**1. Understand Linked Lists**

**Types of Linked Lists**

**Singly Linked List**:

* **Description**: A singly linked list is a linear data structure where each element (node) points to the next node in the sequence. Each node contains two parts: the data and a reference to the next node.
* **Operations**: Allows for efficient insertions and deletions but only supports forward traversal.
* **Structure**:
  + **Node**: Contains data and a reference to the next node.
  + **Head**: Points to the first node in the list.

**Doubly Linked List**:

* **Description**: A doubly linked list is similar to a singly linked list, but each node contains two references: one to the next node and one to the previous node. This allows for bidirectional traversal.
* **Operations**: Allows efficient insertions and deletions, supports both forward and backward traversal.
* **Structure**:
  + **Node**: Contains data, a reference to the next node, and a reference to the previous node.
  + **Head**: Points to the first node.
  + **Tail**: Points to the last node.

**Comparison**:

* **Singly Linked List**: Simpler and uses less memory (one reference per node). Suitable for simpler applications where bidirectional traversal is not needed.
* **Doubly Linked List**: More complex (two references per node) but supports efficient bidirectional traversal and easier node deletion.

**4. Analysis**

**Time Complexity**

* **Add Operation**:
  + **Time Complexity**: O(n) in the worst case, where n is the number of tasks. This is because you may need to traverse the entire list to find the end for adding a new task.
* **Search Operation**:
  + **Time Complexity**: O(n), where n is the number of tasks. This is because you may need to traverse the entire list to find the task with the specified taskId.
* **Traverse Operation**:
  + **Time Complexity**: O(n), where n is the number of tasks. Each task is visited exactly once.
* **Delete Operation**:
  + **Time Complexity**: O(n) in the worst case, where n is the number of tasks. This is because you may need to traverse the list to find the task to delete and then perform the deletion.

**Advantages of Linked Lists Over Arrays for Dynamic Data**

* **Dynamic Size**: Linked lists can grow and shrink dynamically as needed, whereas arrays have a fixed size.
* **Efficient Insertions/Deletions**: Inserting or deleting tasks in a linked list is more efficient compared to arrays, where shifting elements is required.

**When to Use Linked Lists**:

* **When the number of elements is unknown** and changes frequently.
* **When insertions and deletions are frequent** and need to be efficient, especially when the tasks are not always added or removed from the end.

**Disadvantages**:

* **Extra Memory**: Each node requires extra memory for storing the reference to the next node.
* **Sequential Access**: Linked lists do not support direct access to elements by index. Traversal from the head is required to reach a specific node.

In summary, linked lists offer advantages for dynamic data management where frequent insertions and deletions occur. They are preferable to arrays in scenarios where the data size changes frequently and you need efficient management of elements.