**Exercise 5: Task Management System**

**1. Understanding Linked Lists**

**Singly Linked List**: A singly linked list is a data structure that consists of nodes. Each node contains a data part and a reference (or link) to the next node in the sequence. The first node is called the head, and the last node points to null, indicating the end of the list.

**Doubly Linked List**: A doubly linked list is similar to a singly linked list, but each node contains two references: one to the next node and one to the previous node. This allows for traversal in both directions, forward and backward, making certain operations, such as deletion or insertion before a given node, more efficient.

**4.Time Complexity**

**Time Complexity**:

* **Addition of a Task**: Adding a task to the end of a singly linked list takes O(n) time, where n is the number of nodes in the list. This is because we need to traverse the entire list to find the last node.
* **Searching for a Task**: Searching for a task by its ID involves traversing the list, which takes O(n) time in the worst case.
* **Traversal of Tasks**: Traversing all tasks to display them takes O(n) time, as each node needs to be visited.
* **Deletion of a Task**: Deleting a task requires finding the task first, which takes O(n) time. Once found, the deletion itself is O(1), making the overall complexity O(n).

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

* **Dynamic Size**: Linked lists can increase and decrease in size by adding or removing nodes. This is in contrast to arrays, which have a fixed size or require memory consuming resizing operations.
* **Ease of Insertion/Deletion**: Inserting or deleting elements in a linked list is generally more efficient than in an array, especially when dealing with large datasets. In a linked list, these operations only require changing the references (pointers), while in an array, elements may need to be shifted.
* **Memory Utilization**: Linked lists use memory more efficiently when dealing with a large number of insertions and deletions. Arrays may allocate more memory than needed, leading to wasted space, or may require resizing, leading to overhead.