Binary Search Tree

By Mani Abedii

View Full DSA Repository

A clear, compact guide covering essential data structures and algorithms, designed for easy understanding and quick learning.

A **Binary Search Tree (BST)** is a specialized form of a binary tree in which every node satisfies the following property:

- The value of every node in the **left subtree** is **less than** the value of the node.
- The value of every node in the **right subtree** is **greater than** the value of the node.

This property must hold for **every node** in the tree.

BSTs are widely used for efficient searching, insertion, and deletion. Because of the ordering property, operations such as lookup, insertion, and deletion can often be performed in O(h) time, where h is the height of the tree. In a balanced BST, this results in O(log n) time complexity for these operations.

Key Characteristics of a BST:

- Each node has at most two children: left & right
- Left child contains a value smaller than its parent.
- Right child contains a value greater than its parent.
- There are no duplicates in the tree.
- Inorder traversal yields elements in sorted order.
- Height of the BST affects operation efficiency.

BST Operations:

- **Insertion:**

  To insert a node:

  1. Start at the root.
  2. Compare the new value with the current node:
     - If smaller → move left.
     - If larger → move right.
  3. Repeat until a null position is found, then insert the node.

  Time Complexity: **O(h)**

- **Search:**

  To search for a value:

  1. Start at the root.
  2. Compare the value with the current node:
     - If equal → found
     - If smaller → go left
     - If larger → go right
  3. Repeat until the node is found or null pointer is reached.

  Time Complexity: **O(h)**

- **Deletion:**

  Deleting a node in a BST depends on its children:

  1. Leaf node: remove it directly.
  2. Node with one child: replace the node with its child.
  3. Node with two children: replace the node with either:
     - Inorder successor: the smallest node in the right subtree.
     - Inorder predecessor: the largest node in the left subtree.

     This ensures the BST property is preserved

  Time Complexity: **O(h)**

- **Traversal:**

  BST traversal methods are the same as binary trees:

  1. Inorder: produces elements in sorted order.
  2. Preorder: useful for copying the tree.
  3. Postorder: useful for deleting the tree.
  4. Level-order: visits nodes level by level

BST is efficient in search, insertion & deletion for balanced trees while maintaining sorted data, but it can have poor performance when unbalanced & will require additional algorithms to maintain balance. Many advanced trees (AVL, Red-Black) were created to maintain balance automatically.