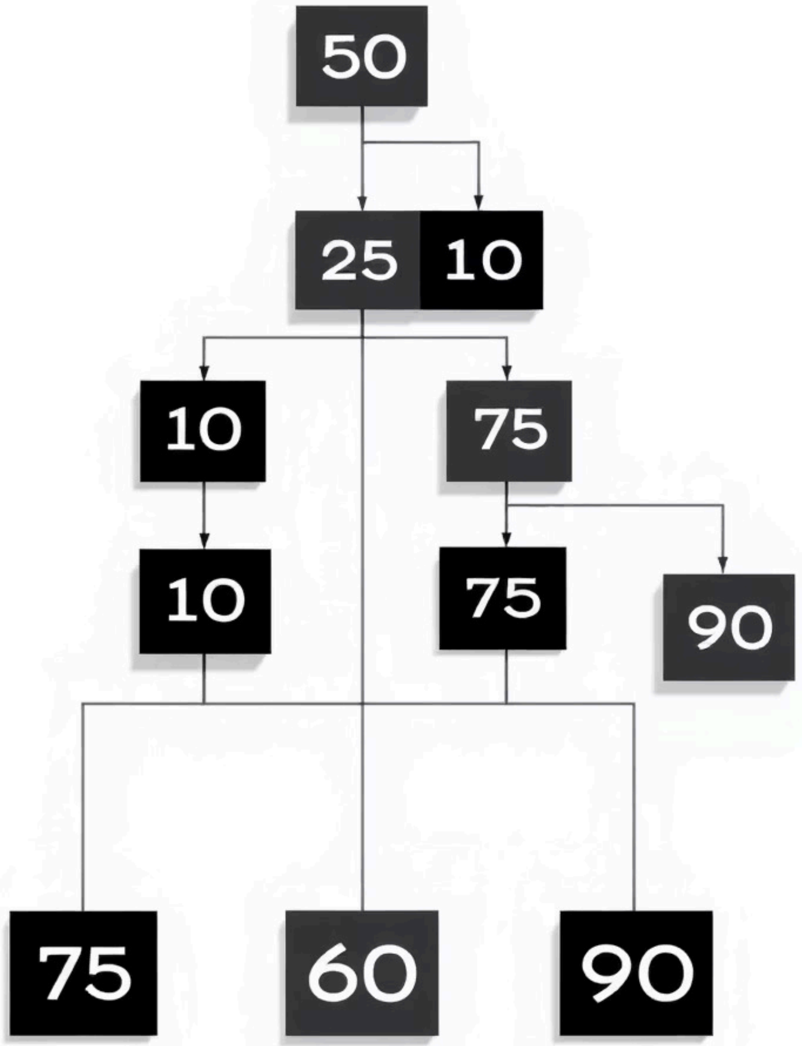


Introduction to Binary Search Trees (BSTs)

A BST is a tree where left nodes have values less than or equal to the parent, and right nodes have greater values.

Operations like search, insertion, and deletion average O(log n) efficiency.

BSTs play a key role in efficient data storage and retrieval.

The Problem: Degenerate BSTs

What is a degenerate BST?

A skewed tree where nodes form a linear structure, like a linked list.

Worst-case complexities

Search, insertion, and deletion degrade to O(n) time complexity.

Common cause

Inserting sorted data (e.g., 1, 2, 3, 4, 5) creates skewed trees.

Height-Balanced BSTs: The Solution

Definition

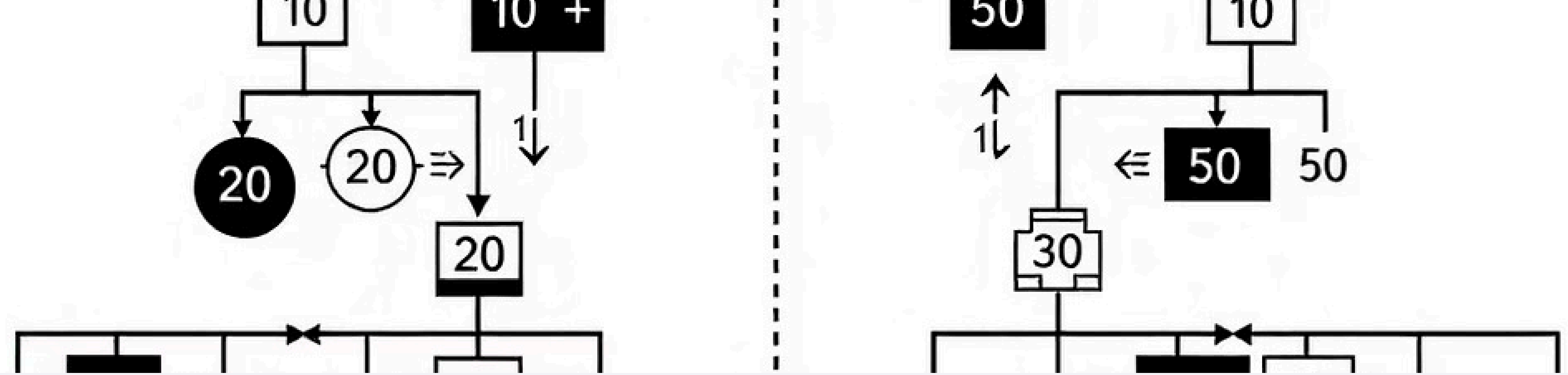
A BST that ensures subtree heights differ minimally to stay balanced.

Why balance?

Maintains O(log n) time complexity for operations by minimizing tree height.

Goal

Achieve efficient search, insertion, and deletion via height minimization.



AVL Trees: Definition and Properties

Origins

Invented by Adelson-Velsky and Landis in 1962.

Balance factor

Difference in height between left and right subtrees is -1, 0, or 1.

Self-balancing

Automatically performs rotations to maintain balance after operations.

AVL Tree Rotations: Left Rotation

1

When to use

Right subtree is too heavy compared to left.

2

Mechanism

Pivot node shifts left, adjusting children accordingly.

3

Effect

Restores balance and reduces tree height.

AVL Tree Rotations: Right Rotation

1

When to use

Left subtree is too heavy relative to right.

2

Mechanism

Pivot node moves right, rearranging involved subtrees.

3

Outcome

Balances the tree and maintains optimal height.

AVL Tree Rotations: Left-Right Rotation

1

Step 1

Perform left rotation on left child subtree.

2

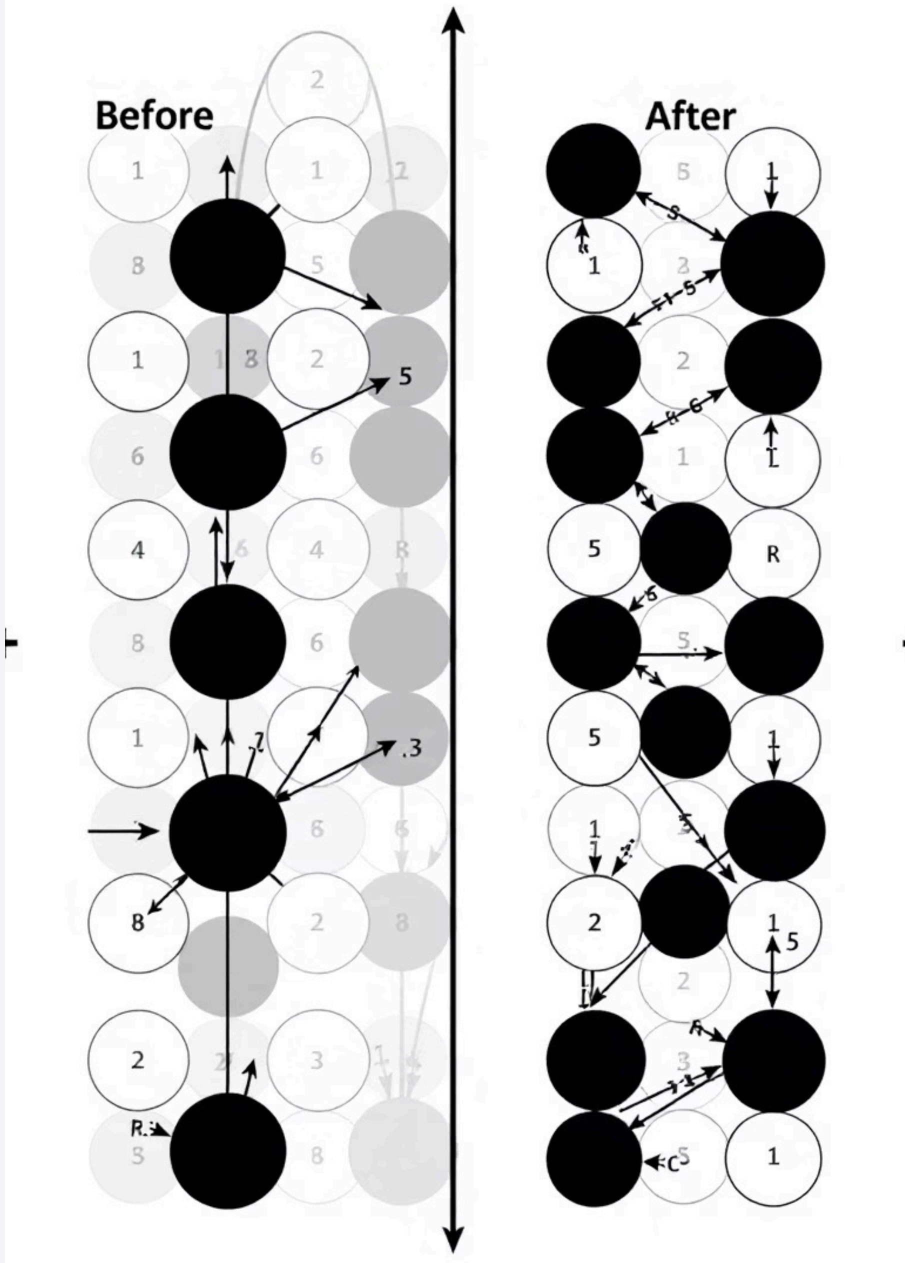
Step 2

Follow with right rotation on the unbalanced node.

3

Purpose

Balances trees with left child's right-heavy subtree.



AVL Tree Rotations: Right-Left Rotation

1

Step 1

Execute right rotation on right child's left subtree.

2

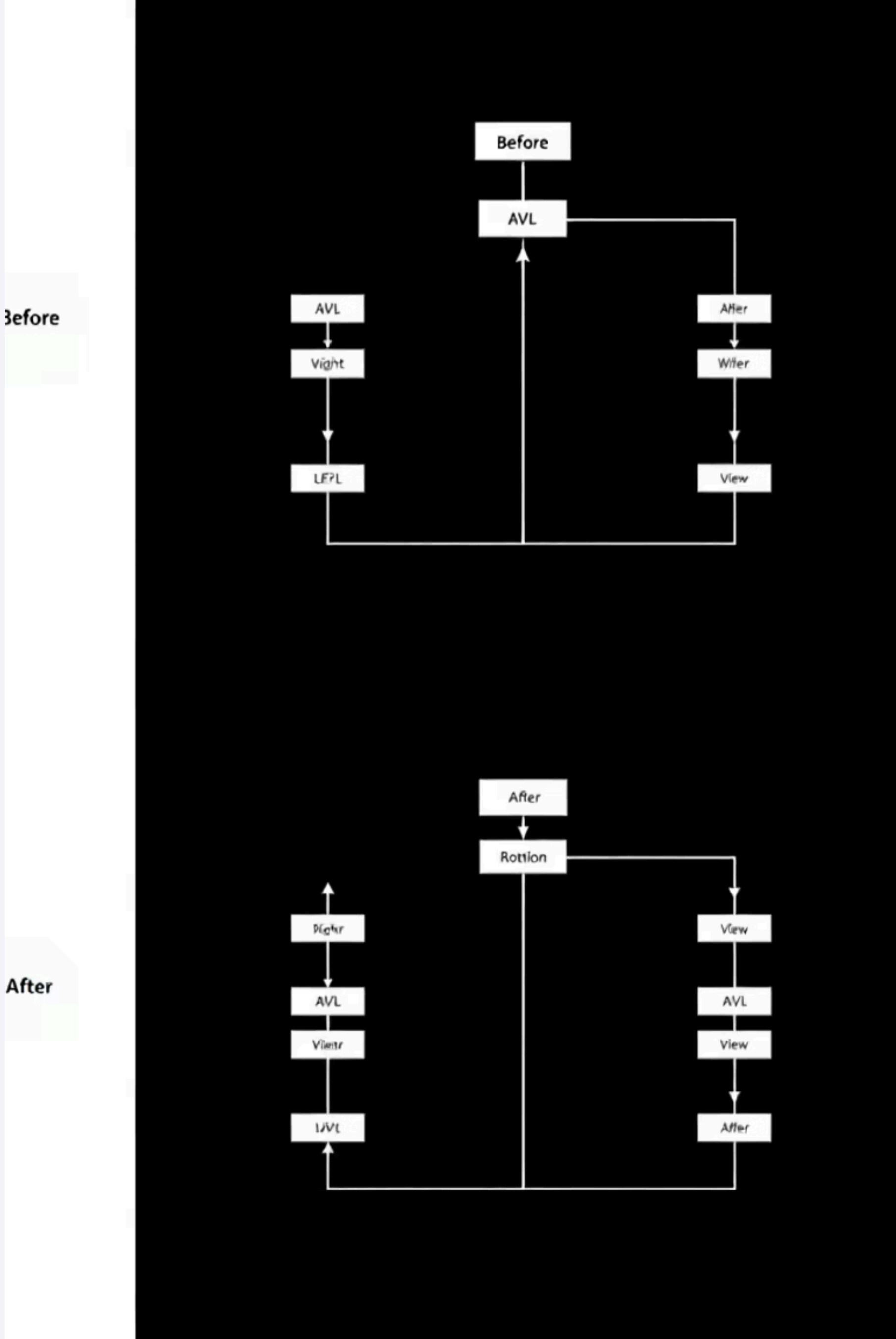
Step 2

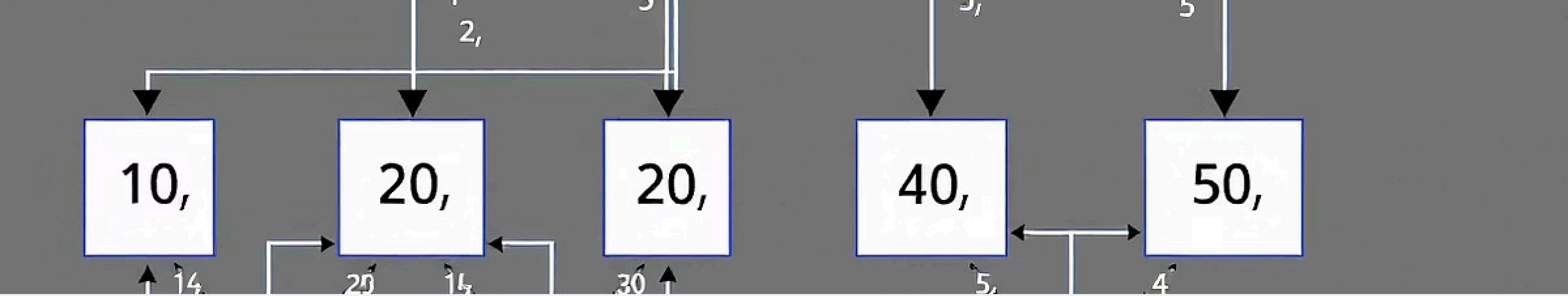
Then perform left rotation on the node causing imbalance.

3

Effectiveness

Fixes imbalance when right child's left subtree is heavy.





Inserting into an AVL Tree: Example

1. **Insert 10, 20, 30**

   Triggers left rotation to maintain balance.

2. **Insert 40, 50**

   Tree continues adjusting to remain balanced.

3. **Insert 5, 4, 3**

   Right rotation triggered to fix left heavy subtree.

Conclusion: AVL Trees Advantages

Performance guarantee

Ensures O(log n) time for search, insert, and delete.

Best for dynamic data

Ideal when insertions and deletions happen frequently.

Trade-offs

More complex implementation in exchange for speed.

Applications

Widely used in databases, indexing, and memory management.