Project Description

Title: Binary Search Tree (BST) Implementation with Range Queries

Overview:
This project implements a generic Binary Search Tree (BST) data structure that stores key–value pairs in order. The design follows the recursive approach and includes several operations to maintain and query the tree. Key features include:

- **Insertion & Search:**
  The tree supports inserting new key–value pairs (or updating existing keys) with the `put` method, and retrieving values using the `get` method. The `contains` method checks for a key's existence.

- **Deletion:**
  Methods such as `delete`, `deleteMin`, and `deleteMax` remove keys from the tree. The deletion procedures update subtree sizes to maintain accurate counts.

- **Order Operations:**
  The BST provides methods for finding the minimum (`min`) and maximum (`max`) keys, the largest key less than or equal to a given key (`floor` / `floor2`), and the smallest key greater than or equal to a given key (`ceiling`). In addition, the tree supports order-statistic operations such as `select` (to get the key of a given rank) and `rank` (to determine how many keys are less than a given key).

- **Traversal & Range Queries:**
  You can iterate over all keys (or keys in a specific range) in sorted order via the `keys` method, and perform level-order traversal using `level_order`.

- **Integrity Checks:**
  (Optionally) the code may include internal consistency checks (e.g., BST property, subtree size consistency, and rank consistency).

- **Testing:**
  The project includes a main testing section that uses simple `if` / `else` conditions to validate each public method (insertion, deletion, search, ordering, and traversal) and prints the results.