**Types of Linked Lists**

**Singly Linked List**

* Each node has:
  + data
  + next (pointer to next node)
* **Traversal** is only possible in one direction (forward).

**Structure:**

[Data | Next] -> [Data | Next] -> [Data | Next] -> null

**Doubly Linked List**

* Each node has:
  + data
  + next (pointer to next node)
  + prev (pointer to previous node)
* **Traversal** is possible in both directions (forward and backward).

**Structure:**

null <- [Prev | Data | Next] <-> [Prev | Data | Next] <-> null

**2. Time Complexity Analysis**

| **Operation** | **Singly Linked List** | **Doubly Linked List** | **Explanation** |
| --- | --- | --- | --- |
| Access (by index) | O(n) | O(n) | Must traverse from head |
| Search | O(n) | O(n) | Linear traversal |
| Insert at Beginning | O(1) | O(1) | Adjust head and pointers |
| Insert at End | O(n) | O(n), O(1)\* | Traverse to end (\*O(1) if tail is maintained) |
| Insert in Middle | O(n) | O(n) | Traverse then insert |
| Delete at Beginning | O(1) | O(1) | Adjust head |
| Delete at End | O(n) | O(n), O(1)\* | Traverse to end (\*O(1) if tail used) |
| Delete in Middle | O(n) | O(n) | Traverse then delete |

3. **Advantages of Linked Lists over Arrays**

| **Feature** | **Linked List** | **Array** |
| --- | --- | --- |
| **Dynamic Size** | Grows/shrinks as needed (no fixed size) | Fixed size; difficult to resize |
| **Insert/Delete (Middle)** | Efficient (just change links) | Costly (requires shifting elements) |
| **Memory Use** | Uses only needed memory | May allocate unused memory |
| **No memory relocation** | New nodes can be scattered | Needs continuous memory block |

**When to Prefer Linked Lists**

Use linked lists when:

* You need **frequent insertions or deletions**.
* The **size of data is dynamic or unknown**.
* You **don’t need fast random access** to elements.