**Time Complexity Analysis**

1. **Add Task:**
   * **Time Complexity:** O(n) (linear time) in the worst case, where n is the number of tasks, due to the need to traverse the list to add the task at the end.
2. **Search Task by ID:**
   * **Time Complexity:** O(n) (linear time) because it involves scanning through the linked list.
3. **Traverse Tasks:**
   * **Time Complexity:** O(n) (linear time) for traversing all tasks in the linked list.
4. **Delete Task by ID:**
   * **Time Complexity:** O(n) (linear time) for finding the task and updating the list.

**Advantages of Linked Lists Over Arrays**

1. **Dynamic Size:**
   * Linked lists can grow or shrink dynamically as needed, unlike arrays which have a fixed size.
2. **Efficient Insertions/Deletions:**
   * Inserting or deleting elements is more efficient (O(1) time complexity) if the position is known, as it doesn't involve shifting elements like in arrays.
3. **Flexibility:**
   * Linked lists provide more flexibility in managing memory as they allocate memory for each element dynamically.

**When to Use Linked Lists:**

* When the number of elements is not known in advance or varies frequently.
* When frequent insertions and deletions are required.