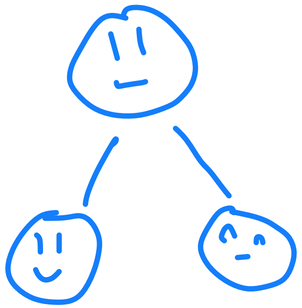
# Data Structures

# Balanced Binary Search Trees

CS 225
Brad Solomon & G Carl Evans

September 22, 2023



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

# Office Hour Space Changes

Construction in the basement space began Thursday

Currently unclear how much space will remain

For now OH will remain online in the basement…

But keep an eye on your email / Discord!

Exam ~~#~~2 – next week!

# Learning Objectives

Briefly review BST review

Discuss the big picture problem with BSTs

Introduce the self-balancing BST

```
template<typename K, typename V>

void _remove(TreeNode *& root, const K & key) {
```

1) find (6) ← *&

2) 3 cases:

0 - child remove

1 - child removal

↳ LL remove

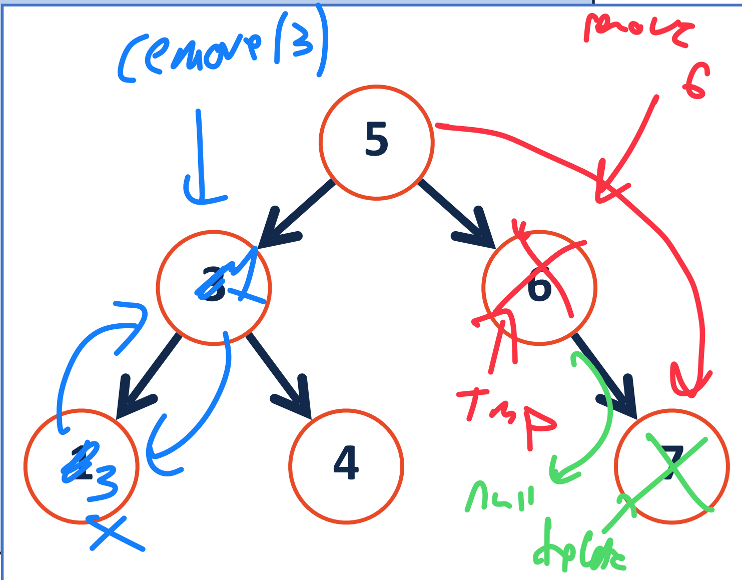
2 - child removal

↳ find IOP / IOS

↳ swap IOP w/ root

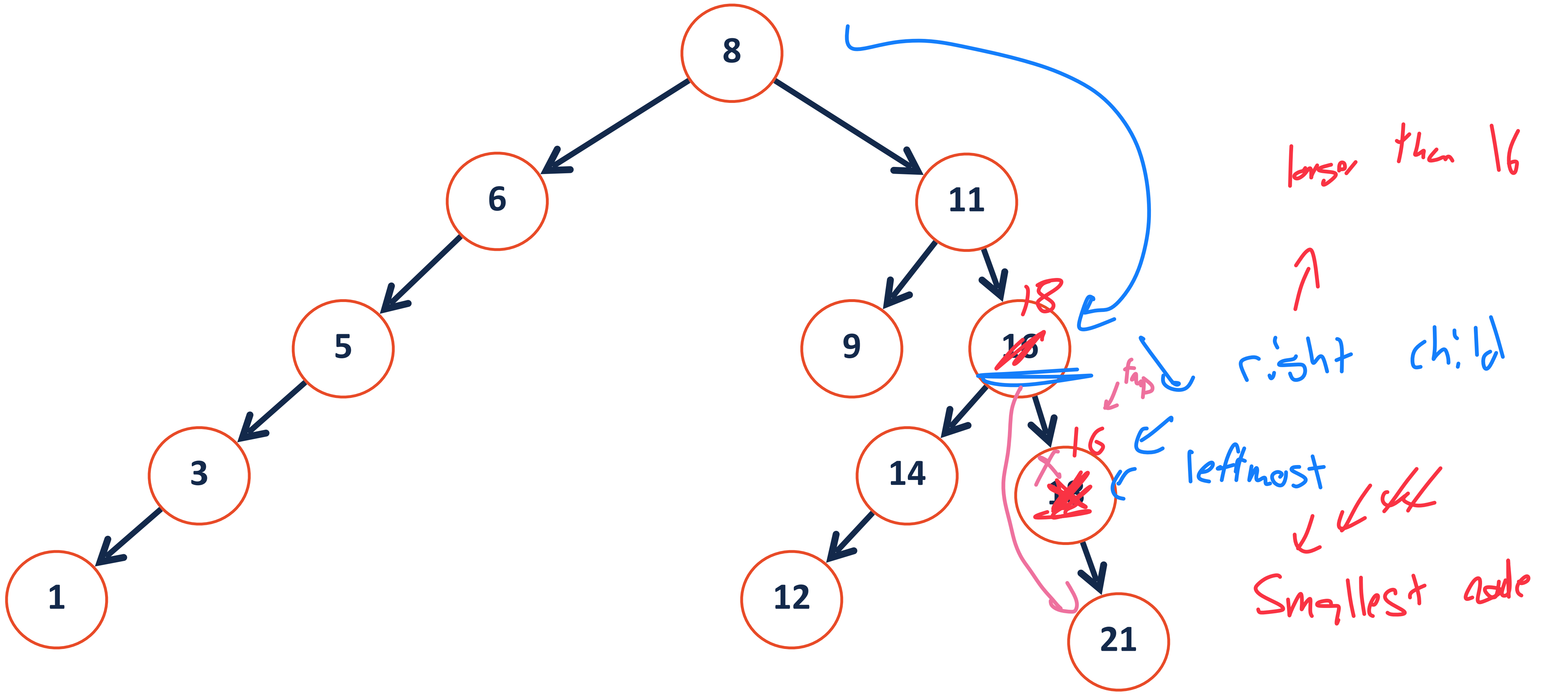
↳ remove (key)

```
}
```



# BST Remove

What will the tree structure look like if we remove node 16 using IOS?



# BST Analysis

Insert
Remove
find $\Big\}\ O(h)$

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?

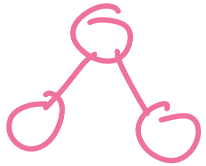
$f(h) \leq n \leq g(h)$

# BST Analysis

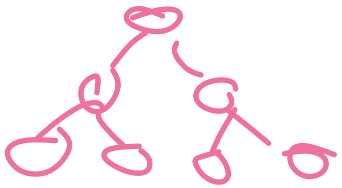
What is the **max** number of nodes in a tree of height $h$ ?

$h = 0$

$h = 1$



$h = 2$



$1 + 2 + 4 + \dots + 2^h = 2^{(h+1)} - 1$

$1$ $\quad$ $3$ $\quad$ $7$

$h \gtrsim O(\log(n))$

$n \leq 2^{h+1} - 1$

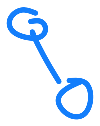
# BST Analysis

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

$h = 0$

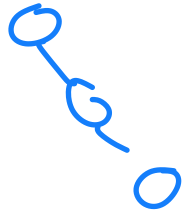
$1$

$h = 1$



$2$

$h = 2$



$3$

$$n \geq h+1$$

$$O(n)$$

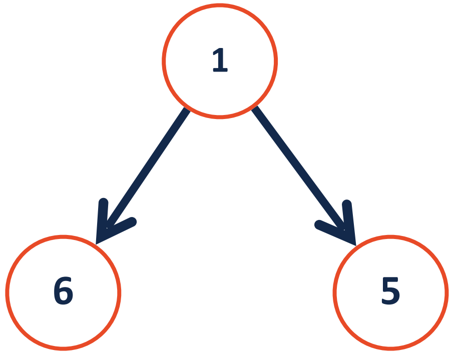
# BST Analysis

A BST of $n$ nodes has a height between:

all ops are $O(h)$

$h$

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

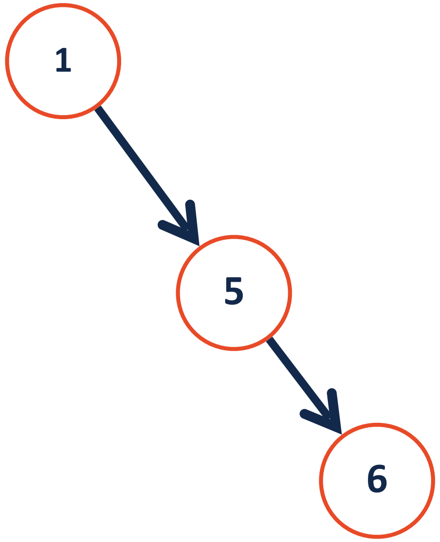



random BST

$h$

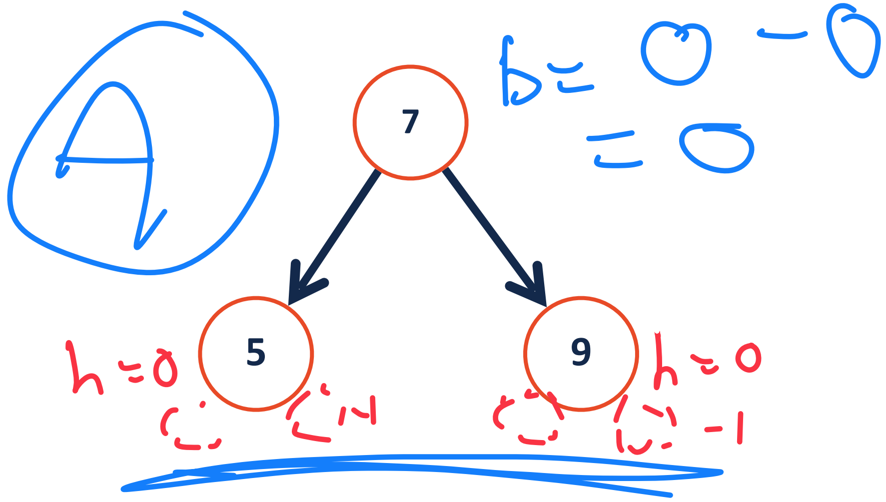
**Upper-bound:** $O(n)$

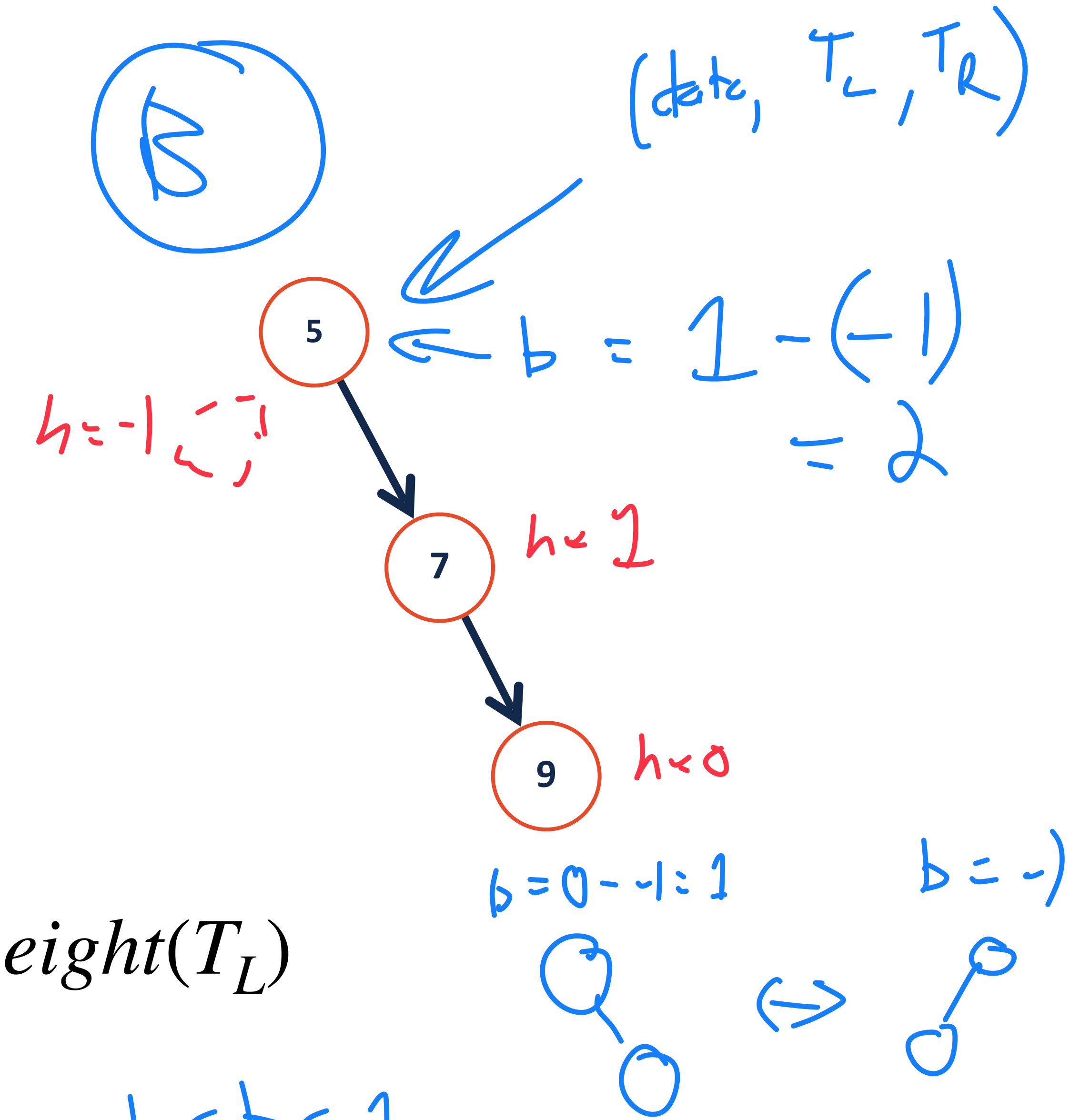
bad



# Height-Balanced Tree

What tree is better?




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

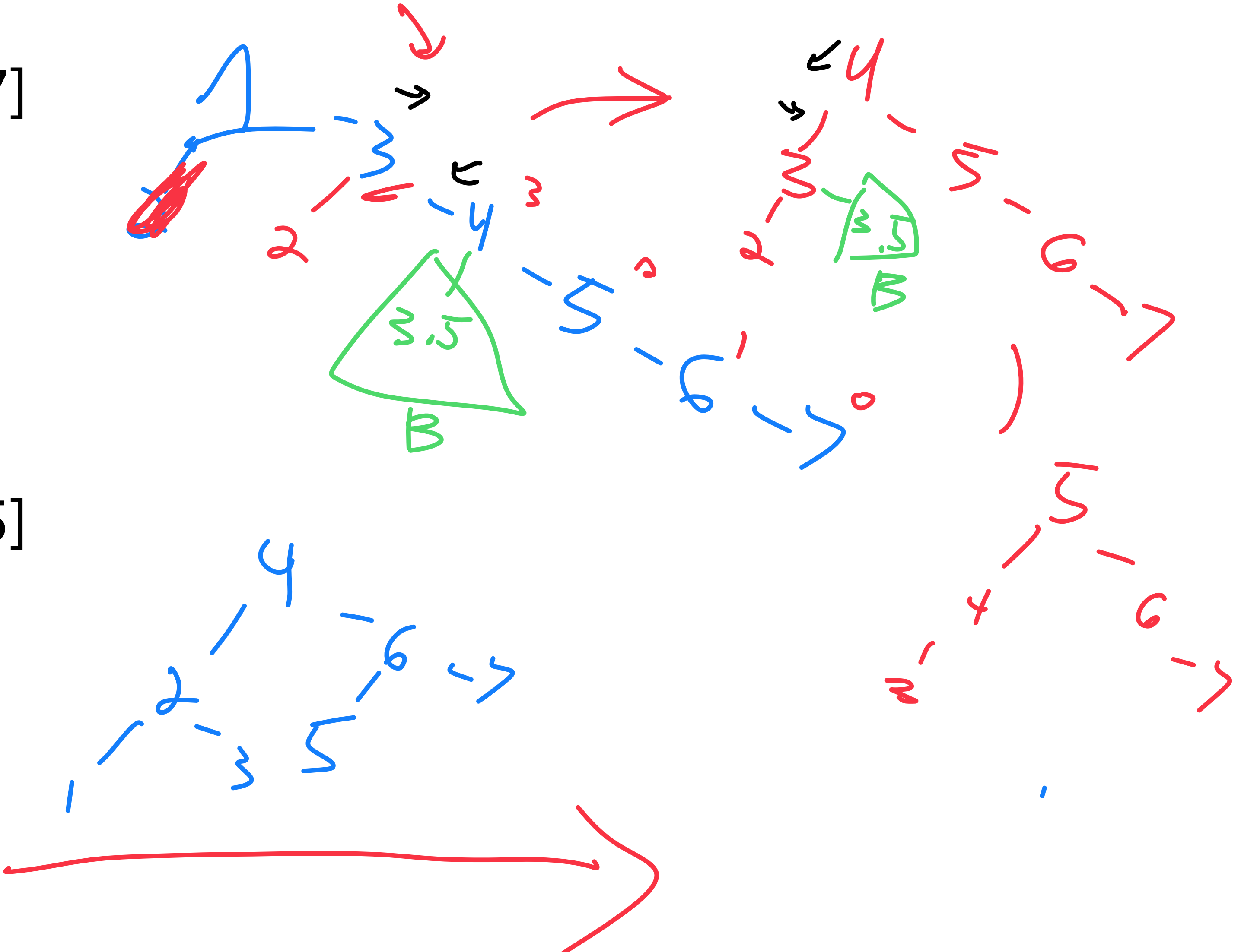
A tree is "balanced" if: $|b| \leq 1$   $-1 \leq b \leq 1$

# Option A: 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]

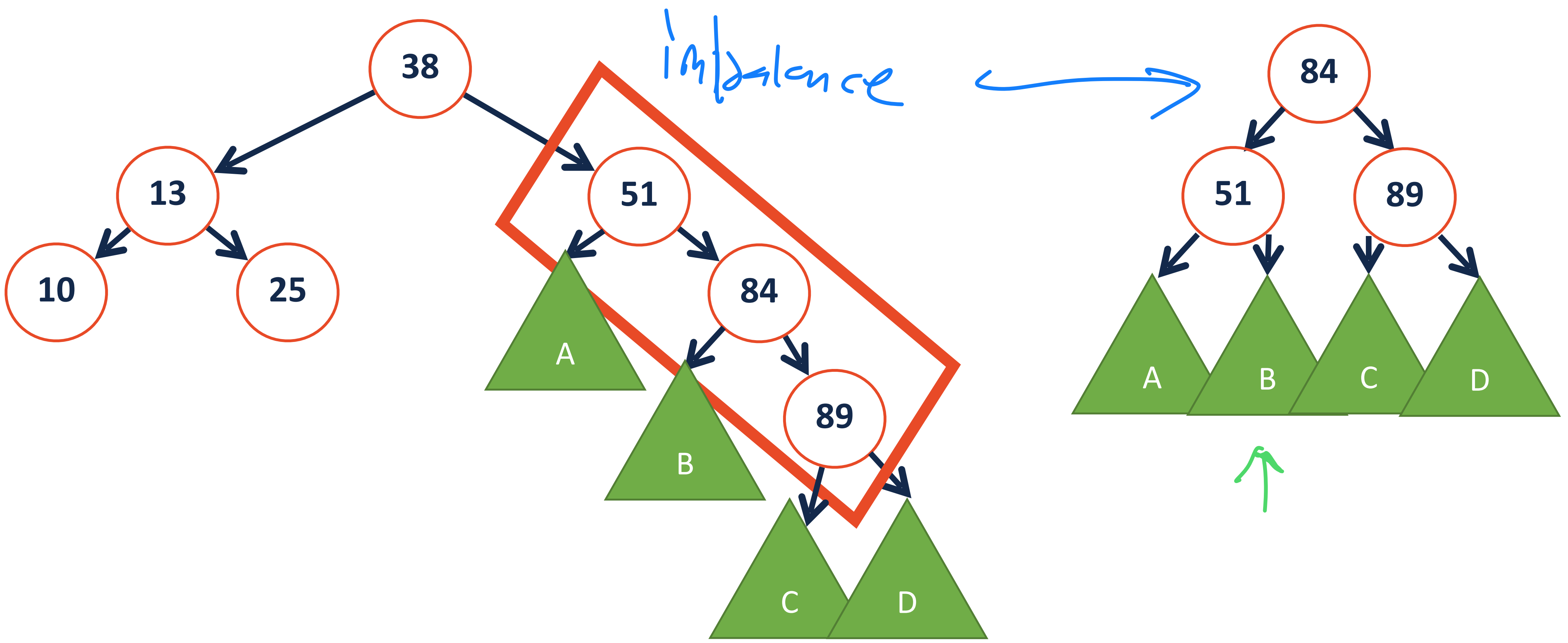
D= 3 - 0



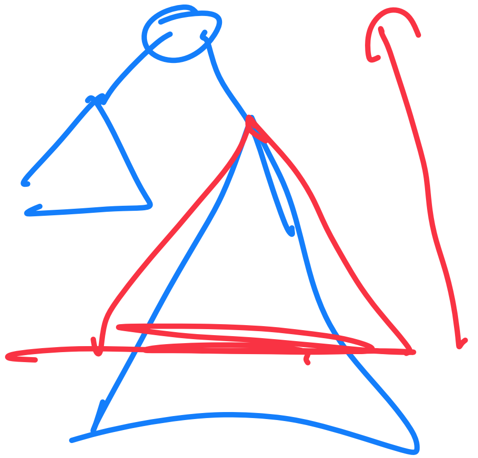
**Insert Order:** [4, 2, 3, 6, 7, 1, 5]

# AVL-Tree: A self-balancing binary search tree

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



# BST Rotations (The AVL Tree)



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

These rotations:

1. modify arrangement of nodes while preserving BST property

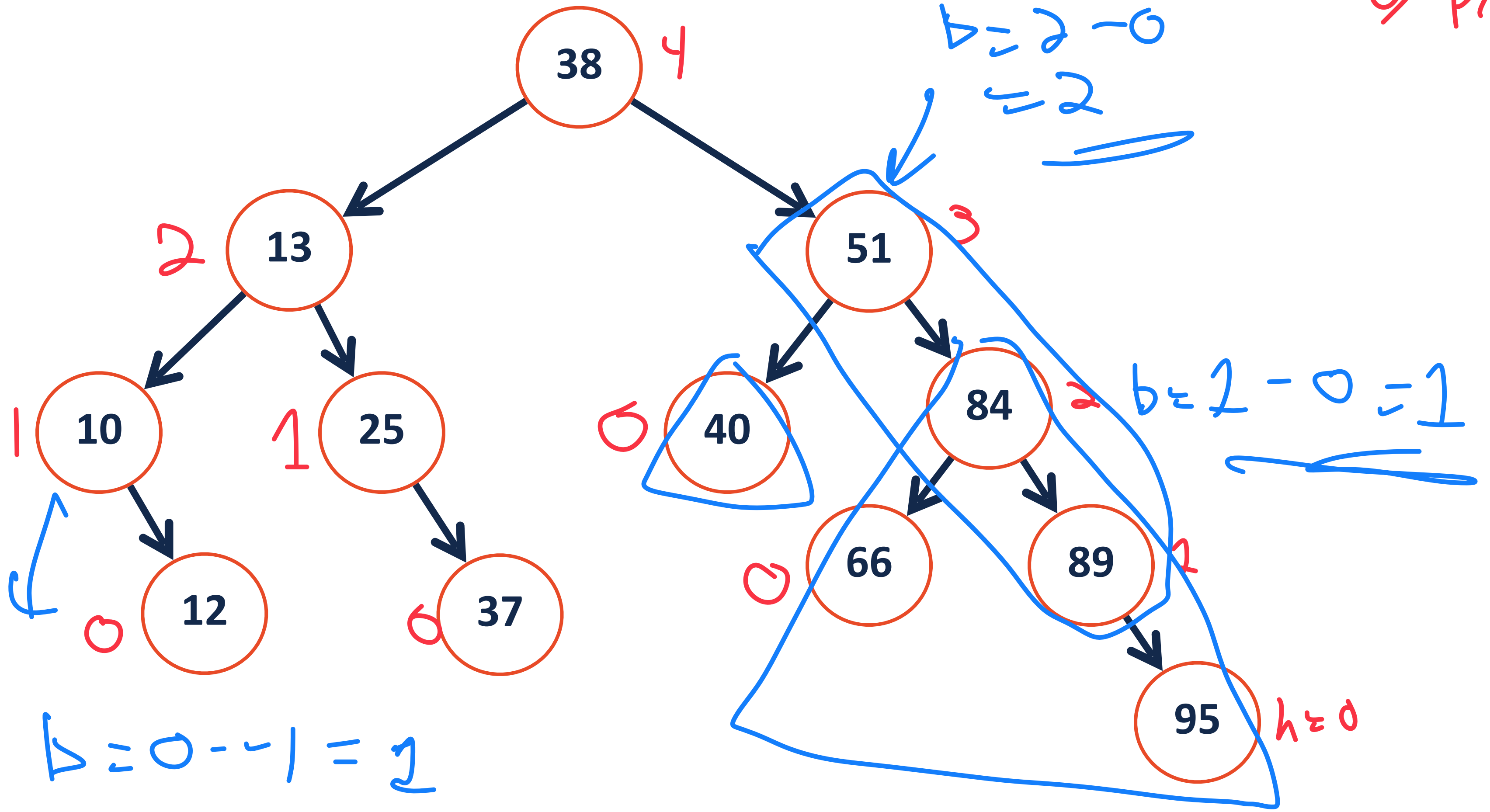
2. Reduce my tree height by one — yes if you do the ~~right~~ rotation

(CORRECT)

# BST Rotations (The AVL Tree)

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

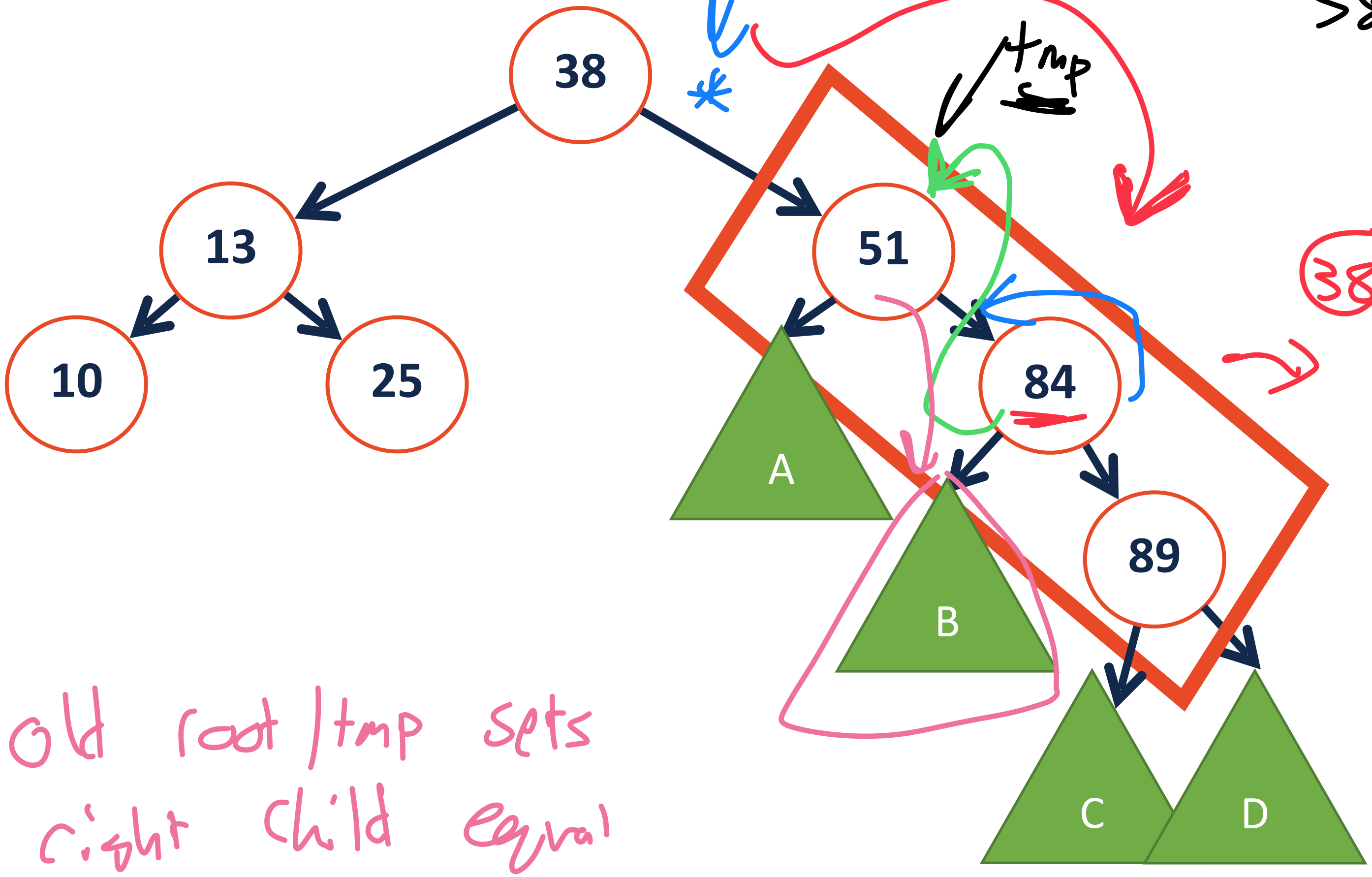
1) Find imbalance
↳ prefer imbalance at root*



# Left Rotation

imbalance @ 51

&

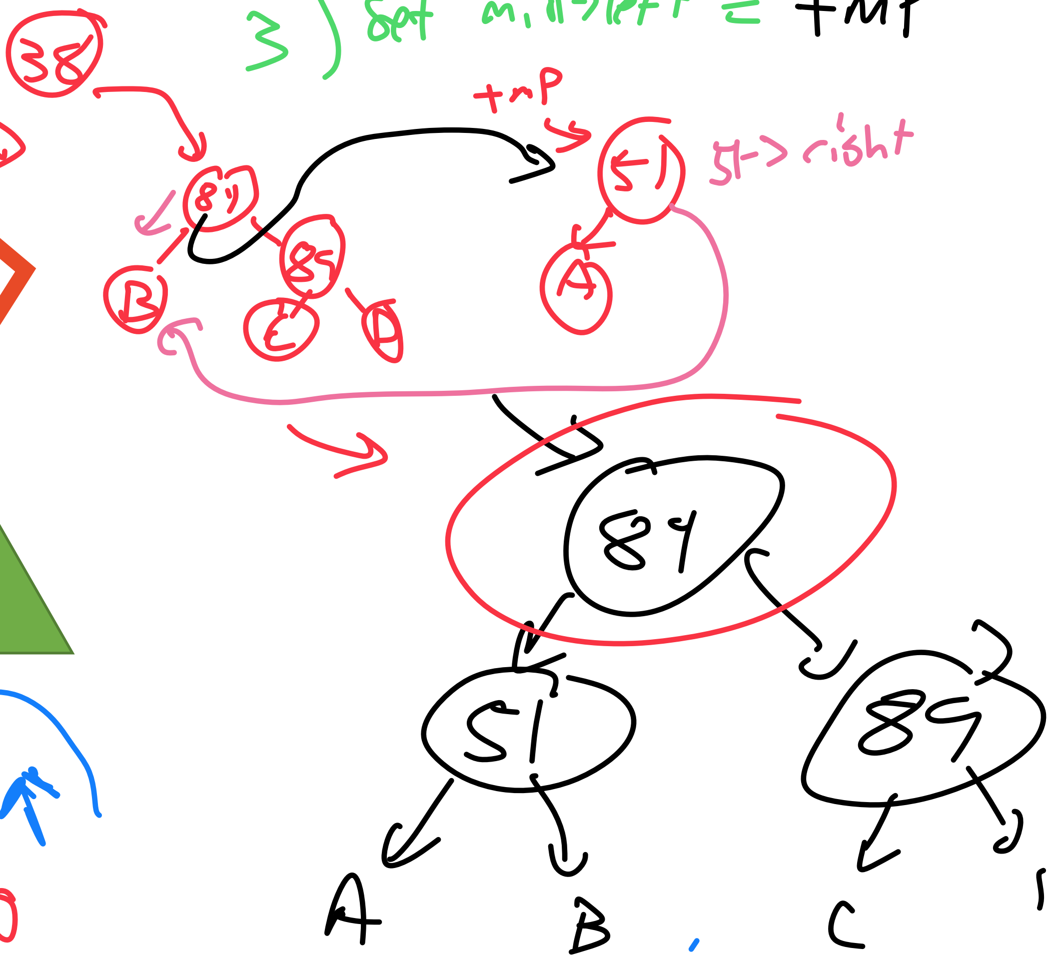



0) tmp = root;

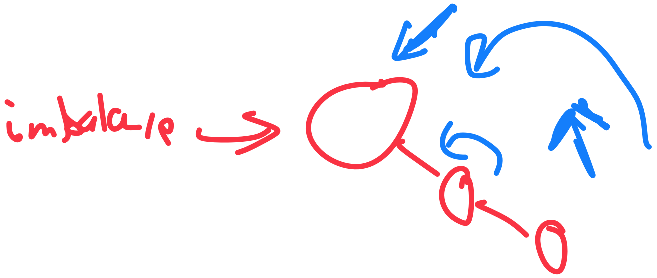
38->right 1) set root to mid node (84)

2) 51->right = 84->left

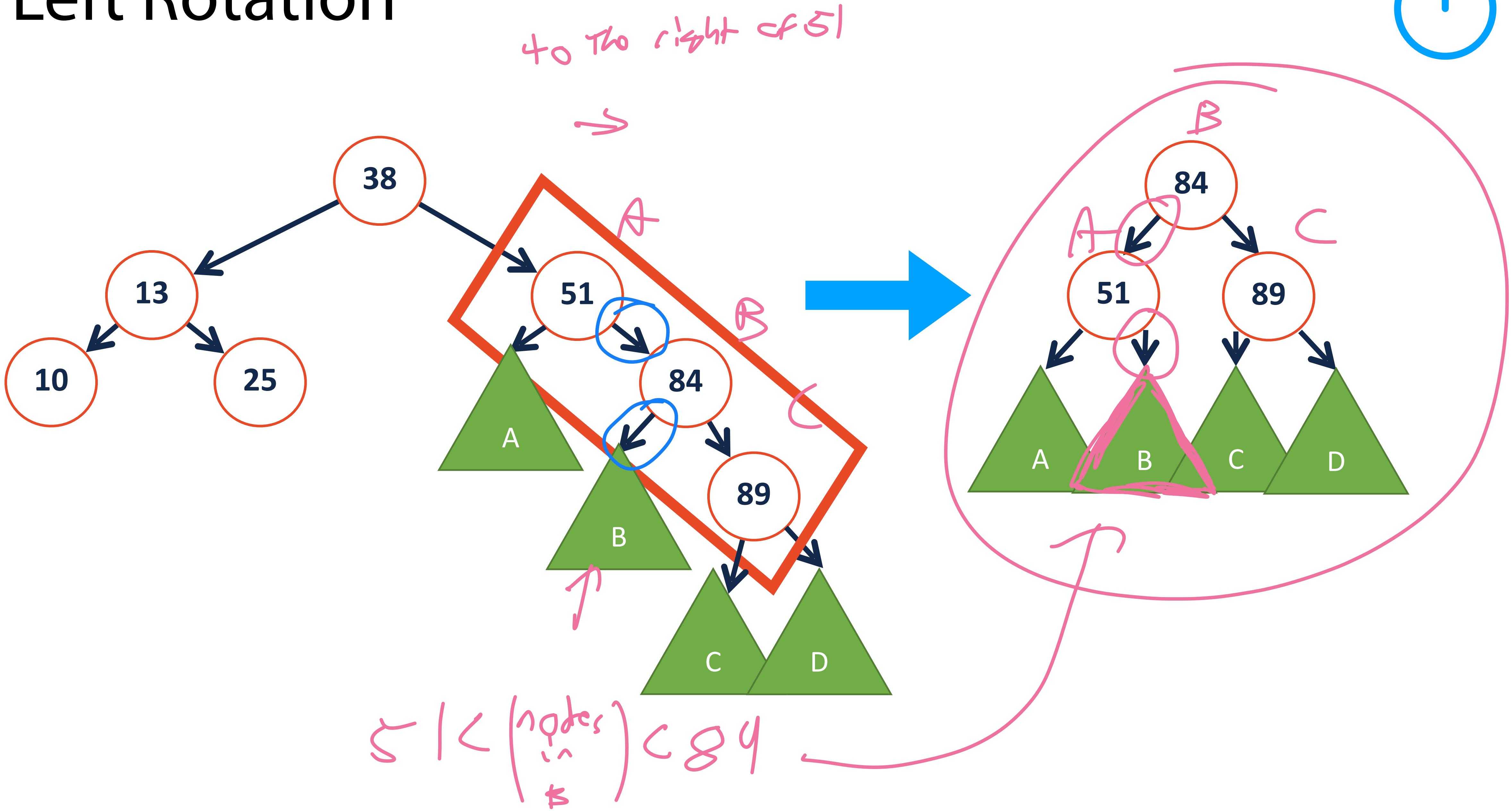
3) set mid->left = tmp



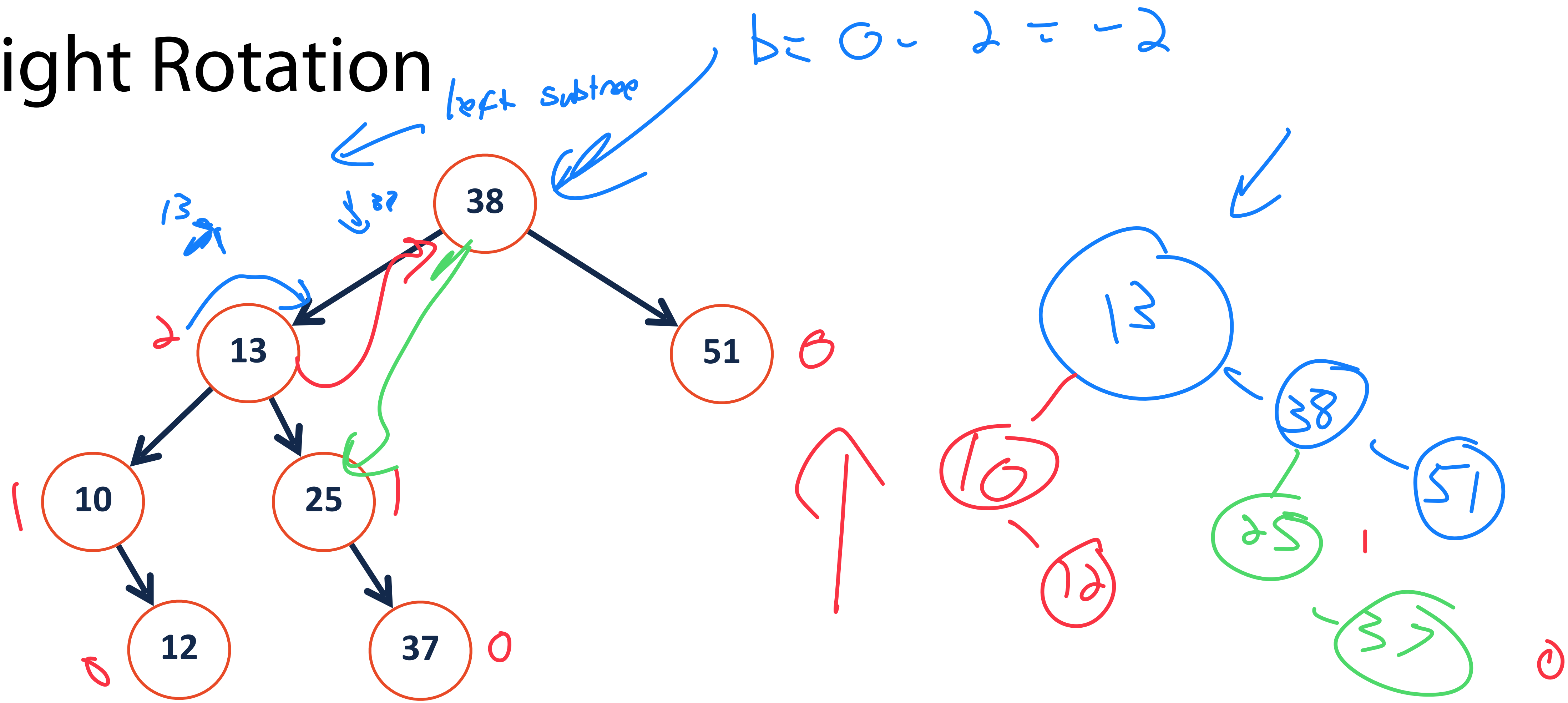

old root/tmp sets right child equal to mid->left



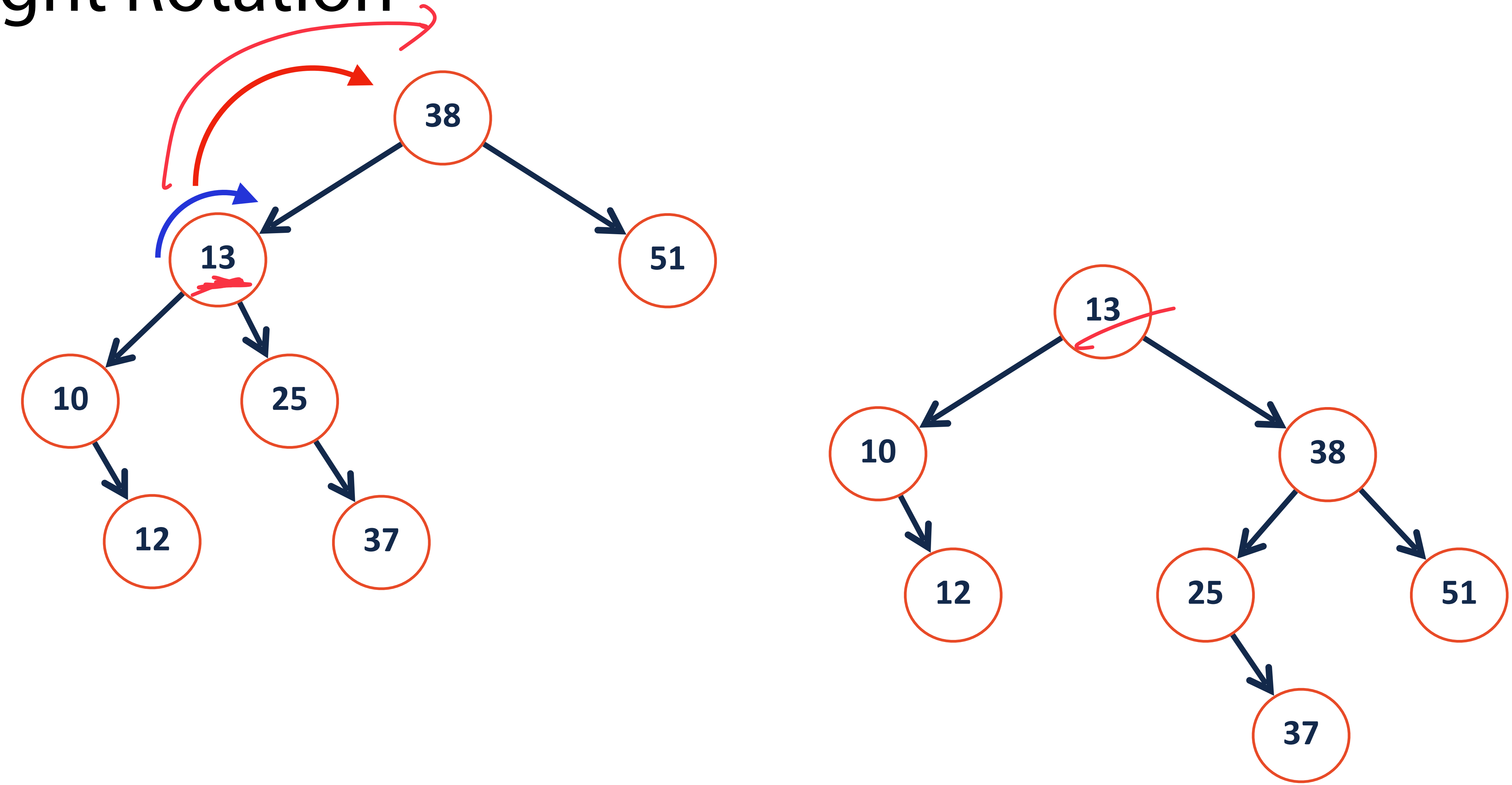
# Left Rotation



# Right Rotation



# Right Rotation

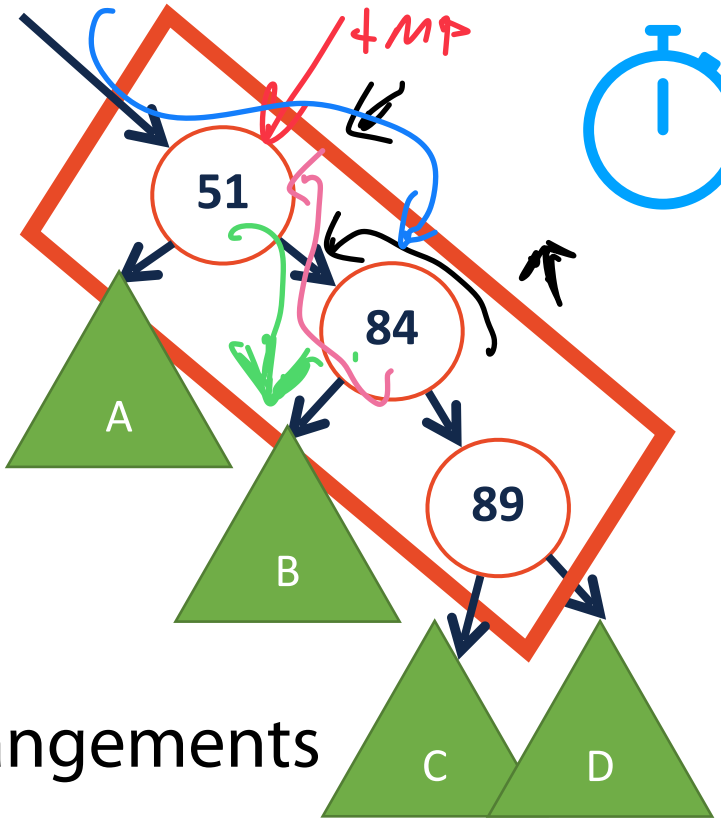


# Coding AVL Rotations

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*?

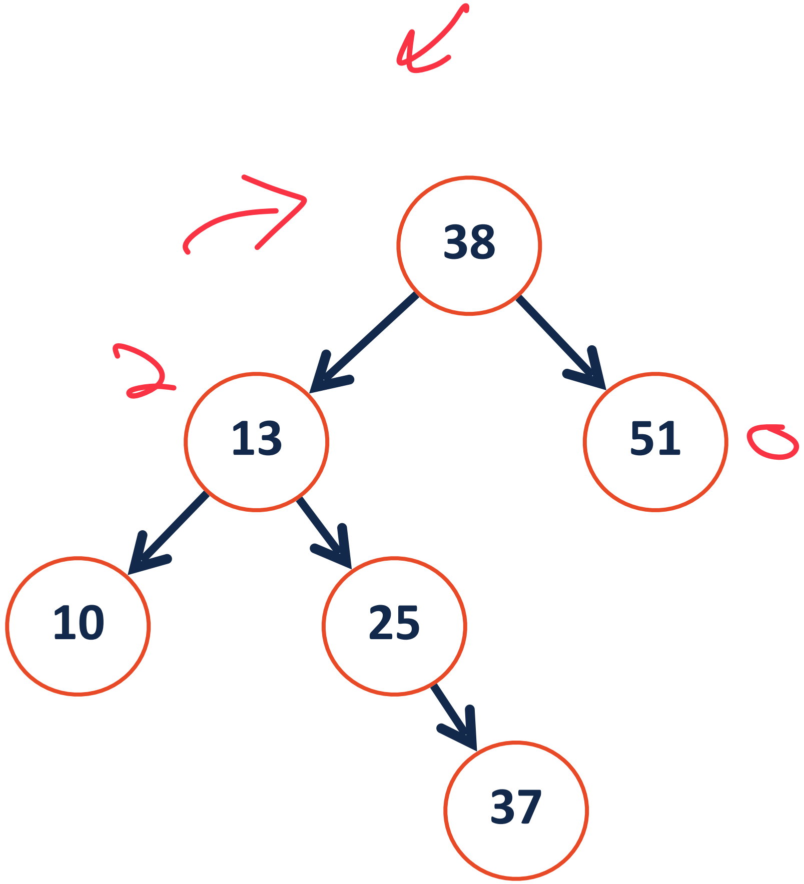
Make a tmp @ current root

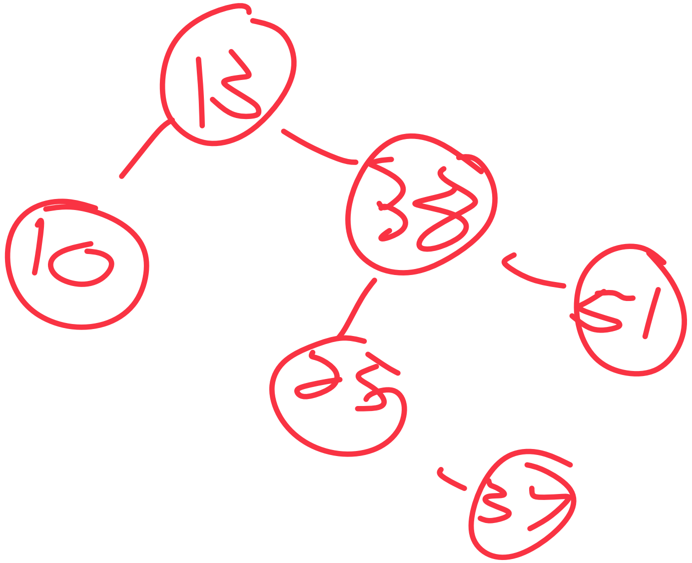
Replace current (root) node with it's right child. (Blue)

Set the right child's left child to be the current node's right (Green)

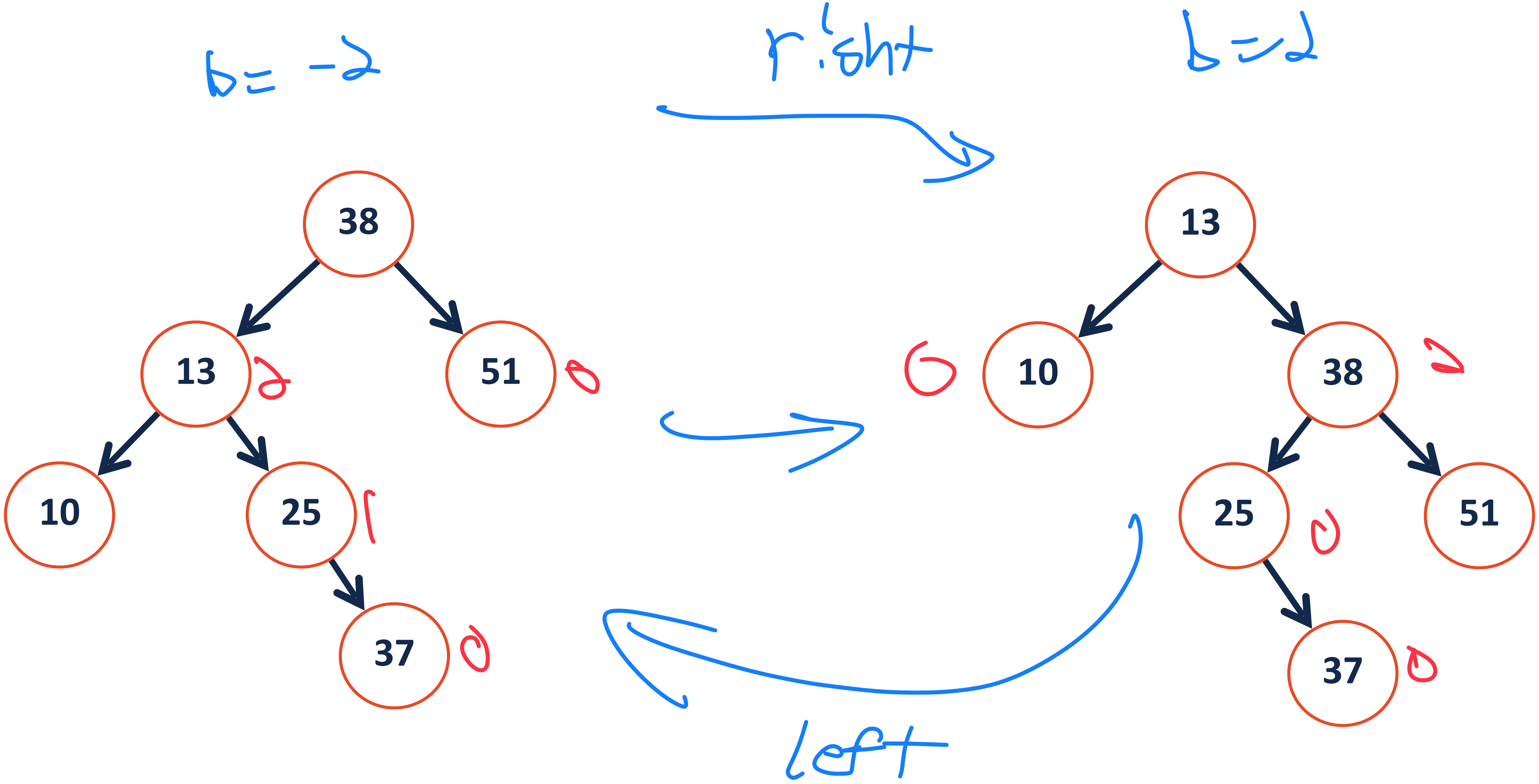
Make the current node the right child's left child (Pink)

# AVL Rotation Practice



# AVL Rotation Practice




Somethings not quite right…