**Exercise 5: Task Management System**

**Understand Linked Lists:**

**Types of Linked Lists**

Linked lists are a fundamental data structure used to store a collection of elements. The main types of linked lists are Singly Linked Lists and Doubly Linked Lists. Below is an in-depth explanation of each type, including their structure, operations, advantages, and disadvantages.

**1. Singly Linked List**

**Structure:**

* Each node in a singly linked list contains two fields:
  + **Data:** Stores the element.
  + **Next:** A reference (or link) to the next node in the sequence.
* The list starts with a special node called the **head**, which points to the first node of the list.
* The last node in the list has its next reference set to null, indicating the end of the list.

**Operations:**

* **Insertion:**
  + **At the Beginning:** Insert a new node and point its next to the current head, then update the head to this new node. (O(1))
  + **At the End:** Traverse to the end of the list and add a new node. (O(n))
  + **At a Given Position:** Traverse to the desired position and adjust the links accordingly. (O(n))
* **Deletion:**
  + **At the Beginning:** Update the head to point to the next node. (O(1))
  + **At the End:** Traverse to the second last node and set its next to null. (O(n))
  + **At a Given Position:** Traverse to the desired position and adjust the links accordingly. (O(n))
* **Search:** Traverse through the list to find a specific element. (O(n))
* **Traversal:** Start from the head and follow the next references to visit each node. (O(n))

**Advantages:**

* **Dynamic Size:** The size of the list can grow and shrink dynamically.
* **Efficient Insertions/Deletions:** Insertions and deletions are efficient at the beginning and the end (for some cases).

**Disadvantages:**

* **Sequential Access:** Elements must be accessed sequentially from the head, making random access inefficient.
* **Extra Memory:** Requires extra memory for storing references.

**2. Doubly Linked List**

**Structure:**

* Each node in a doubly linked list contains three fields:
  + **Data:** Stores the element.
  + **Next:** A reference (or link) to the next node in the sequence.
  + **Previous:** A reference to the previous node in the sequence.
* The list starts with a special node called the **head**, which points to the first node of the list.
* The last node in the list has its next reference set to null, and the first node has its previous reference set to null.

**Operations:**

* **Insertion:**
  + **At the Beginning:** Insert a new node and adjust the next and previous references accordingly. (O(1))
  + **At the End:** Traverse to the end of the list and add a new node. (O(n))
  + **At a Given Position:** Traverse to the desired position and adjust the links accordingly. (O(n))
* **Deletion:**
  + **At the Beginning:** Update the head to point to the next node and set the previous of the new head to null. (O(1))
  + **At the End:** Traverse to the second last node and set its next to null. (O(n))
  + **At a Given Position:** Traverse to the desired position and adjust the links accordingly. (O(n))
* **Search:** Traverse through the list to find a specific element. (O(n))
* **Traversal:**
  + **Forward:** Start from the head and follow the next references. (O(n))
  + **Backward:** Start from the tail and follow the previous references. (O(n))

**Advantages:**

* **Bidirectional Traversal:** Allows traversal in both forward and backward directions.
* **Efficient Insertions/Deletions:** More flexible insertions and deletions as compared to singly linked lists.

**Disadvantages:**

* **Extra Memory:** Requires extra memory for storing two references (next and previous) per node.
* **Complexity:** More complex to implement and manage due to the additional references.

**Analysis:**

**Time Complexity of Each Operation:**

* **Add Operation:** O(1) (at the beginning of the list)
* **Search Operation:** O(n) (traverse through the list to find the task)
* **Traverse Operation:** O(n) (iterate through all nodes)
* **Delete Operation:** O(n) (find the node to delete and adjust links)

**Advantages of Linked Lists over Arrays for Dynamic Data:**

* **Dynamic Size:** Linked lists can grow and shrink dynamically, whereas arrays have a fixed size.
* **Efficient Insertions/Deletions:** Insertions and deletions are more efficient in linked lists, especially for large datasets, as they do not require shifting elements.
* **Memory Utilization:** Linked lists do not require contiguous memory allocation, which can be an advantage in managing memory more flexibly.