# Project Structure

## Files:

1. bst.h: Binary Search Tree Implementation
2. avl.h: AVL Tree Implementation
3. btree.h: B-Tree Implementation
4. main.cpp: Performance Analysis and Demonstration
5. record.h: Record struct

## Binary Search Tree (BST)

Key Functions:

1. insert(const Record& record)

- Inserts a new record into the BST

- Maintains BST ordering property

- Time Complexity: O(h), where h is tree height

2. search(int id)

- Searches for a record by ID

- Returns pointer to record if found, nullptr otherwise

- Time Complexity: O(h)

3. remove(int id)

- Removes a record by ID

- Handles different removal scenarios

- Time Complexity: O(h)

## AVL Tree

Key Functions:

1. insert(const Record& record)

- Inserts record while maintaining tree balance

- Performs rotations to keep tree height balanced

- Time Complexity: O(log n)

2. search(int id)

- Searches for record efficiently

- Guaranteed logarithmic search time

- Time Complexity: O(log n)

3. remove(int id)

- Removes record and rebalances tree

- Maintains AVL tree balance properties

- Time Complexity: O(log n)

## B-Tree

Key Functions:

1. insert(const Record& record)

- Inserts record maintaining B-Tree properties

- Handles node splitting dynamically

- Time Complexity: O(log n)

2. search(int id)

- Efficiently searches across multiple levels

- Optimized for disk-based storage scenarios

- Time Complexity: O(log n)

3. remove(int id)

- Removes record while maintaining B-Tree structure

- More complex deletion logic

- Time Complexity: O(log n)

## Main:

1. generateDummyData(int size)

- Creates synthetic record datasets

- Generates records with unique IDs

- Supports variable dataset sizes

2. measurePerformance()

Measures and reports:

- Insert operation time

- Search operation time

- Delete operation time

- Number of successful searches