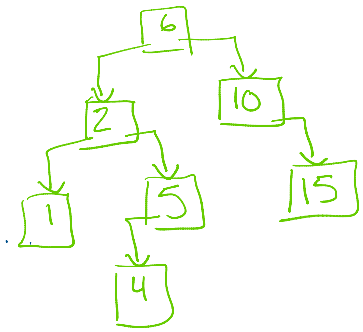
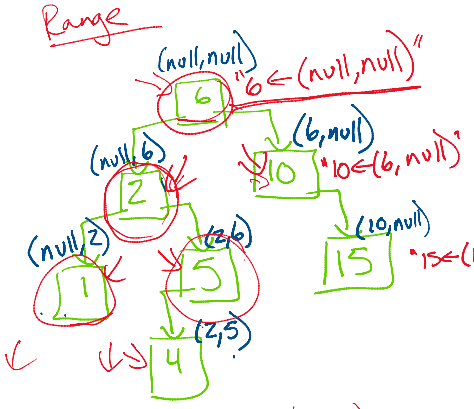
Week 8: BSTs/Recursion

Friday, October 30, 2020 11:54 AM

→ root
→ nodes
  → left
  → right
* 2 rules:
  → left always less than parent
  → right always greater than parent



Range



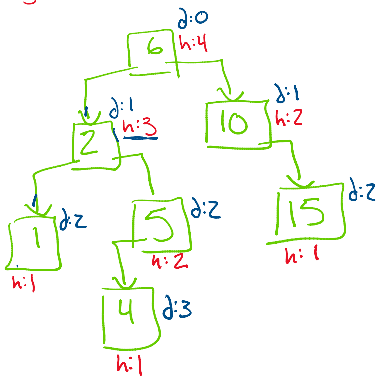
left's range:
→ low: parent's low
→ high: parent

right's range:
→ low: parent
→ high: parent's high

p's

doTreeRange(n.left, left's low, left's hi, set)

height = max(children's height) + 1



height: Start at leaves (bottom) & calc up.
→ height of leaf: 1
→ height of null: 0

depth: Starts at root & calc down
→ depth of root: 0

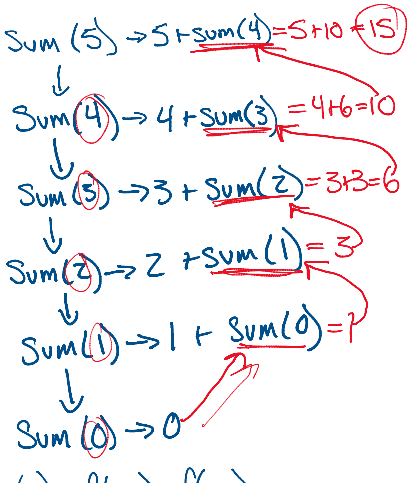
Recursion:
→ call method inside itself

```
int sum(int n) -> return sum of 0 to n
{
    if(n==0) return 0;
    return n + sum(n-1);
}
```

→ if passing info down:
  → put in parameter
→ if passing info up:
  → put in return

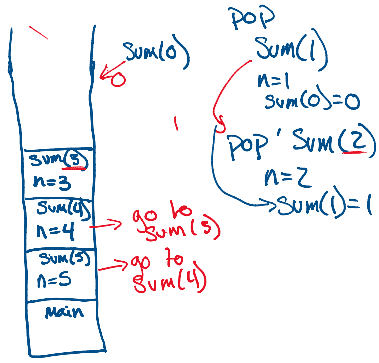
→ Use recursion to solve problems w/ subproblems
  → Subproblems almost identical

Recursion tree



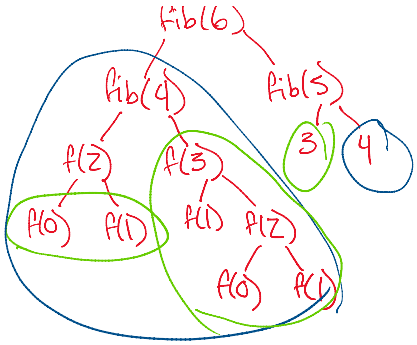
fib(n) = f(n-2) + f(n-1)

0, 1, 1, 2, 3, 5, 8 ...



Call Stack
* code
* "State" / variables

→ use recursion
Problems w/ subproblems
→ Subproblems almost identical
* don't use if recalculating something you already know

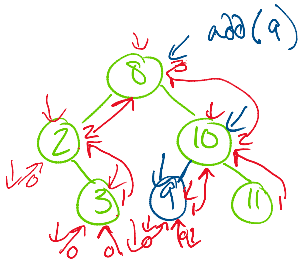


Traversals

→ pre-order:
→ process current node
→ go left
→ go right
* children need info from parent
* check in Range

→ in-order:
→ go left
→ process current
→ go right

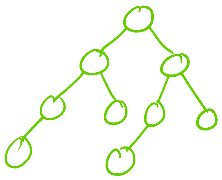
*→ post-order:
→ go left
→ go right
→ process current
* need info from children
* heights



add
→ compare current to what we want to add
→ go left or right
*pre-order

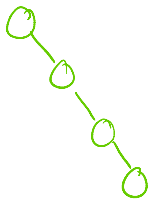
Runtime

best:



→ if going down 1 subtree (add, contains)
→ O(log n)
* dividing # of nodes in half

worst:



O(n)

* if doing operation that looks at every node
→ doHeight
→ checkInRange
O(n)