Assignment 4

1. Singly Linked List:

Java

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

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

class LinkedList {
    Node head;

    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;
        }
    }

    void delete(int    data) {
        if (head == null) {
            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;
        }
    }

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

2. Reverse Singly Linked List:

Java

```
Node reverseList(Node head) {
    Node prev = null;
    Node curr = head;
    Node next;
    while (curr != null) {
        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```

3. Detect Cycle in Linked List:

Java

```
boolean hasCycle(Node head) {
    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;
}
```

4. Merge Two Sorted Linked Lists:

Java

```
Node mergeTwoLists(Node l1, Node l2) {
    Node dummy = new Node(0);
    Node current = dummy;
    while (l1 != null && l2 != null) {
        if (l1.data <= l2.data) {
            current.next = l1;
            l1 = l1.next;
        } else {
            current.next = l2;
            l2 = l2.next;
        }
        current = current.next;
    }
    if (l1 != null) {
        current.next = l1;
    }
    if (l2 != null) {
        current.next = l2;
    }
    return dummy.next;
}
```

5. Find nth Node from End of Linked List:

Java

```
Node nthFromEnd(Node head, int n) {
    Node slow = head;
    Node fast = head;
    for (int i = 0; i < n; i++) {
        if (fast == null) {
            return null;
        }
        fast = fast.next;
    }
    while (fast != null) {
        slow = slow.next;
        fast = fast.next;
    }
    return slow;
}
```

6. Remove Duplicates from Sorted Linked List:

Java

```
Node removeDuplicates(Node head) {
    if (head == null) {
        return null;
    }
    Node current = head;
    while (current.next != null) {
        if (current.data == current.next.data) {
            current.next = current.next.next;
        } else    {
            current = current.next;
        }
    }
    return head;
}
```

7. Doubly Linked List:

Java

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

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

```
class DoublyLinkedList {
    Node head;

    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;
        }
    }

    void    delete(int data) {
        if (head == null) {
            return;
        }
        if (head.data == data) {
            head = head.next;
            if (head != null) {
                head.prev = null;
            }
            return;
        }
        Node temp = head;
        while (temp != null && temp.data != data) {
            temp = temp.next;
        }
        if (temp != null) {
            temp.prev.next = temp.next;
            if (temp.next != null) {
                temp.next.prev = temp.prev;
            }
        }
    }

    void traverse() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
    }
}
```

8. Reverse Doubly Linked List:

Java

```
Node reverseList(Node head) {
    if (head == null || head.next == null) {
        return head;
    }
    Node temp = head;
    while (temp != null) {
        Node prev = temp.prev;
        temp.prev = temp.next;
        temp.next = prev;
        temp = temp.prev;
    }
    return head.prev;
}
```

9. Add Two Numbers Represented by Linked Lists:

Java

```
Node addTwoNumbers(Node l1, Node l2) {
    Node dummy = new Node(0);
    Node current = dummy;
    int carry = 0;
    while (l1 != null || l2 != null) {
        int x = (l1 != null) ? l1.data : 0;
        int y = (l2 != null) ? l2.data : 0;
        int sum = x + y + carry;
        carry = sum / 10;
        current.next = new    Node(sum % 10);
        current = current.next;
        if (l1 != null) {
            l1 = l1.next;
        }
        if (l2 != null) {
            l2 = l2.next;
```

```
        }
    }
    if (carry > 0) {
        current.next = new Node(carry);
    }
    return dummy.next;
}
```

10. Rotate a Linked List by k Places:

Java

```
Node rotateList(Node head, int k) {
    if (head == null || head.next == null || k == 0) {
        return head;
    }
    int length = 0;
    Node temp = head;
    while (temp != null) {
        length++;
        temp = temp.next;
    }
    k %= length;
    if (k == 0) {
        return head;
    }
    Node fast = head;
    Node slow = head;
    for (int i = 0; i < k; i++) {
        fast = fast.next;
    }
    while (fast.next != null) {
        slow = slow.next;
        fast = fast.next;
    }
    fast.next = head;
    head = slow.next;
    slow.next = null;
    return head;
}
```

11. Flatten a Multilevel Doubly Linked List:

Java

```
Node flatten(Node head) {
    if (head == null) {
        return head;
    }
    Node curr = head;
    while (curr != null) {
        if (curr.child != null) {
            Node next = curr.next;
            curr.next    = curr.child;
            curr.child.prev = curr;
            curr.child = null;
            Node tail = flatten(curr.next);
            tail.next = next;
            if (next != null) {
                next.prev = tail;
            }
        }
        curr = curr.next;
    }
    return head;
}
```

12. Split a Circular Linked List into Two Halves:

Java

```
Node[] splitList(Node head) {
    Node slow = head;
    Node fast = head;
    while (fast.next != head && fast.next.next != head) {
        slow = slow.next;
        fast = fast.next.next;
    }
    Node    head2 = slow.next;
    slow.next = head;
    return new Node[]{head, head2};
}
```

13. Insert a Node in a Sorted Circular Linked List:

Java

```
Node insertInSortedCircularList(Node head, int data) {
    Node newNode = new Node(data);
    if (head == null) {
        newNode.next = newNode;
        return newNode;
    }
    Node temp = head;
    while (temp.next != head && temp.next.data    < data) {
        temp = temp.next;
    }
    newNode.next = temp.next;
    temp.next = newNode;
    return head;
}
```

14. Check if Two Linked Lists Intersect:

Java

```
Node findIntersection(Node headA, Node headB) {
    if (headA == null || headB == null) {
        return null;
    }
    Node A = headA;
    Node B = headB;
    while (A != B) {
        A = A == null ? headB : A.next;
        B = B == null ? headA : B.next;
    }
    return    A;
}
```

15. Find the Middle Element of a Linked List in One Pass:

Java

```
Node findMiddle(Node head) {
    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;
}
```