📘 Binary Search Tree (BST) – Recursive vs Iterative

📌 Introduction

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

- Left child < Parent
- Right child > Parent
- No duplicates (in standard BST).

Searching in BST can be implemented in two ways:

1. **Recursive Approach** – Uses function calls.
2. **Iterative Approach** – Uses loops.

Both methods have the same time complexity but differ in space usage and performance.

⚡ Comparison Table

Aspect	Recursive BST Search	Iterative BST Search
Code Simplicity	Very simple and elegant. Shorter code.	Slightly longer code, less elegant.
Time Complexity (Best)	O(1) (if key is at root)	O(1) (if key is at root)
Time Complexity (Average)	O(log n) (balanced BST)	O(log n) (balanced BST)
Time Complexity (Worst)	O(n) (skewed BST)	O(n) (skewed BST)
Space Complexity	O(h), where h = tree height (stack frames).	O(1), no extra memory needed.
Overhead	Function call overhead due to recursion.	No overhead, uses loop.
Risk	Stack overflow possible in skewed trees.	No stack overflow risk.
Performance	Slower in practice (due to recursion overhead).	Faster in practice (no recursion overhead).
Ease of Understanding	Easier for learning and small problems.	Preferred in large-scale applications.

🛠️ Programs Implemented

Recursive BST

- Insert nodes recursively.
- Search for elements using recursion.
- Returns a pointer to the node if found, otherwise NULL.

Iterative BST

- Insert nodes recursively (for fairness in comparison).
- Search for elements using a while loop.
- Returns a pointer to the node if found, otherwise NULL.

📊 Final Conclusion

- Both **Recursive** and **Iterative BST search** have **O(log n)** average time complexity.
- **Iterative search is more efficient** in terms of memory and execution speed.
- **Recursive search is easier to implement** and better for learning purposes.
