Exercise5

1. Understanding Linked Lists

Types of Linked Lists:

- **Singly Linked List:**
  - **Structure:** Each node contains data and a reference to the next node in the list. The last node points to `null`.
  - **Operations:** Allows efficient insertion and deletion of nodes, especially at the beginning or middle of the list. However, searching for an element requires traversing from the head node.
- **Doubly Linked List:**
  - **Structure:** Each node contains data, a reference to the next node, and a reference to the previous node. This allows bidirectional traversal.
  - **Operations:** More flexible for insertion and deletion operations compared to a singly linked list, as you can traverse the list in both directions.

4. Analysis

Time Complexity:

- **Add Task:**
  - **Best Case:** O(1) if adding at the head or if the list is empty.
  - **Worst Case:** O(n) as it requires traversal to find the end of the list.
- **Search Task:**
  - **Best Case:** O(1) if the task is found at the head.
  - **Worst Case:** O(n) if the task is at the end or not found.
- **Traverse Tasks:**
  - **Time Complexity:** O(n) as each node must be visited to display all tasks.
- **Delete Task:**
  - **Best Case:** O(1) if deleting the head.
  - **Worst Case:** O(n) if the task is at the end or if the list needs to be traversed to find the node.

Advantages of Linked Lists Over Arrays for Dynamic Data:

- **Dynamic Size:** Linked lists can grow or shrink dynamically without the need for resizing or reallocating memory.
- **Efficient Insertions/Deletions:** Adding or removing elements is more efficient compared to arrays, as it doesn’t require shifting elements. Insertion and deletion operations are O(1) when done at the beginning or known positions.

Limitations of Linked Lists:

- **Memory Overhead:** Each node requires extra memory for storing a reference to the next node.
- **Sequential Access:** Accessing elements requires sequential traversal, making random access less efficient compared to arrays.