**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

1. **Understand Linked Lists:**

**Types of Linked Lists**

1. **Singly Linked List**:
   * **Structure**: Each node contains data and a reference to the next node.
   * **Traversal**: Can be traversed in one direction (from head to tail).
   * **Operations**:
     + **Insert**: O(1) if inserting at the beginning, O(n) if inserting at the end (unless a tail reference is maintained).
     + **Delete**: O(1) if deleting the head, O(n) if searching for an element to delete.
     + **Search**: O(n) as you may need to traverse the entire list.
2. **Doubly Linked List**:
   * **Structure**: 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).
   * **Operations**:
     + **Insert**: O(1) if inserting at the beginning or end (with tail reference), O(n) if inserting at a specific position.
     + **Delete**: O(1) if deleting the head or tail (with tail reference), O(n) if searching for an element to delete.
     + **Search**: O(n) as you may need to traverse the entire list.

**4. Analysis:**

**Time Complexity Analysis**

* **Singly Linked List**:
  + **Insert**:
    - Beginning: O(1)
    - End: O(n) (or O(1) if a tail reference is maintained)
    - Middle: O(n)
  + **Delete**:
    - Beginning: O(1)
    - End: O(n) (or O(1) if a tail reference is maintained)
    - Middle: O(n)
  + **Search**: O(n)
* **Doubly Linked List**:
  + **Insert**:
    - Beginning: O(1)
    - End: O(1) (with tail reference)
    - Middle: O(n)
  + **Delete**:
    - Beginning: O(1)
    - End: O(1) (with tail reference)
    - Middle: O(n)
  + **Search**: O(n)

**Advantages of Linked Lists over Arrays for Dynamic Data**

1. **Dynamic Size**: Linked lists can easily grow or shrink in size by adding or removing nodes. Arrays have a fixed size, and resizing them can be expensive as it requires creating a new array and copying elements.
2. **Efficient Insertions/Deletions**: Insertions and deletions in linked lists are generally more efficient, especially when dealing with elements at the beginning or end of the list (O(1) operations). In arrays, these operations can be O(n) due to the need to shift elements.
3. **Memory Usage**: Linked lists use memory more efficiently for dynamic data because they allocate memory as needed. Arrays require a predefined size, leading to potential wastage of memory or the need for resizing operations.
4. **No Contiguous Memory Requirement**: Linked lists do not require contiguous memory blocks, which can be advantageous in systems with fragmented memory. Arrays require contiguous memory, which can be a limitation in some cases.