Arrays

Definition: A collection of elements identified by index or key.

Key Operations:

- **Access:** O(1)
- **Insertion:**
    - At the end: O(1)
    - At the beginning or middle: O(n)
- **Deletion:**
    - From the end: O(1)
    - From the beginning or middle: O(n)
- **Search:**
    - Linear Search: O(n)
    - Binary Search (sorted array): O(log n)

Use Cases: When you need fast access to elements and the size of the collection is fixed.

Linked Lists

Definition: A linear collection of data elements, called nodes, where each node points to the next node.

Types:

- **Singly Linked List:** Nodes point to the next node.
- **Doubly Linked List:** Nodes point to both the next and previous nodes.

Key Operations:

- **Access:** O(n)
- **Insertion:**
    - At the beginning: O(1)
    - At the end: O(1) (if tail pointer is maintained)
    - In the middle: O(n)
- **Deletion:**
    - From the beginning: O(1)
    - From the end: O(n)
    - From the middle: O(n)
- **Search:** O(n)

Use Cases: When you need efficient insertion and deletion at the beginning or end, and do not require fast access to elements by index.

Stacks

Definition: A collection of elements that follows the Last-In-First-Out (LIFO) principle.

Key Operations:

- **Push (insert):** O(1)
- **Pop (remove):** O(1)
- **Peek (top element):** O(1)

Use Cases: Function call management (call stack), expression evaluation (postfix, prefix), backtracking algorithms.

Queues

Definition: A collection of elements that follows the First-In-First-Out (FIFO) principle.

Key Operations:

- **Enqueue (insert):** O(1)
- **Dequeue (remove):** O(1)
- **Peek (front element):** O(1)

Use Cases: Order processing, task scheduling, breadth-first search in graphs.

Heaps

Definition: A specialized tree-based data structure that satisfies the heap property. For a max heap, each parent node is greater than or equal to its children; for a min heap, each parent node is less than or equal to its children.

Key Types:

- **Min Heap:** The smallest element is at the root.
- **Max Heap:** The largest element is at the root.

Key Operations:

- **Insertion:** O(log n)
- **Deletion (Extract Max/Min):** O(log n)
- **Peek (Get Max/Min):** O(1)
- **Heapify:** O(n) for building a heap from an arbitrary array.

Use Cases: Priority queues, scheduling algorithms, graph algorithms (Dijkstra's, Prim's), heapsort.

Trees

Definition: A hierarchical structure with a root node and child nodes forming a parent-child relationship.

Types:

- **Binary Tree:** Each node has at most two children.
- **Binary Search Tree (BST):** A binary tree where the left child contains values less than the parent and the right child contains values greater than the parent.

Key Operations:

- **Access/Search (BST):** O(log n) on average, O(n) in the worst case (unbalanced tree)
- **Insertion (BST):** O(log n) on average, O(n) in the worst case
- **Deletion (BST):** O(log n) on average, O(n) in the worst case
- **Traversal (In-order, Pre-order, Post-order):** O(n)

Use Cases: Hierarchical data representation (file systems), quick search, insert, delete operations (with balanced trees like AVL or Red-Black trees).

AVL Trees

Definition: A self-balancing binary search tree where the difference between the heights of the left and right subtrees of any node is at most one.

Key Operations:

- **Insertion:** O(log n)
- **Deletion:** O(log n)
- **Search:** O(log n)

Key Concepts:

- **Balance Factor:** The difference in heights between the left and right subtrees. It should be -1, 0, or 1 for all nodes.
- **Rotations:** Used to maintain balance during insertion and deletion.
  - **Single Rotation (Left or Right)**
  - **Double Rotation (Left-Right or Right-Left)**

Use Cases: When you need guaranteed O(log n) time complexity for search, insertion, and deletion operations, making it ideal for applications like databases.

Minimum Spanning Tree (MST)

Definition: A spanning tree of a graph that connects all the vertices together, without any cycles, and with the minimum possible total edge weight.

Key Algorithms:

- **Kruskal's Algorithm:** Adds edges in increasing order of weight, avoiding cycles.
  - **Time Complexity:** $O(E \log E)$ or $O(E \log V)$, where E is the number of edges and V is the number of vertices.
- **Prim's Algorithm:** Starts from an arbitrary node and grows the MST by adding the smallest edge that connects a vertex in the MST to a vertex outside it.
  - **Time Complexity:** $O(V^2)$ or $O(E + V \log V)$ with a priority queue.

Use Cases: Network design (e.g., electrical grids, computer networks), approximation algorithms for NP-hard problems.

Graphs

Definition: A collection of nodes (vertices) and edges connecting some or all of the nodes.

Types:

- **Undirected Graph:** Edges have no direction.
- **Directed Graph (Digraph):** Edges have direction.
- **Weighted Graph:** Edges have weights or costs associated with them.
- **Unweighted Graph:** Edges have no weights.

Key Operations:

- **Traversal (BFS, DFS):** $O(V + E)$ where V is the number of vertices and E is the number of edges.
- **Shortest Path (Dijkstra's, Bellman-Ford):** $O(E + V \log V)$ for Dijkstra's with a priority queue.
- **Minimum Spanning Tree (Kruskal's, Prim's):** $O(E \log V)$ for Kruskal's with a priority queue.

Use Cases: Network routing, social network analysis, dependency resolution.

Hash Tables

Definition: A collection of key-value pairs, where each key is mapped to a value using a hash function.

Key Operations:

- **Access/Search:** O(1) on average, O(n) in the worst case (due to collisions)
- **Insertion:** O(1) on average, O(n) in the worst case
- **Deletion:** O(1) on average, O(n) in the worst case

Use Cases: Fast data retrieval (caches), indexing databases, associative arrays.

Recursion

Definition: A programming technique where a function calls itself to solve a smaller instance of the problem.

Key Concepts:

- **Base Case:** The condition under which the recursion stops.
- **Recursive Case:** The part of the function where the function calls itself with a smaller problem instance.
- **Stack Overflow:** A potential issue where too many recursive calls cause the call stack to exceed its limit.

Time Complexity Analysis:

- **Factorial Function:** O(n)
- **Fibonacci Sequence (Naive Recursion):** O(2^n) due to overlapping subproblems.
- **Fibonacci Sequence (Memoization/Dynamic Programming):** O(n)

Use Cases: Divide and conquer algorithms (e.g., mergesort, quicksort), backtracking (e.g., solving mazes, n-queens problem), dynamic programming.

Advantages of Recursion:

- Simplifies code for problems that have a natural recursive structure (e.g., tree traversals).
- Helps in solving complex problems by breaking them down into simpler subproblems.

Disadvantages:

- Can lead to high memory usage due to the call stack.
- Often slower than iterative solutions due to function call overhead.