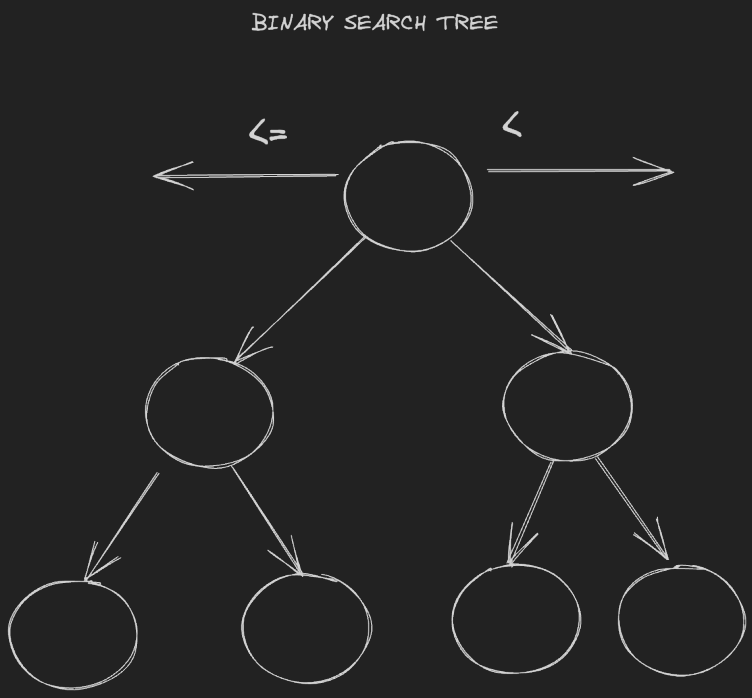
Operation	Balanced Tree	Skewed Tree
Find	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

Definition

A binary search tree (BST) is a **binary tree data structure** that satisfies the following properties:

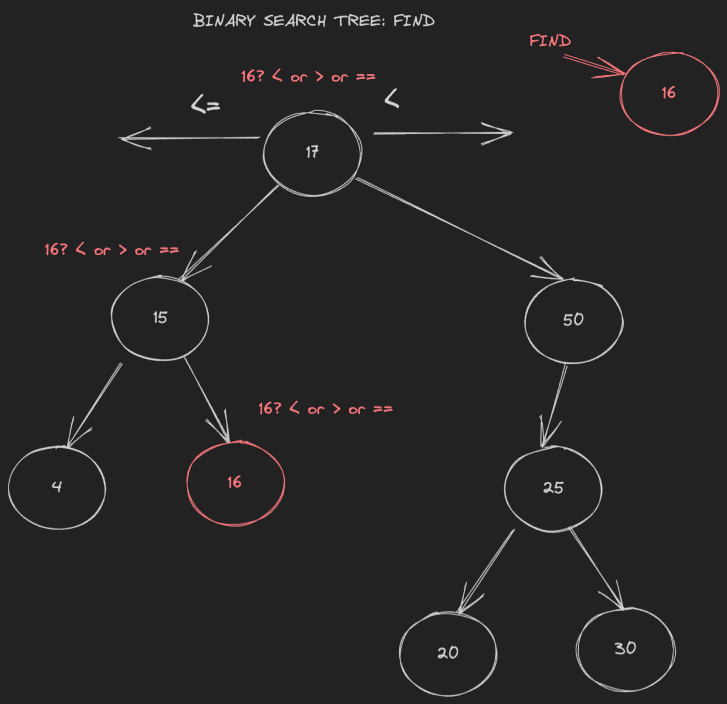
- Each node in the BST has a value.
- The value of every node in the left subtree is less than the value of the node.
- The value of every node in the right subtree is greater than or equal to the value of the node.
- Both the left and right subtrees of any node are themselves binary search trees.



Find

To search for a specific value in a BST, we start at the root node and compare the value with the current node's value. If the value is equal, we have found the node. If the value is less than the current node's value, we move to the left child. If the value is greater, we move to the right child. We repeat this process until we find the node with the desired value or reach a null (leaf) node.

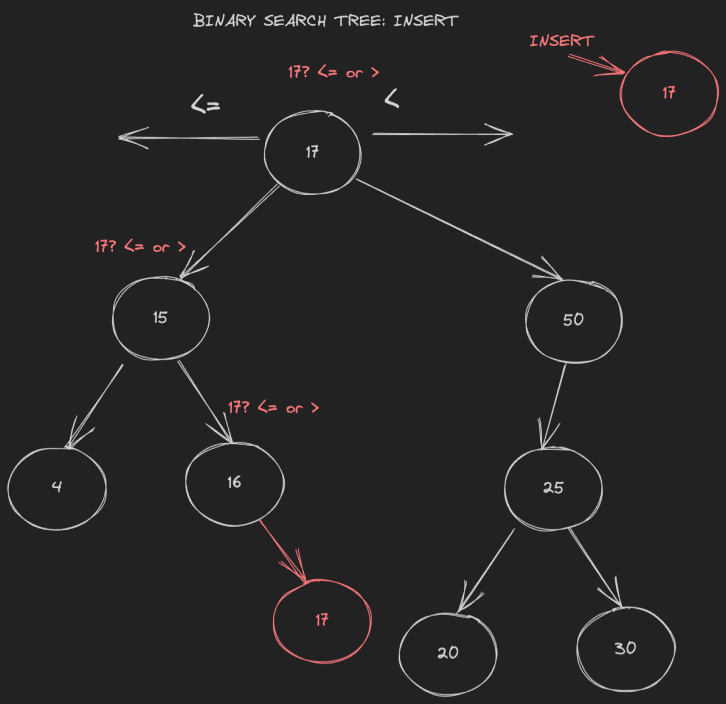
Running time: $O(\log n)$ on average for a balanced BST, $O(n)$ in the worst case for a skewed tree.



Insert

To insert a value into a BST, we start at the root node and compare the value with the current node's value. If the value is less than the current node's value and the left child is null, we create a new node and set it as the left child. If the value is greater and the right child is null, we create a new node and set it as the right child. If the value is less or greater and the corresponding child is not null, we move to that child and repeat the insertion process. This ensures that the BST property is maintained.

Running time: $O(\log n)$ on average for a balanced BST, $O(n)$ in the worst case for a skewed tree

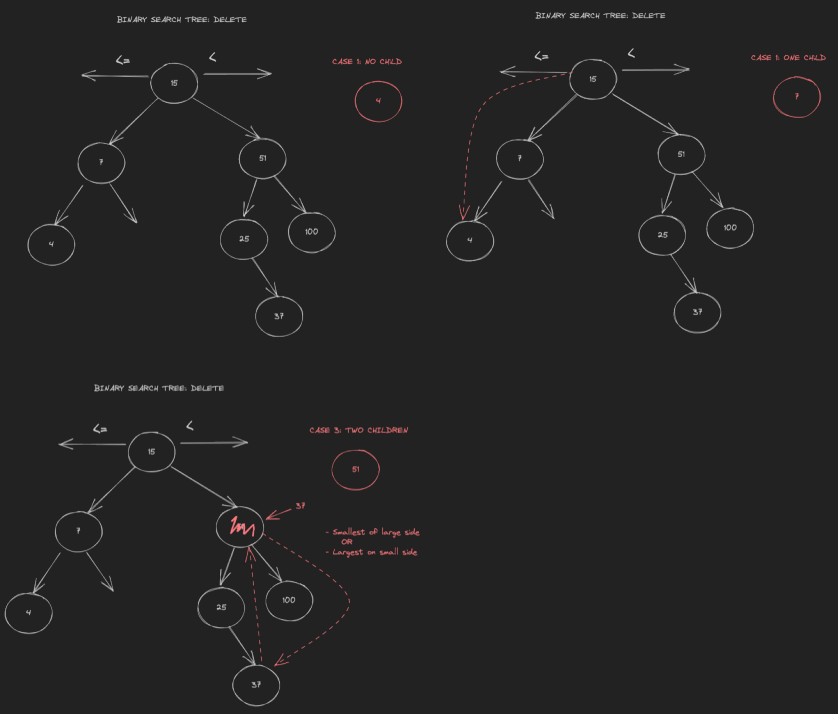


Delete

Deleting a node from a BST can be more complex than insertion or search operations. There are three cases to consider:

1. The node to be deleted is a leaf node (has no children). In this case, we simply remove the node from the tree.
2. The node to be deleted has only one child. In this case, we replace the node with its child.
3. The node to be deleted has two children. In this case, we find the node's in-order successor (the smallest value in the right subtree) or in-order predecessor (the largest value in the left subtree) and replace the node's value with the successor/predecessor value. Then we recursively delete the successor/predecessor node.

Running time: $O(\log n)$ on average for a balanced BST, $O(n)$ in the worst case for a skewed tree



Use Case

- Efficient searching and retrieval of data in a sorted manner.
- Implementing various algorithms like binary search and binary heap.
- Maintaining data in a sorted order while allowing efficient insertion and deletion operations.