Binary Search Trees (BSTs)

📖 **Key Concept:**
A **Binary Search Tree (BST)** is a **binary tree** where:
✔ Left subtree values **<** root value.
✔ Right subtree values **>** root value.
✔ **Inorder traversal** results in **sorted order**.

🌳 **BST Operations:**
🔍 **Search:**
- Uses **recursive traversal** to locate a key.
- Complexity: **O(log n)** (if balanced).

➕ **Insert:**
- Adds a node while maintaining BST properties.
- Worst case **O(n)** (if unbalanced).

➖ **Remove:**
- Three cases: **Leaf node**, **One child**, **Two children**.
- Replace with **max left subtree** or **min right subtree**.

⚡ **BST Efficiency & Balance:**
- **Balanced BSTs (AVL, Splay Trees)** → **O(log n)** operations.
- **Unbalanced BSTs (linked list shape)** → **O(n)** worst-case complexity.
- Maintaining **balance** is crucial for performance.

📌 **Takeaway:**
BSTs provide **efficient searching, insertion, and deletion**, but balancing techniques **(AVL, 2-3 Trees, Splay Trees)** are needed to **optimize performance** in real-world applications.