Activity 1 – Lists, ArrayList vs LinkedList, Singly Linked List

Part 1 – Concept Analysis

Feature	ArrayList	LinkedList
Internal Structure	Dynamic Array (contiguous memory)	Doubly Linked List (nodes with pointers)
Access Time Complexity	O(1)	O(n)
Insertion Time Complexity	O(n)	O(1) (if node reference known)
Deletion Time Complexity	O(n)	O(1) (if node reference known)
Memory Usage	Lower (Less overhead)	Higher (Data + pointers)

Data Structure Selection:

• Student record system: ArrayList (frequent index access/search).

• Browser history: LinkedList or Stack (sequential navigation).

• Online shopping cart: LinkedList (frequent add/remove).

• Undo/Redo feature: Stack (often implemented using LinkedList).

Part 2 – Coding Task

Task A – ArrayList Program (Student Marks)

```
import java.util.ArrayList;
import java.util.Collections;

public class StudentMarks {
    public static void main(String[] args) {
        ArrayList<Integer> marks = new ArrayList<>();

        marks.add(85);
        marks.add(92);
        marks.add(78);
        marks.add(65);
        marks.add(88);

        marks.add(2, 95);

        int minMark = Collections.min(marks);
        marks.remove(Integer.valueOf(minMark));

        System.out.println("Final List: " + marks);
    }
}
```

Task B – LinkedList as Queue (Ticket Booking)

```
import java.util.LinkedList;
import java.util.Queue;

public class TicketQueue {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        queue.add("Customer 1");
        queue.add("Customer 2");
        queue.add("Customer 3");
        queue.add("Customer 4");
        queue.add("Customer 5");

        queue.poll();
        queue.poll();

        System.out.println("Remaining Queue: " + queue);
    }
}
```

Part 3 – Singly Linked List Implementation

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

public class SinglyLinkedList {
    Node head;

    public void insertAtBeginning(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }

    public void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) {
            head = newNode;
            return;
        }
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = newNode;
    }

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

Part 5 – Viva Questions

- Why is ArrayList access faster? Uses contiguous memory enabling O(1) index calculation.
- Why does LinkedList consume more memory? Stores additional pointer references per node.
- Insertion at beginning in SLL: O(1).
- Singly vs Doubly: Doubly stores an additional previous pointer.
- RandomAccess interface: Marker interface indicating fast constant-time access.