Linked List

Definition:

A linked list is a linear data structure in which elements are stored in nodes, and each node points to the next node in the sequence, forming a chain-like structure. Unlike arrays, linked lists do not have a fixed size in memory, and their size can be dynamically adjusted during runtime.

Key Components of a Linked List:

1. **Node:**
   - The basic building block of a linked list is a node.
   - Each node contains two fields:
     - Data: The actual information stored in the node.
     - Next: A reference (or link) to the next node in the sequence.
2. **Head:**
   - The first node in the linked list is called the head.
   - The head points to the beginning of the list.

3. **Tail:**
    - The last node in the linked list is called the tail.
    - The tail's "next" reference is typically set to null, indicating the end of the list.

Types of Linked Lists:

1. **Singly Linked List:**
    - Each node points to the next node in the sequence.
    - It only allows for traversing the list in one direction (forward).
2. **Doubly Linked List:**
    - Each node has two references: one to the next node and another to the previous node.
    - Allows for traversal in both forward and backward directions.
3. **Circular Linked List:**
    - The last node in the list points back to the first node, creating a circular structure.

Time Complexity:

Operation	Singly Linked List (SLL)	Doubly Linked List (DLL)
Display	O(n)	O(n)
Search	O(n)	O(n)
Insertion at the Beginning (SLL/DLL)	O(1)	O(1)
Insertion at the End (SLL/DLL)	O(n)	O(1)

Insertion After Node (SLL/DLL)	O(n)	O(1)
Insertion Before Node (DLL)	N/A	O(1)
Insertion at Position (SLL/DLL)	O(n)	O(n)
Deletion at the Beginning (SLL/DLL)	O(1)	O(1)
Deletion at the End (SLL/DLL)	O(n)	O(1)
Deletion After Node (SLL/DLL)	O(n)	O(1)
Deletion Before Node (DLL)	N/A	O(1)
Deletion at Position (SLL/DLL)	O(n)	O(n)

Difference Between Single Linked List and Double Linked List:

Feature	Single Linked List	Double Linked List
Node Structure	Each node contains data and a pointer to the next node.	Each node contains data, a pointer to the next node, and a pointer to the previous node.
Traversal	Can only be traversed in the forward direction.	Can be traversed in both the forward and backward direction.
Insertion	Requires the location of the node before the insertion point.	Only requires the location of the insertion point.
Deletion	Requires the location of the node before the deletion point.	Only requires the location of the node to be deleted.
Memory Usage	Uses less memory as it requires only one pointer per node.	Uses more memory as it requires two pointers per node.
Applications	Suitable for applications where only forward traversal is needed, such as **stacks**.	Suitable for applications where both forward and backward traversal is needed, such as

		implementing a **Music Player Playlist functionality.**

Drawbacks of Single Linked List:

1. Unidirectional Traversal
2. Inefficient Removal of Nodes
3. No Direct Access to Previous Node
4. Extra Pointer for Tail
5. Limited Application in Certain Algorithms
6. Difficulty in Finding the kth Element from the End
7. Not Suitable for Some Algorithms
8. Difficulty in Detecting Loop

Differences between Linked Lists and Arrays:

Feature	Linked List (LL)	Array
Memory Allocation	Dynamic, non-contiguous	Static, contiguous
Size Flexibility	Dynamic, can change during runtime	Fixed, predefined size
Insertion/Deletion	Efficient for insertions/deletions at any position	Inefficient for insertions/deletions in the middle
Random Access	Inefficient, O(n) time complexity for access	Efficient, O(1) time complexity for access
Memory Overhead	Higher due to storage of pointers	Lower, as no additional pointers
Memory Usage	May use more memory due to pointers	Efficient usage of memory

Contiguity	Non-contiguous, scattered in memory	Contiguous, stored in a single block
Implementation Complexity	Generally simpler to implement	Straightforward implementation
Size Determination	Not required to specify size initially	Requires specifying size at declaration
Application Flexibility	Well-suited for dynamic data structures	Suitable for scenarios with fixed-size data

Advantages and Disadvantages of Linked Lists:

Advantages of Linked Lists:

1. **Dynamic Size:**
    - **Advantage:** Easily accommodates varying data sizes as memory is dynamically allocated.
    - **Use Case:** Ideal for situations where the size of the data is not known in advance.
2. **Efficient Insertion and Deletion:**
    - **Advantage:** Adding or removing elements is efficient, requiring adjustments to pointers.
    - **Use Case:** Suitable for scenarios involving frequent insertions and deletions.
3. **No Wastage of Memory:**
    - **Advantage:** Memory utilization is efficient as each element can be allocated precisely as needed.
    - **Use Case:** Useful when optimizing memory usage is a critical concern.
4. **Ease of Implementation:**
    - **Advantage:** Simple to implement and understand, making it accessible for various applications.
    - **Use Case:** Commonly used in educational settings and introductory programming courses.

5. **Versatility:**
   - **Advantage:** Supports various data structures, including stacks, queues, and symbol tables.
   - **Use Case:** Versatile for implementing different algorithms and data structures.

Disadvantages of Linked Lists:

1. **Sequential Access:**
   - **Disadvantage:** Accessing elements requires sequential traversal from the beginning.
   - **Impact:** Inefficient for scenarios requiring random access or searching.
2. **Extra Memory Usage:**
   - **Disadvantage:** Additional memory is needed for storing pointers, increasing overhead.
   - **Impact:** Higher space complexity compared to arrays in certain cases.
3. **Complexity of Implementation:**
   - **Disadvantage:** Implementing linked lists can be more complex than arrays.
   - **Impact:** May pose challenges in scenarios where simplicity is a priority.
4. **Lack of Cache Locality:**
   - **Disadvantage:** Poor cache locality can result in suboptimal performance.
   - **Impact:** May affect performance in applications with high memory access patterns.
5. **Memory Overhead:**
   - **Disadvantage:** Requires additional memory for storing pointers, impacting efficiency.

- **Impact:** Increases memory overhead, particularly in resource-constrained environments.

Applications of Single Linked List:

1. Dynamic Memory Allocation
2. Implementation of Stacks
3. Implementation of Queues
4. Traversal and Search Operations
5. Undo Mechanisms