Data Structures
AVL Analysis

CS 225
Brad Solomon

October 5, 2025

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

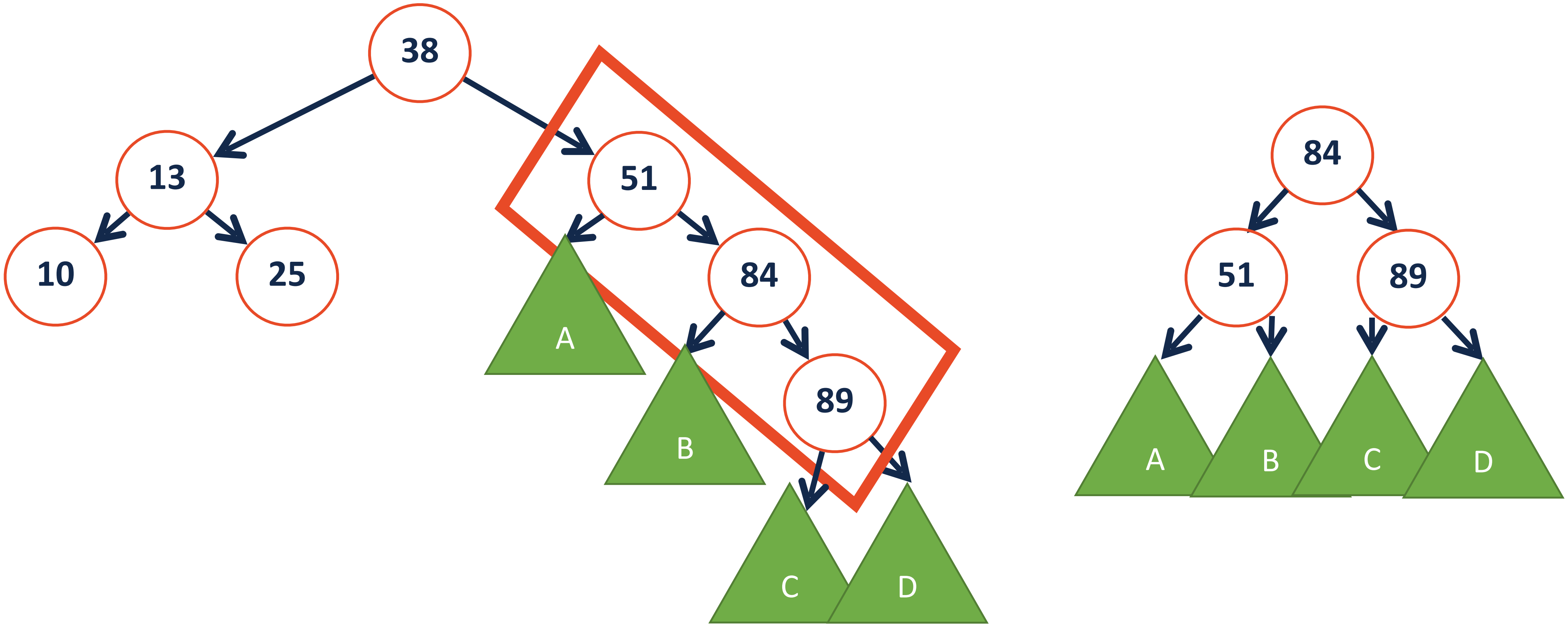
Learning Objectives

Review AVL trees

Prove that the AVL Tree speeds up all operations

AVL Tree

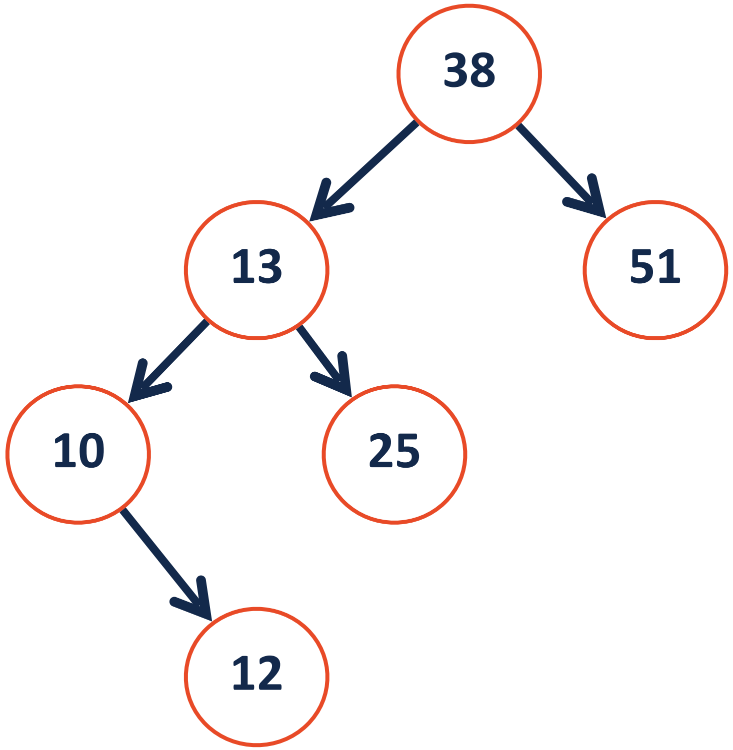
The AVL tree is a modified binary search tree that is **balanced**

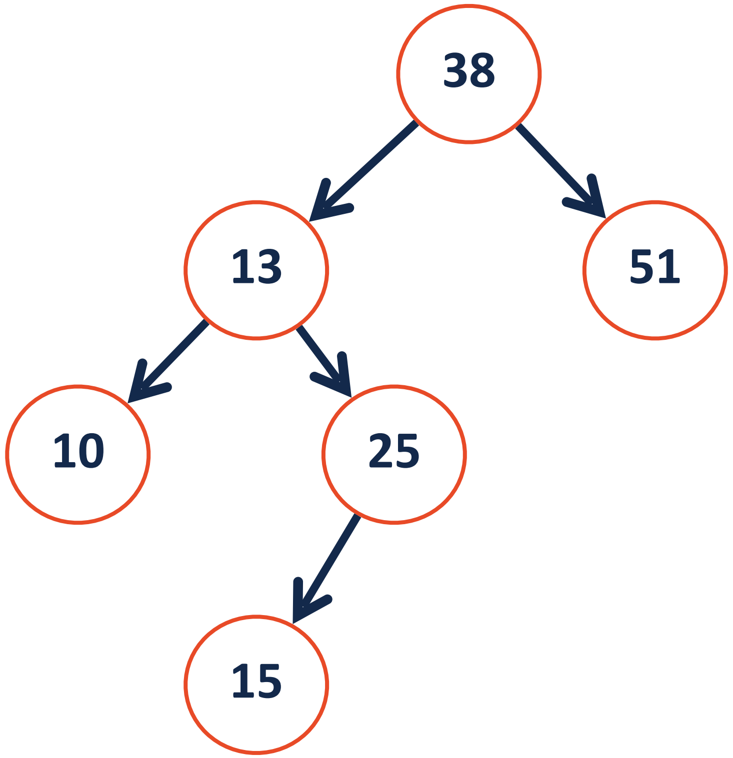


Height balance: $b = height(T_R) - height(T_L)$

AVL Rotations

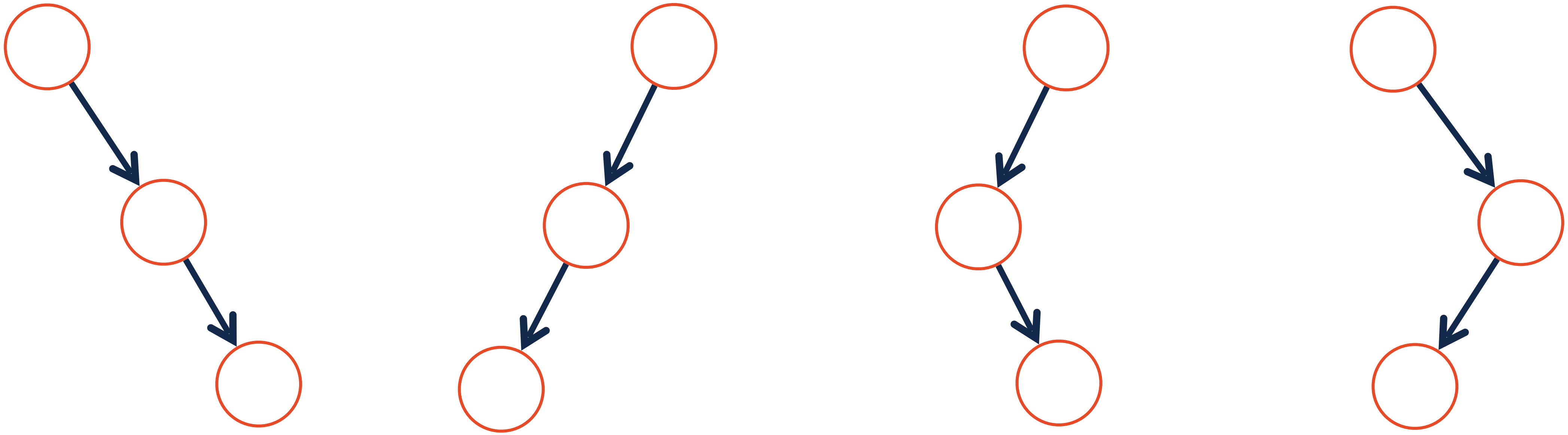
We can identify which rotation to do using **balance**





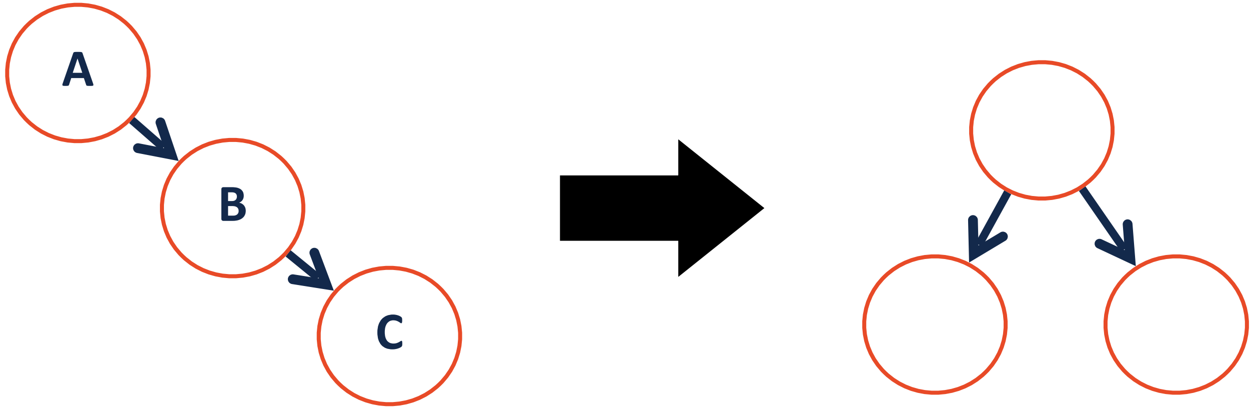
AVL Rotations

	Left	Right	LeftRight	RightLeft



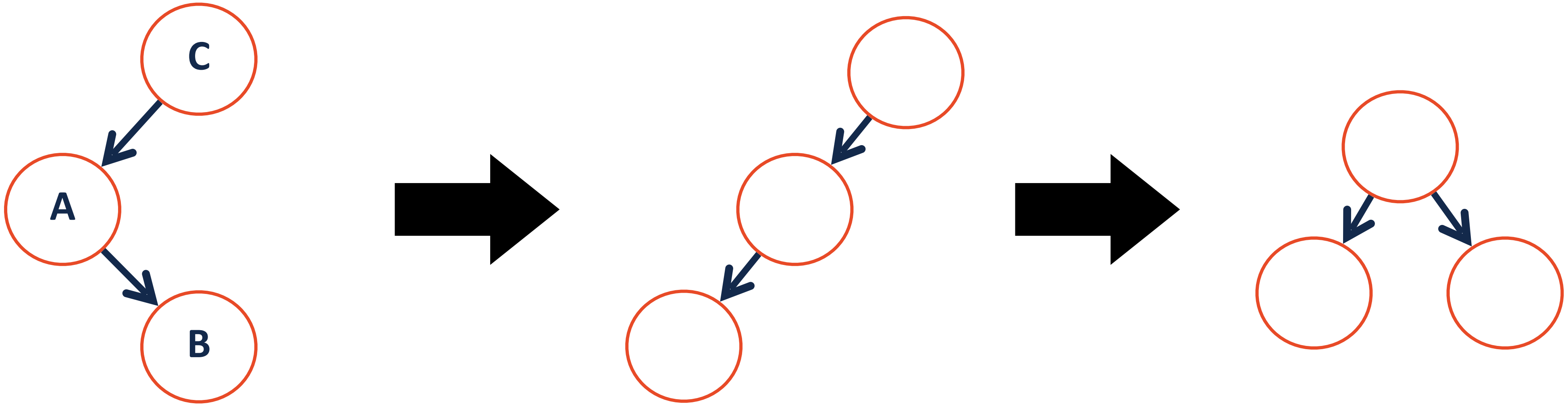
Root Balance:	2	-2	-2	2
Child Balance:	1	-1	1	-1

AVL Tree Rotations

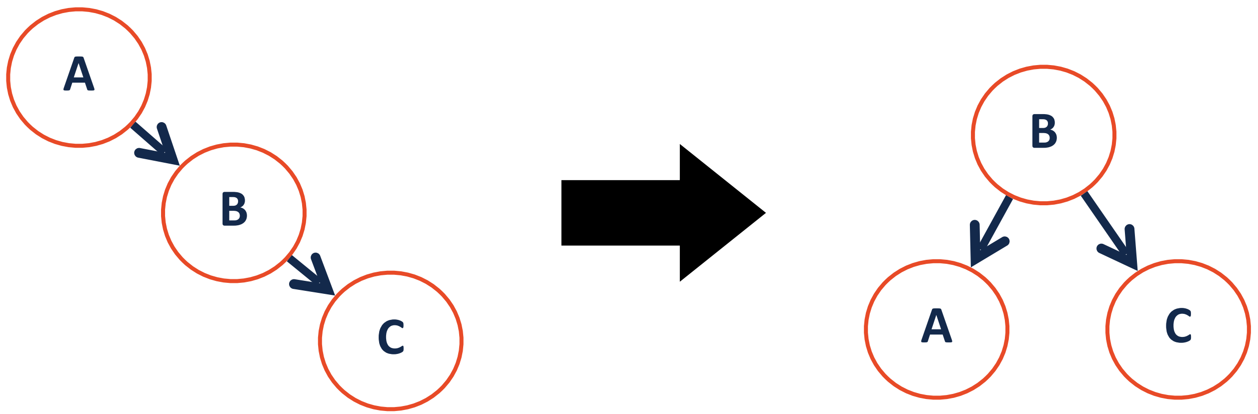


All rotations are O(1)

All rotations reduce subtree height by one

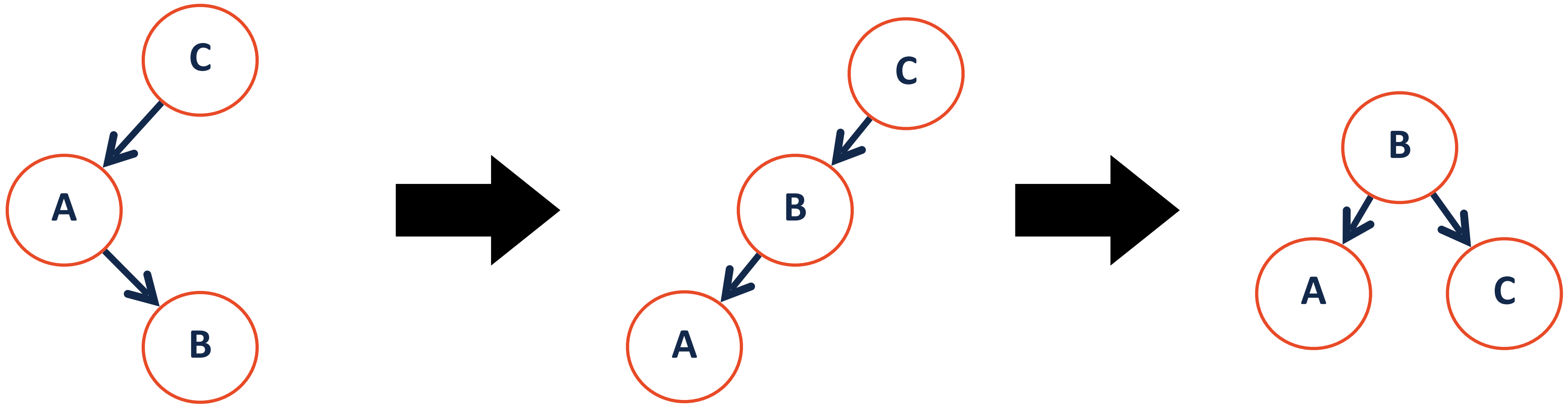


AVL Tree Rotations



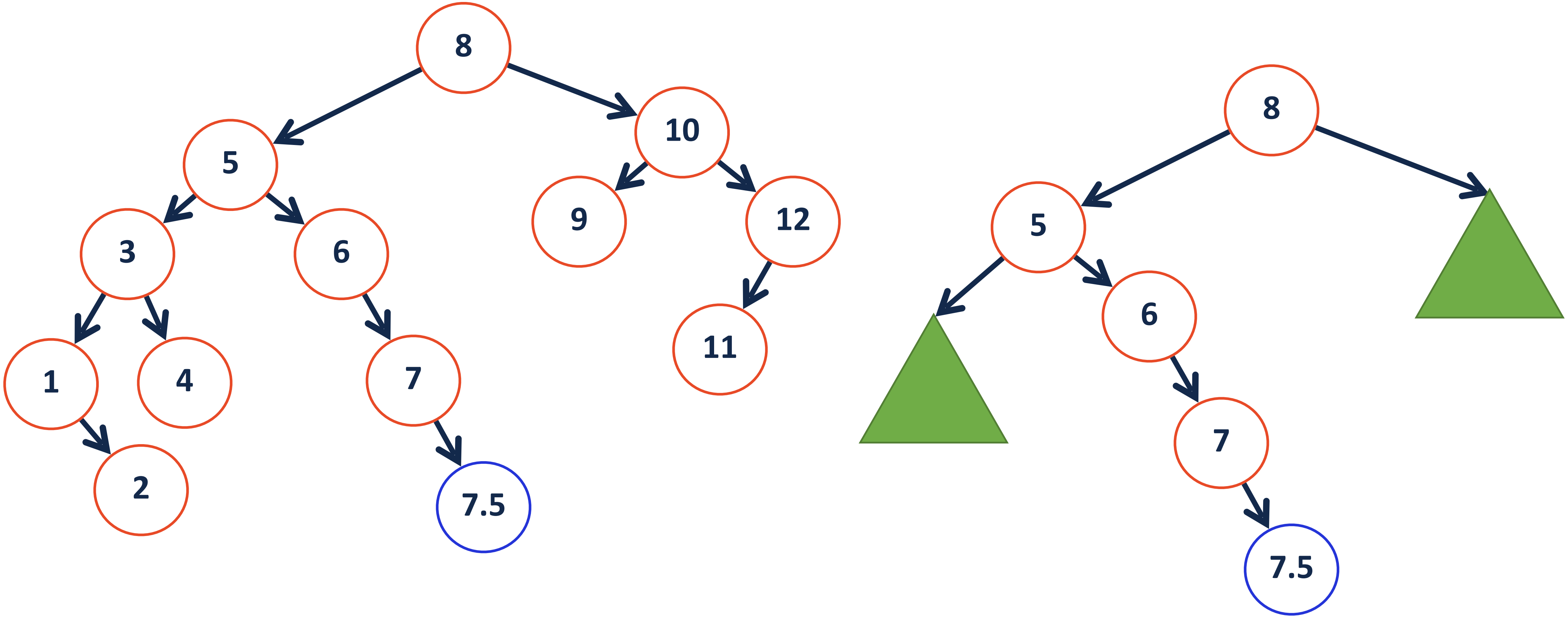
All rotations are O(1)

All rotations reduce subtree height by one



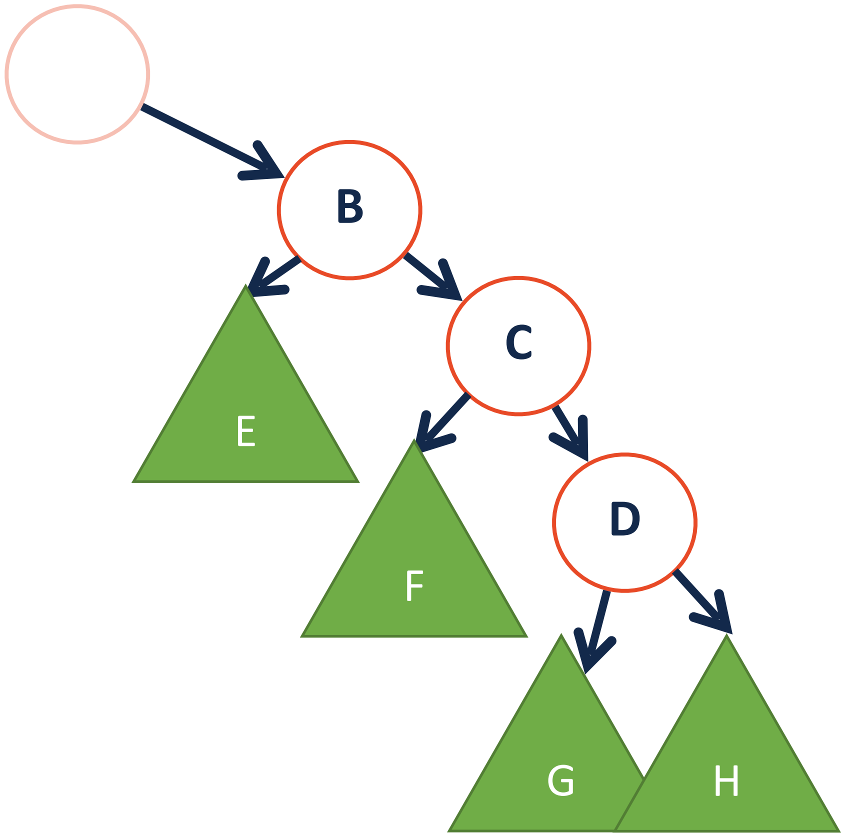
AVL Tree Rotations

All rotations are local (subtrees are not impacted)



AVL Tree Rotations

All rotations preserve BST property



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 vs BST ADT

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

```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

How does the constraint on balance affect the core functions?

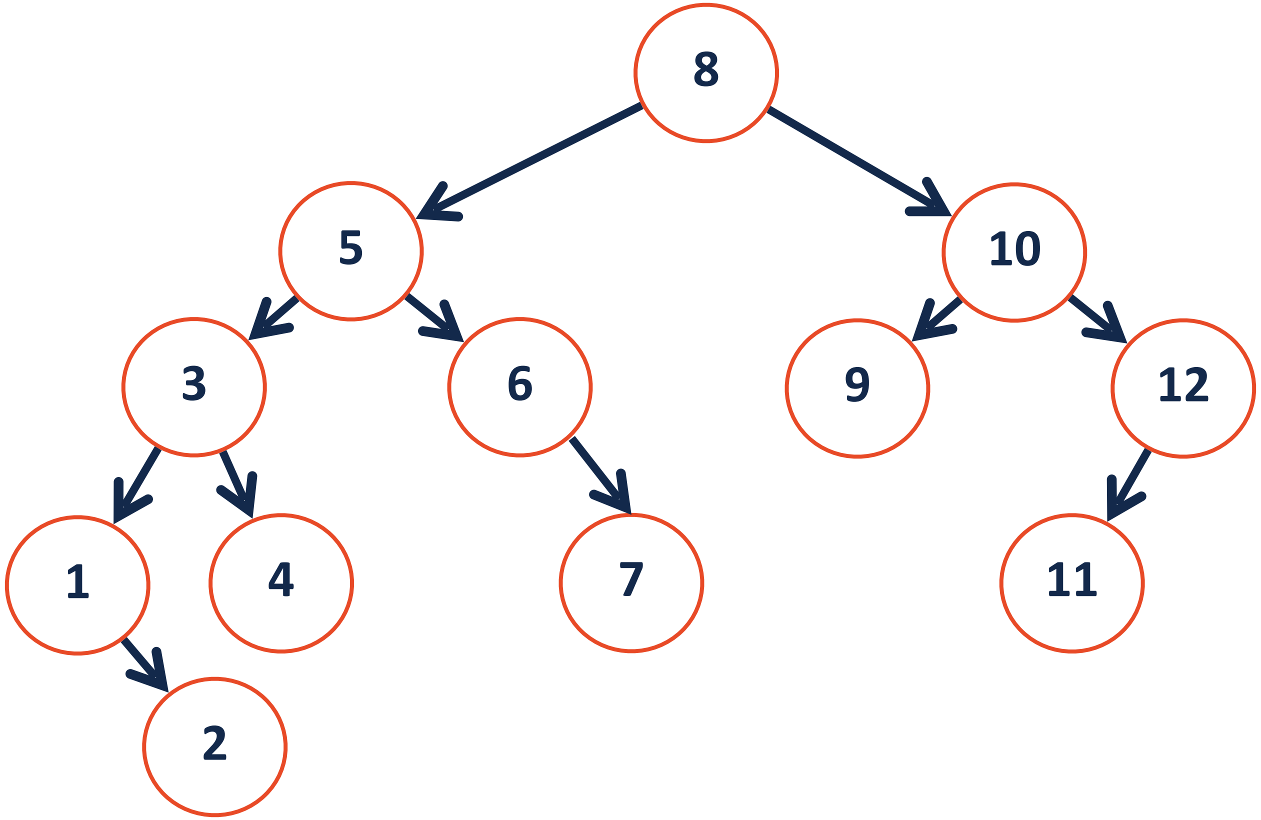
Find

Insert

Remove

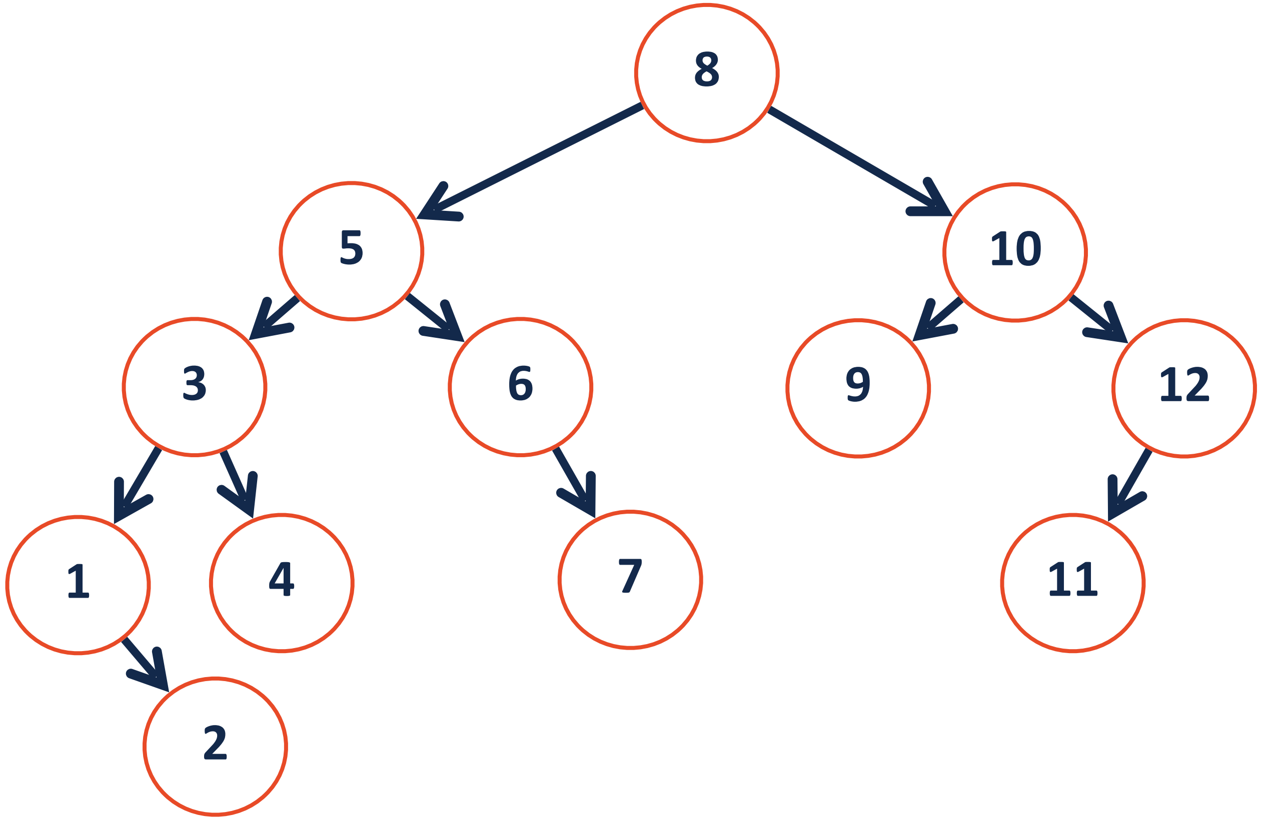
AVL Find

`_find(7)`



AVL Insertion

`_insert(6.5)`



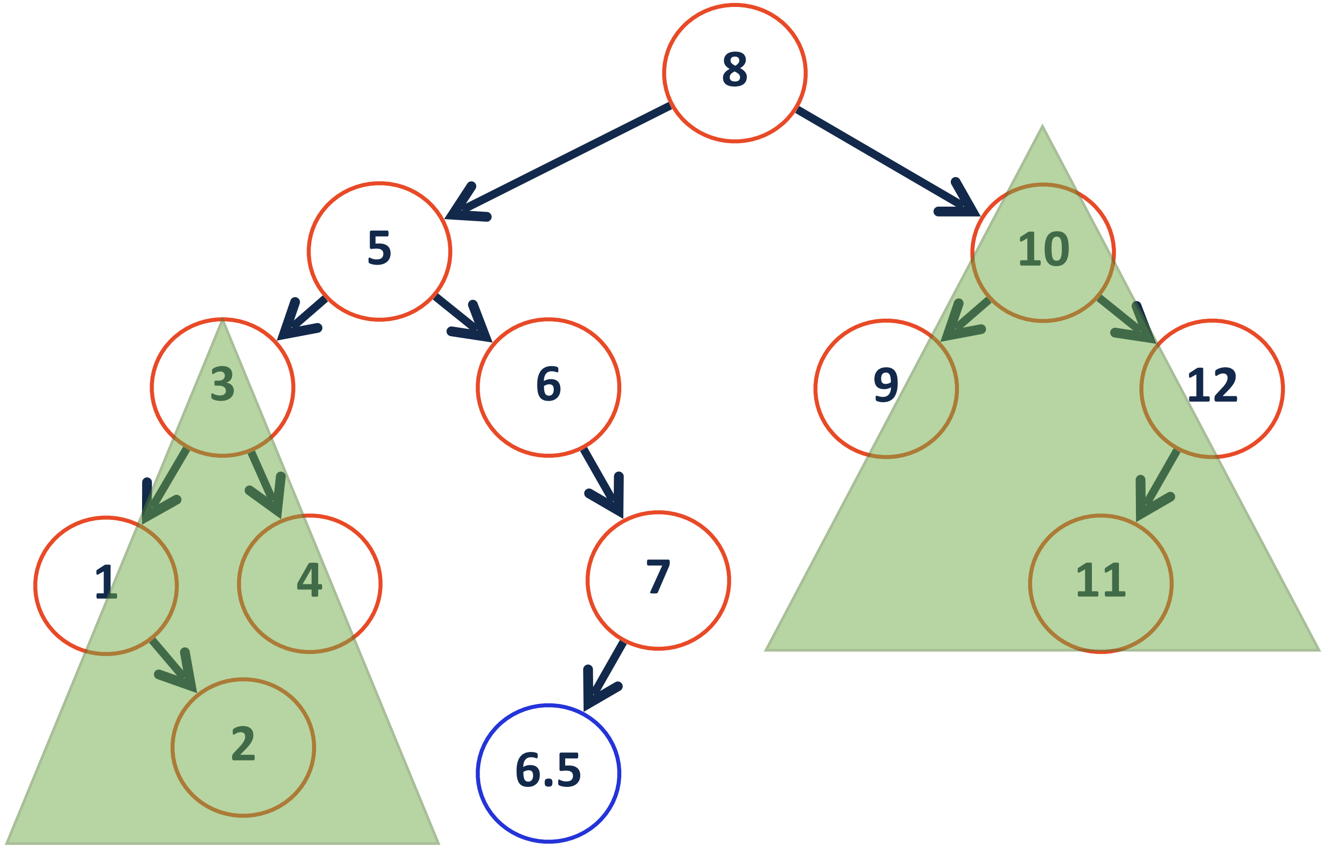

```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

AVL Insertion

_insert(6.5)

Insert (recursive pseudocode):

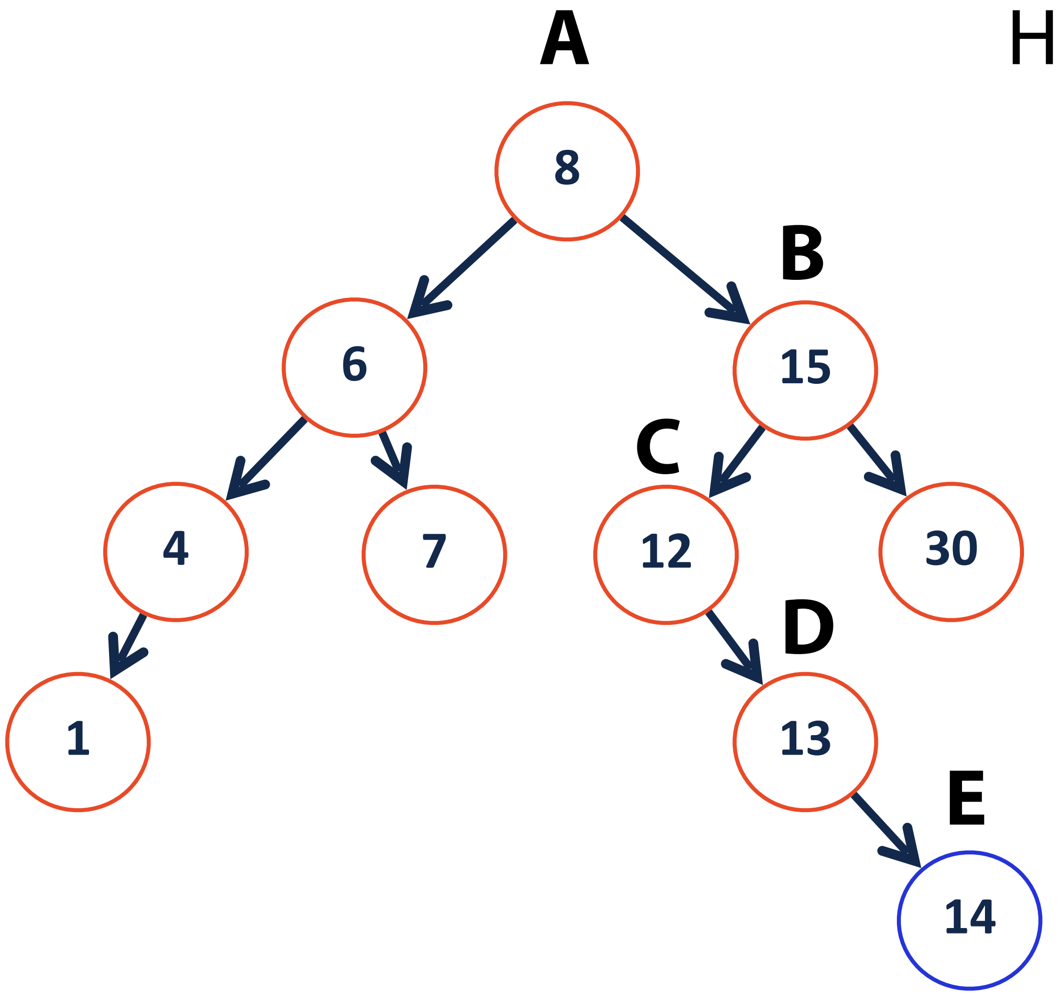
1. Insert at proper place
2. Check for imbalance
3. Rotate, if necessary
4. Update height




```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

AVL Insertion Practice

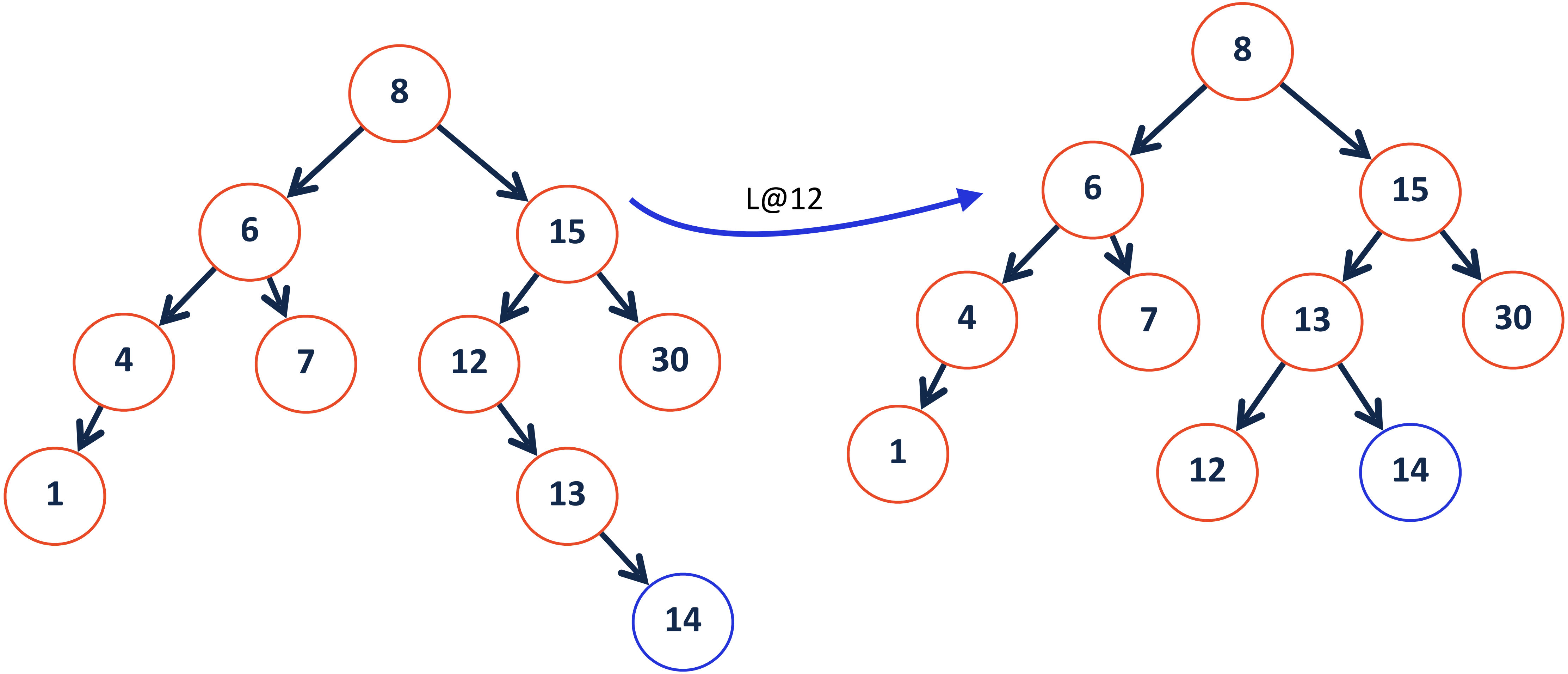
`_insert(14)`



Having inserted 14, where do we rotate?

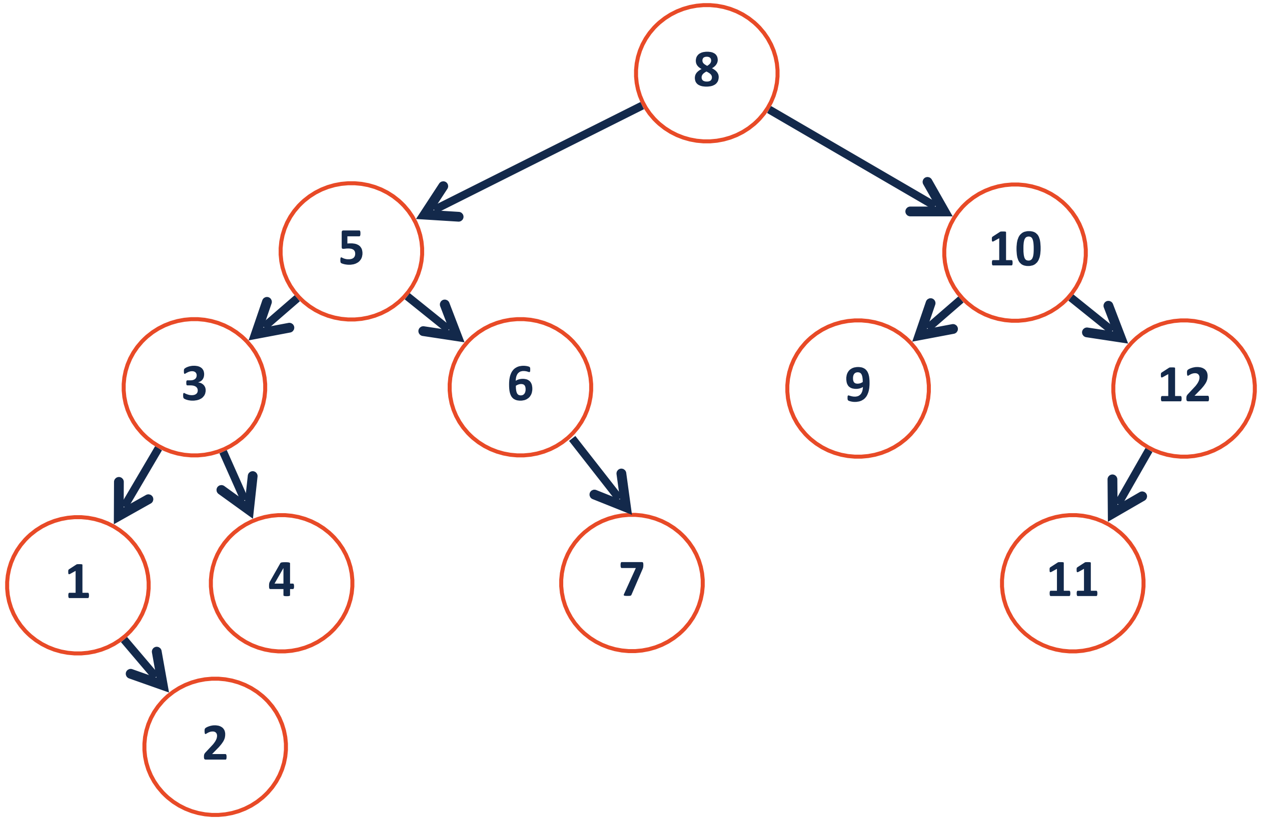
AVL Insertion Practice

`_insert(14)`



AVL Insertion

Given an AVL is balanced, insert can insert **at most** one imbalance



AVL Insertion Logic

Insert *may* increase height by at most **one**

A rotation *always* reduces the height of the subtree by **one**

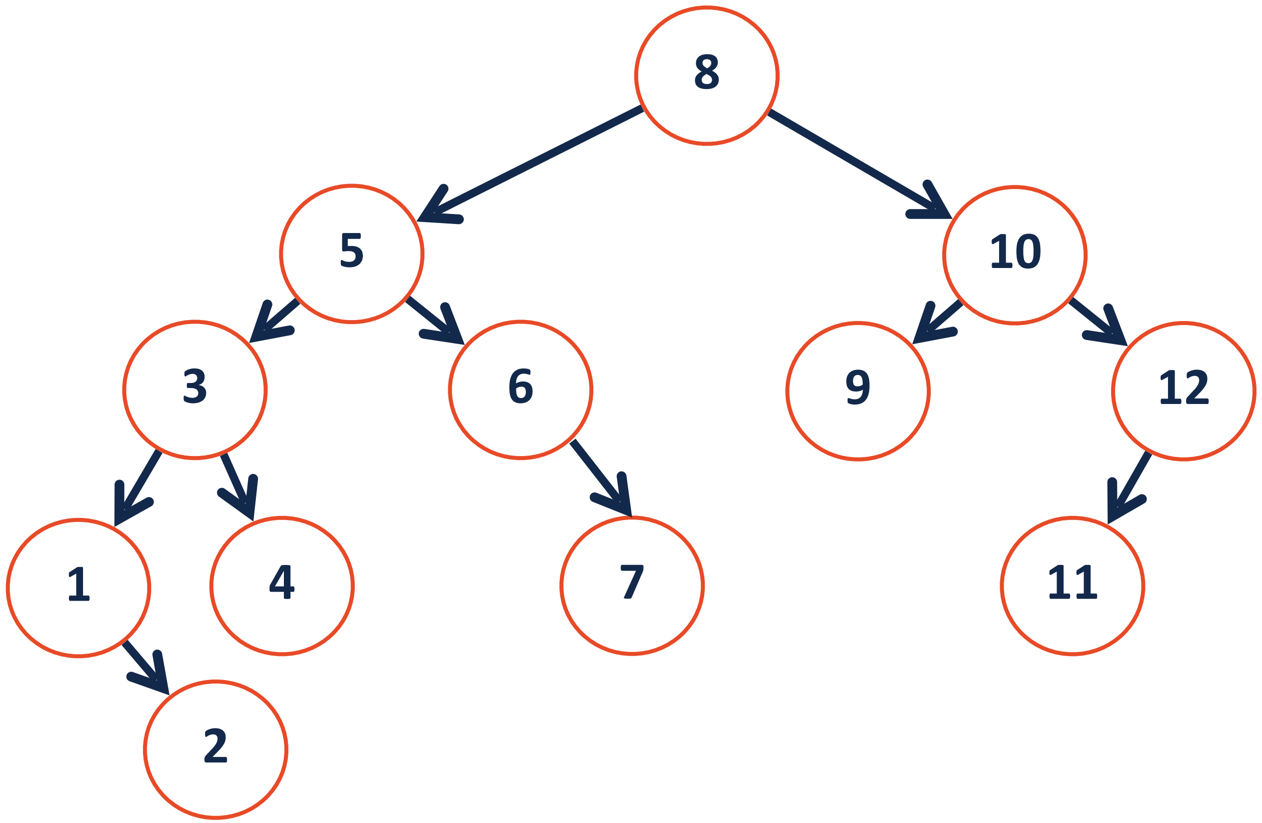
A single* rotation restores balance and corrects height!

What is the Big O of performing a single rotation?

What is the Big O of insert?

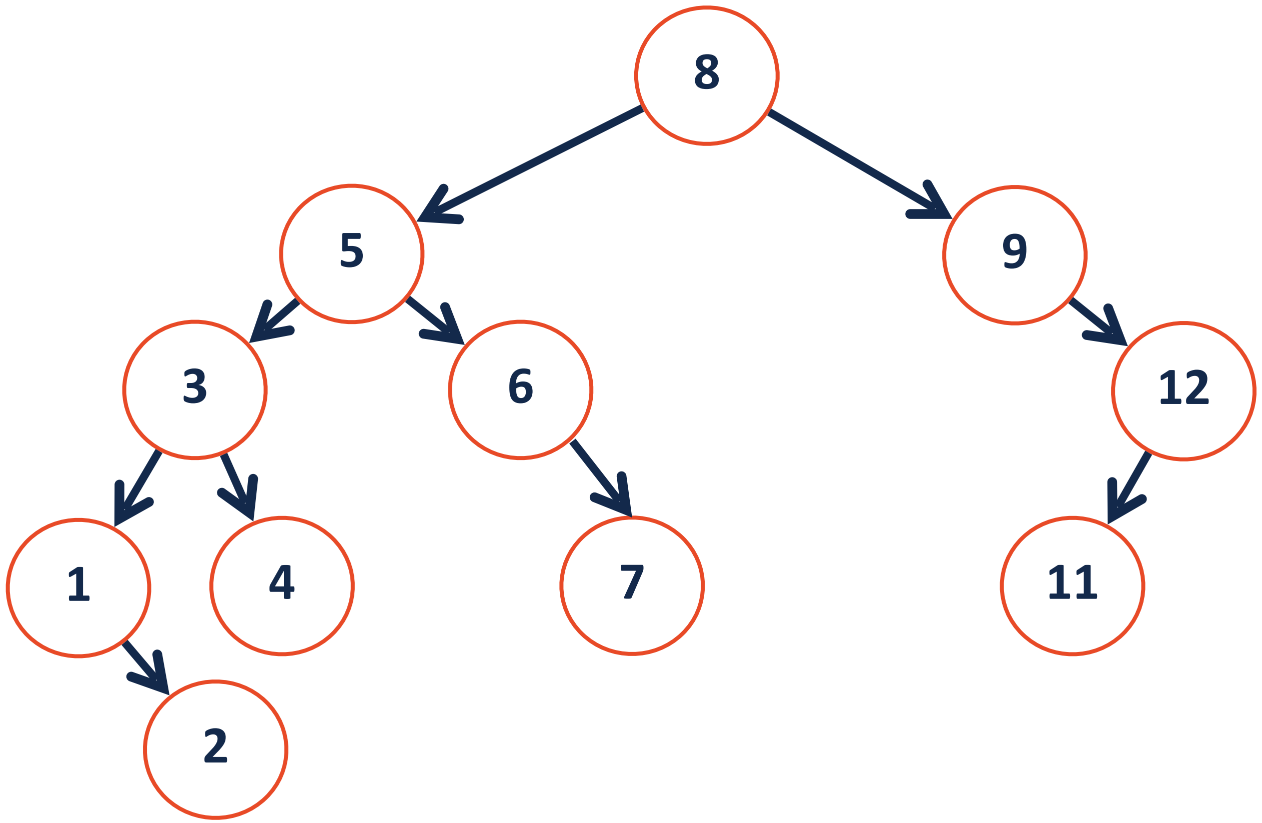
AVL Remove

`_remove(10)`



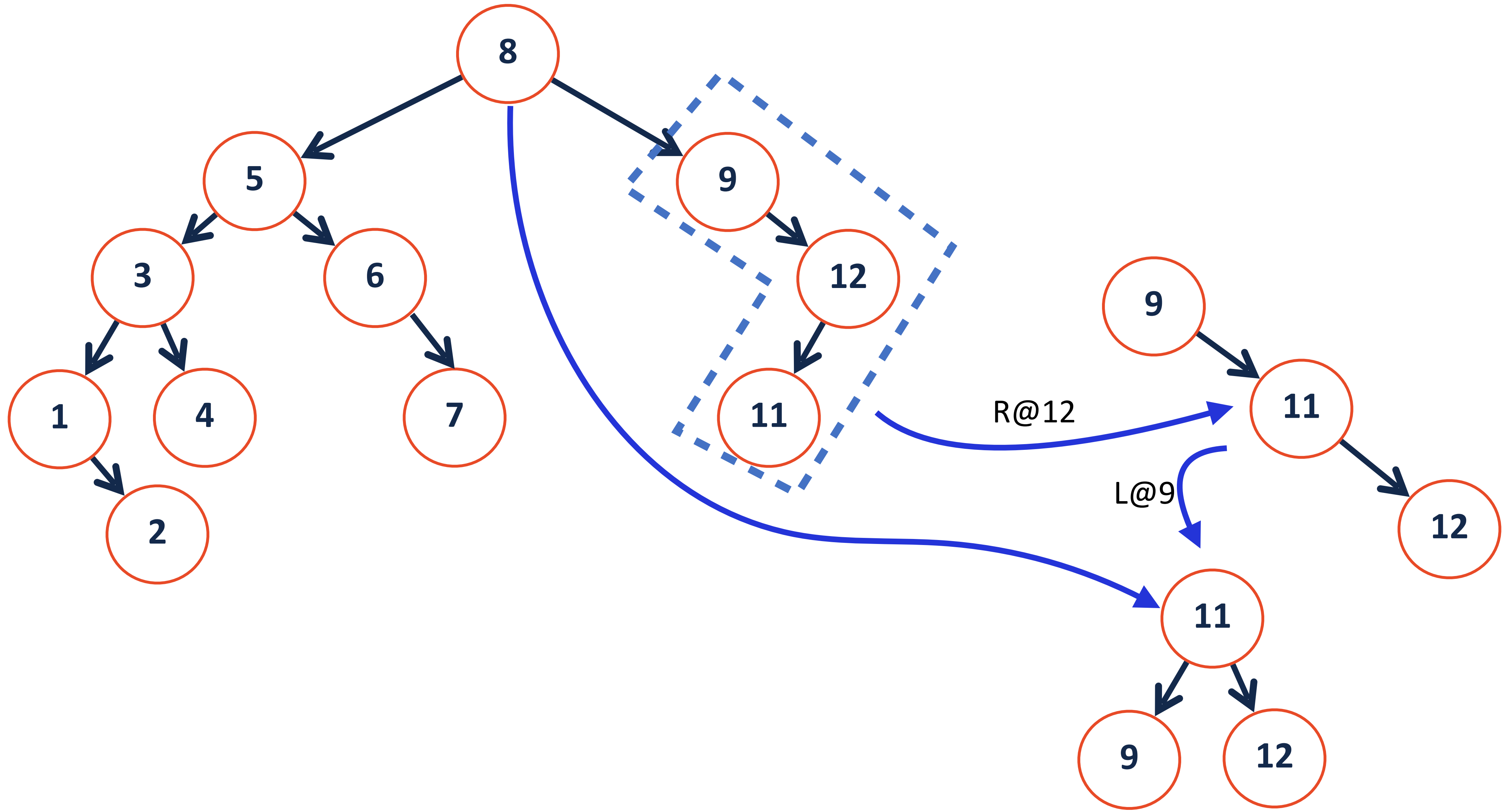
AVL Remove

`_remove(10)`



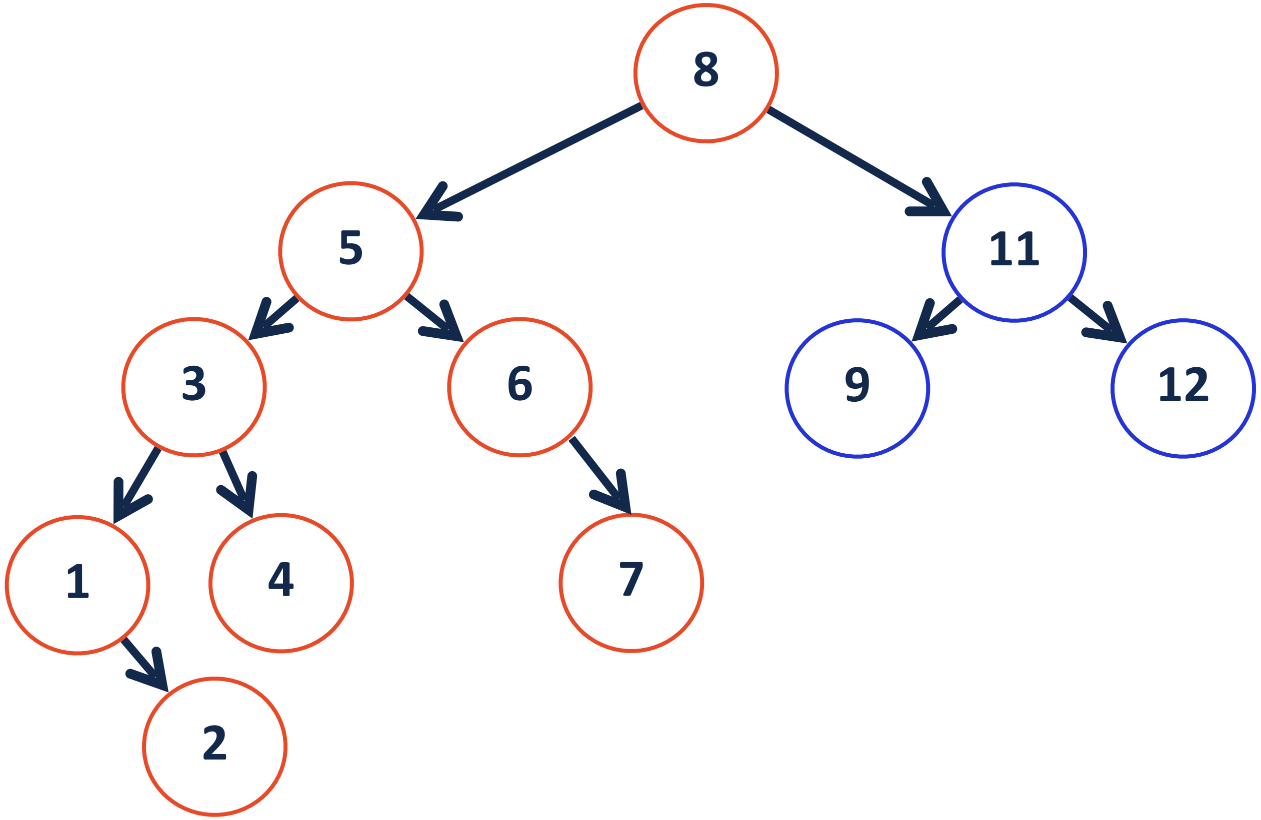
AVL Remove

`_remove(10)`



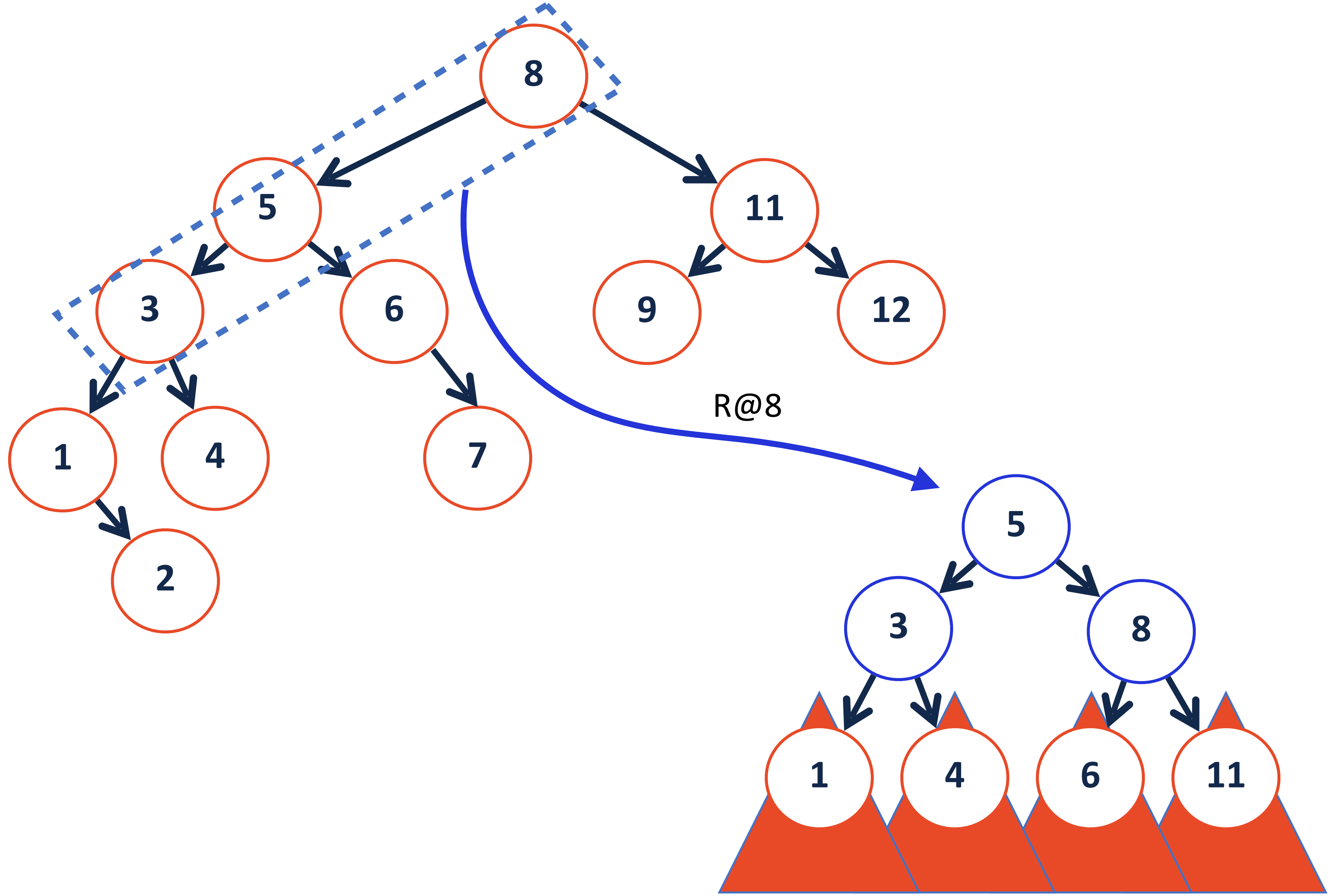
AVL Remove

`_remove(10)`



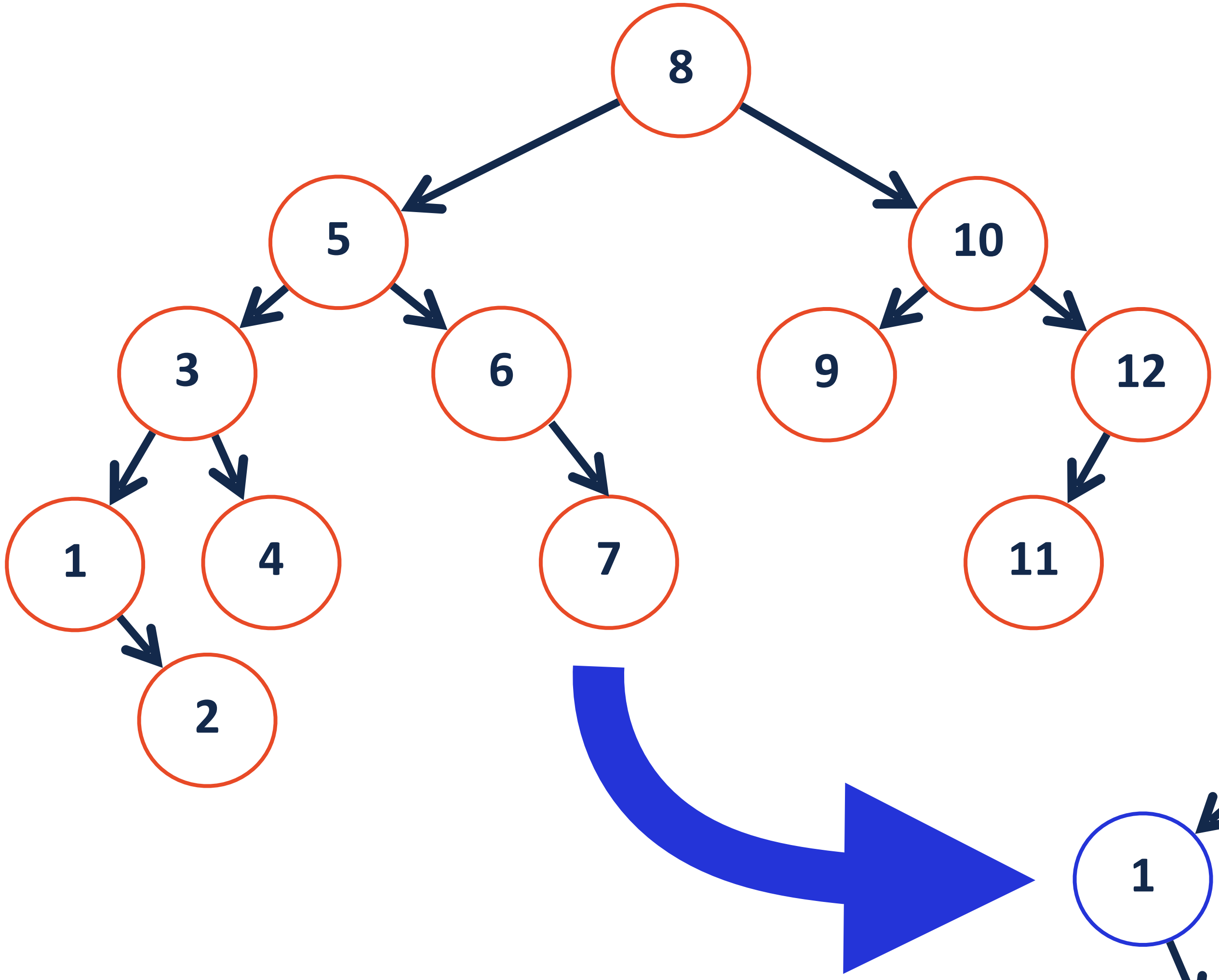
AVL Remove

_remove(10)



AVL Remove

`_remove(10)`



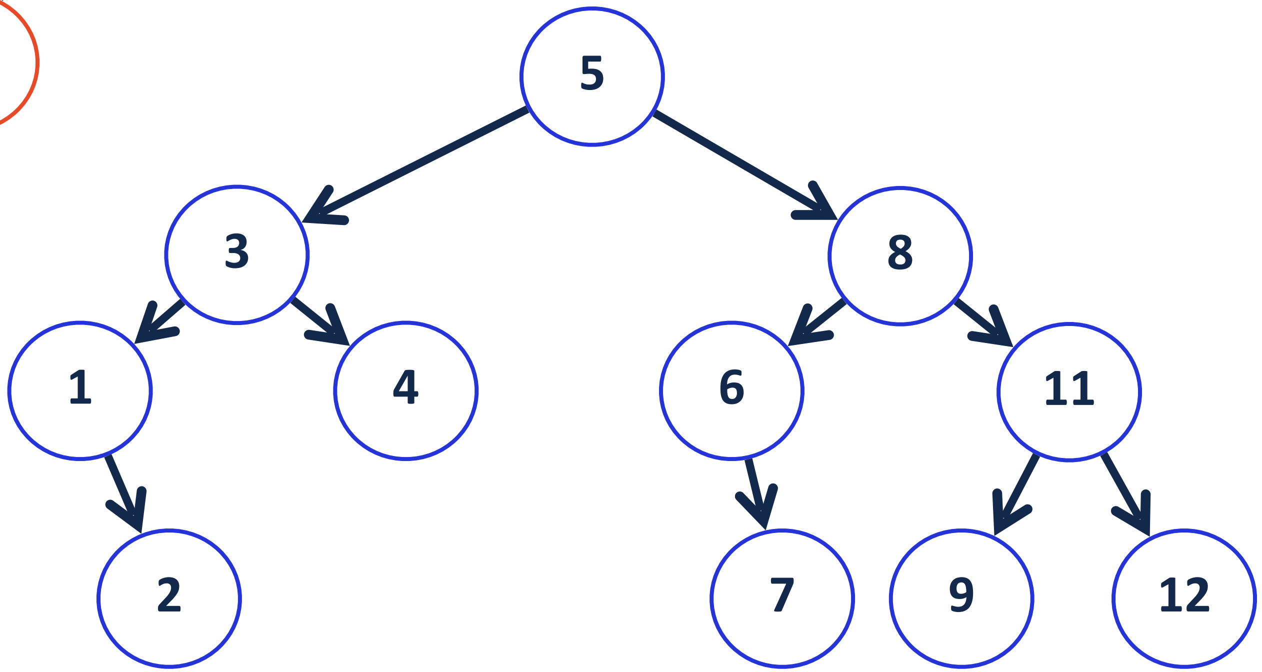
Remove (pseudo code):

1: Remove at proper place

2: Check for imbalance

3: Rotate, if necessary

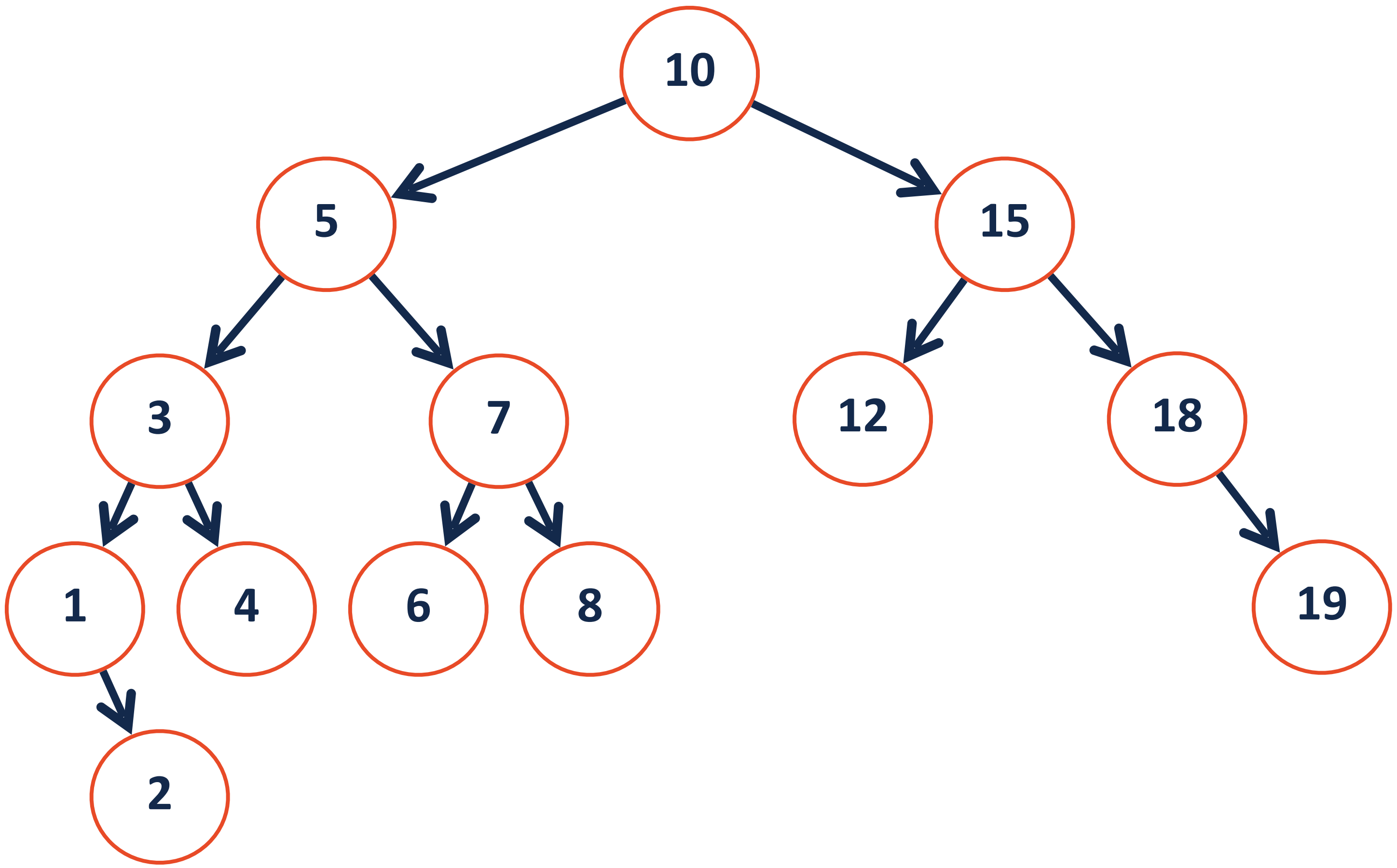
4: Update height



AVL Remove

`_remove(12)`

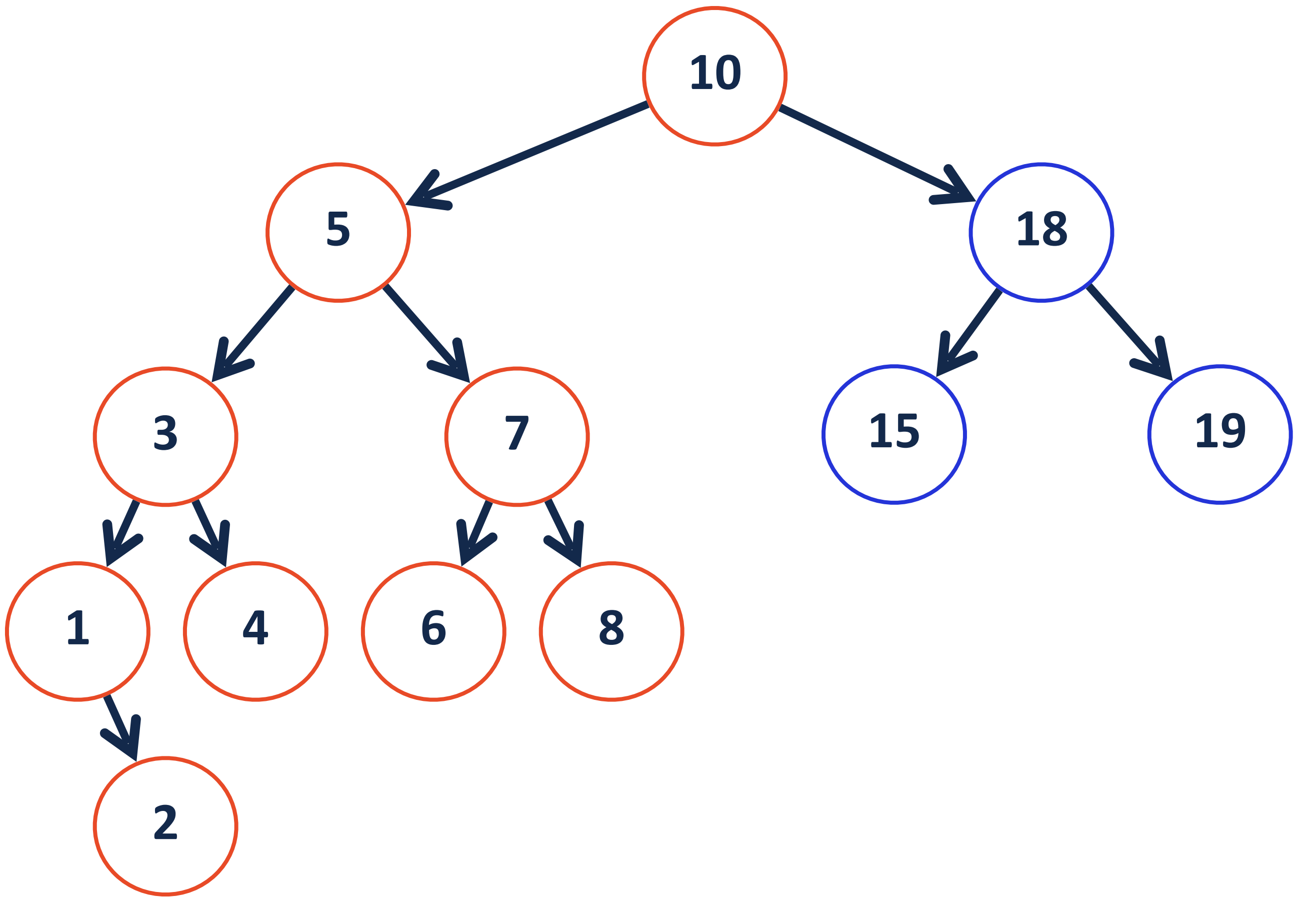
Remove can cause an imbalance at every level



AVL Remove

`_remove(12)`

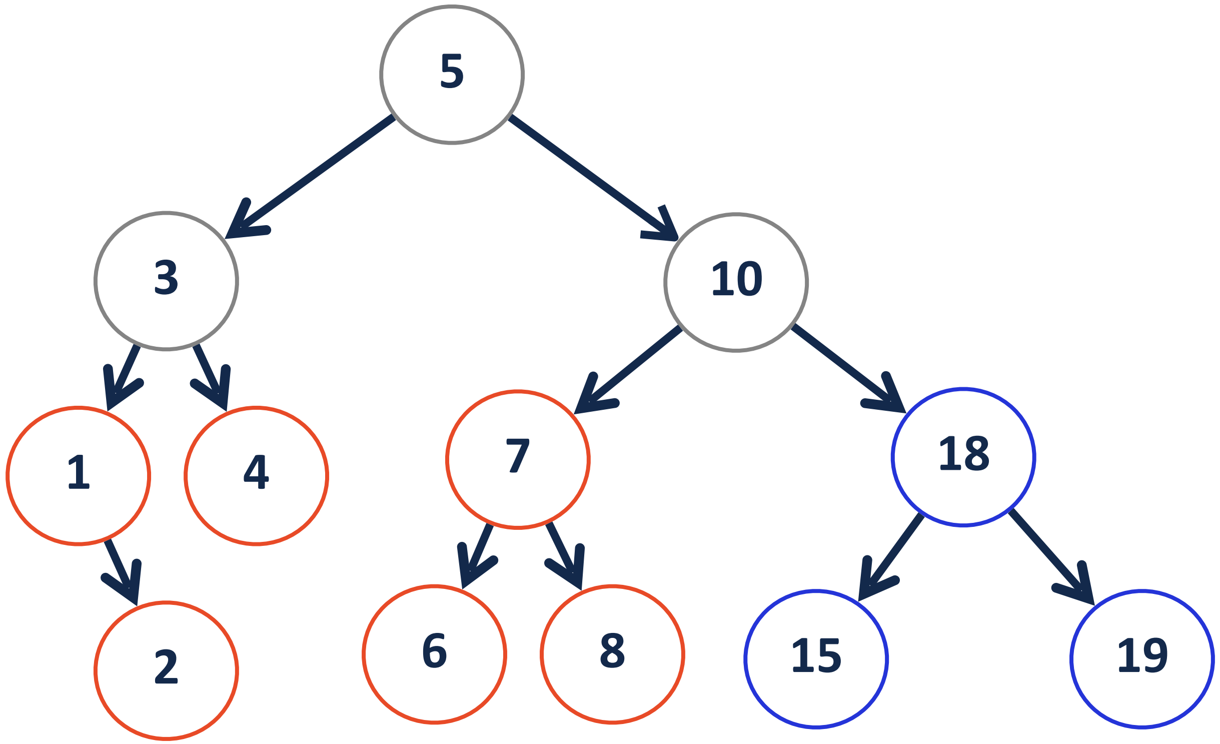
Remove can cause an imbalance at every level



AVL Remove

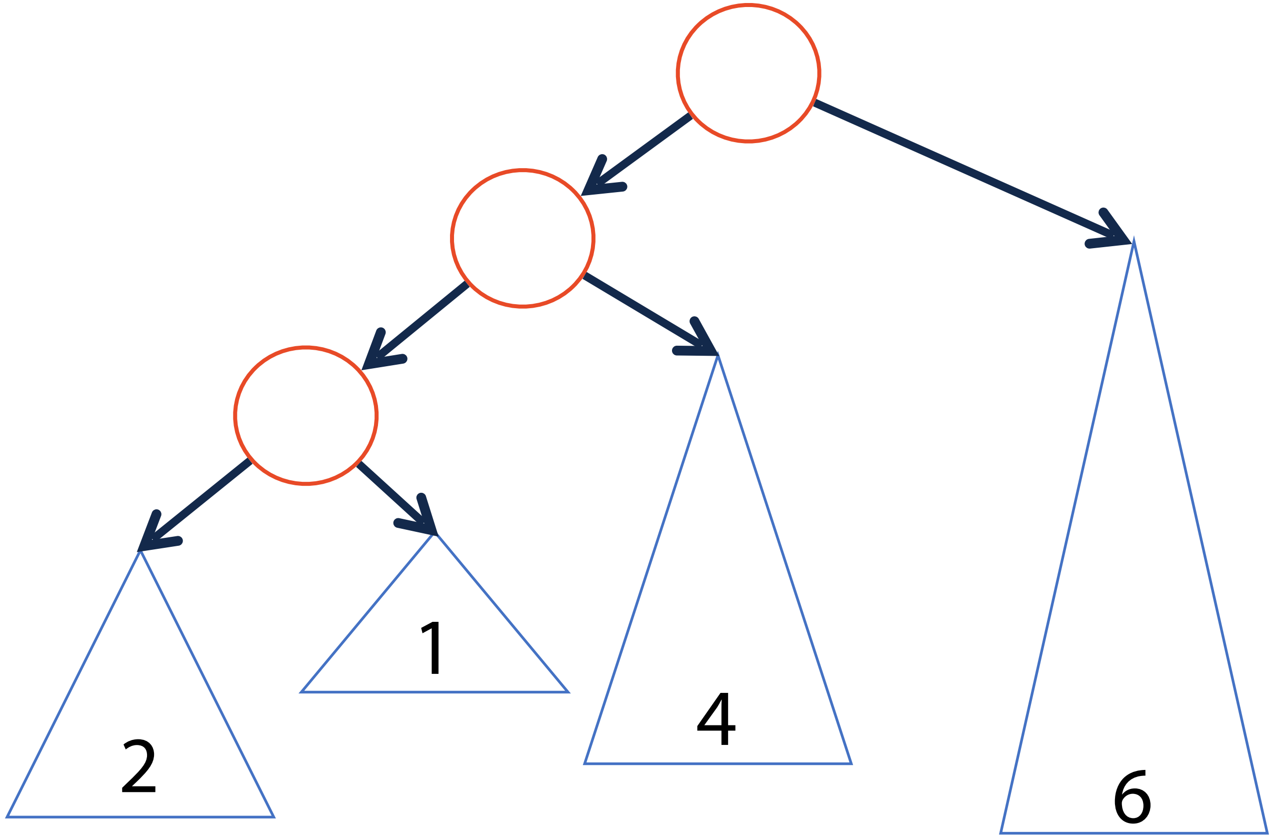
`_remove(12)`

Remove can cause an imbalance at every level



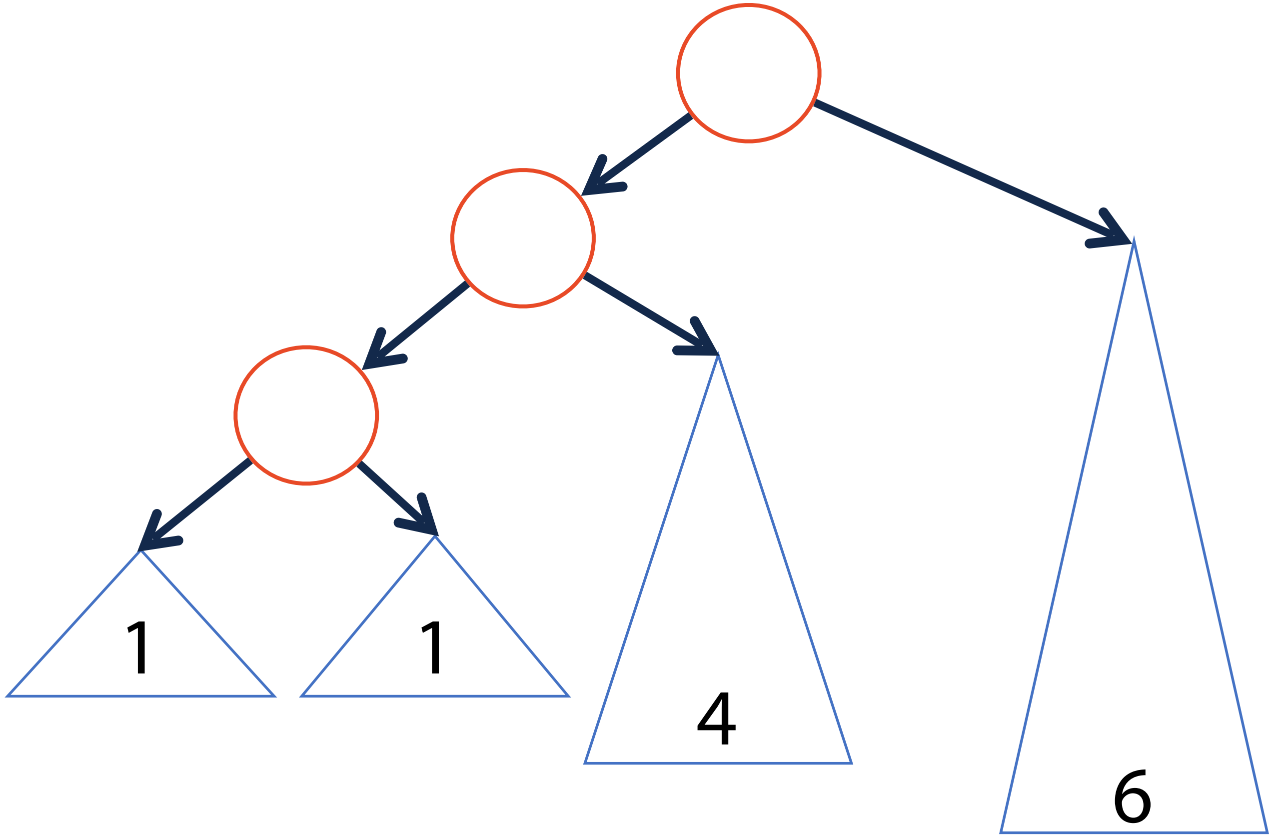
AVL Remove

Remove can cause an imbalance at every level



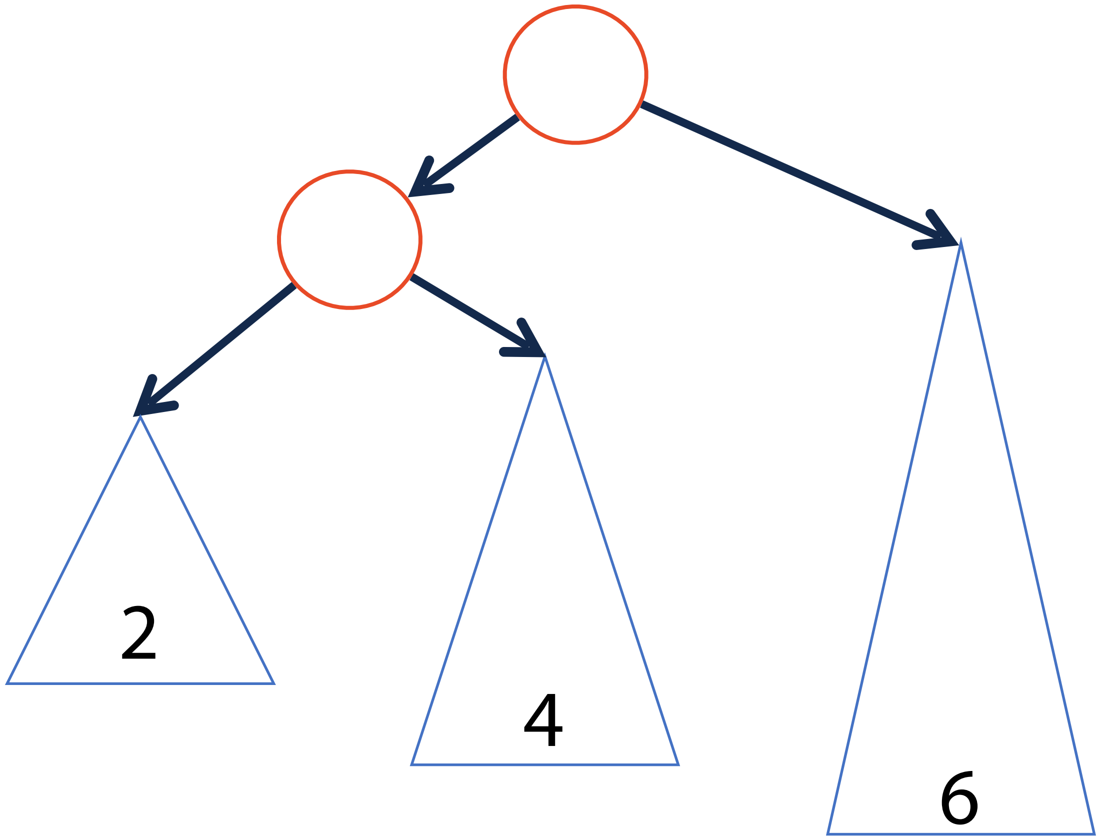
AVL Remove

Remove can cause an imbalance at every level



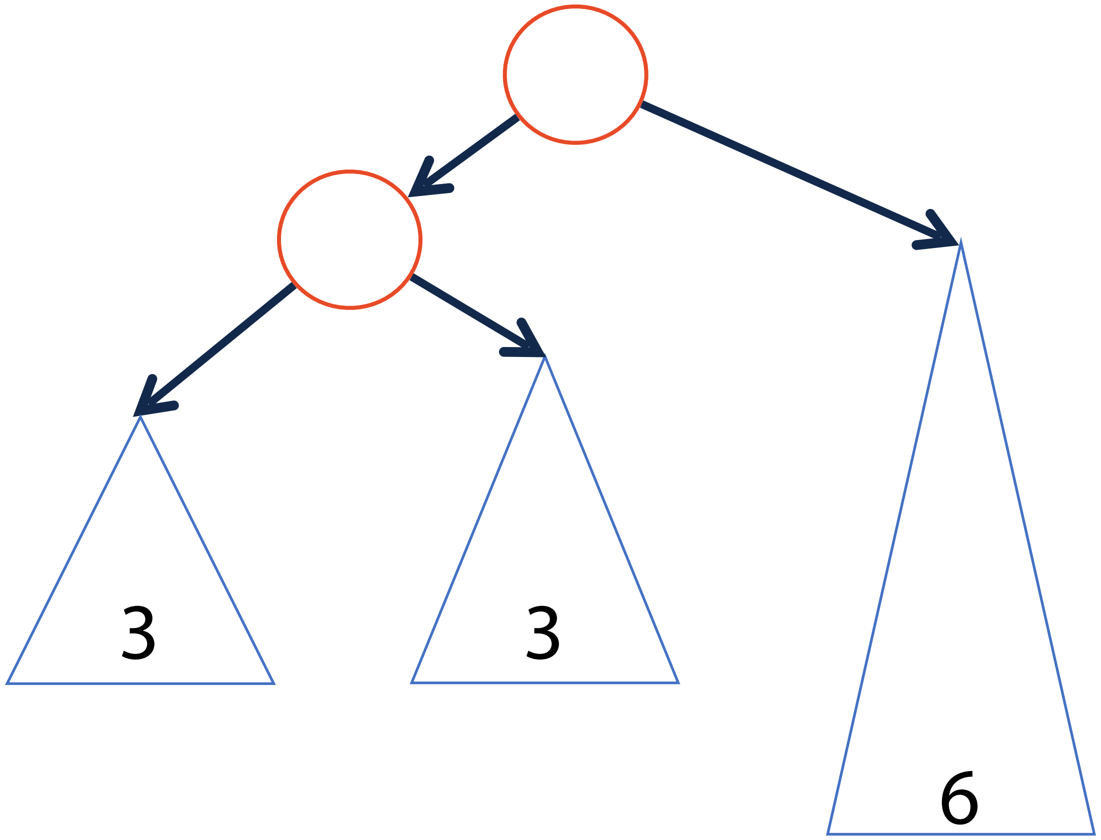
AVL Remove

Remove can cause an imbalance at every level



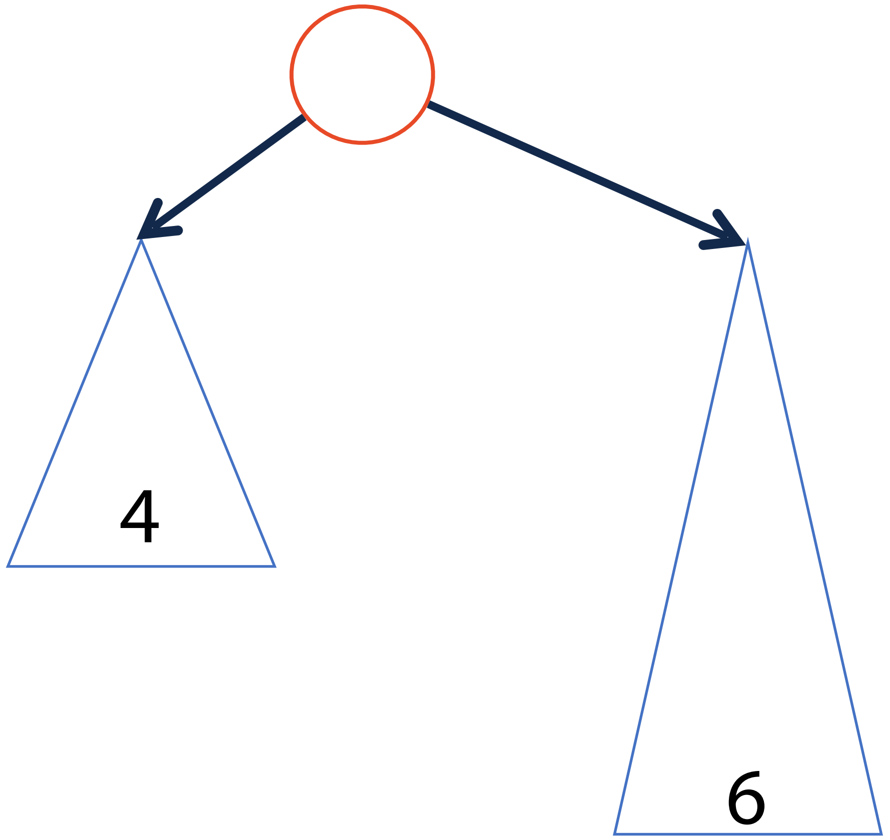
AVL Remove

Remove can cause an imbalance at every level



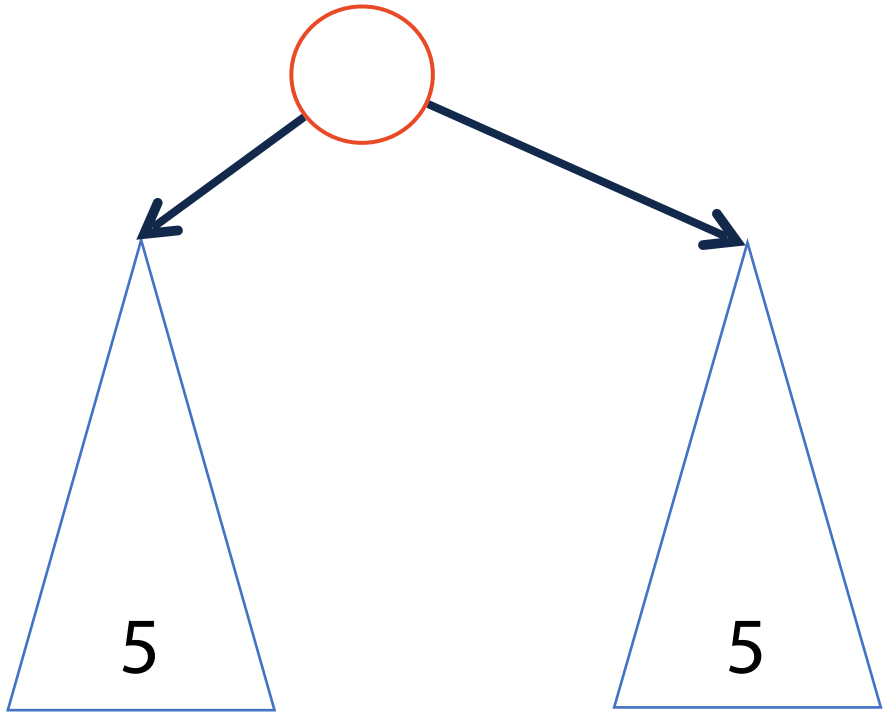
AVL Remove

Remove can cause an imbalance at every level



AVL Remove

Remove can cause an imbalance at every level



AVL Remove

An AVL remove step can reduce a subtree height by at most:

But a rotation ***reduces*** the height of a subtree by one!

We might have to perform a rotation at every level of the tree!

What is the Big O of performing a single rotation?

What is the Big O of remove?

AVL Tree Analysis

For an AVL tree of height h:

Find runs in: ___________.

Insert runs in: ___________.

Remove runs in: ___________.

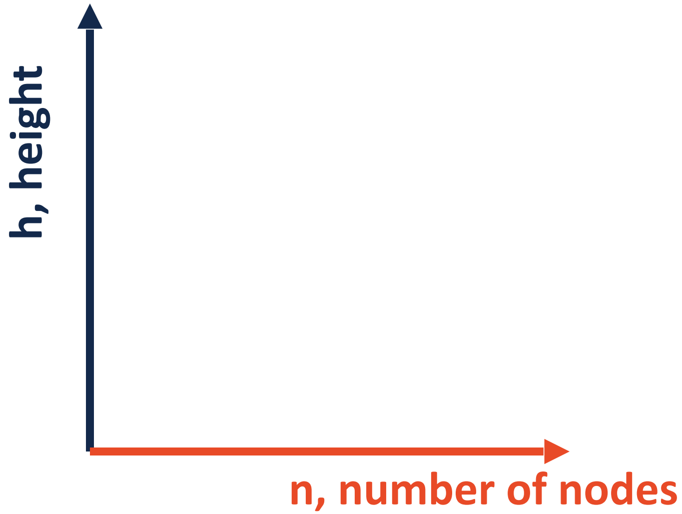
Claim: The height of the AVL tree with n nodes is: **___________.**

AVL Tree Analysis

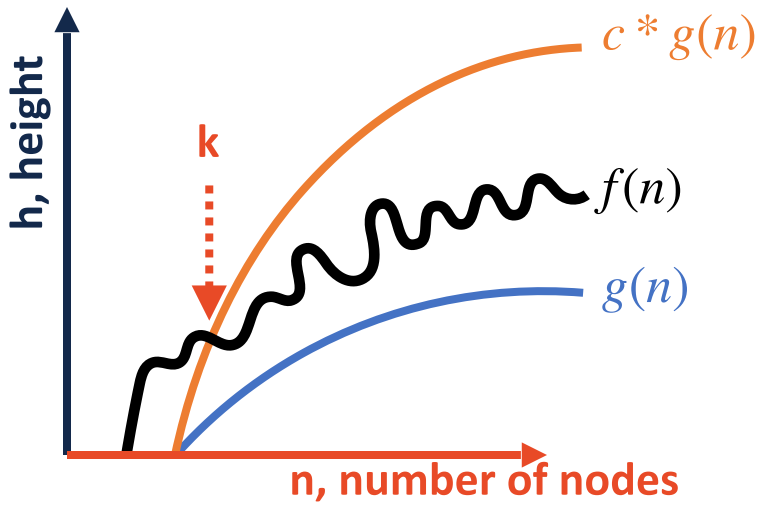
Definition of big-O:

$$f(n) \text{ is } O(g(n)) \text{ iff } \exists c, k \text{ s.t. } f(n) \le cg(n)\ \forall n > k$$

...or, with pictures:

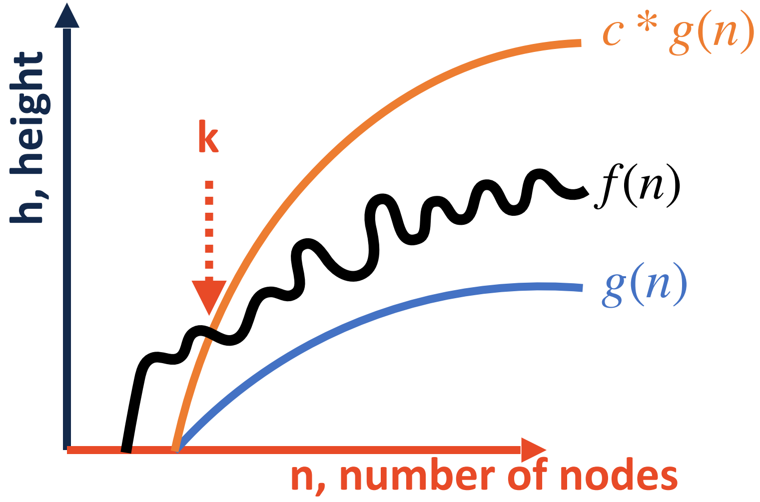


AVL Tree Analysis

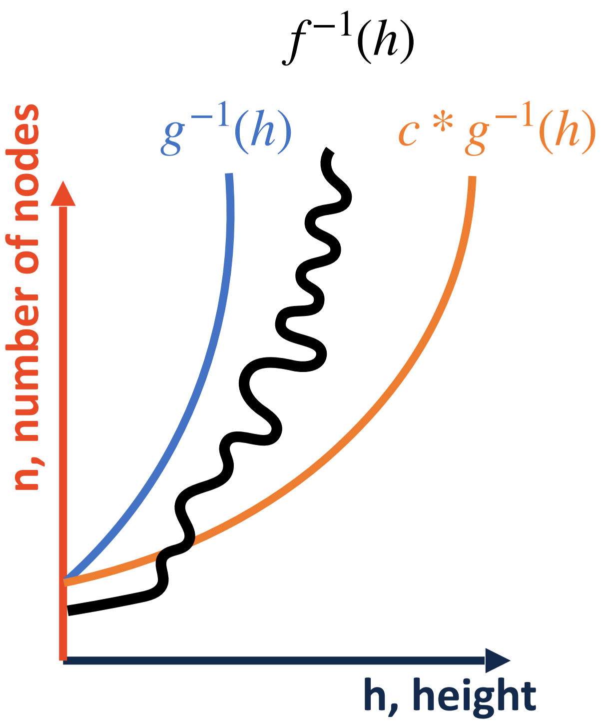


The height of the tree, **f(n)**, will always be <u>less than</u> **c × g(n)** for all values where **n > k**.

AVL Tree Analysis



$f(n)$ = "Tree height given nodes"



$f^{-1}(h)$ = "Nodes in tree given height"

The number of nodes in the tree, **f⁻¹(h)**, will always be <u>greater than</u> **c × g⁻¹(h)** for all values where **n > k**.

Plan of Action

Since our goal is to find the lower bound on **n** given **h**, we can begin by defining a function given **h** which describes the smallest number of nodes in an AVL tree of height **h**:

$N(h)$ = minimum number of nodes in an AVL tree of height h

Simplify the Recurrence

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

Simplify the Recurrence

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

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

Simplify the Recurrence

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

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

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

Simplify the Recurrence

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

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

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

1) Know characteristic equation? Get answer immediately!

Simplify the Recurrence

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

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

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

2) Unroll: $N(h) > 2N(h-2) = 2\left(2(N(h-4)\right) = 2^k\left(N(h-2k)\right)$

Simplify the Recurrence

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

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

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

2) Unroll: $N(h) > 2N(h-2) = 2\left(2(N(h-4)\right) = 2^k\left(N(h-2k)\right)$

When $h - 2k = 0, k = h/2$. Thus $N(h) > 2^{h/2}$

Simplify the Recurrence

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

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

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

3) Intuit approximate shape from recursion

Simplify the Recurrence

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

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

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

By whatever strategy you like: $N(h) > 2^{h/2}$

State a Theorem

Theorem: An AVL tree of height h has at least $N(h) > 2^{h/2}$.

Proof by Induction:

I. Consider an AVL tree and let **h** denote its height.

II. Base Case: ________________

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

III. Base Case: ________________

An AVL tree of height ____ has at least ____ nodes.

Prove a Theorem

IV. Induction Step: Assume for all heights $i < h, N(i) \geq 2^{i/2}$.

Prove that $N(h) \geq 2^{h/2}$

Prove a Theorem

IV. Induction Step: Assume for all heights $i < h, N(i) \geq 2^{i/2}$.

Prove that $N(h) \geq 2^{h/2}$

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

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

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

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

$$N(h) > 2^{h/2}$$

Prove a Theorem

V. Using a proof by induction, we have shown that:

Prove a Theorem

V. Using a proof by induction, we have shown that:

$N(h) \geq 2^{h/2}$, where $N(h)$ is the **min # of nodes of a tree of height h**

But we need to know n, the **# of nodes in any tree of height h**

Prove a Theorem

V. Using a proof by induction, we have shown that:

$N(h) \geq 2^{h/2}$, where $N(h)$ is the **min # of nodes of a tree of height h**

But we need to know n, the **# of nodes in any tree of height h**

$$n \geq N(h)$$

$$log(n) \geq \frac{h}{2}$$

$$h \leq 2\ log(n)$$

AVL Runtime Proof

An upper-bound on the height of an AVL tree is **O(lg(n))**:

N(h) := Minimum # of nodes in an AVL tree of height h

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

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

$> 2 \times 2^{(h-2)/2} = 2^{(h-2)/2+1} = 2^{h/2}$

Theorem #1:

Every AVL tree of height h has at least $2^{h/2}$ nodes.

AVL Runtime Proof

An upper-bound on the height of an AVL tree is **O(lg(n))**:

# of nodes (n) ≥ N(h) > $2^{h/2}$

n > $2^{h/2}$

lg(n) > h/2

2 × lg(n) > h

h < 2 × lg(n) ***, for h ≥ 1***

Proved: The maximum number of nodes in an AVL tree of height h is less than 2 × lg(n).

Summary of Balanced BST

Pros:

Cons:

Every Data Structure So Far

	Unsorted Array	Sorted Array	Unsorted Linked List	Sorted Linked List	Binary Tree	BST	AVL
Find							
Insert							
Remove							
Traverse							

Cache Locality / Memory Management

From an engineering perspective, linked lists were much worse than array lists due to memory locality!

Why are trees any different?

Can we make a tree thats good at ‘tree things’ AND memory local?