COMP26120 Algorithms and Complexity
Topic 2: Data Structures

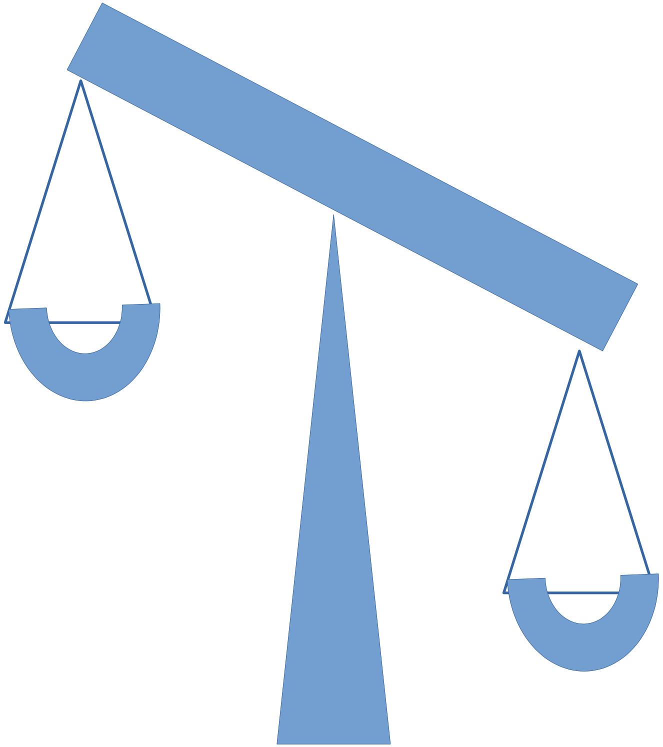
AVL Trees
aka: Self-Balancing BST

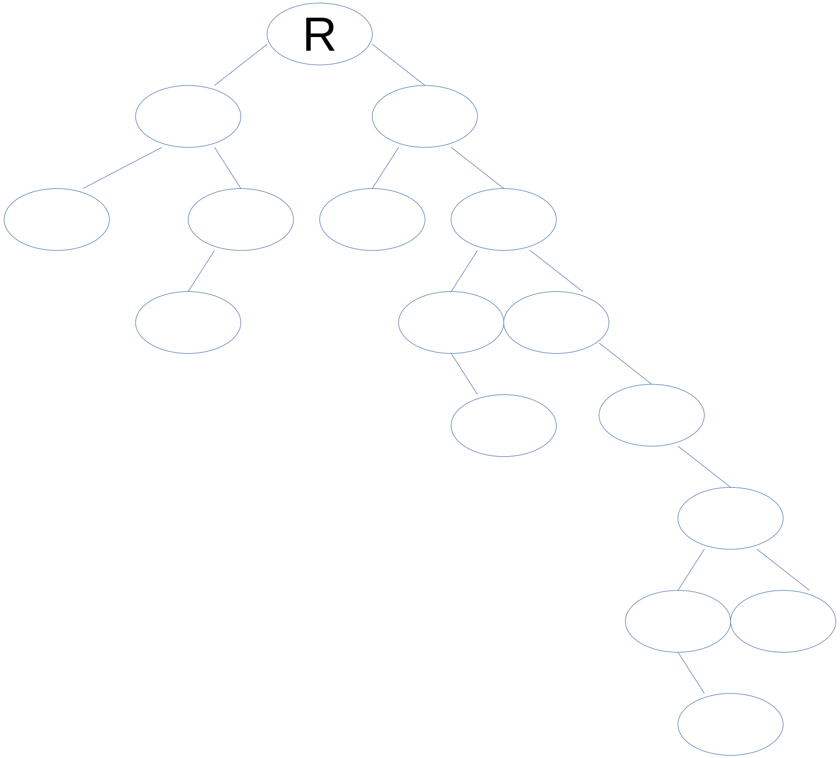
Dr. Thomas Carroll
thomas.carroll@manchester.ac.uk
All information on Blackboard

Learning Objectives

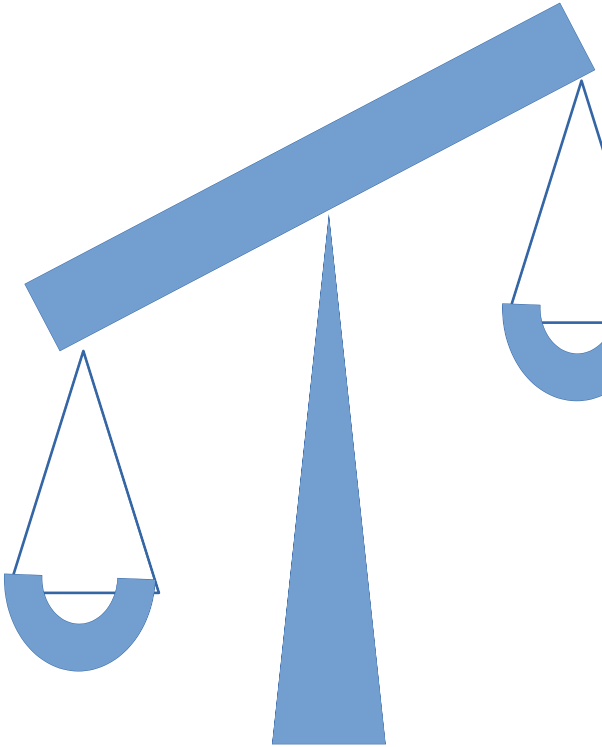
- Understand the Concept of an AVL Tree
- Understand Balance in an AVL Tree
- Understand Rotation Operations
- Recall the Complexity of Operations in AVL Tree

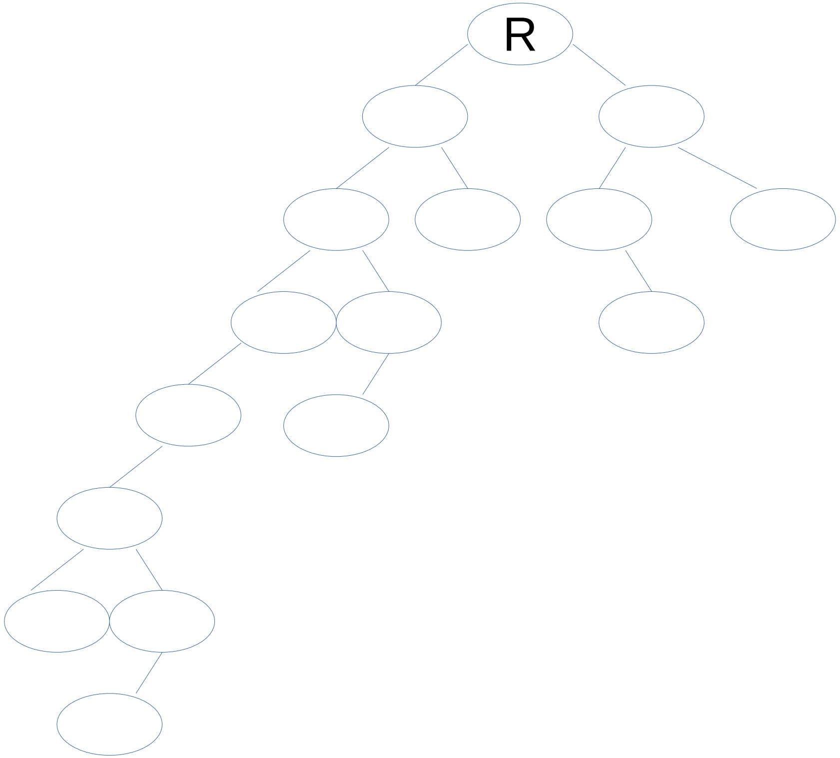
All in the Balance



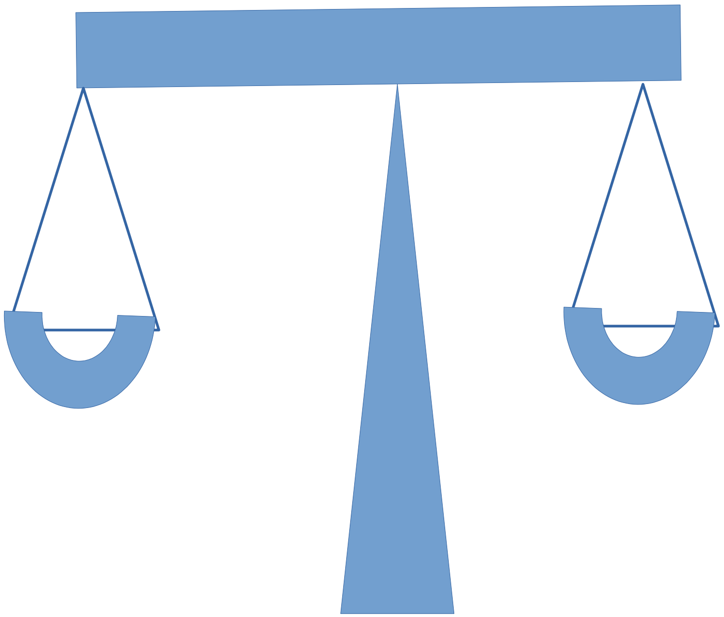


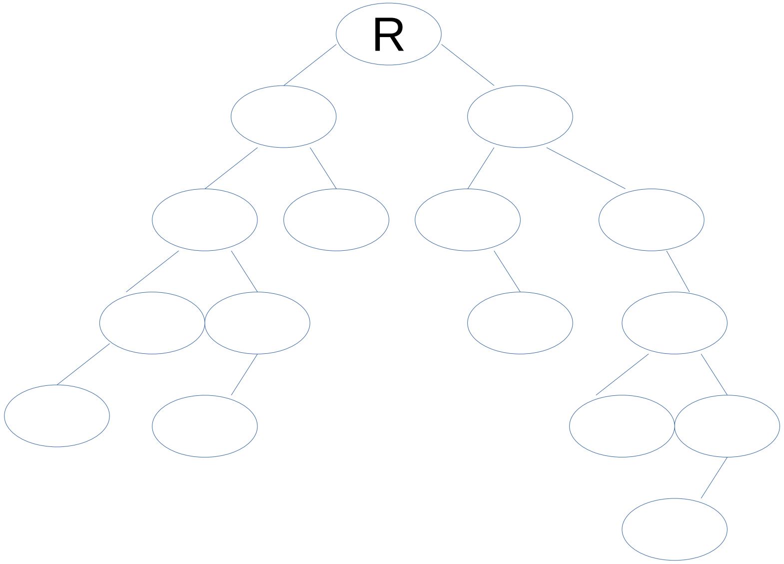
All in the Balance





All in the Balance

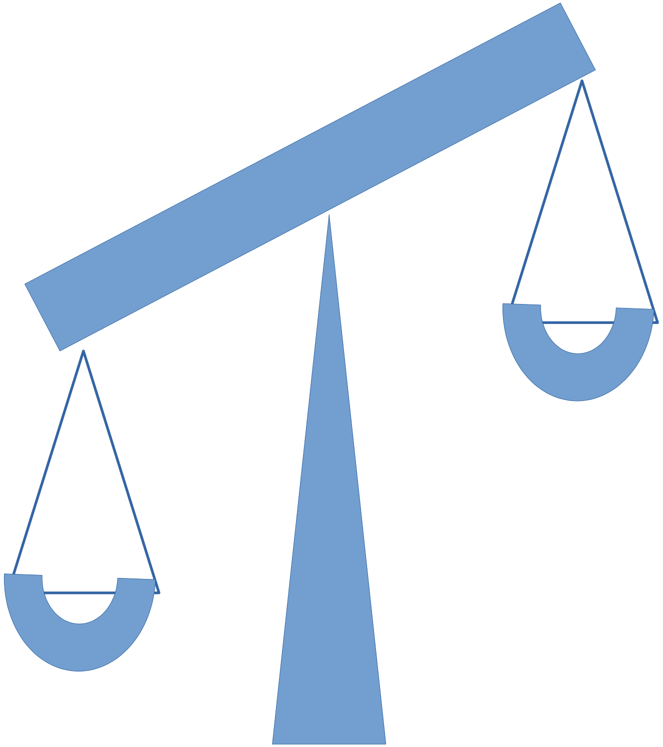


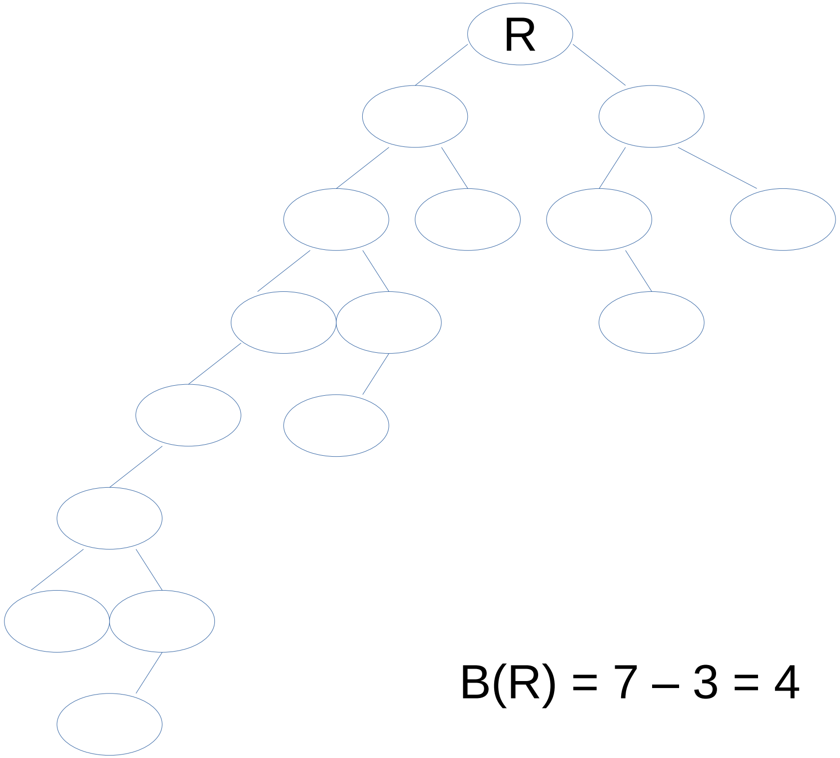


Height Balance Property

- B(n) = H(n.leftChild()) - H(n.rightChild)
  - >0 = LEFT HEAVY
  - <0 = RIGHT HEAVY
  - 0 = Just Right :)

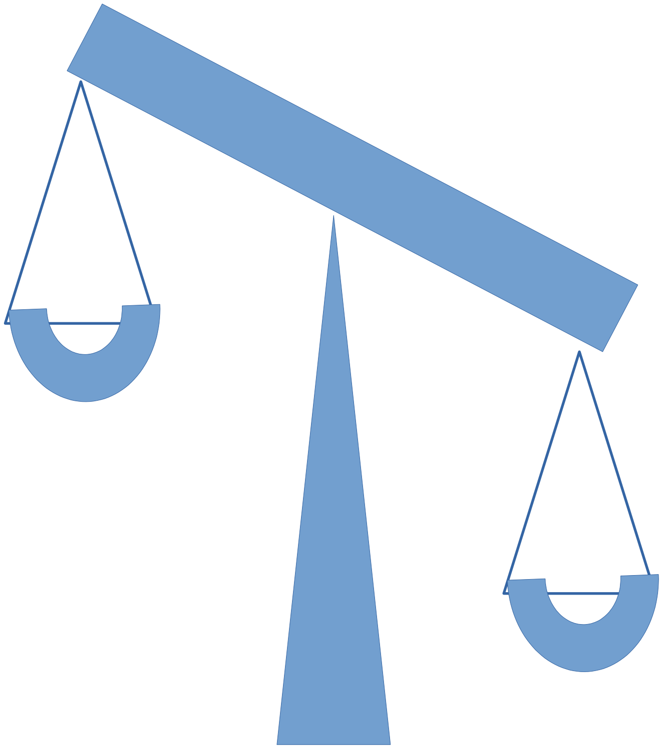
Left Heavy

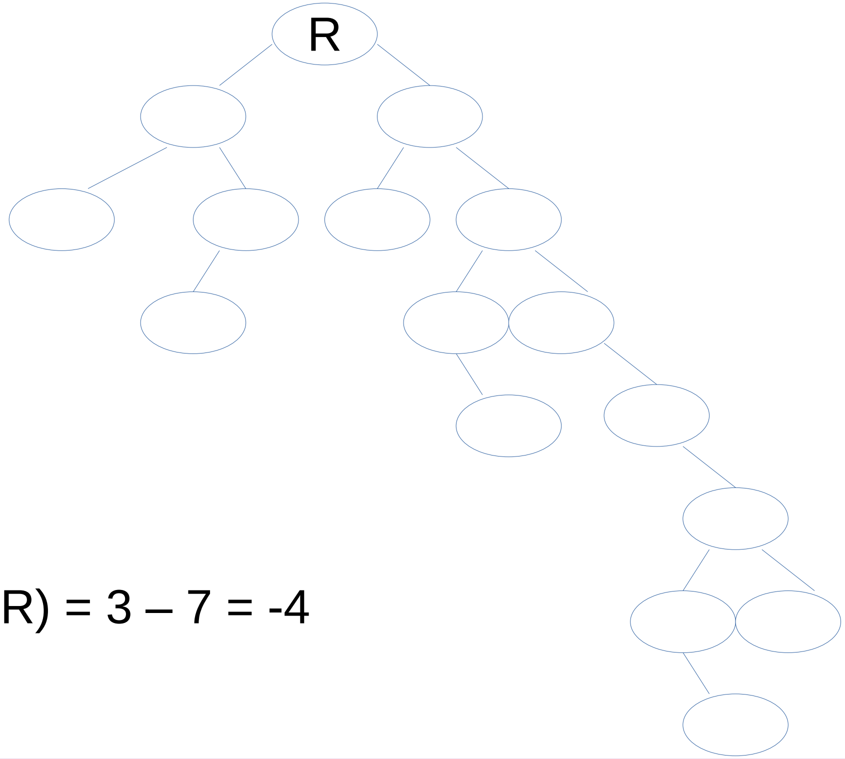




B(R) = 7 – 3 = 4

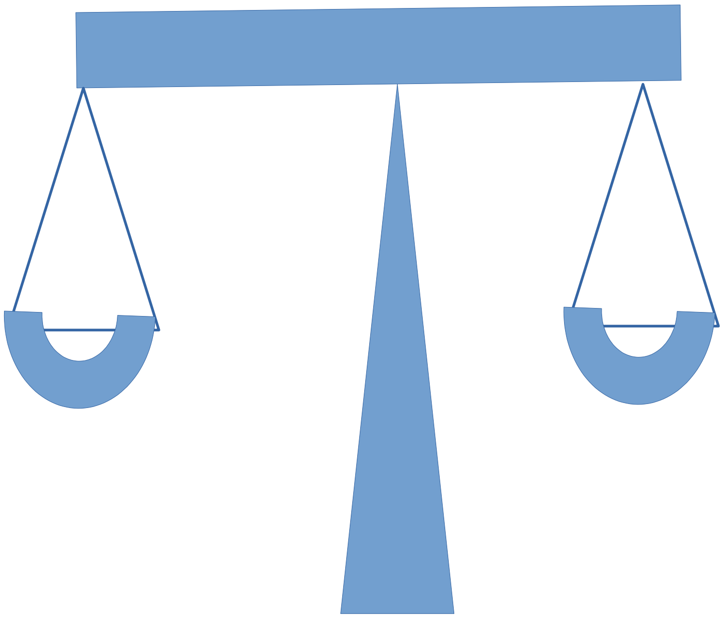
Right Heavy

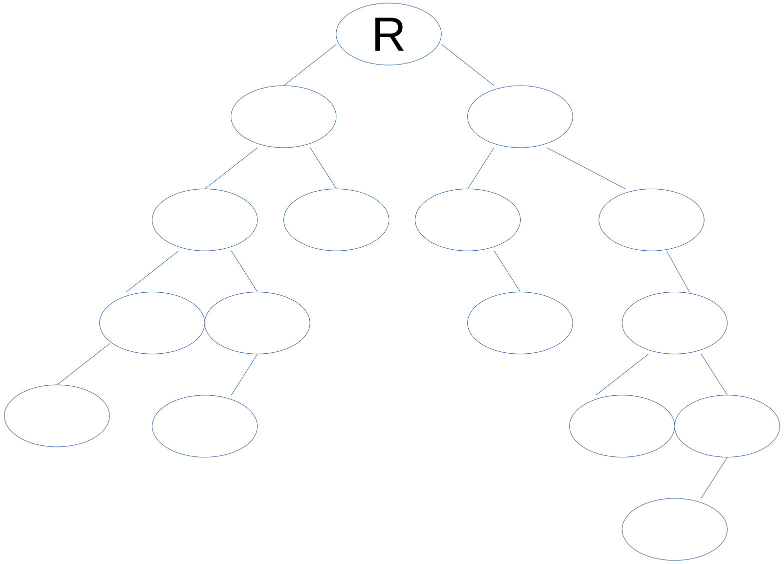




$B(R) = 3 - 7 = -4$

Just Right?





B(R) = 4 – 5 = -1

AVL Tree

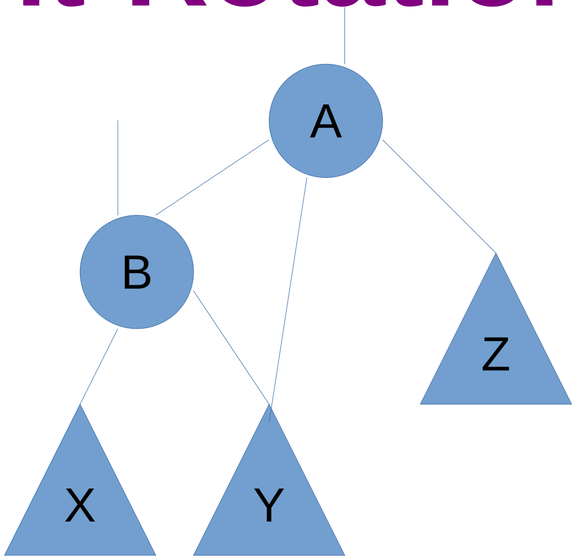
- An AVL Tree is a special type of binary search tree
- It has **another** invariant:
  - Balance at all nodes n is -1 <= B(n) <= 1
    - **|B(n)| <= 1**
  - Plus the existing BST invariant:
    - Left < Node <= Right
- The **Height Balance Property** ensures that subtrees are of (roughly) equal height
- **Rotation operations** occur after insert/delete to maintain the height balance property

Rotation

- A **Rotation** is a tree-manipulation operation
- It moves nodes around, whilst keeping the invariants that must be kept
- In AVL trees, rotation changes the root of a subtree.

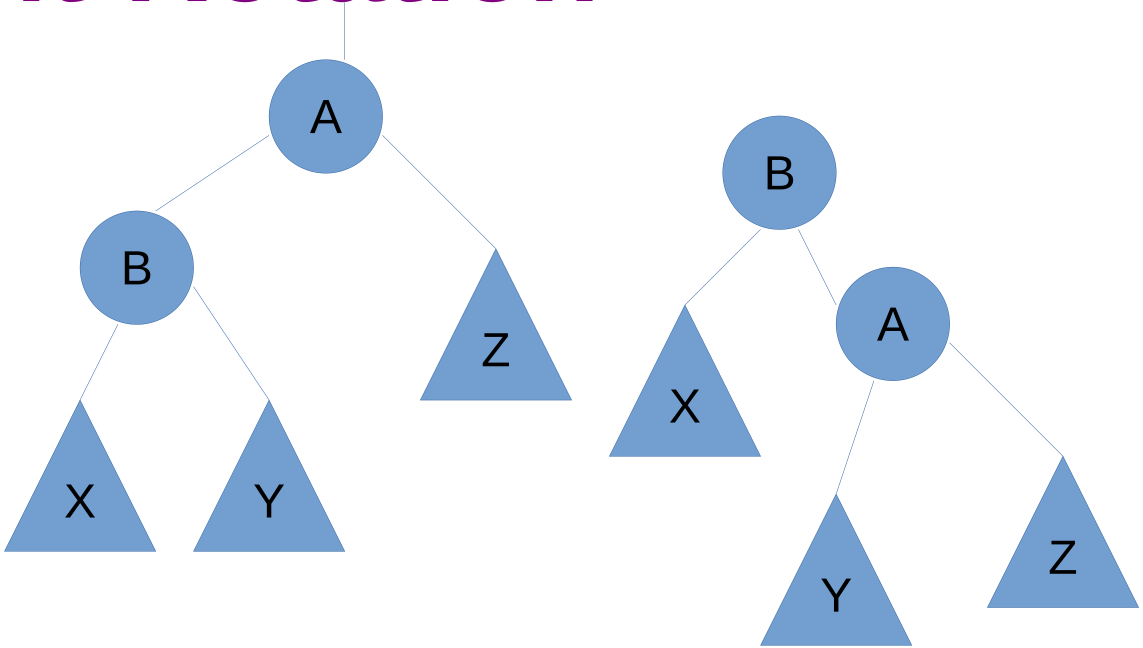
Right-Rotation

- A right rotation shifts nodes to the right of the root!



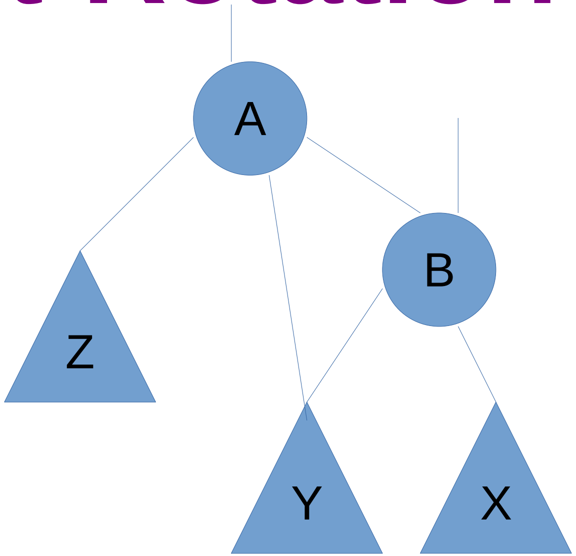
Right-Rotation

- A right rotation shifts nodes to the right of the root!



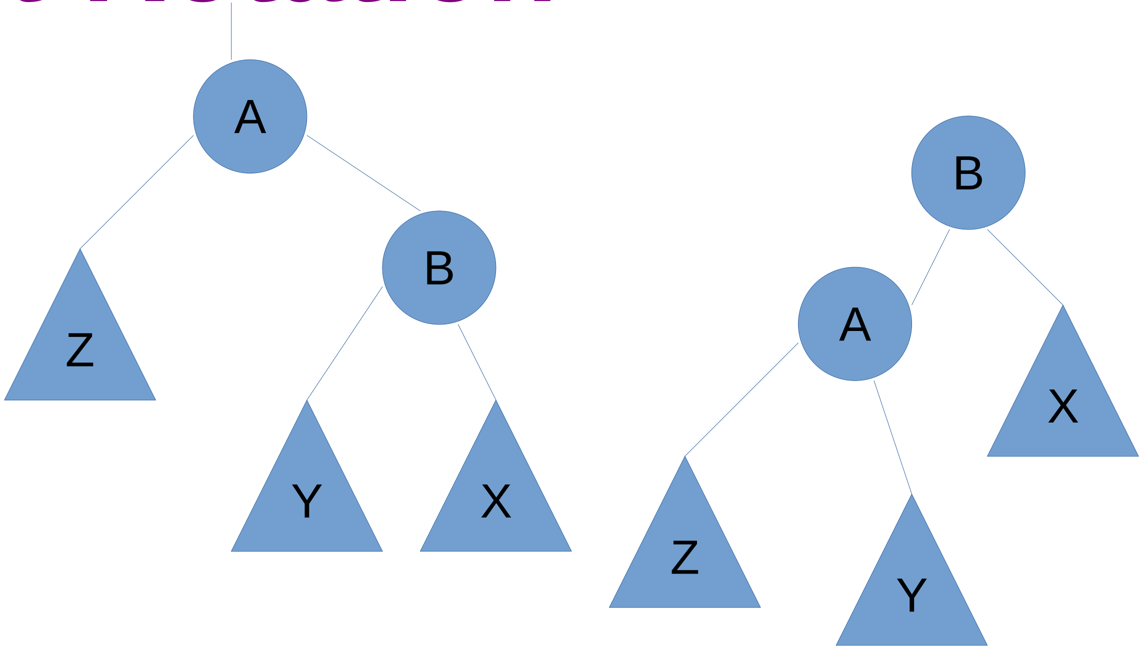
Left-Rotation

- A left rotation shifts nodes to the left of the root!



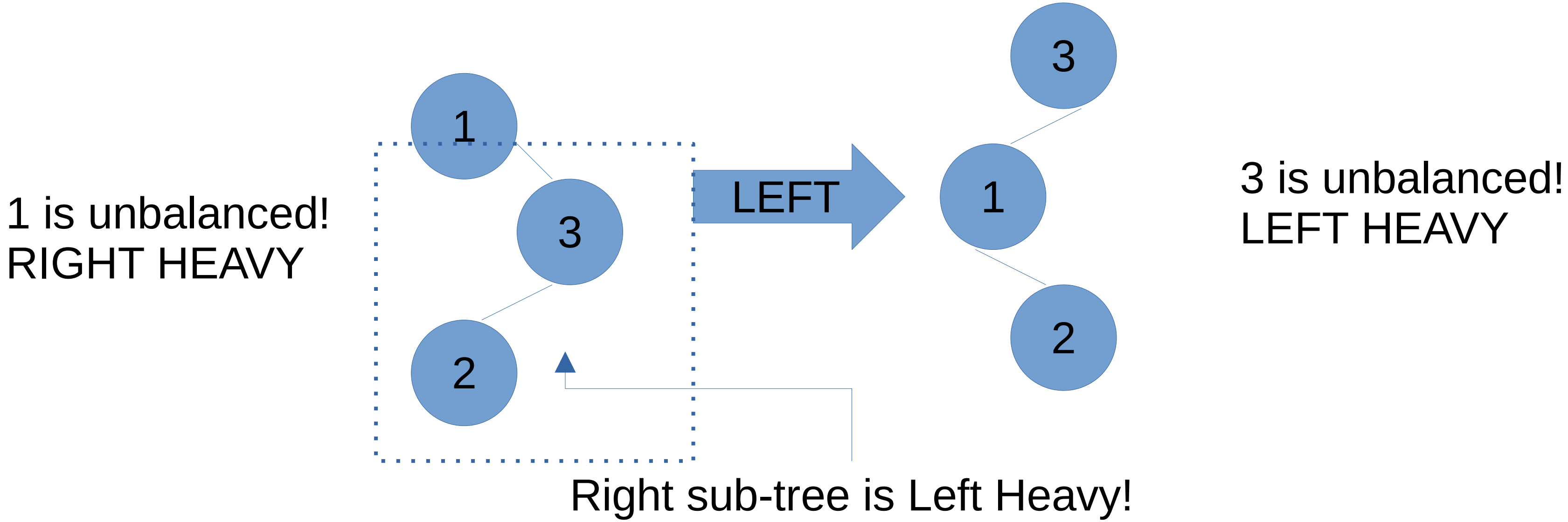
Left-Rotation

- A left rotation shifts nodes to the left of the root!



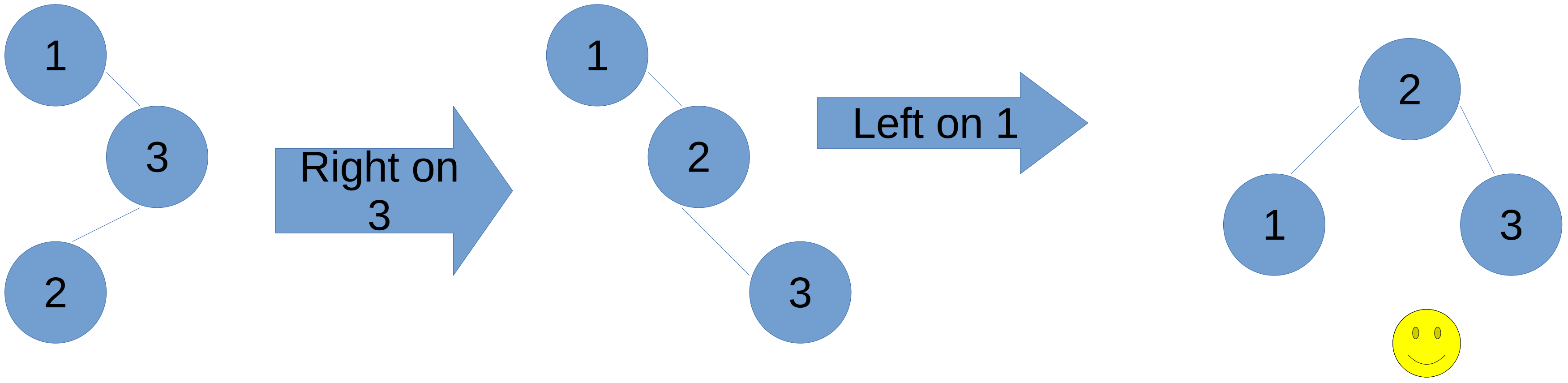
Double Rotations

- Sometimes, a single rotation is not good enough:



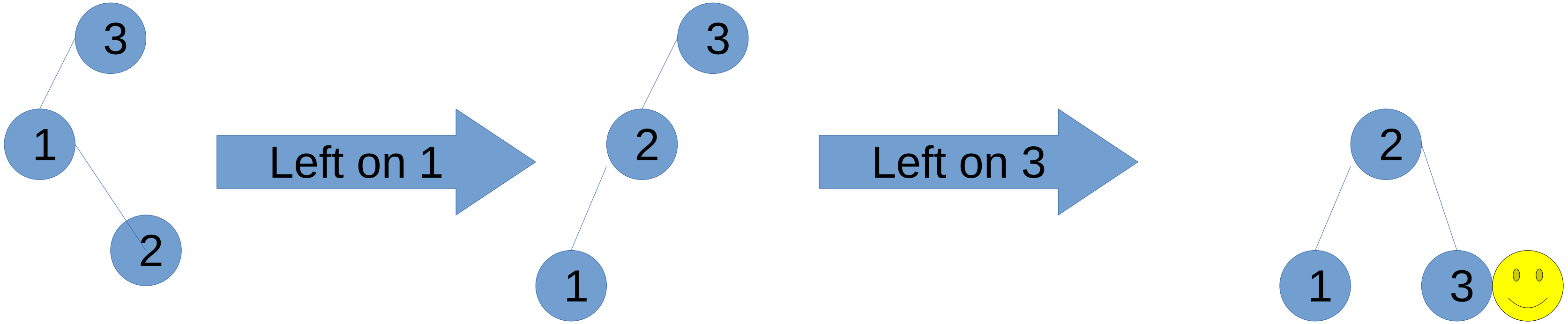
Left-Right Rotation

- Use this when you have a LEFT heavy RIGHT subtree
- Right rotate on the left subtree
- Left rotate on original root



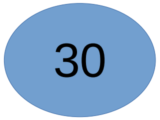
Right-Left Rotation

- Use this when you have a RIGHT heavy LEFT subtree
- Left rotate on the Left subtree
- Right rotate on original root



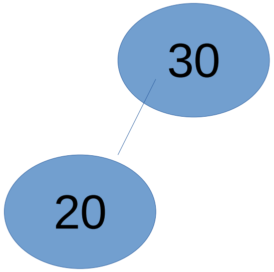
Worked Example

Insert 30



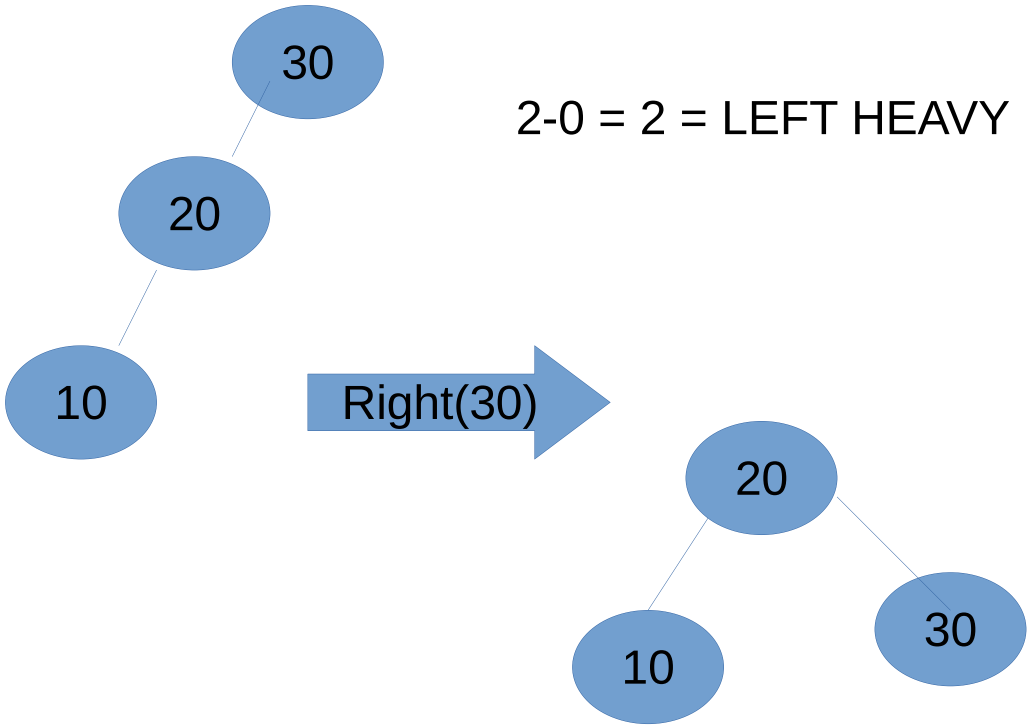
Worked Example

Insert 20



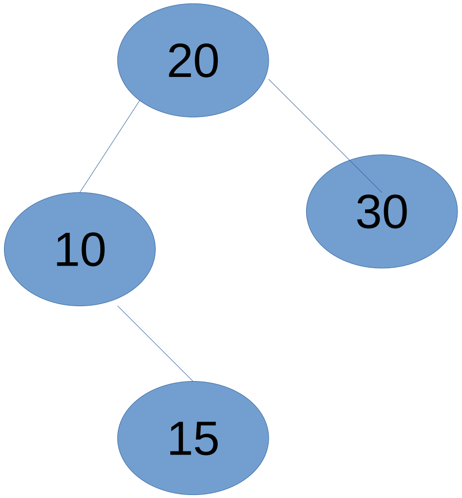
Worked Example

Insert 10



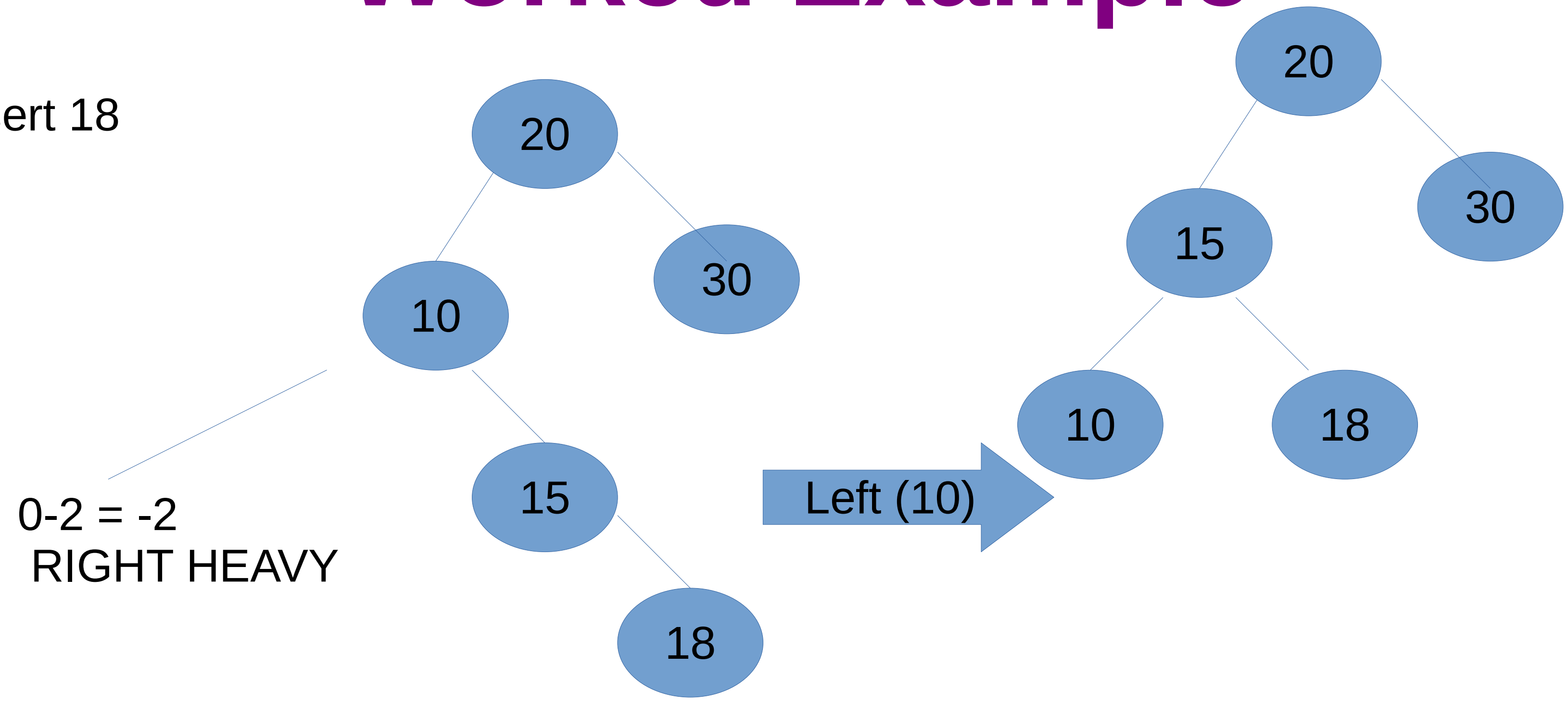
Worked Example

Insert 15

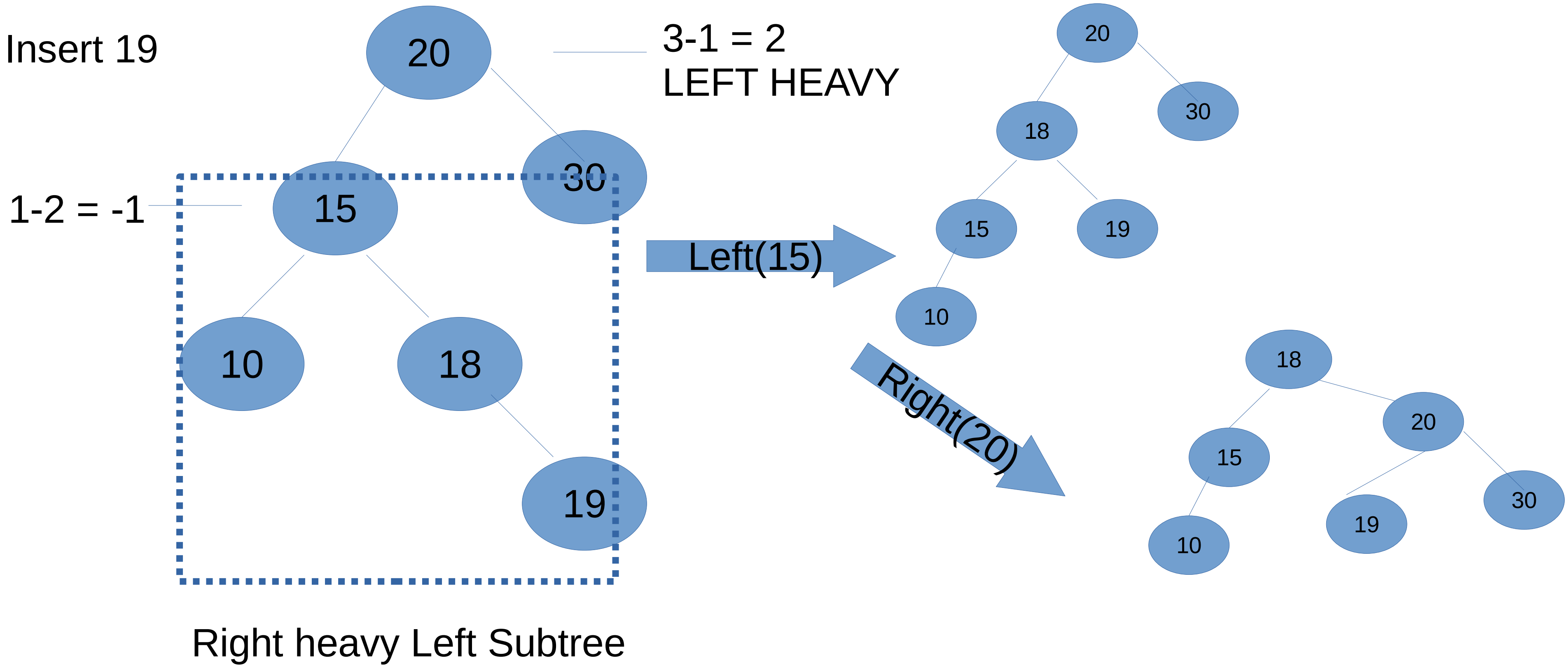


Worked Example

Insert 18



Worked Example

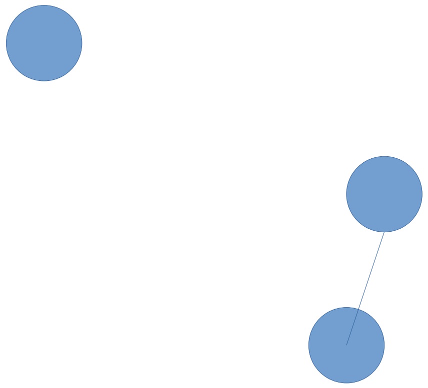


Complexities of AVL Tree

- For reasoning about the complexities of AVL Tree, we need to think about the **height** of the tree…
- Claim:
  - Height of AVL tree with n nodes is O(log n)

Proof

- Ask the question: What is the minimum number of nodes in a tree of height *h*?
  - Represent as N(h)
  - N(1) = 1
  - N(2) = 2

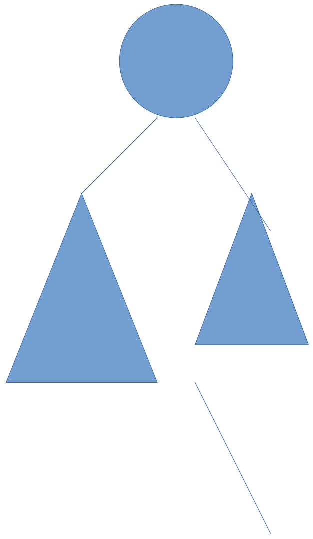


h >= 3?

- AVL tree nodes at height *h* have two children of some min heights:
  - One with height *h-1*
  - One with height *h-2*
- Min number of nodes:

  $N(h) = N(h-1) + N(h-2) + 1$

  $N(h) > 2N(h-2)$



Must be balanced!

Can we Generalise?

$N(h) > 2N(h-2)$

$N(h-2) > 2N(h-4)$

$N(h) > 2*2N(h-4) = 4N(h-4)$

$N(h-4) > 2N(h-6)$

$N(h) > 4*2N(h-6) = 8N(h-6)$

$\mathbf{N(h) > 2^i N(h-2i)}$

Can we use N(1)?

$N(h) > 2^i N(h-2i)$

$N(1) = 1, N(2) = 2$

$h-2i = 1$
$i=h/2-1$

$N(h) > 2^{h/2-1} * N(1)$
$N(h) > 2^{h/2-1}$

- Take the log of both sides:

$\log N(h) > \log 2^{h/2-1}$

$\log N(h) > h/2 - 1$

$h < 2 \log n(h) + 2$

So, h is O(log n)