**Task Management System**Types of Linked Lists

* Singly Linked List: Each node points to the next node in the sequence. It allows traversal in one direction (forward). Each node contains:
  + Data: The value or data of the node.
  + Next: A reference to the next node.

Advantages:

* + Efficient insertions and deletions as they do not require shifting elements like arrays.

Disadvantages:

* + Limited to single-direction traversal.
  + Requires additional memory for storing the reference (pointer) to the next node.
* Doubly Linked List: Each node points to both the next node and the previous node. It allows traversal in both directions (forward and backward).

Advantages:

* + Bidirectional traversal.
  + Easier to delete nodes since you have a reference to the previous node.

Disadvantages:

* + Requires more memory for storing an additional reference (pointer) to the previous node.
  + Slightly more complex operations compared to singly linked lists.

**Implementation**

1. **Add Task**: Adds a new task to the end of the list. Time Complexity: O(n) due to traversal.
2. **Search Task**: Searches for a task by ID. Time Complexity: O(n) due to traversal.
3. **Traverse Tasks**: Prints all tasks in the list. Time Complexity: O(n) due to traversal.
4. **Delete Task**: Deletes a task by ID, handling edge cases (e.g., deletion at head). Time Complexity: O(n) due to traversal and possible element shifting.

* **Add Operation**: Time complexity is O(n) due to traversal to find the end of the list. Efficient for dynamic data as no need to resize.
* **Search Operation**: Time complexity is O(n) due to traversal. Searching in a singly linked list is linear.
* **Traverse Operation**: Time complexity is O(n) as it involves iterating through all nodes.
* **Delete Operation**: Time complexity is O(n) due to traversal to find the node. However, deletion is efficient as it does not require shifting elements.

**Advantages of Linked Lists Over Arrays**

* **Dynamic Size**: Linked lists can easily grow and shrink in size without the need for resizing or reallocating memory.
* **Efficient Insertions/Deletions**: Inserting or deleting nodes does not require shifting elements as in arrays, making these operations more efficient.

**Limitations of Linked Lists**

* **Memory Overhead**: Each node requires additional memory for storing the pointer/reference to the next node.
* **No Direct Access**: Accessing elements requires traversal from the head node, making random access less efficient compared to arrays.