4. Analysis:

Time Complexity:

- addTask(): O(n)

→ Must traverse to the end to add a new task.

- searchTask(): O(n)

→ Requires checking each node until a match is found or list ends.

- traverseTasks(): O(n)

→ Visits each node exactly once.

- deleteTask(): O(n)

→ Searches for the node to delete and then updates references.

Space Complexity:

- O(n) where n is the number of tasks stored.

- Each node requires additional space for the reference to the next node.

Advantages Over Arrays:

- Dynamic size: No need to know the number of tasks in advance.

- Efficient insertion/deletion at the head or tail (with tail reference).

- Better suited for systems where tasks are frequently added/removed.

Limitations:

- No random access: You must traverse the list to reach an element.

- Higher memory usage due to object references.

- Slower searches compared to arrays with direct indexing.

Real-World Application:

- Ideal for queues, real-time task processors, memory-efficient systems where dynamic allocation is preferred.

- Can be extended with doubly linked or circular structures for more advanced behaviors.