**Task Management System**

**Singly Linked List:**

Each node contains data and a reference (or link) to the next node.

- **Traversal:** Can only be traversed in one direction (from head to tail).

**- Insertion/Deletion:** Easier to implement insertions and deletions at the beginning or end but requires traversal to modify nodes in the middle.

- **Memory Usage:** Uses less memory per node compared to doubly linked lists, as it only stores one reference.

**Doubly Linked List:**

Each node contains data, a reference to the next node, and a reference to the previous node.

**- Traversal:** Can be traversed in both directions (from head to tail and tail to head).

**- Insertion/Deletion:** Easier to insert or delete nodes from both ends and the middle due to bidirectional references.

- **Memory Usage:** Uses more memory per node due to the extra reference for the previous node.

**Time Complexity Analysis:**

1. **Add Task**: O(n) - Due to traversal to the end of the list
2. **Search Task**: O(n) - Requires traversing the list to find the task.
3. **Traverse Tasks**: O(n)- Linear traversal of all elements.
4. **Delete Task**: O(n) - Requires traversal to find the task and then updating the list.

**Advantages of Linked Lists over Arrays:**

**Dynamic Size:**

Advantage: Linked lists can expand and shrink dynamically, avoiding the need to predefine size or handle resizing manually.

**Memory Utilization:**

Advantage: Memory is allocated as needed, avoiding wasted space unlike arrays where space must be set up in advance.

**Flexibility:**

Advantage: Linked lists do not require contiguous memory allocation, which can be beneficial in fragmented memory environments.

**Ease of Expansion:**

Advantage: Linked lists can efficiently handle changing data sizes without the need for reallocating or copying elements, as in arrays.