◊ Here we will break down our request for the Binary Search Tree (BST), Red-Black Tree (RBT), and AVL Tree. We'll go step by step for each of these, starting with their basic structure and then implementing the operations like adding, querying, and deleting. I'll implement each tree from scratch and explain each step along the way.

1. Part 1: Binary Search Tree (BST)
   - ⇨ A Binary Search Tree (BST) is a tree data structure where each node has at most two children, referred to as the left and right child. The left child contains a value smaller than the parent node, and the right child contains a value greater than the parent node.
   - ⇨ Node structure

   Each node in the BST will contain:

   - o value: The integer value stored in the node.
   - o left: A reference to the left child (subtree).
   - o right: A reference to the right child (subtree).

⇨ 1.2. Operations

We need to implement the following operations:

- Insert: Insert a new value into the tree while maintaining the BST property.
- Search: Check if a value exists in the tree.
- Delete: Remove a node and re-adjust the tree to maintain the BST property.
- In-order traversal: Traverse the tree in order

---

2. Part 2: Red-Black Tree (RBT)

A Red-Black Tree is a balanced binary search tree where each node has an extra bit for storing the color (either red or black). It satisfies certain properties that help keep the tree balanced, such as:

- Every node is either red or black.
- The root is always black.
- Red nodes cannot have red children (i.e., no two red nodes can be adjacent).
- Every path from a node to its descendant leaves has the same number of black nodes.

Red-Black trees are more complicated than BSTs because they require additional operations for balancing the tree (rotations and color changes).

We will implement the Red-Black Tree insertions (without deletions for simplicity).

⇨ 2.1. Node structure

Each node will have:

- value: Integer value.
- color: Color of the node (Red or Black).
- left, right: Left and right child nodes.
- parent: Parent of the node.

---

3. Part 3: AVL Tree

⇨ An **AVL Tree** is a self-balancing binary search tree, where the difference between the heights of the left and right subtrees cannot be more than one for all nodes. We need to implement insertions, deletions, and rotations to maintain balance.