**TASK MANAGEMENT SYSTEM**

A Singly Linked List is a type of linked list where each node only contains a reference (or "link") to the next node in the list. Each node typically consists of two parts: the data and the reference to the next node. This means that traversal can only occur in one direction, from the beginning of the list to the end. Insertion and deletion of nodes can be efficient, but finding a specific node or traversing the list in reverse order can be more challenging.

Doubly Linked List differs from a regular linked list. In this structure, each node has links to both the next and previous nodes in the sequence. This design lets you move through the list in both directions. It makes finding specific nodes or adding/removing nodes at any spot easier. But it has drawbacks. It uses more memory and is a bit harder to set up. People often use Linked Lists when they need to add and remove nodes a lot.

Time Complexity:

**Add Task**: O(n) - To add a new task would lead to traversing to the end of the list.

**Search Task**: O(n) - To find a task, one has to traverse the list.

**Delete Task**: O(n) - First, it needs to find the task; then the task will have to be removed.

**Traverse Tasks**: O(n) - It requires iterating over the list to traverse all tasks.

Advantages of Linked Lists Over Arrays for Dynamic Data:

Dynamic Size: A linked list's size may go on increasing or decreasing during execution, but an array has a fixed size.

Efficient Insertions/Deletions: Insertion and deletion in a linked list are more efficient when compared to arrays, particularly at the start or middle of the list. This will need shifting the elements in arrays, which is time-consuming.

Memory Consumption: In the case of a linked list, memory is consumed only for the actual elements available, whereas in an array, a fixed memory is consumed irrespective of whether the array is full or not. Additional Overhead in Linked Lists Due to Pointers.