**Exercise 5: Task Management System**

* + **Explain the different types of linked lists (Singly Linked List, Doubly Linked List).**

Types of Linked Lists

1. Singly Linked List

A Singly Linked List is a type of linked list where each node contains data and a reference (or pointer) to the next node in the sequence. The last node points to null, indicating the end of the list.

Structure:

* Node: Contains data and a reference to the next node.
* Head: The first node in the list.

Advantages:

* Simple Implementation: Easier to implement compared to other types of linked lists.
* Memory Efficient: Requires less memory per node since it only stores one reference.

Disadvantages:

* Unidirectional Traversal: Can only be traversed in one direction (from head to end).
* No Backward Traversal: Cannot move backward in the list without additional data structures.

**2. Doubly Linked List**

A **Doubly Linked List** is a type of linked list where each node contains data, a reference to the next node, and a reference to the previous node. This allows traversal in both directions.

**Structure**:

* **Node**: Contains data, a reference to the next node, and a reference to the previous node.
* **Head**: The first node in the list.
* **Tail**: The last node in the list.

**Advantages**:

* **Bidirectional Traversal**: Can be traversed in both directions (forward and backward).
* **Easier Deletion**: Deleting a node is easier since each node has a reference to its previous node.

**Disadvantages**:

* **More Memory**: Requires more memory per node due to the additional reference.
* **Complex Implementation**: Slightly more complex to implement compared to singly linked lists.

**Circular Linked List**

A **Circular Linked List** is a variation of the linked list where the last node points back to the first node, forming a circular structure. This can be implemented for both singly and doubly linked lists.

**Types of Circular Linked Lists:**

1. **Singly Circular Linked List**: Each node has a reference to the next node, and the last node points back to the first node.
2. **Doubly Circular Linked List**: Each node has references to both the next and previous nodes, and the last node’s next reference points to the first node, while the first node’s previous reference points to the last node.

**Singly Circular Linked List**

**Structure**:

* **Node**: Contains data and a reference to the next node.
* **Head**: The first node in the list.
* **Tail**: The last node in the list, which points back to the head.

**Advantages**:

* **Circular Traversal**: Can traverse the list starting from any node and eventually return to the starting node.
* **Efficient for Circular Buffers**: Useful in applications where circular traversal is needed, such as in circular buffers.

**Disadvantages**:

* **Complexity**: Slightly more complex to implement and manage compared to a simple singly linked list.
* **No Null Termination**: The end of the list is not marked by null, which can complicate certain operations.

**Doubly Circular Linked List**

**Structure**:

* **Node**: Contains data, a reference to the next node, and a reference to the previous node.
* **Head**: The first node in the list.
* **Tail**: The last node in the list, which points back to the head, and the head’s previous reference points to the tail.

**Advantages**:

* **Bidirectional Circular Traversal**: Can traverse the list in both directions and return to the starting node.
* **Efficient for Complex Operations**: Useful in applications requiring complex operations like bidirectional traversal and circular traversal.

**Disadvantages**:

* **More Memory**: Requires more memory per node due to the additional reference.
* **Complex Implementation**: More complex to implement and manage compared to a singly circular linked list.
  + **Analyze the time complexity of each operation.**
  + **Discuss the advantages of linked lists over arrays for dynamic data.**

**Time Complexity Analysis:**

* **addTask(Task task):** O(n), where n is the number of tasks in the linked list. This is because we need to traverse the entire list to add a new task at the end.
* **searchTask(int taskId): O(n),** where n is the number of tasks in the linked list. This is because we need to traverse the entire list to find a task with a specific ID.
* **traverse(): O(n),** where n is the number of tasks in the linked list. This is because we need to traverse the entire list to print all tasks.
* **deleteTask(int taskId): O(n),** where n is the number of tasks in the linked list. This is because we need to traverse the entire list to find and delete a task with a specific ID.

**Advantages of Linked Lists over Arrays for Dynamic Data:**

1. **Efficient Insertion and Deletion:**Linked lists allow for efficient insertion and deletion of elements at any position, whereas arrays require shifting elements, which can be time-consuming for large datasets.
2. **Dynamic Size:**Linked lists can grow or shrink dynamically as elements are added or removed, whereas arrays have a fixed size that needs to.