**Task Management System**

**Exploring Linked Lists**

**Types of Linked Lists**

1. **Singly Linked List**: This type of linked list consists of nodes where each node contains data and a reference to the next node in the sequence. While traversal is straightforward, accessing previous nodes is inefficient since there are no backward links.
2. **Doubly Linked List**: Each node in a doubly linked list holds data, a reference to the next node, and a reference to the previous node. This structure allows for efficient traversal in both directions and simplifies node deletion, though it uses more memory due to the extra references.

**Performance Analysis**

**Time Complexity for Linked List Operations:**

1. **Adding a Task**:
   * **Time Complexity**: O(n). To add a task, you typically need to traverse the list to find the appropriate insertion point.
2. **Searching for a Task**:
   * **Time Complexity**: O(n). Searching involves traversing the list until the desired task is located.
3. **Traversing the List**:
   * **Time Complexity**: O(n). Each node must be visited once during traversal, making this operation linear in time complexity.
4. **Deleting a Task**:
   * **Time Complexity**: O(n). Finding the node to delete requires traversal, and removing it involves updating pointers to maintain list integrity.

**Advantages of Linked Lists Over Arrays**

1. **Dynamic Size**: Linked lists can adjust their size dynamically, making them more suitable for situations where the number of elements changes frequently.
2. **Efficient Insertions and Deletions**: Linked lists allow for more efficient insertions and deletions, especially at the beginning or in the middle of the list, since these operations only involve updating pointers rather than shifting elements.
3. **Memory Usage**: Linked lists allocate memory for nodes as needed, avoiding the potential waste of memory that can occur with arrays if they are not fully utilized.

**Disadvantages of Linked Lists**

1. **Access Time**: Accessing elements in a linked list is slower compared to arrays. Since nodes are not contiguous in memory, it takes O(n) time to find a specific element.
2. **Memory Overhead**: Each node in a linked list requires additional memory for storing references to other nodes, which can add up, especially in large lists.
3. **Cache Locality**: Arrays benefit from better cache performance due to their contiguous memory layout, whereas linked lists suffer from poorer cache locality because their elements are scattered across memory.

**When to Use Linked Lists**

Linked lists are particularly advantageous in scenarios where the size of the data collection is dynamic and where frequent insertions or deletions are required. For static-sized collections with a need for frequent access to elements, arrays might be more appropriate due to their faster access times and better cache performance.