**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

***Steps:***

Understand Linked Lists:

Explain the different types of linked lists (Singly Linked List, Doubly Linked List).

Ans:Types of Linked Lists

1. *Singly Linked List (SLL):*

Structure: Each node contains data and a reference (or pointer) to the next node in the sequence.

Traversal: Can be done in one direction (forward).

Operations:

1. Insertion: Can be done at the beginning, end, or any position.
2. Deletion: Requires traversal to the node before the one to be deleted.

Example: Node(data) -> Node(data) -> Node(data) -> None

1. *Doubly Linked List (DLL):*

Structure: Each node contains data, a reference to the next node, and a reference to the previous node.

Traversal: Can be done in both directions (forward and backward).

Operations:

Insertion: Can be done at the beginning, end, or any position.

Deletion: Requires traversal to the node itself, but references to previous and next nodes make it easier.

Example: None <- Node(data) <-> Node(data) <-> Node(data) -> None

***Setup:***

Create a class Task with attributes like taskId, taskName, and status.

Code: TaskMS.java

***Implementation:***

Implement a singly linked list to manage tasks.

Implement methods to add, search, traverse, and delete tasks in the linked list.

Code: TaskMS.java

***Analysis:***

1. Analyse the time complexity of each operation.

#### 

#### **Singly Linked List (SLL)**

**Insertion**:

*At the beginning:* O(1)O(1)O(1) (Constant time, only the head reference needs to be updated).

*At the end:* O(n)O(n)O(n) (Linear time, need to traverse to the end of the list).

*At a specific position:* O(n)O(n)O(n) (Linear time, need to traverse to the position).

**Deletion**:

*From the beginning:* O(1)O(1)O(1) (Constant time, only the head reference needs to be updated).

*From the end:* O(n)O(n)O(n) (Linear time, need to traverse to the second last node).

*From a specific position:* O(n)O(n)O(n) (Linear time, need to traverse to the position).

**Traversal**: O(n)O(n)O(n) (Linear time, need to visit each node once).

#### 

#### 

#### 

#### 

#### **Doubly Linked List (DLL)**

**Insertion:**

*At the beginning:* O(1)O(1)O(1) (Constant time, update head and previous pointer of the former head).

*At the end:* O(1)O(1)O(1) if tail is maintained, otherwise O(n)O(n)O(n).

*At a specific position:* O(n)O(n)O(n) (Linear time, need to traverse to the position).

**Deletion**:

*From the beginning:* O(1)O(1)O(1) (Constant time, update head and the next node’s previous pointer).

*From the end:* O(1)O(1)O(1) if tail is maintained, otherwise O(n)O(n)O(n).

*From a specific position:* O(n)O(n)O(n) (Linear time, need to traverse to the position).

**Traversal**: O(n)O(n)O(n) (Linear time, can traverse forward or backward).

1. Discuss the advantages of linked lists over arrays for dynamic data.

**Advantages :**

**Dynamic Size**: Linked lists can easily grow and shrink in size by adding or removing nodes, while arrays have a fixed size unless resized (which involves copying elements to a new array).

**Efficient Insertions/Deletions**: Insertions and deletions in linked lists are more efficient compared to arrays. In arrays, shifting of elements is required after insertion/deletion, leading to O(n)O(n)O(n) complexity, whereas linked lists handle these operations in O(1)O(1)O(1) or O(n)O(n)O(n) depending on the position.

**Memory Utilisation**: Linked lists use memory efficiently by allocating memory only when needed, unlike arrays that may allocate extra space that may not be used.

**Flexibility**: Linked lists provide more flexibility in terms of data structure modifications, allowing the implementation of more complex data structures like stacks, queues, and graphs more easily.

**No Wasted Space**: Arrays can have unused space if the reserved size is larger than the actual number of elements, while linked lists do not have this issue as they only use space for actual elements.