5. Task Management System

Types of Linked Lists:

Linked Lists are a dynamic data structure consisting of a sequence of nodes, where each node contains data and a reference (or pointer) to the next node in the sequence. They allow for efficient insertion and deletion operations. There are several types of linked lists:

1. Singly Linked List

- **Description**: A singly linked list consists of nodes where each node contains data and a pointer to the next node. The list starts with a head node and ends with a null reference.
- **Structure**: Each node contains:
  - `data`: The value stored in the node.
  - `next`: A reference to the next node in the list.
- **Traversal**: Traversing a singly linked list is done in one direction, from the head to the tail.
- **Operations**:
  - Insertion can be done at the beginning, middle, or end.
  - Deletion can also be performed at various positions.

2. Doubly Linked List

- **Description**: A doubly linked list consists of nodes that contain data, a reference to the next node, and a reference to the previous node. This allows traversal in both directions.
- **Structure**: Each node contains:
  - `data`: The value stored in the node.
  - `next`: A reference to the next node in the list.
  - `prev`: A reference to the previous node in the list.
- **Traversal**: Allows traversing in both forward and backward directions.

Analysis

Add Operation:

- **At the Beginning**: O(1) - A new node can be added by simply updating the head pointer.
- **At the End**: O(n) for singly linked lists (need to traverse to the end); O(1) for doubly linked lists if a tail pointer is maintained.

- **In the Middle**: O(n) for both singly and doubly linked lists due to the need to traverse to the insertion point.

Search Operation:

- **Time Complexity**: O(n) - Requires traversing the list from the head (or tail in doubly linked lists) to find the target value.

Traverse Operation:

- **Time Complexity**: O(n) - Each node needs to be visited once to traverse the entire list.

Delete Operation:

- **At the Beginning**: O(1) - Update the head pointer to remove the first node.
- **At the End**: O(n) for singly linked lists (need to traverse to find the previous node); O(1) for doubly linked lists if a tail pointer is maintained.
- **In the Middle**: O(n) - Need to traverse to find the node to be deleted, and then update pointers.

Advantages of Linked Lists Over Arrays for Dynamic Data

1. **Dynamic Size**:

   Linked lists can grow and shrink dynamically, allowing for efficient use of memory when the number of elements is not known in advance or changes frequently.
2. **Efficient Insertions and Deletions**:

   Linked lists allow for O(1) time complexity for insertions and deletions at the beginning, and potentially at the end (if a tail pointer is maintained), compared to O(n) for shifting elements in an array.
3. **Memory Utilization**:

Linked lists do not require contiguous memory allocation like arrays, which can lead to better memory utilization and can help avoid fragmentation issues.