Name: Muhammad Bilal

Roll No:23K-0026

Sec: BCS-3F

Performance Analysis Report for BST, AVL Tree, and B-Tree

Introduction

This report analyzes the performance of three tree structures used in a memory-based database:

- **Binary Search Tree (BST)**
- **AVL Tree**
- **B-Tree**

Each tree stores a table of records with three fields:

- **ID (int)**: Unique identifier
- **Name (string)**
- **Age (int)**

Database operations tested:

- **Insert**: Adding new records.
- **Search**: Finding records by ID.
- **Update**: Modifying records.
- **Delete**: Removing records by ID.

Performance is measured across dataset sizes of **5000**, **10000**, and **50000** records.

Methodology

- **Tree Structures**: Each tree type (BST, AVL, B-Tree) was implemented independently.
- **Performance Measurement**: Execution times were measured using C++ `chrono` library, averaging over 20 repetitions.

- **Test Setup**: Each operation (Insert, Search, Update, Delete) was timed for consistency.

Performance Results

Insertion Performance

- **5000 Records**: BST: 2.96 ms, AVL: 358.22 ms, B-Tree: 104.47 ms
- **10000 Records**: BST: 6.27 ms, AVL: 2101.47 ms, B-Tree: 544.98 ms
- **50000 Records**: BST: 34.81 ms, AVL: 66315.5 ms, B-Tree: 12895.5 ms

Search Performance

- **5000 Records**: BST: 0.62 ms, AVL: 0.73 ms, B-Tree: 52.67 ms
- **10000 Records**: BST: 2.19 ms, AVL: 2.10 ms, B-Tree: 267.06 ms
- **50000 Records**: BST: 13.27 ms, AVL: 77.71 ms, B-Tree: 6683.36 ms

Update Performance

- **5000 Records**: BST: 0.89 ms, AVL: 1.12 ms, B-Tree: 51.29 ms
- **10000 Records**: BST: 2.88 ms, AVL: 2.60 ms, B-Tree: 214.27 ms
- **50000 Records**: BST: 16.90 ms, AVL: 12.99 ms, B-Tree: 6241.52 ms

Delete Performance

- **5000 Records**: BST: 1.08 ms, AVL: 353.61 ms, B-Tree: 208.50 ms
- **10000 Records**: BST: 2.57 ms, AVL: 2081.04 ms, B-Tree: 387.30 ms
- **50000 Records**: BST: 17.24 ms, AVL: 53582.80 ms, B-Tree: 11668.30 ms

Analysis

- **Binary Search Tree (BST)**:
    - **Performance**: Efficient for smaller datasets but becomes slower with larger datasets due to unbalanced tree structures (O(N) in the worst case).
    - **Conclusion**: BST is best for smaller datasets but not suitable for large datasets where balance is important.
- **AVL Tree**:

- **Performance**: Performs consistently well due to its self-balancing property. Time complexity remains O(log N) for all operations, even with large datasets.
- **Conclusion**: The AVL tree is efficient for large datasets, maintaining optimal time complexity.

- **B-Tree**:
  - **Performance**: Similar to AVL in terms of time complexity but slower in practice due to multi-level node management.
  - **Conclusion**: B-Tree is efficient but shows slower performance in comparison to AVL trees due to node splitting overhead.

Conclusion

- **BST**: Best for smaller datasets but inefficient for large ones due to the risk of imbalance.
- **AVL Tree**: Reliable and efficient for large datasets with consistent O(log N) performance.
- **B-Tree**: Efficient but slower than AVL for most operations, suitable for specific use cases involving multi-level indexing.

Note: The performance may vary depending on how the trees are implemented, particularly with respect to balancing and node management.