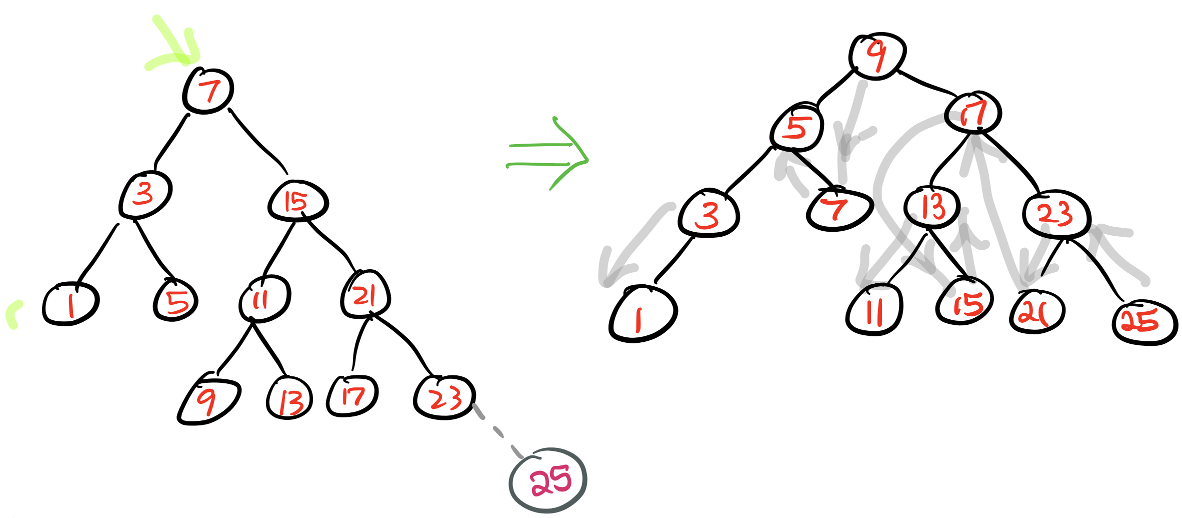
Balanced Binary Search Trees

- BSTs are great dictionaries, but can become unbalanced which leads to $O(n)$ running time per operation.

$\therefore$ we seek to do a little "housekeeping" as we go along to ensure the tree does not get out of balance...

Keeping it perfectly balanced....



if you have 11 items, the least-height tree has $h = 3$

Keeping it balanced, but not perfectly ....

$x \to BF$

Defn: The **balance factor** at a node (x) is

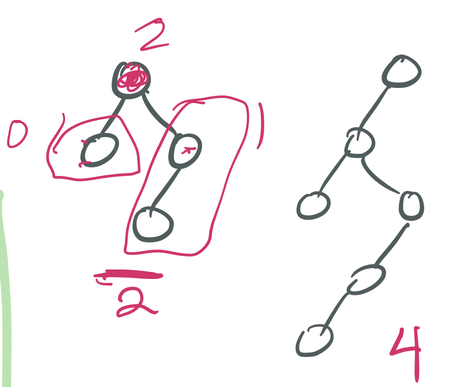
$$h(T_r) - h(T_\ell)$$

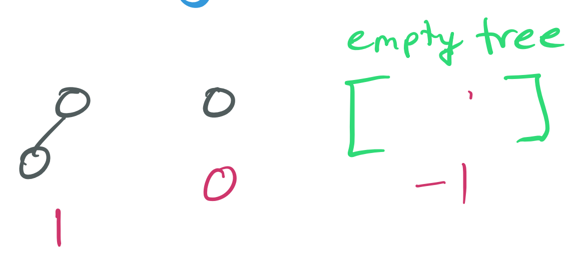
where T_r is right subtree and
T_ℓ is left subtree.

Defn: An AVL tree is a BST
where $\forall$ nodes have balance factor $\in \{-1, 0, 1\}$

A note about tree **height**, $h(T)$.

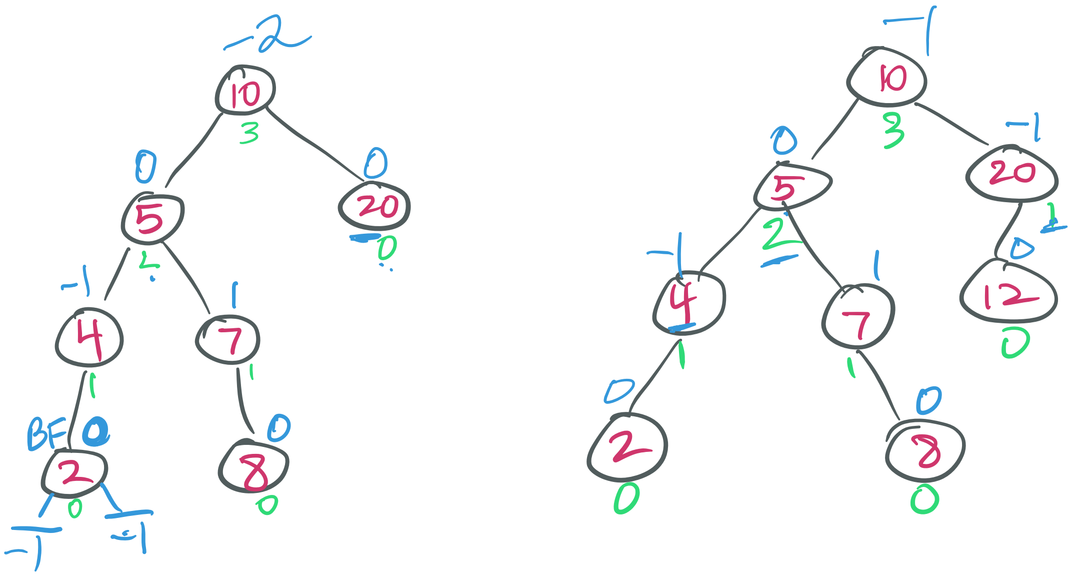
height is # edges on longest leaf-root path.





$$h(T) = 1 + \max(h(T_r), h(T_\ell))$$

Examples of AVL tree, and not AVL tree



Two things that would make AVL trees interesting:

① Are they "balanced enough" that heights stay $O(\log n)$?

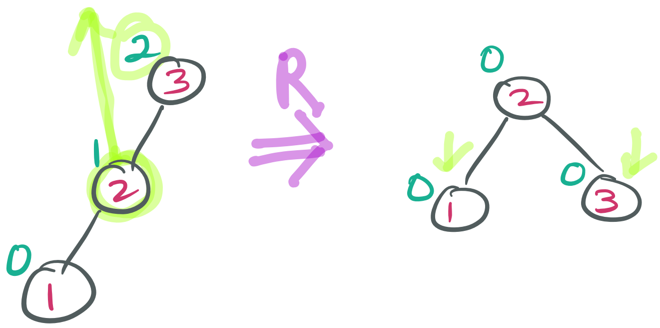
② Can we maintain "balance" (AVL-style) in time $O(\log n)$ per operation?

(Insert, Delete, Search, Max, Min, Succ, Pred)

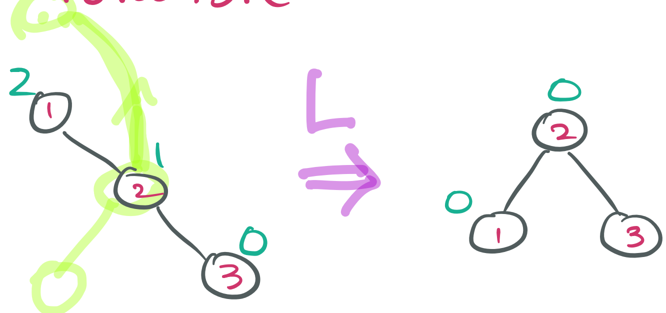
$\underbrace{\text{Insert, Delete}}_{\text{may need rebalancing.}}$

Tools for maintaining balance – AVL style:

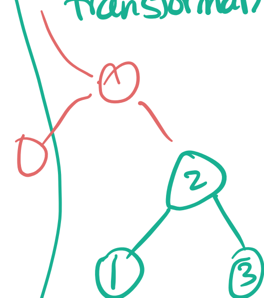
R - rotation



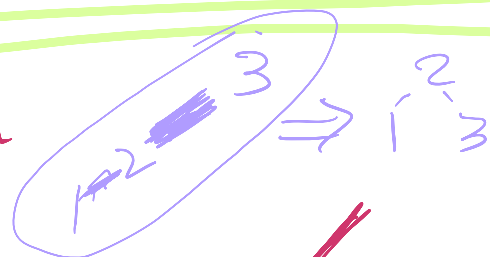
L - rotation

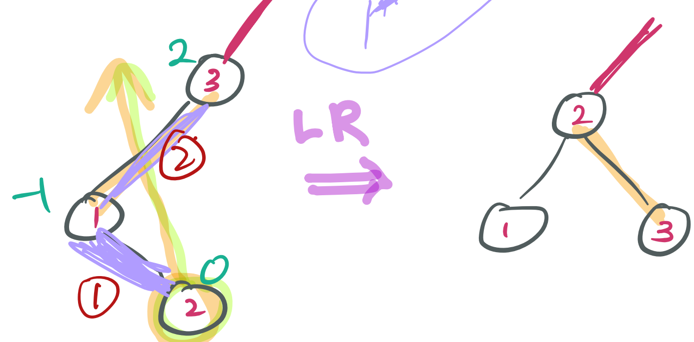


Single rotations are local transformations



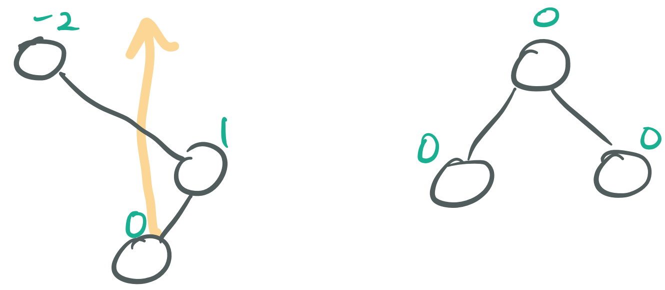
LR - rotation



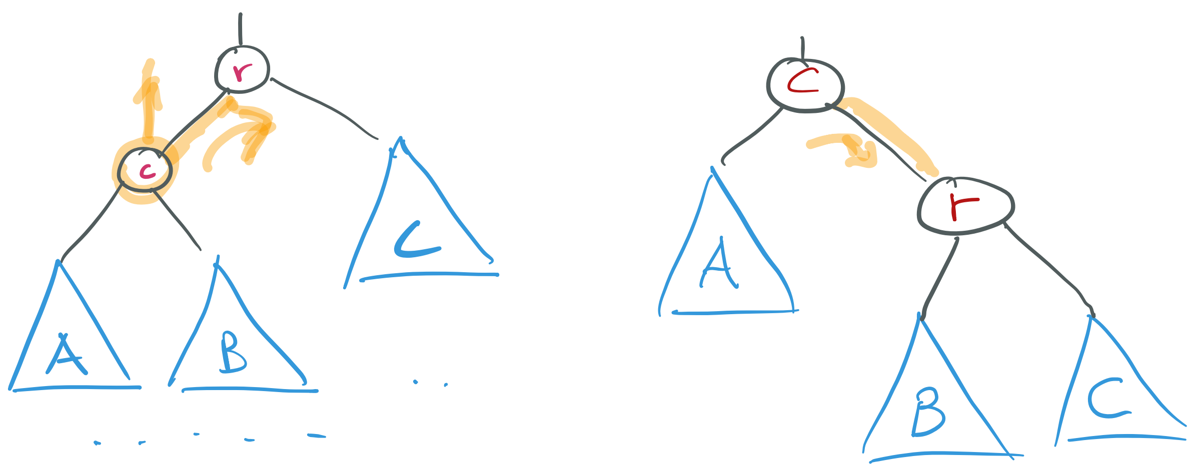


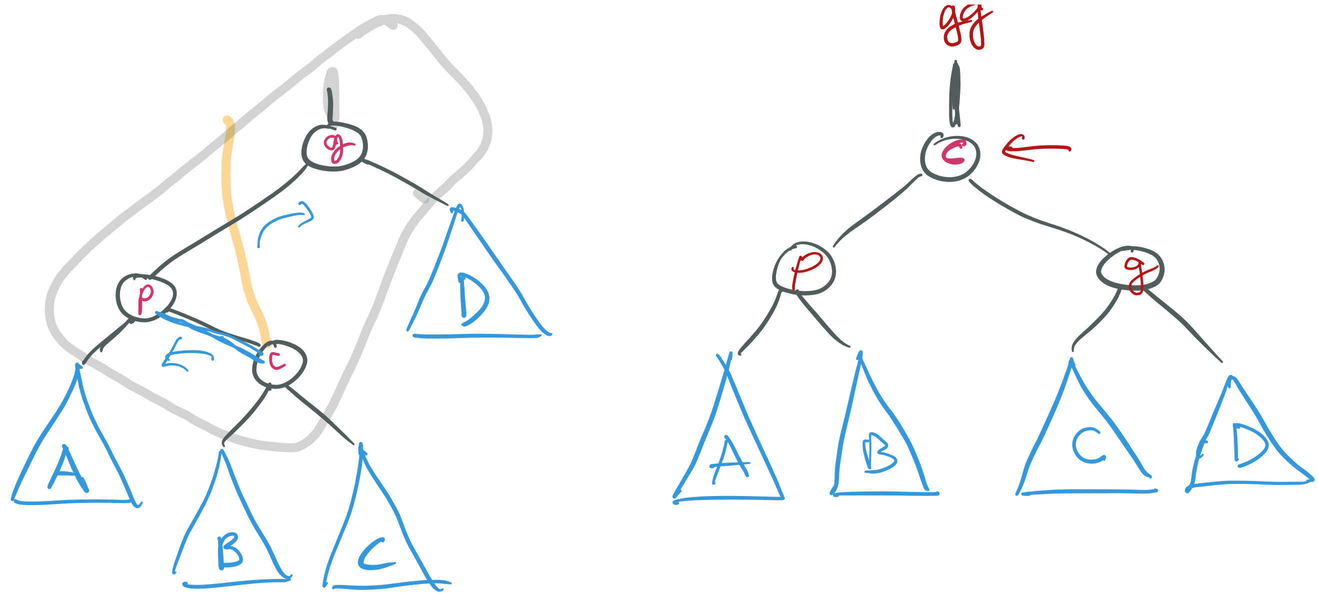
double rotations are also local

RL - rotation



Single Rotation $\underline{R}$





Each can be done in $O(1)$ time, but may propogate up ... total is $O(h)$. *

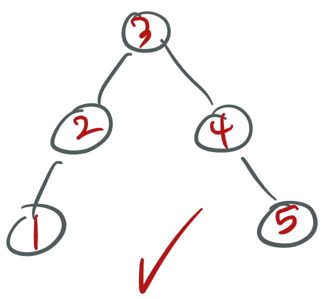
Is it an effective enough "rebalancing" that the height is kept small?

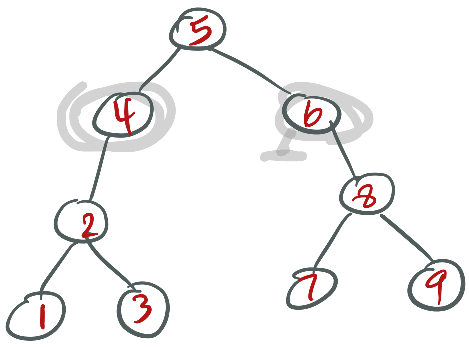
Theorem: $\forall$ AVL tree T, $h(T)$

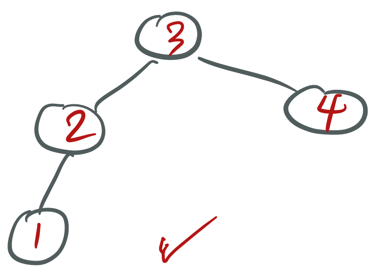
$$\lfloor \lg n \rfloor \le h(T) < 1.4405 \lg(n+2) - 1.3277$$

Questions

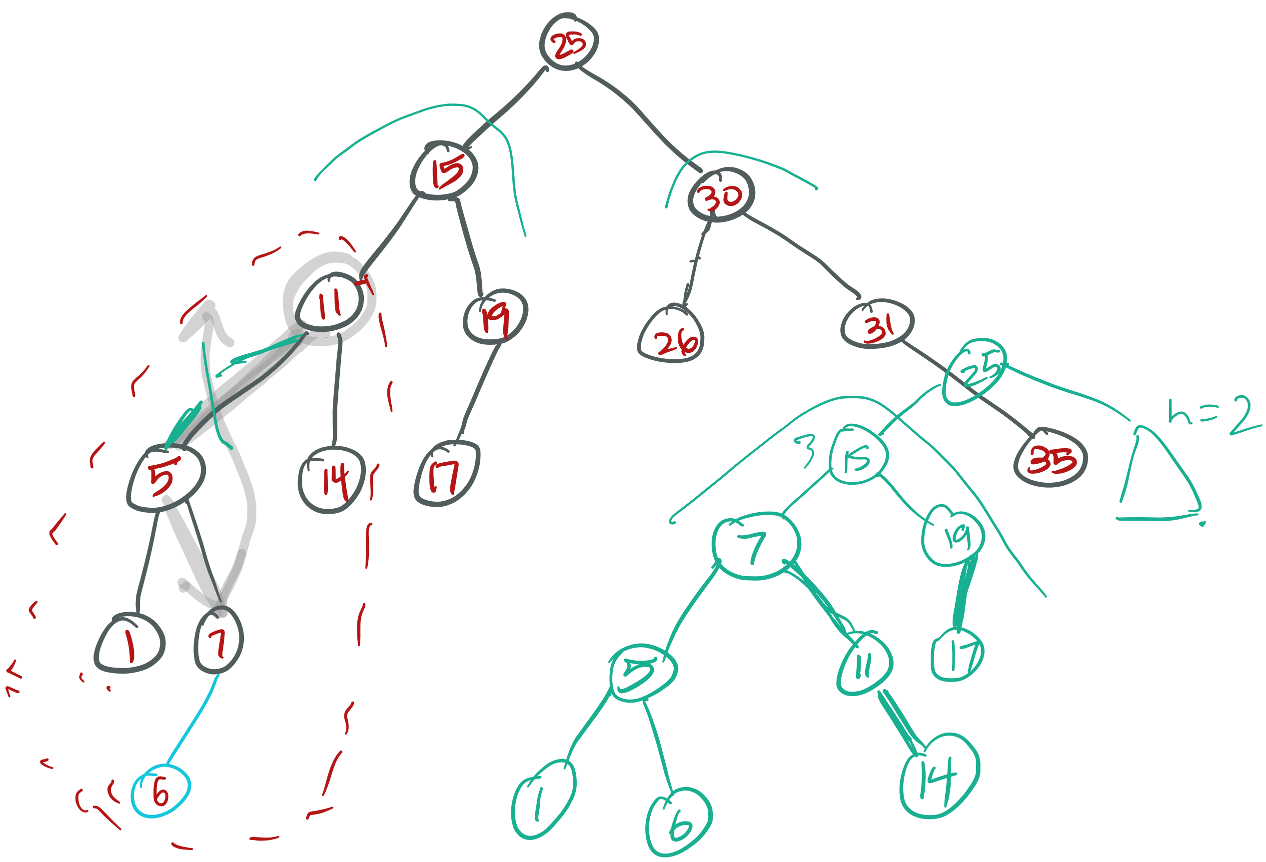
1. Which of these BSTs are AVL?







* How might the rotations propogate up?



Wikipedia - AVL trees.

```
/* after inserting z as child of x, height of
   z's subtree is now one more than the BFs indicate*/

for ( x=z->parent; x!=NULL; x=z->parent)

    if ( z is a right child)
        if z->BF > 0    // x is right-heavy
            g = x->parent
            if z->BF < 0
                w = rotateRightLeft(x,z)
            else
                w = rotateLeft(x,z)
        else
            if x->BF < 0
                x->BF = 0
                break
            ++ x->BF
            z=x
            continue
```

```
else        // z is a left child
    if  x->BF < 0
        g = x->parent
        if z->BF > 0
            w = rotateLeftRight(x, z)
        else
            w = rotateRight(x, z)
    else
        if x->BF > 0
            x->BF = 0
            break
        x->BF = -1
        z = x
        continue

w->parent = g
if  g != NULL                              x == x->parent->
    if (x is a left child)                             left
        g->left = w
```

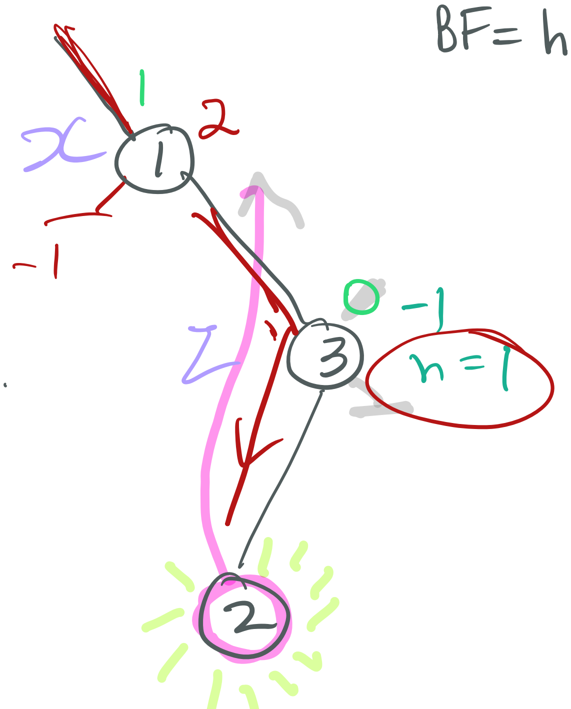
else $g \to right = w$

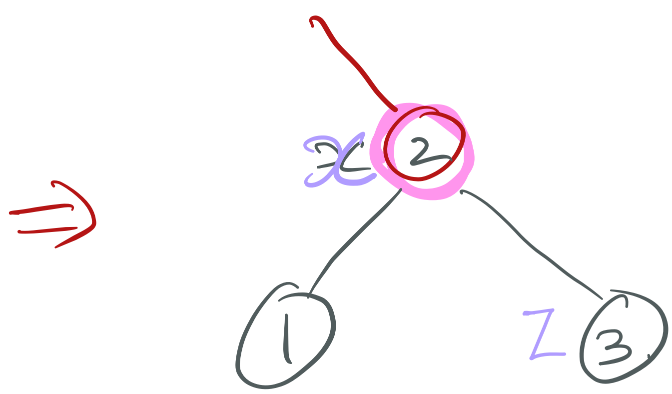
else

w is made root of tree

break

$BF = h_r - h_l$





Test Review: Unit Task Scheduling.

	a	b	c	d	e	f	g
deadline	1	1	4	6	2	5	6
	~~10~~	5	3	~~6~~	~~7~~	~~10~~	1

1	2	3	4	5	6	
a	e	g	c	f	d	b