Analysis

Linked Lists:

- **Singly Linked List**:
  - **Structure**: Each node contains a value and a reference to the next node. It allows traversal in one direction (from head to tail).
  - **Operations**: Efficient for insertions and deletions when the position is known, but accessing elements requires traversal from the head.

- **Doubly Linked List**:
  - **Structure**: Each node contains a value and two references: one to the next node and one to the previous node. It allows traversal in both directions (from head to tail and tail to head).
  - **Operations**: Provides more flexibility in traversal and easier deletions as it has references to both next and previous nodes.

Time Complexity:

- **Add Operation**: O(n) in the worst case if we need to traverse the entire list to find the end. O(1) if adding at the head.
- **Search Operation**: O(n) in the worst case because we may need to traverse the entire list.
- **Traverse Operation**: O(n) because it involves visiting every node in the list.
- **Delete Operation**: O(n) in the worst case if the task is near the end or not found. O(1) if deleting the head node.

Advantages of Linked Lists over Arrays:

- **Dynamic Size**: Linked lists can grow and shrink dynamically, unlike arrays which have a fixed size.
- **Efficient Insertions/Deletions**: Insertions and deletions are generally more efficient because they don't require shifting elements, as with arrays. Linked lists only require updating pointers.
- **Flexibility**: Linked lists provide more flexibility for frequent modifications, such as adding or removing elements.