**Exercise 5: Task Management System**

**Step 1: Understand Linked Lists**

**Singly Linked List**

* **Description:** A data structure where each node contains a data part and a reference (or pointer) to the next node in the sequence.
* **Advantages:**
  + **Dynamic Size:** Can grow and shrink dynamically without the need for reallocating or reorganizing the entire data structure.
  + **Efficient Insertions/Deletions:** Adding or removing elements at the beginning of the list or in the middle can be done in constant time (O(1)) if the position is known.
* **Disadvantages:**
  + **No Direct Access:** Accessing elements is linear in time complexity (O(n)).
  + **More Memory:** Requires extra memory for storing references.

**Doubly Linked List**

* **Description:** Similar to singly linked lists but each node contains an additional reference to the previous node.
* **Advantages:**
  + **Bidirectional Traversal:** Can be traversed in both forward and backward directions.
  + **Efficient Deletions:** Can delete nodes efficiently as you can access previous nodes directly.
* **Disadvantages:**
  + **More Memory:** Requires more memory for storing an extra reference to the previous node.
  + **Complex Operations:** Operations like insertion and deletion are more complex due to handling of two references.

**Time Complexity of Operations**

* **Add Task:**
  + **Best-case:** O(1) - Adding a task to an empty list.
  + **Worst-case:** O(n) - Adding a task to the end of the list.
* **Search Task:**
  + **Best-case:** O(1) - Finding the task at the beginning of the list.
  + **Average-case:** O(n) - On average, the task will be somewhere in the middle.
  + **Worst-case:** O(n) - The task is at the end of the list or not present.
* **Traverse Tasks:**
  + **Time complexity:** O(n) - All tasks need to be visited.
* **Delete Task:**
  + **Best-case:** O(1) - Deleting the first task.
  + **Average-case:** O(n) - On average, shifting half the elements.
  + **Worst-case:** O(n) - Deleting the last task.

**Advantages of Linked Lists Over Arrays for Dynamic Data**

* **Dynamic Size:** Linked lists can grow and shrink dynamically without the need for reallocating or reorganizing the entire data structure.
* **Efficient Insertions/Deletions:** Adding or removing elements in linked lists is generally more efficient, especially for operations that involve the beginning or middle of the list, due to the constant time complexity (O(1)) if the position is known.
* **Memory Utilization:** Linked lists use memory proportionally to their size, unlike arrays, which may allocate more memory than necessary.

**Limitations of Linked Lists**

* **No Direct Access:** Accessing elements by index is not possible in constant time; it requires linear time complexity (O(n)).
* **More Memory:** Requires extra memory for storing references/pointers to the next (and previous, in doubly linked lists) nodes.
* **Overhead:** Linked lists have a larger memory overhead compared to arrays due to storing additional references.