**5\_Task Management System**

**a. Understanding Linked Lists**

* A **Singly Linked List** is a simple chain of nodes where each node holds data and a reference to the next node. It’s lightweight and easy to implement. Since it only points forward, you can only move in one direction through the list. Adding or removing items at the front is super-fast (O(1)), but doing so elsewhere takes more time (O(n)).
* A **Doubly Linked List**, on the other hand, steps it up, each node points not just to the next node, but also to the previous one. This makes going backward through the list just as easy as going forward. It uses a bit more memory because of the extra pointer. Insertion and deletion are still fast if you already have the node pointer, otherwise, it's still O(n) to find the spot.

**b. Time Complexity**

|  |  |
| --- | --- |
| **Operation** | **Complexity** |
| Add Task (end) | O(n) |
| Search Task | O(n) |
| Delete Task | O(n) |
| Traverse | O(n) |

**c. Why Use Linked Lists Instead of Arrays?**

Imagine you’re building a task manager where tasks keep getting added and removed randomly.

* An **Array** is like a fixed-size rack, if we want to add more, we often need to build a bigger rack and shift everything over. Inserting or deleting items in the middle? We’ll need to shuffle elements around. But the big plus is that we can jump to any element instantly using an index (O(1)).
* A **Linked List**, in contrast, is flexible like a dynamic train, we can easily add or remove carriages (nodes) from the front or back without affecting others. It doesn’t need to be resized, and memory is allocated as needed. While we can't directly jump to a node like in an array (you have to travel node by node), it’s ideal for cases like task queues where we're mostly dealing with the ends.