Trees – Binary Search Tree (BST)

1. Introduction

A **Binary Search Tree (BST)** is a special type of **binary tree** that follows a **specific ordering property**.

Because of this property, BSTs allow **efficient searching, insertion, and deletion** of elements.

BSTs are widely used in **databases, searching algorithms, and indexing systems**.

---

2. What is a Binary Search Tree?

A **Binary Search Tree** is a binary tree where for **every node**:

- All values in the **left subtree** are **less than** the node's value
- All values in the **right subtree** are **greater than** the node's value

This property must hold **recursively for every subtree**.

---

3. BST Property (Key Rule)

For any node N:

```
Left Subtree < N < Right Subtree
```

This ordering is what makes searching fast.

---

4. Structure of a Binary Search Tree

Example BST:

```
    50
   /  \
  30   70
```

```
  / \   / \
 20  40  60  80
```

- Left values are smaller
- Right values are larger
- No duplicate values (in standard BST)

---

5. Why Use a Binary Search Tree?

BST is used because it:

- Reduces search time
- Organizes data efficiently
- Maintains sorted structure
- Supports dynamic insertions and deletions

---

6. Searching in a BST (Plain English Logic)

1. Start at the root
2. If the value equals the current node → found
3. If the value is smaller → go to left subtree
4. If the value is larger → go to right subtree
5. Repeat until found or NULL

---

7. Insertion in a BST (Concept)

To insert a value:

1. Start from the root
2. Compare the value with current node
3. Move left or right based on comparison
4. Insert the value at the correct leaf position

BST property is preserved after insertion.

---

8. Deletion in a BST (Overview)

Deletion depends on the number of children:

1. **Leaf Node** – Delete directly
2. **One Child** – Replace node with its child
3. **Two Children** – Replace with inorder successor or predecessor

(Deletion is more complex but very important.)

---

9. Traversal in BST

BST supports all tree traversals:

- Preorder
- Inorder
- Postorder
- Level Order

🔷 **Inorder Traversal of BST gives sorted output**.

---

10. Time Complexity of BST Operations

Operation	Average Case	Worst Case
Search	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

Worst case occurs when BST becomes **skewed**.

---

11. Balanced vs Skewed BST

Type	Height	Performance
Balanced BST	O(log n)	Efficient
Skewed BST	O(n)	Poor

Balanced trees are preferred for performance.

---

12. Advantages of Binary Search Tree

- Fast searching
- Maintains sorted data
- Dynamic structure
- Efficient insertion and deletion

---

13. Limitations of Binary Search Tree

- Can become skewed
- No guaranteed balance
- Performance degrades in worst case
- More complex than arrays

---

14. Applications of BST

- Database indexing
- Symbol tables
- Searching systems
- File organization
- Memory allocation
- Autocomplete systems

---

15. BST vs Binary Tree

Feature	Binary Tree	Binary Search Tree
Ordering	No order	Ordered
Searching	Inefficient	Efficient
Traversal Output	Random	Sorted (Inorder)

16. Summary

- BST is a binary tree with ordering property
- Left subtree < root < right subtree
- Enables fast search operations
- Inorder traversal gives sorted order
- Performance depends on tree height