**1. Linked Lists**

**Singly Linked List**

**Description:**  
• A linear data structure where each node contains:  
 1. **Data**  
 2. **Pointer to the next node**  
• Last node points to **null**.  
• Traversal is **one-way** (forward only).

**Use Case:**  
• Efficient insertion/deletion at beginning or middle without shifting elements.

**Doubly Linked List**

**Description:**  
• Each node contains:  
 1. **Data**  
 2. **Pointer to the next node**  
 3. **Pointer to the previous node**  
• Supports **bidirectional traversal**.

**Use Case:**  
• Useful when two-way navigation is needed (e.g., back/forward task navigation).

**4. Analysis**

**Time Complexity of Linked List Operations**

| **Operation** | **Singly Linked List** | **Doubly Linked List** |
| --- | --- | --- |
| **Add (beginning)** | O(1) | O(1) |
| **Add (end)** | O(n)\* | O(n)\* |
| **Add (middle/index)** | O(n) | O(n) |
| **Delete (beginning)** | O(1) | O(1) |
| **Delete (end)** | O(n) | O(n) |
| **Delete (middle)** | O(n) | O(n) |
| **Search** | O(n) | O(n) |
| **Traverse** | O(n) | O(n) |

\*Can be reduced to O(1) if a **tail pointer** is maintained.

**Advantages of Linked Lists Over Arrays**

| **Feature** | **Arrays** | **Linked Lists** |
| --- | --- | --- |
| **Memory Allocation** | Fixed size (static) | Dynamic, grows as needed |
| **Insertion/Deletion** | Costly (requires shifting) | Efficient (no shifting needed) |
| **Memory Usage** | May waste space or overflow | Uses only needed memory |
| **Access Time (by index)** | O(1) | O(n) |
| **Data movement on changes** | High overhead | Low overhead |

**When to Use Linked Lists**

| **Situation** | **Recommended Use** |
| --- | --- |
| Task list grows/shrinks dynamically | Linked List |
| Frequent additions/deletions | Linked List |
| Need for random/index access | Use Arrays instead |