Node newNode = new Node(data);

      if (head == null) {

         head = newNode;
```

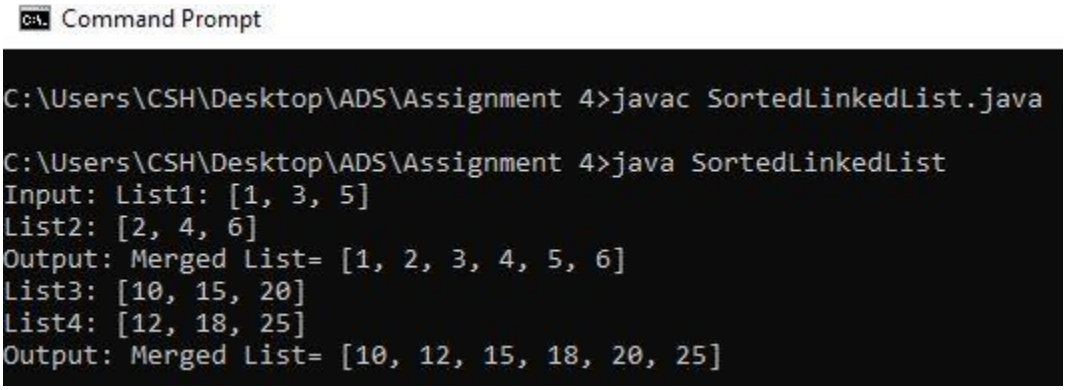
```
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }
}

public static SortedLinkedList merge(SortedLinkedList list1, SortedLinkedList list2) {
    SortedLinkedList mergedList = new SortedLinkedList();
    Node p1 = list1.head;
    Node p2 = list2.head;

    while (p1 != null && p2 != null) {
        if (p1.data <= p2.data) {
            mergedList.insert(p1.data);
            p1 = p1.next;
        } else {
            mergedList.insert(p2.data);
            p2 = p2.next;
        }
    }

    while (p1 != null) {
        mergedList.insert(p1.data);
        p1 = p1.next;
    }
    while (p2 != null) {
```

```
            mergedList.insert(p2.data);
            p2 = p2.next;
        }

        return mergedList;
    }

    public void printList() {
        Node temp = head;
        System.out.print("[");
        while (temp != null) {
            System.out.print(temp.data);
            if (temp.next != null) {
                System.out.print(", ");
            }
            temp = temp.next;
        }
        System.out.println("]");
    }

    public static void main(String[] args) {

        SortedLinkedList list1 = new SortedLinkedList();
        list1.insert(1);
        list1.insert(3);
        list1.insert(5);

        SortedLinkedList list2 = new SortedLinkedList();
        list2.insert(2);
```

```
list2.insert(4);
list2.insert(6);

System.out.print("Input: " + "List1: ");
list1.printList();
System.out.print("List2: ");
list2.printList();

SortedLinkedList mergedList1 = SortedLinkedList.merge(list1, list2);
System.out.print("Output: " + "Merged List= ");
mergedList1.printList();

SortedLinkedList list3 = new SortedLinkedList();
list3.insert(10);
list3.insert(15);
list3.insert(20);

SortedLinkedList list4 = new SortedLinkedList();
list4.insert(12);
list4.insert(18);
list4.insert(25);

System.out.print("List3: ");
list3.printList();
System.out.print("List4: ");
list4.printList();

SortedLinkedList mergedList2 = SortedLinkedList.merge(list3, list4);
System.out.print("Output: " + "Merged List= ");
```

```
        mergedList2.printList();
    }
}
```

Output –



5. Find the nth node from the end of a linked list.

- **Test Case 1**:
  Input: List = [10, 20, 30, 40, 50], n = 2
  Output: 40
- **Test Case 2**:
  Input: List = [5, 15, 25, 35], n = 4
  Output: 5

Code –

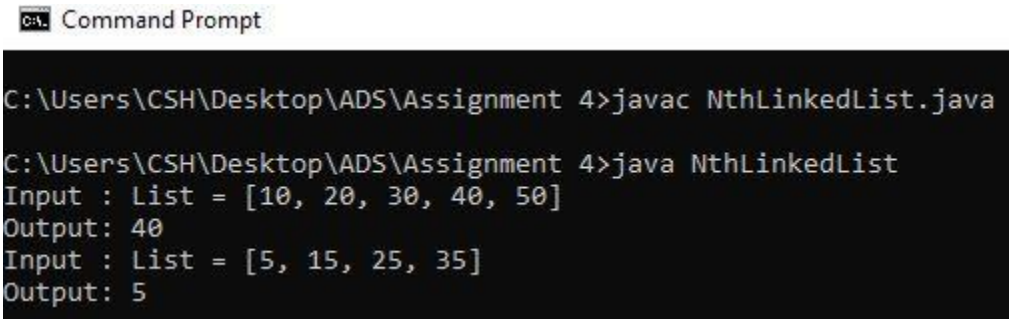
```
class NthLinkedList {


    private class Node {
        int data;
        Node next;


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


    private Node head;
```

```
public 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;
    }
}

public int findNthFromEnd(int n) {
    Node first = head;
    Node second = head;

    for (int i = 0; i < n; i++) {
        if (first == null) {
            throw new IllegalArgumentException("List is shorter than " + n + " nodes.");
        }
        first = first.next;
    }

    while (first != null) {
        first = first.next;
        second = second.next;
    }
```

```
    return second.data;
}

public void printList() {
    Node temp = head;
    System.out.print("Input : List = [");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(", ");
        }
        temp = temp.next;
    }
    System.out.println("]");
}

public static void main(String[] args) {

    NthLinkedList list1 = new NthLinkedList();
    list1.insert(10);
    list1.insert(20);
    list1.insert(30);
    list1.insert(40);
    list1.insert(50);

    list1.printList();
    int n1 = 2;
    System.out.println("Output: " + list1.findNthFromEnd(n1));
```

```
        NthLinkedList list2 = new NthLinkedList();
        list2.insert(5);
        list2.insert(15);
        list2.insert(25);
        list2.insert(35);

        list2.printList();
        int n2 = 4;
        System.out.println("Output: " + list2.findNthFromEnd(n2));
    }
}
```

Output –



6. Remove duplicates from a sorted linked list.

- **Test Case 1**:
  Input: List = [1, 1, 2, 3, 3, 4]
  Output: List = [1, 2, 3, 4]
- **Test Case 2**:
  Input: List = [7, 7, 8, 9, 9, 10]
  Output: List = [7, 8, 9, 10]

Code –

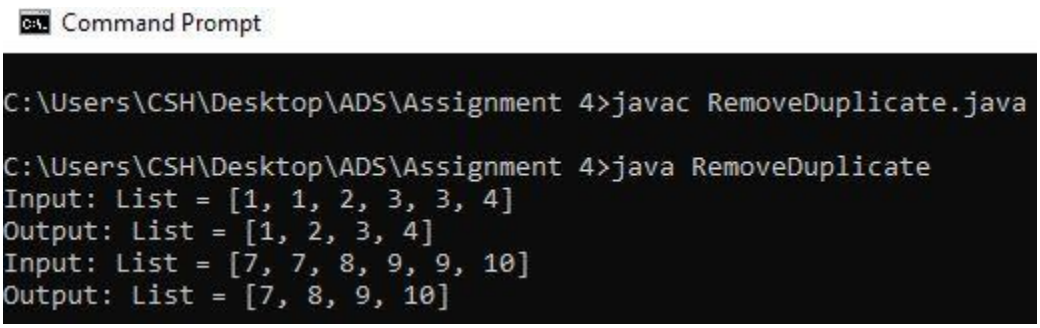
```
class RemoveDuplicate {

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

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

private Node head;

public 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;
    }
}

public void removeDuplicates() {
    Node current = head;

    while (current != null && current.next != null) {
        if (current.data == current.next.data) {
            current.next = current.next.next;
        } else {
```

```
            current = current.next;
        }
    }
}

public void printList() {
    Node temp = head;
    System.out.print("List = [");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(", ");
        }
        temp = temp.next;
    }
    System.out.println("]");
}

public static void main(String[] args) {

   RemoveDuplicate list1 = new RemoveDuplicate();
    list1.insert(1);
    list1.insert(1);
    list1.insert(2);
    list1.insert(3);
    list1.insert(3);
    list1.insert(4);

    System.out.print("Input: ");
```

```
        list1.printList();
        list1.removeDuplicates();
        System.out.print("Output: ");
        list1.printList();

        RemoveDuplicate list2 = new RemoveDuplicate();
        list2.insert(7);
        list2.insert(7);
        list2.insert(8);
        list2.insert(9);
        list2.insert(9);
        list2.insert(10);

        System.out.print("Input: ");
        list2.printList();
        list2.removeDuplicates();
        System.out.print("Output: ");
        list2.printList();
    }
}
```

Output –



7. Implement a doubly linked list with insert, delete, and traverse operations.

- **Test Case 1**:
  Input: Insert 10 → Insert 20 → Insert 30 → Delete 20
  Output: List = [10, 30]
- **Test Case 2**:
  Input: Insert 1 → Insert 2 → Insert 3 → Delete 1
  Output: List = [2, 3]

Code –

```
class DoublyLinkedList {

   private class Node {
      int data;
      Node next;
      Node prev;

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

   private Node head;

   public 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;
         newNode.prev = temp;
      }
   }

   public void delete(int data) {
      if (head == null) {
         System.out.println("List is empty.");
         return;
      }

      Node temp = head;

      if (temp.data == data) {
```

```
            head = head.next;
            if (head != null) {
                head.prev = null;
            }
            return;
        }
        while (temp != null && temp.data != data) {
            temp = temp.next;
        }
        if (temp == null) {
            System.out.println("Node with data " + data + " not found.");
            return;
        }
        if (temp.next != null) {
            temp.next.prev = temp.prev;
        }
        if (temp.prev != null) {
            temp.prev.next = temp.next;
        }
    }

    public void traverse() {
        Node temp = head;
        System.out.print("List = [");
        while (temp != null) {
            System.out.print(temp.data);
            if (temp.next != null) {
                System.out.print(", ");
            }
            temp = temp.next;
        }
        System.out.println("]");
    }

    public static void main(String[] args) {

        DoublyLinkedList list1 = new DoublyLinkedList();
        list1.insert(10);
        list1.insert(20);
        list1.insert(30);

        System.out.print("Input: ");
        list1.traverse();
        list1.delete(20);
        System.out.print("Output: ");
        list1.traverse();

        DoublyLinkedList list2 = new DoublyLinkedList();
```

```
        list2.insert(1);
        list2.insert(2);
        list2.insert(3);

        System.out.print("Input: ");
        list2.traverse();
        list2.delete(1);
        System.out.print("Output: ");
        list2.traverse();
    }
}
```

Output –

Command Prompt

```
C:\Users\CSH\Desktop\ADS\Assignment 4>javac DoublyLinkedList.java

C:\Users\CSH\Desktop\ADS\Assignment 4>java DoublyLinkedList
Input: List = [10, 20, 30]
Output: List = [10, 30]
Input: List = [1, 2, 3]
Output: List = [2, 3]
```

8. Reverse a doubly linked list.

- **Test Case 1**:
  Input: List = [5, 10, 15, 20]
  Output: List = [20, 15, 10, 5]
- **Test Case 2**:
  Input: List = [4, 8, 12]
  Output: List = [12, 8, 4]

Code –

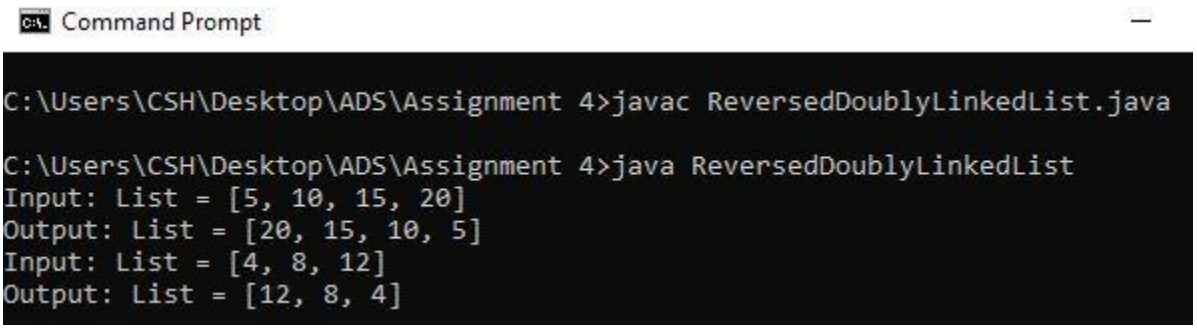
```
class ReversedDoublyLinkedList {


    private class Node {

        int data;

        Node next;

        Node prev;


        public Node(int data) {

            this.data = data;

            this.next = null;

            this.prev = null;
```

```
    }
}

private Node head;
public 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;
        newNode.prev = temp;
    }
}
public void reverse() {
    Node current = head;
    Node temp = null;

    while (current != null) {
        temp = current.prev;
        current.prev = current.next;
        current.next = temp;
        current = current.prev;
    }
    if (temp != null) {
        head = temp.prev;
```

```
    }
}
public void traverse() {
    Node temp = head;
    System.out.print("List = [");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(", ");
        }
        temp = temp.next;
    }
    System.out.println("]");
}

public static void main(String[] args) {

    ReversedDoublyLinkedList list1 = new ReversedDoublyLinkedList();
    list1.insert(5);
    list1.insert(10);
    list1.insert(15);
    list1.insert(20);

    System.out.print("Input: ");
    list1.traverse();
    list1.reverse();
    System.out.print("Output: ");
    list1.traverse();
```

```
        ReversedDoublyLinkedList list2 = new ReversedDoublyLinkedList();
        list2.insert(4);
        list2.insert(8);
        list2.insert(12);

        System.out.print("Input: ");
        list2.traverse();
        list2.reverse();
        System.out.print("Output: ");
        list2.traverse();
    }
}
```

Output –



9.Add two numbers represented by linked lists.

- **Test Case 1**:
  Input: List1 = [2 → 4 → 3], List2 = [5 → 6 → 4] (243 + 465)
  Output: Sum List = [7 → 0 → 8]
- **Test Case 2**:
  Input: List1 = [9 → 9 → 9], List2 = [1] (999 + 1)
  Output: Sum List = [0 → 0 → 0 → 1]

Code –

```
class SumOfNumber {

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

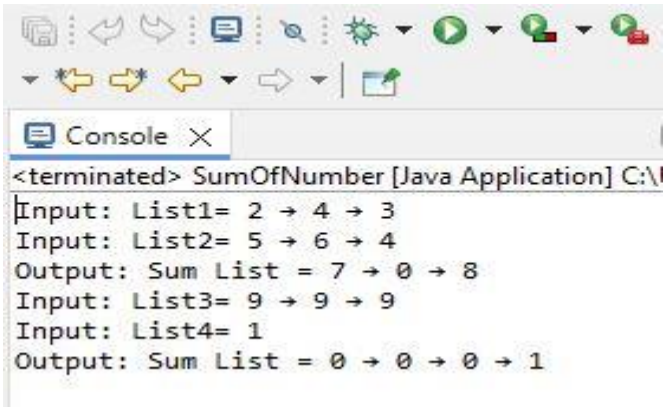
```
    public Node(int data) {
        this.data = data;
        this.next = null;
    }
}

private Node head;
public 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;
    }
}
public static SumOfNumber addTwoLists(SumOfNumber list1, SumOfNumber list2) {
    Node p1 = list1.head;
    Node p2 = list2.head;
    SumOfNumber result = new SumOfNumber();

    int carry = 0;
    while (p1 != null || p2 != null || carry != 0) {
        int sum = carry;
```

```
        if (p1 != null) {
            sum += p1.data;
            p1 = p1.next;
        }

        if (p2 != null) {
            sum += p2.data;
            p2 = p2.next;
        }

        carry = sum / 10;

        result.insert(sum % 10);
    }

    return result;
}

public void traverse() {
    Node temp = head;
    System.out.print("List = ");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(" → ");
        }
        temp = temp.next;
    }
    System.out.println();
```

```
}

public static void main(String[] args) {

    SumOfNumber list1 = new SumOfNumber();
    list1.insert(2);
    list1.insert(4);
    list1.insert(3);

    SumOfNumber list2 = new SumOfNumber();
    list2.insert(5);
    list2.insert(6);
    list2.insert(4);

    System.out.print("Input List1: ");
    list1.traverse();
    System.out.print("Input List2: ");
    list2.traverse();

    SumOfNumber sumList1 = addTwoLists(list1, list2);
    System.out.print("Output: Sum List = ");
    sumList1.traverse();

    SumOfNumber list3 = new SumOfNumber();
    list3.insert(9);
    list3.insert(9);
    list3.insert(9);

    SumOfNumber list4 = new SumOfNumber();
```

```
        list4.insert(1);


        System.out.print("Input List3: ");
        list3.traverse();
        System.out.print("Input List4: ");
        list4.traverse();


        SumOfNumber sumList2 = addTwoLists(list3, list4);
        System.out.print("Output: Sum List = ");
        sumList2.traverse();
    }
}
```

Output –



10. Rotate a linked list by k places.

- **Test Case 1**:
  Input: List = [10, 20, 30, 40, 50], k = 2
  Output: List = [30, 40, 50, 10, 20]
- **Test Case 2**:
  Input: List = [5, 10, 15, 20], k = 3
  Output: List = [20, 5, 10, 15]

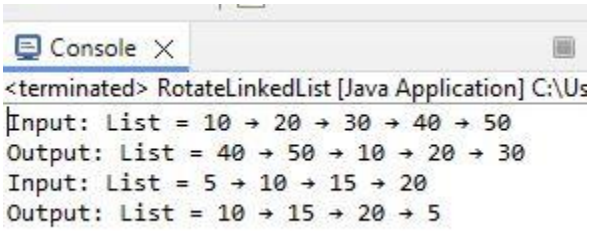
Code –

```
package ads;

public class RotateLinkedList {
	private class Node {
	  int data;
	  Node next;
```

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

private Node head;
public 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;
    }
}
public void rotate(int k) {
    if (head == null || head.next == null || k <= 0) {
        return;
    }
    Node temp = head;
    int length = 1;
    while (temp.next != null) {
        temp = temp.next;
        length++;
    }
    temp.next = head;

    k = k % length;
    int stepsToNewHead = length - k;

    Node newTail = head;
    for (int i = 0; i < stepsToNewHead - 1; i++) {
        newTail = newTail.next;
    }
    head = newTail.next;
    newTail.next = null;
}
public void traverse() {
    Node temp = head;
    System.out.print("List = ");
    while (temp != null) {
        System.out.print(temp.data);
        if (temp.next != null) {
            System.out.print(" → ");
        }
        temp = temp.next;
    }
    System.out.println();
}

   public static void main(String[] args) {
```

```
            RotateLinkedList list1 = new RotateLinkedList();
        list1.insert(10);
        list1.insert(20);
        list1.insert(30);
        list1.insert(40);
        list1.insert(50);

        System.out.print("Input: ");
        list1.traverse();
        list1.rotate(2);
        System.out.print("Output: ");
        list1.traverse();

        // Test Case 2
        RotateLinkedList list2 = new RotateLinkedList();
        list2.insert(5);
        list2.insert(10);
        list2.insert(15);
        list2.insert(20);

        System.out.print("Input: ");
        list2.traverse();
        list2.rotate(3);
        System.out.print("Output: ");
        list2.traverse();
    }
}
```

Output –



11. Flatten a multilevel doubly linked list.

- **Test Case 1**:
  Input: List = [1 → 2 → 3, 3 → 7 → 8, 8 → 10 → 12]
  Output: Flattened List = [1 → 2 → 3 → 7 → 8 → 10 → 12]
- **Test Case 2**:
  Input: List = [1 → 2 → 3, 2 → 5 → 6, 6 → 7 → 9]
  Output: Flattened List = [1 → 2 → 5 → 6 → 7 → 9 → 3]

Code –

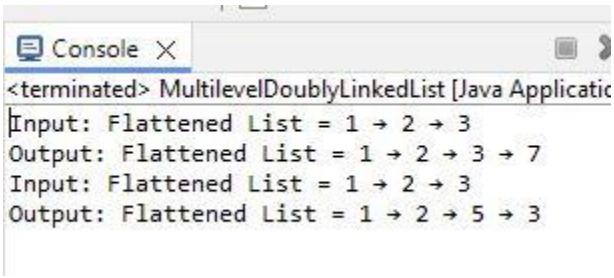
```
package ads;

public class MultilevelDoublyLinkedList {
        private class Node {
                int data;
                Node next;
```

```
    Node prev;
    Node child;

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

private Node head;

public 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;
        newNode.prev = temp;
    }
}
public void addChild(int parentData, int childData) {
    Node parent = findNode(parentData);
    if (parent != null) {
        Node childNode = new Node(childData);
        childNode.next = parent.child;
        parent.child = childNode;
        if (childNode.next != null) {
            childNode.next.prev = childNode;
        }
    }
}
private Node findNode(int data) {
    Node temp = head;
    while (temp != null) {
        if (temp.data == data) {
            return temp;
        }
        temp = temp.next;
    }
    return null;
}
public Node flatten() {
    return flattenList(head);
}
private Node flattenList(Node node) {
    Node curr = node;
    Node tail = node;

    while (curr != null) {
```

```
            if (curr.child != null) {
                Node childTail = flattenList(curr.child);
                Node nextNode = curr.next;

                curr.next = curr.child;
                curr.child.prev = curr;

                childTail.next = nextNode;

                if (nextNode != null) {
                    nextNode.prev = childTail;
                }
                curr.child = null;

                tail = childTail;
            } else {
                tail = curr;
            }
            curr = curr.next;
        }

        return tail;
    }

    public void traverse() {
        Node temp = head;
        System.out.print("Flattened List = ");
        while (temp != null) {
            System.out.print(temp.data);
            if (temp.next != null) {
                System.out.print(" → ");
            }
            temp = temp.next;
        }
        System.out.println();
    }
public static void main(String[] args) {
        MultilevelDoublyLinkedList list1 = new MultilevelDoublyLinkedList();
        list1.insert(1);
        list1.insert(2);
        list1.insert(3);
        list1.addChild(3, 7);
        list1.addChild(7, 8);
        list1.addChild(8, 10);
        list1.addChild(10, 12);

        System.out.print("Input: ");
        list1.traverse();
        list1.flatten();
        System.out.print("Output: ");
        list1.traverse();

        // Test Case 2
        MultilevelDoublyLinkedList list2 = new MultilevelDoublyLinkedList();
        list2.insert(1);
```

```
            list2.insert(2);
            list2.insert(3);
            list2.addChild(2, 5);
            list2.addChild(5, 6);
            list2.addChild(6, 7);
            list2.addChild(7, 9);

            System.out.print("Input: ");
            list2.traverse();
            list2.flatten();
            System.out.print("Output: ");
            list2.traverse();
        }
    }
```

Output –



12. Split a circular linked list into two halves.

- **Test Case 1**:
  Input: Circular List = [1 → 2 → 3 → 4 → 5 → 6 → (back to 1)]
  Output: List1 = [1 → 2 → 3], List2 = [4 → 5 → 6]
- **Test Case 2**:
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)]
  Output: List1 = [10 → 20], List2 = [30 → 40]

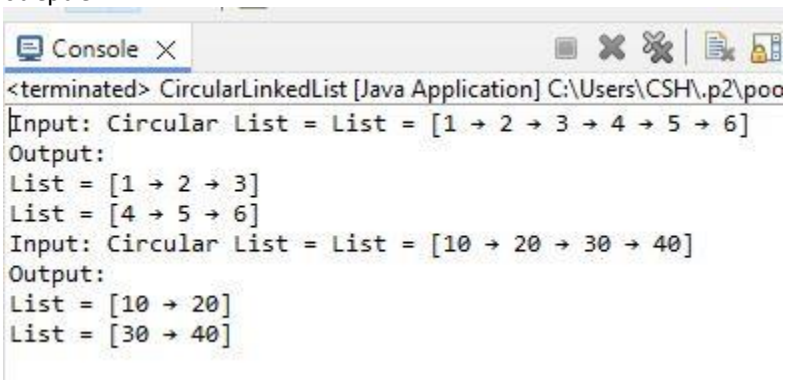
Code –

```
package ads;

public class CircularLinkedList {
    private class Node {
      int data;
      Node next;

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

  private Node head;
  public void insert(int data) {
      Node newNode = new Node(data);
      if (head == null) {
          head = newNode;
          newNode.next = head;
```

```
        } else {
            Node temp = head;
            while (temp.next != head) {
                temp = temp.next;
            }
            temp.next = newNode;
            newNode.next = head;
        }
    }
    public void split() {
        if (head == null) return;

        Node slow = head;
        Node fast = head;

        while (fast.next != head && fast.next.next != head) {
            slow = slow.next;
            fast = fast.next.next;
        }
        Node head1 = head;
        Node head2 = slow.next;
        slow.next = head1;
        Node temp = head2;

        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = head2;

        printList(head1);
        printList(head2);
    }
    public void printList(Node start) {
        if (start == null) return;

        Node temp = start;
        System.out.print("List = [");
        do {
            System.out.print(temp.data);
            temp = temp.next;
            if (temp != start) {
                System.out.print(" → ");
            }
        } while (temp != start);
        System.out.println("]");
    }
      public static void main(String[] args) {
             CircularLinkedList list1 = new CircularLinkedList();
              list1.insert(1);
              list1.insert(2);
              list1.insert(3);
              list1.insert(4);
              list1.insert(5);
              list1.insert(6);
```

```
            System.out.print("Input: Circular List = ");
            list1.printList(list1.head);
            System.out.println("Output: ");
            list1.split();

            CircularLinkedList list2 = new CircularLinkedList();
            list2.insert(10);
            list2.insert(20);
            list2.insert(30);
            list2.insert(40);

            System.out.print("Input: Circular List = ");
            list2.printList(list2.head);
            System.out.println("Output: ");
            list2.split();
        }
    }
```

Output –



13. Insert a node in a sorted circular linked list.

- **Test Case 1**:
  Input: Circular List = [10 → 20 → 30 → 40 → (back to 10)], Insert 25
  Output: Circular List = [10 → 20 → 25 → 30 → 40 → (back to 10)]
- **Test Case 2**:
  Input: Circular List = [5 → 15 → 25 → (back to 5)], Insert 10
  Output: Circular List = [5 → 10 → 15 → 25 → (back to 5)]

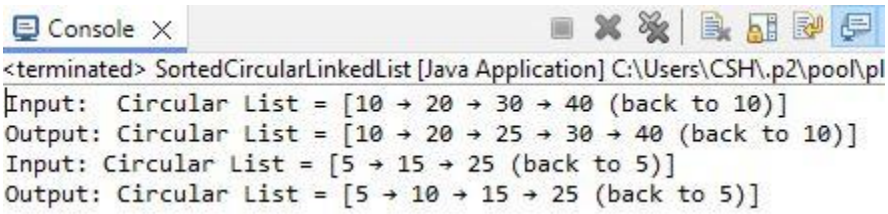
Code –

```
package ads;

public class SortedCircularLinkedList {
    private class Node {
      int data;
      Node next;

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

  private Node head;
  public void insert(int data) {
```

```
    Node newNode = new Node(data);
    if (head == null) {
        head = newNode;
        newNode.next = head;
        return;
    }
    if (data < head.data) {
        Node temp = head;
        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = newNode;
        newNode.next = head;
        head = newNode;
        return;
    }
    Node current = head;
    while (current.next != head && current.next.data < data) {
        current = current.next;
    }
    newNode.next = current.next;
    current.next = newNode;
}
public void printList() {
    if (head == null) return;

    Node temp = head;
    System.out.print("Circular List = [");
    do {
        System.out.print(temp.data);
        temp = temp.next;
        if (temp != head) {
            System.out.print(" → ");
        }
    } while (temp != head);
    System.out.println(" (back to " + head.data + ")]");
}

   public static void main(String[] args) {
          SortedCircularLinkedList list1 = new SortedCircularLinkedList();
          list1.insert(10);
          list1.insert(20);
          list1.insert(30);
          list1.insert(40);

          System.out.print("Input:  ");
          list1.printList();
          list1.insert(25);
          System.out.print("Output: ");
          list1.printList();

          SortedCircularLinkedList list2 = new SortedCircularLinkedList();
          list2.insert(5);
          list2.insert(15);
          list2.insert(25);
```

```
            System.out.print("Input: ");
            list2.printList();
            list2.insert(10);
            System.out.print("Output: ");
            list2.printList();
        }
    }
```

Output –



14. Check if two linked lists intersect, and find the intersection point if they do.

- **Test Case 1**:
  Input: List1 = [1 → 2 → 3 → 4 → 5], List2 = [6 → 7 → 4 → 5]
  Output: Intersection Point = 4
- **Test Case 2**:
  Input: List1 = [10 → 20 → 30 → 40], List2 = [15 → 25 → 35]
  Output: No Intersection

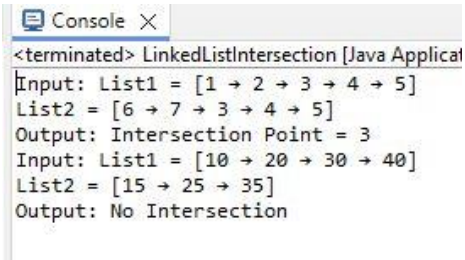
Code –

```
package ads;

public class LinkedListIntersection {
    private static class Node {
      int data;
      Node next;

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

  private Node head;
  public 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;
      }
  }
```

```java
public static Integer getIntersectionPoint(Node head1, Node head2) {
    if (head1 == null || head2 == null) {
        return null;
    }

    Node current1 = head1;
    Node current2 = head2;
    int length1 = 0, length2 = 0;
    while (current1 != null) {
        length1++;
        current1 = current1.next;
    }
    while (current2 != null) {
        length2++;
        current2 = current2.next;
    }
    current1 = head1;
    current2 = head2;
    int diff = Math.abs(length1 - length2);
    if (length1 > length2) {
        for (int i = 0; i < diff; i++) {
            current1 = current1.next;
        }
    } else {
        for (int i = 0; i < diff; i++) {
            current2 = current2.next;
        }
    }

    while (current1 != null && current2 != null) {
        if (current1 == current2) {
            return current1.data;
        }
        current1 = current1.next;
        current2 = current2.next;
    }

    return null;
}
public void printList() {
    Node temp = head;
    System.out.print("[");
    while (temp != null) {
        System.out.print(temp.data);
        temp = temp.next;
        if (temp != null) {
            System.out.print(" → ");
        }
    }
    System.out.println("]");
}
   public static void main(String[] args) {
         LinkedListIntersection list1 = new LinkedListIntersection();
    list1.insert(1);
    list1.insert(2);
```

```
        list1.insert(3);
        list1.insert(4);
        list1.insert(5);

        LinkedListIntersection list2 = new LinkedListIntersection();
        list2.insert(6);
        list2.insert(7);
        list2.head.next.next = list1.head.next.next;

        System.out.print("Input: List1 = ");
        list1.printList();
        System.out.print("List2 = ");
        list2.printList();

        Integer intersectionPoint1 = getIntersectionPoint(list1.head, list2.head);
        if (intersectionPoint1 != null) {
            System.out.println("Output: Intersection Point = " + intersectionPoint1);
        } else {
            System.out.println("Output: No Intersection");
        }
        LinkedListIntersection list3 = new LinkedListIntersection();
        list3.insert(10);
        list3.insert(20);
        list3.insert(30);
        list3.insert(40);

        LinkedListIntersection list4 = new LinkedListIntersection();
        list4.insert(15);
        list4.insert(25);
        list4.insert(35);

        System.out.print("Input: List1 = ");
        list3.printList();
        System.out.print("List2 = ");
        list4.printList();

        Integer intersectionPoint2 = getIntersectionPoint(list3.head, list4.head);
        if (intersectionPoint2 != null) {
            System.out.println("Output: Intersection Point = " + intersectionPoint2);
        } else {
            System.out.println("Output: No Intersection");
        }
    }
}
```

Output –



15. Find the middle element of a linked list in one pass.

- **Test Case 1**:
  Input: List = [1, 2, 3, 4, 5]
  Output: Middle = 3
- **Test Case 2**:
  Input: List = [11, 22, 33, 44, 55, 66]
  Output: Middle = 44

Code –

```
package ads;

public class FindMiddleElement {
	private static class Node {
	  int data;
	  Node next;

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

    private Node head;
    public 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;
        }
    }
    public Integer findMiddle() {
        if (head == null) {
            return null;
        }

        Node slow = head;
        Node fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        return slow.data;
    }
    public void printList() {
        Node temp = head;
        System.out.print("[");
        while (temp != null) {
            System.out.print(temp.data);
```

```
            temp = temp.next;
            if (temp != null) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
    public static void main(String[] args) {
          FindMiddleElement list1 = new FindMiddleElement();
        list1.insert(1);
        list1.insert(2);
        list1.insert(3);
        list1.insert(4);
        list1.insert(5);

        System.out.print("Input: List = ");
        list1.printList();

        Integer middle1 = list1.findMiddle();
        System.out.println("Output: Middle = " + middle1);

        FindMiddleElement list2 = new FindMiddleElement();
        list2.insert(11);
        list2.insert(22);
        list2.insert(33);
        list2.insert(44);
        list2.insert(55);
        list2.insert(66);

        System.out.print("Input: List = ");
        list2.printList();

        Integer middle2 = list2.findMiddle();
        System.out.println("Output: Middle = " + middle2);
    }
}
```

Output –