Activity 2 – Advanced Problem Solving (Lists)

Part 1 – Scenario Based Analysis

• Real-time chat: LinkedList (or ArrayList if appending). O(1) insertion at end. LinkedList better if deleting from top frequently.

• Music playlist (reordered): LinkedList. O(1) pointer changes vs O(n) shifting in arrays.

• Student database (search by index): ArrayList. O(1) access time.

• Browser Back/Forward: Doubly LinkedList or Stacks. O(1) prev/next movement.

• Online exam (sequential answers): ArrayList. Efficient storage and fast access.

Part 2 – Coding Challenges

Task A – ArrayList Advanced

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

public class ArrayListAdvanced {
    public static void main(String[] args) {
        ArrayList<Integer> numbers = new ArrayList<>();
        int[] input = {12, 5, 8, 20, 33, 7, 4, 15, 9, 30};
        for (int i : input) numbers.add(i);

        numbers.removeIf(n -> n % 2 == 0);

        if (!numbers.isEmpty()) {
            System.out.println("Max: " + Collections.max(numbers));
            System.out.println("Min: " + Collections.min(numbers));
        }

        numbers.sort(Collections.reverseOrder());
        System.out.println("Sorted Descending: " + numbers);
    }
}
```

Task B – LinkedList Implementation (Hospital Queue)

```
import java.util.LinkedList;

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

        for (int i = 1; i <= 5; i++) queue.add("Patient " + i);

        queue.addFirst("Emergency Patient");

        queue.removeFirst();
        queue.removeFirst();

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

```
}
```

Part 3 – Singly Linked List

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

public class SLL {
    Node head;

    public void insertAtPos(int data, int pos) {
        Node newNode = new Node(data);
        if (pos == 0) {
            newNode.next = head;
            head = newNode;
            return;
        }
        Node temp = head;
        for (int i = 0; i < pos - 1 && temp != null; i++) {
            temp = temp.next;
        }
        if (temp != null) {
            newNode.next = temp.next;
            temp.next = newNode;
        }
    }

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

    public int countNodes() {
        int count = 0;
        Node temp = head;
        while (temp != null) { count++; temp = temp.next; }
        return count;
    }

    public int findMiddle() {
        if (head == null) return -1;
        Node slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow.data;
    }

    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

• Amortized Time Complexity: Average time per operation over a sequence (ArrayList add is O(1) amortized).

• Why ArrayList resizing is expensive? Requires creating new array and copying elements (O(n)).

• How LinkedList stores elements? As Node objects containing data and next reference.

• Singly vs Doubly: Singly (next only, less memory). Doubly (next + prev, more memory).

• Space complexity of LinkedList: O(n), storing data and pointer references.