Algorithms and Data Structures for Data Science

Balanced Binary Search Trees

CS 277
Brad Solomon

March 18, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Reminder: Exam 2 this week!

Reminder: I’m out of town starting tomorrow

Wednesday lecture will be async online.

??

Friday lab will be run by TAs

MP → ???

Learning Objectives

Review tree runtimes for binary search trees

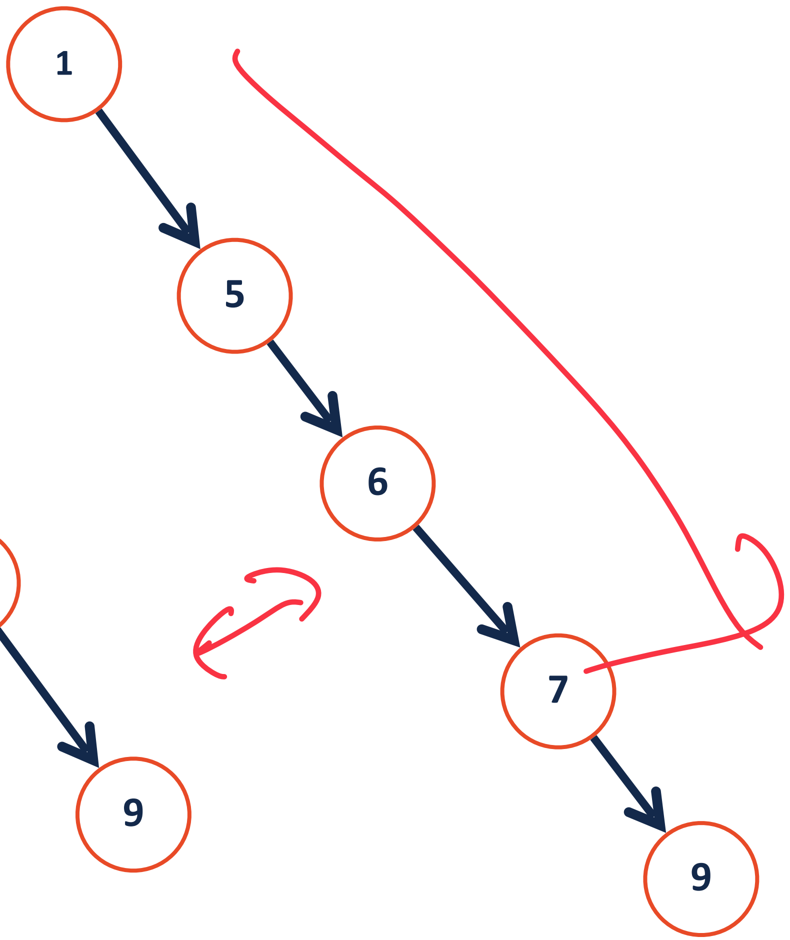
Introduce the AVL tree - balance

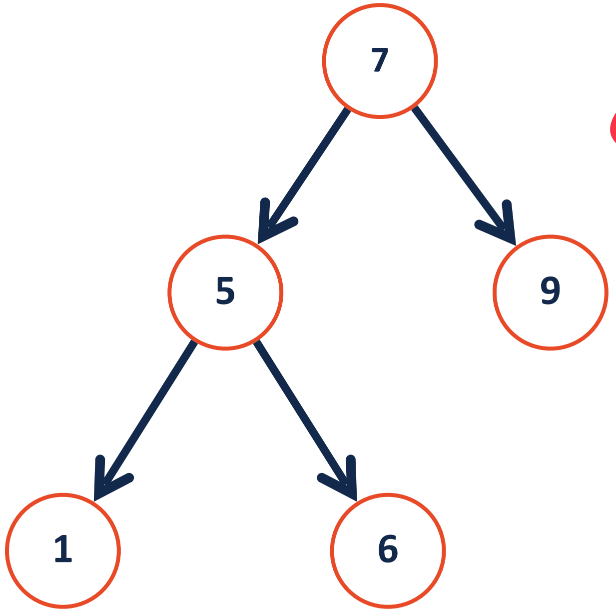
Demonstrate how AVL tree rotations work

BST Analysis – Running Time

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

worst case
height : 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?

$$f(n) \leq h \leq g(n)$$

$$f(h) \leq n \leq g(h)$$

BST Analysis

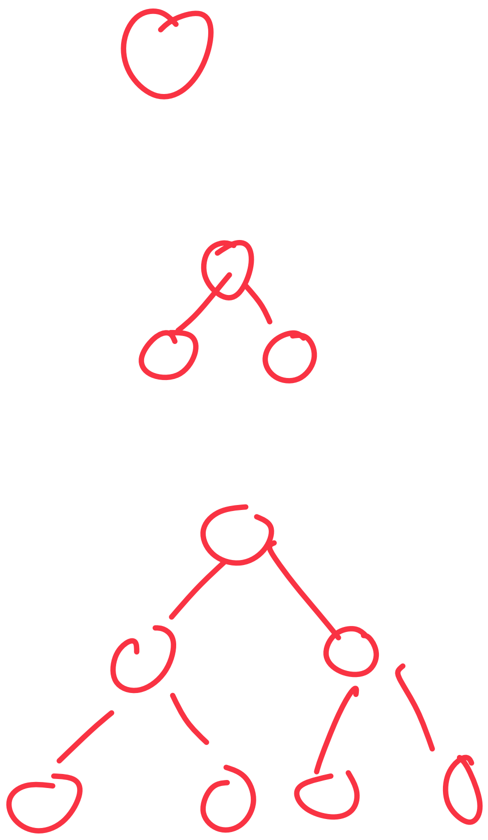
Draw some examples!

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

$h = 0$

$h = 1$

$h = 2$



1

2

4

$2^0 + 2^1 + 2^2 + 2^3 \dots + 2^h$

$n = 2^{h+1} - 1$ max nodes

log

$\log n \sim h + 1$

min height

$h \approx O(\log n)$

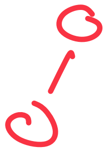
BST Analysis

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

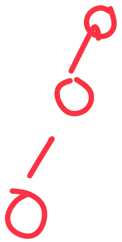
$h = 0$

min is $h+1$

$h = 1$



$h = 2$

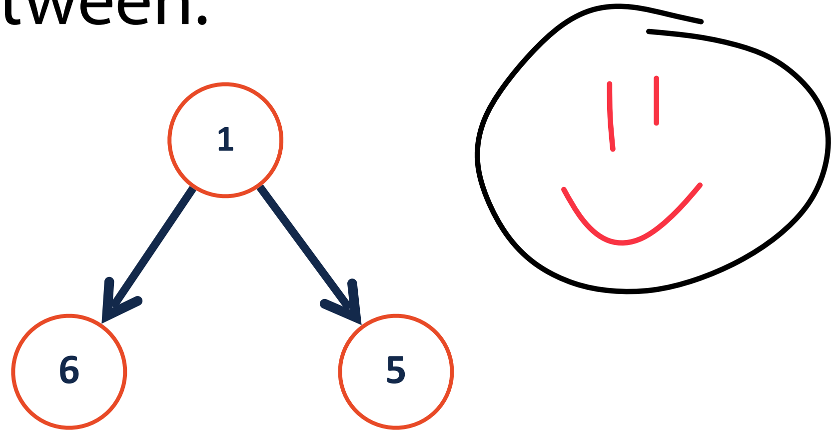


BST Analysis

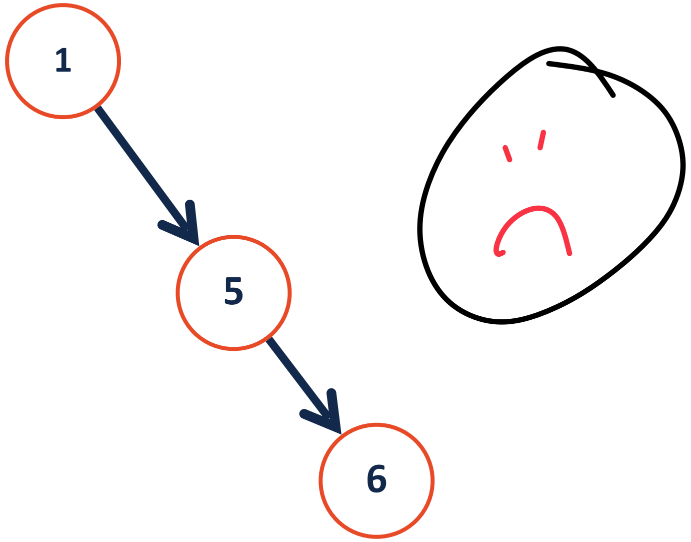
A BST of n nodes has a height between:

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

$h = O(\log n)$



Upper-bound: $O(n)$

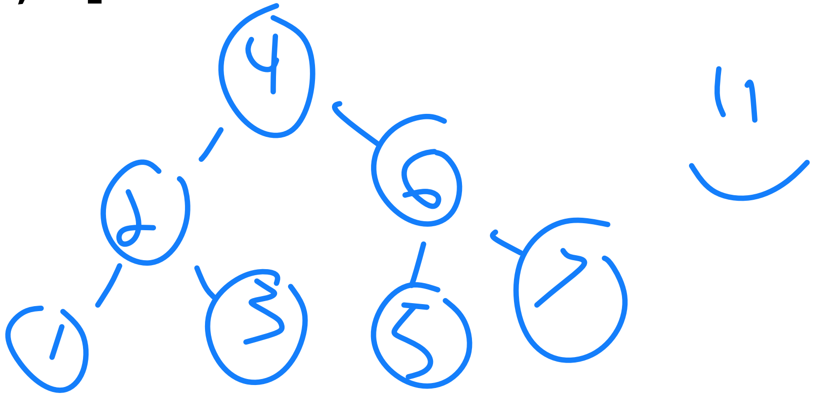


Correcting bad insert order

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

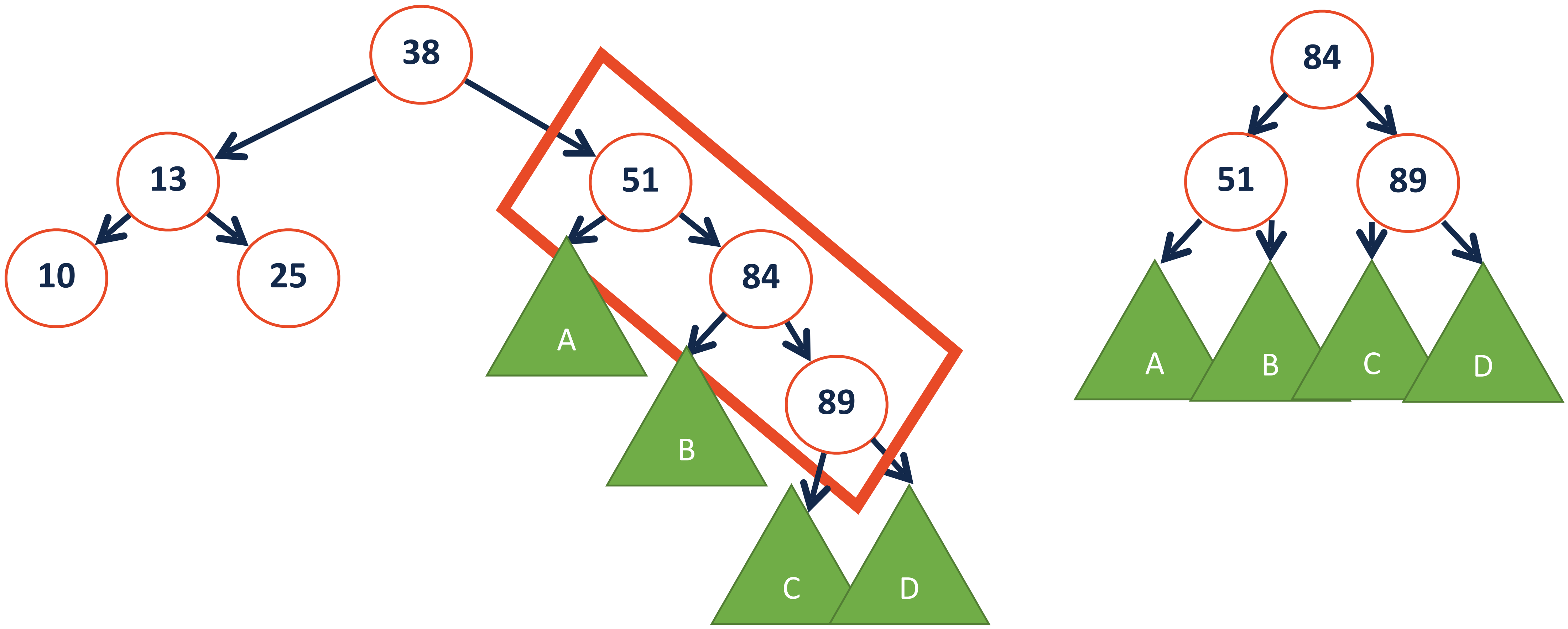
Insert Order: [1, 3, 2, 4, 5, 6, 7]

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



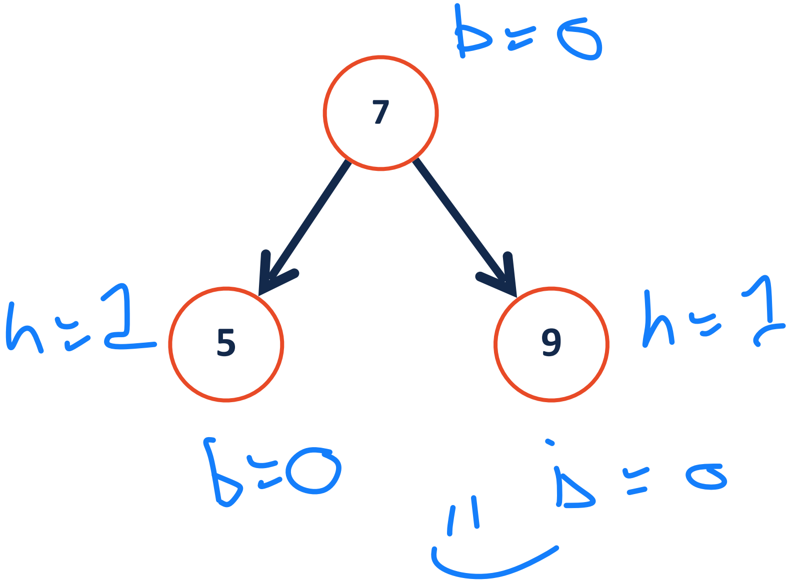
AVL-Tree: A self-balancing binary search tree

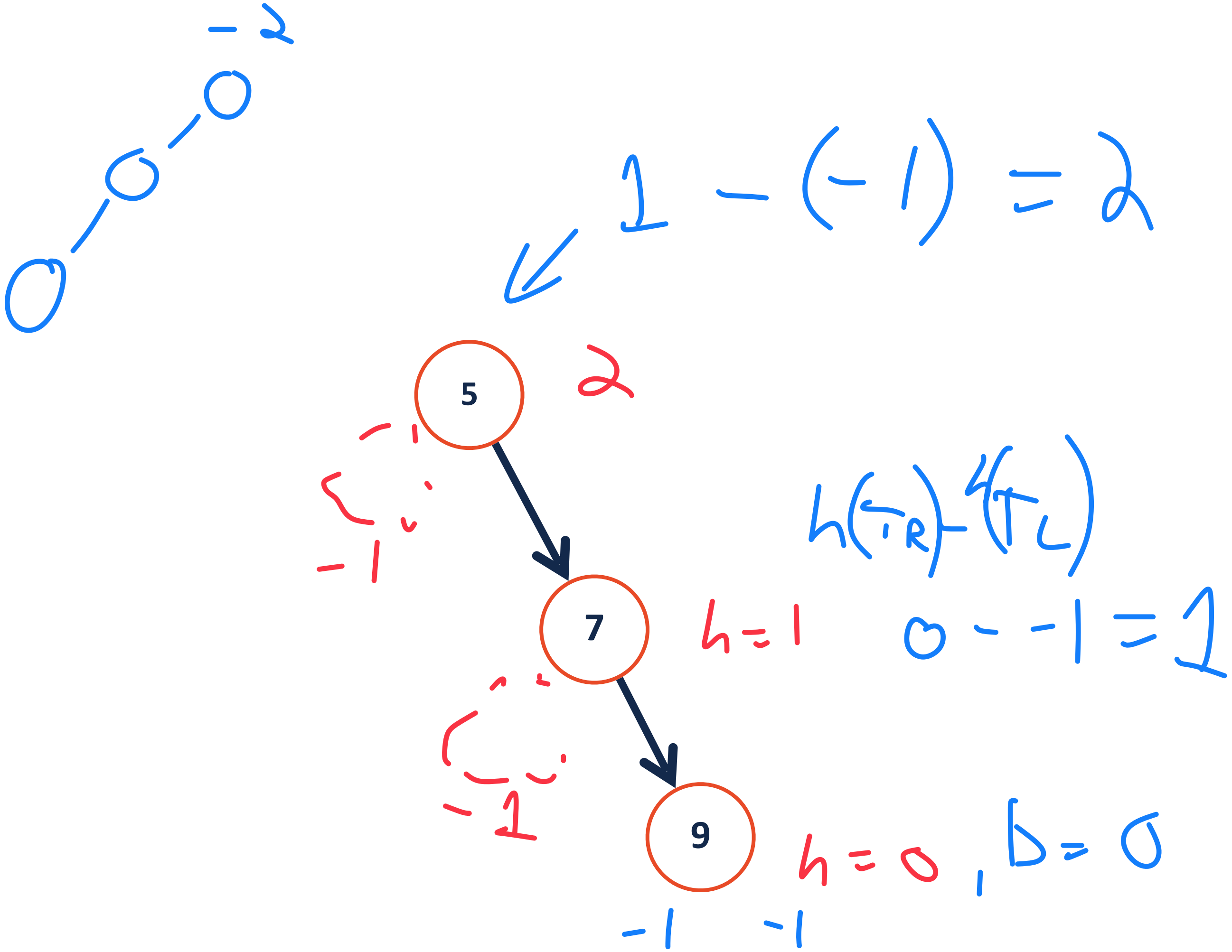
Rather than fixing an insertion order, just correct the tree as needed!



Height-Balanced Tree

What tree is better?



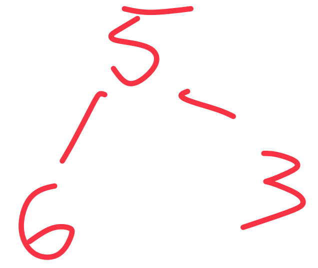


Height balance: $b = height(T_R) - height(T_L)$

A tree is "balanced" if: for all nodes, $|balance| \leq 1$

BST Rotations (The AVL Tree)

B.T.



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

B.S.T.

These rotations:

1. Modify the order of nodes and keep BST property

   ↳ left is smaller

   ↳ right is larger

2. Reduce tree height by one

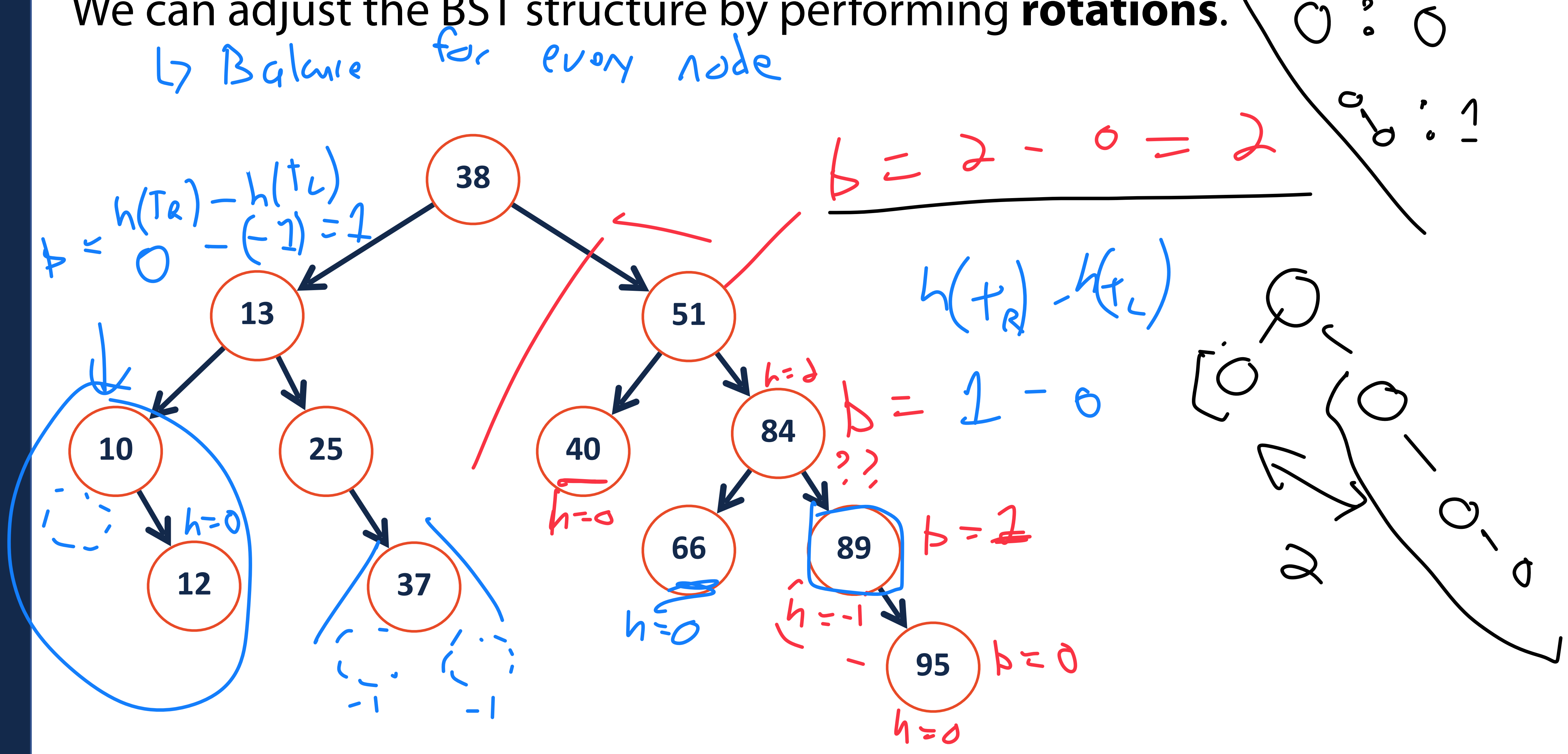
BST Rotations (The AVL Tree)

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

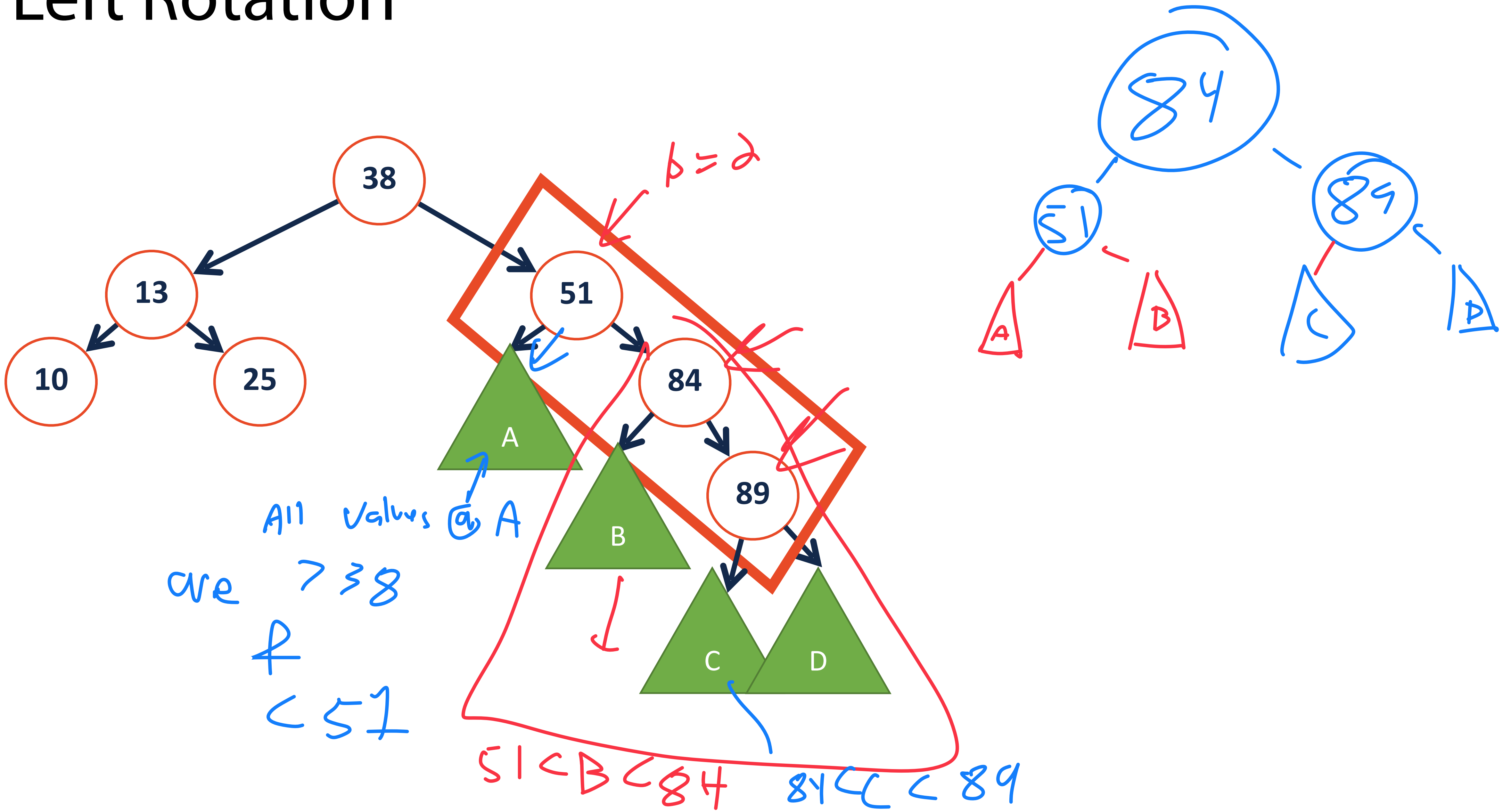
↳ Balance for every node

height

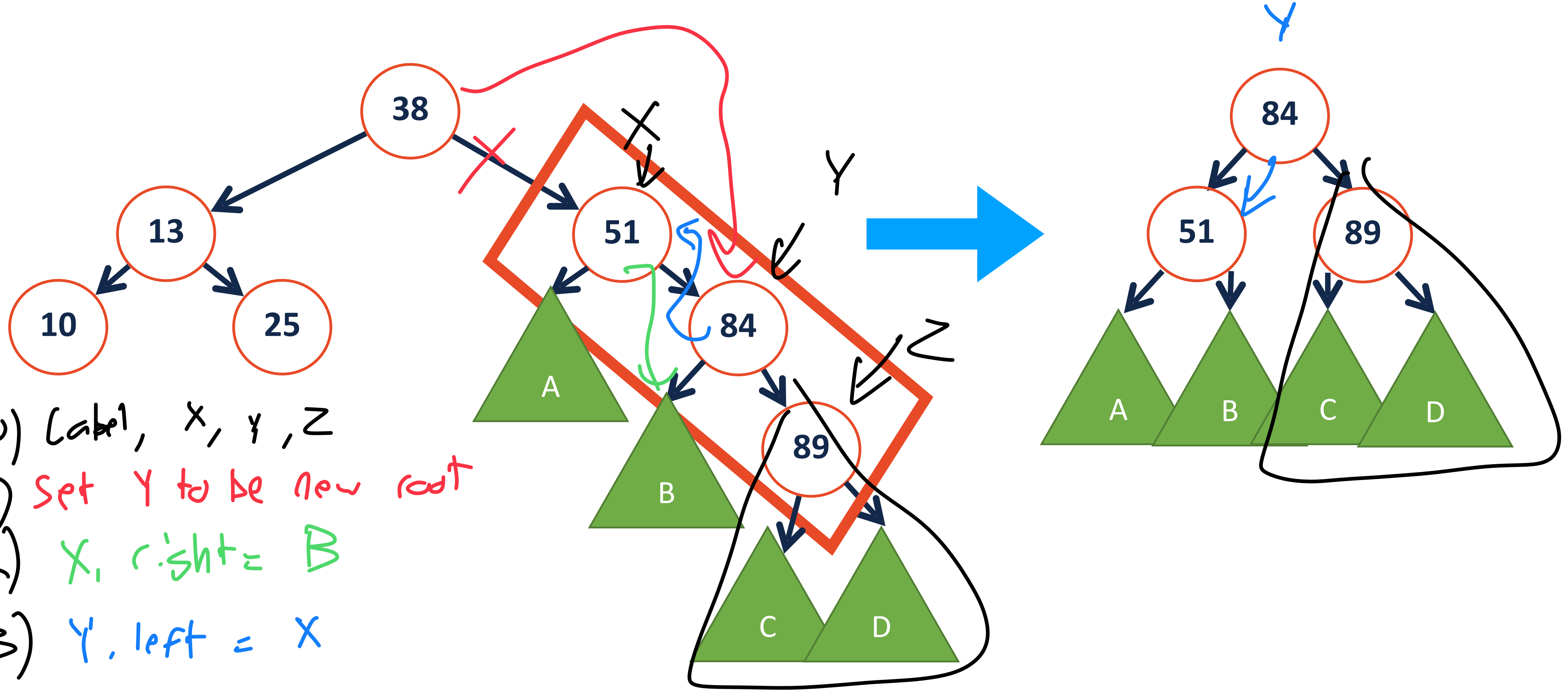
None : -1



Left Rotation



Left Rotation

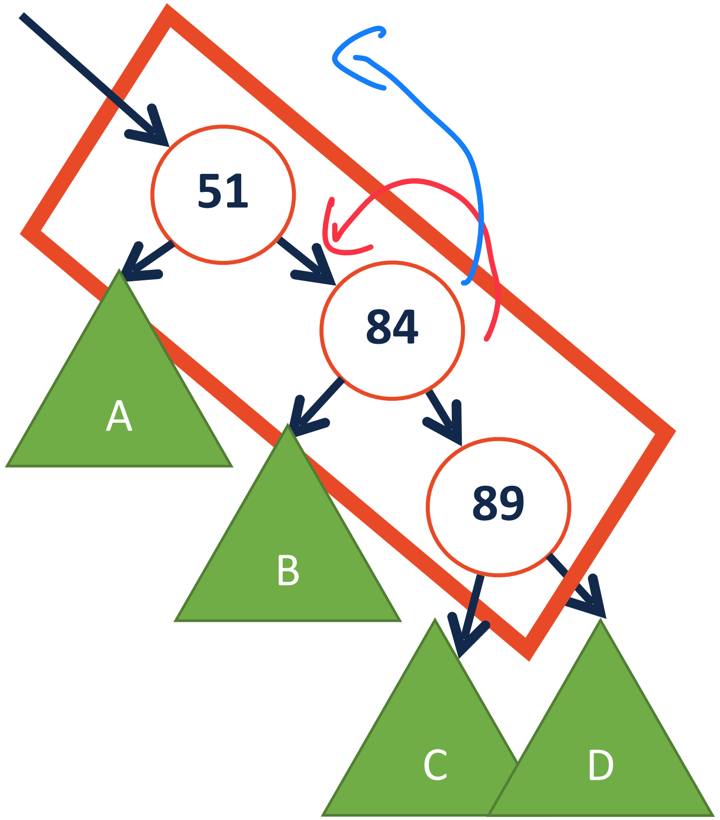


0) Label, x, y, z
1) Set Y to be new root
2) X. right = B
3) Y. left = X

Coding AVL Rotations

Two ways of visualizing:

1) Think of an arrow 'rotating' around the center



Coding AVL Rotations

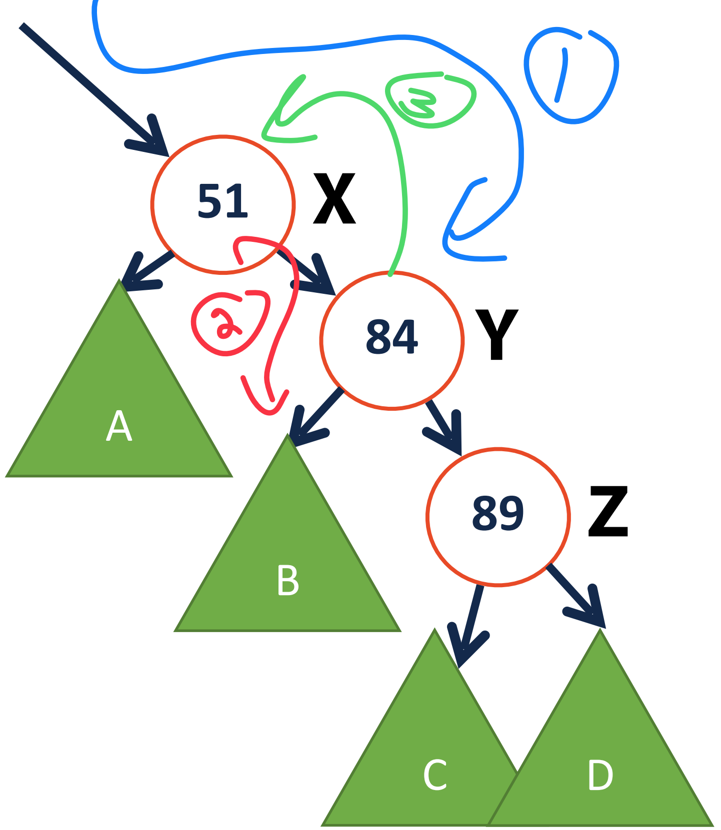
Two ways of visualizing:

2) The rotation will always do the following:

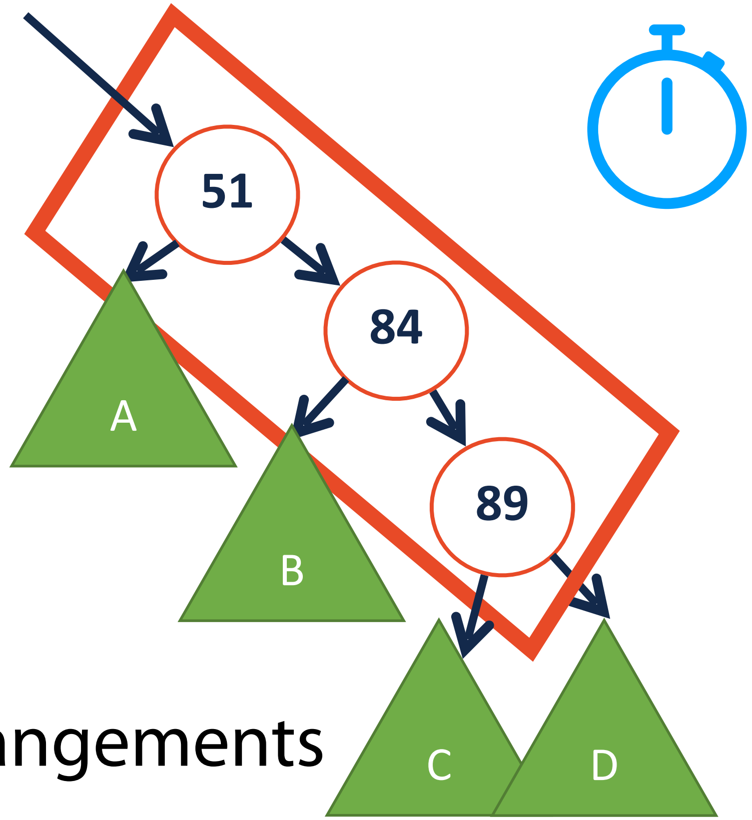
Make node **Y** the new root

Make the subtree **B X**'s right child.

Make node **X** the left child of node **Y**



Coding AVL Rotations



Two ways of visualizing:

1) Think of an arrow 'rotating' around the center

2) Recognize that there's a concrete order for rearrangements

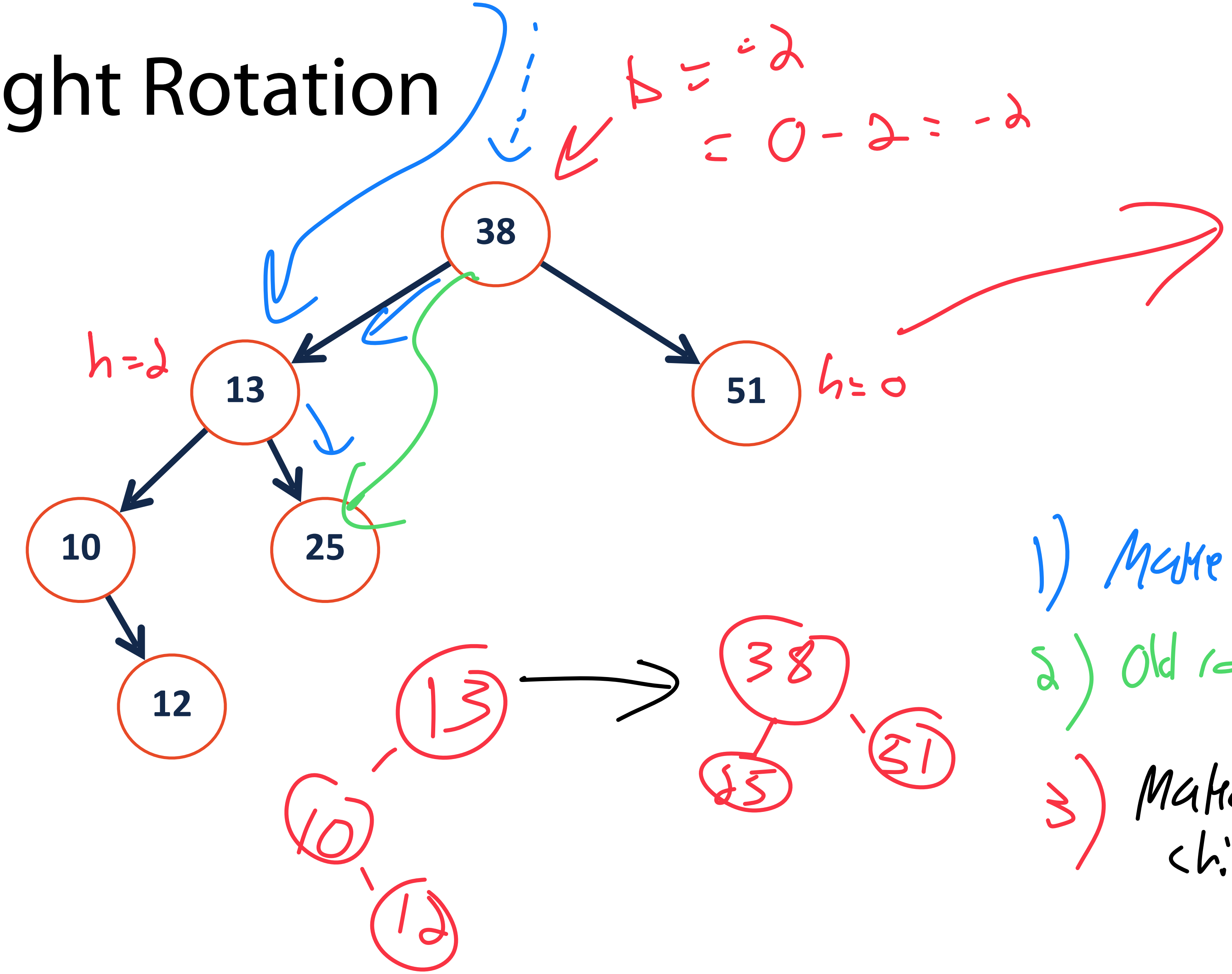
Ex: Unbalanced at current (root) node and need to *rotateLeft*?

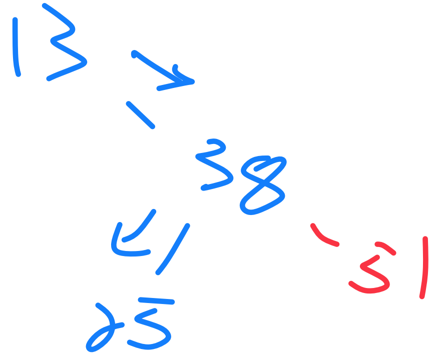
Replace current (root) node with it's right child.

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



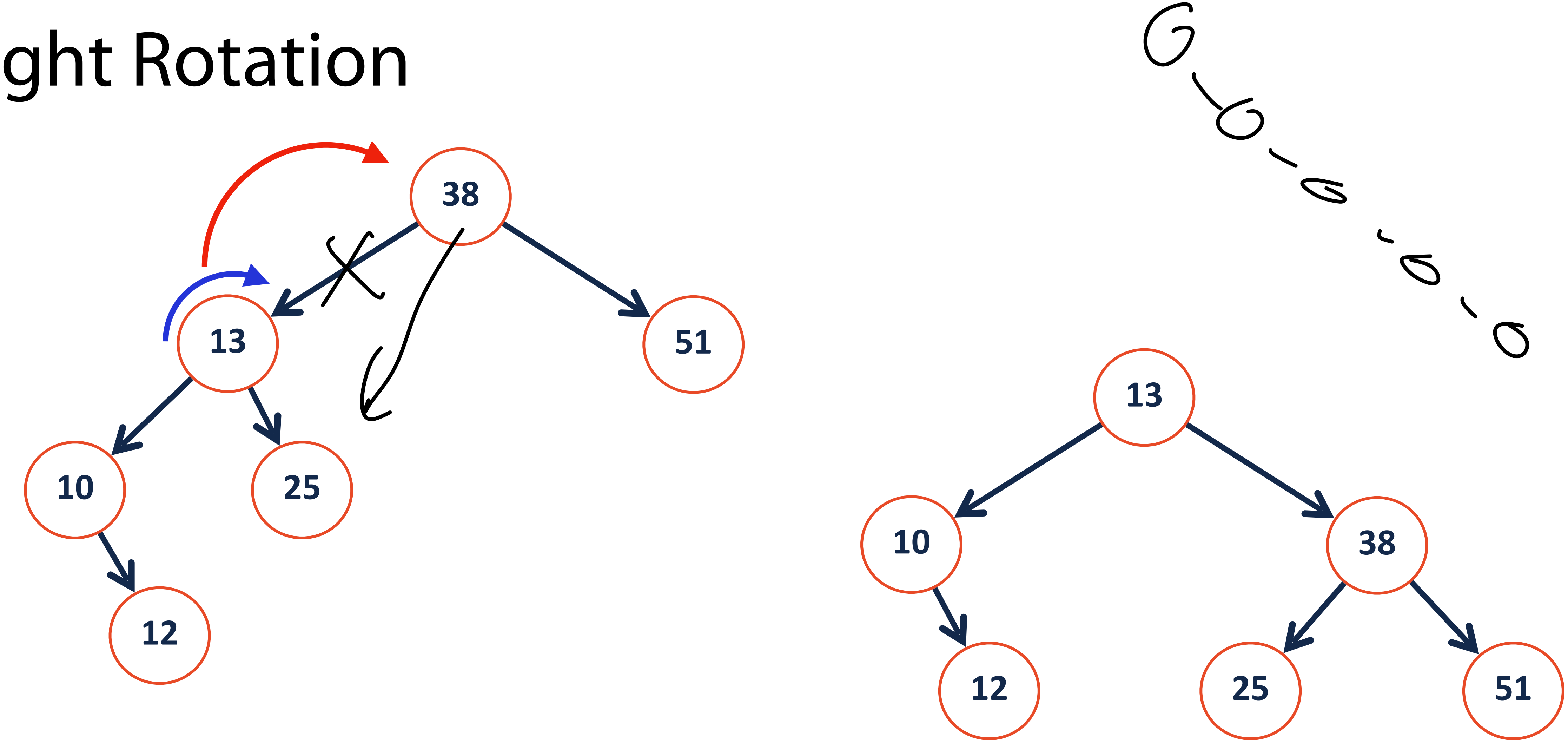


1) Make 13 new root

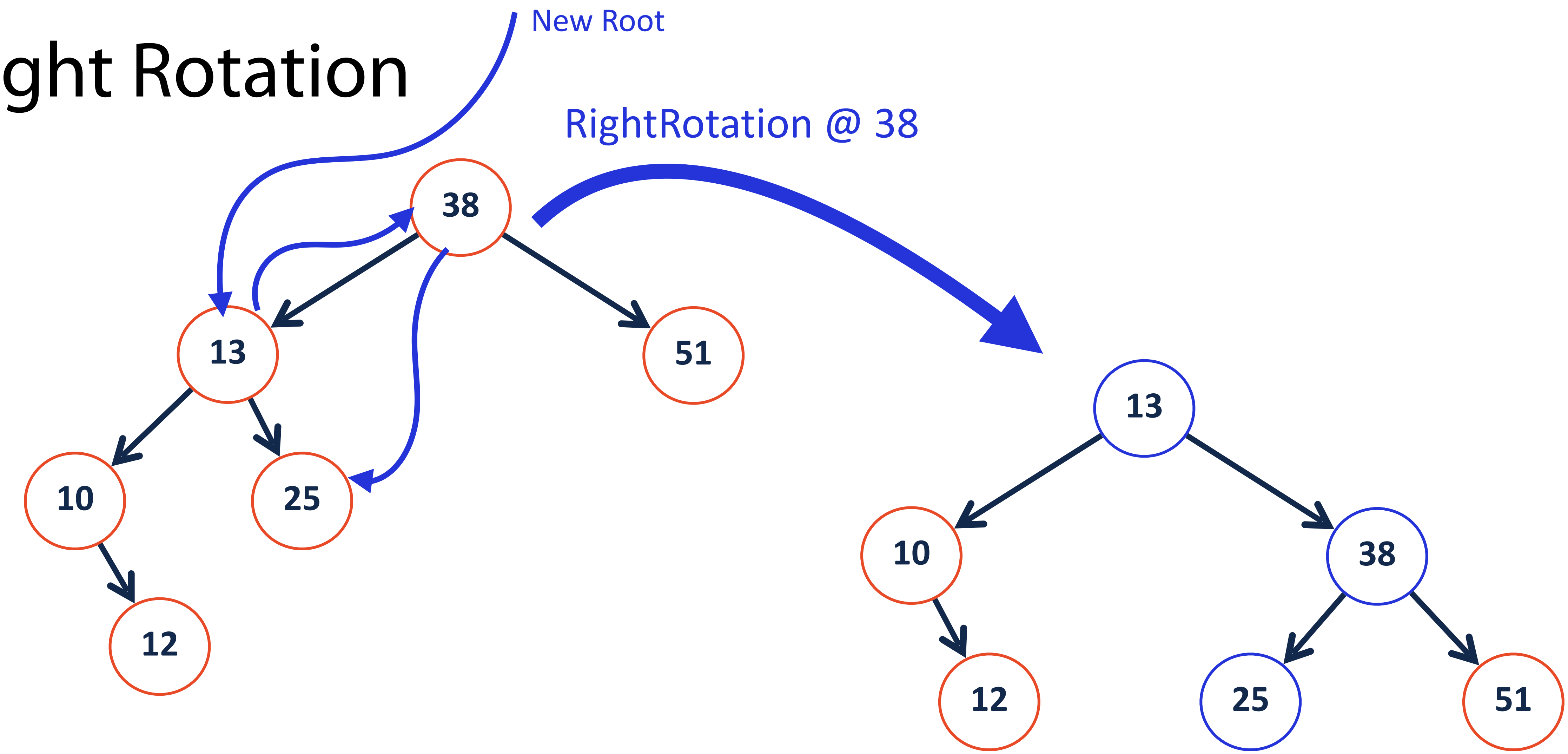
2) Old root left = new root right

3) Make 38 right child of 13

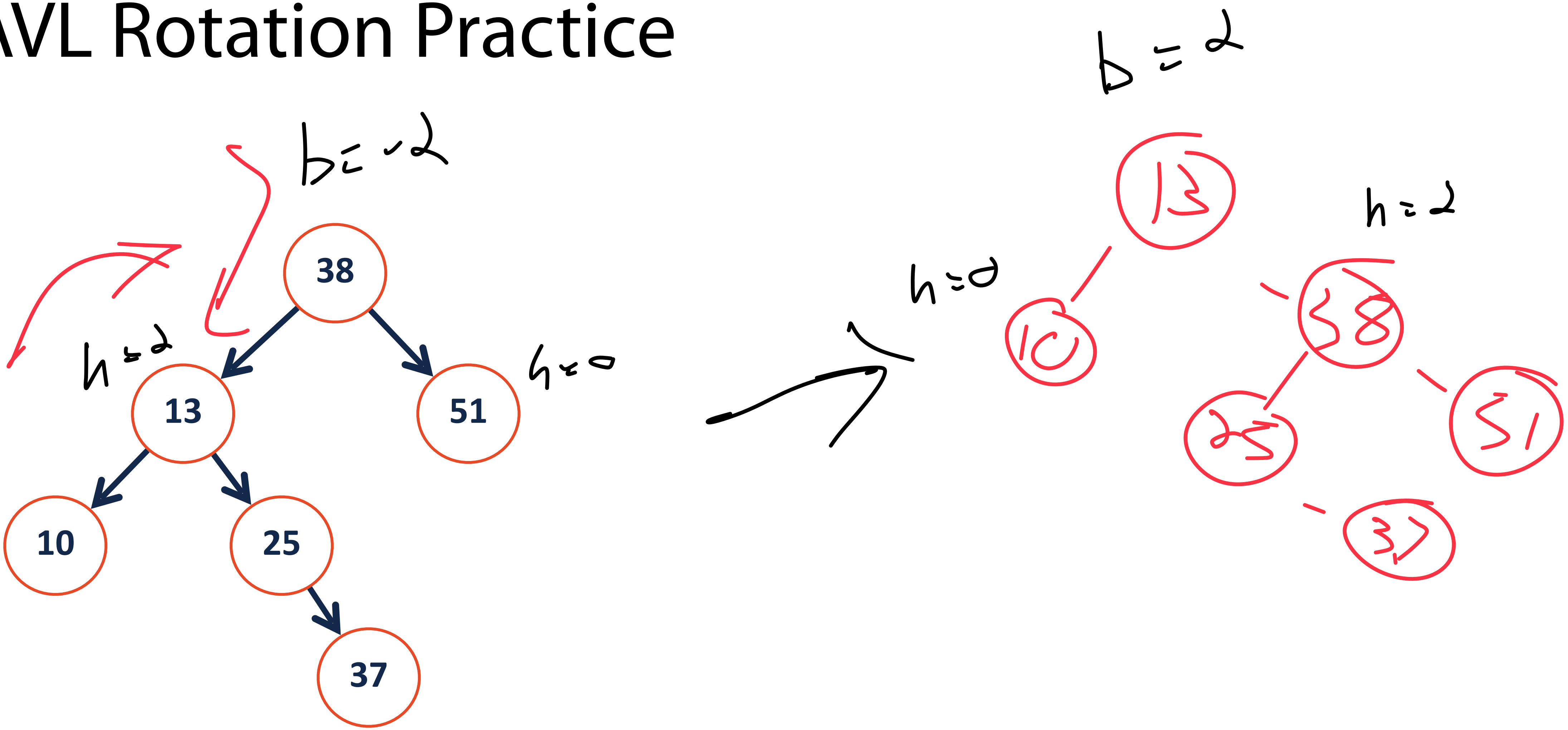
Right Rotation



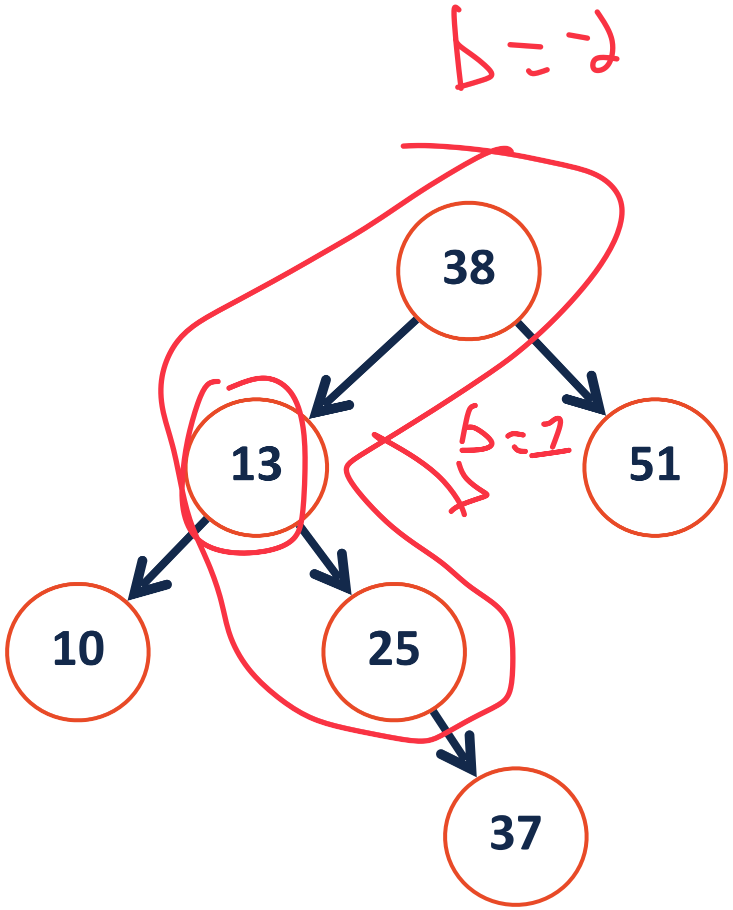
Right Rotation

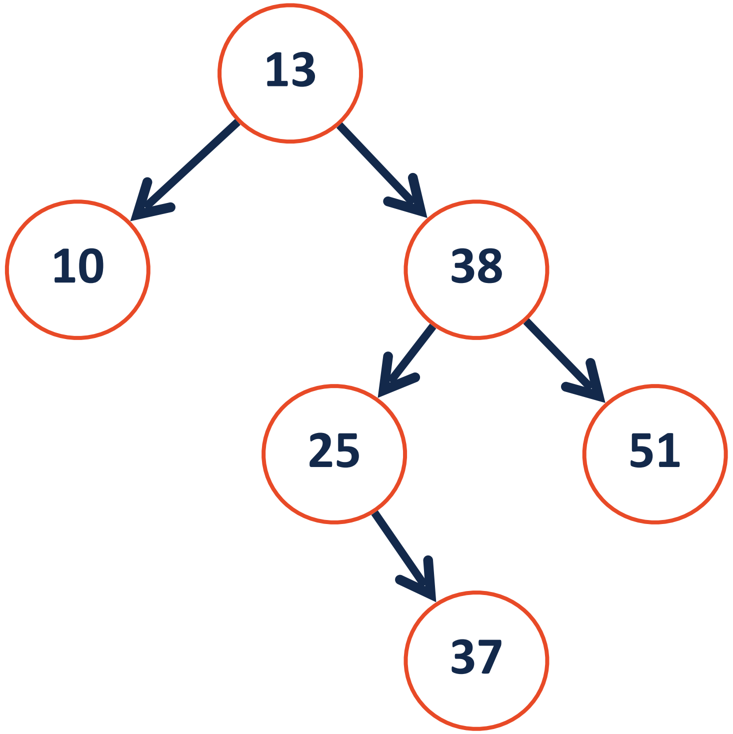


AVL Rotation Practice



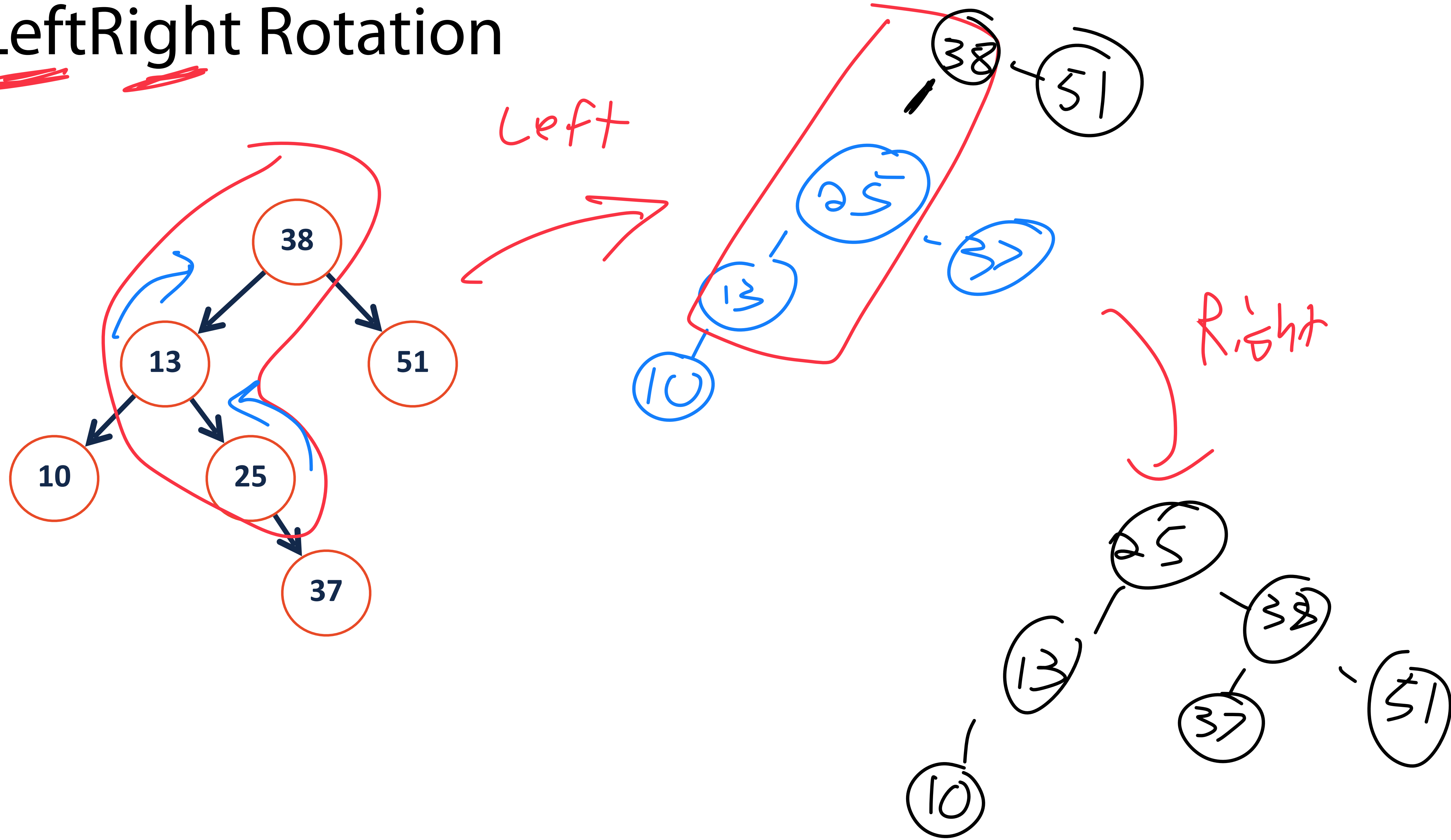
AVL Rotation Practice



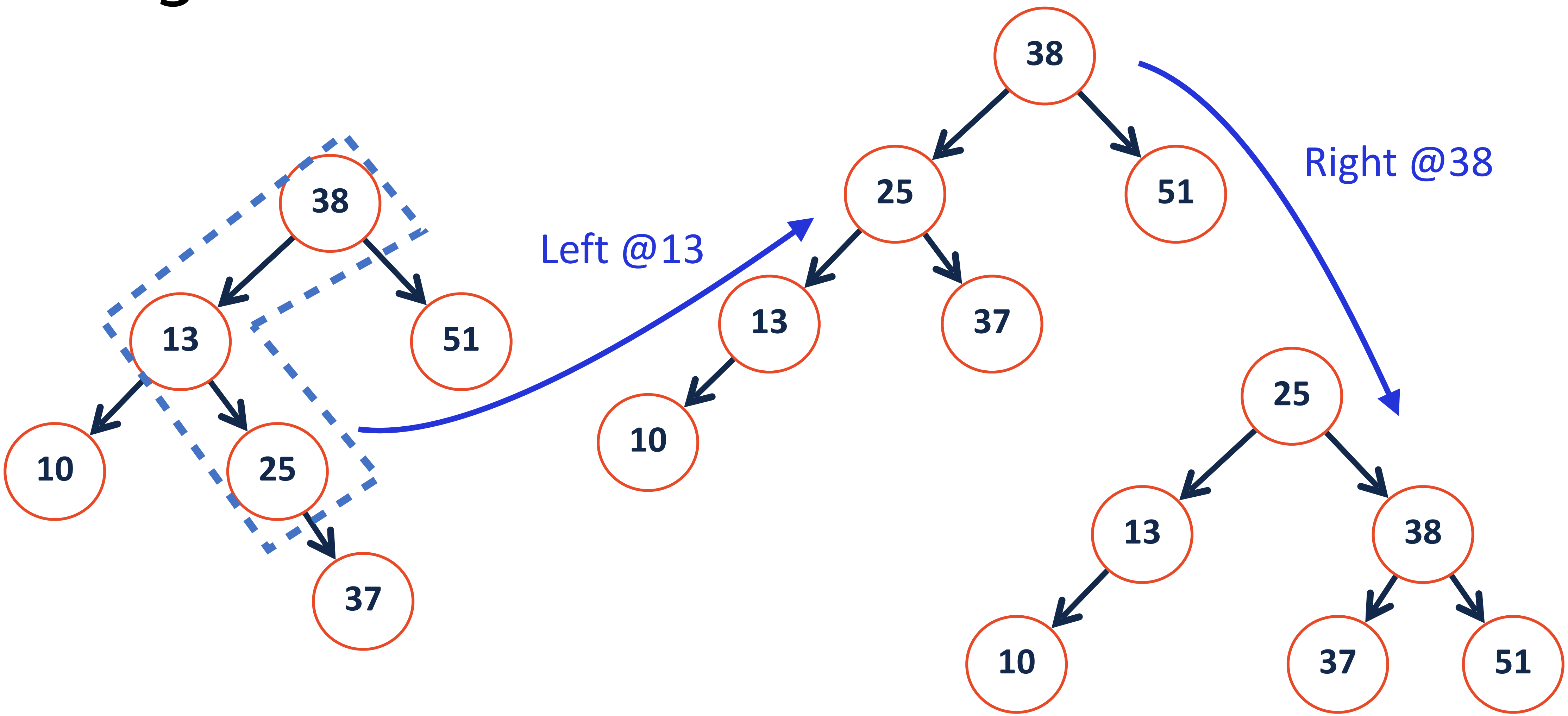


Somethings not quite right…

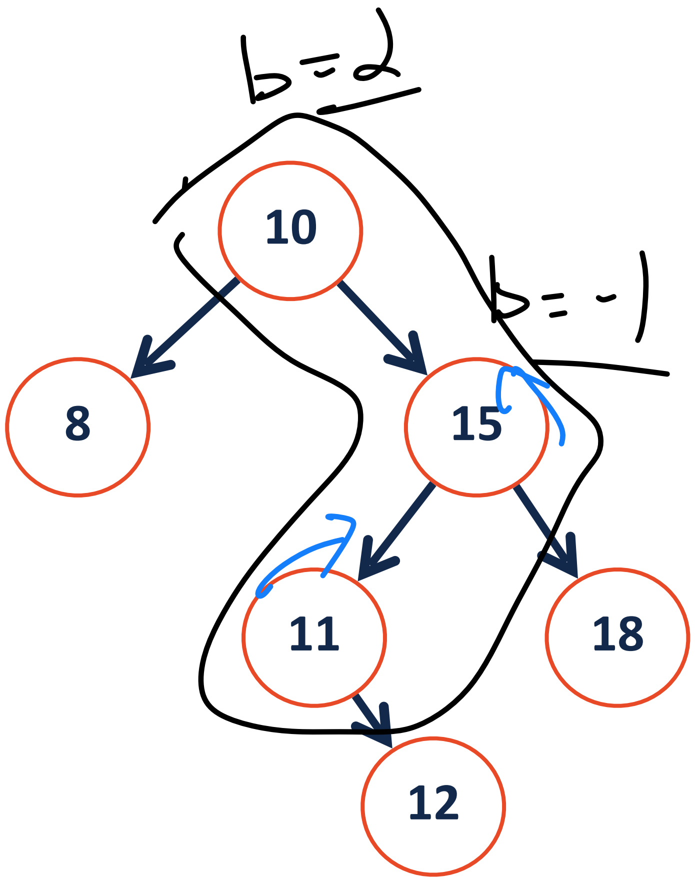
LeftRight Rotation

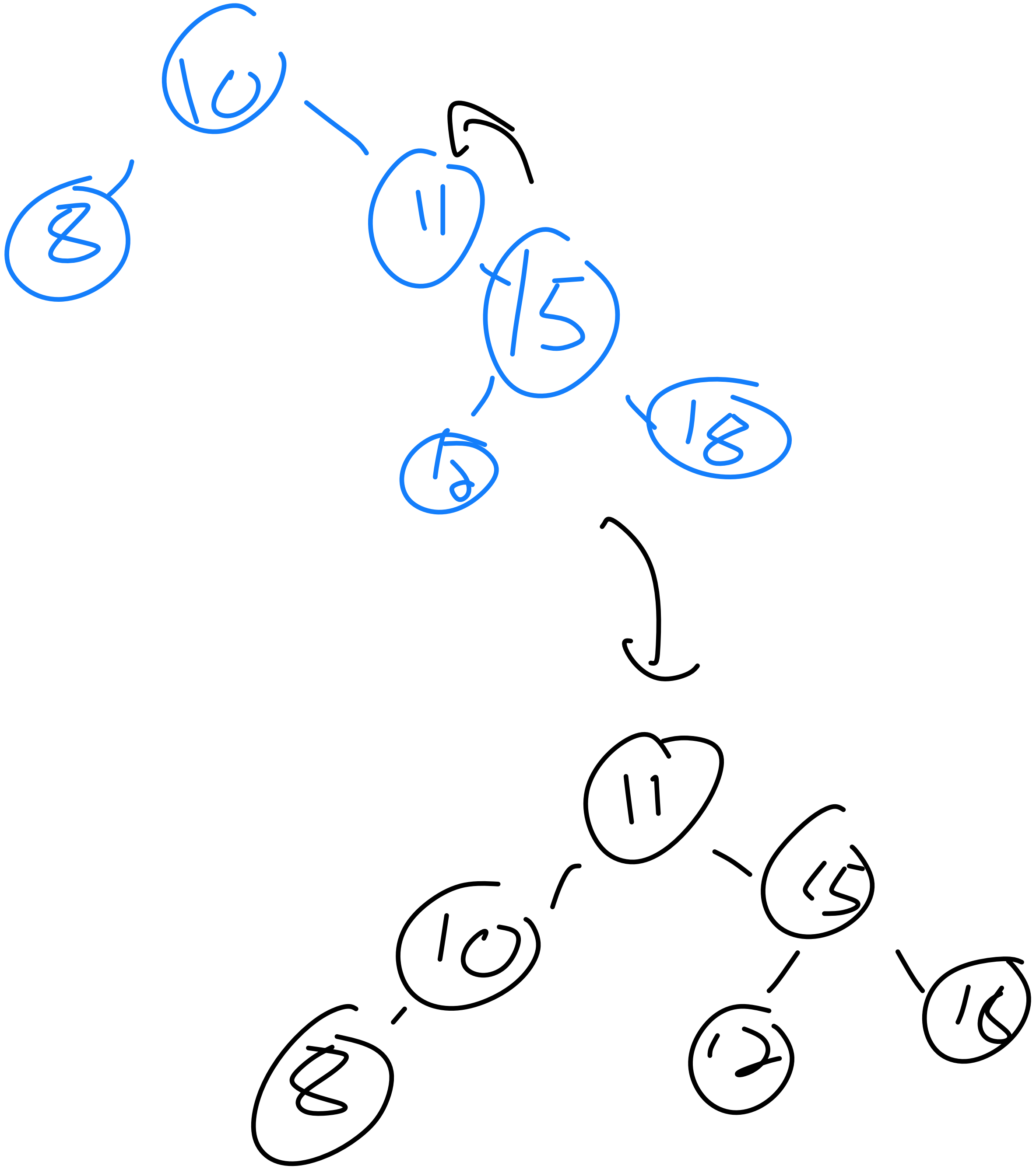


LeftRight Rotation



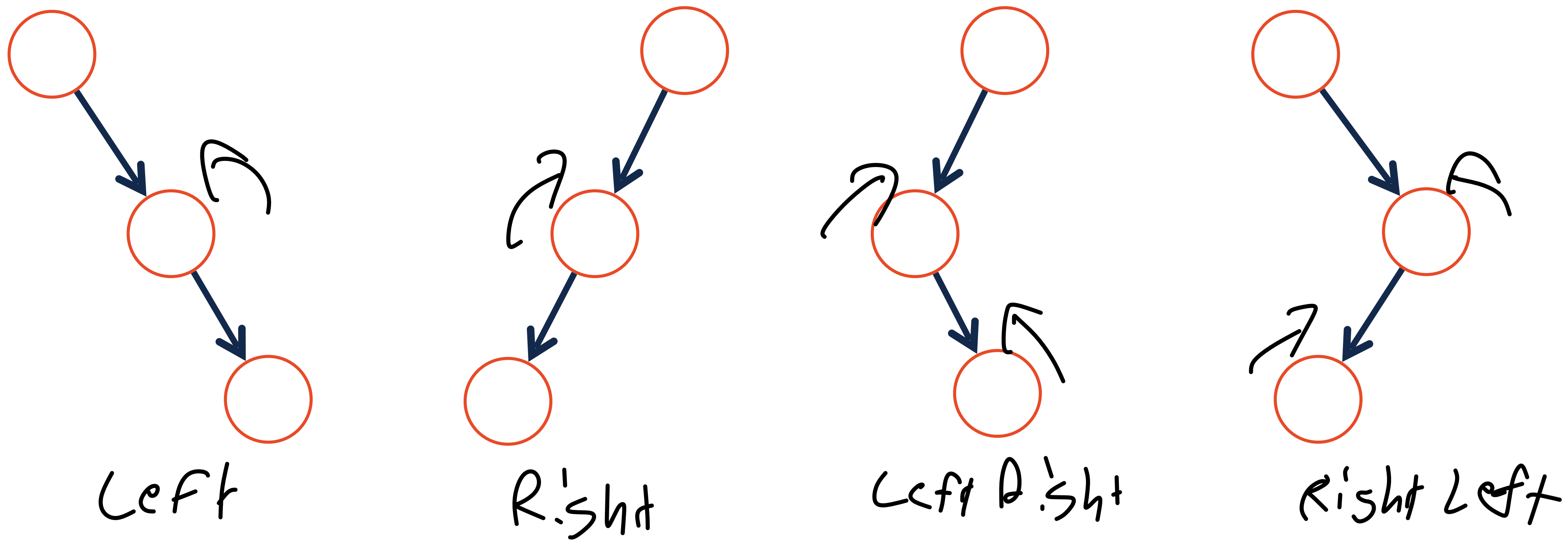
RightLeft Rotation





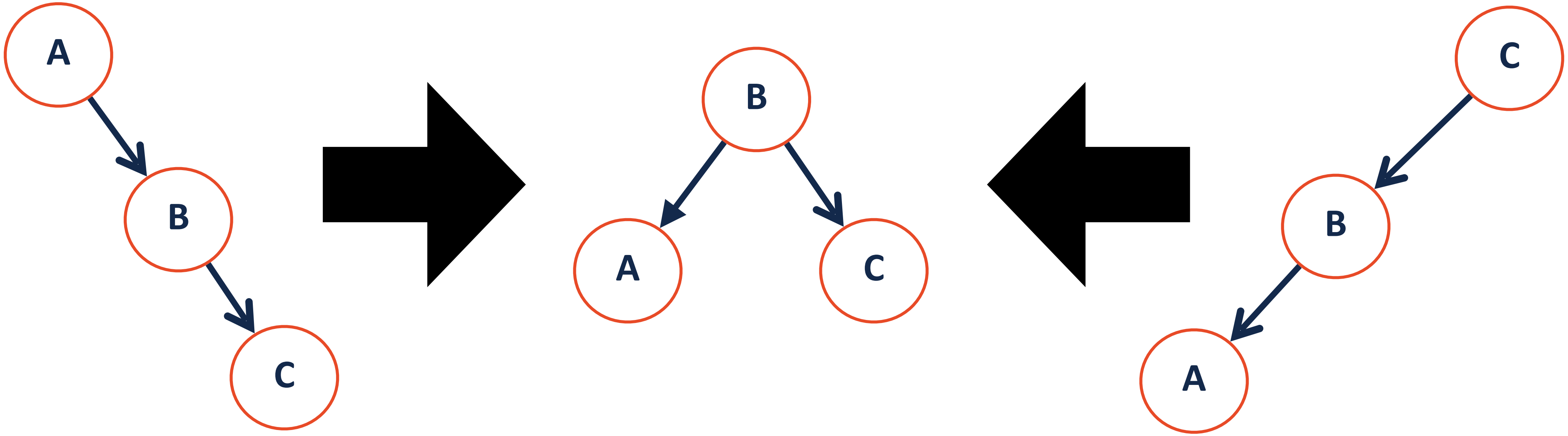
AVL Rotations

Four kinds of rotations:



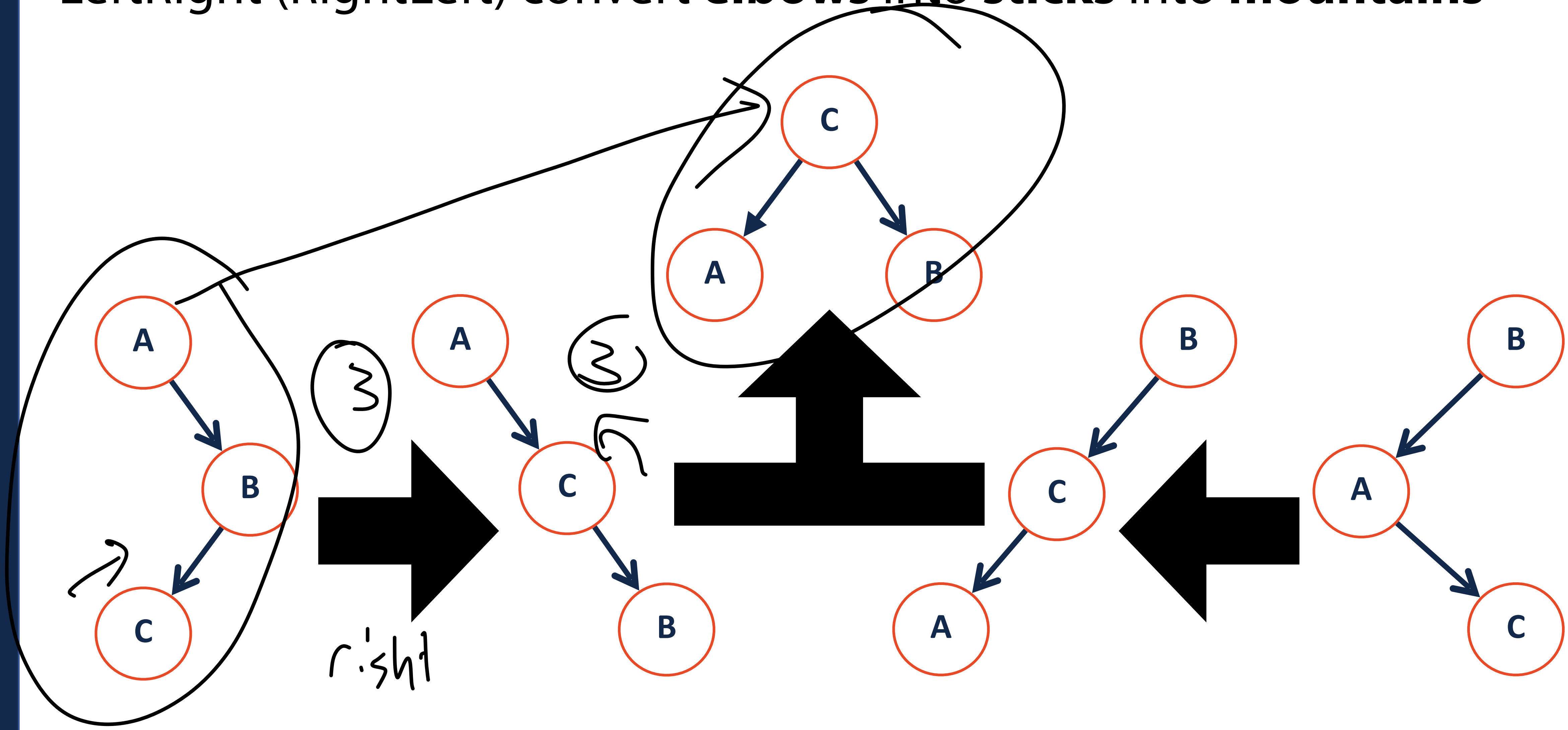
AVL Rotations

Left and right rotation convert **sticks** into **mountains**



AVL Rotations

LeftRight (RightLeft) convert **elbows** into **sticks** into **mountains**



AVL Rotations

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

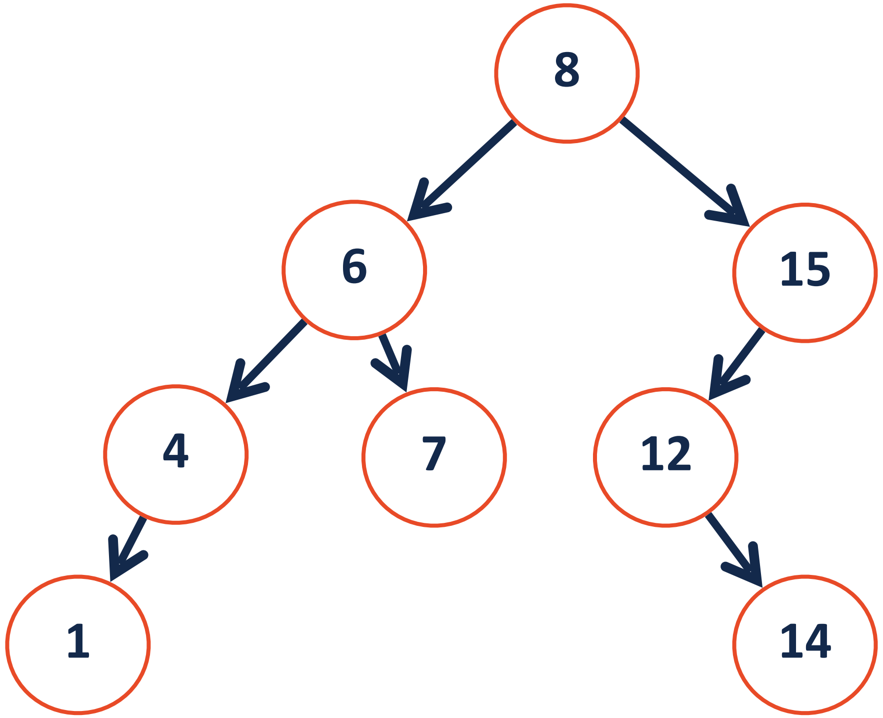
1. All rotations are local (subtrees are not impacted)

2. The running time of rotations are constant

3. The rotations maintain BST property

Goal: We want a height bounded BST

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?

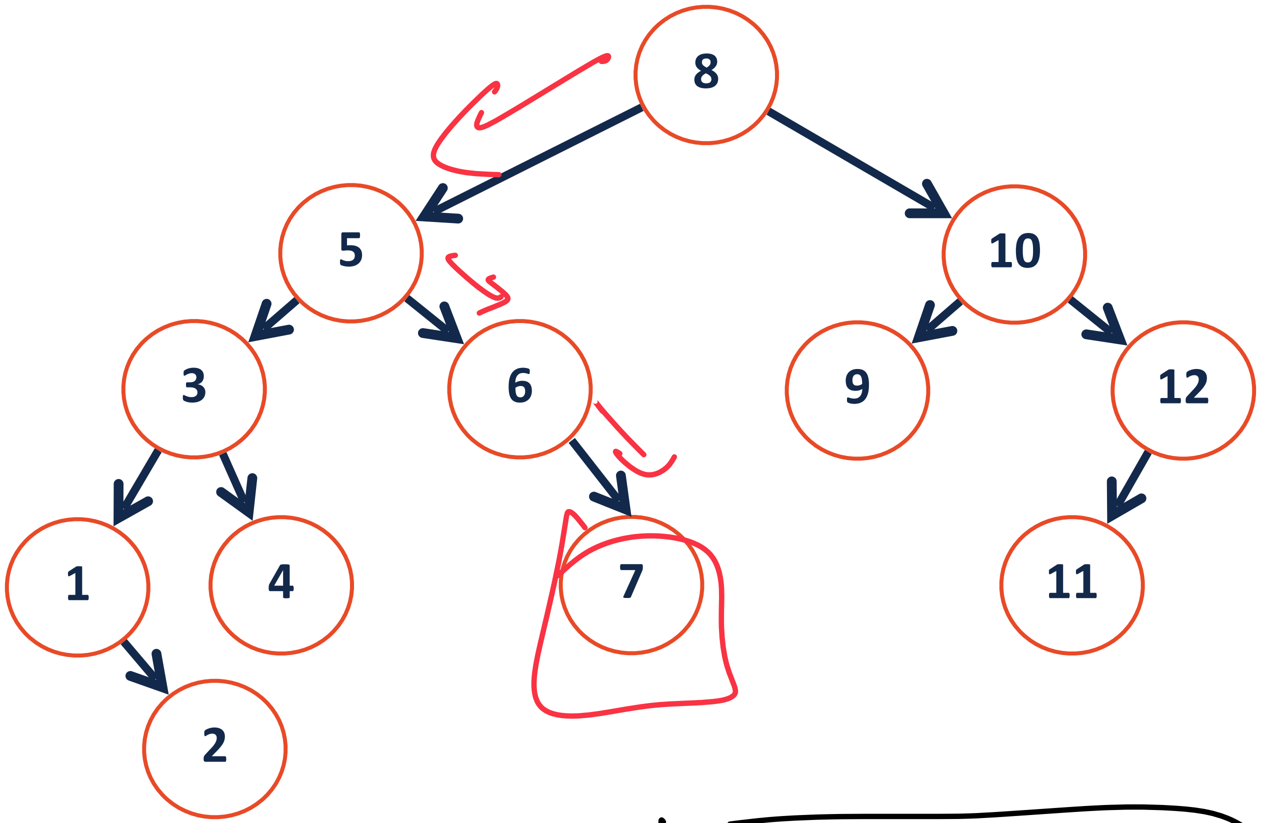
Find

Insert

Remove

AVL Find

`_find(7)`



N nodes = height is $\boxed{O(\log n)}$

AVL Insertion

Given AVL tree
↳ was balanced!

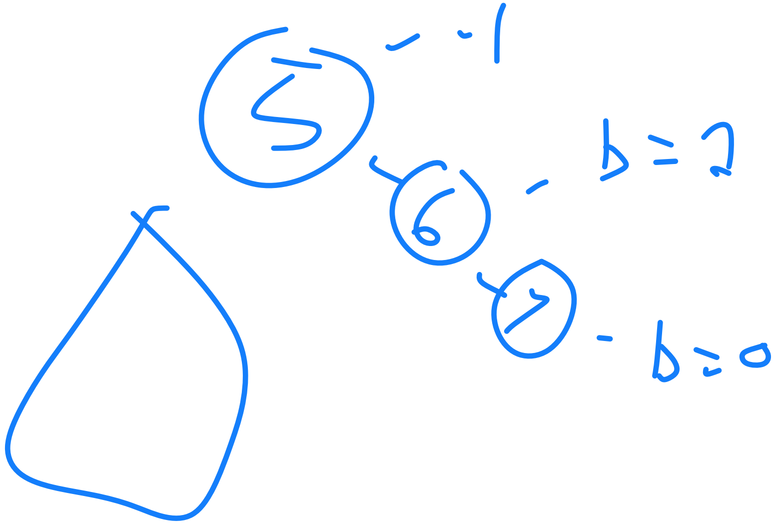
`_insert(6.5)`

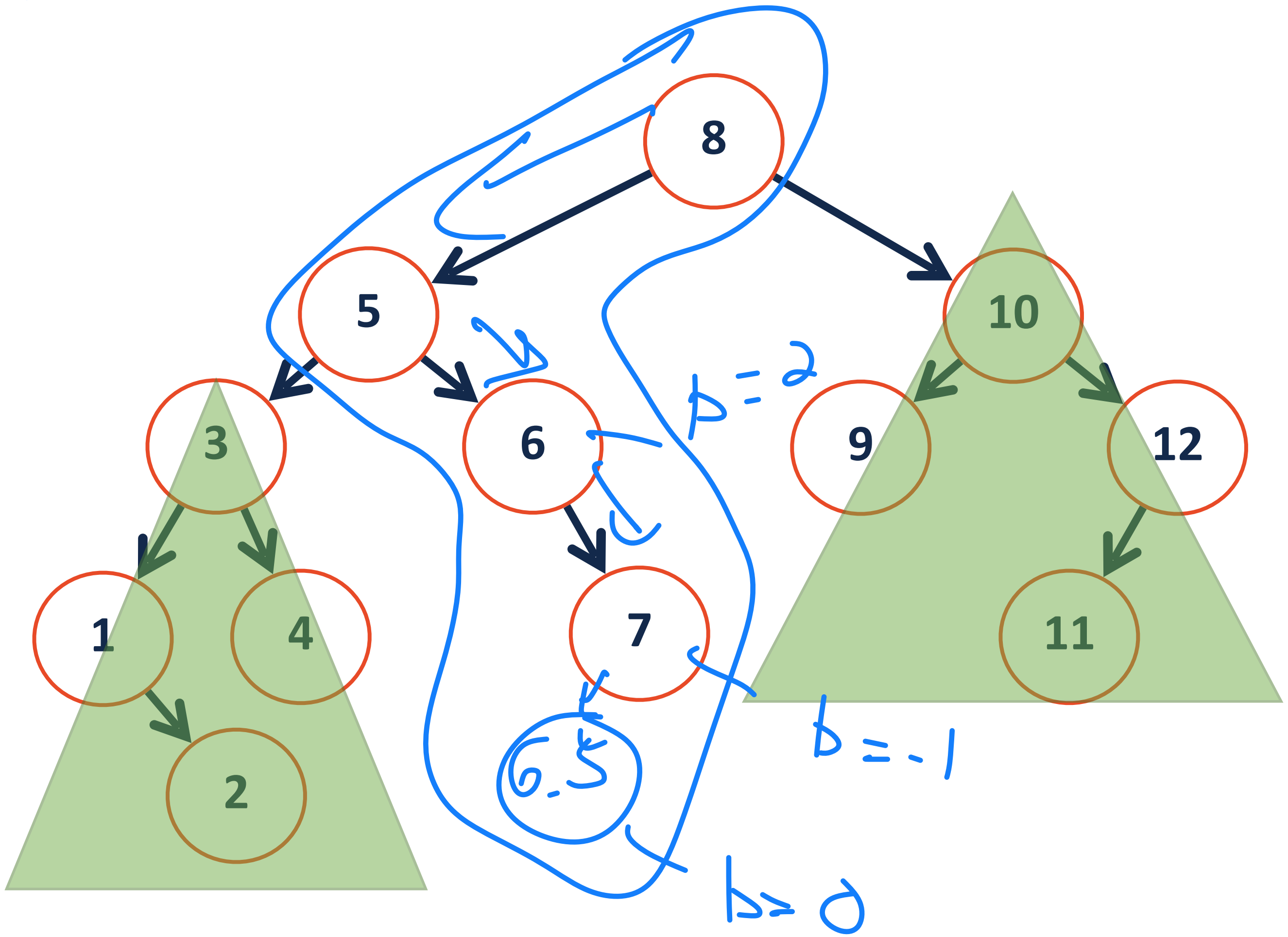
Insert (pseudo code):

1: Insert at proper place

2: Check for imbalance

3: Rotate, if necessary





AVL Insertion

`_insert(6.5)`

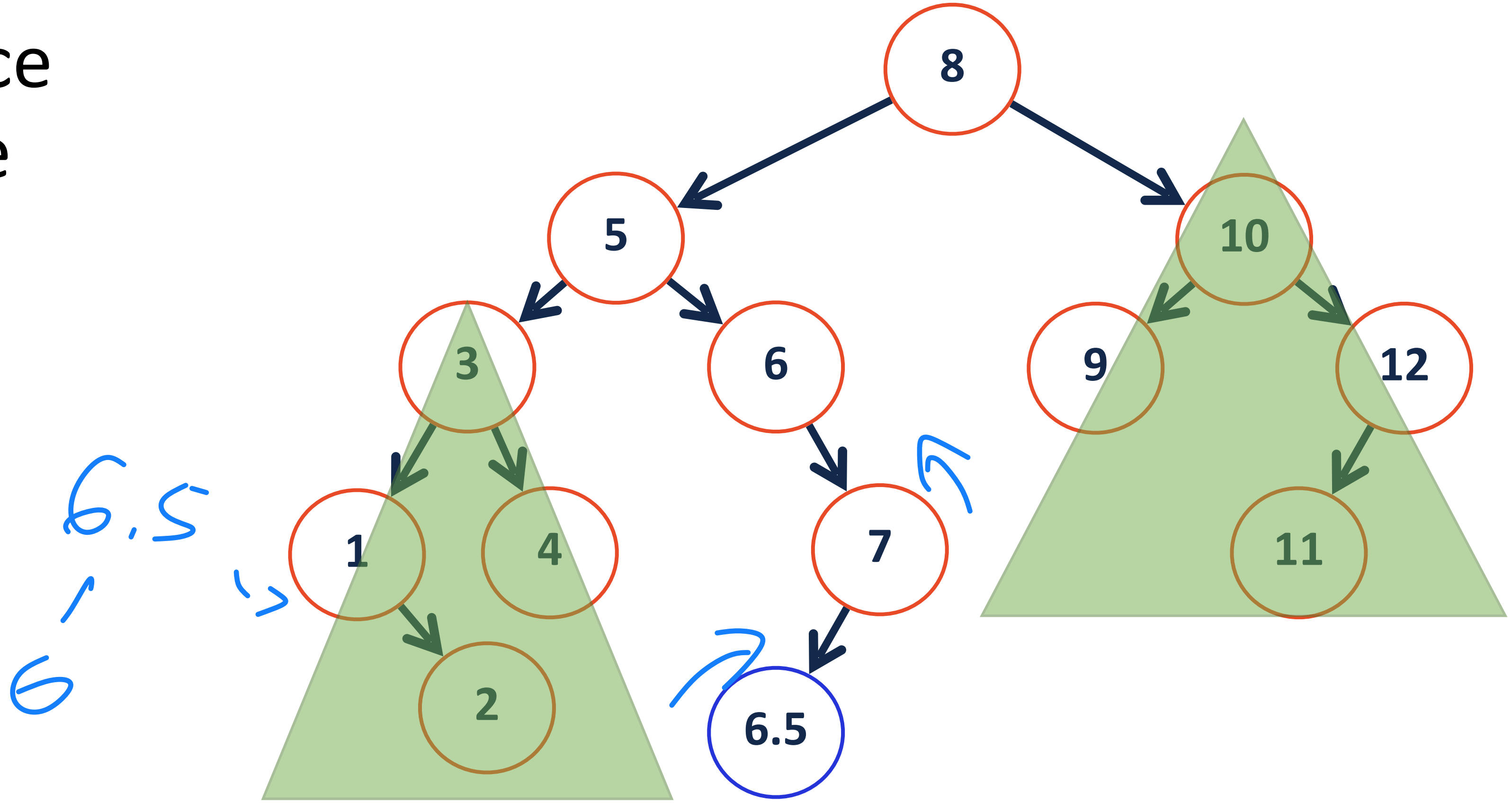
Insert (pseudo code):

1: Insert at proper place

2: Check for imbalance

3: Rotate, if necessary

6 6.5



AVL Insertion

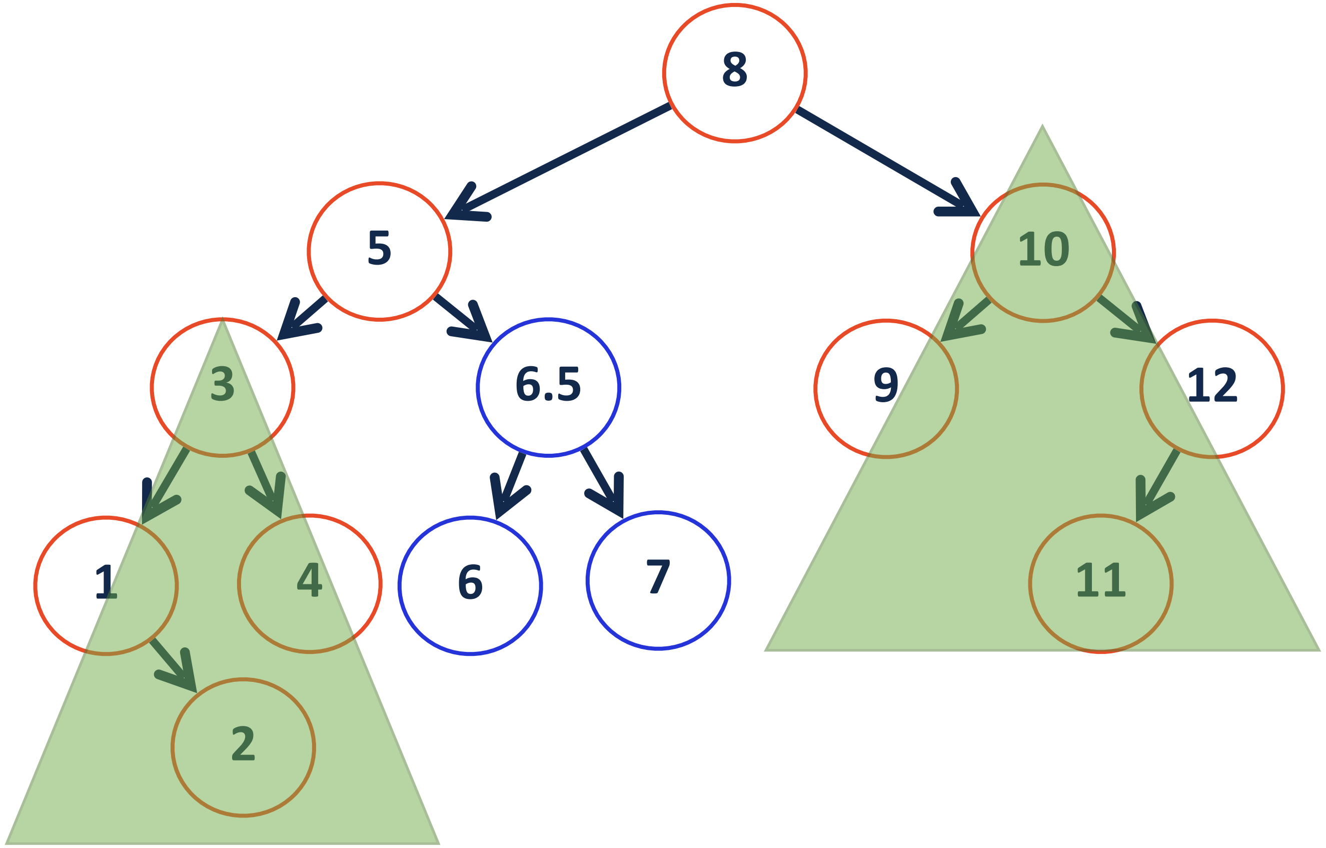
`_insert(6.5)`

Insert (pseudo code):

1: Insert at proper place

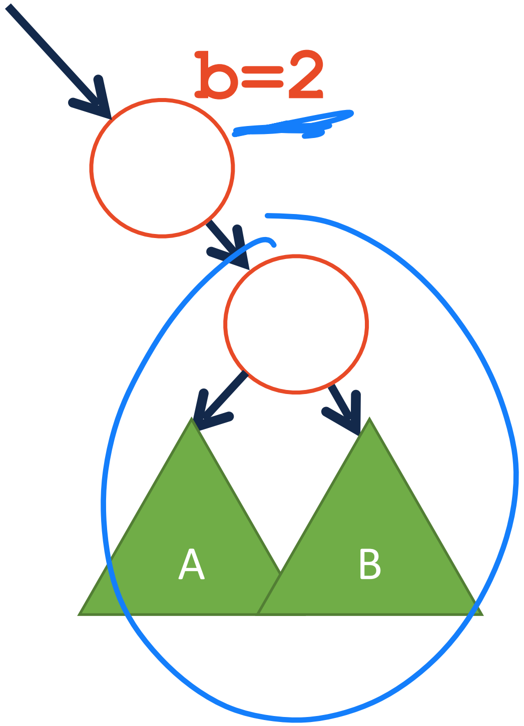
2: Check for imbalance

3: Rotate, if necessary



AVL Insertion – what rotation to call for what insert

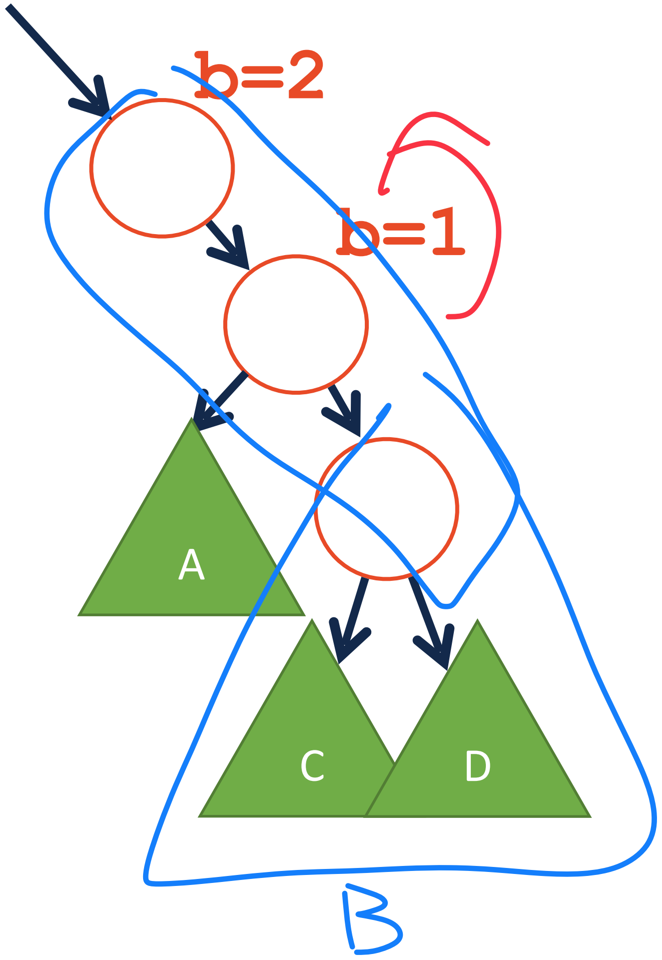
Given an AVL is balanced, insert can insert **at most** one imbalance

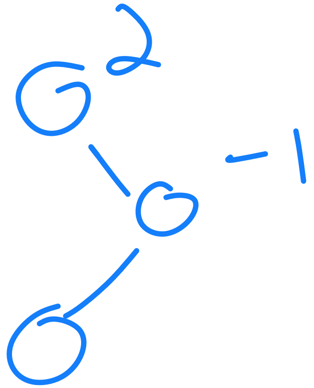


AVL Insertion

If we insert in B, I must have a balance pattern of **2, 1**

Conceptual point: (+) #s are right heavy



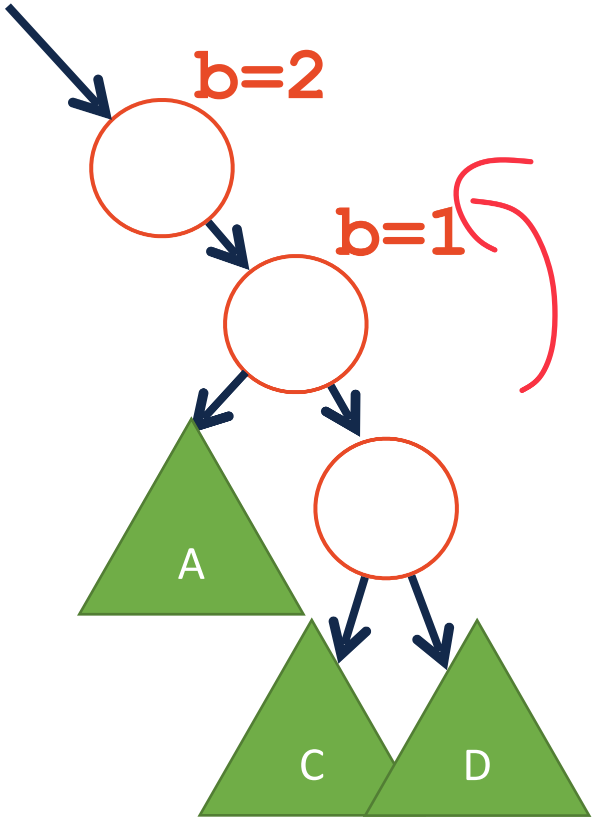


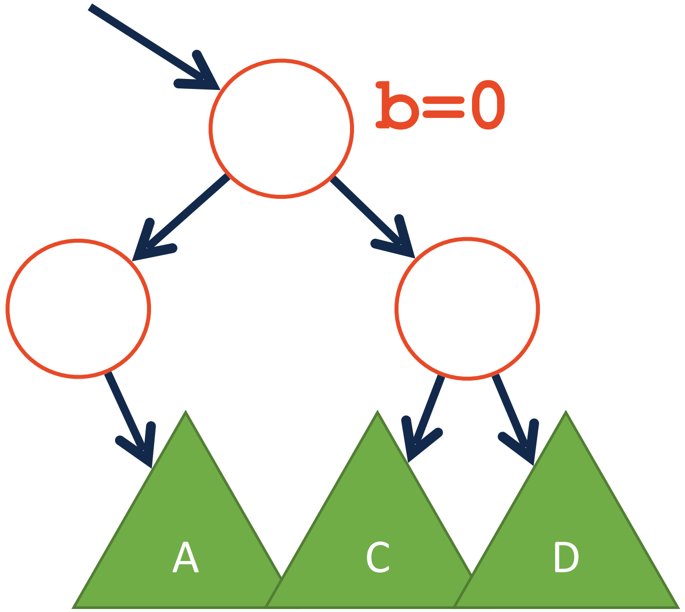
Right heavy in right direction

↳ Left rotation

AVL Insertion

A **left** rotation fixes our imbalance in our local tree.



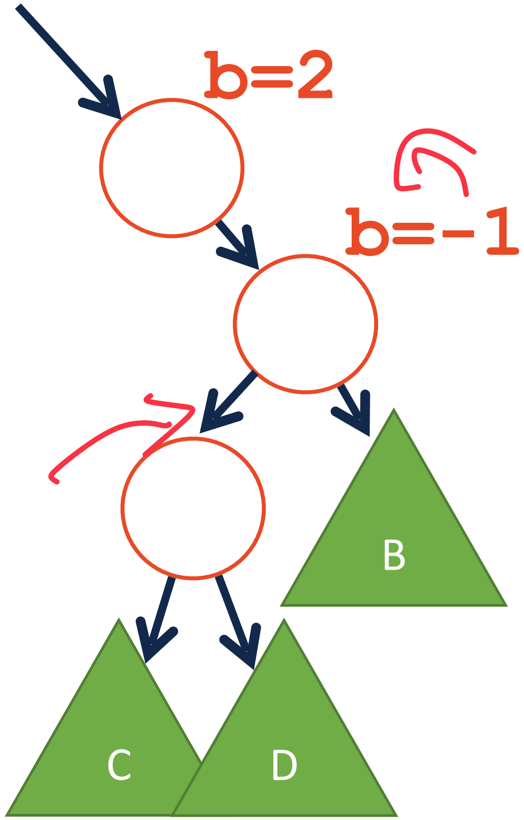


After rotation, subtree has **pre-insert height**. (Overall tree is balanced)

AVL Insertion

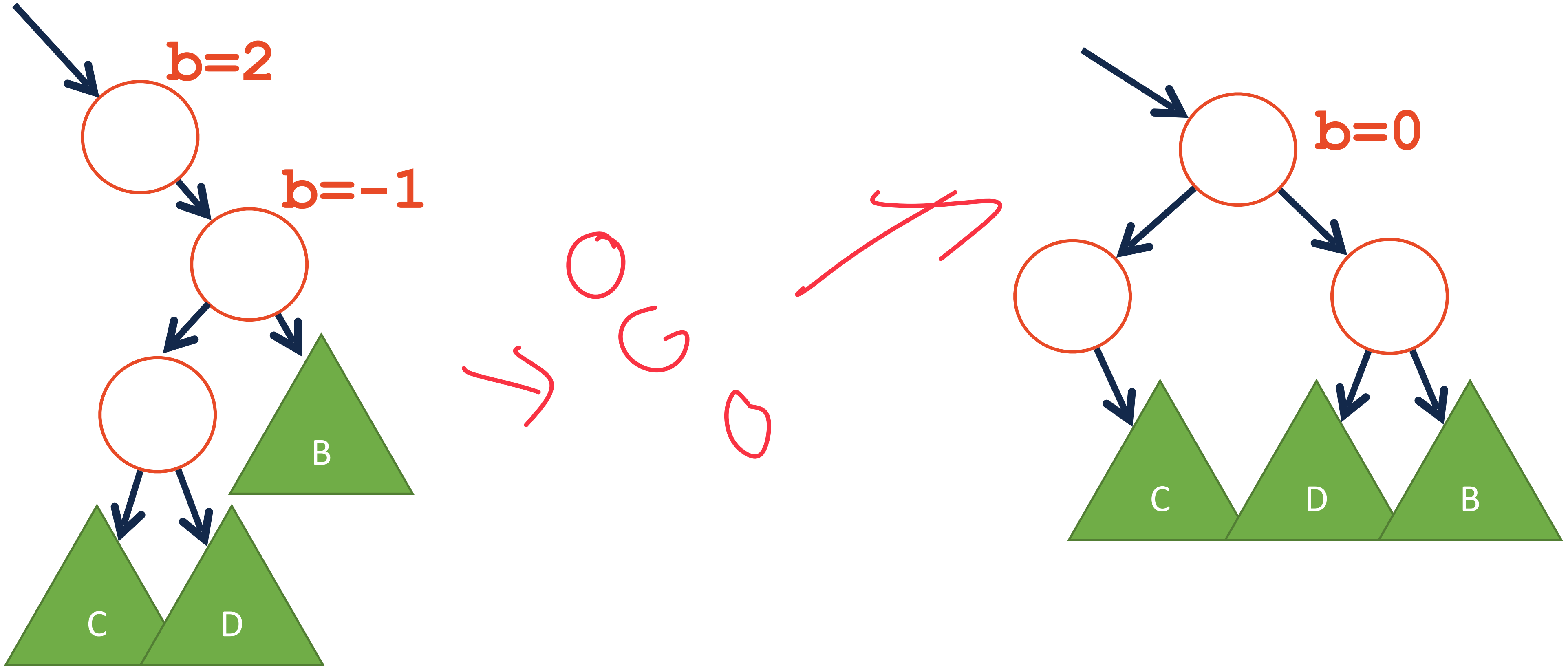
If we insert in A, I must have a balance pattern of **2, -1**

2, -1 → right left



AVL Insertion

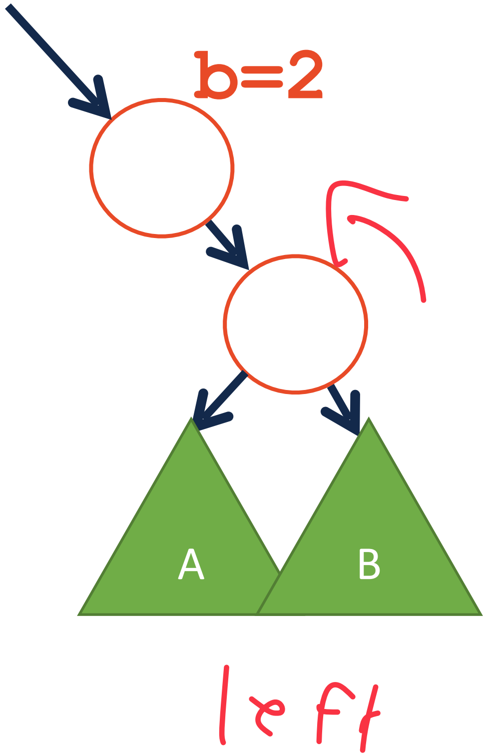
A **rightLeft** rotation fixes our imbalance in our local tree.

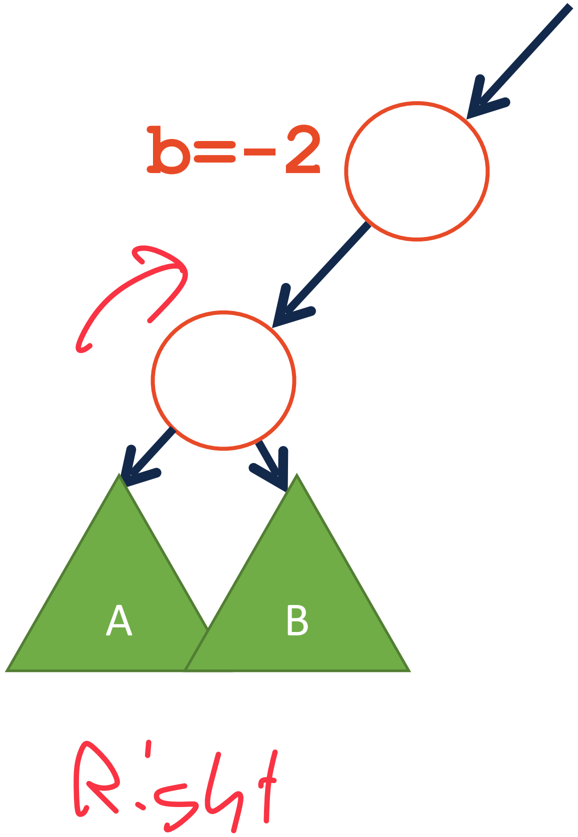


After rotation, subtree has **pre-insert height**. (Overall tree is balanced)

AVL Insertion

The other rotations are a direct mirror:



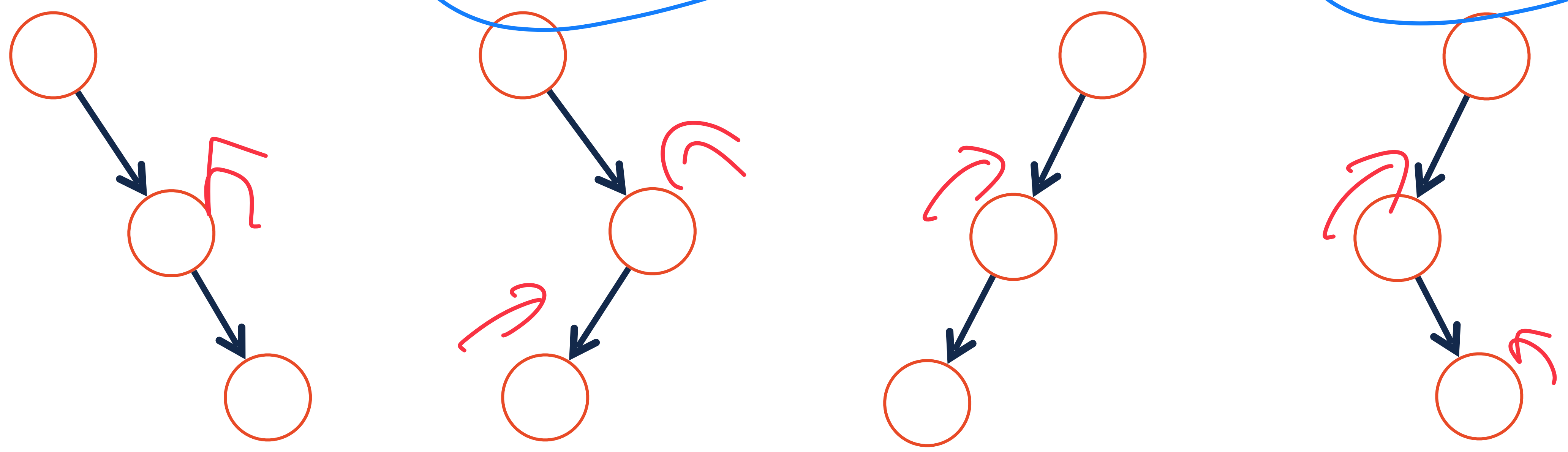


AVL Insertion

(+) is right

If we know our imbalance direction, we can call the correct rotation.

Left | **RightLeft** | **Right** | **LeftRight**



2, 1 | 2, -1 | -2, -1 | -2, 1

AVL Insertion

Insert *may* increase height by at most:

A rotation reduces the height of the subtree by:

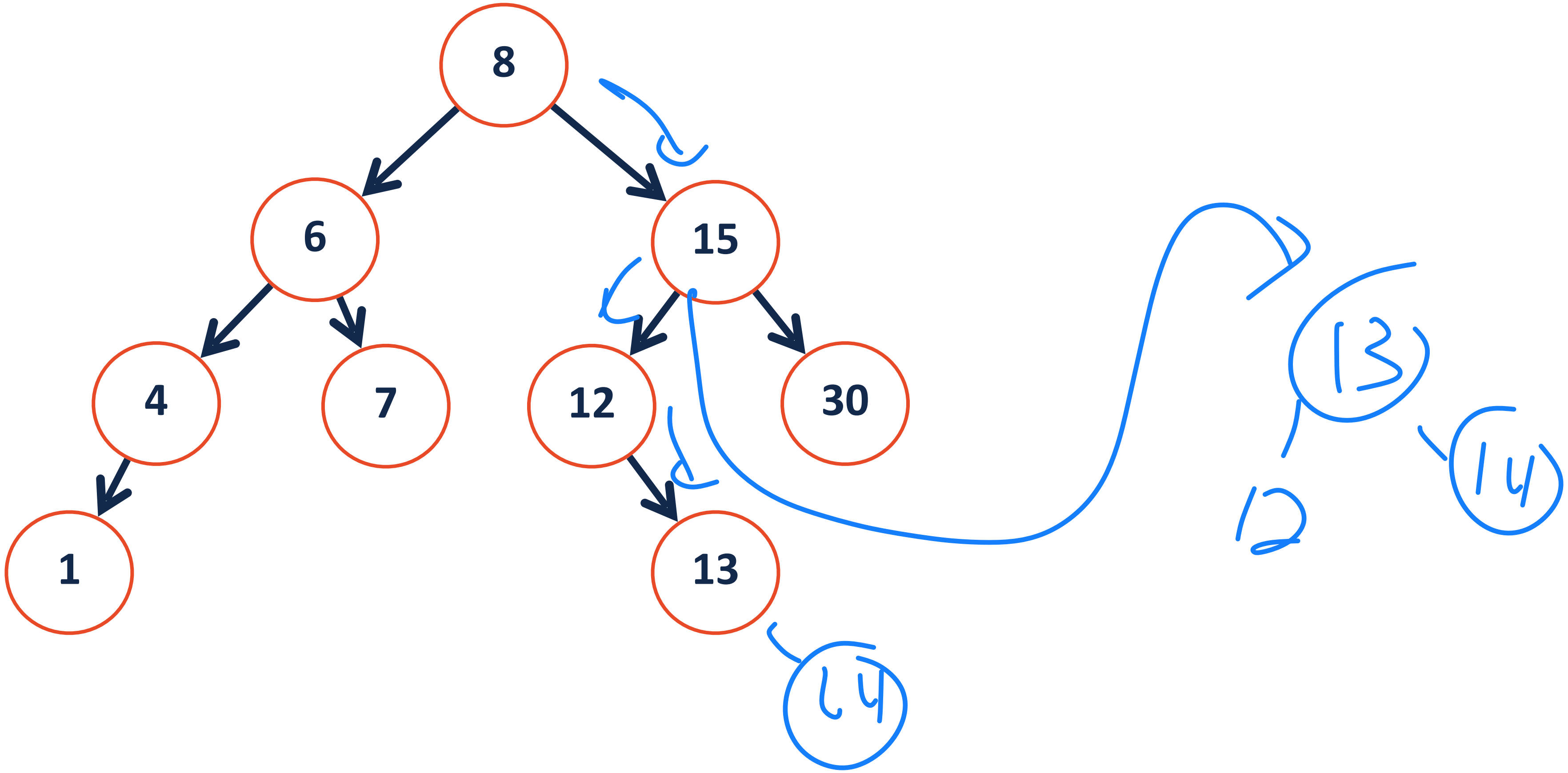
A single* rotation restores balance and corrects height!

What is the Big O of performing our rotation? $O(1)$

What is the Big O of insert? $O(\log n)$

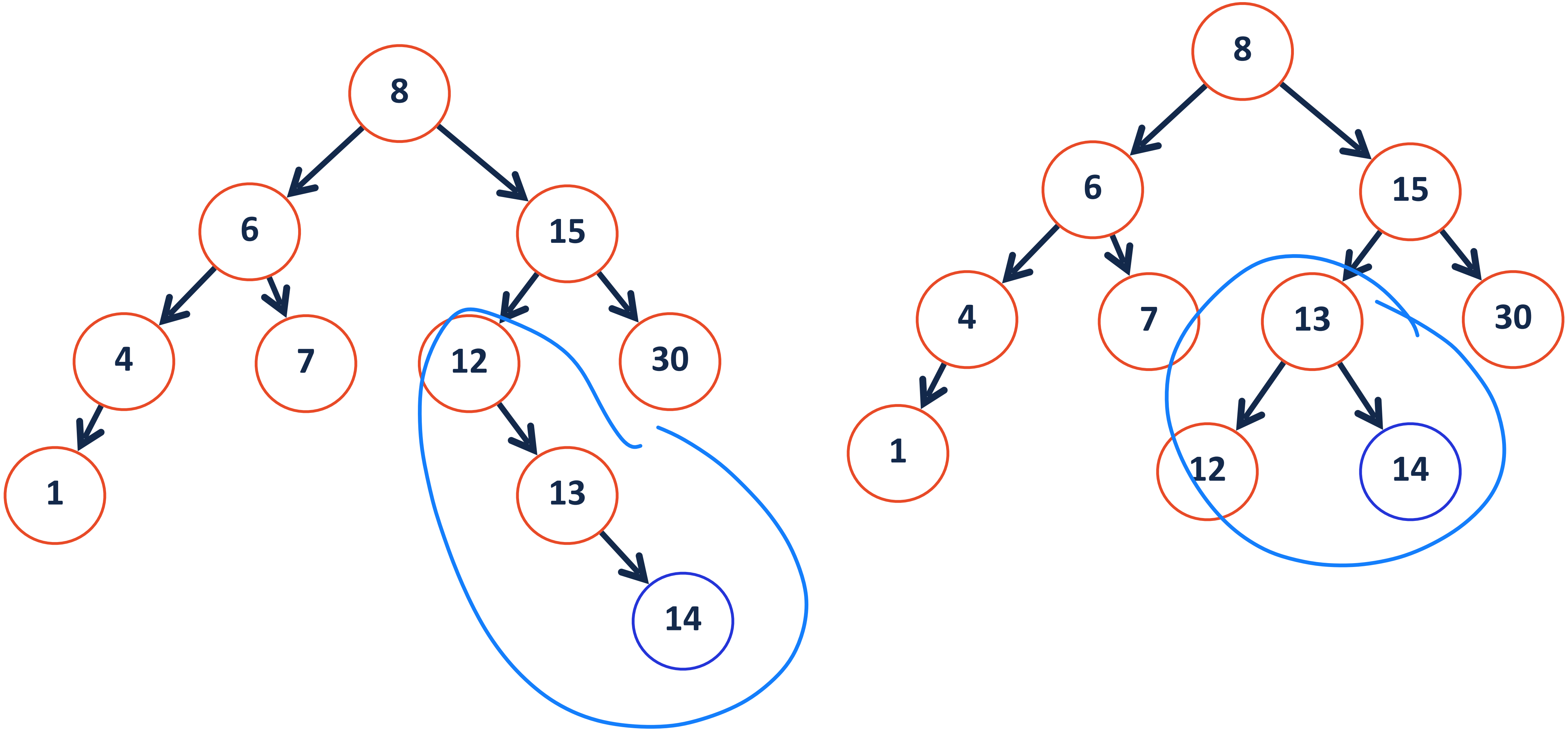
AVL Insertion Practice

`_insert(14)`



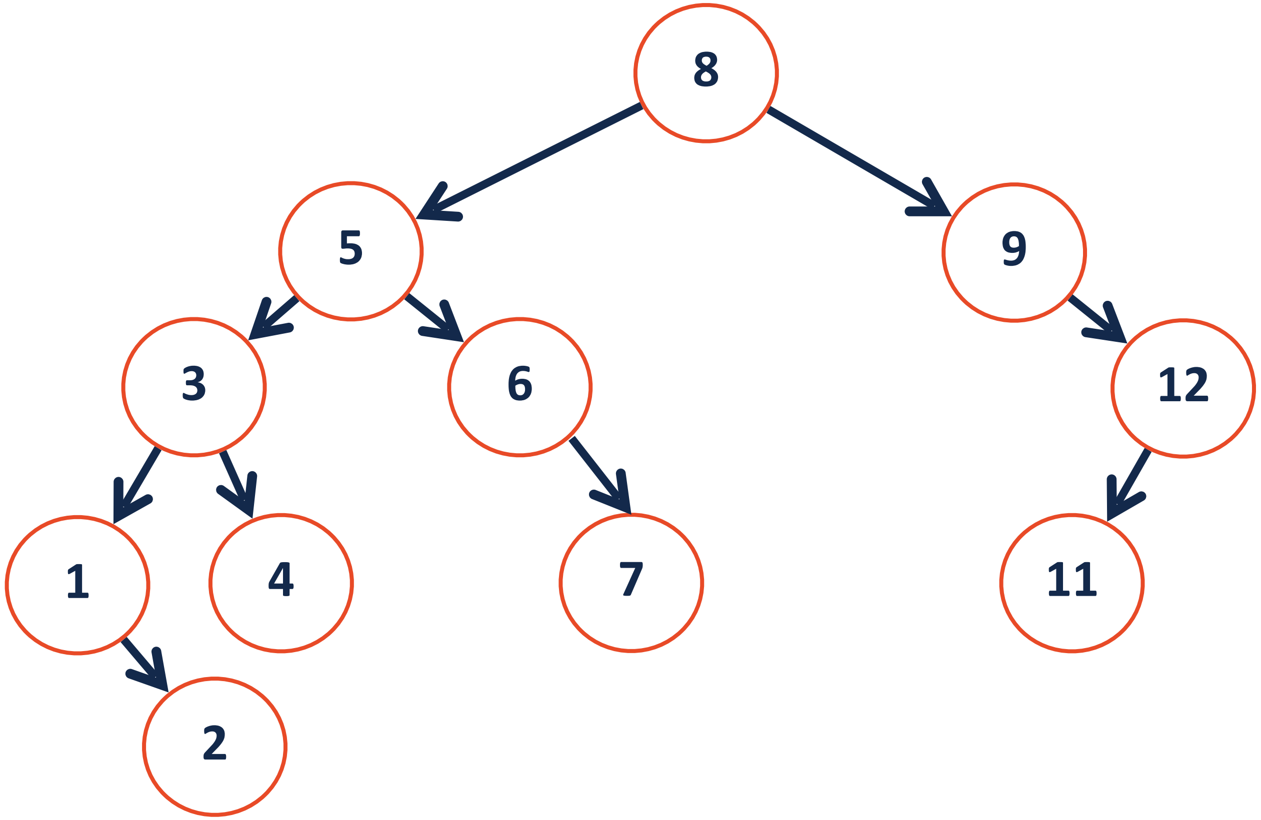
AVL Insertion Practice

`_insert(14)`



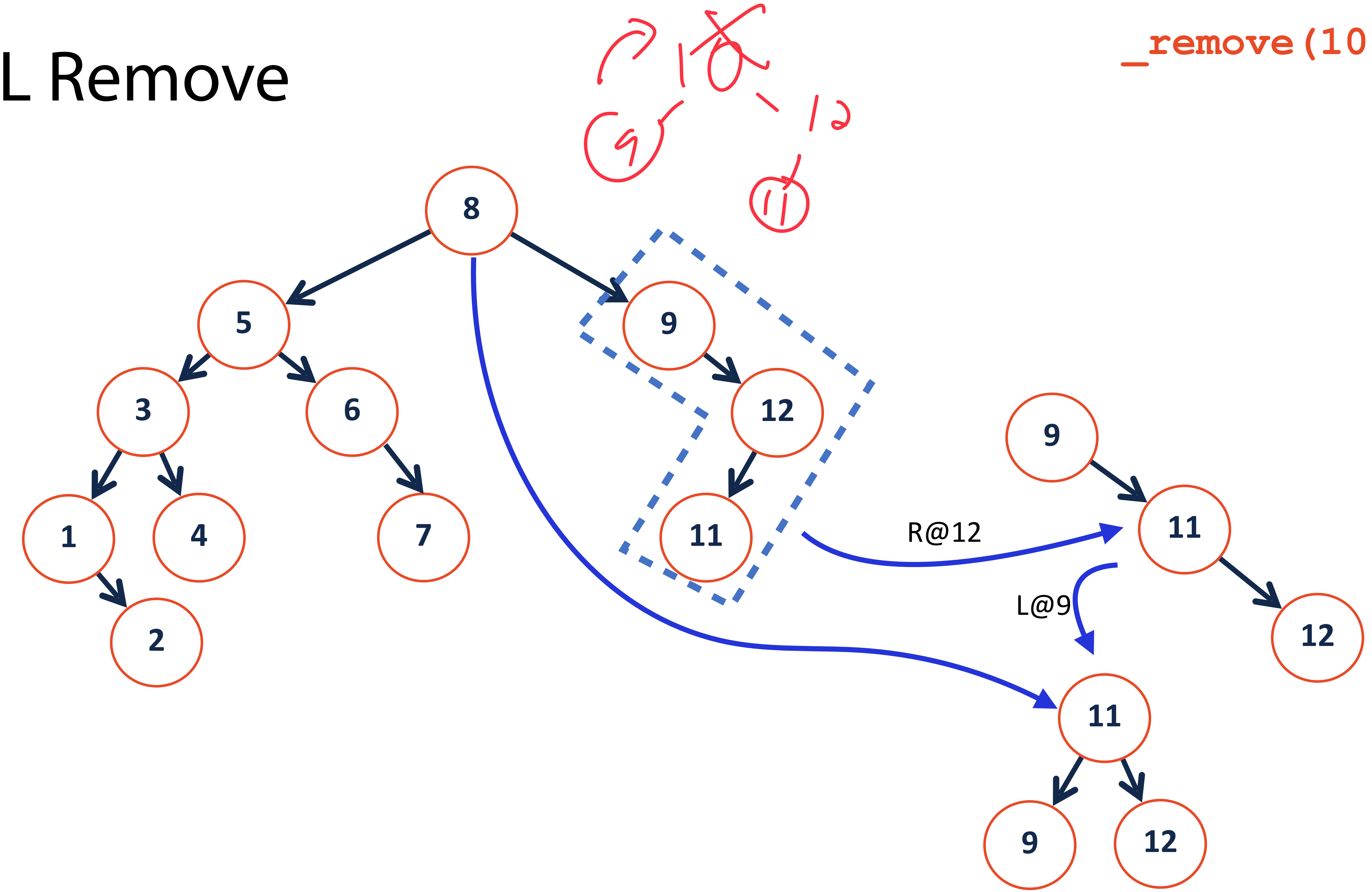
AVL Remove

`_remove(10)`



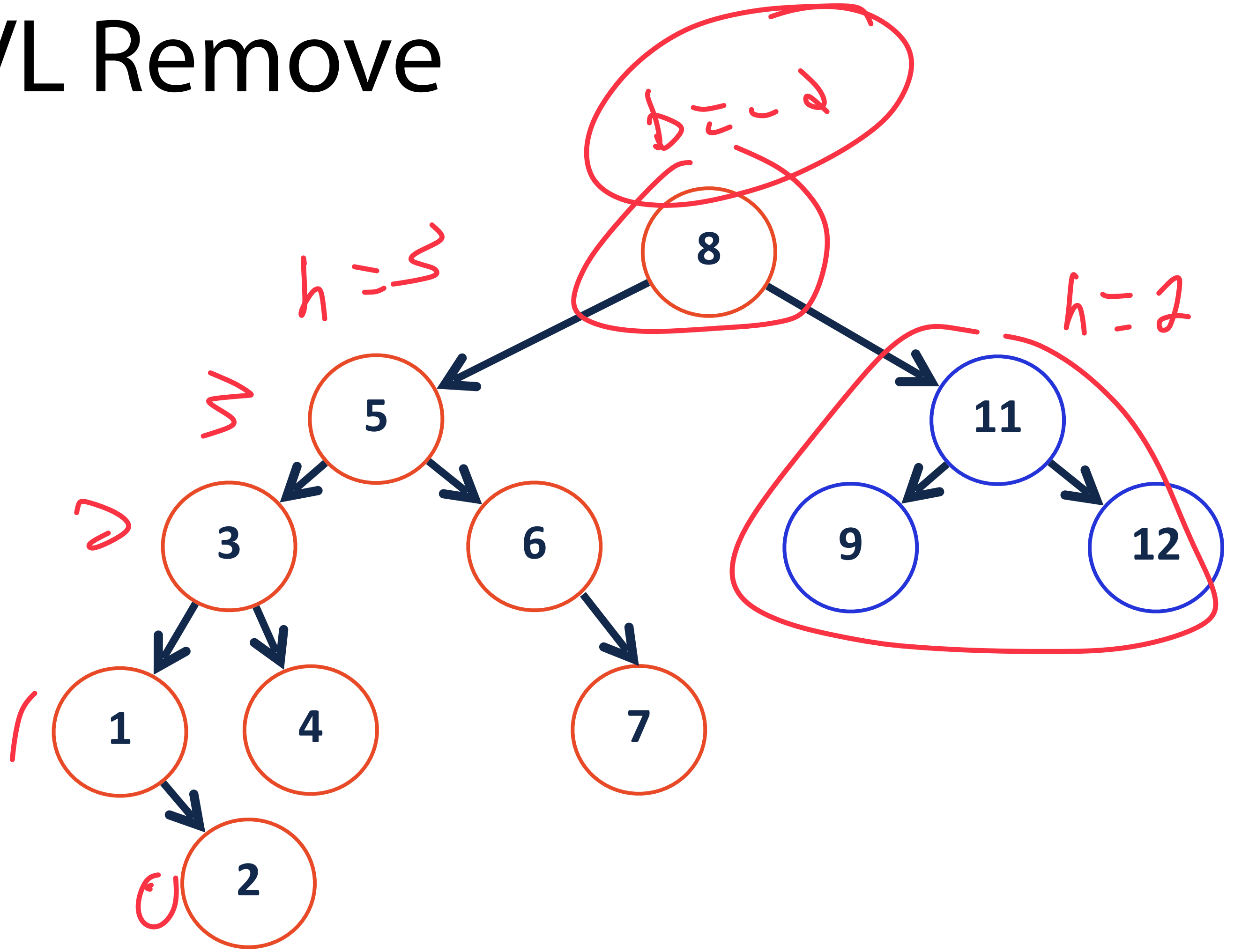
AVL Remove

`_remove(10)`



AVL Remove

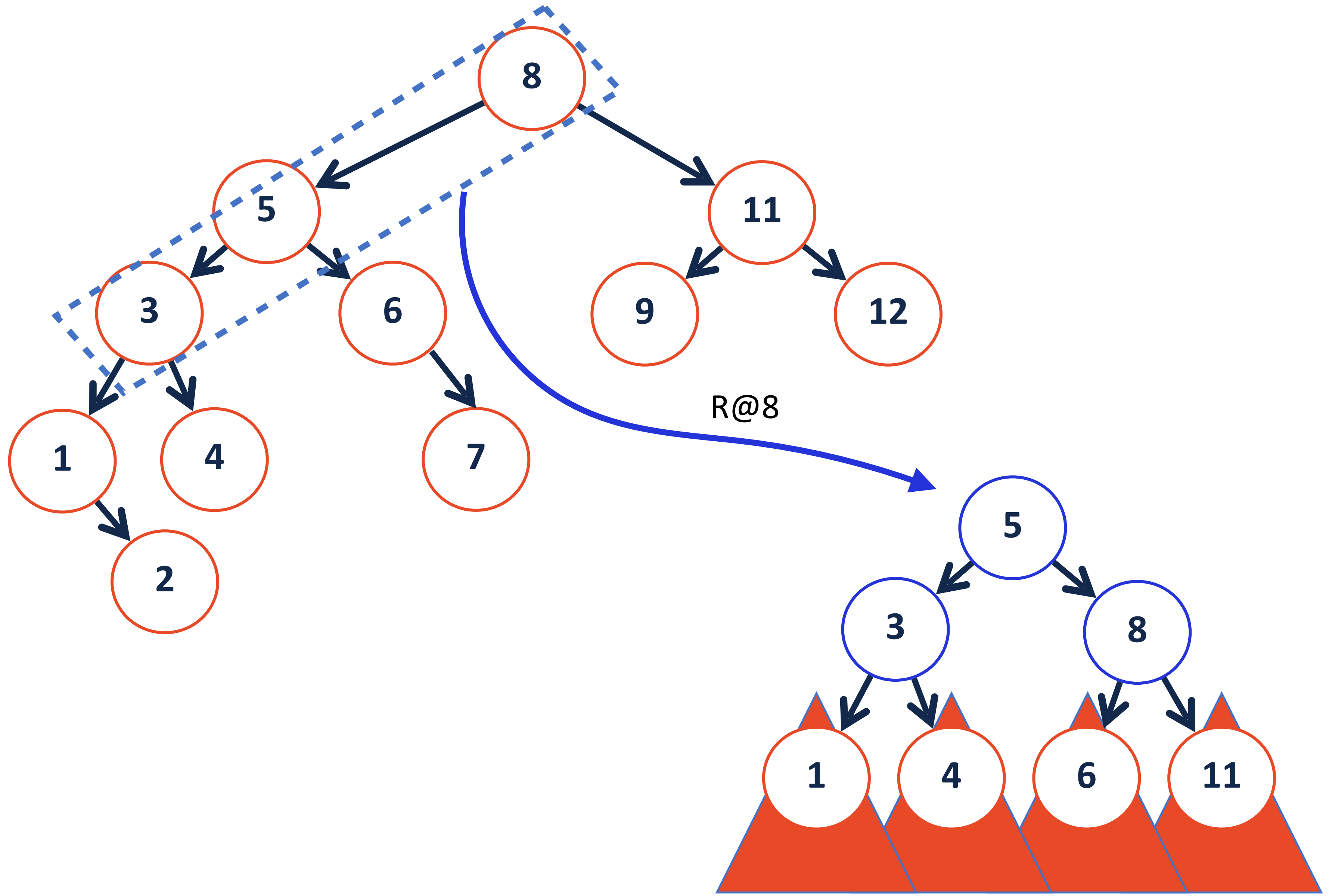
`_remove(10)`



$2 - 3 = -2$

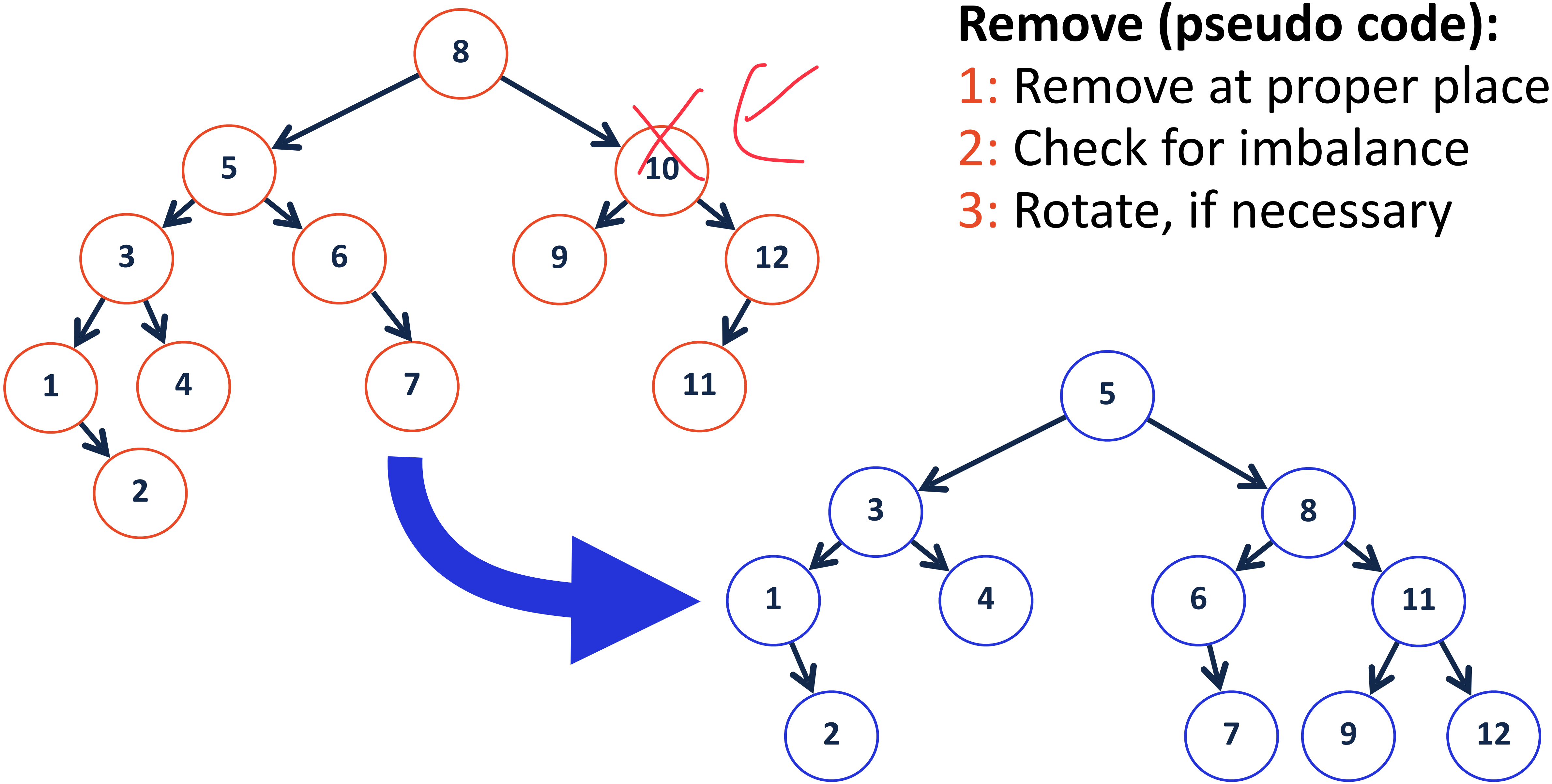
AVL Remove

`_remove(10)`



AVL Remove

`_remove(10)`



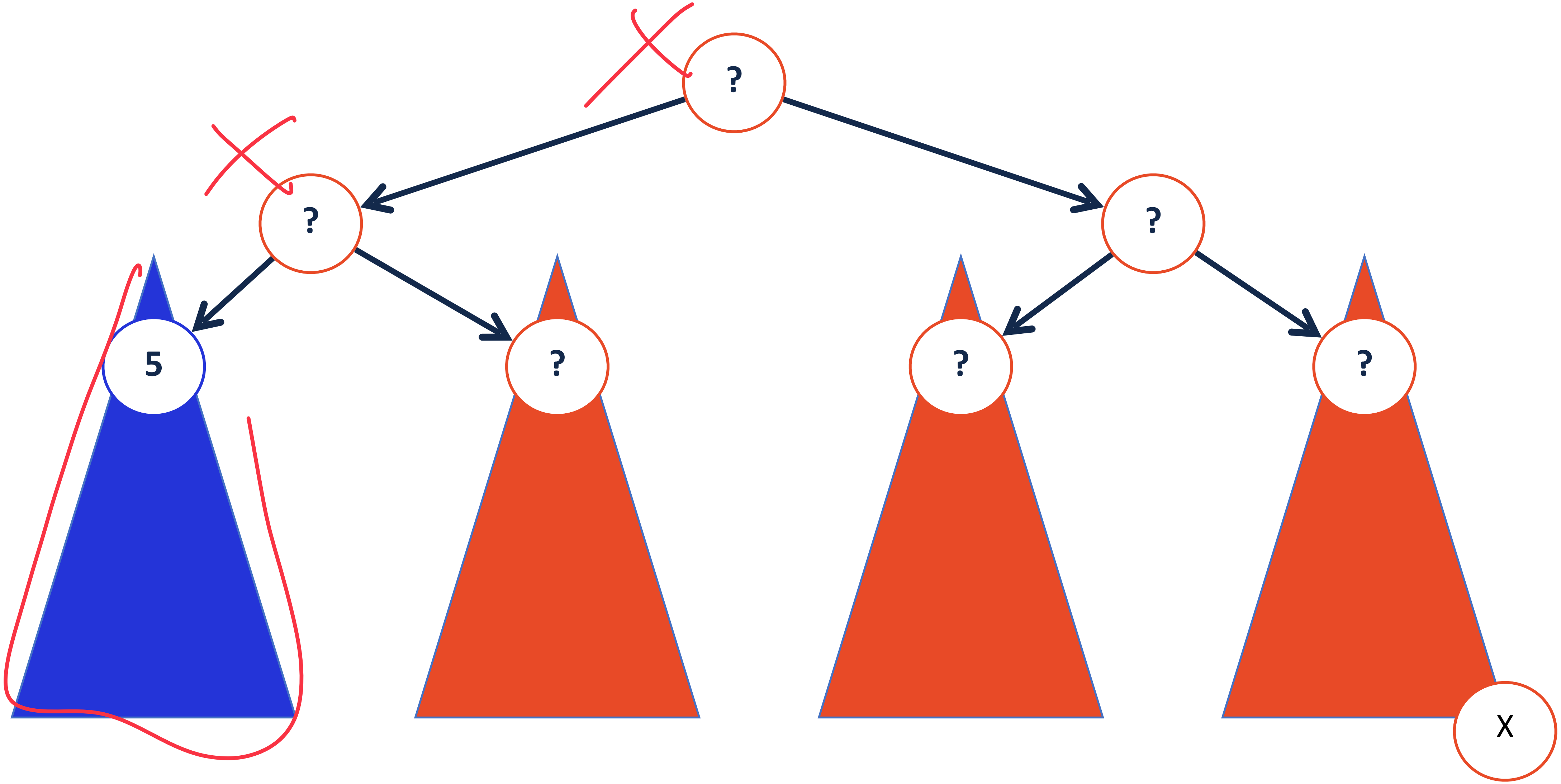
Remove (pseudo code):

1: Remove at proper place

2: Check for imbalance

3: Rotate, if necessary

AVL Remove



AVL Removal

Removal ***may*** reduce height by at most: 1

A rotation ***always*** reduces the height of the subtree by: 1

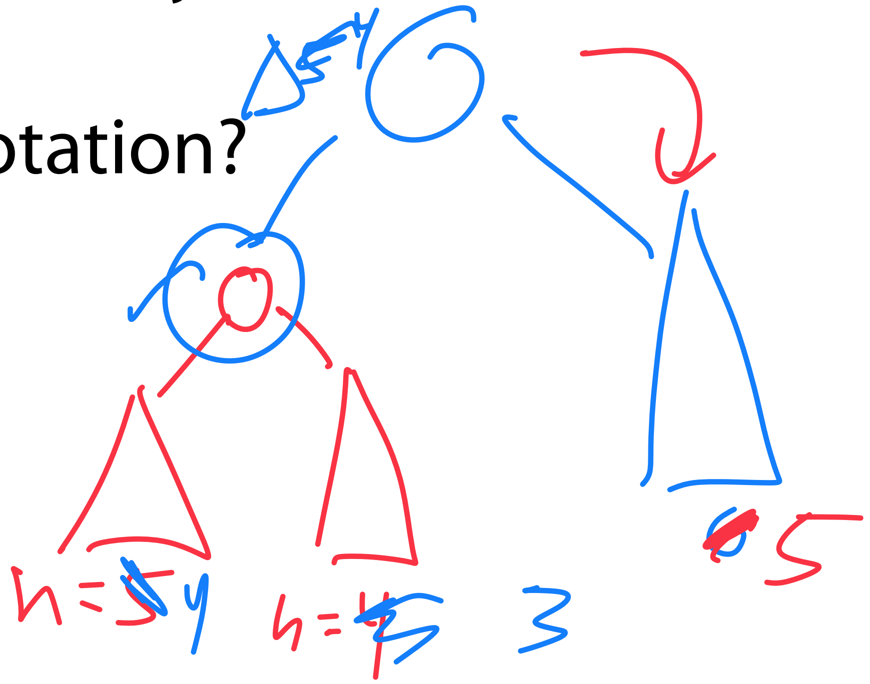
We might have to perform a rotation at every level of the tree!

What is the Big O of performing a single rotation?

↳ O(1)

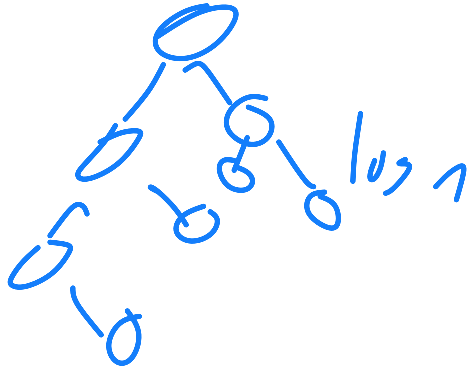
What is the Big O of removal?

↳ O(log n)



AVL Tree Analysis

$h = O(\log n)$



For AVL tree of height h, we know:

find runs in: $O(\log n)$.

insert runs in: $O(\log n)$.

1 rotation + 1 insert

remove runs in: $O(\log n)$.

h rotations + 1 remove

$O(\log n)$ $O(\log n)$

Claim: For a balanced binary search tree $h = \log(n)$.

Whats next?

Trees → Graph

A non-linear data structure defined recursively as a collection of nodes where each node contains a value and zero or more connected nodes.

graph

(In CS 277) a tree is also:

cycles

~~1) Acyclic — contains no cycles~~

~~2) Rooted — root node connected to all nodes~~