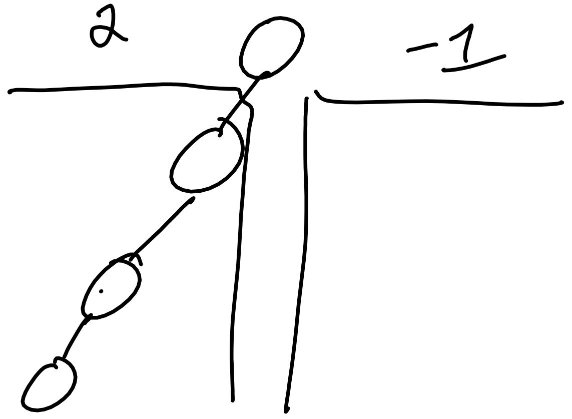
BST - binary tree w/ invariant:

all data keys in L subtree is <

all data in R subtree is >

AVL tree - BST w/ invariant

height of child subtrees differs by at most 1

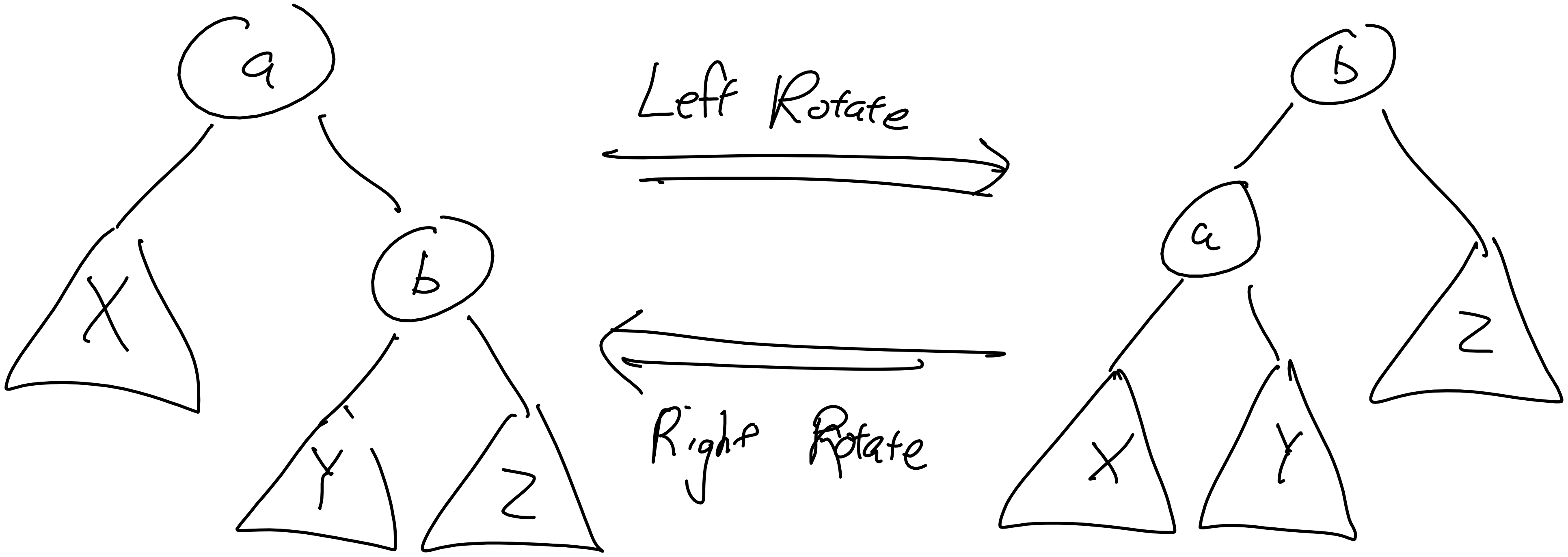


$O(\text{height})$ $\begin{cases} \text{get} \\ \text{insert} \\ \text{update} \\ \text{remove} \end{cases}$

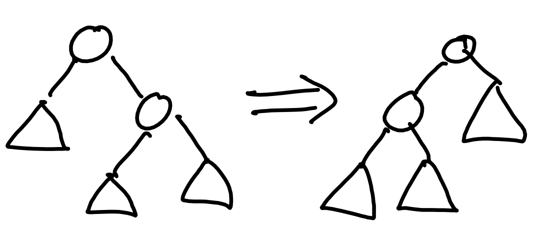
Rotation

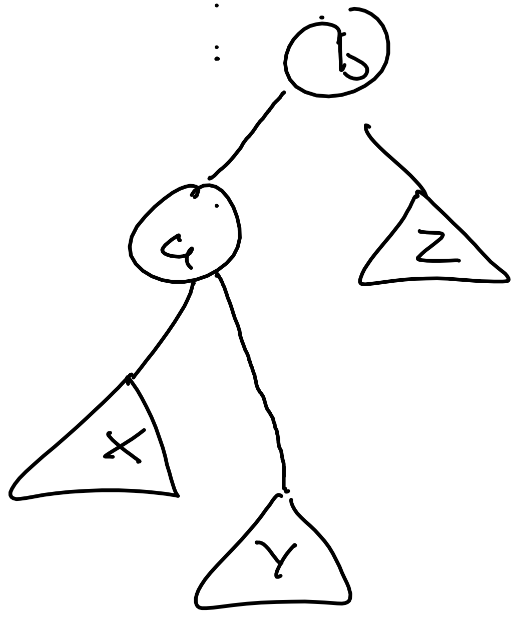
◯ node

△ subtree (possibly empty)

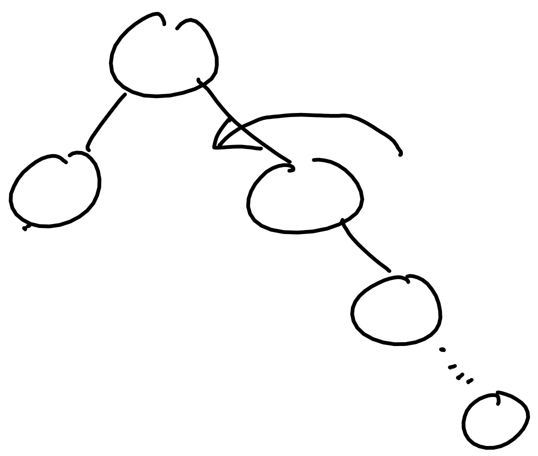


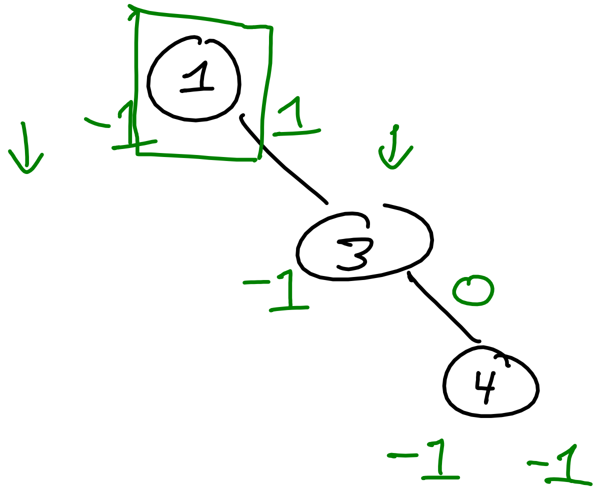
```
Function rotateLeft(Node node):
    If node has no right child:
        ~~ (raise exception)
    a ← node
    b ← node → right
    X ← node → left
    Y ← b → left
    Z ← b → right
    a → right ← Y
    b → left ← a
    Return b
```

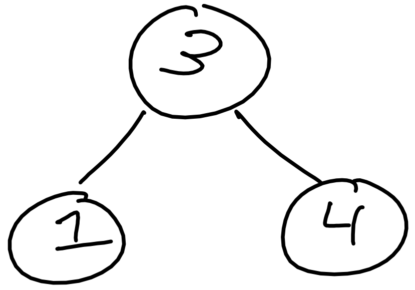




Insert

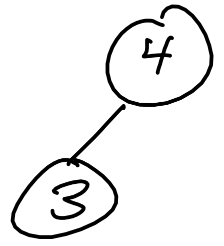




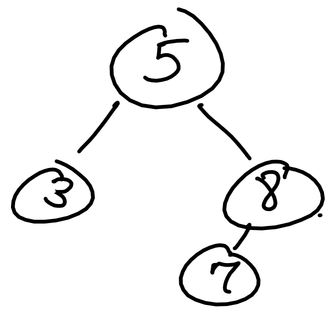


```
Function insert(Node root, K key, V value):
    If root is empty:
        Return new Node(key, value)
    Else If root->key > key:
        root->left ← insert(root->left, key, value)
        root ← rebalance(root)
        recalcHeight(root).
        Return root
    Else .....
```

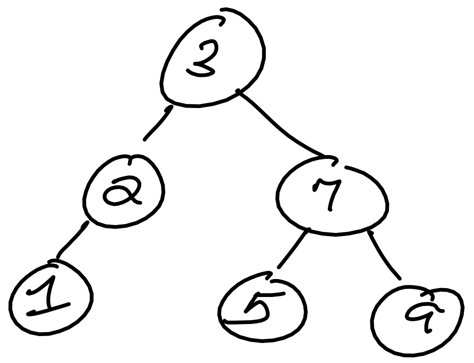
Rebalance Exercise



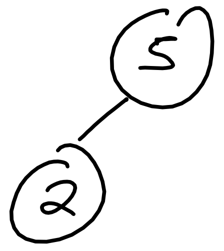
insert 2



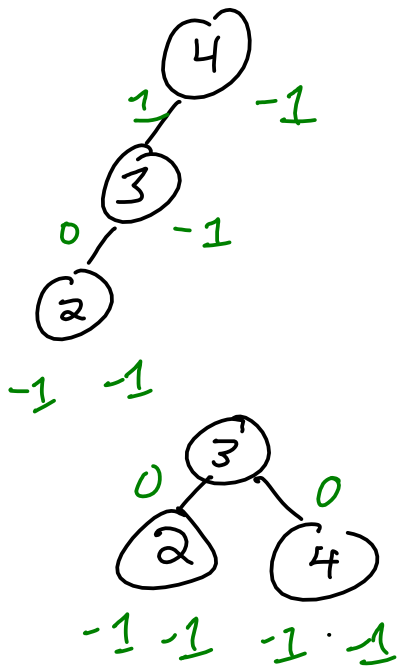
insert 6

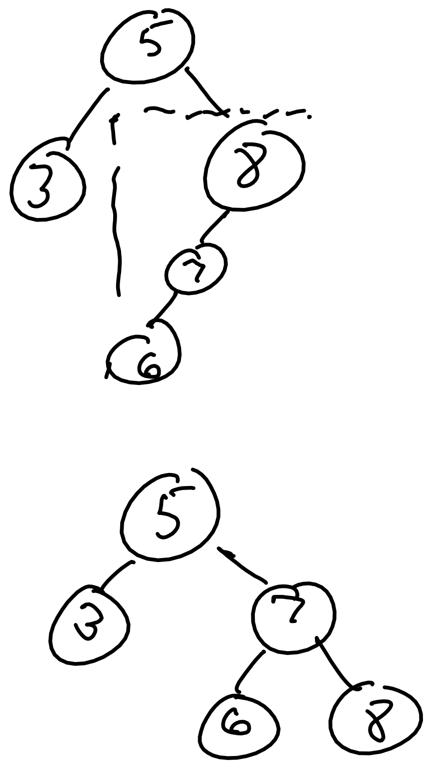


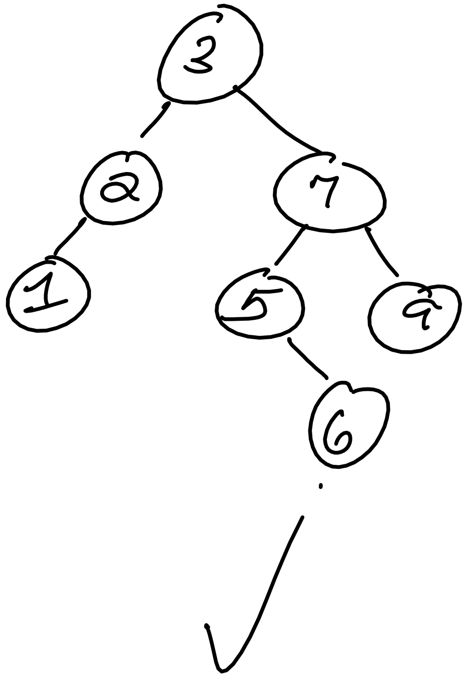
insert 6

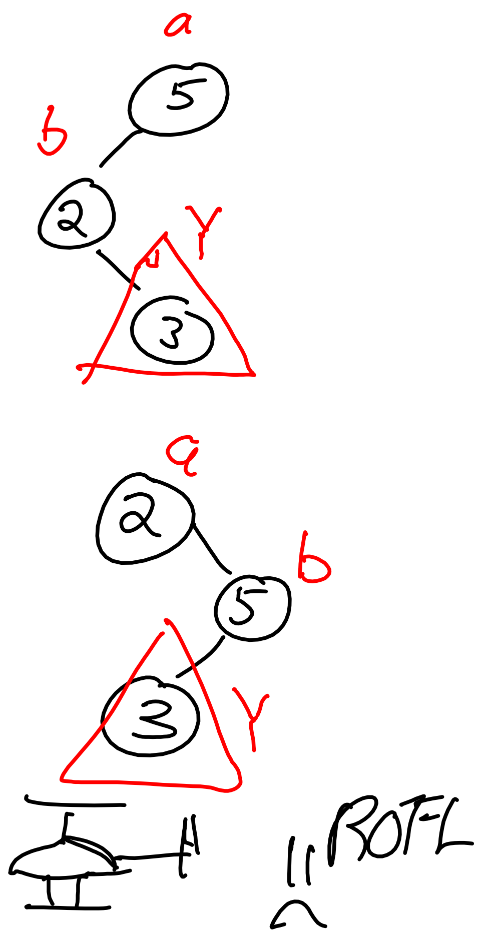


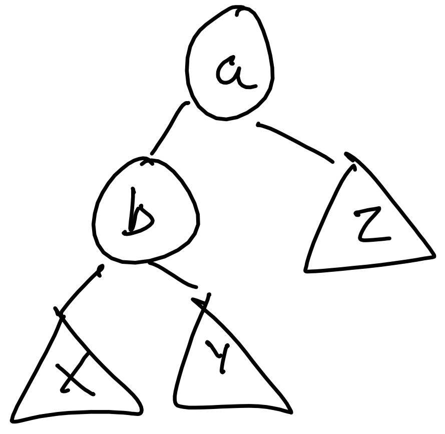
insert 3







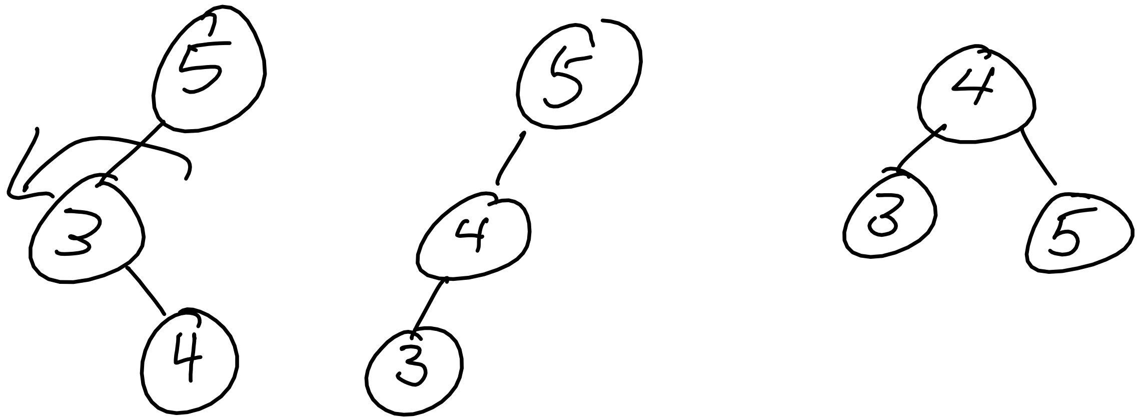


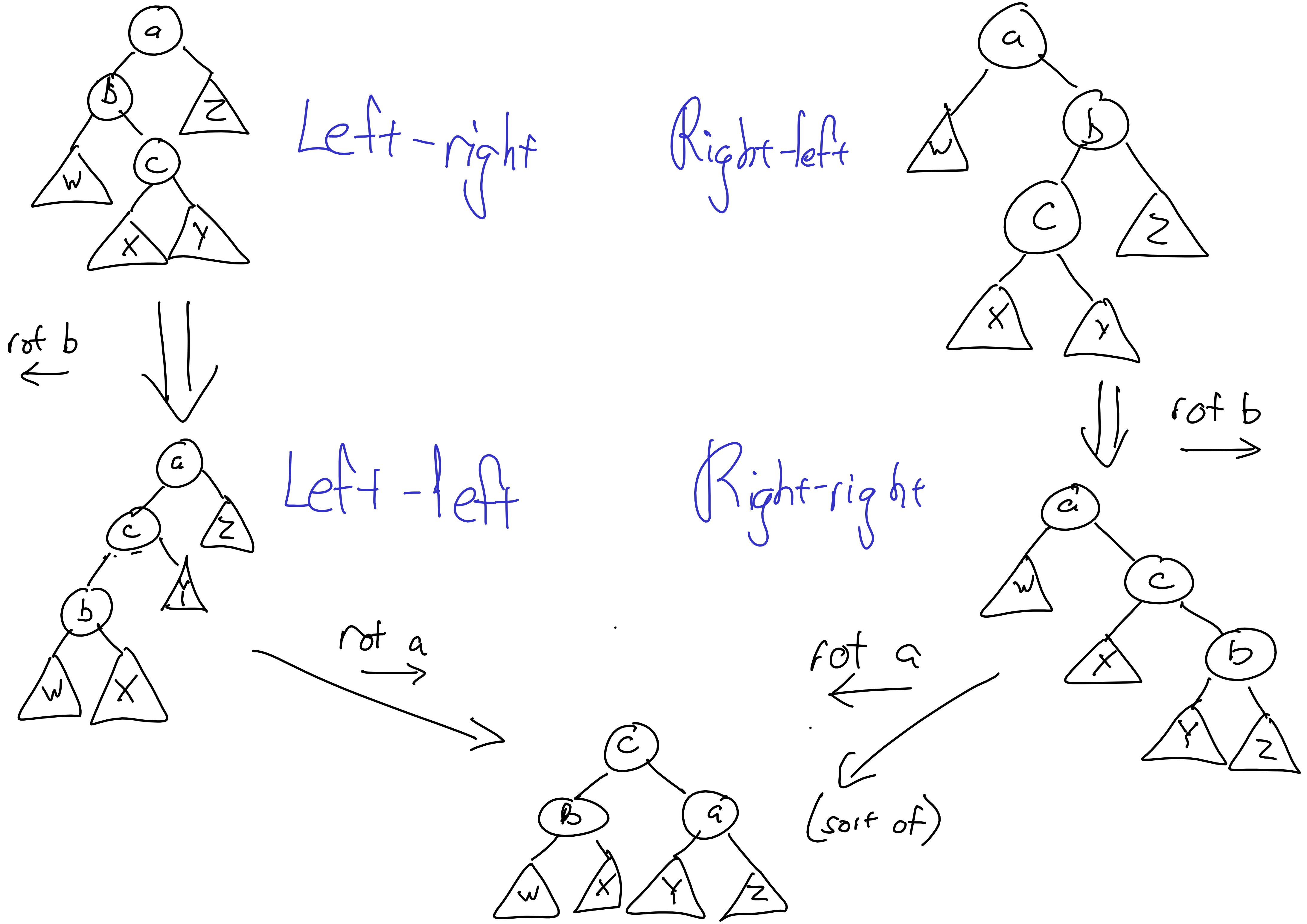


X < Y X ≥ Y

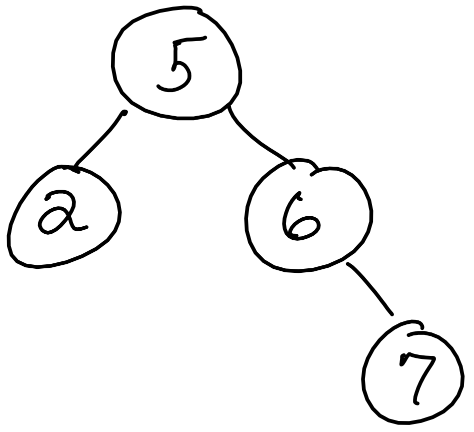
if HX < HY

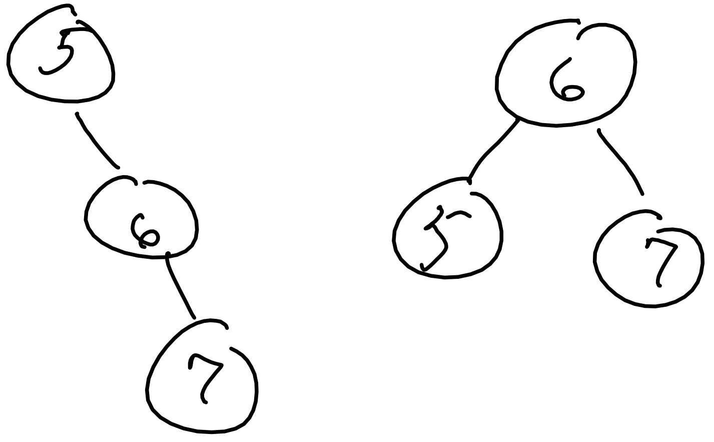
Rotate b left



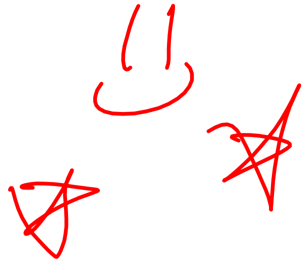

Left-right
Right-left
rot b
rot b
Left-left
Right-right
rot a
rot a
(sort of)

```
Function rebalance(Node root):
    d ← root→right→height − root→left→height
    If d > 1:
Right-left  {   d2 ← root→right→left→height − root→right→right→height
    ⇓       {   If d2 > 0:
Right-Right {       root→right ← rotateRight(root→right)
        root ← rotateLeft(root)
        Return root
    Else If d < −1
Left  {
      {           ⋮
    Else
        Return root
```





Remove 2



Last week — BSTs

AVL trees : balanced BST

all (important) operations : $O(\log(n))$
time