Program to Build a Binary Search Tree (BST)

Mr. Akash Yadav

Assistant Professor

Artificial Intelligence & Data Science

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Write a Program to Build a Binary Search Tree (BST)**

Lesson Objectives
Understand the concept of Binary Search Tree (BST) and its properties.
Learn how to insert elements into a BST.
Explore recursive and iterative insertion logic.
Analyze time and space complexity of BST creation

Problem Statement:

Write a program to build a Binary Search Tree (BST) by inserting elements one by one.

The program should:

1. Accept a sequence of integer values.
2. Insert each element according to BST rules.
3. Display the tree using inorder traversal.

Concept

A Binary Search Tree (BST) is a binary tree where:

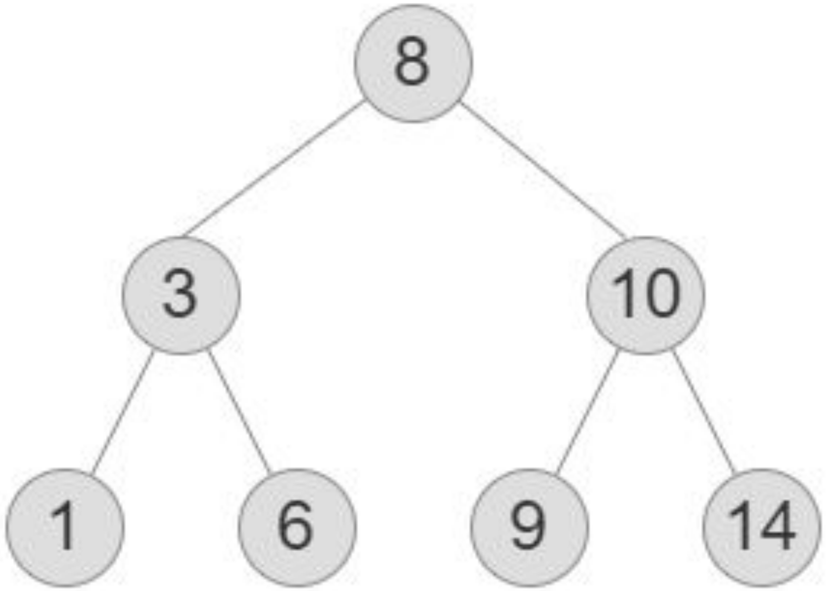
1. The left child has a smaller value than its parent.
2. The right child has a larger value than its parent.
3. No duplicate elements are stored.
4. Inorder traversal of a BST gives a sorted sequence.

Algorithm/Logic

1. Start with an empty tree (root = null).
2. For each element:
   a. If the tree is empty → new node becomes root.
   b. If value < root.data → insert into left subtree.
   c. If value > root.data → insert into right subtree.
3. Repeat recursively until correct position is found.
4. Use inorder traversal to display the BST.
5. Inorder order: Left → Root → Right.
6. This ensures ascending order output.

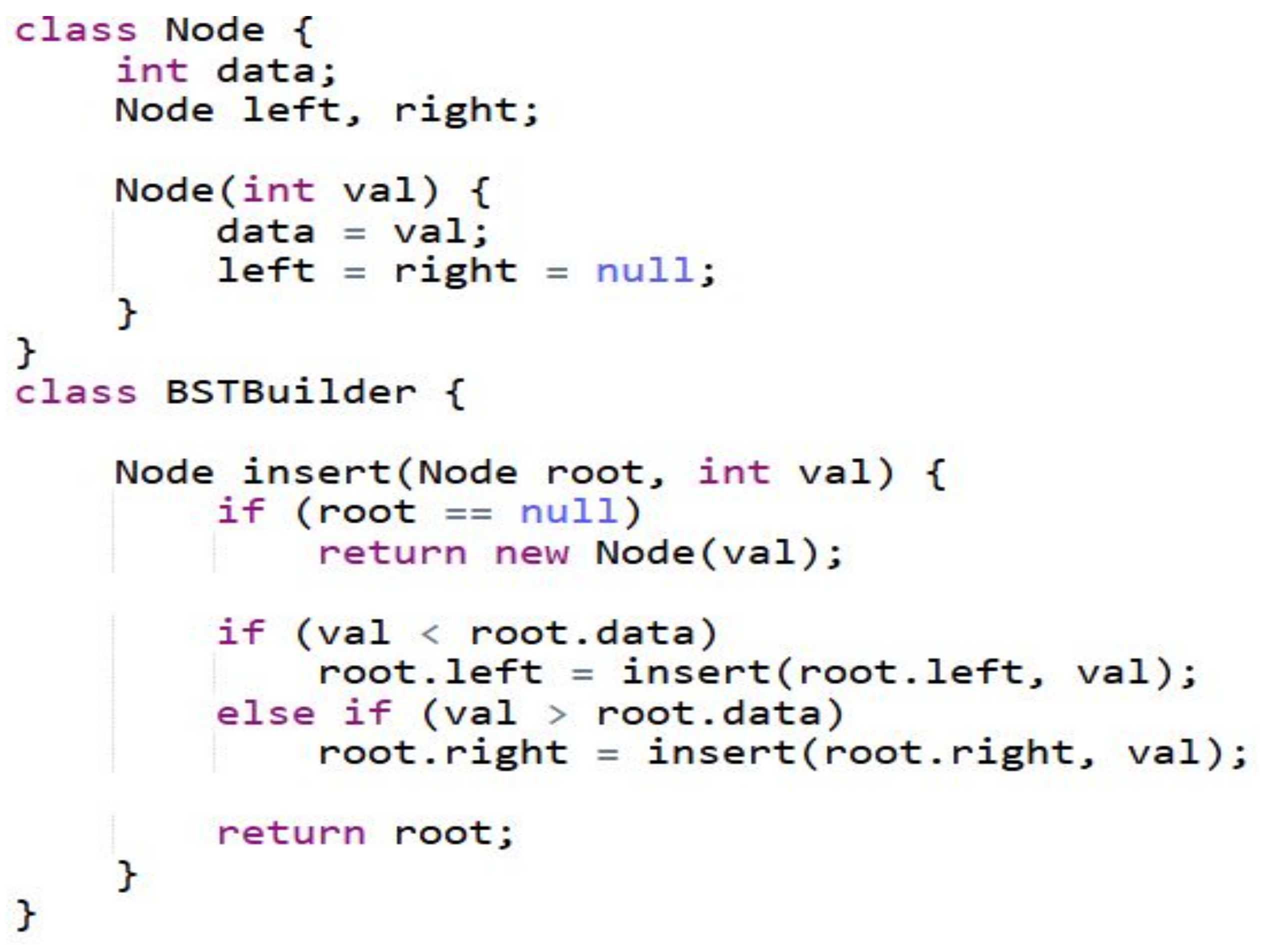
Visualization

Insert sequence: 8, 3, 10, 1,6, 9, 14

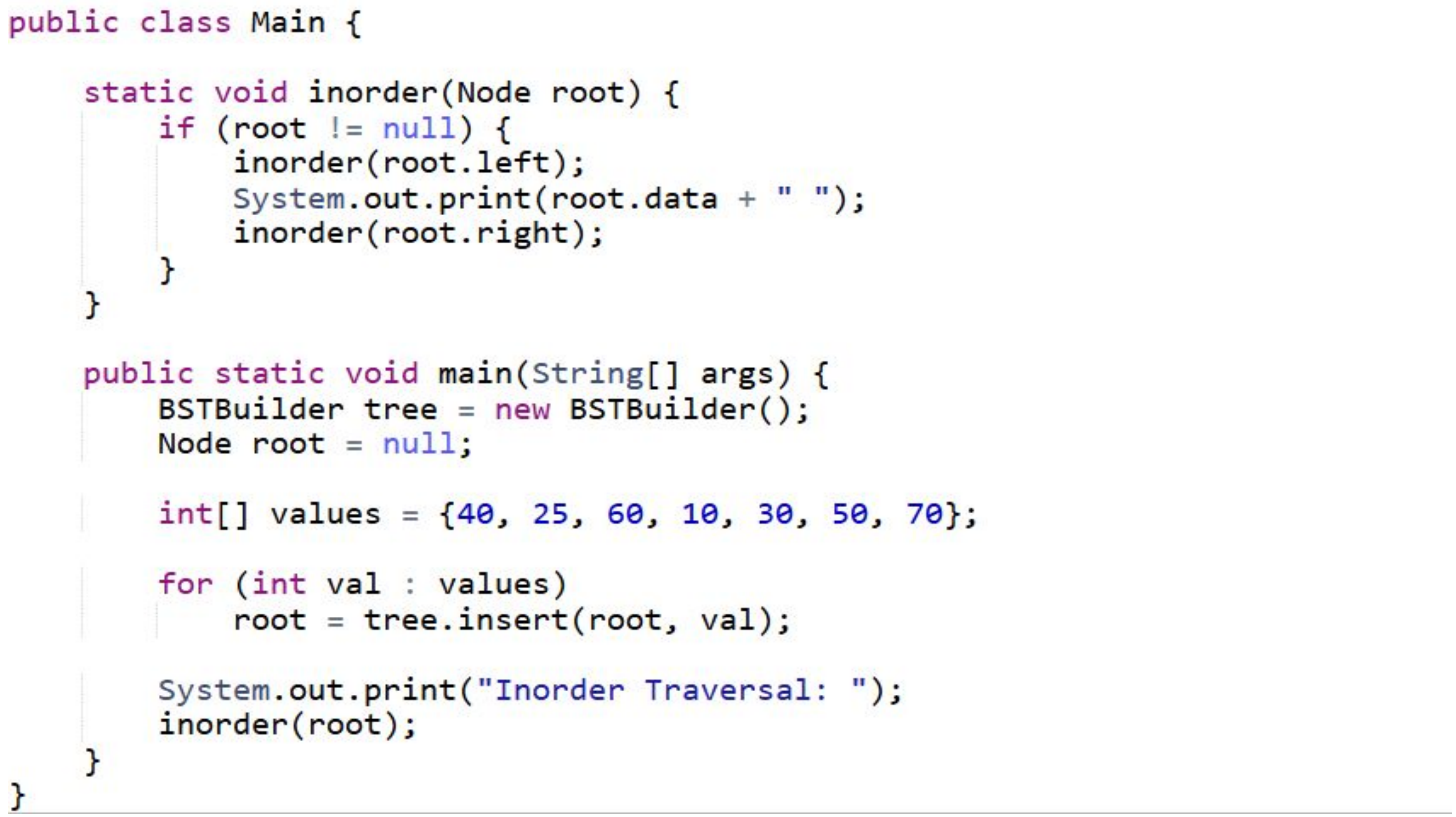


Inorder Traversal: 1,3,6,8,9,10, 14

Code Implementation

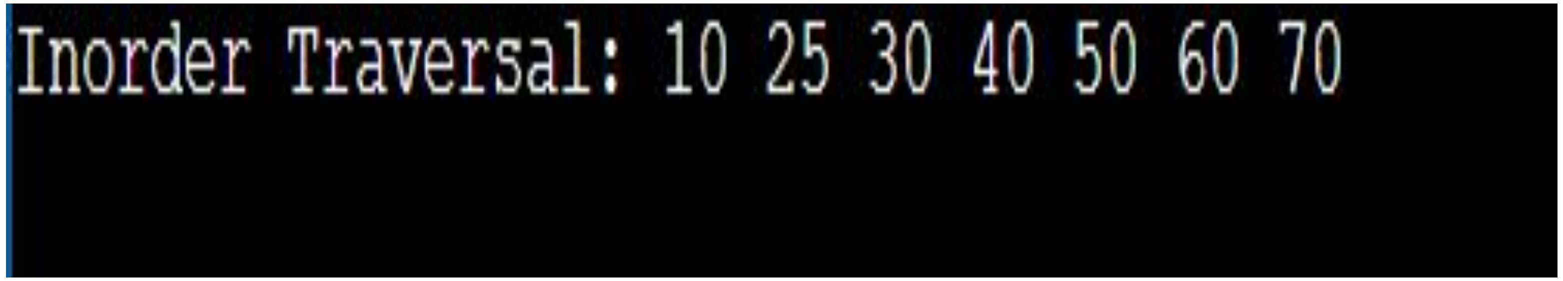


```java
class Node {
    int data;
    Node left, right;

    Node(int val) {
        data = val;
        left = right = null;
    }
}
class BSTBuilder {

    Node insert(Node root, int val) {
        if (root == null)
            return new Node(val);

        if (val < root.data)
            root.left = insert(root.left, val);
        else if (val > root.data)
            root.right = insert(root.right, val);

        return root;
    }
}
```



```java
public class Main {

    static void inorder(Node root) {
        if (root != null) {
            inorder(root.left);
            System.out.print(root.data + " ");
            inorder(root.right);
        }
    }

    public static void main(String[] args) {
        BSTBuilder tree = new BSTBuilder();
        Node root = null;

        int[] values = {40, 25, 60, 10, 30, 50, 70};

        for (int val : values)
            root = tree.insert(root, val);

        System.out.print("Inorder Traversal: ");
        inorder(root);
    }
}
```

Output



```
Inorder Traversal: 10 25 30 40 50 60 70
```

Example Walkthrough

1. Insert 40 → becomes root.
2. Insert 25 → goes left of 40.
3. Insert 60 → goes right of 40.
4. Insert 10, 30, 50, 70 → placed recursively.
5. Final structure matches BST rules.

Time & Space Complexity

Time Complexity:

O(h) per insertion → h = height of tree.

O(n log n) average case for n elements.

Space Complexity:

O(h) due to recursion stack.

Summary

1. BST maintains ordered data using left and right subtrees.
2. Insertion follows comparison-based positioning.
3. Inorder traversal displays sorted elements.
4. Average complexity: O(log n), Worst case: O(n).

Practice Questions:

1. Insert into a Binary Search Tree— LeetCode #701

🔗 https://leetcode.com/problems/insert-into-a-binary-search-tree/

Concept: Build a BST by inserting given nodes.

Why Practice: Strengthens understanding of insertion logic.

Practice Questions:

2. Search in a Binary Search Tree— LeetCode #700

🔗 https://leetcode.com/problems/search-in-a-binary-search-tree/

Concept: Find an element in an existing BST.

Why Practice: Reinforces traversal and search in BST.

Practice Questions:

3. Construct BST from Preorder Traversal — LeetCode #1008

🔗 https://leetcode.com/problems/construct-binary-search-tree-from-preorder-traversal/

Concept: Rebuild BST using preorder sequence with recursion.

Why Practice: Improves understanding of BST structure and traversal.

Thanks