Chapter 8: Search Trees (Part 2)

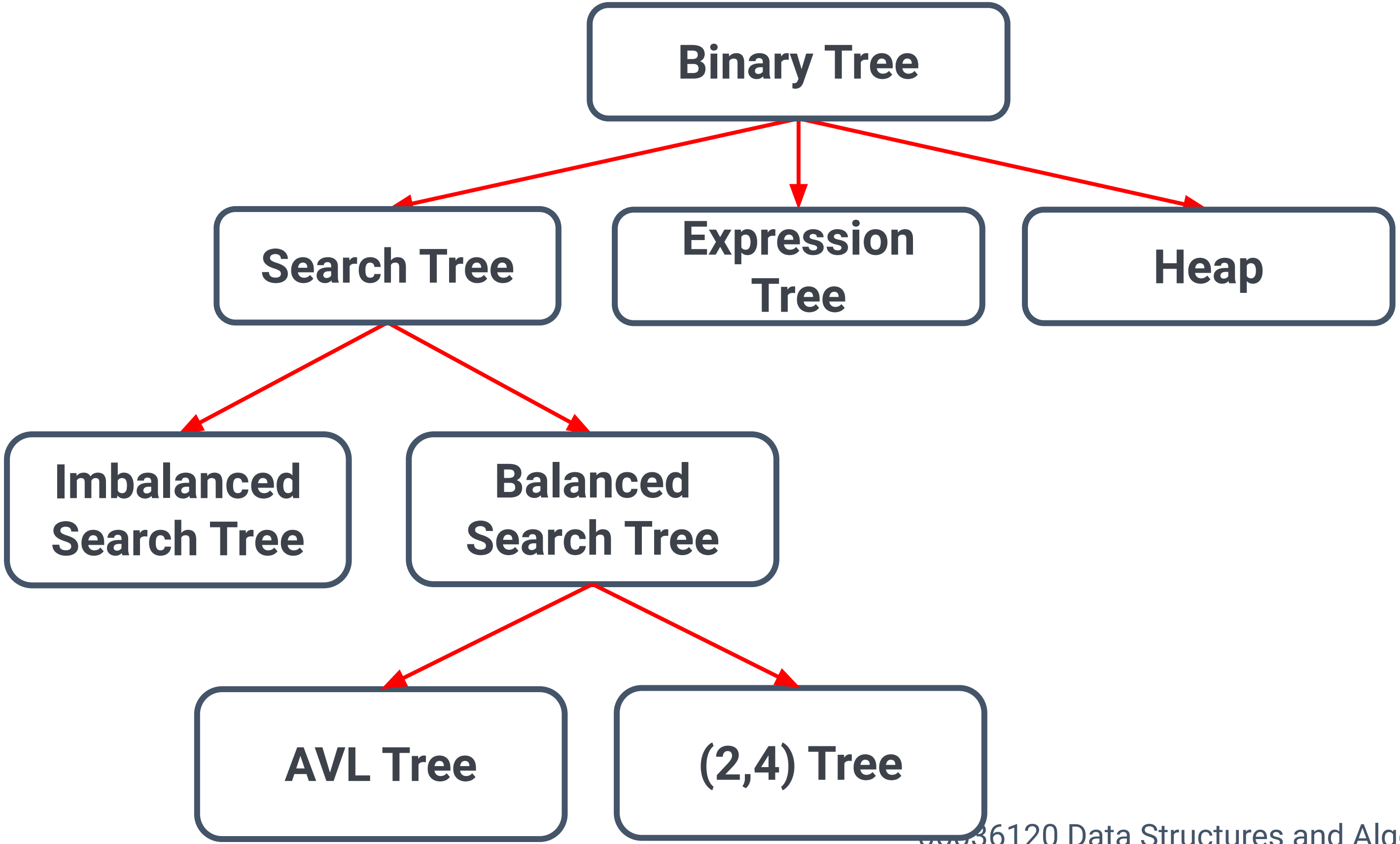
Dr. Sirasit Lochanachit

Outline

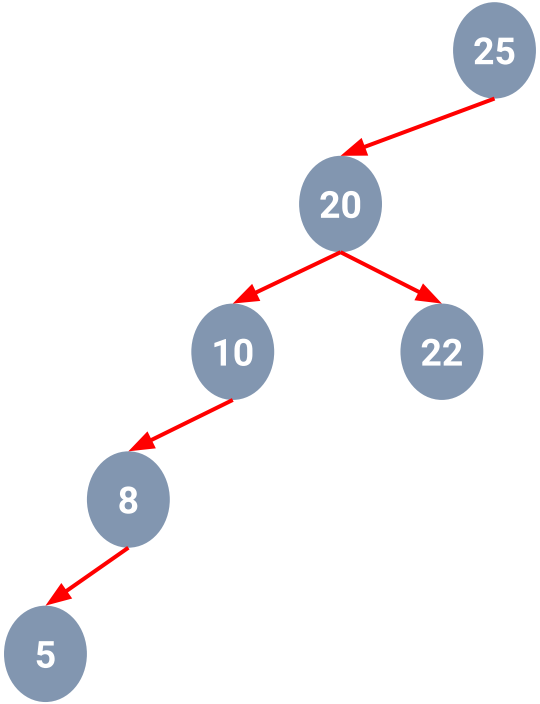
AVL Trees:

- Definition and Balance Factor
- Balancing Algorithms and Operation examples

Types of Binary Trees (Revisited)

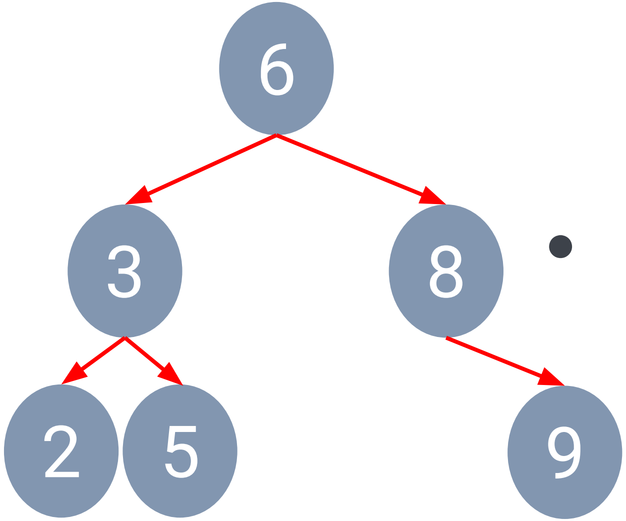


Binary Search Tree (Revisited)



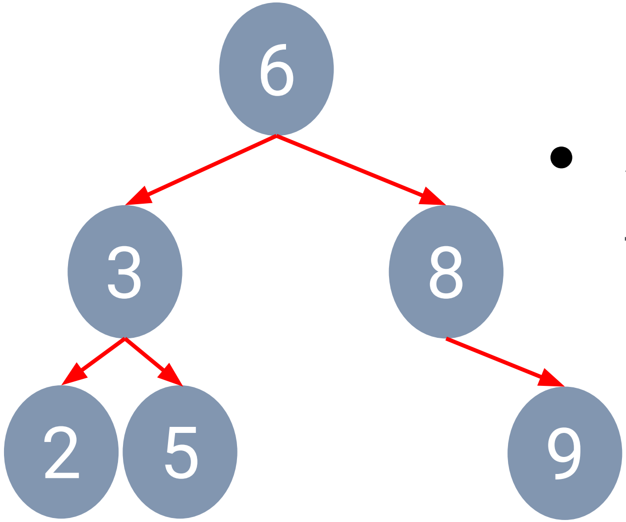
- Running time of <u>inserting node</u> is also proportional to the **height of tree** (i.e. $\log_2 n$ or n) == O(h).
- A **balanced search tree** has the same number of nodes in both left and right subtree.
  - Worst-case performance is $O(\log_2 n)$.
- Inserting keys in sorted order would construct an **imbalanced tree**.
  - Provides poor performance of $O(n)$.

Balanced Binary Search Tree



- A **balanced binary search tree (BST)** maintains the balance through a rotation operation which consequently provides a better performance.
- Several types of binary tree that automatically ensure balance
  - AVL tree
  - Splay tree
  - Red-black tree

AVL Tree

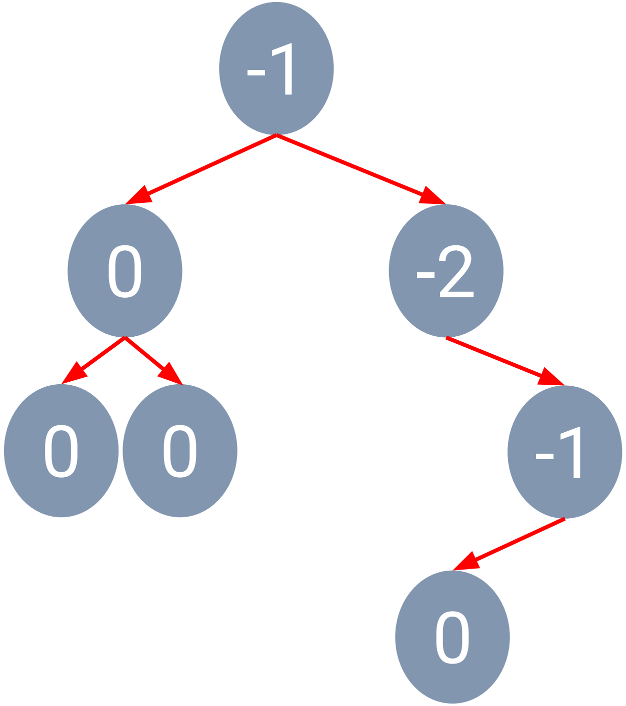


- AVL tree is named after its inventors:

  G.M. **A**delson-**V**elskii and E.M. **L**andis.

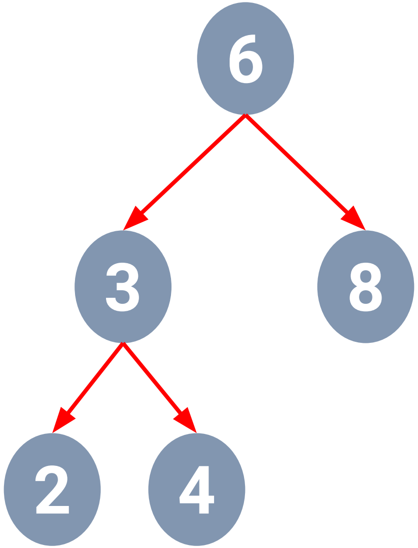
- AVL tree introduces a **balance factor** for each node in the tree.

Balance Factor in AVL Tree

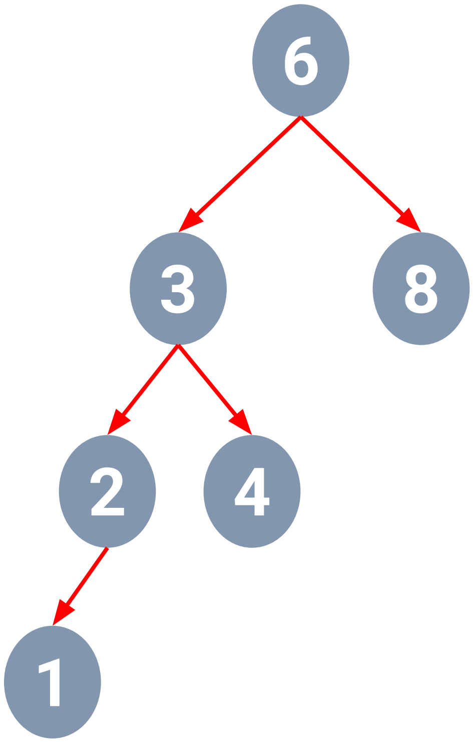


- AVL tree is considered to be balanced when the balance factor is -1, 0, or 1.
  - $|H_{left} - H_{right}| <= 1$
- AVL tree uses **trinode restructuring**, involving reconfigurations of three nodes.

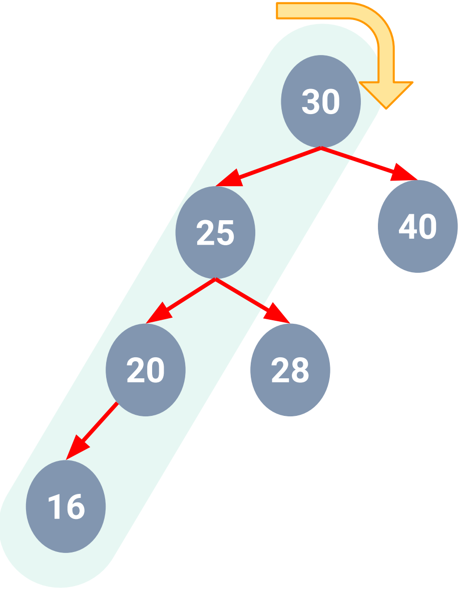
Balance Factor in AVL Tree



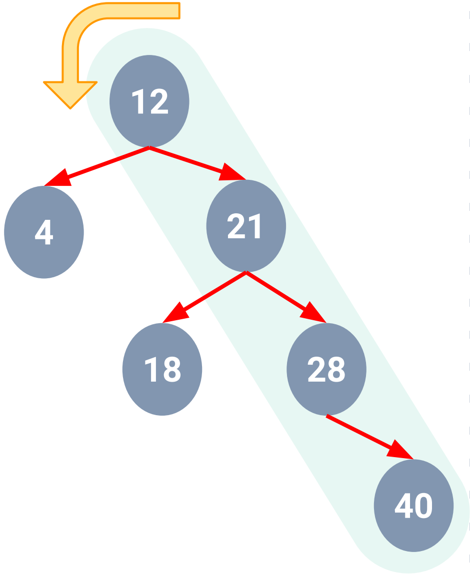
Balance Factor in AVL Tree



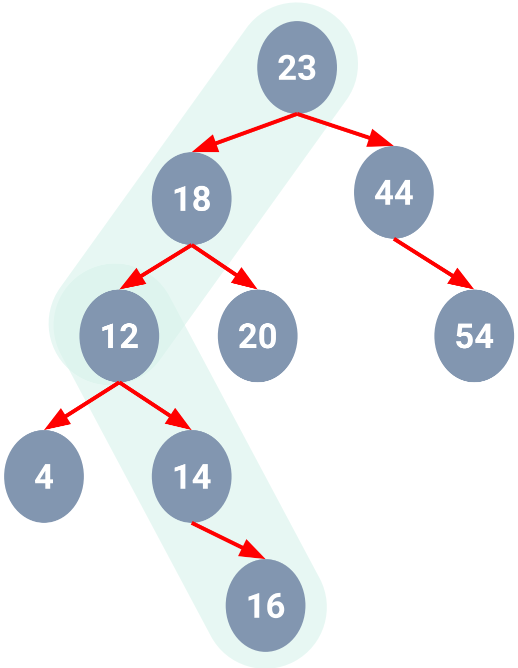
Balancing AVL Tree



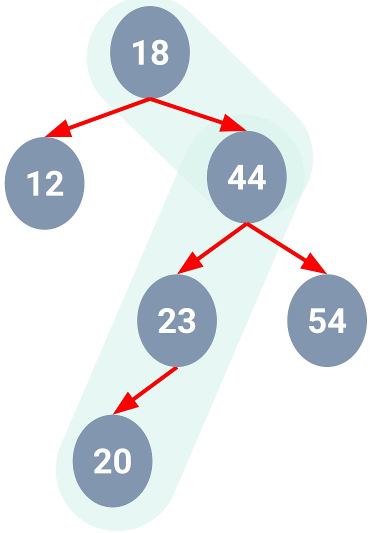
Left of Left



Right of Right

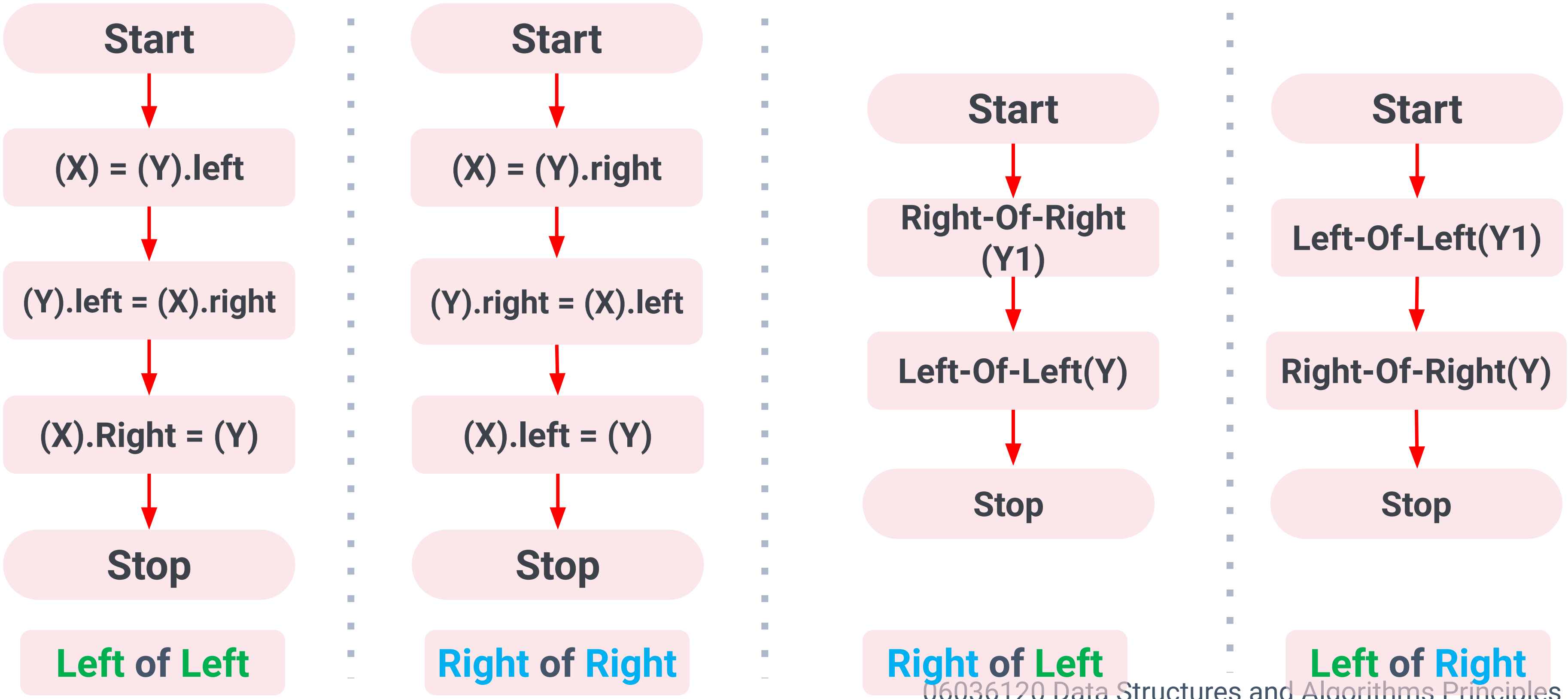


Right of Left

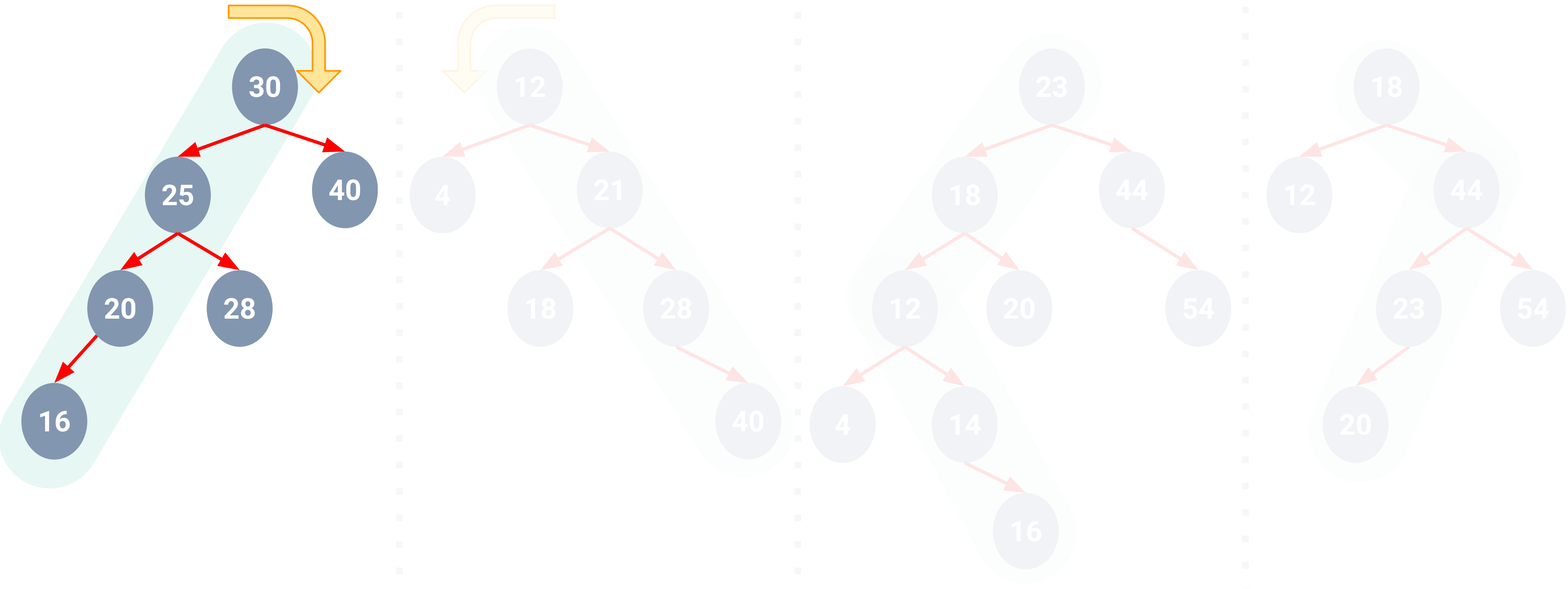


Left of Right

Balancing AVL Tree



Balancing AVL Tree (LoL)



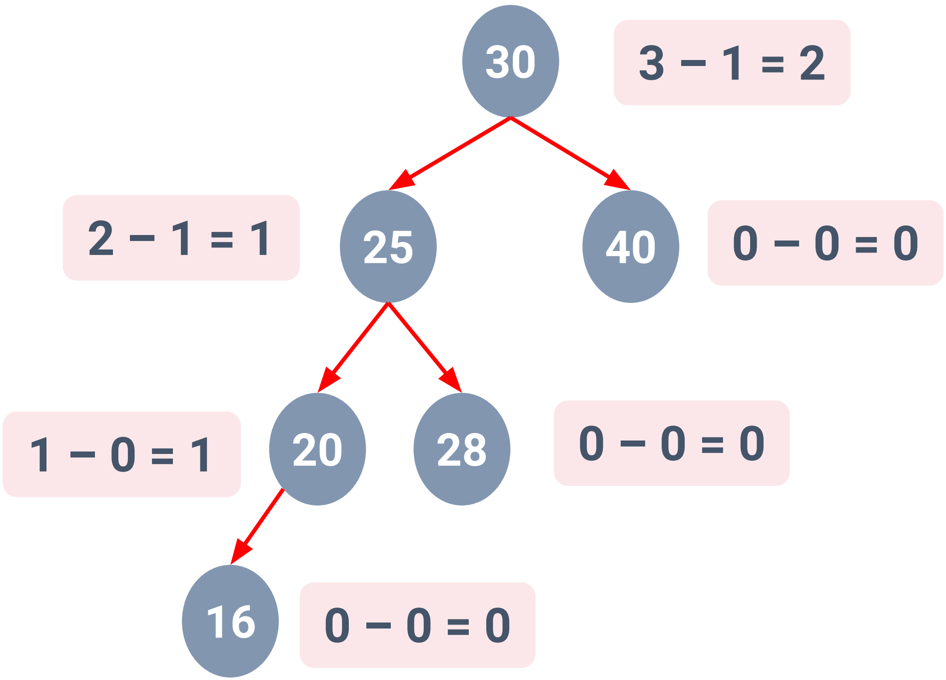
Left of Left

Right of Right

Right of Left

Left of Right

Balancing AVL Tree (LoL)

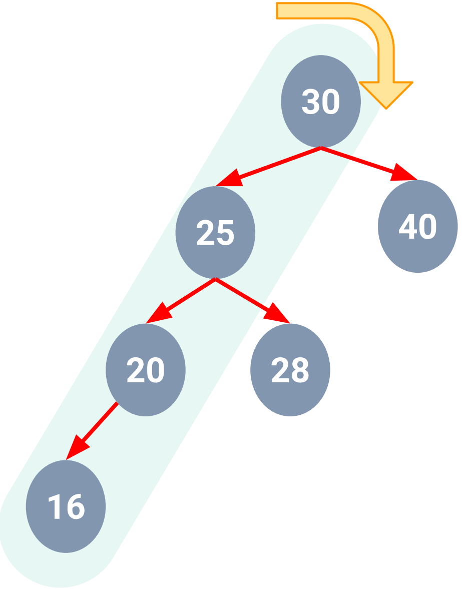


A tree is **left-heavy** with a balance factor of 2 at the root.

Requires a **right rotation**.

Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$

Right Rotation

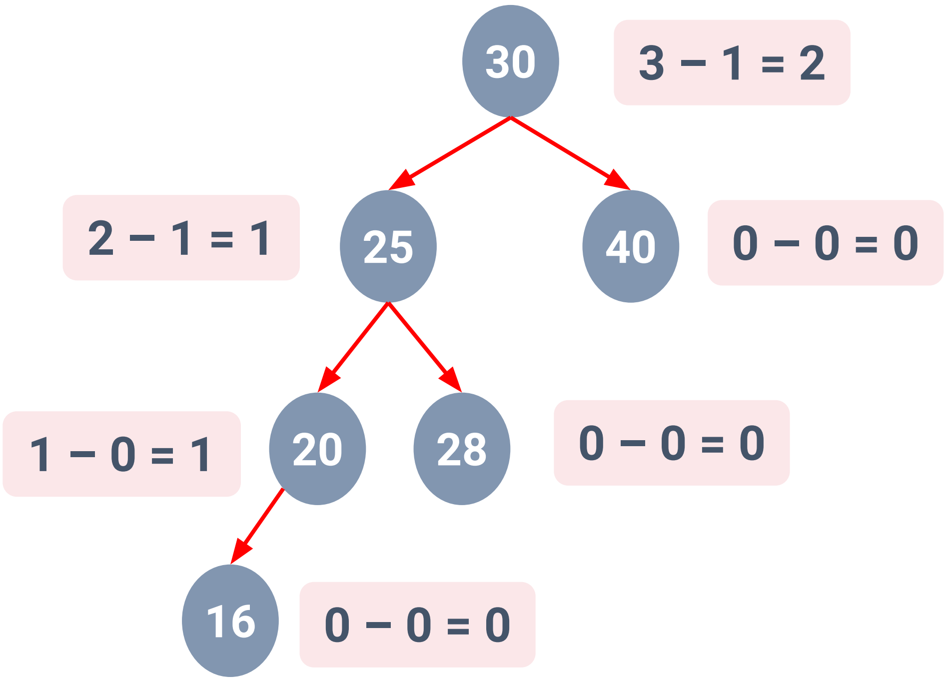


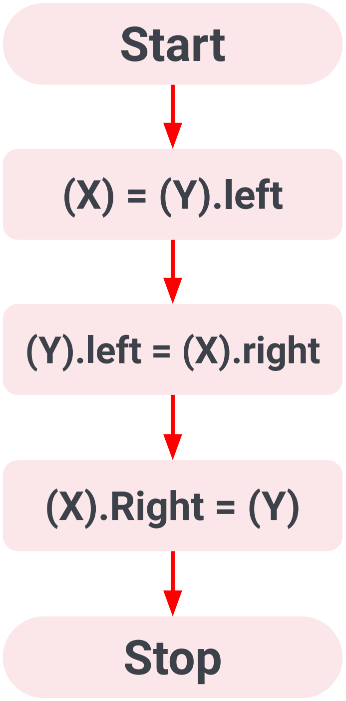
Left of Left

To perform a right rotation (at node 30), do **4 steps** below:

1. Promote the left child to be the root of the subtree.
2. Move the old root to the right as a right child of the new root.
3. If the new root already had a right child,
   - The right child become the left child of the old root.
4. Update parents pointers (if exist).

Balancing AVL Tree (LoL)

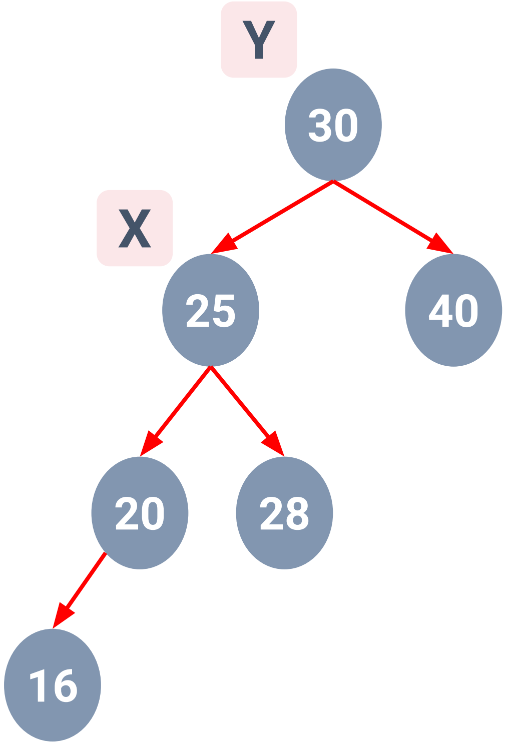




Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$

where (Y) is a node which is a rotated node

Balancing AVL Tree (LoL)



Start

(X) = (Y).left

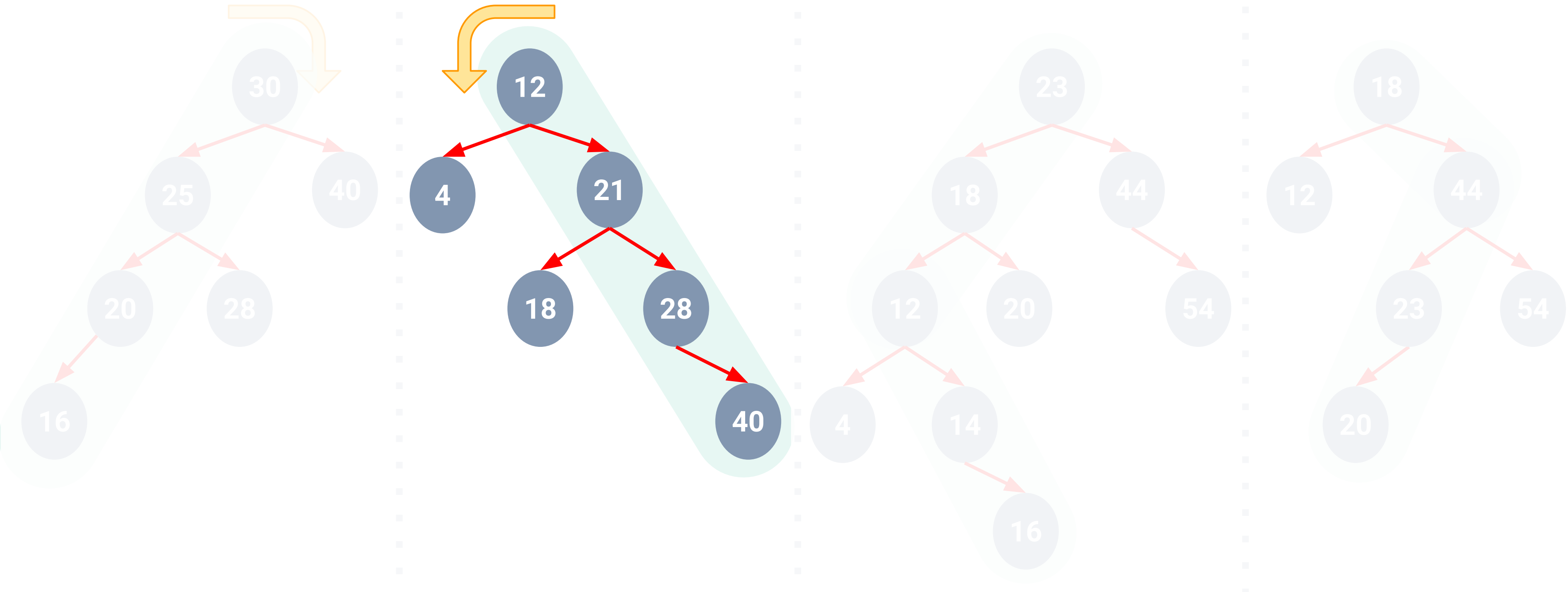
(Y).left = (X).right

(X).Right = (Y)

Stop

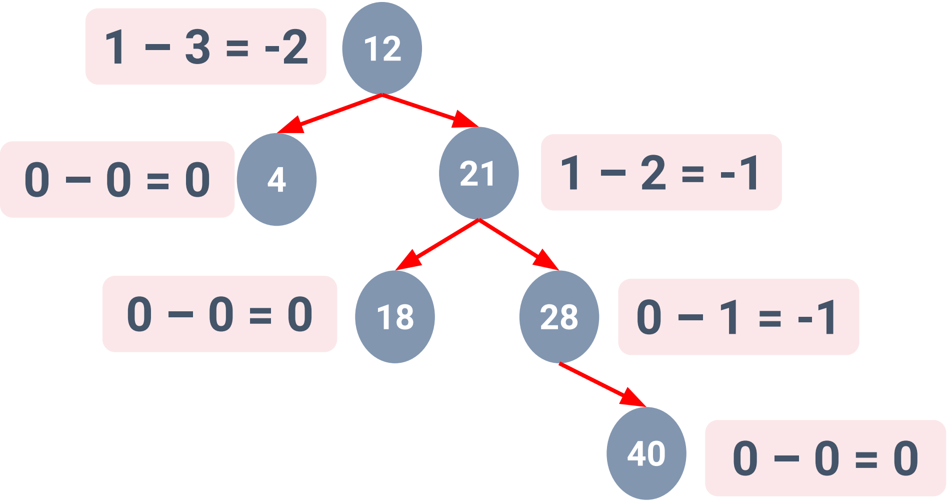
where (Y) is a node which is a rotated node

Balancing AVL Tree



Left of Left | **Right of Right** | Right of Left | Left of Right

Balancing AVL Tree (RoR)

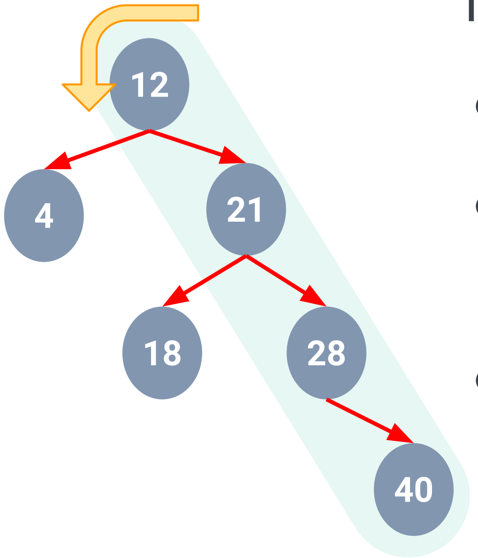


A tree is **right-heavy** with a balance factor of -2 at the root.

Requires a **left rotation**.

Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$

Left Rotation



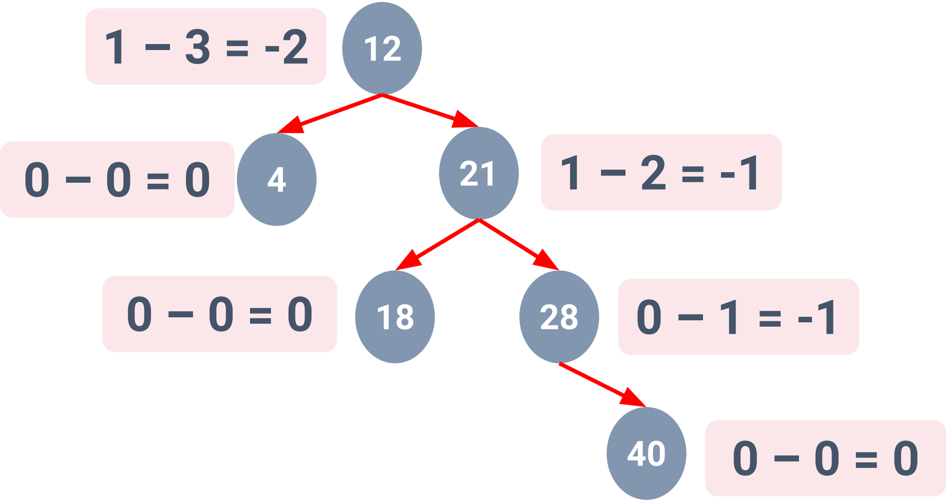
Right of Right

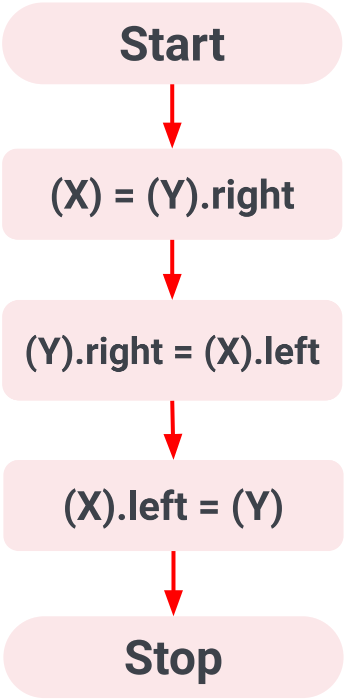
To perform a left rotation (at node 12), do **4 steps** below:

- Promote the right child to be the root of the subtree.
- Move the old root to the left to be the left child of the new root.
- If the new root already had a left child,
  - The left child become the right child of the old root.
- Update parents pointers (if exist).

Faculty of
Information Technology
King Mongkut's Institute of Technology Ladkrabang

Balancing AVL Tree (RoR)

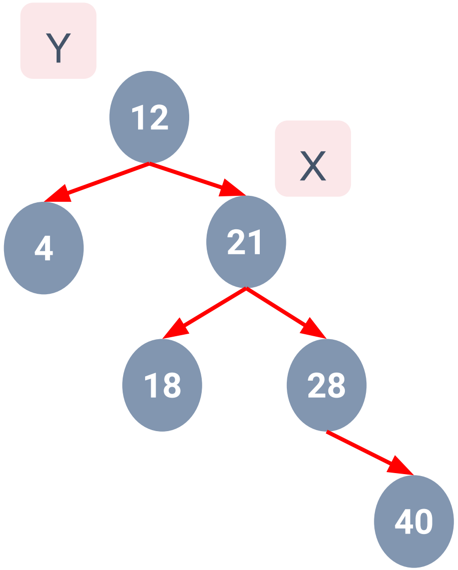




Balance Factor Condition is $|H_{left} - H_{right}| \leq 1$

where (Y) is a node which is a rotated node

Balancing AVL Tree (RoR)



Start

(X) = (Y).right

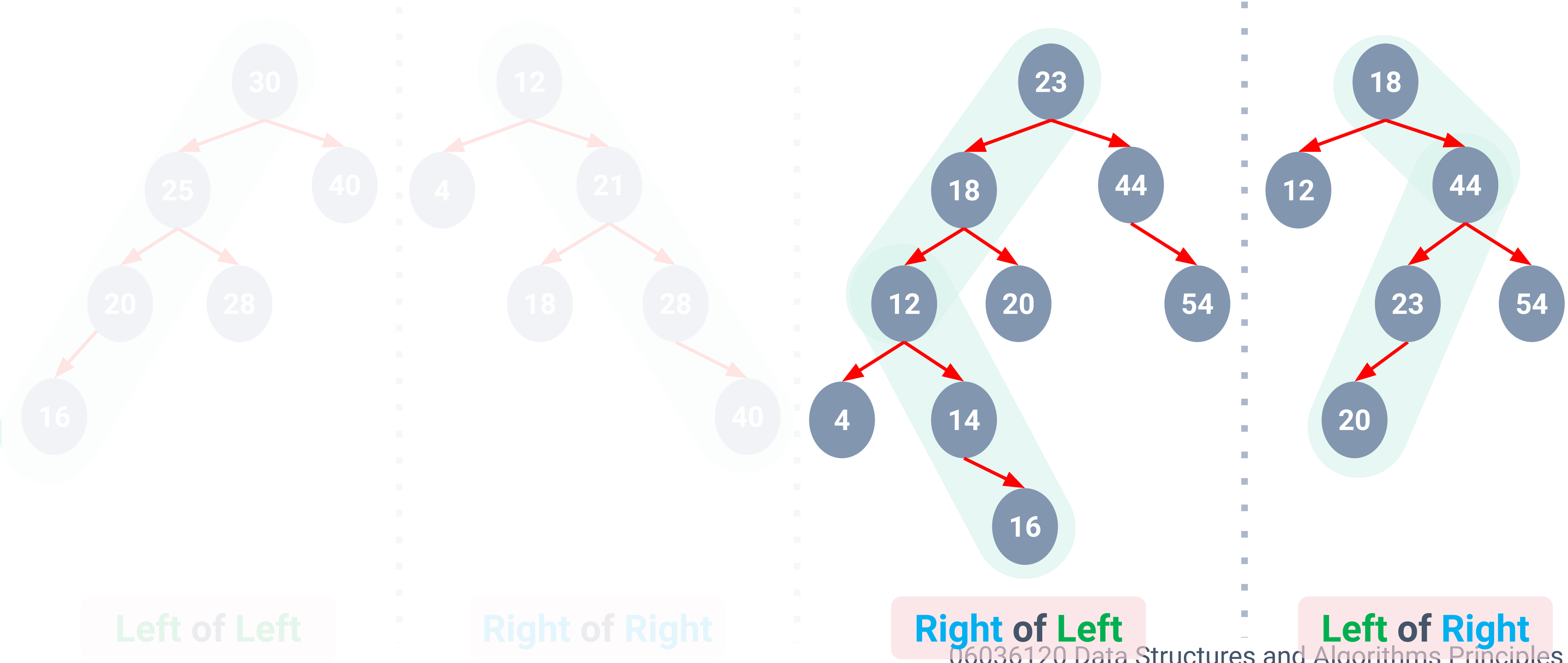
(Y).right = (X).left

(X).left = (Y)

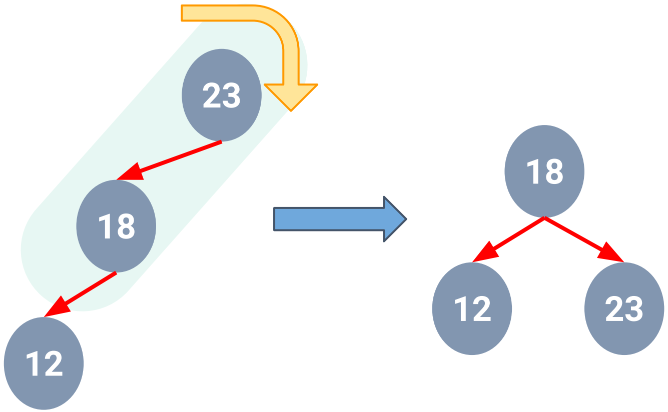
Stop

where (Y) is a node which is a rotated node

Balancing AVL Tree



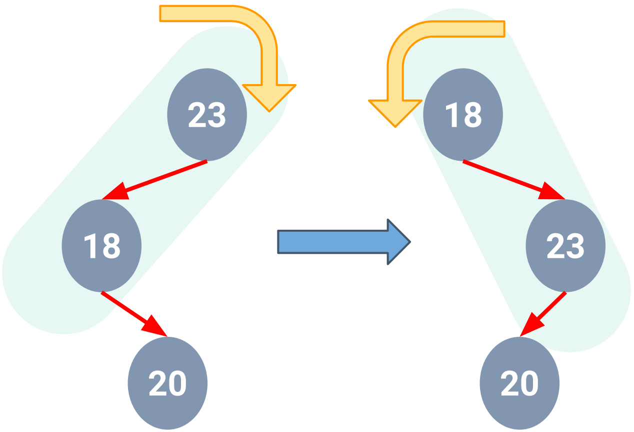
Right Rotation



Left Heavy

Balanced

Right Rotation



Left Heavy

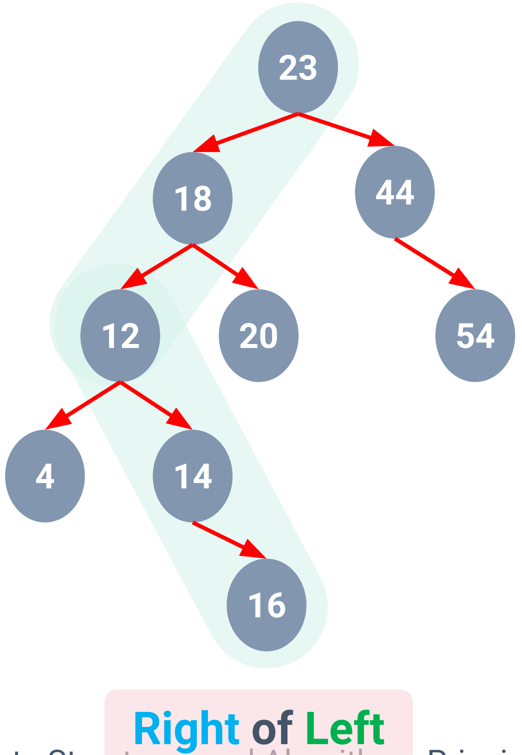
Right Heavy

Right Rotation

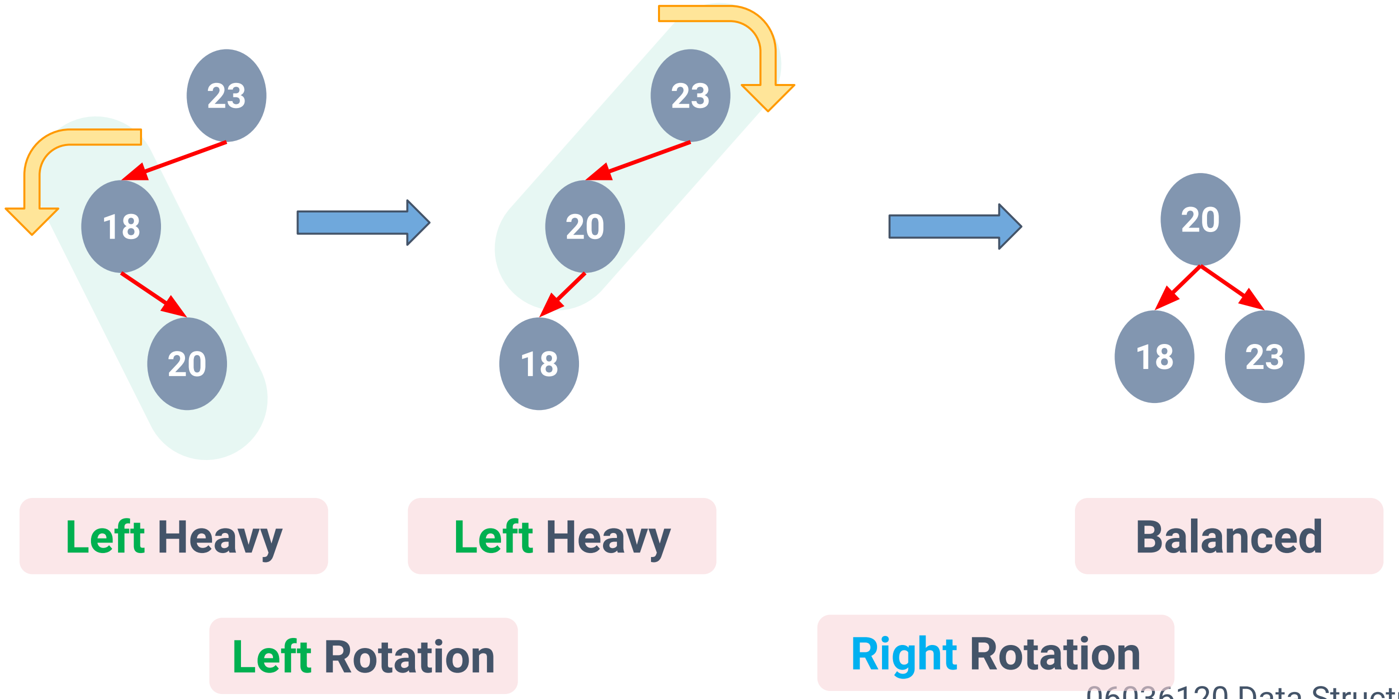
Left then Right Rotation

To solve this problem, there are additional rules:

- If a subtree needs a **right rotation**,
  - Check the balance factor of the left child.
  - If the left child is right-heavy, then do **left rotation on the left child**.
  - Then do right rotation on the subtree.



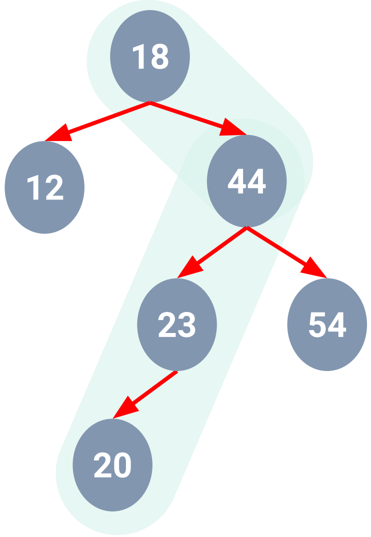
Left then Right Rotation



Right then Left Rotation

Additional rules:

- If a subtree needs a **left rotation**,
    - Check the balance factor of the right child.
    - If the right child is left-heavy, then do **right rotation on the right child**.
    - Then do left rotation on the subtree.



Left of Right

Balancing AVL Tree

Left of Left

Start → (X) = (Y).left → (Y).left = (X).right → (X).Right = (Y) → Stop

Right of Right

Start → (X) = (Y).right → (Y).right = (X).left → (X).left = (Y) → Stop

Right of Left

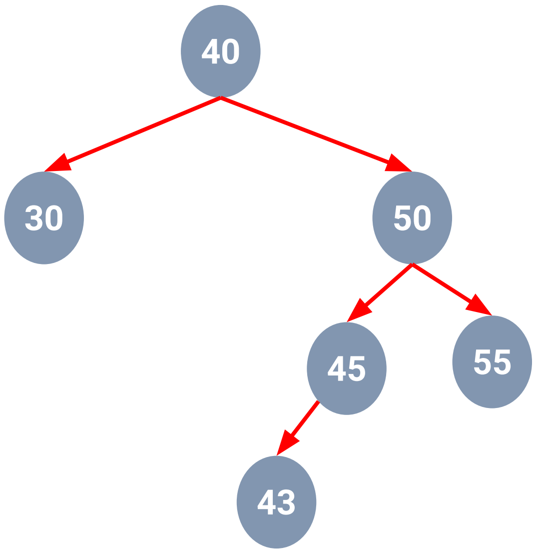
Start → Right-Of-Right (Y1) → Left-Of-Left(Y) → Stop

Left of Right

Start → Left-Of-Left(Y1) → Right-Of-Right(Y) → Stop

Balancing AVL Tree Example

Example: Rebalance the given AVL tree.

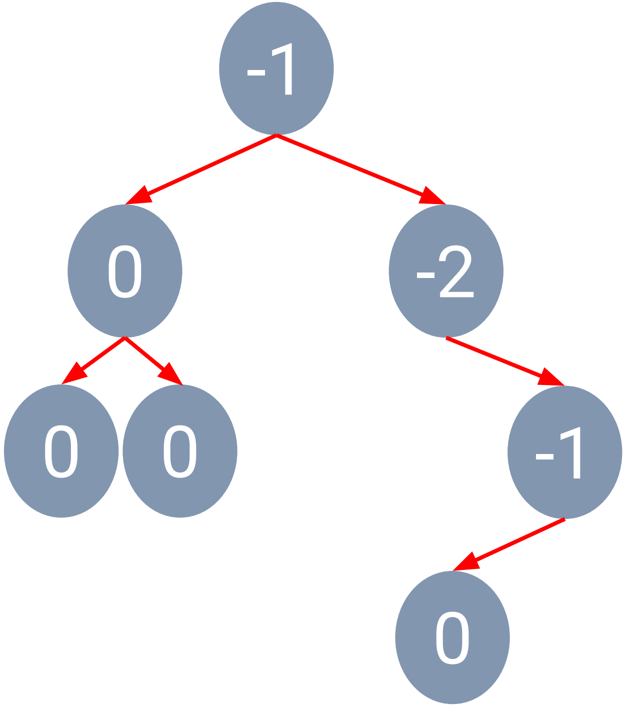


Balancing AVL Tree Exercise

Exercise: Insert the nodes into an AVL tree according to this order: 40, 50, 65 and rebalance the AVL tree.

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang

Summary



- AVL tree ensures that accessing the node costs only ***O*(log$_2$n)** time after deleting or inserting a node.