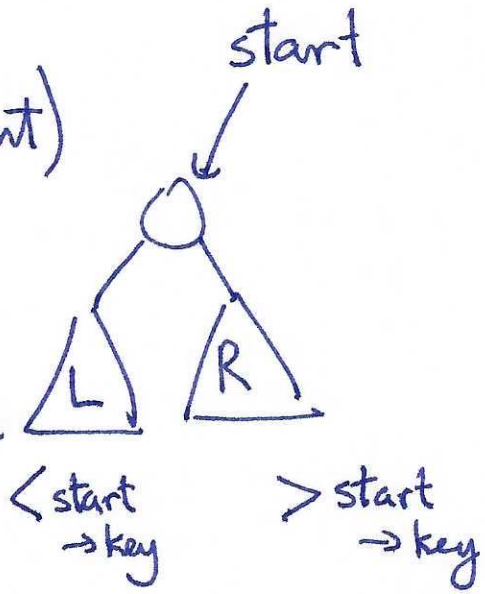
Tree traversals

Method of visiting the nodes of a tree in a specific order

InOrder, PreOrder, PostOrder
Traversal

Print keys of nodes. (Key is int)




```
void InOrder(Node* start)
{
    if (start == NULL) return; //base case
    InOrder(start->left);
    print(start->key);
    InOrder(start->right);
}
```

Print L in sorted order
Print start
Print R in sorted order

```
void PreOrder(Node* start)
{
    //Base case
    print(start->key);
    PreOrder(start->left);
    PreOrder(start->right);
}
```

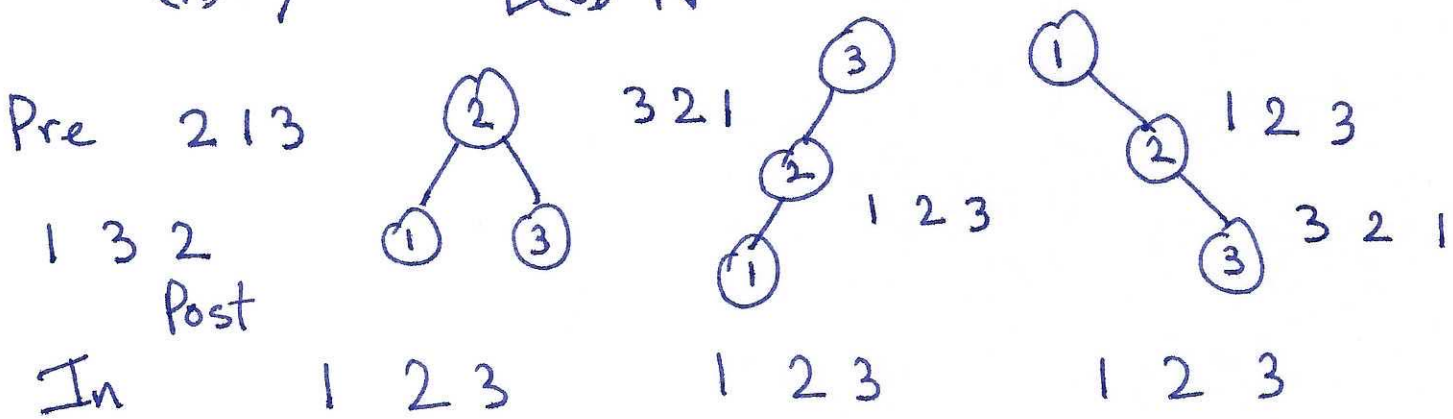
```
void PostOrder(Node* start)
{
    //Base case
    PostOrder(start->left);
    PostOrder(start->right);
    print(start->key);
}
```

Q. Which traversal prints keys in sorted order?

(A) In (B) (G) Pre (B) Post

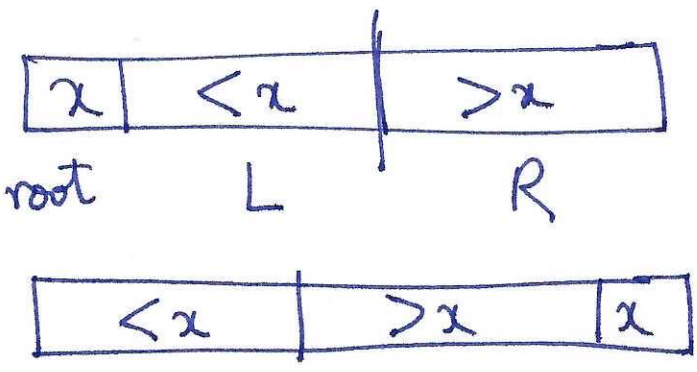
Q. Given in-order traversal of a BST, can you reconstruct the BST?

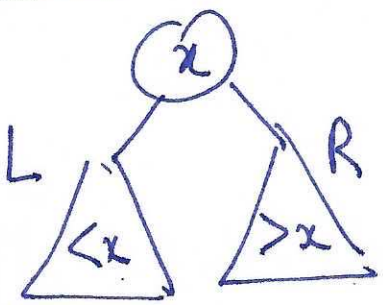
(R) Y (G) N



Q. Given PreOrder traversal of a BST, can you reconstruct the BST? (Proof by induction)

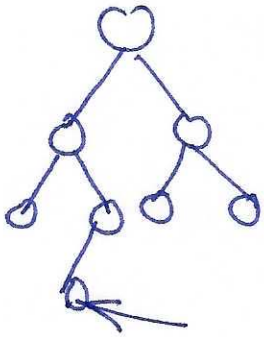
(R) Y (G) N





Self-balancing BSTs

What is worst-case running time of insert/delete/find in BST?



Suppose there are n nodes in a BST.

Suppose the max depth is D.

What is running time of find?

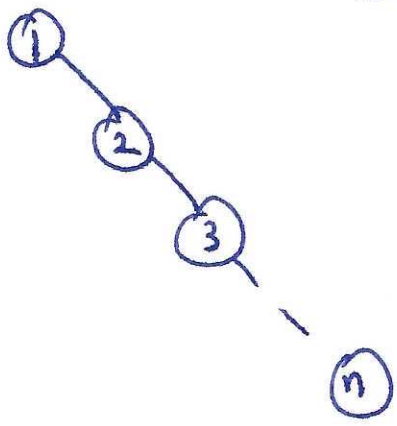
→ "distance" of the furthest leaf to the root

→ # ancestors in the path

(R) $\Theta(1)$ ~~(G) $\Theta(D)$~~ (B) $\Theta(n)$

What is the largest possible depth?

(R) $\Theta(1)$ (G) $\Theta(\log n)$ ~~(B)~~ $\Theta(n)$



Insert in sorted order

BST looks like linked list

find, insert, delete, findmin, findmax, succ, pred all run in $\Theta(D)$ in worst case.

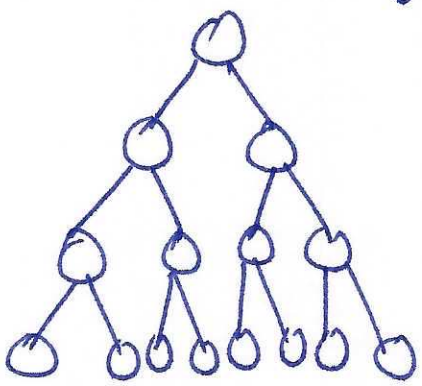
Q. For a BST with n nodes, what is the smallest possible (max) depth?

(R) $\Theta(1)$ ~~(G)~~ $\Theta(\log n)$ (B) $\Theta(n)$

Lemma: In any BST with n nodes, there exists a node at depth at least ~~$\log_2 n - 1$~~ $\lg n - 1$.

Perfectly balanced BST has max depth of exactly $\lceil \lg n - 1 \rceil$.

Depth is 3
$\lceil (\lg 15) - 1 \rceil$



15 nodes

Self-balancing BST

Always maintains max depths of $\Theta(\lg n)$, so all operations we discussed run in $\Theta(\lg n)$.

↳ More complex: requires tree "rebalancing" to reduce depth.

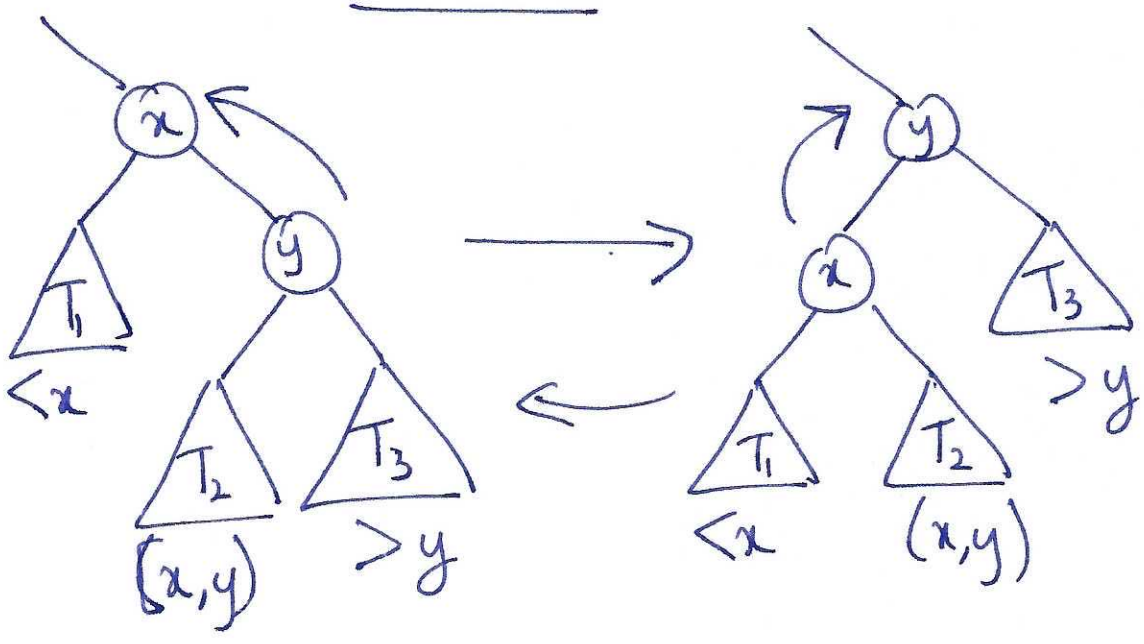
60's AVL Tree: Adelson-Velskii, Landis tree } Conceptually simpler

60's 2-3 Tree: Not a BST, nodes have many keys

Red-Black Tree : C++ stdlib map

B-Tree : used in databases (generalizes 2-3 trees)

Rotation

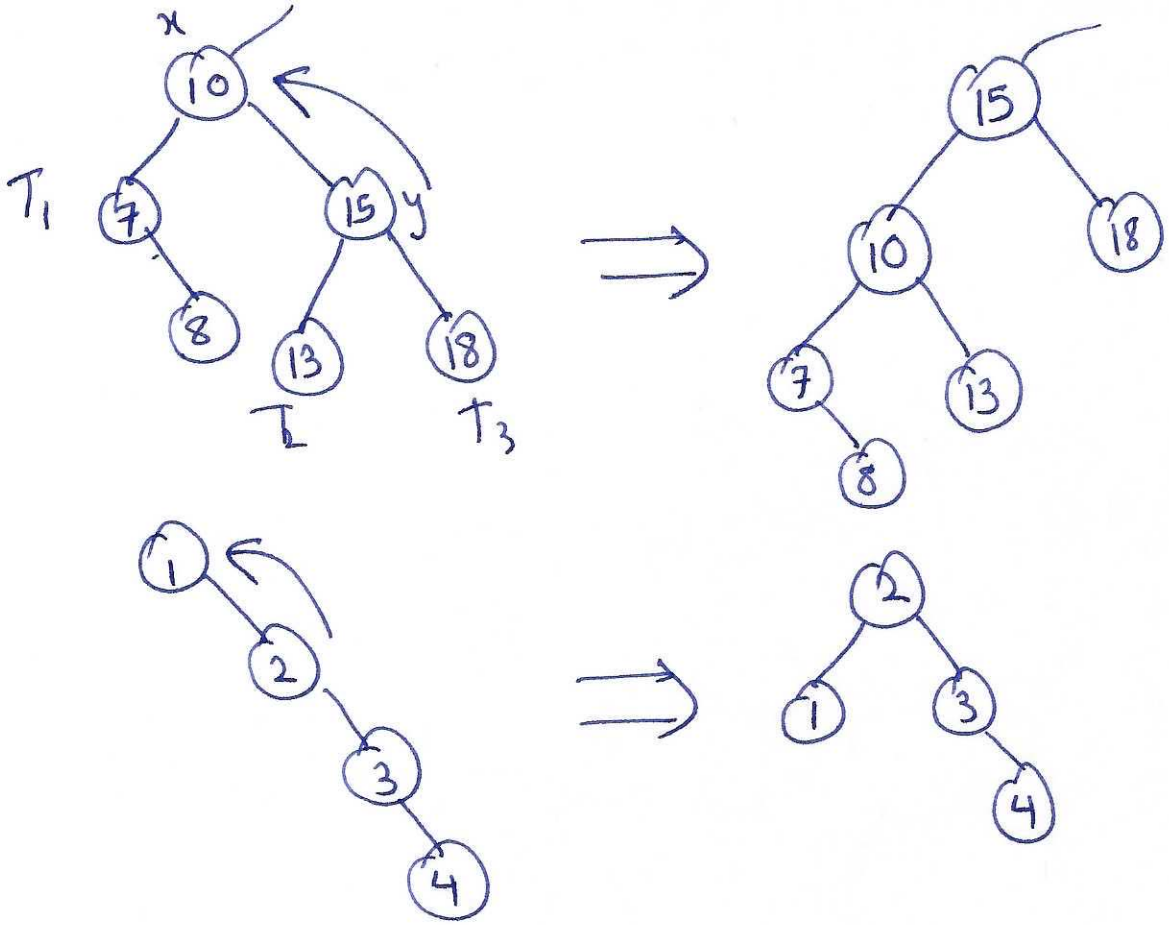


Suppose tree has n nodes. How much time to perform ~~one operation~~ a rotation in the worst case?

~~(R)~~ $\Theta(1)$ (G) $\Theta(\log n)$ (B) $\Theta(n)$

We only change (1) children of x and y.
(2) parents of x and y
(3) parents of roots of T_1, T_2

No change within T_1, T_2, T_3.



AVL Tree

Height (of a subtree): Max distance of a leaf to the root of the subtree

AVL Trees maintain a special height property.

x : node

BalanceFactor(x) = Height(left subtree of x) − Height(right subtree of x)

AVL property/invariant:

$\forall$ x, BalanceFactor(x) $\in \{-1, 0, 1\}$

Lemma: An AVL tree with height h has at least F_{h+2} nodes (F_k is Fibonacci number)

$F_0 = 0 \quad F_1 = 1 \quad F_2 = 1 \quad F_3 = 2 \qquad F_k = F_{k-1} + F_{k-2}$.

$$\boxed{F_{h+2} \geq \phi^{h-2}} \qquad \phi = \text{Golden Ratio} = \frac{1+\sqrt{5}}{2}$$

If an AVL tree has n nodes, what is a bound on the height? $h \leq \log_\phi n = \Theta(\lg n)$

$\log_2 n \leftarrow$ Perfectly balanced

$\lim_{h \to \infty} \frac{F_h}{\phi^h} = \Theta(1) \qquad 2 > \phi \geq 1.618$

$$F_h \sim \frac{\phi^h}{\sqrt{5}}$$