1. Arrays

- **Description**: Collection of elements identified by index.
- **Operations**:
    - Access: `array[i]`
    - Insertion: Add at the end or specific index.
    - Deletion: Remove from the end or specific index.
    - Traversal: Loop through elements.
- **Time Complexities**:
    - Access: O(1)
    - Insertion: O(1) (end), O(n) (middle)
    - Deletion: O(1) (end), O(n) (middle)
    - Search: O(n)
- **Applications**: Lookup tables, matrices, database indexing.

2. Linked Lists

- **Description**: Sequence of nodes, each containing data and a reference to the next node.
- **Types**: Singly, Doubly, Circular.
- **Operations**:
    - Insertion: Add node at the beginning, end, or middle.
    - Deletion: Remove node from beginning, end, or middle.
    - Traversal: Loop through nodes.
- **Time Complexities**:
    - Access: O(n)
    - Insertion: O(1) (beginning), O(n) (end or middle)
    - Deletion: O(1) (beginning), O(n) (end or middle)
    - Search: O(n)
- **Applications**: Implementing stacks and queues, adjacency lists in graphs.

3. Stacks

- **Description**: Collection of elements with Last-In-First-Out (LIFO) access.
- **Operations**:
    - Push: Add element to the top.
    - Pop: Remove element from the top.
    - Peek: Access top element.
- **Time Complexities**:
    - Push: O(1)
    - Pop: O(1)

    - Peek: O(1)
- **Applications**: Function call management, undo mechanisms, syntax parsing.

4. Queues

- **Description**: Collection of elements with First-In-First-Out (FIFO) access.
- **Types**: Simple, Circular, Priority.
- **Operations**:
    - Enqueue: Add element to the rear.
    - Dequeue: Remove element from the front.
    - Front: Access front element.
- **Time Complexities**:
    - Enqueue: O(1)
    - Dequeue: O(1)
    - Front: O(1)
- **Applications**: Scheduling tasks, resource management, buffering.

5. Trees

- **Description**: Hierarchical structure with nodes connected by edges.
- **Types**: Binary Tree, Binary Search Tree (BST), AVL Tree, Red-Black Tree, Heap, Trie.
- **Operations**:
    - Insertion: Add a node.
    - Deletion: Remove a node.
    - Traversal: In-order, Pre-order, Post-order.
    - Search: Find a node.
- **Time Complexities**:
    - Insertion: O(log n) for balanced trees, O(n) for unbalanced.
    - Deletion: O(log n) for balanced trees, O(n) for unbalanced.
    - Search: O(log n) for balanced trees, O(n) for unbalanced.
    - Traversal: O(n)
- **Applications**: Hierarchical data representation, databases (B-trees), memory management (heaps).

6. Graphs

- **Description**: Set of vertices connected by edges.
- **Types**: Directed, Undirected, Weighted, Unweighted.
- **Representations**: Adjacency Matrix, Adjacency List.
- **Operations**:
    - Traversal: Depth-First Search (DFS), Breadth-First Search (BFS).

    - Shortest Path: Dijkstra's, Bellman-Ford, Floyd-Warshall.
    - Minimum Spanning Tree: Kruskal's, Prim's.
- **Time Complexities**:
    - Traversal: O(V + E)
    - Shortest Path: O(E + V log V) for Dijkstra's, O(V^3) for Floyd-Warshall.
    - MST: O(E log V) for Kruskal's and Prim's.
- **Applications**: Network routing, social networks, dependency resolution.

7. Hashing

- **Description**: Technique to map keys to array indices using hash functions.
- **Collision Resolution**: Chaining, Open Addressing.
- **Operations**:
    - Insertion: Add key-value pair.
    - Deletion: Remove key-value pair.
    - Search: Find value by key.
- **Time Complexities**:
    - Insertion: O(1)
    - Deletion: O(1)
    - Search: O(1)
- **Applications**: Implementing dictionaries, caches, database indexing.

8. Heaps

- **Description**: Complete binary tree used for priority queues.
- **Types**: Min-Heap, Max-Heap.
- **Operations**:
    - Insertion: Add an element.
    - Deletion: Remove root element.
    - Peek: Access root element.
- **Time Complexities**:
    - Insertion: O(log n)
    - Deletion: O(log n)
    - Peek: O(1)
- **Applications**: Priority queues, graph algorithms (Dijkstra's, Prim's).

Summary Table

Data Structure	Access	Search	Insertion	Deletion	Applications

Array	O(1)	O(n)	O(1)/O(n)	O(1)/O(n)	Matrices, lookup tables
Linked List	O(n)	O(n)	O(1)	O(1)	Implementing stacks/queues, graphs
Stack	O(n)	O(n)	O(1)	O(1)	Function calls, undo mechanisms
Queue	O(n)	O(n)	O(1)	O(1)	Scheduling, buffering
Binary Tree	O(log n)	O(log n)	O(log n)	O(log n)	Hierarchical data, databases
BST	O(log n)	O(log n)	O(log n)	O(log n)	Searching, sorting
Heap	O(n)	O(n)	O(log n)	O(log n)	Priority queues, sorting
Hash Table	O(1)	O(1)	O(1)	O(1)	Dictionaries, caching
Graph	O(V+E)	O(V+E)	O(V+E)	O(V+E)	Network routing, dependencies

Additional Notes

- **Time Complexity Notation**: `O(1)` is constant, `O(n)` is linear, `O(log n)` is logarithmic, `O(V + E)` is for graph traversals where V is vertices and E is edges.
- **Balanced vs. Unbalanced Trees**: Balanced trees maintain O(log n) operations, unbalanced can degrade to O(n).
- **Choosing the Right Structure**: Depends on specific needs like speed, memory usage, and type of operations frequently performed.