Algorithms and Data Structures for Data Science
AVL Trees

CS 277
Brad Solomon

April 3, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

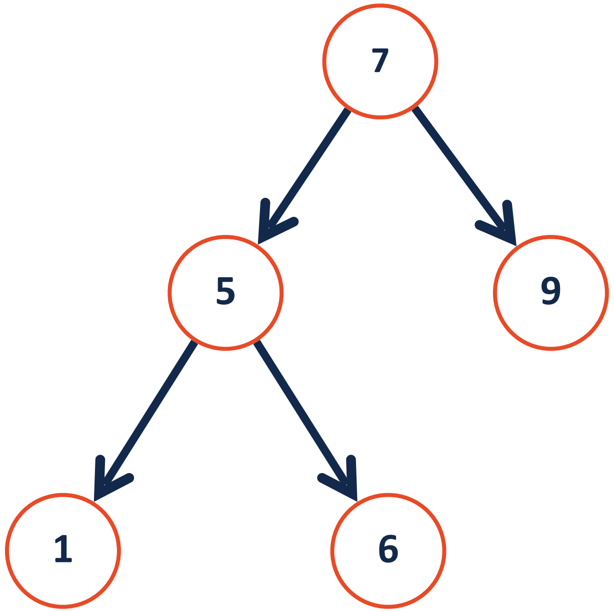
Review tree runtimes and introduce more tree terminology

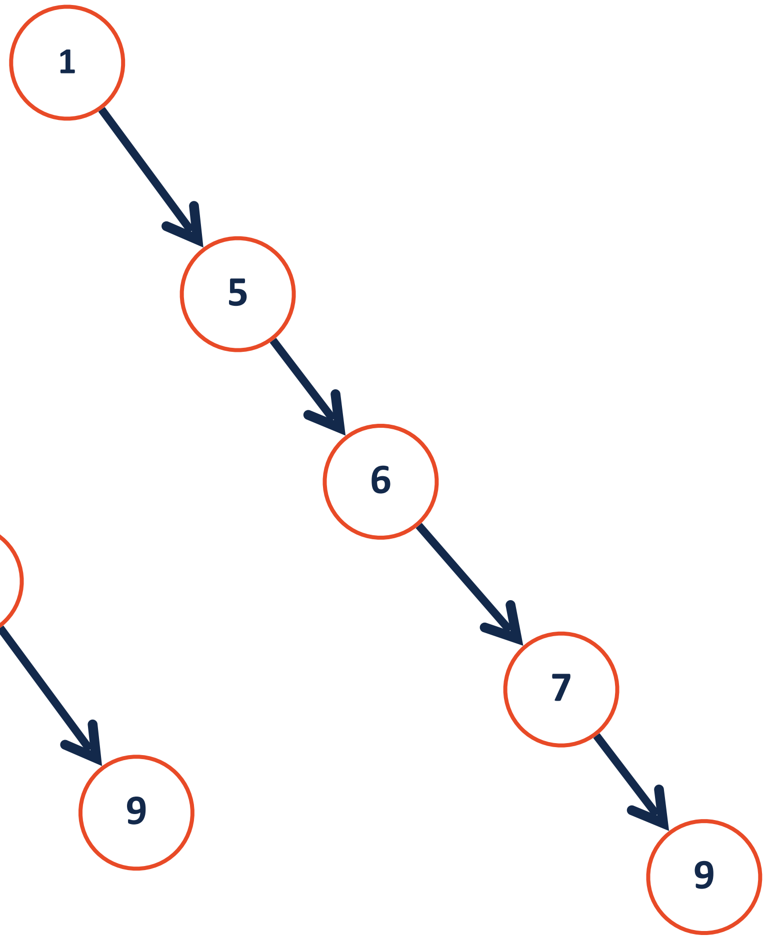
Introduce the AVL tree

Demonstrate how AVL tree rotations work

BST Analysis – Running Time

	BST Worst Case
find	O(h)
insert	O(h)
delete	O(h)
traverse	O(n)





BST Analysis

Every operation on a BST depends on the **height** of the tree.

… how do we relate $O(h)$ to n, the size of our dataset?

BST Analysis

What is the max number of nodes in a tree of height h ?

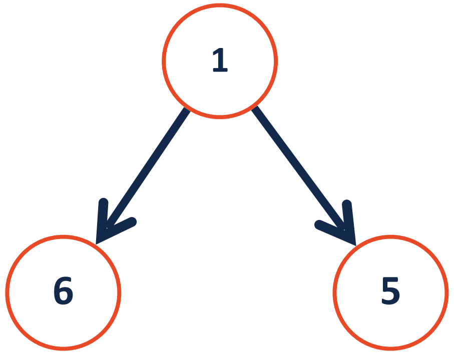
BST Analysis

What is the min number of nodes in a tree of height h ?

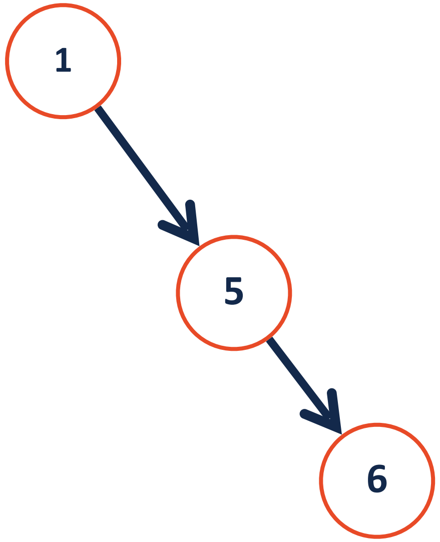
BST Analysis

A BST of n nodes has a height between:

Lower-bound: $O(\log n)$



Upper-bound: $O(n)$



Fixing Tree Height

1. Only use trees with certain pre-defined properties

2. Make sure the order of our insert is near perfect

3. Correct an unbalanced tree when we see it

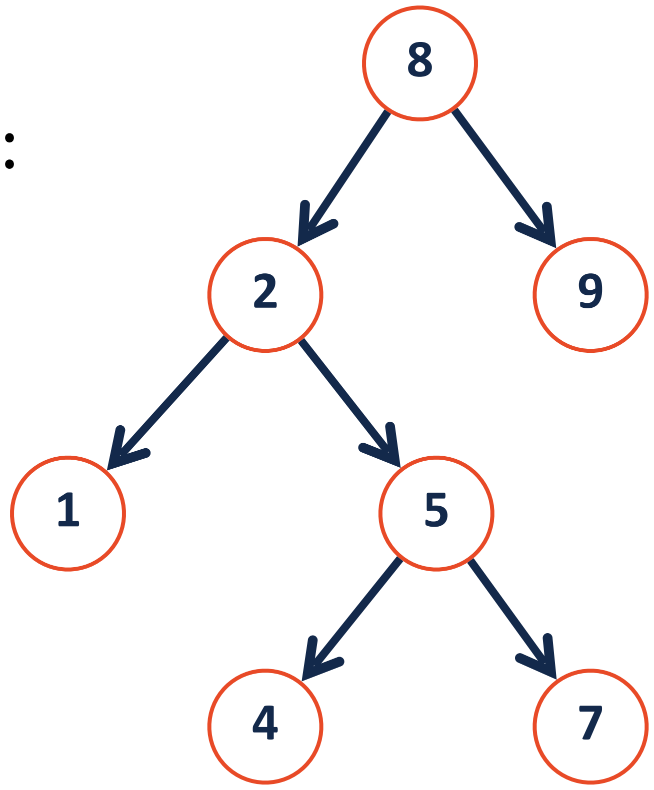
Tree Property: "Full"

A tree F is **full** if (and only if) for each node n:

n has zero children

OR

n has two children (which are full trees)

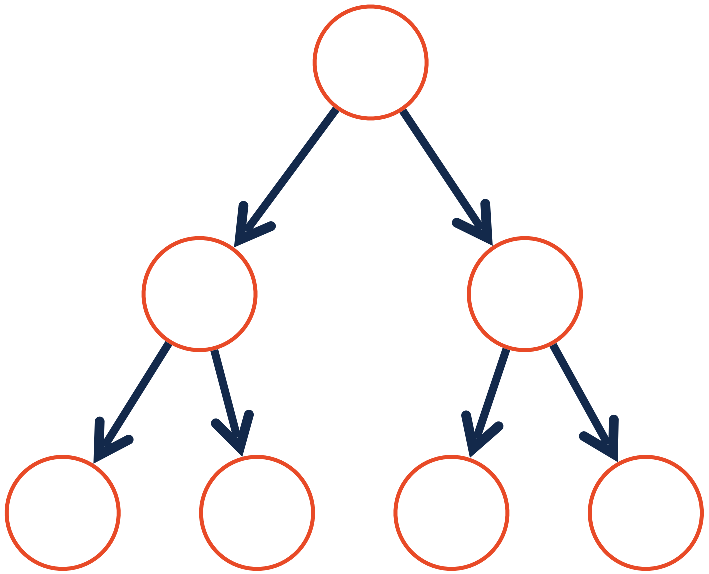


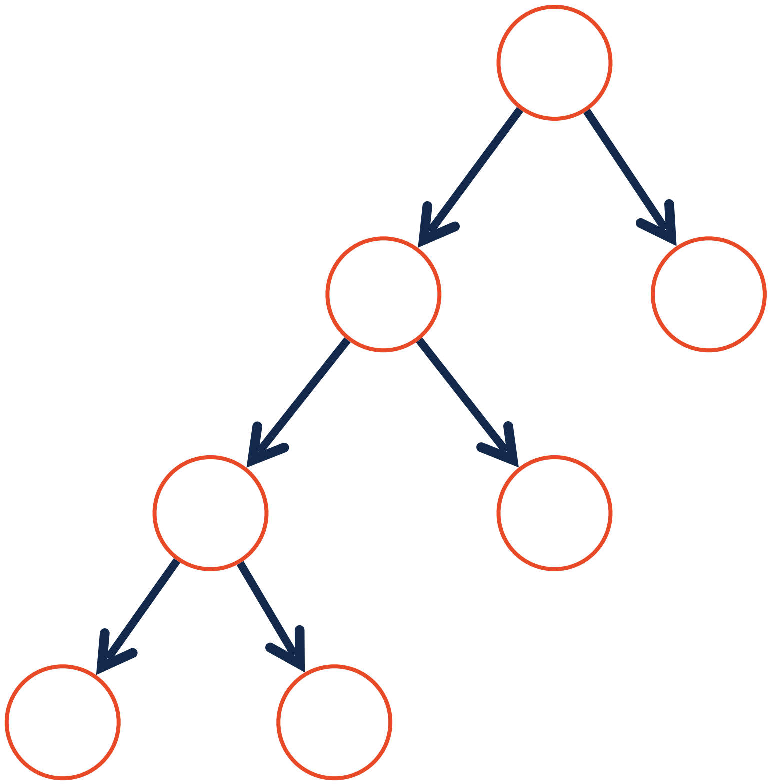
Tree Property: "Full"

Given n nodes, what are the min and max height of a **full tree**?

Tree Property: “Full”

Given n nodes, what are the min and max height of a **full tree**?





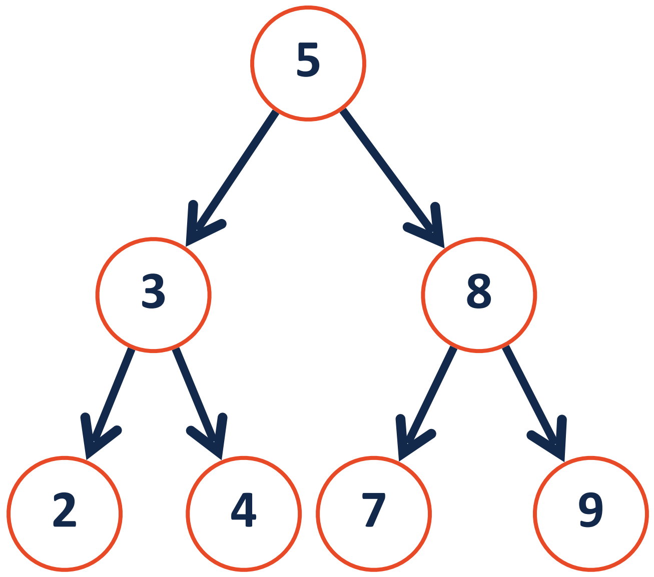
Tree Property: "Perfect"

A **perfect** tree P_h is a tree of height h where:

Every internal node has two children

AND

Every leaf has the same depth / level in the tree

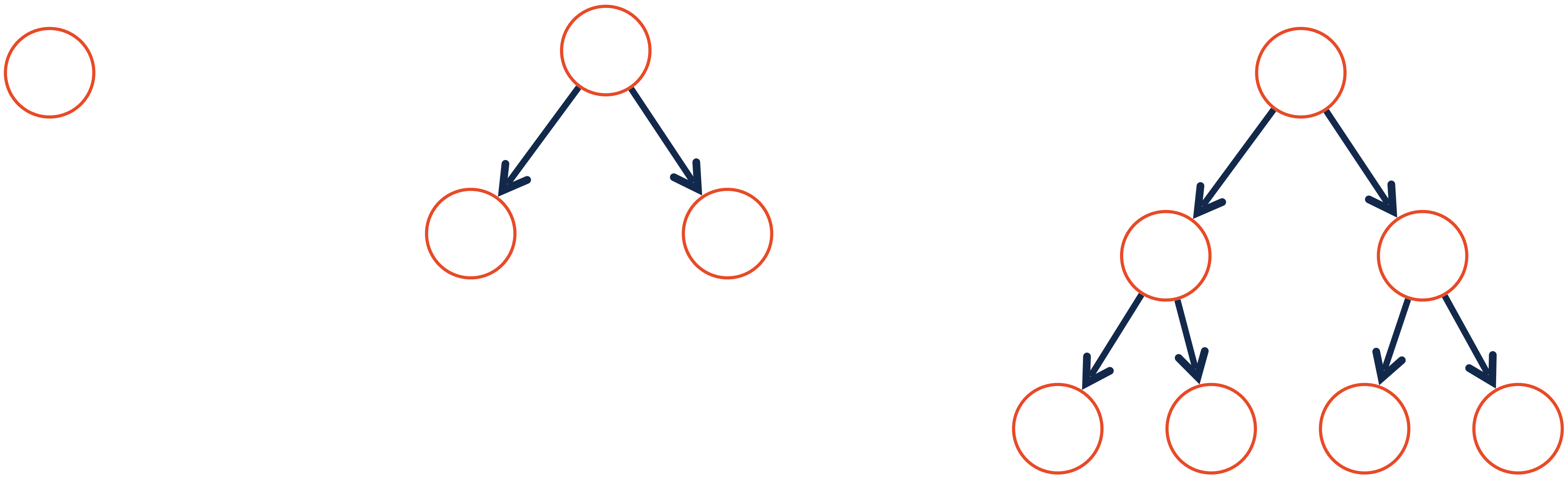


Tree Property: "Perfect"

Given n nodes, what are the min and max height of a **perfect tree**?

Tree Property: "Perfect"

Given n nodes, what are the min and max height of a **perfect tree**?



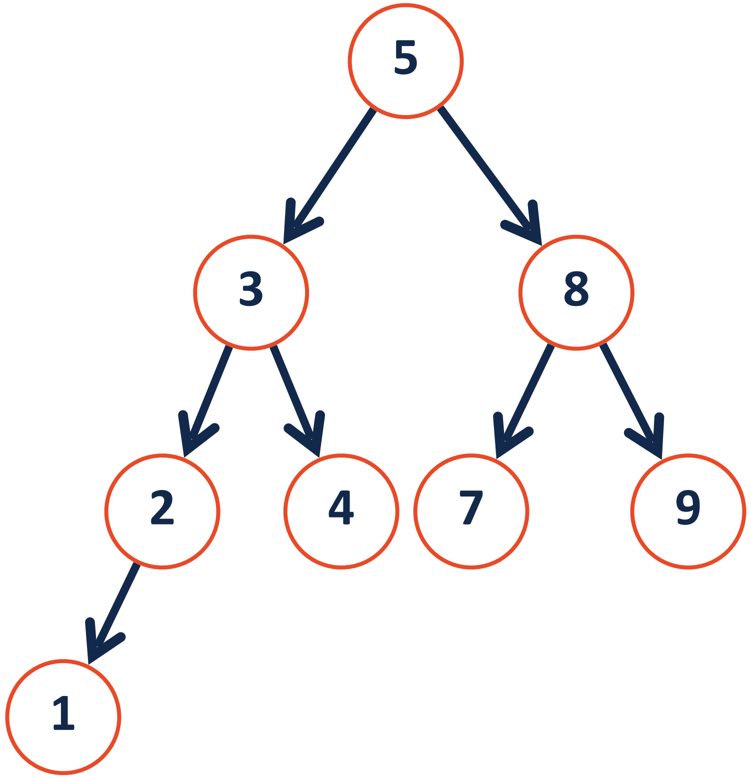
Tree Property: "Complete"

A **complete** tree C_h is a tree of height h where:

Every level is completely filled except the last

AND

Every leaf in the last level is 'pushed to the left'

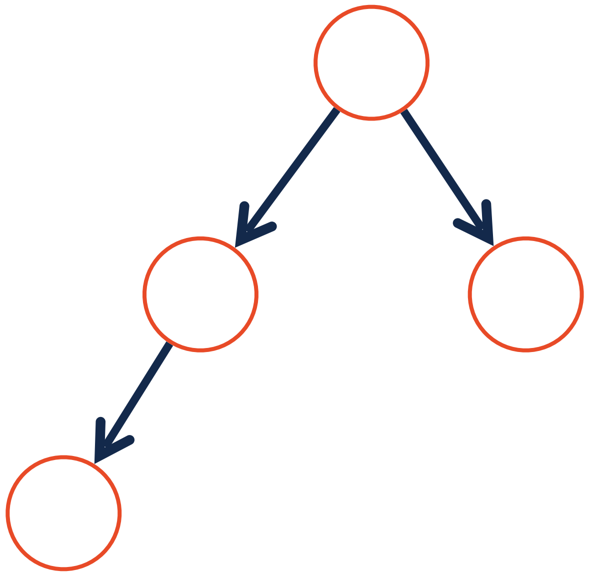


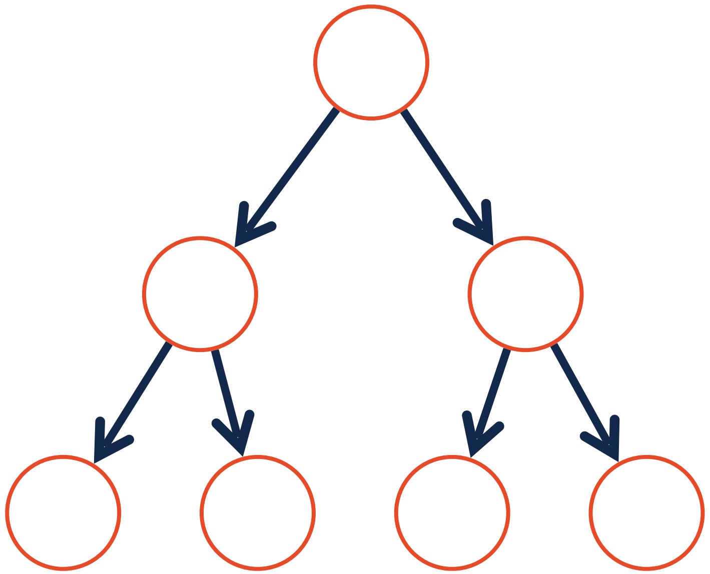
Tree Property: “Complete”

Given n nodes, what are the min and max height of a **complete tree**?

Tree Property: “Complete”

Given n nodes, what are the min and max height of a **complete tree**?





Tree Properties

A node in a **full** tree contains either zero or two children anywhere

Only the leaves in a **perfect** tree contain zero children. (All other have 2)

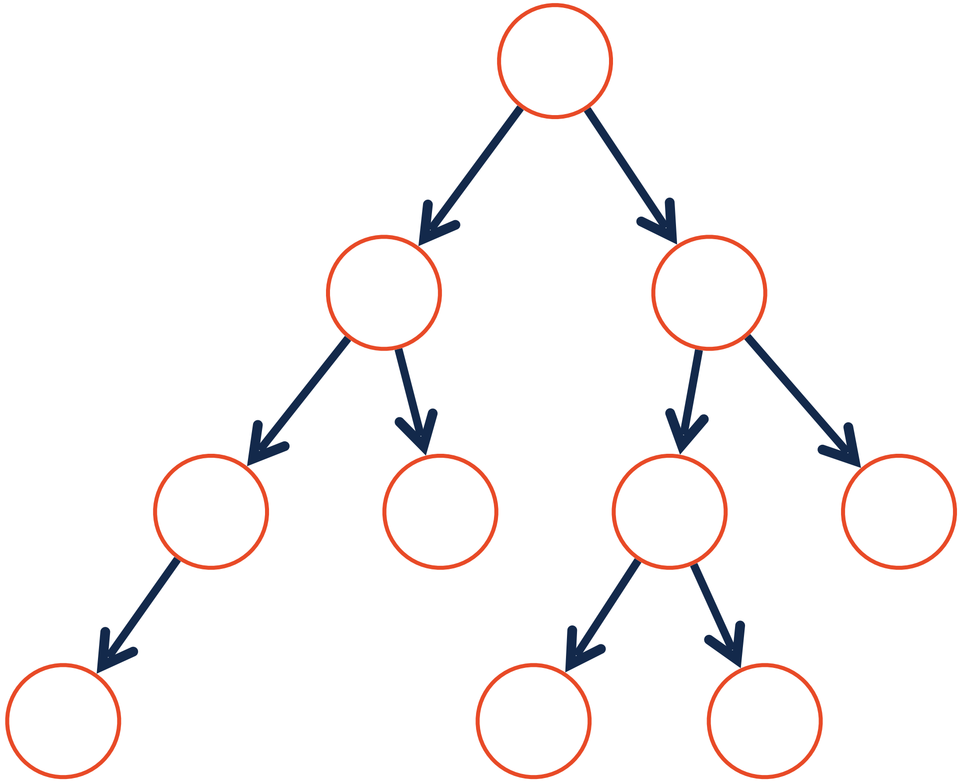
All leaves in a **perfect** tree are at the same level

Every level in the **complete** tree is full except the last level

The last level is 'pushed to the level' in a **complete** tree

Tree Properties

What properties does the following tree have (Full, Complete, Perfect)?



Correcting bad insert order

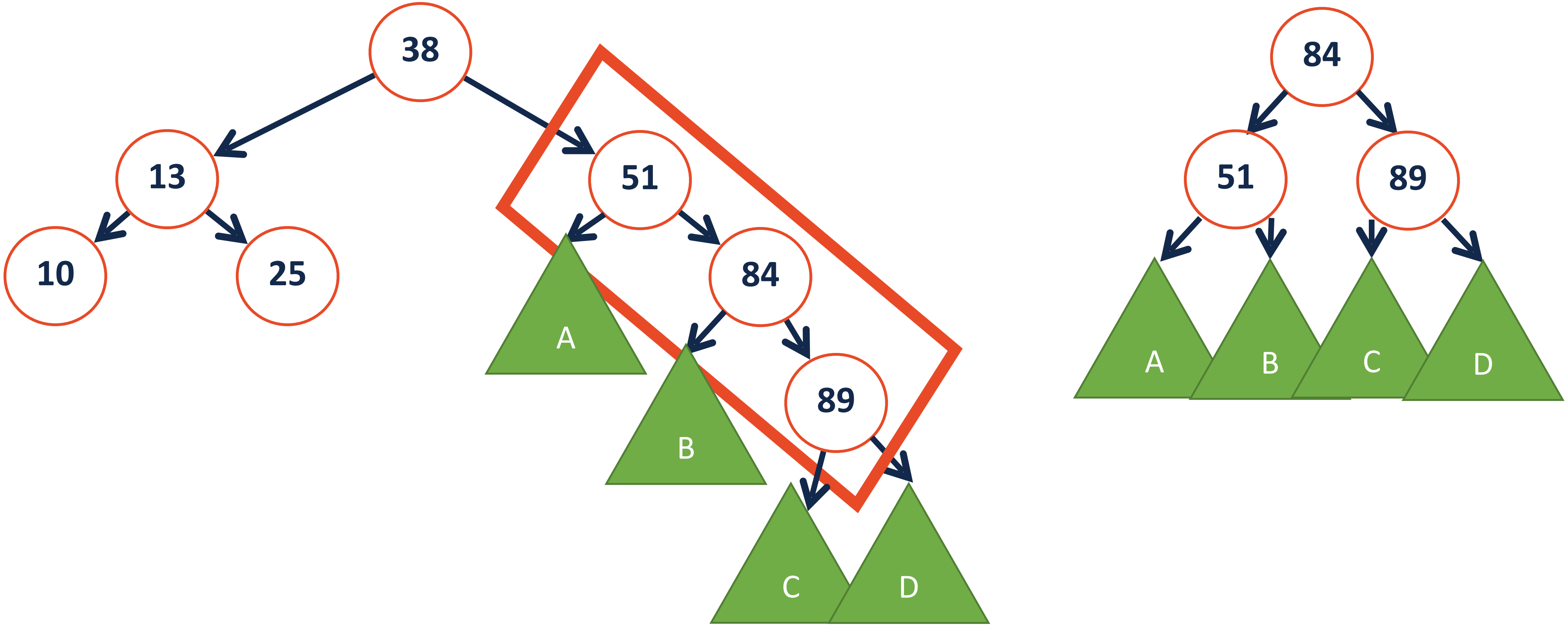
The height of a BST depends on the order in which the data was inserted

Insert Order: [1, 3, 2, 4, 5, 6, 7]

Insert Order: [4, 2, 3, 6, 7, 1, 5]

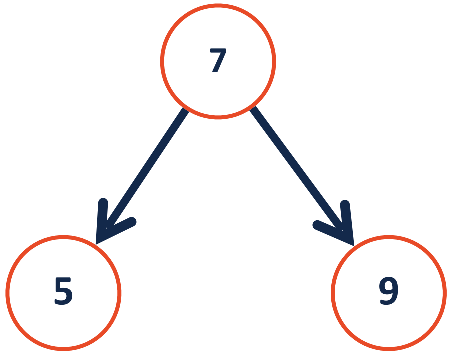
AVL-Tree: A self-balancing binary search tree

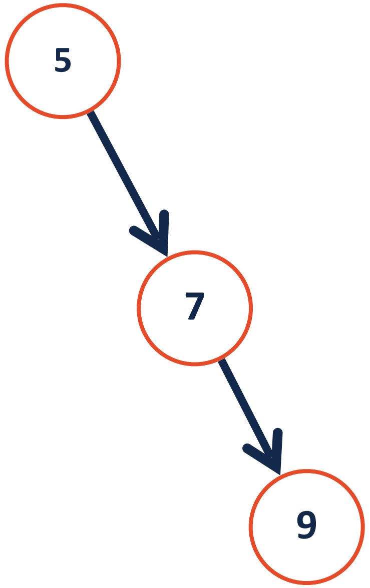
Rather than fixing an insertion order, just correct the tree as needed!



Height-Balanced Tree

What tree is better?





Height balance: $b = height(T_R) - height(T_L)$

A tree is "balanced" if:

BST Rotations (The AVL Tree)

We can adjust the BST structure by performing **rotations**.

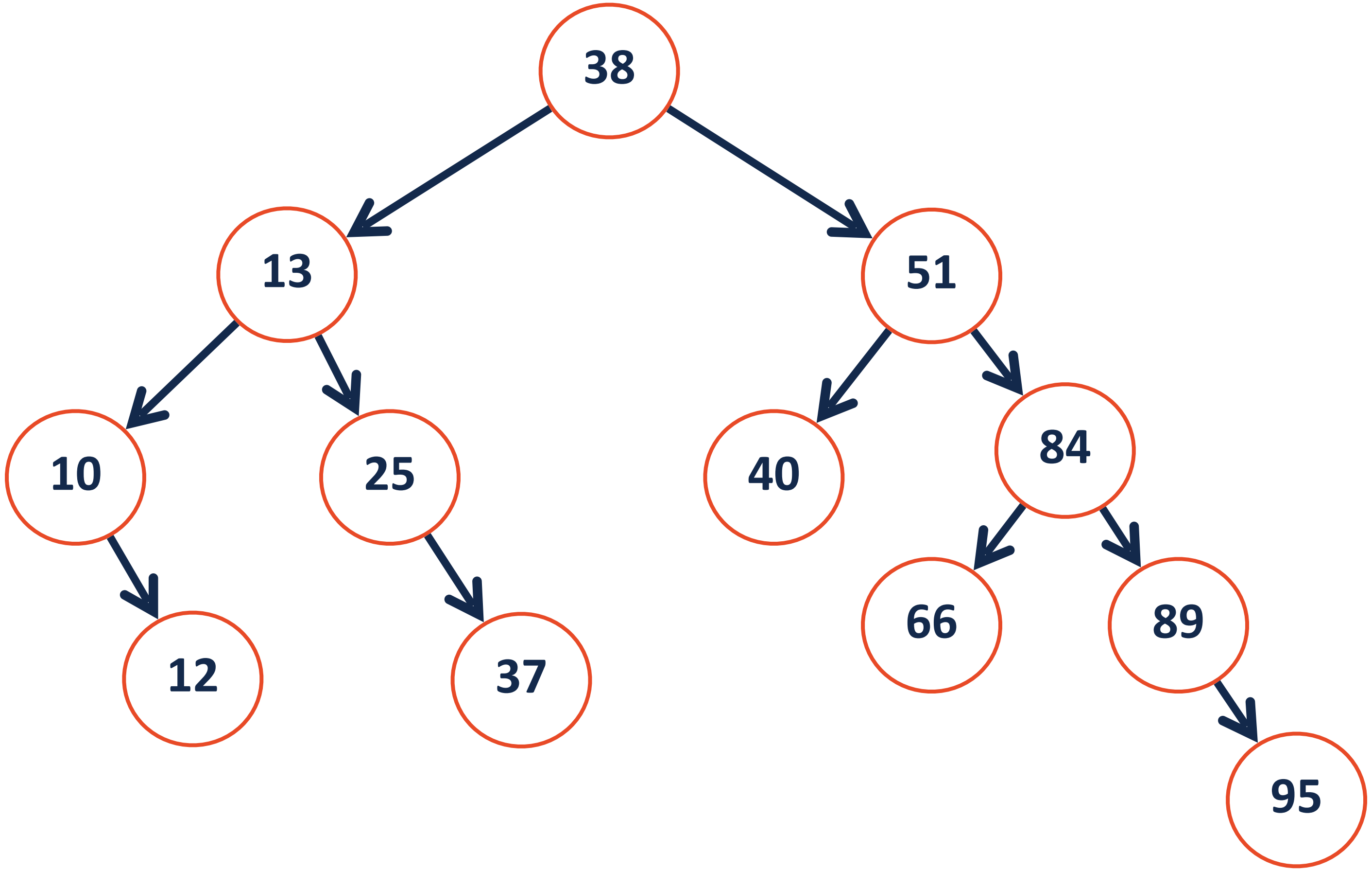
These rotations:

1.

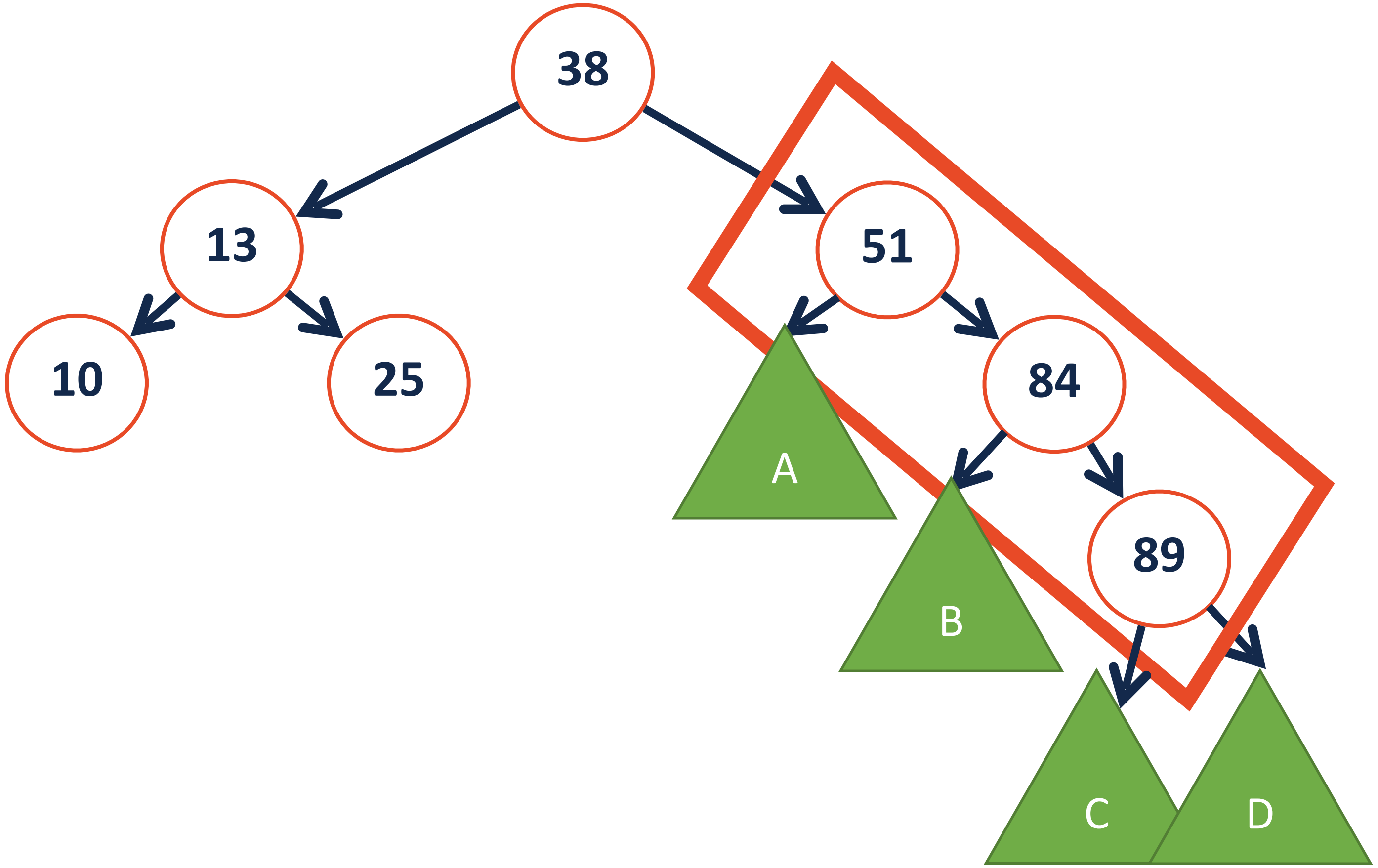
2.

BST Rotations (The AVL Tree)

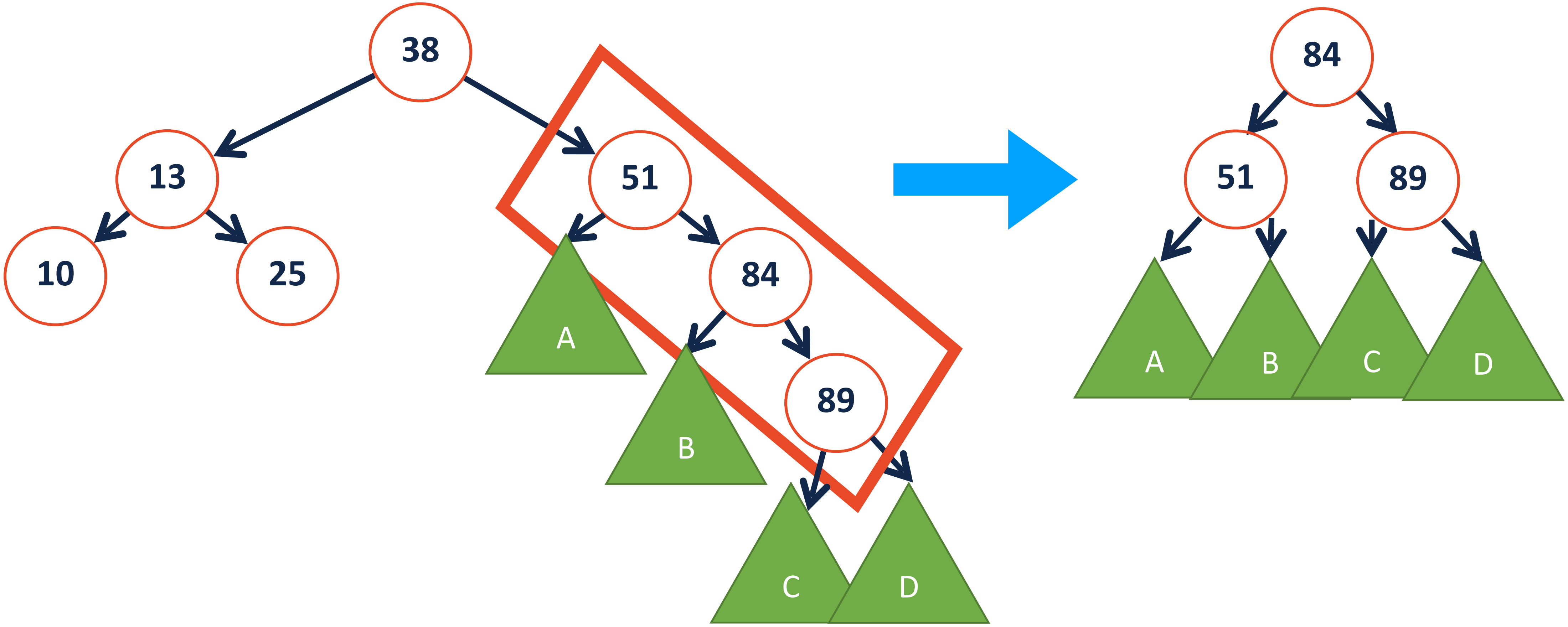
We can adjust the BST structure by performing **rotations**.



Left Rotation



Left Rotation

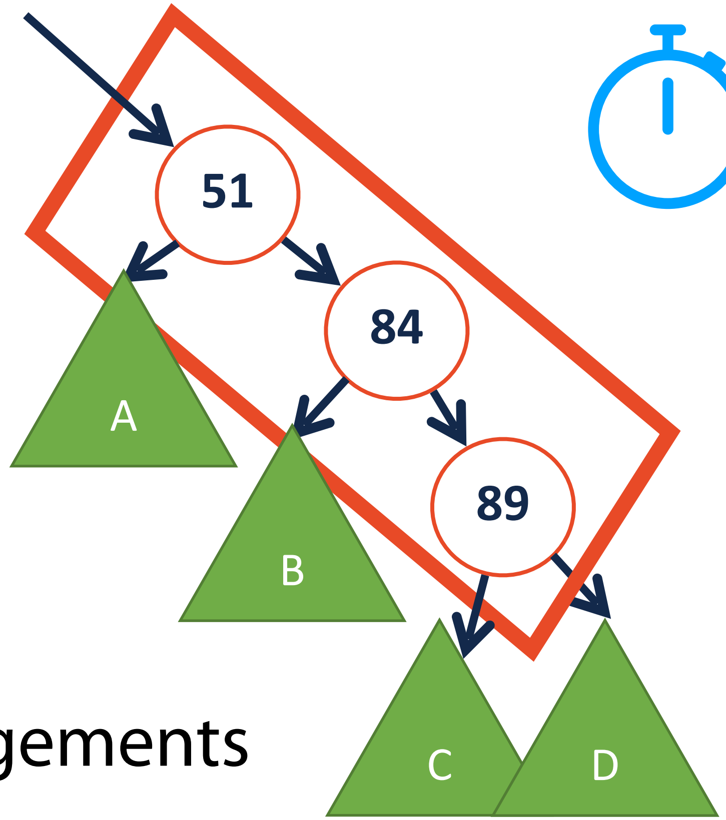


Coding AVL Rotations

Two ways of visualizing:

Think of an arrow 'rotating' around the center

Recognize that there's a concrete order for rearrangements



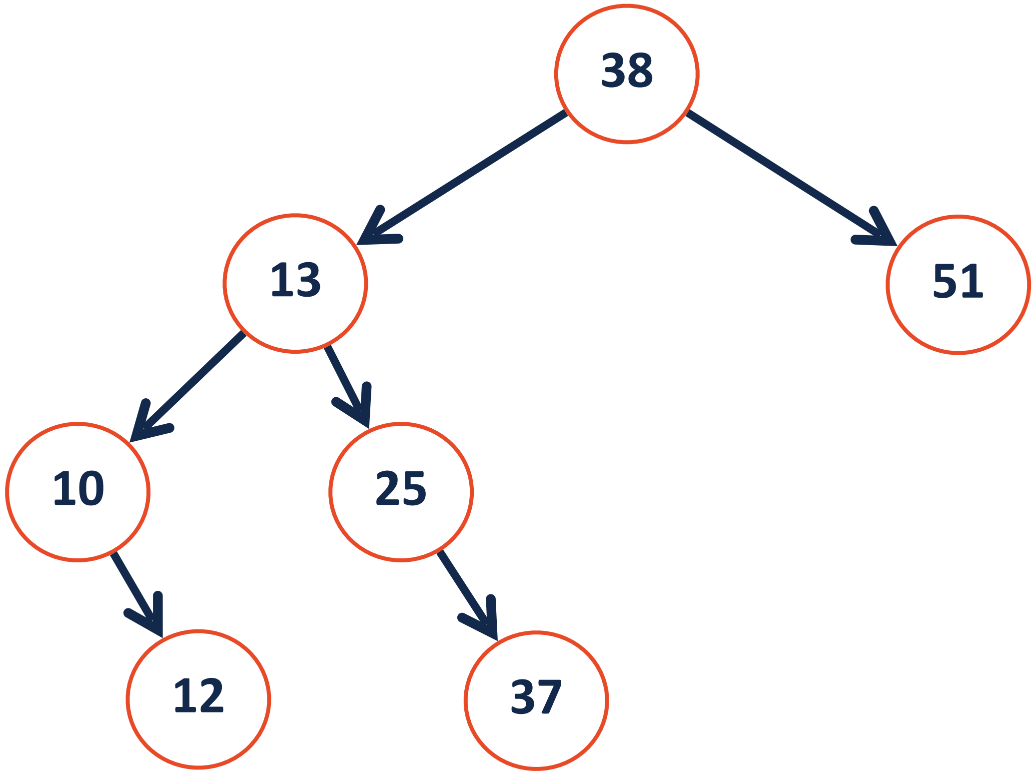
Ex: Unbalanced at current (root) node and need to *rotateLeft*?

Replace current (root) node with it's right child.

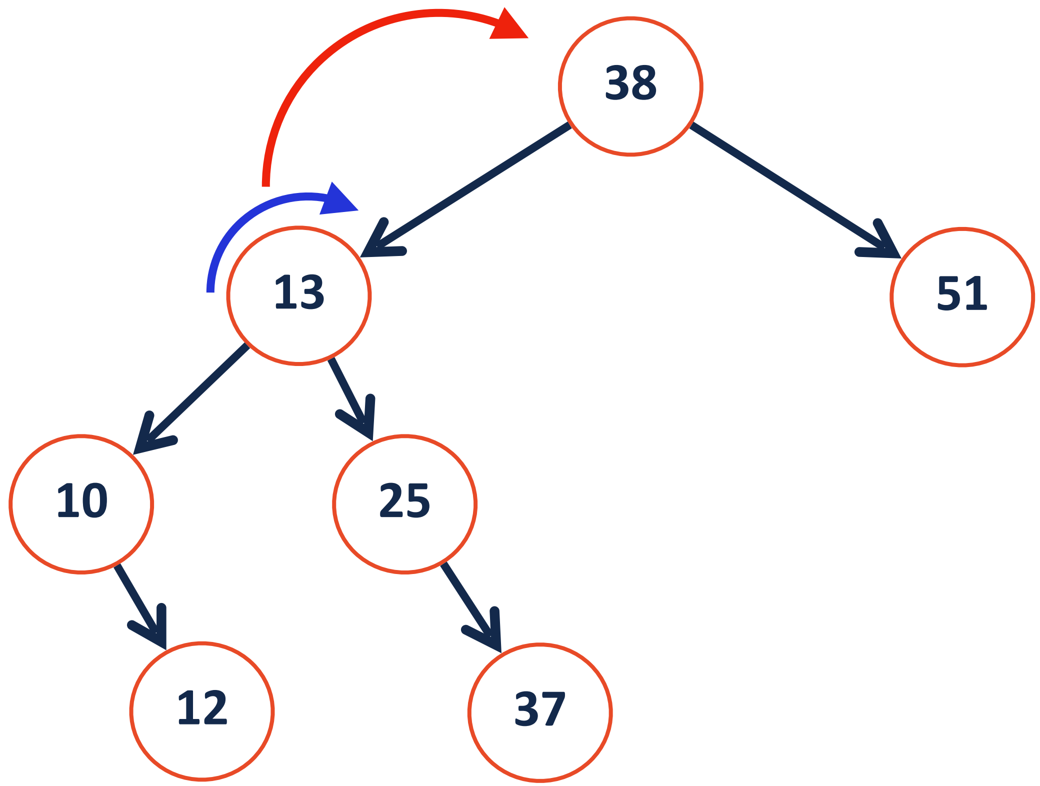
Set the right child's left child to be the current node's right

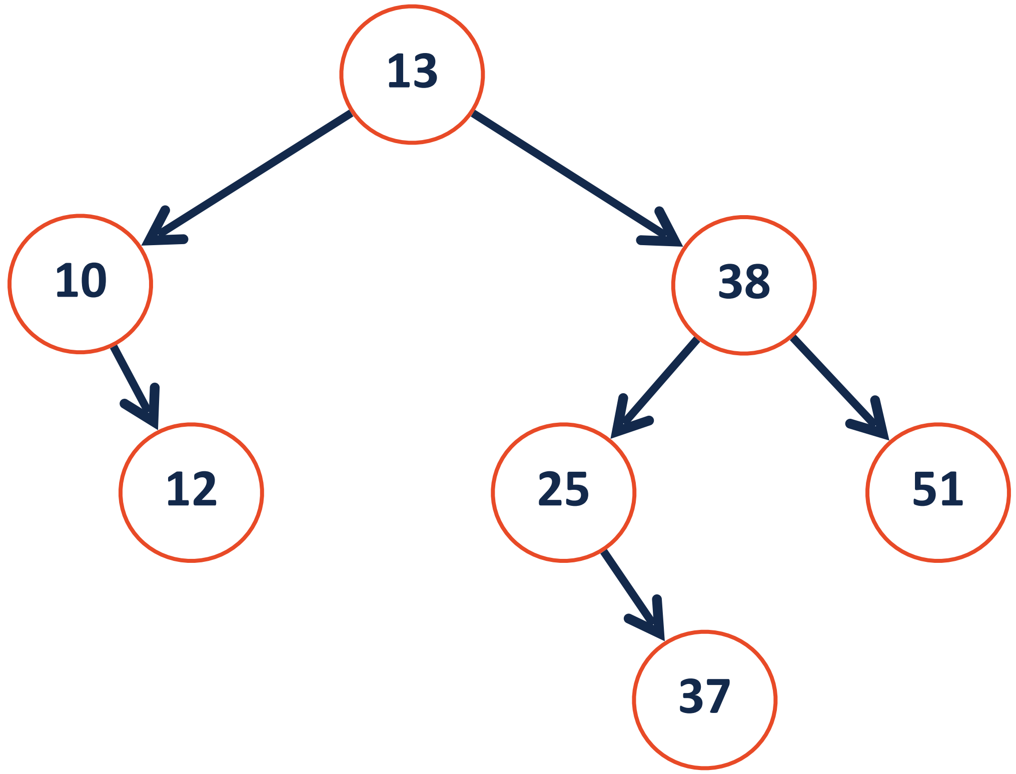
Make the current node the right child's left child

Right Rotation

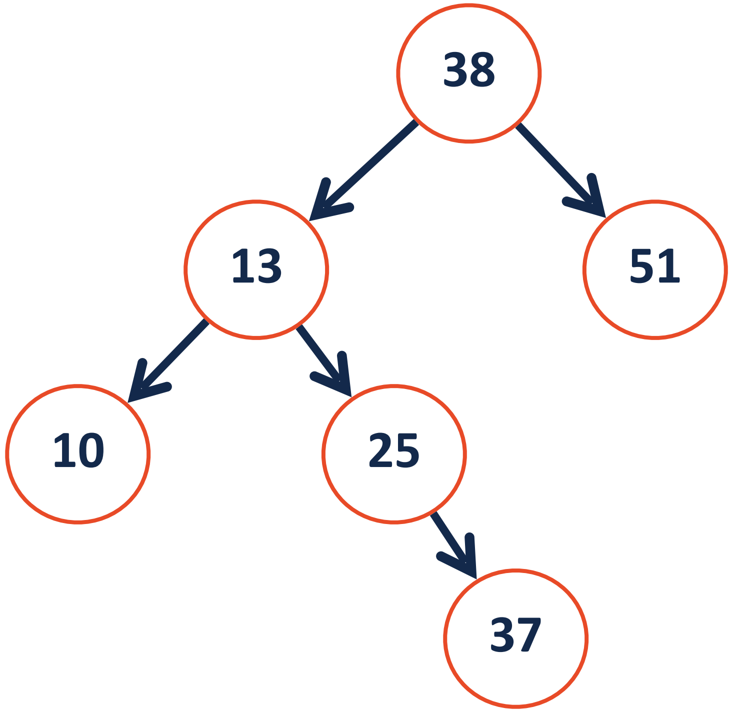


Right Rotation

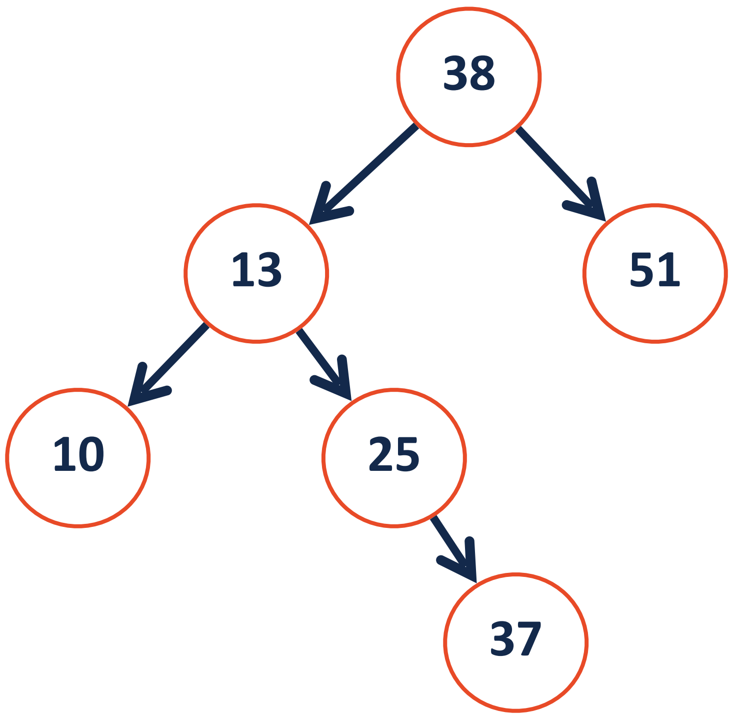


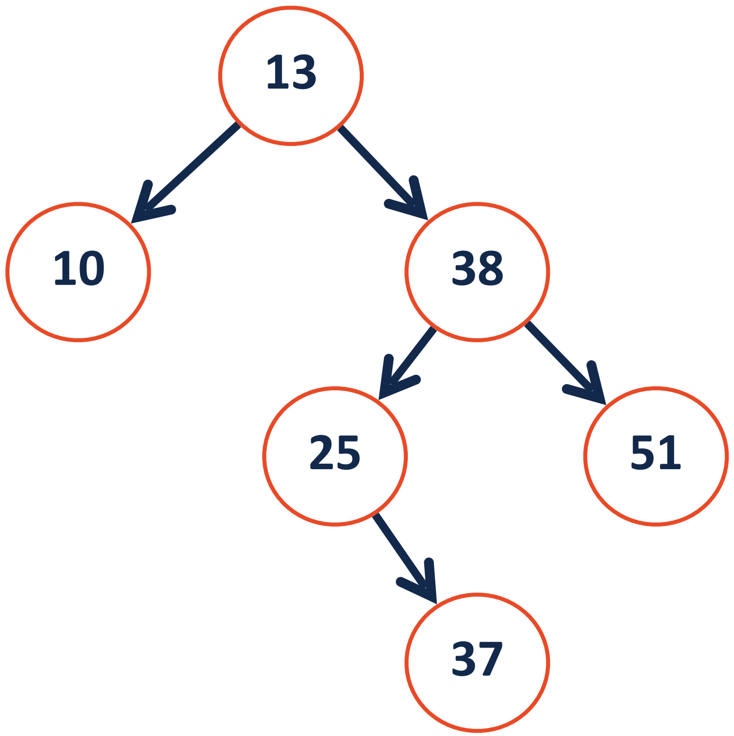


AVL Rotation Practice



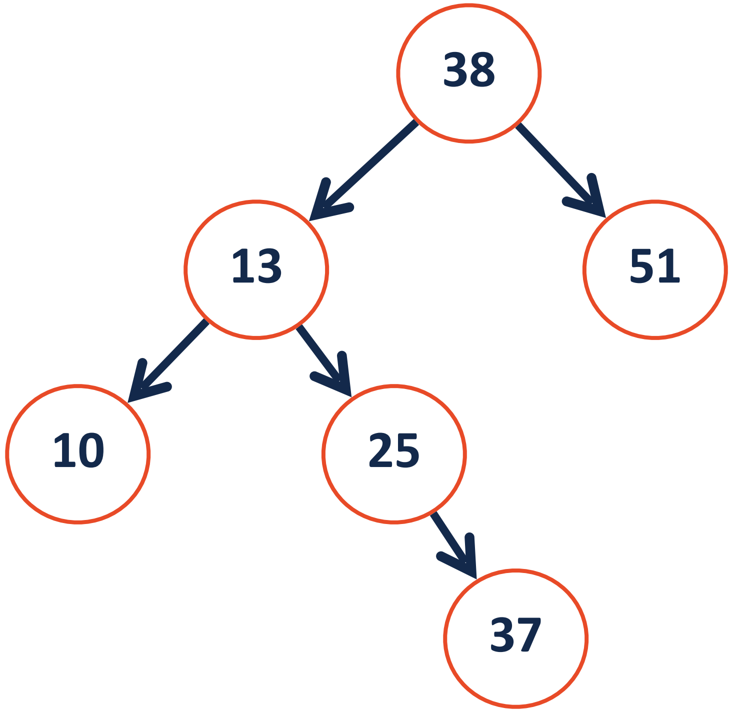
AVL Rotation Practice



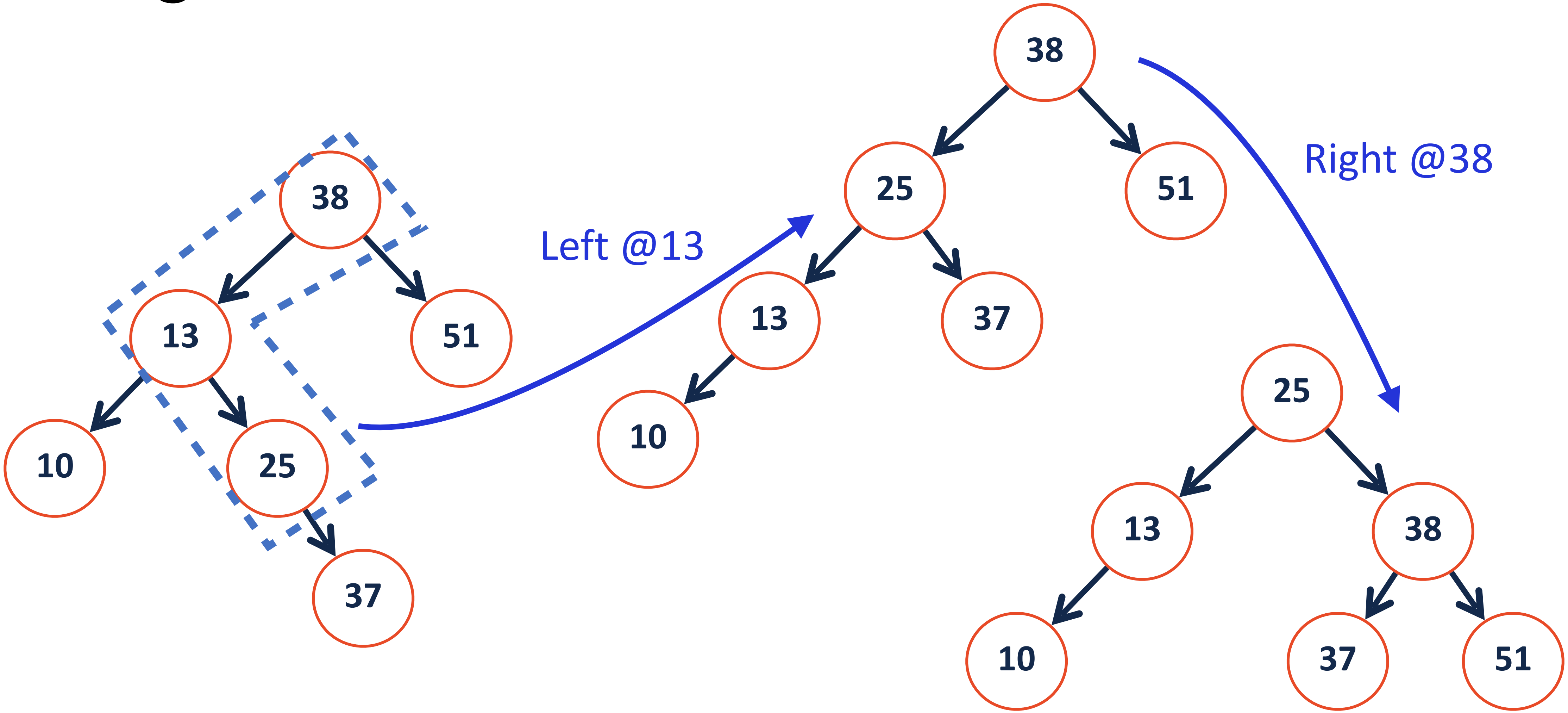


Somethings not quite right…

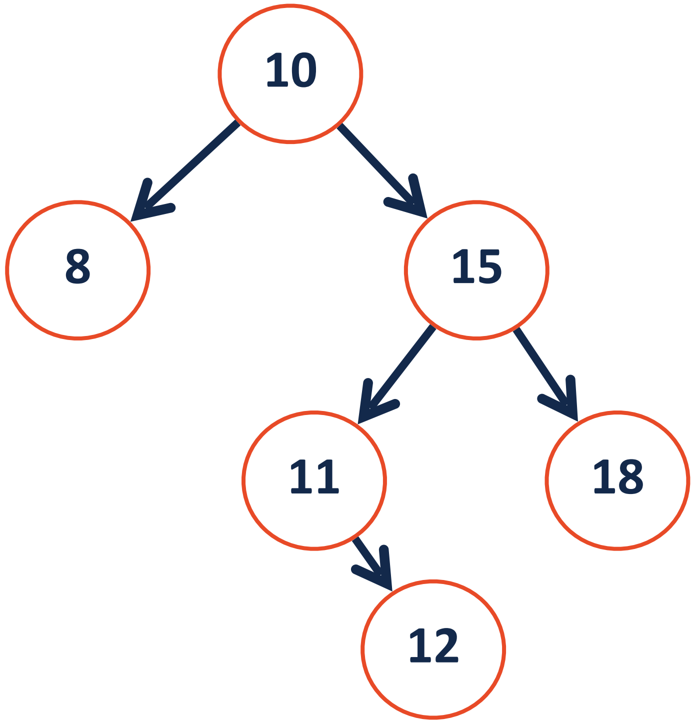
LeftRight Rotation



LeftRight Rotation

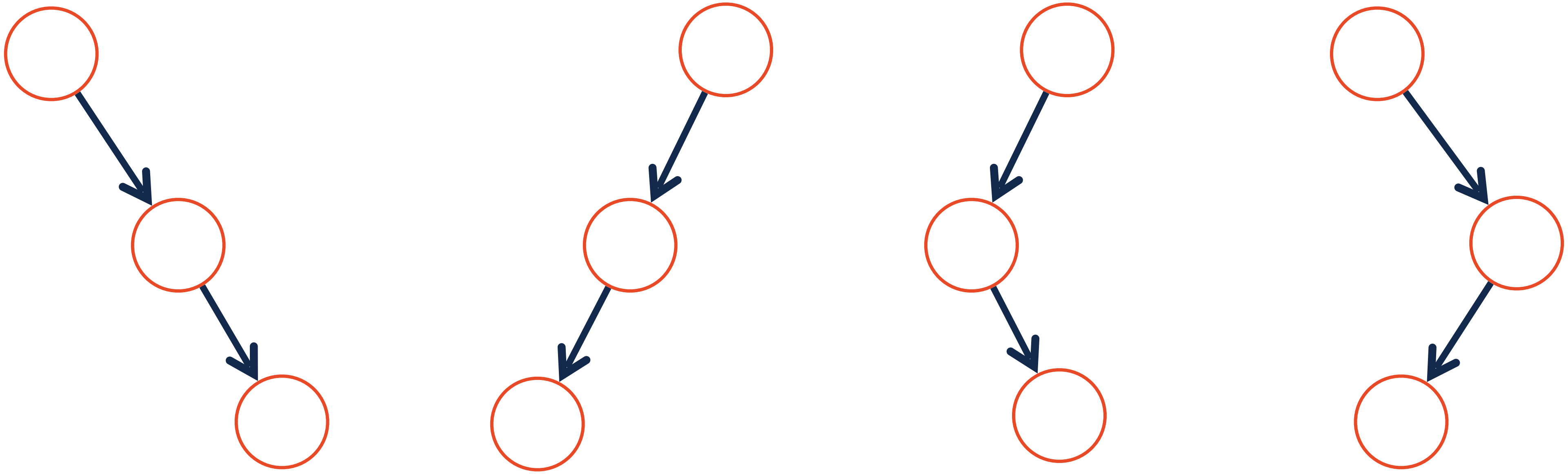


RightLeft Rotation



AVL Rotations

Four kinds of rotations:



AVL Rotations

Four kinds of rotations: (L, R, LR, RL)

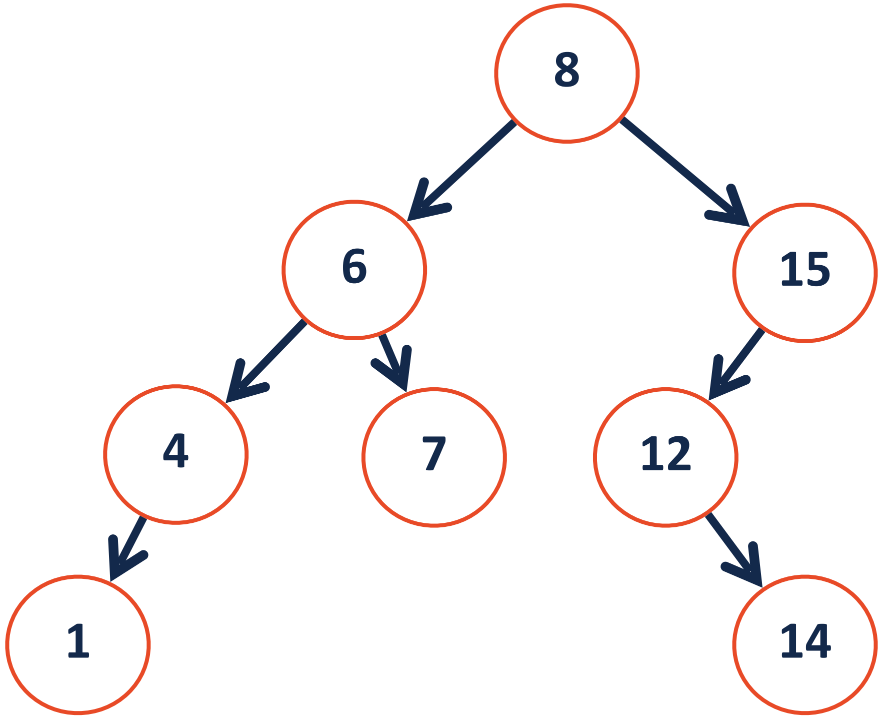
1. All rotations are local (subtrees are not impacted)

2. The running time of rotations are constant

3. The rotations maintain BST property

Goal:

AVL Rotation Practice



AVL vs BST ADT

The AVL tree is a modified binary search tree that rotates **when necessary**

How does the constraint on balance affect the core functions?

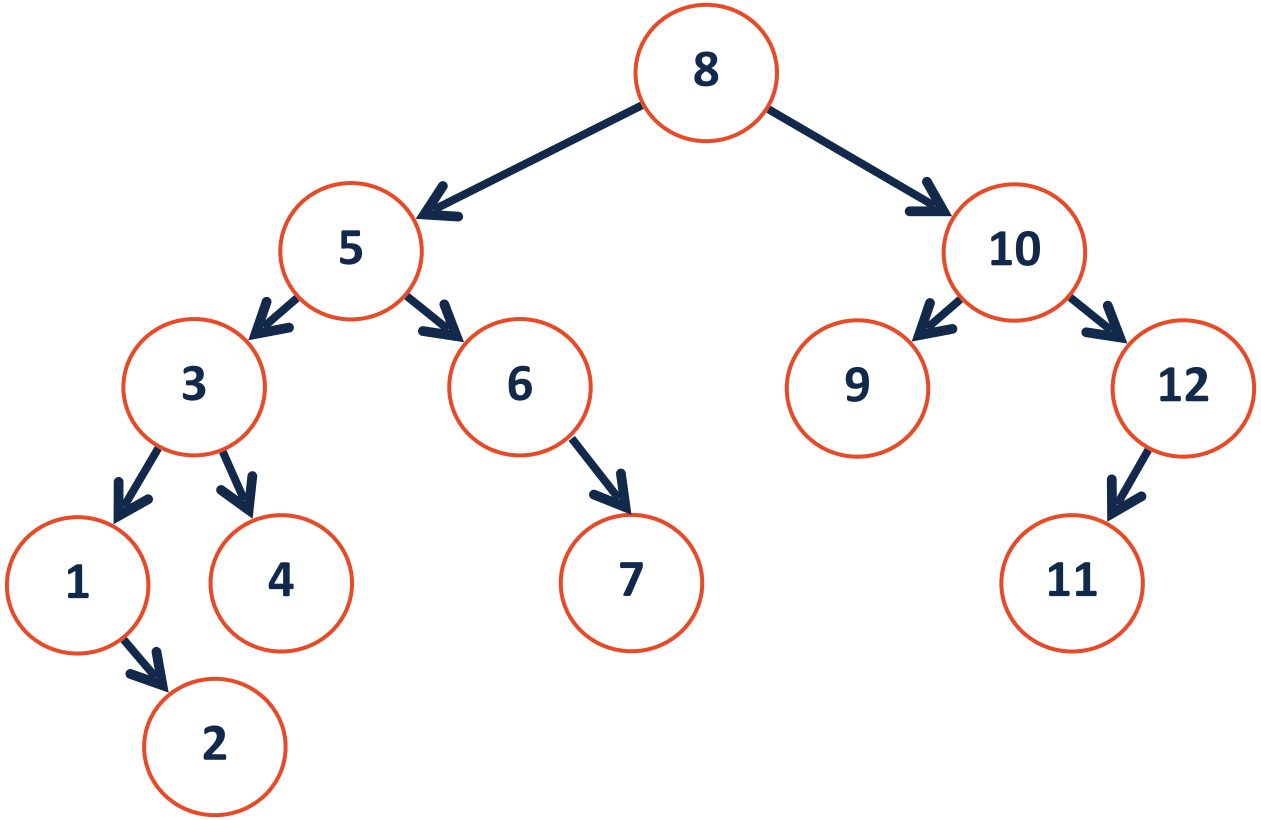
Find

Insert

Remove

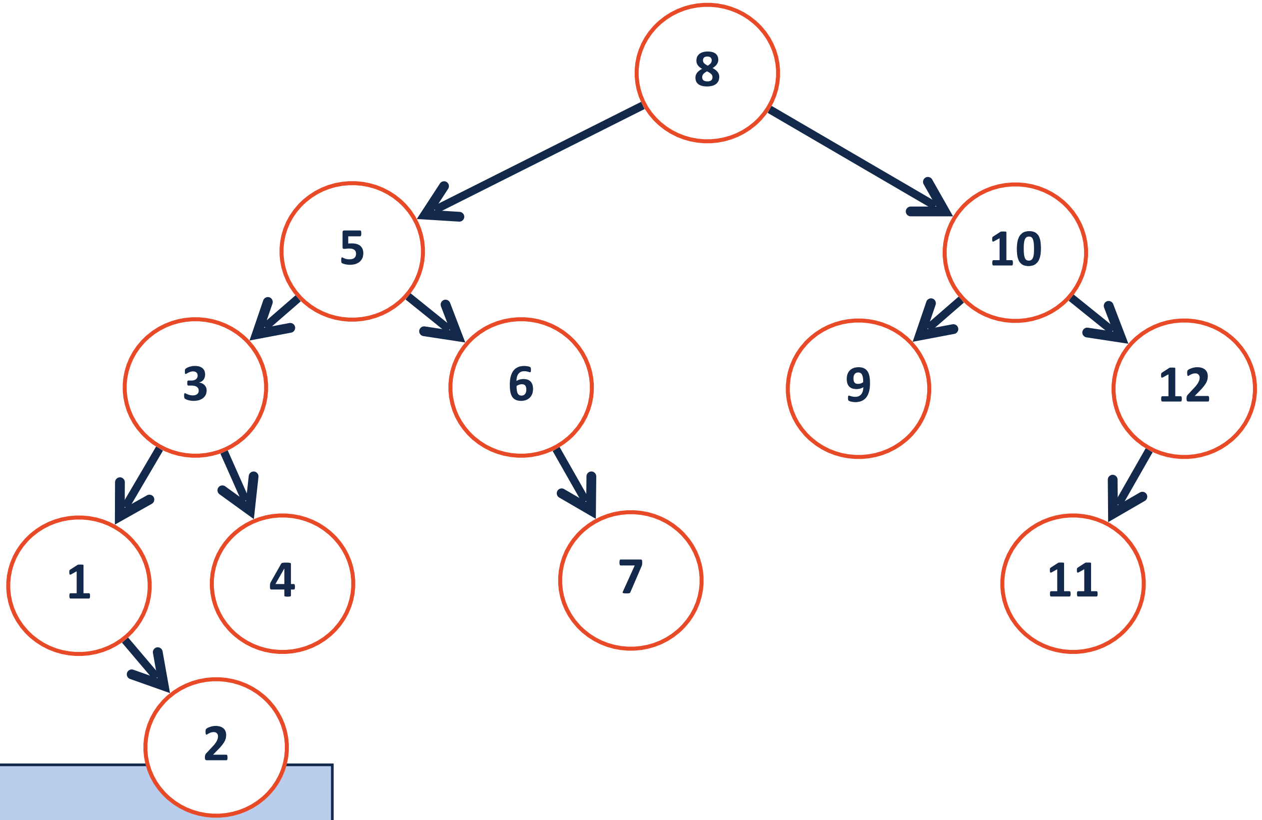
AVL Find

`_find(7)`



AVL Insertion

_insert(6.5)



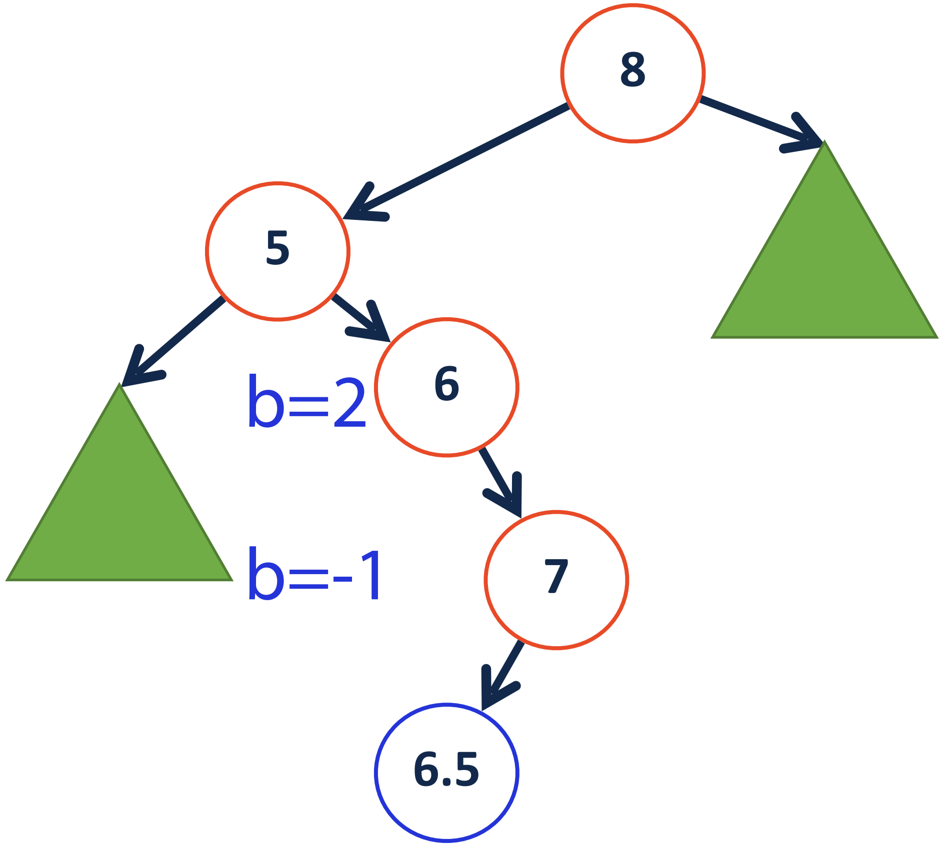

```
def insert_helper(node, key, val):
    if node == None:
        return avlNode(key, val)

    if key < node.key:
        node.left = insert_helper(node.left, key, val)
    else:
        node.right = insert_helper(node.right, key, val)

    return rebalance(node)
```

AVL Insertion

`_insert(6.5)`

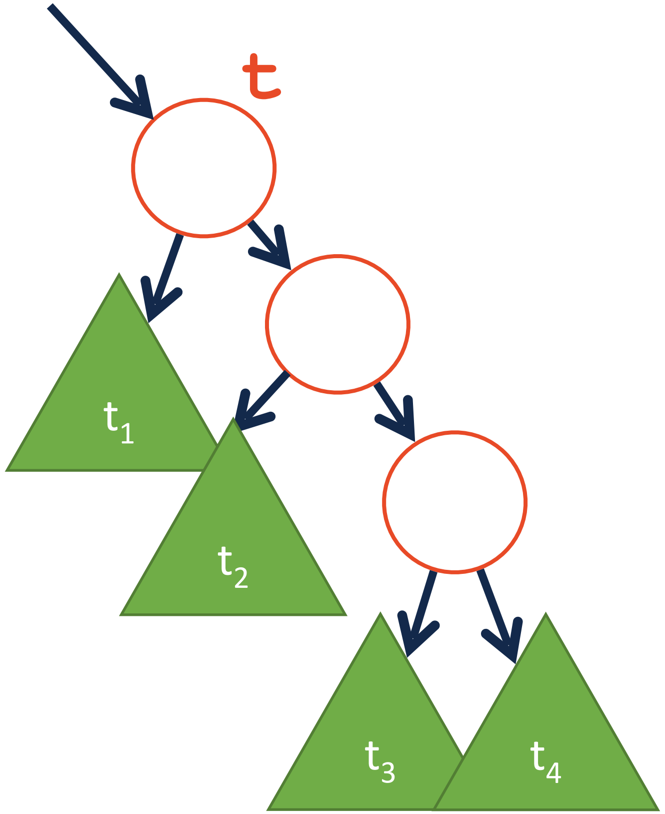



```
def insert_helper(node, key, val):

    ...

    return rebalance(node)
```

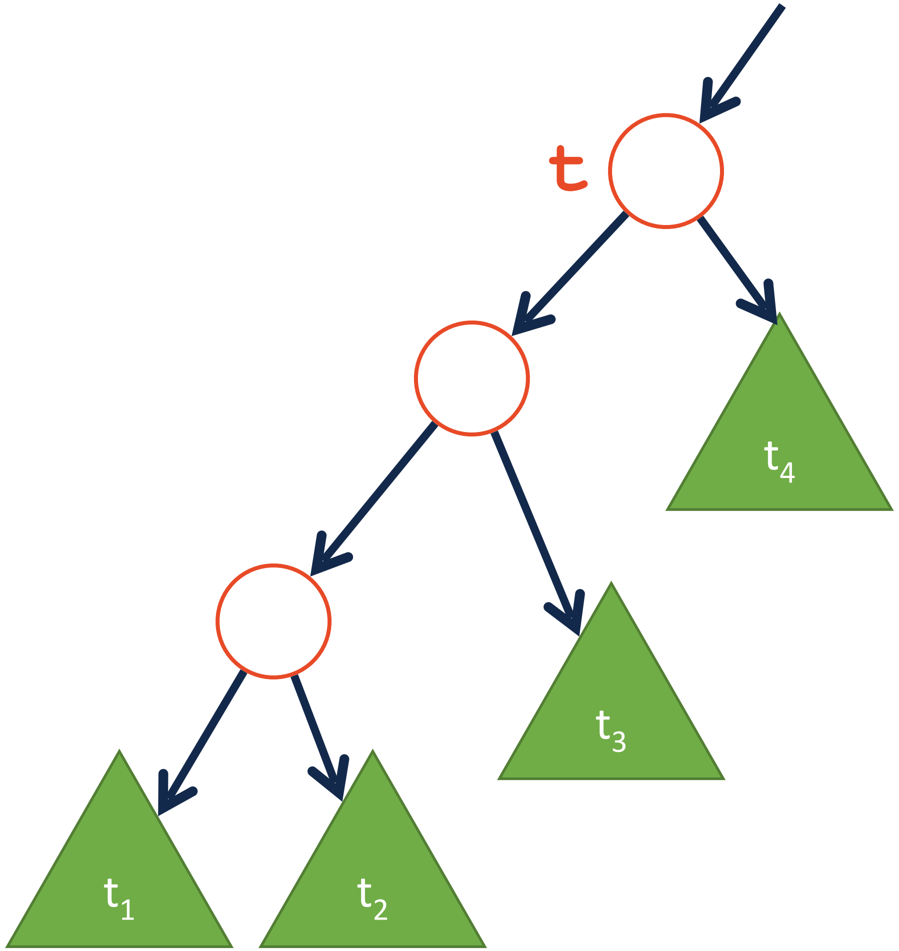
Rebalancing on insert



Theorem:
If an insertion occurred in subtrees $\mathbf{t_3}$ or $\mathbf{t_4}$ and an imbalance was first detected at **t**, then a _______________ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** _______ and the balance factor of **t->right** is _______.

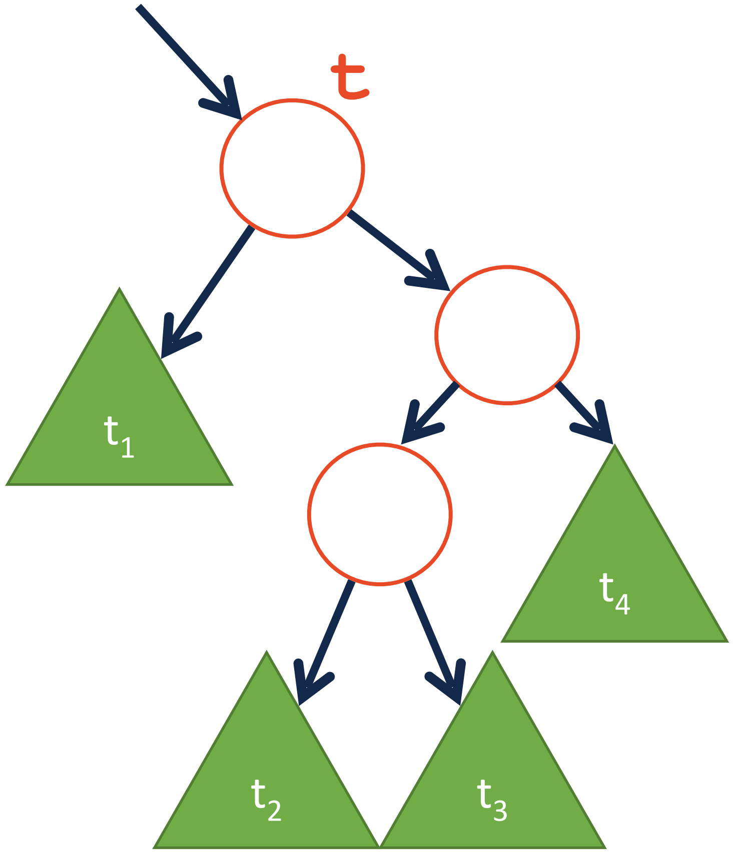
Rebalancing on insert



Theorem:
If an insertion occurred in subtrees $\mathbf{t_1}$ or $\mathbf{t_2}$ and an imbalance was first detected at **t**, then a _______________ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** _______ and the balance factor of **t->left** is _______.

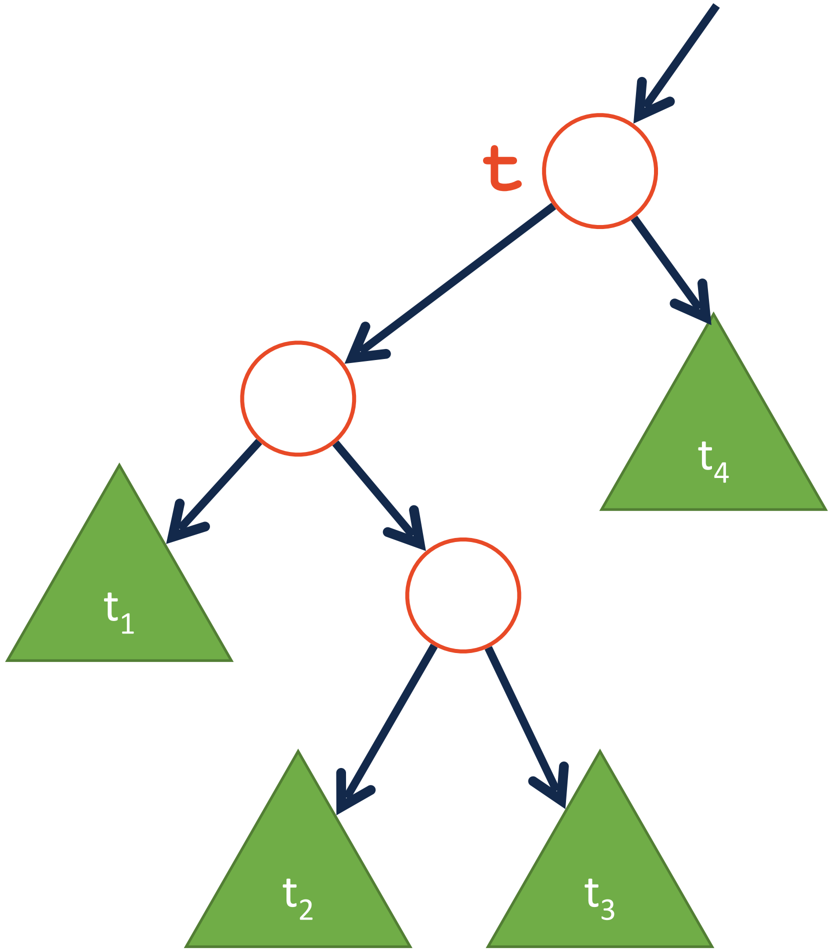
Rebalancing on insert



Theorem:
If an insertion occurred in subtrees $\mathbf{t_2}$ or $\mathbf{t_3}$ and an imbalance was first detected at **t**, then a _______________ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** _______ and the balance factor of **t->right** is _______.

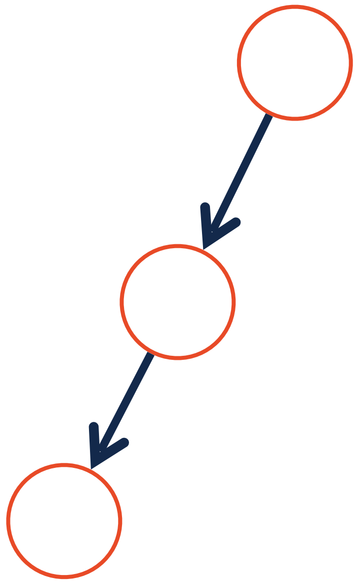
Rebalancing on insert

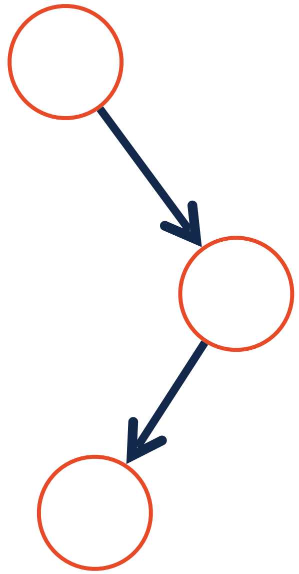


Theorem:
If an insertion occurred in subtrees $\mathbf{t_2}$ or $\mathbf{t_3}$ and an imbalance was first detected at **t**, then a _______________ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** _______ and the balance factor of **t->left** is _______.

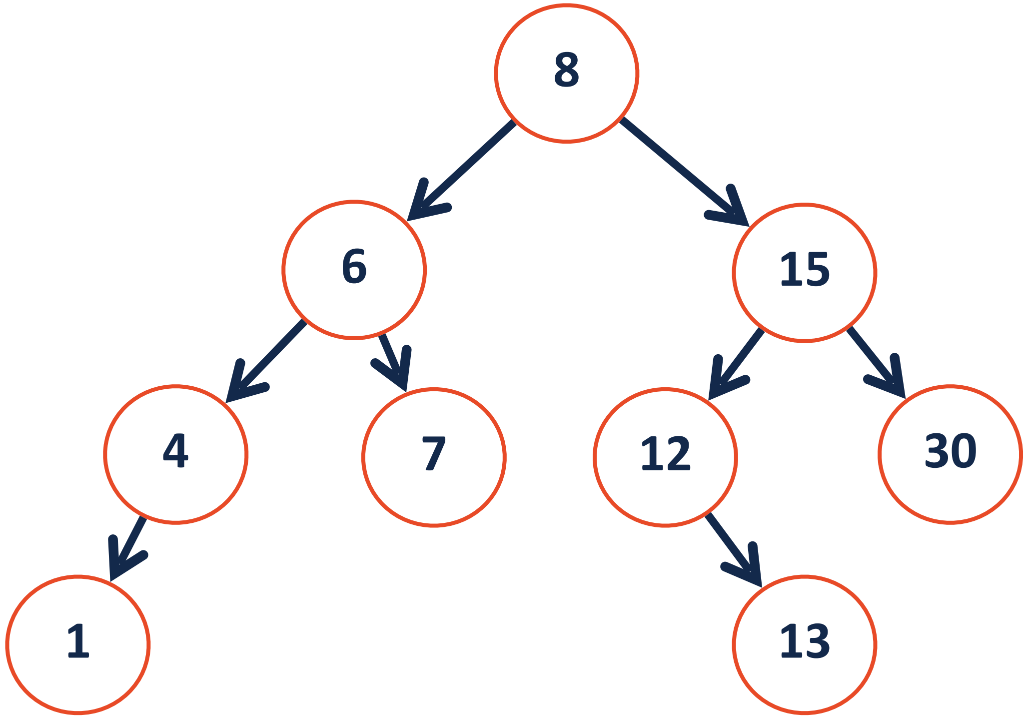
Rebalancing on insert





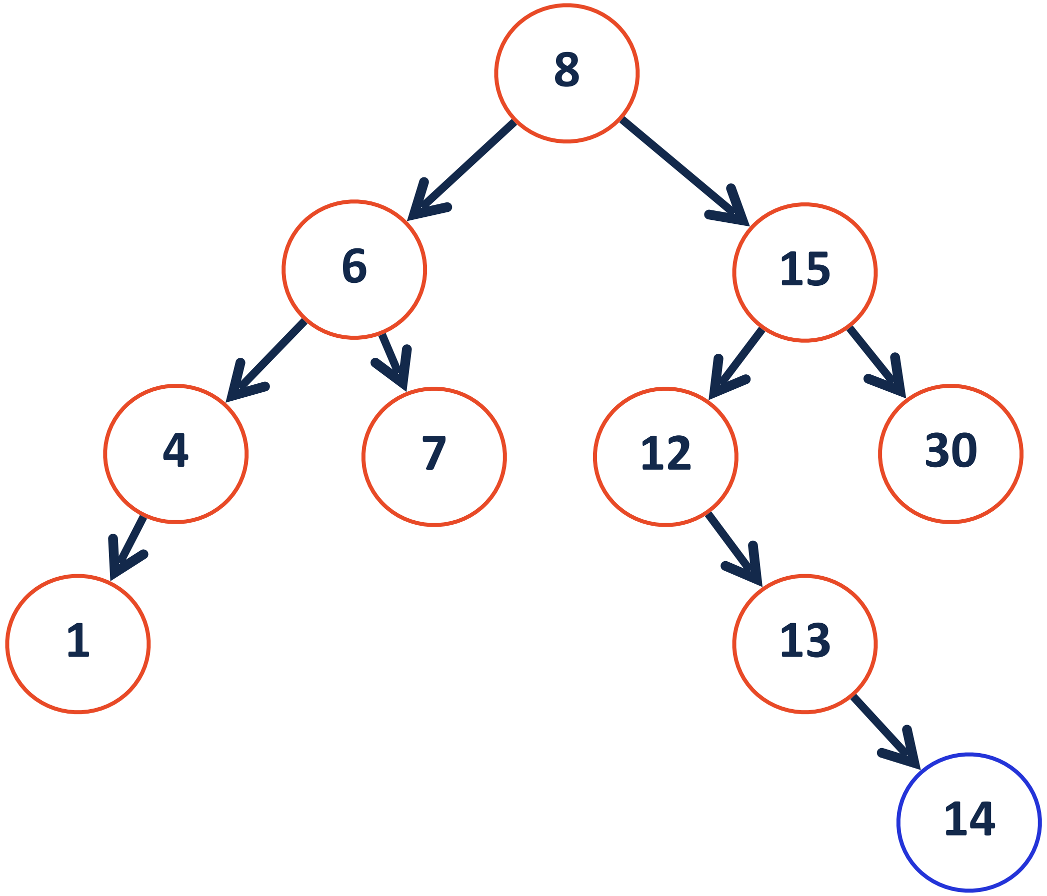
AVL Insertion Practice

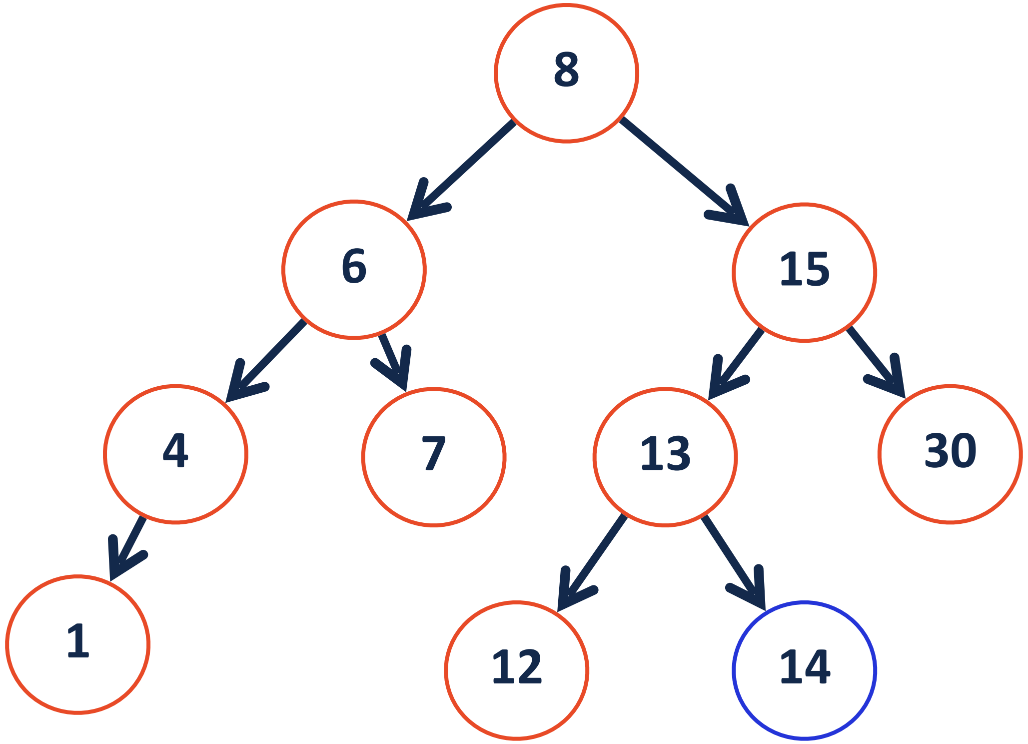
`_insert(14)`



AVL Insertion Practice

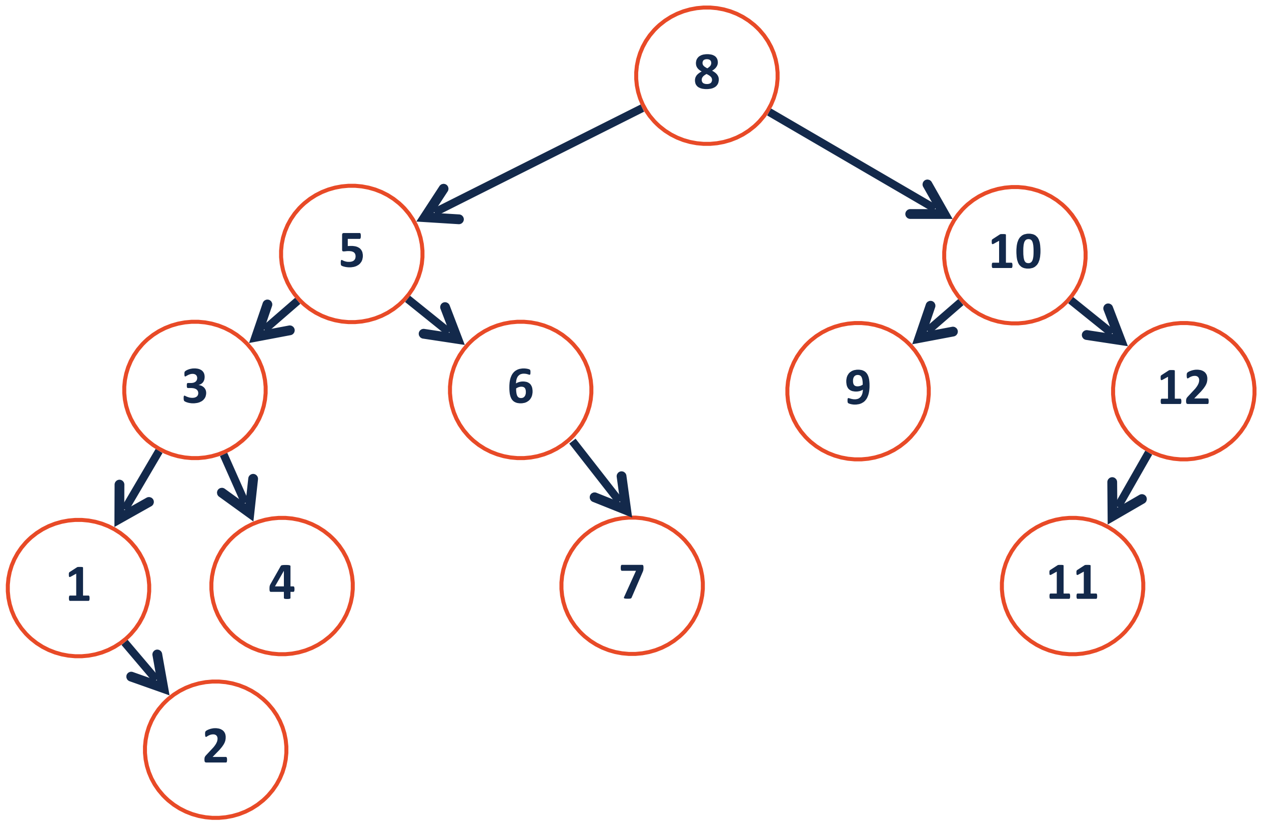
`_insert(14)`





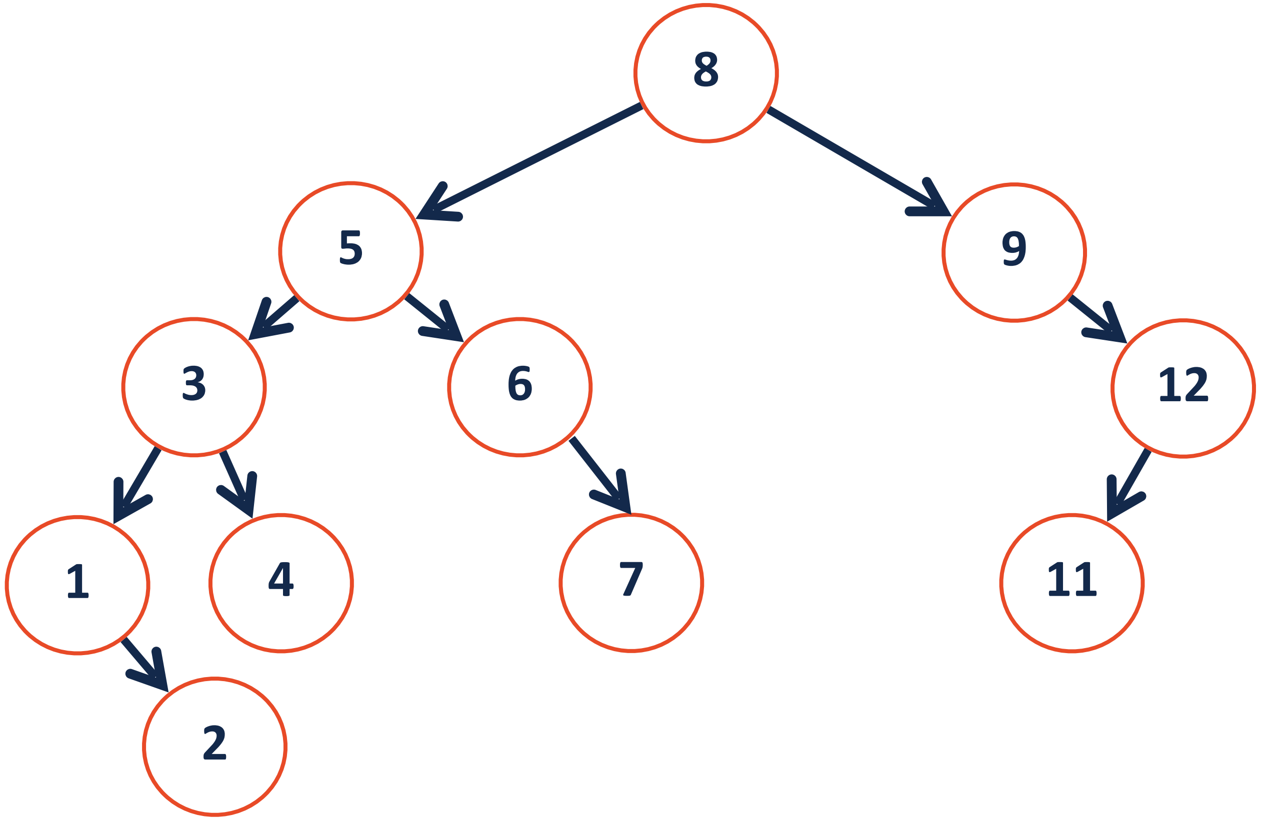
AVL Remove

`_remove(10)`



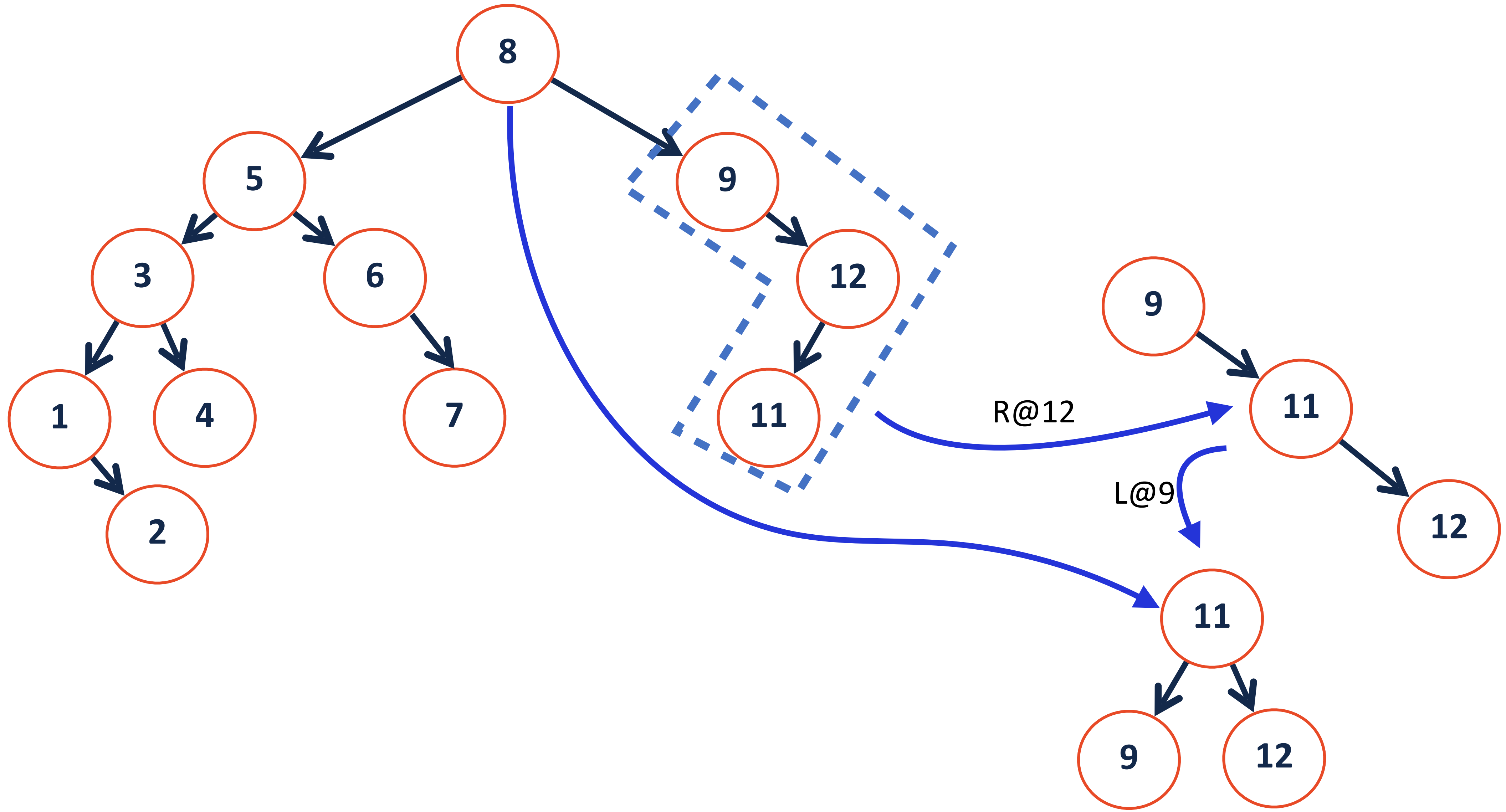
AVL Remove

`_remove(10)`



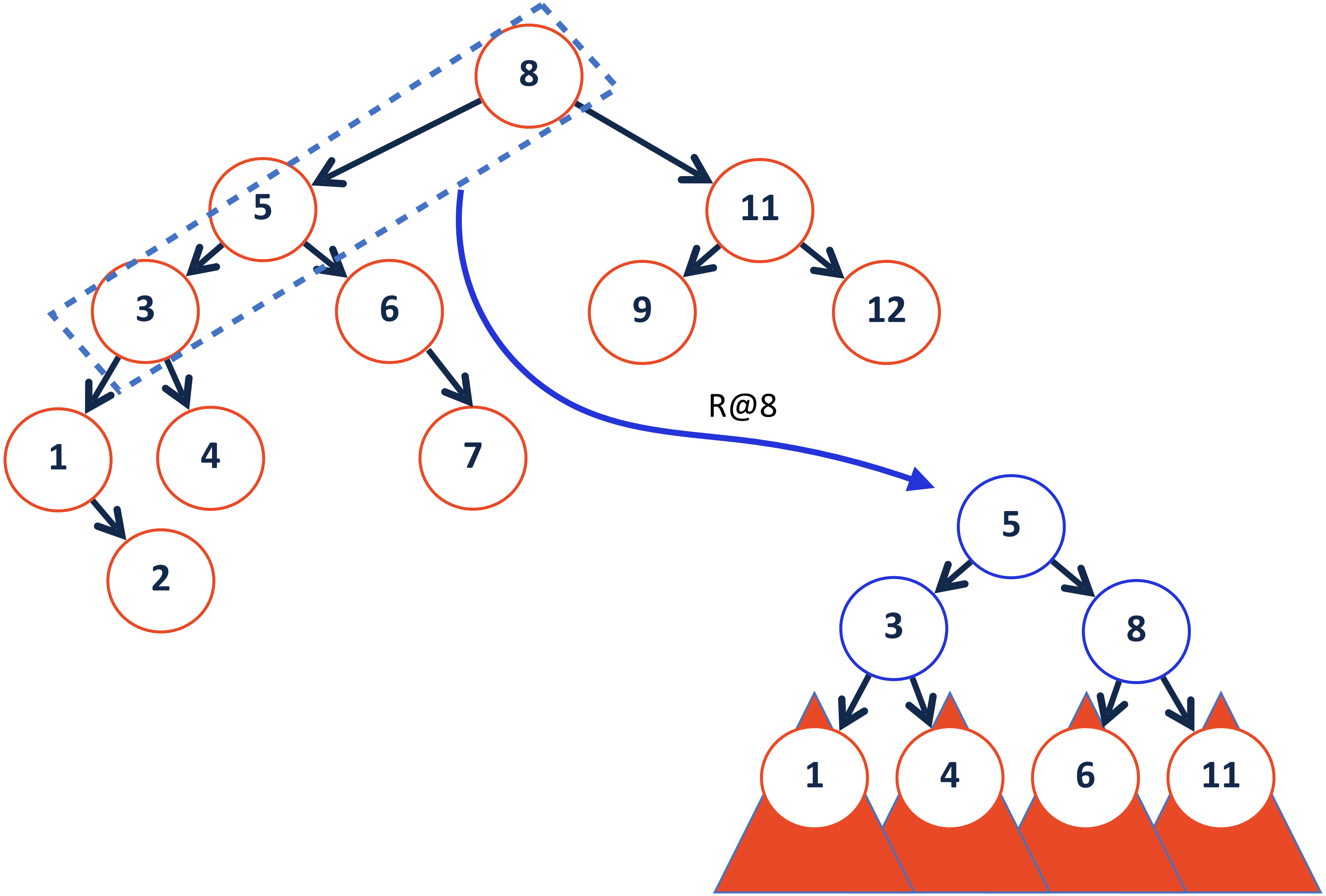
AVL Remove

`_remove(10)`



AVL Remove

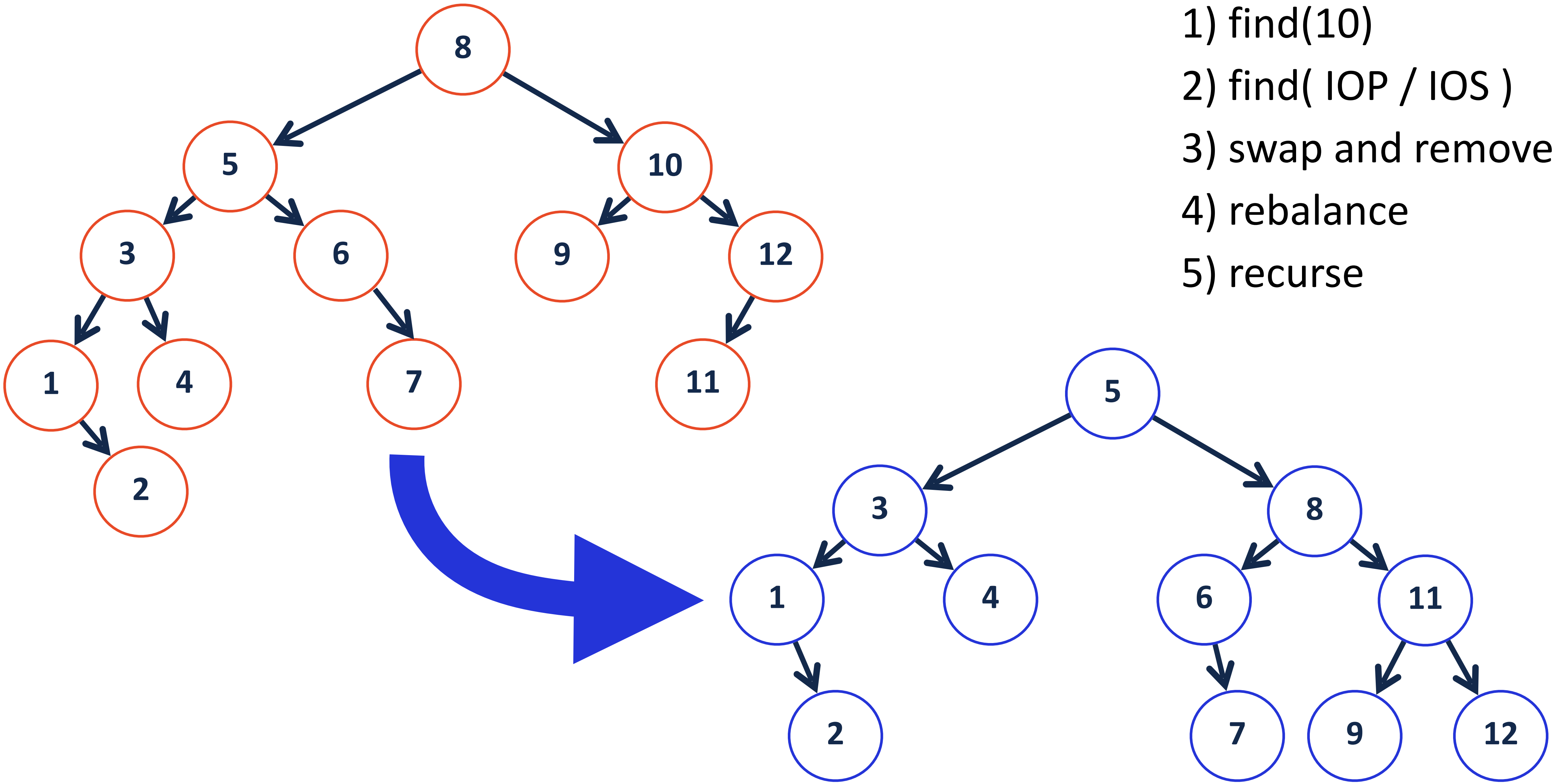
`_remove(10)`



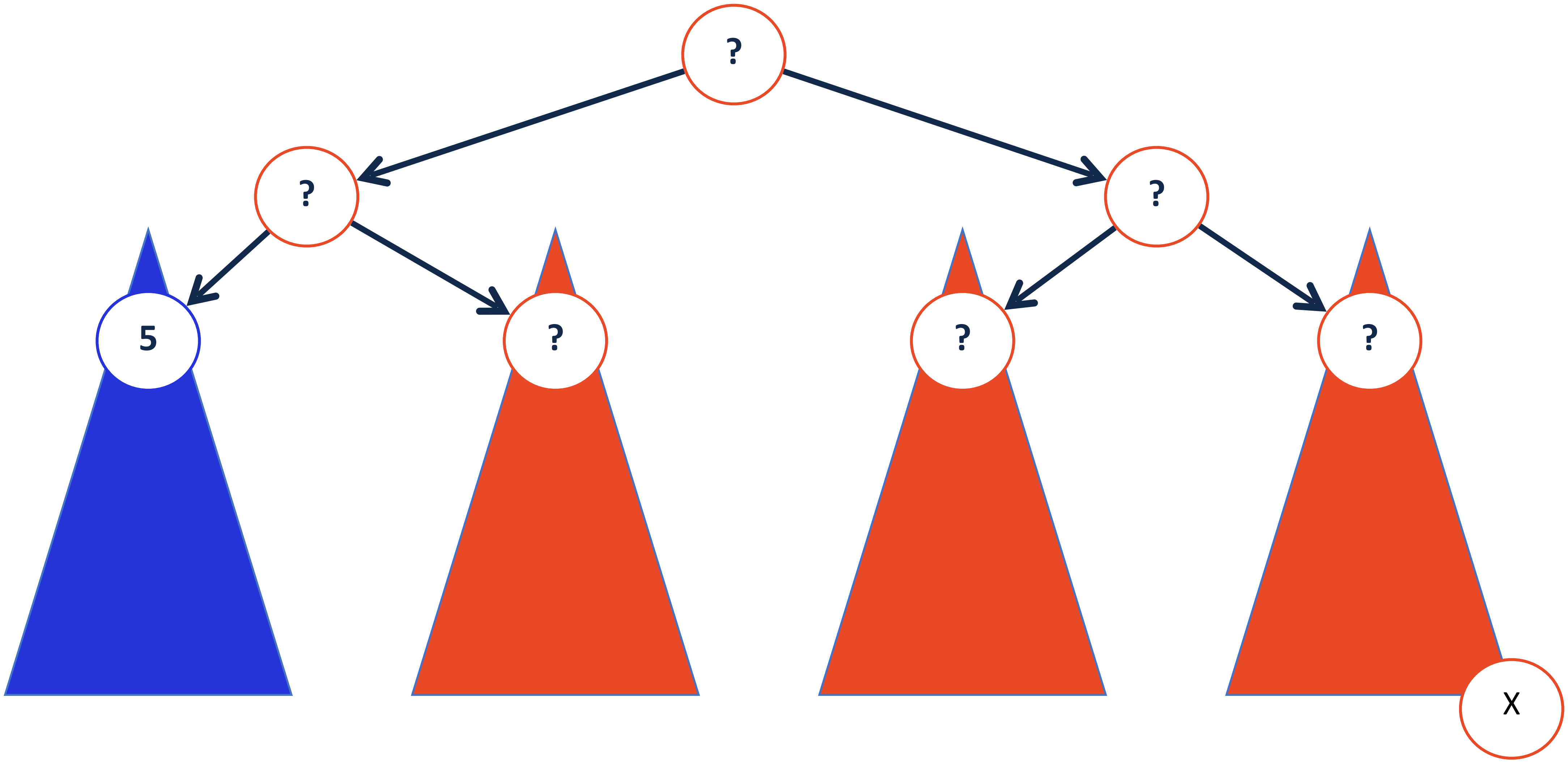
AVL Remove

`_remove(10)`

1) find(10)
2) find(IOP / IOS)
3) swap and remove
4) rebalance
5) recurse



AVL Remove



AVL Tree Analysis

For AVL tree of height h, we know:

find runs in: ____________.

insert runs in: ____________.

remove runs in: ____________.

We will argue that: h is ___________.