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

Singly Linked List: singly linked list consists of nodes where each node contains a data element and a reference (or pointer) to the next node in the sequence. efficient for one-way traversal and requires less memory per node.

Doubly Linked List: A doubly linked list consists of nodes where each node contains a data element, a reference to the next node, and a reference to the previous node. supports two-way traversal and more efficient

1. Analyze the time complexity of each operation.

1. Singly Linked List

Add Operation:

At the Beginning: O(1)

Inserting a node at the head of the list is constant time since it only involves updating the head pointer.

At the End: O(n)

Inserting a node at the end requires traversing the entire list to find the last node unless you maintain a tail pointer.

At a Specific Position: O(n)

Requires traversing the list to reach the desired position.

Search Operation:

Time Complexity: O(n)

Searching for an element requires traversing the list from the head to the end, resulting in linear time complexity.

Traverse Operation:

Time Complexity: O(n)

Traversing all nodes requires visiting each node once, leading to linear time complexity.

Delete Operation:

At the Beginning: O(1)

Removing the head node involves updating the head pointer.

At the End: O(n)

Removing the last node requires traversing the list to find the second-to-last node.

At a Specific Position: O(n)

Requires traversing the list to reach the node before the one to be deleted, and then updating pointers.

2. Doubly Linked List

Add Operation:

At the Beginning: O(1)

Inserting a node at the head involves updating the head pointer and adjusting pointers of the new node and the old head.

At the End: O(1) (if tail pointer is maintained)

Inserting at the end is efficient if a tail pointer is available; otherwise, it’s O(n) to find the end.

At a Specific Position: O(n)

Requires traversal to the desired position, but insertion itself is O(1) once the position is reached.

Search Operation:

Time Complexity: O(n)

Searching requires traversing from the head to the tail or vice versa, resulting in linear time complexity.

Traverse Operation:

Time Complexity: O(n)

Traversing through all nodes requires visiting each node once in either direction.

Delete Operation:

At the Beginning: O(1)

Removing the head node involves updating the head pointer and adjusting pointers of the new head.

At the End: O(1) (if tail pointer is maintained)

Removing the last node is efficient with a tail pointer; otherwise, it’s O(n) to find the end.

At a Specific Position: O(1) (once position is reached)

Removing a node at a specific position is efficient as it involves updating pointers of adjacent nodes.

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

Dynamic Size: Linked lists can easily grow and shrink, unlike arrays which have a fixed size once allocated.

Efficient Insertions/Deletions: Adding or removing elements is quick in linked lists, especially at the beginning or middle, while arrays require shifting elements.

Memory Utilization: Linked lists use memory efficiently by allocating only as needed, whereas arrays may waste space or require resizing.

Flexibility: Linked lists handle changing data sizes better and do not need reallocation, unlike arrays which need resizing and copying.