**Linked Lists Overview:**

* **Singly Linked List:** A linear data structure where each element (called a node) contains a data part and a reference (or link) to the next node in the sequence. The last node's next reference is usually null, indicating the end of the list.
  + **Advantages:** Simple structure, easy to implement, efficient insertions and deletions (O(1) if you have a pointer/reference to the node).
  + **Disadvantages:** No backward traversal, requires additional memory for storing references.
* **Doubly Linked List:** Each node contains two references: one to the next node and another to the previous node, allowing bidirectional traversal.
  + **Advantages:** Allows traversal in both directions, more flexible for operations like deletions.
  + **Disadvantages:** More complex and requires more memory per node due to the extra reference.

**Time Complexity of Operations:**

* **Add Task:** O(n) in the worst case (appending at the end of the list).
* **Search Task:** O(n), as it may require traversing the entire list.
* **Traverse Tasks:** O(n), as each node needs to be accessed.
* **Delete Task:** O(n) in the worst case (if the task is at the end or not found).

**Advantages of Linked Lists Over Arrays:**

* **Dynamic Size:** Linked lists do not need a predefined size and can grow or shrink as needed.
* **Efficient Insertions/Deletions:** Insertions and deletions are generally more efficient than in arrays, as they do not require shifting elements.
* **Memory Usage:** Linked lists allocate memory as needed, which can be more efficient than allocating a large array upfront.