**Step 1: Understand Linked Lists**

**Types of Linked Lists:**

1. **Singly Linked List:**
   * **Definition:** A singly linked list consists of nodes where each node contains data and a reference (link) to the next node in the sequence.
   * **Structure:** Each node has two parts:
     + data: Stores the actual data (e.g., task details).
     + next: A reference to the next node in the list.
   * **Operations:**
     + **Insertion:** Can be done at the beginning, end, or middle.
     + **Deletion:** Requires traversal to find the node to be deleted.
     + **Traversal:** Involves iterating from the head node to the end.
2. **Doubly Linked List:**
   * **Definition:** A doubly linked list consists of nodes where each node has references to both the next and previous nodes.
   * **Structure:** Each node has three parts:
     + data: Stores the actual data.
     + next: A reference to the next node.
     + prev: A reference to the previous node.
   * **Operations:** Similar to singly linked lists but allows traversal in both directions.

**Advantages of Linked Lists:**

* **Dynamic Size:** Can grow or shrink dynamically, unlike arrays with fixed size.
* **Efficient Insertions/Deletions:** Adding or removing nodes does not require shifting elements, as in arrays.

**Disadvantages:**

* **Extra Memory:** Requires extra memory for storing references (pointers).
* **Sequential Access:** Cannot access elements directly by index, only sequentially.

**Step 4: Analysis**

**Time Complexity Analysis:**

1. **Add Task:**
   * **Best Case:** O(1) – When adding to an empty list or as the new tail node.
   * **Worst Case:** O(n) – When traversing the entire list to add at the end.
2. **Search Task by ID:**
   * **Best Case:** O(1) – When the task is the head node.
   * **Worst Case:** O(n) – When the task is at the end or not present.
3. **Traverse Tasks:**
   * **Time Complexity:** O(n) – You must visit each node in the list.
4. **Delete Task by ID:**
   * **Best Case:** O(1) – When the task is the head node.
   * **Worst Case:** O(n) – When the task is at the end or not present.

**Advantages of Linked Lists over Arrays:**

* **Dynamic Size:** Linked lists can grow or shrink dynamically, unlike arrays which require resizing.
* **Efficient Insertions/Deletions:** Inserting or deleting elements is more efficient in linked lists as it does not involve shifting elements like in arrays.

**Disadvantages of Linked Lists:**

* **Memory Overhead:** Each node requires extra memory for the pointer/reference.
* **Sequential Access:** Direct access to elements is not possible; traversal is required.

**When to Use Linked Lists:**

* **Dynamic Size:** When the number of elements is not known in advance or changes frequently.
* **Frequent Insertions/Deletions:** When insertions and deletions are more frequent compared to access operations.