**Exercise 5: Task Management System**

**Scenario:**

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

**Steps:**

1. **Understand Linked Lists:**

* Explain the different types of linked lists (Singly Linked List, Doubly Linked List).

***Singly Linked List:***

Structure: Comprises nodes where each node holds a data element and a reference (or pointer) to the next node in the sequence.

Operations: Provides efficient insertions and deletions at the beginning of the list. Traversal is needed for operations involving the end or middle of the list.

Use Case: Ideal for implementing linear data structures like stacks and queues.

***Doubly Linked List:***

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

Operations: Supports bidirectional traversal, which makes insertions and deletions more efficient.

Use Case: Suitable for implementing more complex data structures such as deques and for scenarios requiring bidirectional traversal.

1. **Setup:**
   * Create a class **Task** with attributes like **taskId**, **taskName**, and **status**.
2. **Implementation:**
   * Implement a singly linked list to manage tasks.
   * Implement methods to **add**, **search**, **traverse**, and **delete** tasks in the linked list.
3. **Analysis:**

* Analyze the time complexity of each operation.

1. **Add Task:** O(n) for adding a task at the end because it requires traversing the list to reach the end.
2. **Search Task:** O(n) as it involves traversing the list to locate the desired task.
3. **Traverse Tasks:** O(n) since it requires visiting each node in the list.
4. **Delete Task:** O(n) because it may need traversal to find and remove the task.

* Discuss the advantages of linked lists over arrays for dynamic data.

1. Dynamic Size: Linked lists can dynamically grow or shrink in size, whereas arrays have a fixed size that cannot be altered after creation.
2. Efficient Insertions/Deletions: Inserting or deleting elements in the middle of a linked list is more efficient (constant time O(1) for operations at the head) compared to arrays, where such operations have a time complexity of O(n).