Exercise 5: Task Management System

1.Understand Linked Lists:

- Singly Linked List: Each node points to the next node. Suitable for forward traversal.

- Doubly Linked List: Each node points to both the next and previous nodes. Allows forward and backward traversal.

- Linked lists are dynamic and efficient for frequent insertions and deletions.

2. Setup:

Creating a class Task with attributes like taskId, taskName, and status.

```
public class Task {
    int taskId;
    String taskName;
    String status;

    public Task(int taskId, String taskName, String status) {
        this.taskId = taskId;
        this.taskName = taskName;
        this.status = status;
    }

    @Override
    public String toString() {
        return "Task[ID=" + taskId + ", Name=" + taskName + ", Status=" + status + "]";
    }
}
```

3. Implementation:

Implement a singly linked list to manage tasks and include methods to add, search, traverse, and delete tasks.

```
class Node {
    Task task;
    Node next;

    public Node(Task task) {
        this.task = task;
        this.next = null;
    }
}

public class TaskManager {
    private Node head;

    public void addTask(Task task) {
        Node newNode = new Node(task);
        if (head == null) {
            head = newNode;
        } else {
            Node temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newNode;
        }
    }



    public Task searchTask(int taskId) {
        Node current = head;
        while (current != null) {
            if (current.task.taskId == taskId) {
                return current.task;
            }
            current = current.next;
        }
        return null;
    }

    public void traverseTasks() {
        Node current = head;
        while (current != null) {
            System.out.println(current.task);
            current = current.next;
        }
    }

    public void deleteTask(int taskId) {
        if (head == null) return;

        if (head.task.taskId == taskId) {
            head = head.next;
            return;
        }

        Node current = head;
        while (current.next != null && current.next.task.taskId != taskId) {
            current = current.next;
        }

        if (current.next != null) {
            current.next = current.next.next;
        }
    }
}
```

4. Analysis:

- Add: O(n) to insert at the end.

- Search: O(n) as it traverses nodes linearly.

- Traverse: O(n) to visit each node.

- Delete: O(n) as it may need to traverse to find the node.

Advantages of Linked Lists over Arrays:

- Dynamic size: No need to define capacity ahead of time.

- Efficient insertion/deletion without shifting elements.

- Ideal for applications with unpredictable data growth or frequent updates.