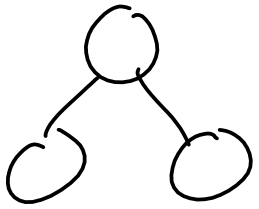
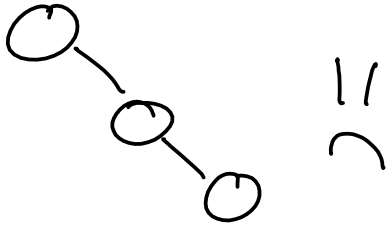
AVL Tree: BST w/ a balancing property

"balanced": $\forall$ nodes $\in$ tree. height(node.left) height(node.right) differ by at most 1



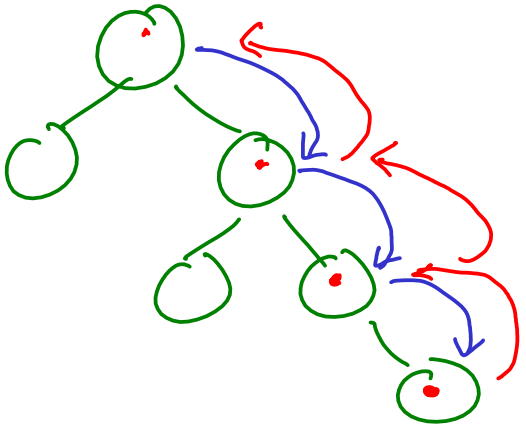
"full" height is $O(\log n)$



!! height is $O(n)$

Balancing so that height is $O(\log n)$

Method insert(K key, V value):



If this->key < key:

If this->right == NULL:

this->right = new Node(key, value)

Else:

this->right -> insert(key, value)

need constant time rebalance

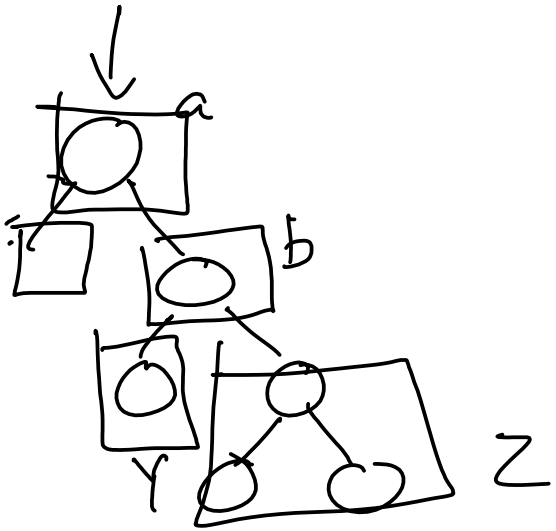
rebalance()

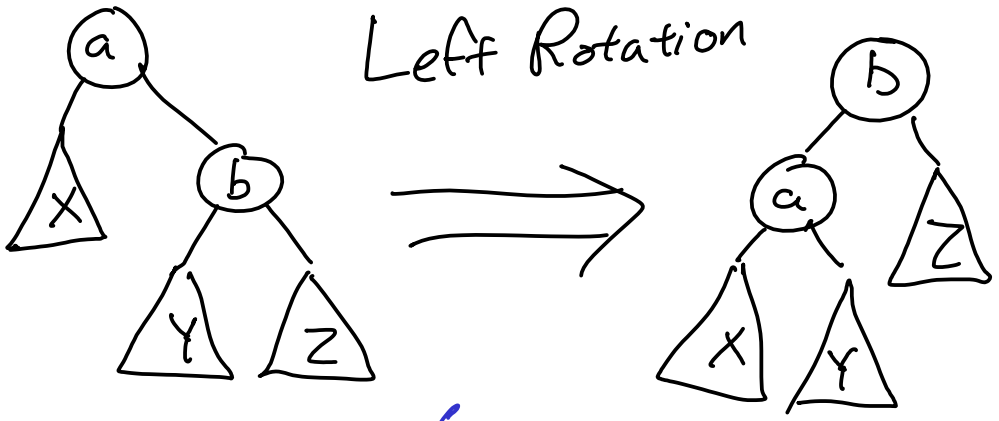
recompute_height()

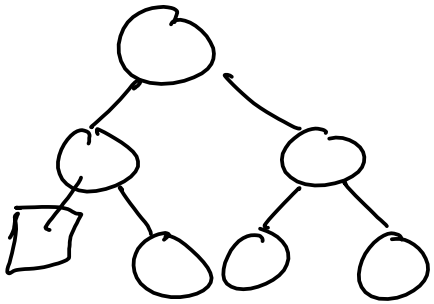
Method rebalance():

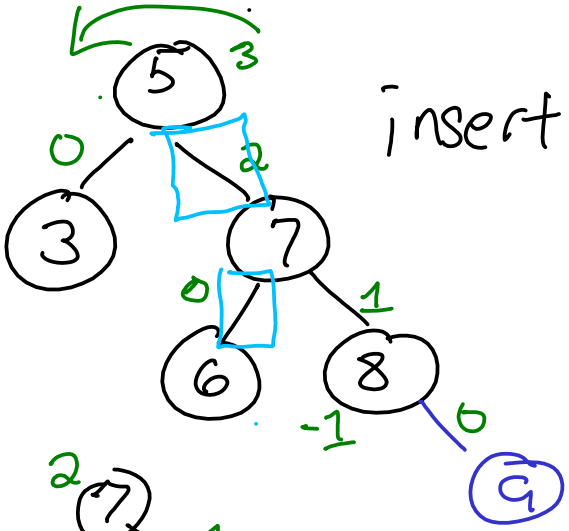
If left subtree height + 1 < right subtree height:

Rotate Left!

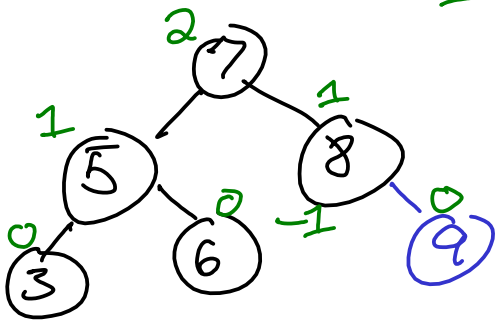


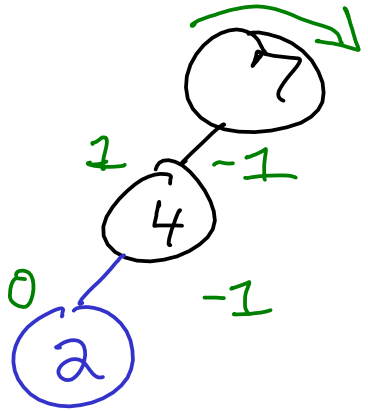




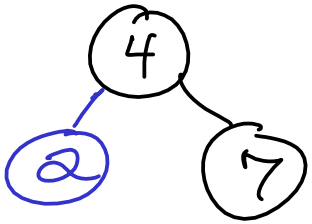


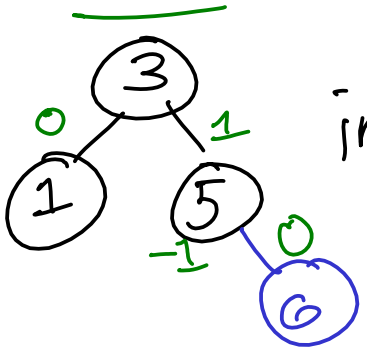
insert 9



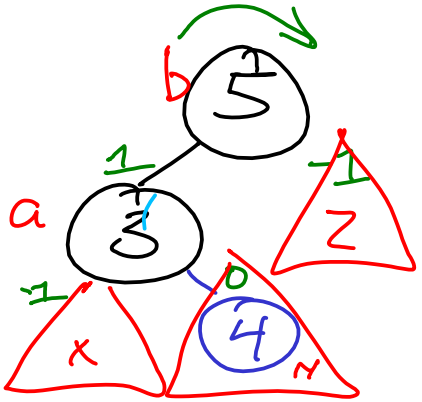


insert 2

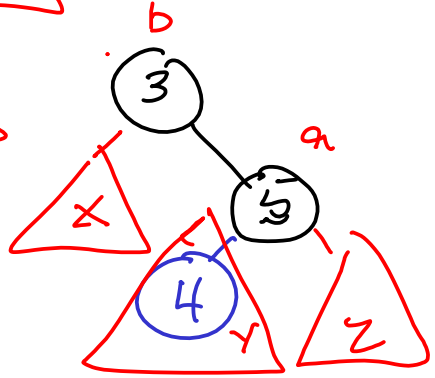


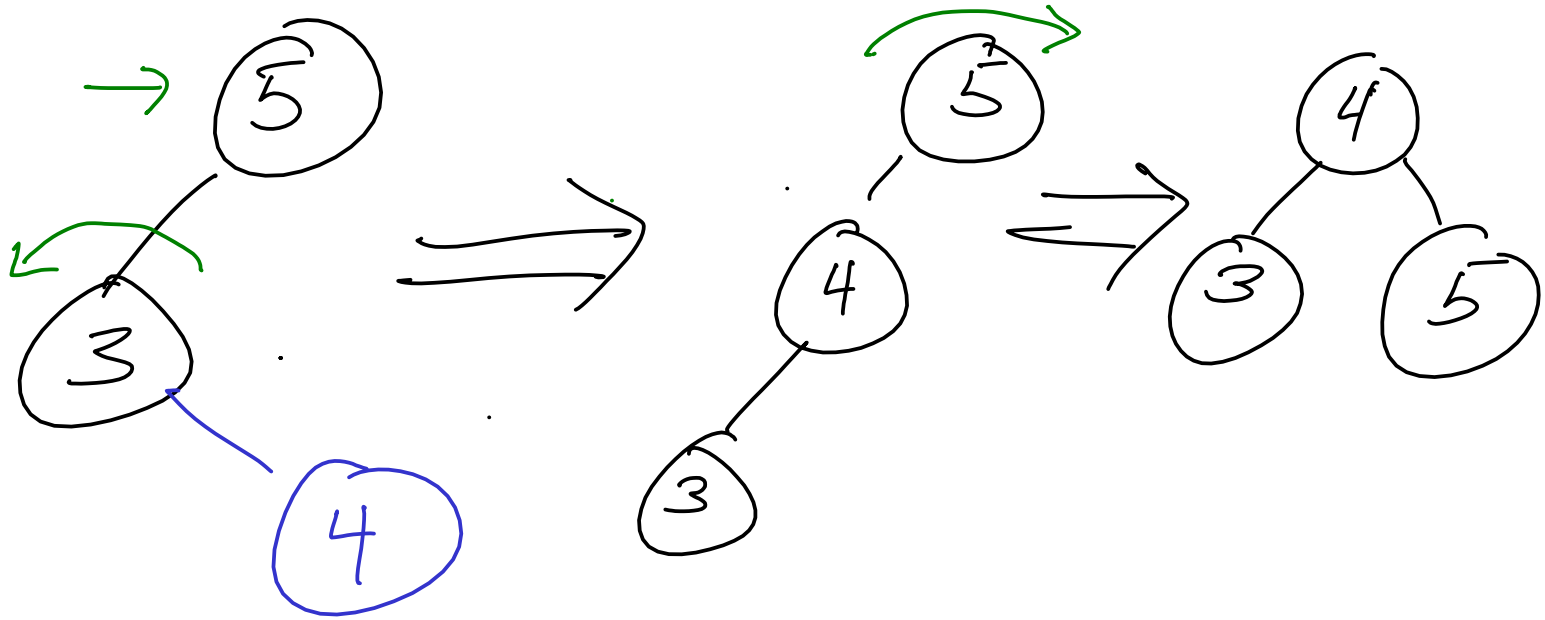


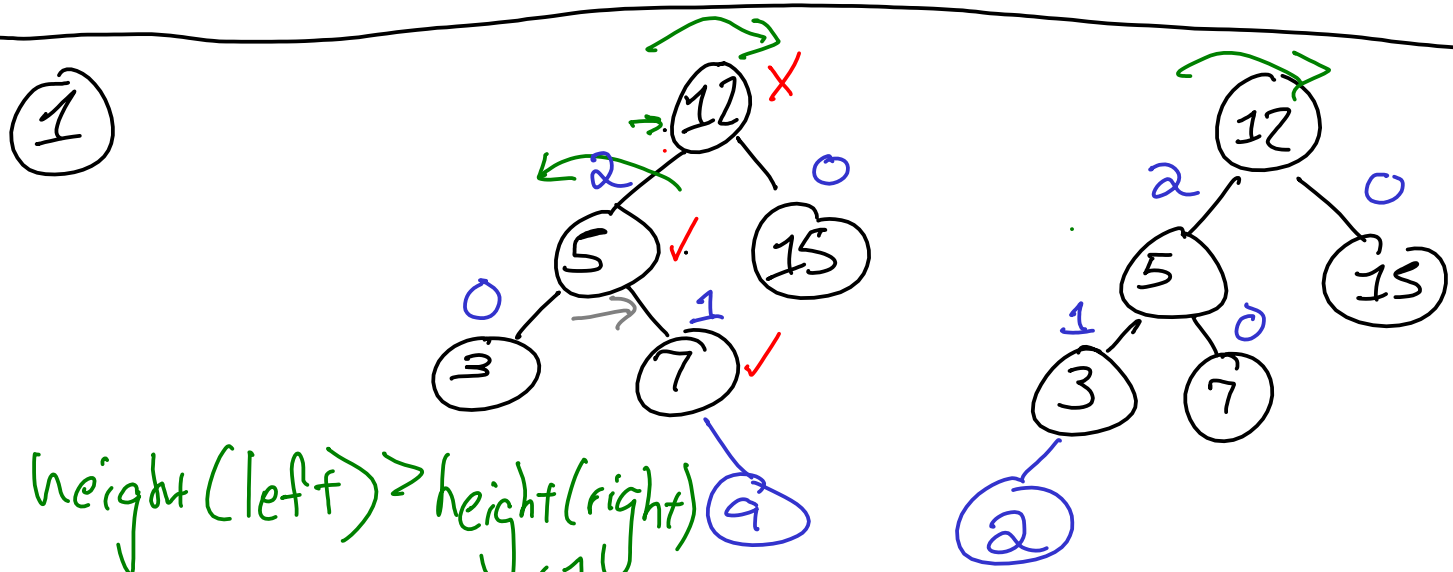
insert 6



insert 4



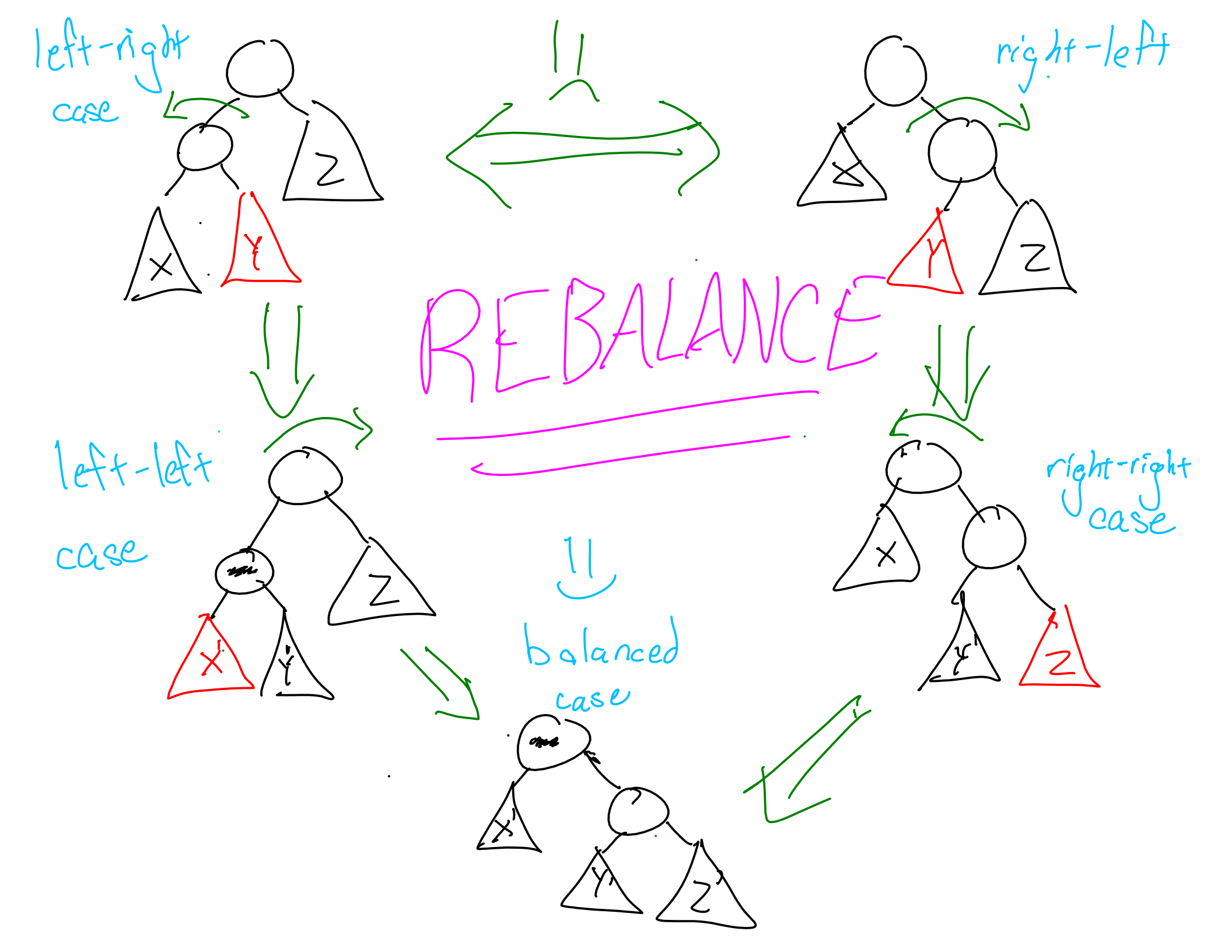


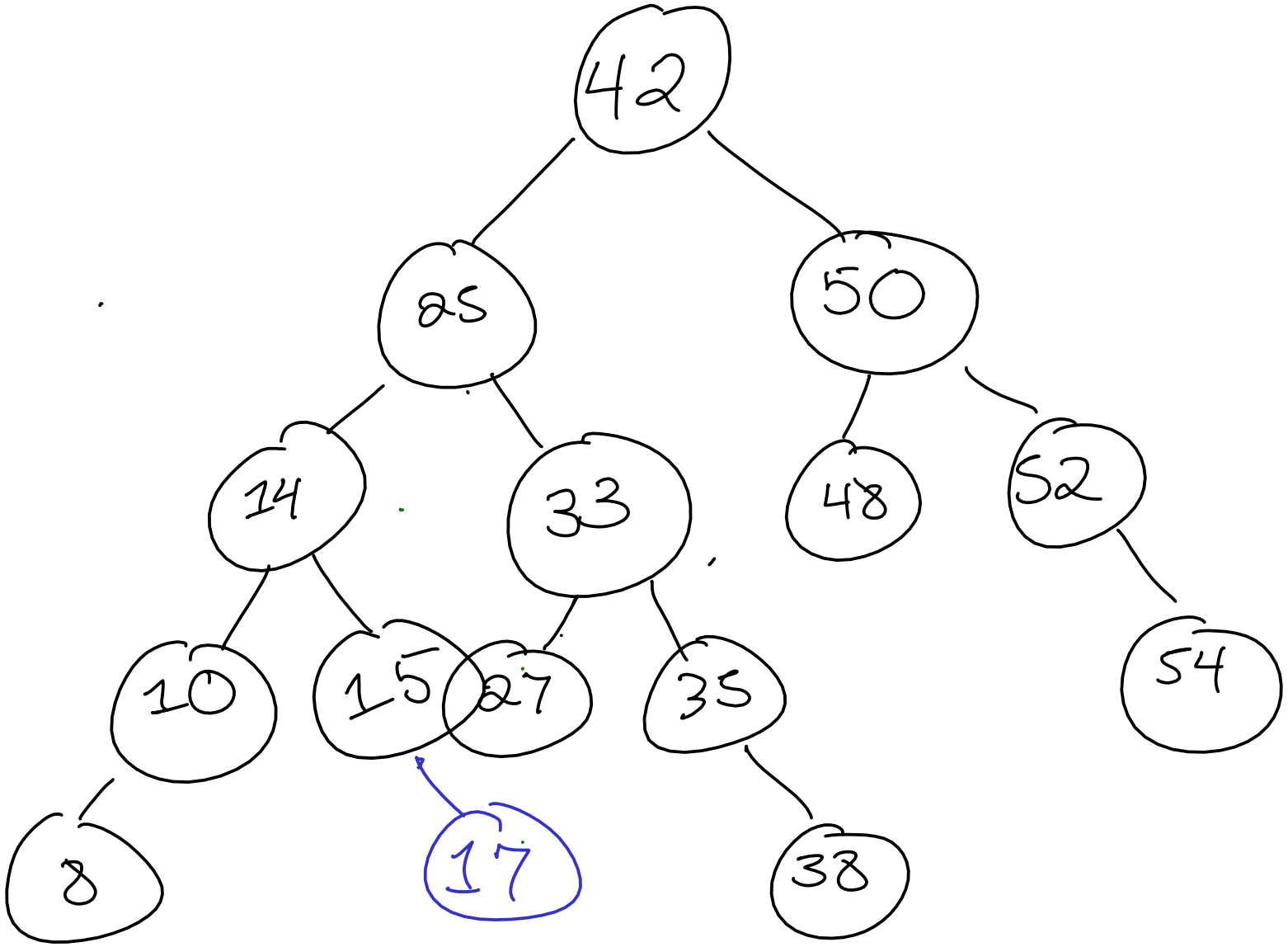


If height(left) > height(right) +1

If height(left.left) < height(left.right)

this->left -> rotate_left()


left-right
case
right-left
case
REBALANCE
left-left
case
right-right
case
balanced
case
X
Y
Z


42
25
50
14
33
48
52
10
15
27
35
54
8
17
38