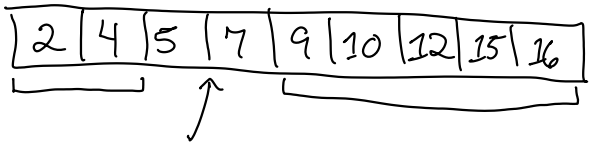
3	aperson 1	pw1
5	aperson 2	pw2
7	aperson 3	pw3

List<User*> users

	LL	AL
get(0)	O(n)	O(1)
get(1)		
⋮		

If my list is sorted: binary search: O(log n)



n * O(1) = O(n)

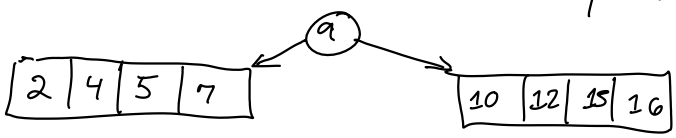
1,000,000,000 users
≈ 2^{30} users

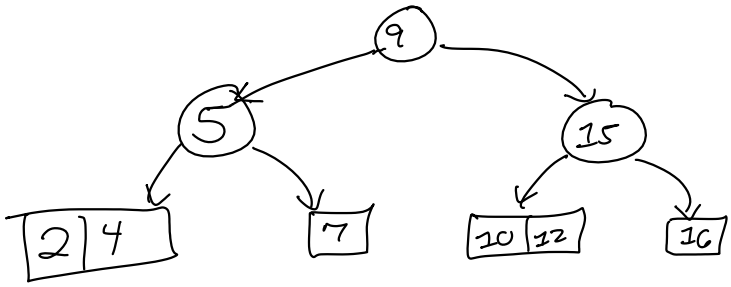
$\frac{O(n)}{1{,}000{,}000{,}000}$ $\frac{O(\log n)}{30}$

Suppose: all new users have # > than existing users:

amortized O(1)

Suppose: all new users have # > than existing users
or # exactly in the middle:





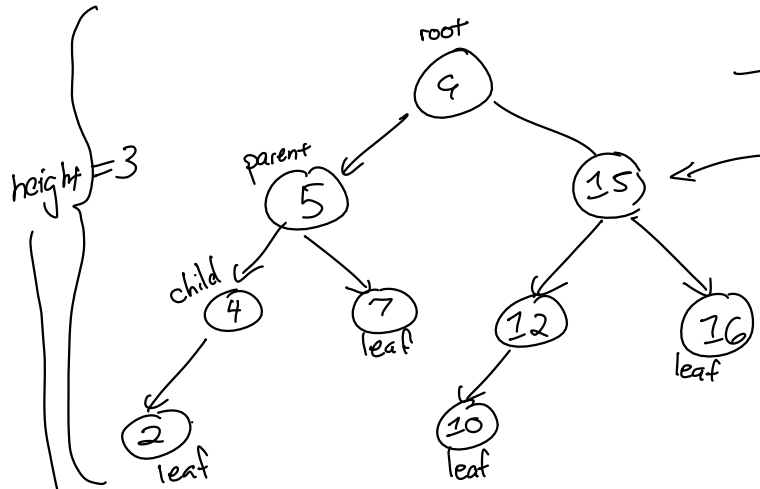

h=1


h=0

h=-1

Example of a Binary tree:

free where each node has at most 1 left child & 1 right child



Tree

Node

leaf: node w/ no children

root: the node w/ no parent

Binary Search Tree:

Binary Tree where all left descendents are less, all right descendents are greater

height: # of edges (steps) from root to the furthest leaf

password get(username)

```
V    get(K)
void insert(K, V)
void update(K, V)
V    remove(K)
int  size()
bool isEmpty()
```

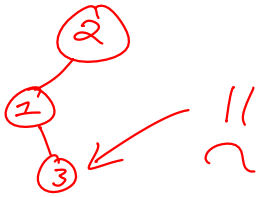
} ADT Dictionary

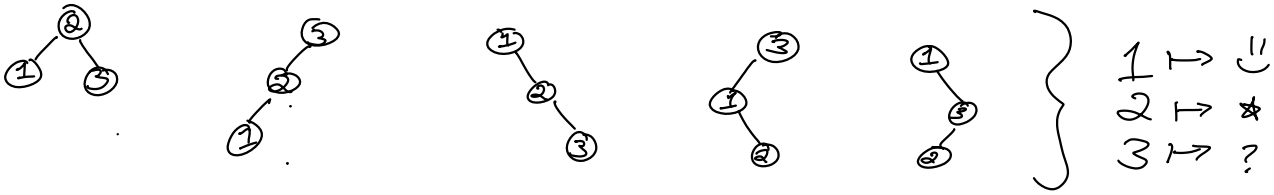
Binary Search Tree (BST) is a kind of Dictionary

Linked List is a kind of List

Exercise: all binary trees of size 2

of size 3





BSTs

invariant: at each node, all left descendants have lesser key, all right descendants have greater key

```
Method get(key):
    If root == null:
        (throw exception)
    End If
    node ← findInSubtree(key, root)
    Return node.value
End Method

Method findInSubtree(key, root)
    If root == null:
        
    If root.key == key:
        Return root
    Else If root.key < key:
        Return findInSubtree(key, root.right)
    Else
        Return findInSubtree(key, root.left)
    End If
End Method
```