**Home Tasks:**

1. List Advantages and disadvantages of linked List

**Advantages of Linked List**

| **Advantage** | **Description** |
| --- | --- |
| **Dynamic Size** | Linked lists can grow or shrink in size during runtime (no need to define size beforehand like arrays). |
| **Efficient Insert/Delete** | Insertion and deletion (especially at the beginning or middle) are fast—**O(1)** if you have the reference to the node. No shifting like arrays. |
| **No Memory Waste** | Uses memory only when needed. No pre-allocation or empty slots like arrays. |
| **Easy to Implement Stacks/Queues** | Linked lists naturally support stack and queue operations using addFirst, removeFirst, etc. |
| **Good for Frequent Insert/Delete** | Ideal when your app performs frequent insertions/deletions (e.g., real-time editing tools, schedulers). |

**Disadvantages of Linked List**

| **Disadvantage** | **Description** |
| --- | --- |
| **Slow Access (No Indexing)** | You can’t directly access elements by index (like arr[5] in arrays). Must traverse from head — **O(n)**. |
| **Extra Memory Per Node** | Each node needs additional memory for a pointer/reference (next, and possibly prev in doubly linked lists). |
| **Poor Cache Locality** | Nodes may be scattered in memory, reducing performance due to poor CPU cache utilization. |
| **Complexity** | More complex to implement and debug compared to arrays (especially with doubly or circular links). |
| **No Backtracking (in Singly Linked List)** | In singly linked lists, you can’t move backward without additional logic or modifying the structure. |

2. Applications of Linked list

* Dynamic Memory Allocation
* Implementing Data Structures
* Operating Systems
* Undo/Redo Functionality
* Web Browsers
* Music or Video Playlist Navigation
* Image Viewer
* CPU Task Scheduling
* Blockchain and Versioning
* Real-Time Multiplayer Games