Binary Search Trees

CS16: Introduction to Data Structures & Algorithms

Spring 2018

Outline

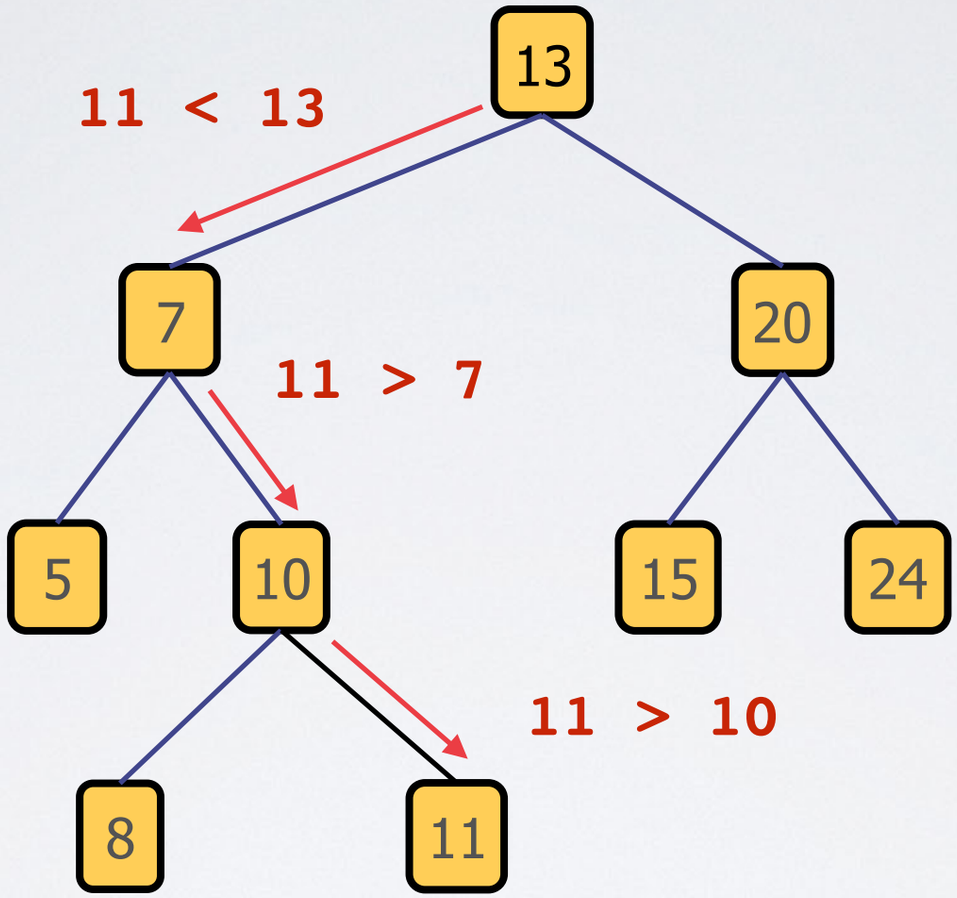
- Binary Search Trees
- Searching BSTs
- Adding to BSTs
- Removing from BSTs
- BST Analysis
- Balancing BSTs

by CynthT
http://cyntht.deviantart.com/

Binary Search Trees

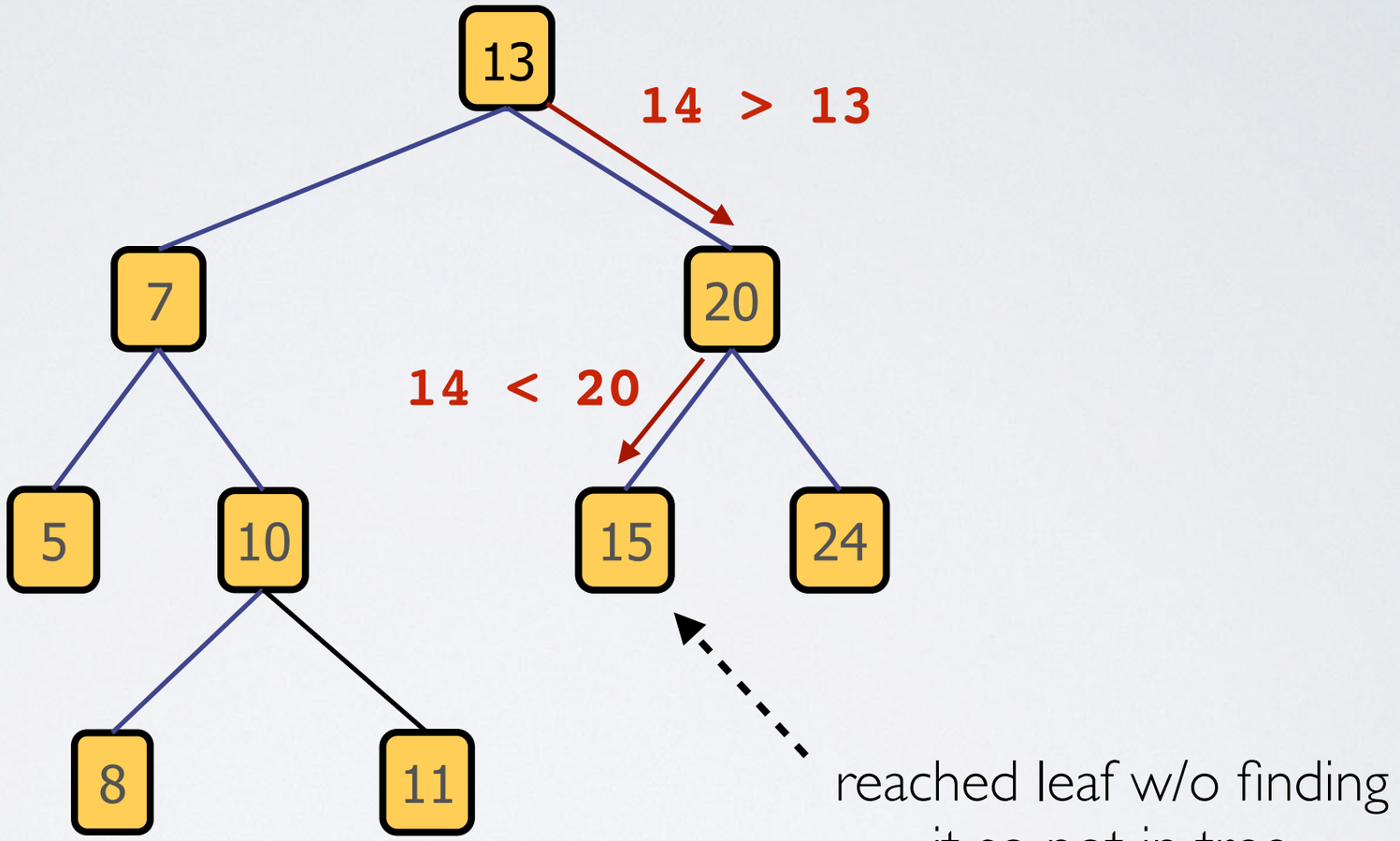
- Binary trees with special property
  - For each node
    - left descendants have lower value than node
    - right descendants have higher value than node
- In-order traversal gives nodes in order

Searching a BST



- Find `11`
- Each comparison tells us whether to go left or right

Searching a BST



- What if item isn't in tree?
- Find **14**

Binary Search Tree — Find( )

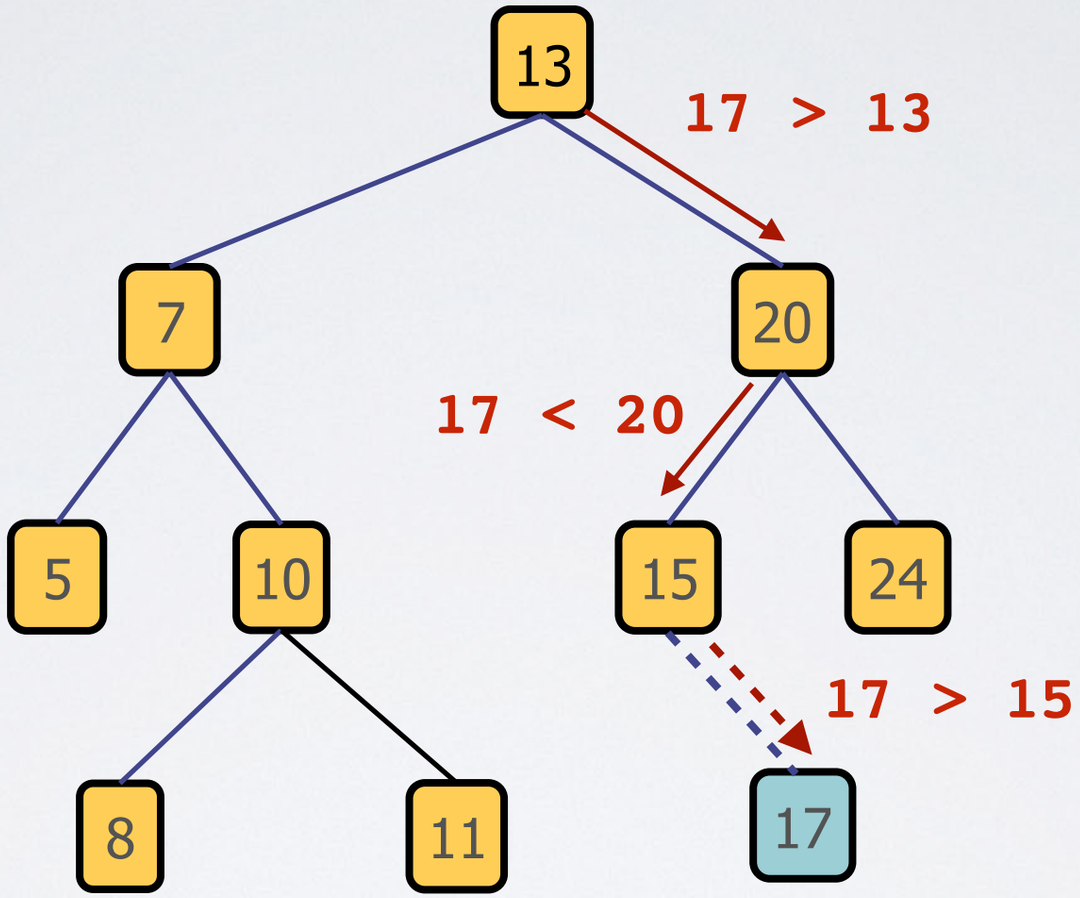
```
function find(node, toFind):
   if node.data == toFind:
      return node

   else if toFind < node.data and node.left != null:
      return find(node.left, toFind)

   else if toFind > node.data and node.right != null:
      return find(node.right, toFind)

   return null
```

Inserting in a BST



- To insert, perform a search and add as new leaf
- Insert **17**

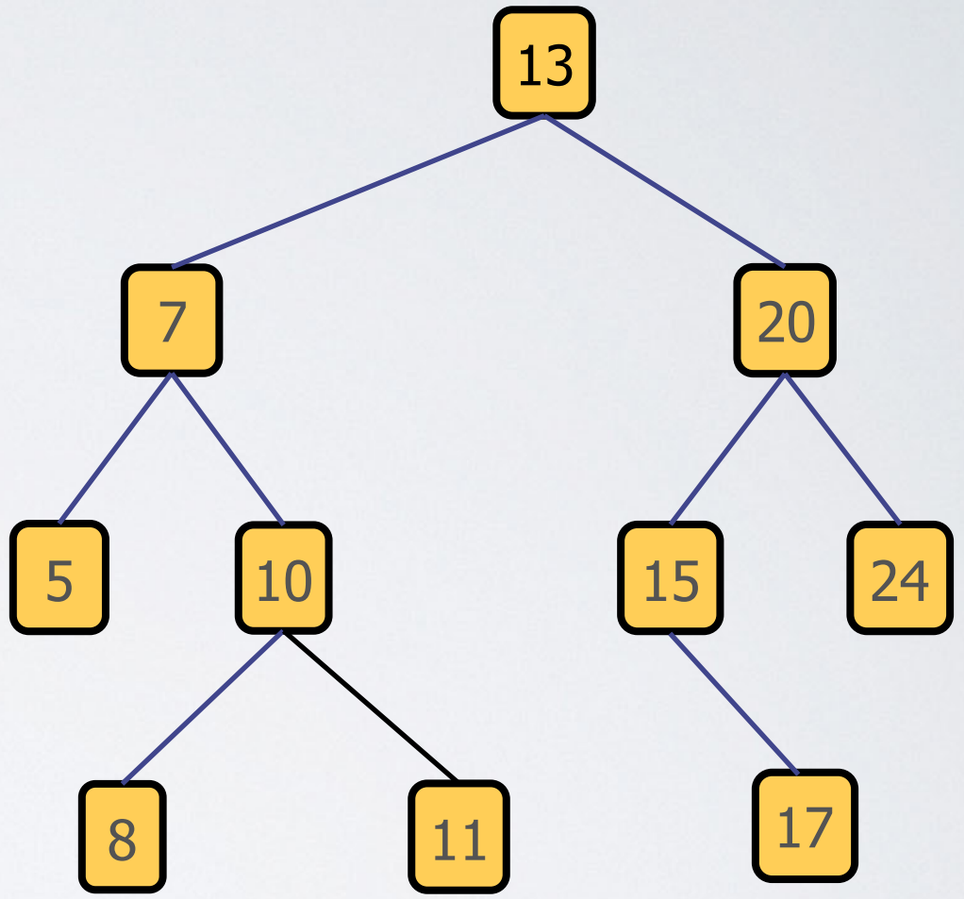
Binary Search Tree — Insert( )

```
function insert(node, toInsert):
  if node.data == toInsert:  # data already in tree
    return

  if toInsert < node.data:
    if node.left == null:    # add as left child
      node.addLeft(toInsert)
    else:
      insert(node.left, toInsert)
  else:
    if node.right == null:   # add as right child
      node.addRight(toInsert)
    else:
      insert(node.right, toInsert)
```

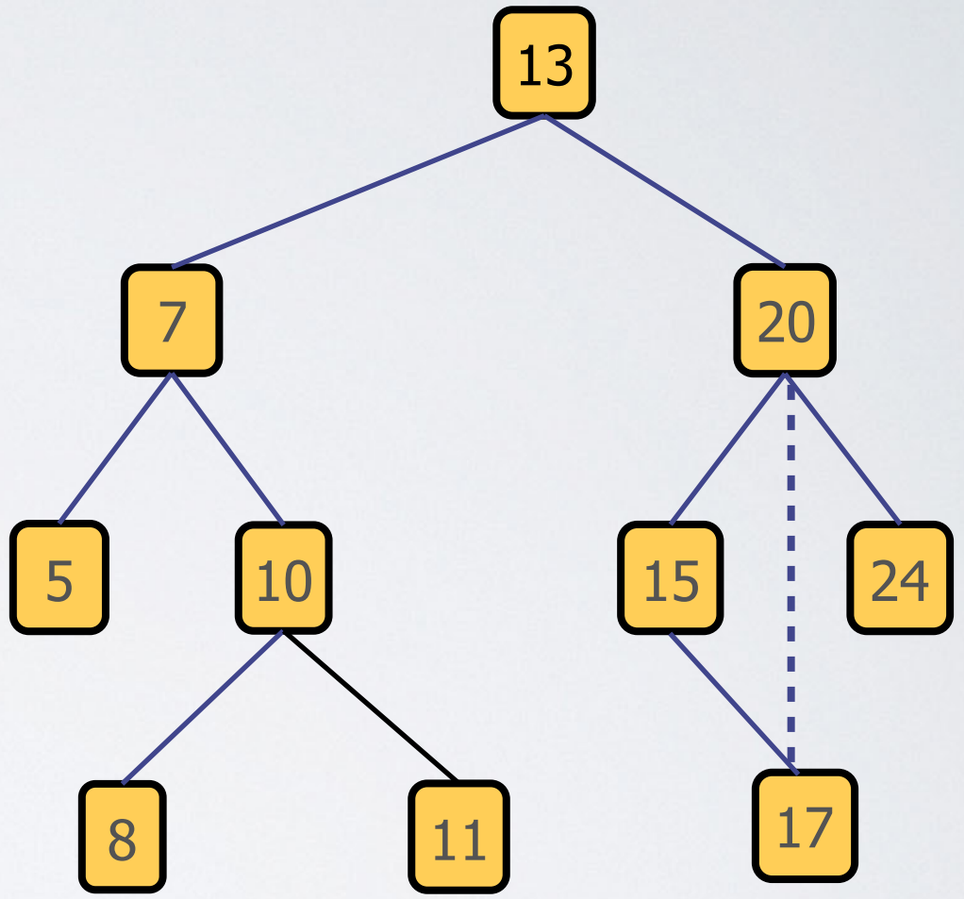
Removing from a BST

- Can be tricky
- Three cases to consider
  - Removing a leaf: easy, just do it
  - Removing internal node w/ **1** child (e.g., **15**)
  - Removing internal node w/ **2** children (e.g., **7**)



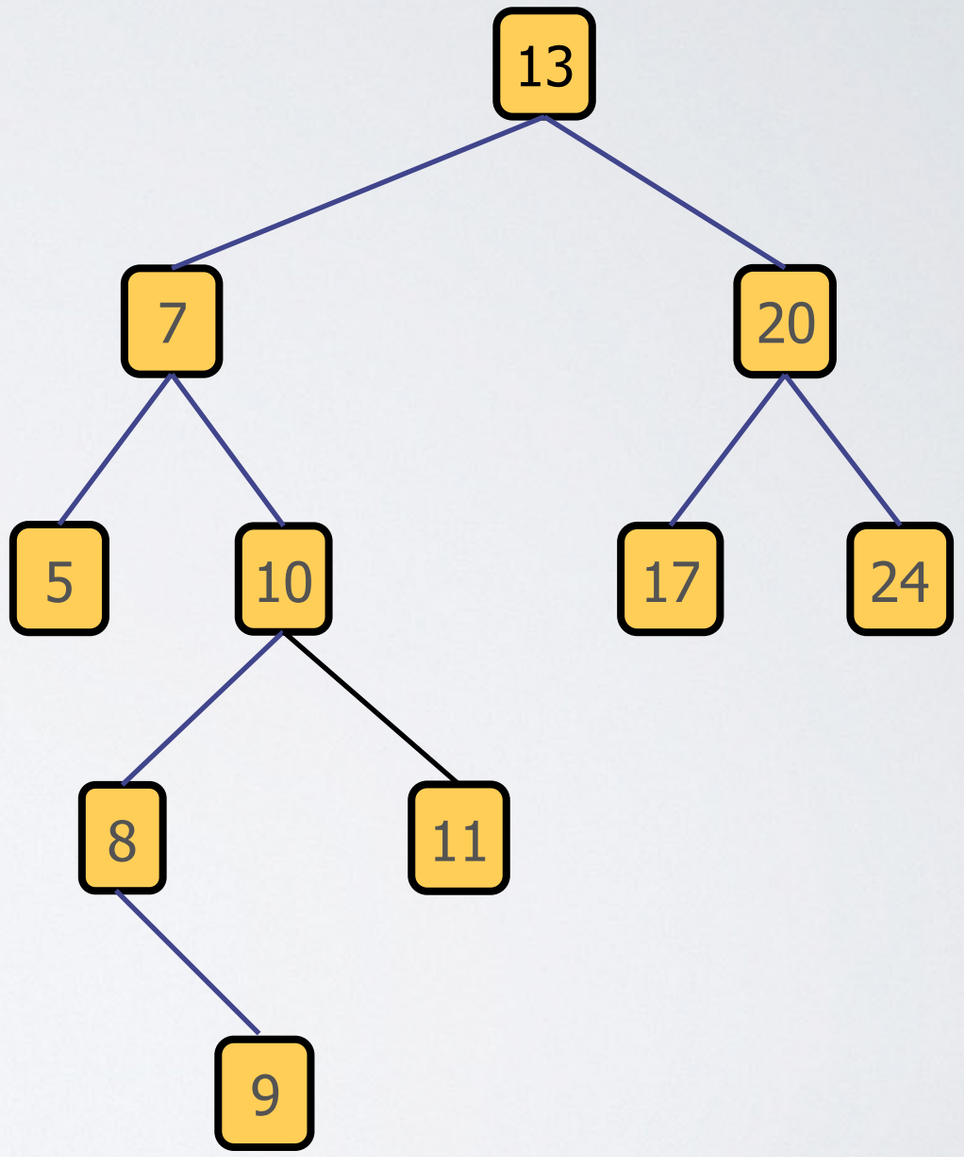
Removing from a BST - Case #2

- Removing internal node w/ `1` child
- Strategy
  - "Splice out" node by connecting its parent to its child
- Example: remove `15`
  - set parent's left pointer to 17
  - remove 15's pointer
  - no more references to 15 so erased (garbage collected)
  - BST order is maintained



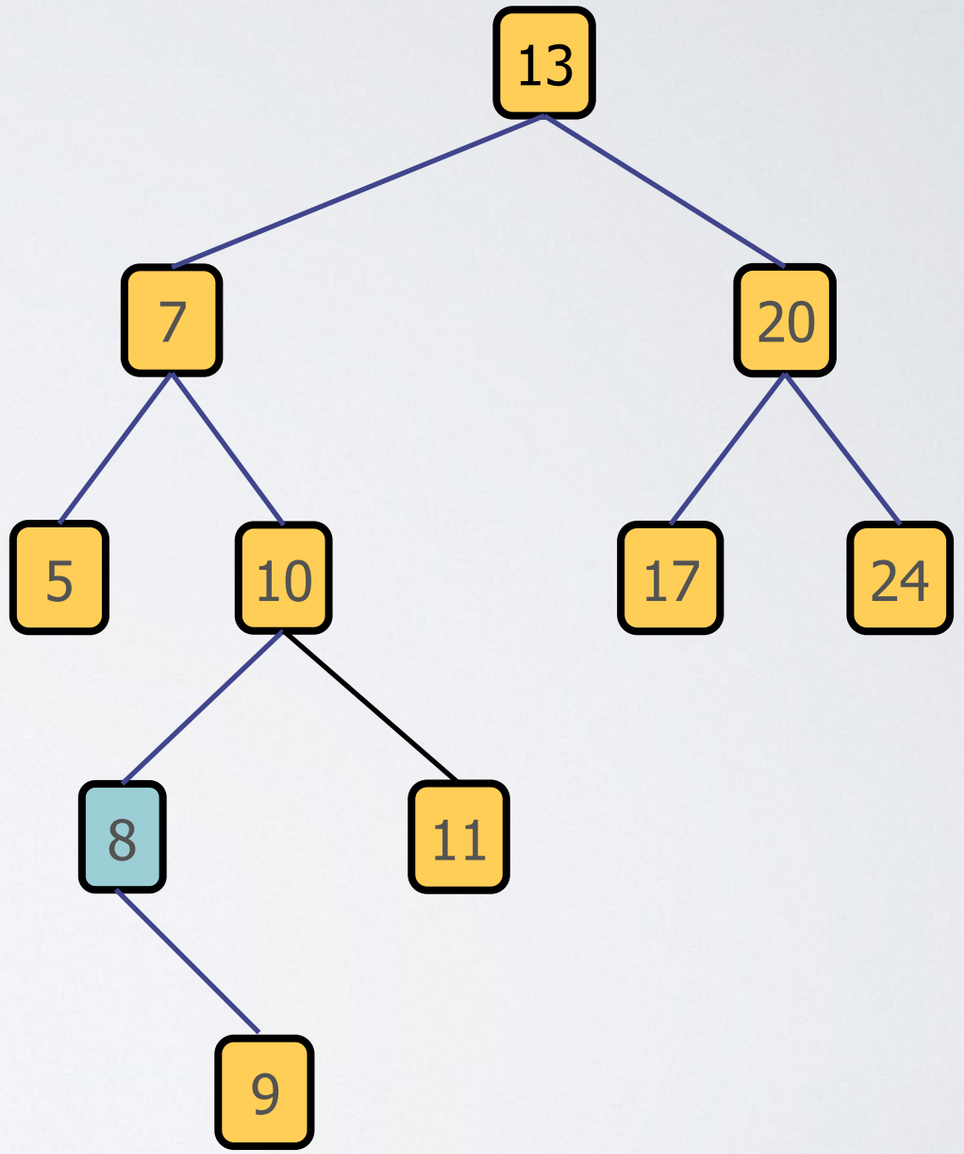
Removing from a BST - Case #3

- Removing internal node w/ **2** children
- Replace node w/ successor
  - successor: next largest node
- Delete successor
  - Successor a.k.a. the in-order successor
- Example: remove **7**
  - What is successor of **7**?



Removing from a BST - Case #3

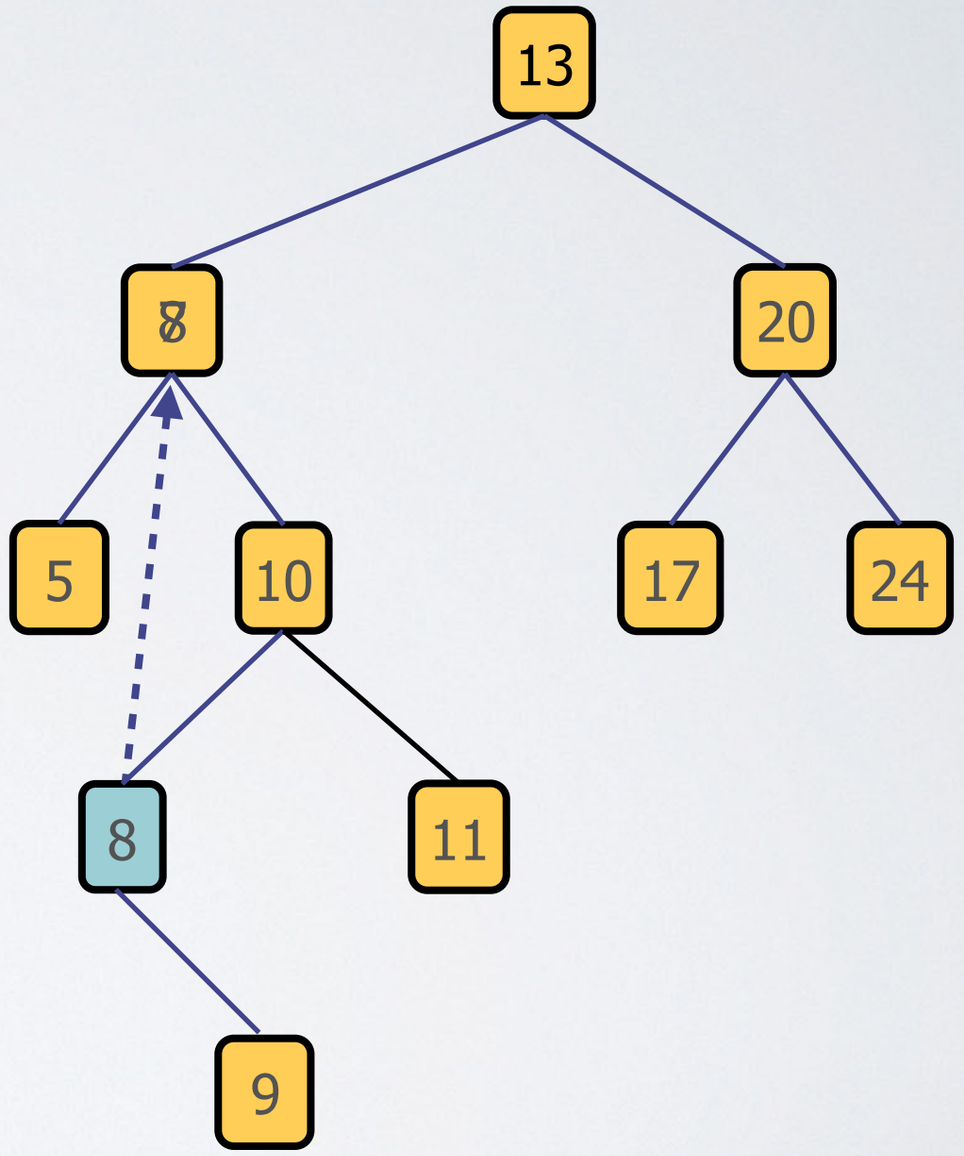
- Since node has **2** children…
  - …it has a right subtree
- Successor is leftmost node in right subtree
- 7's successor is **8**




```
successor(node):
   curr = node.right
   while (curr.left != null):
      curr = curr.left
   return curr
```

Removing from a BST - Case #3

- Now, replace node with successor
- Observation
  - Successor can't have left sub-tree
  - …otherwise its left child would be successor
  - so successor only has right child
- Remove successor using Case #1 or #2
  - Here, use case #2 (internal w/ 1 child)
- Successor removed and BST order restored



Binary Search Tree — Remove( )

```
function remove(node):
   if node has no children:  # case 1
      node.parent.removeChild(node)
   else if node only has left child:   # case 2a
      if node.parent.left == node:  # node is a left child
         node.parent.left = node.left
      else:
         node.parent.right = node.left
else if node only has right child:  # case 2b
   if node.parent.left == node:
      node.parent.left = node.right
   else:
      node.parent.right = node.right
   else:  # case 3 (node has two children)
      nextNode = successor(node)
      node.data = nextNode.data  #replace w/ nextNode
      remove(nextNode) # nextNode has at most one child
```

Successor vs. Predecessor

- In Remove( )
  - OK to remove **in-order predecessor** instead of in-order successor

    Randomly picking between the two keeps tree balanced
- In Case #3
  - Predecessor is rightmost node of left subtree

Binary Search Tree — Remove( )

Activity #1

2 min

Binary Search Tree — Remove( )

Activity #1

2 min

Binary Search Tree — Remove( )

Activity #1

1 min

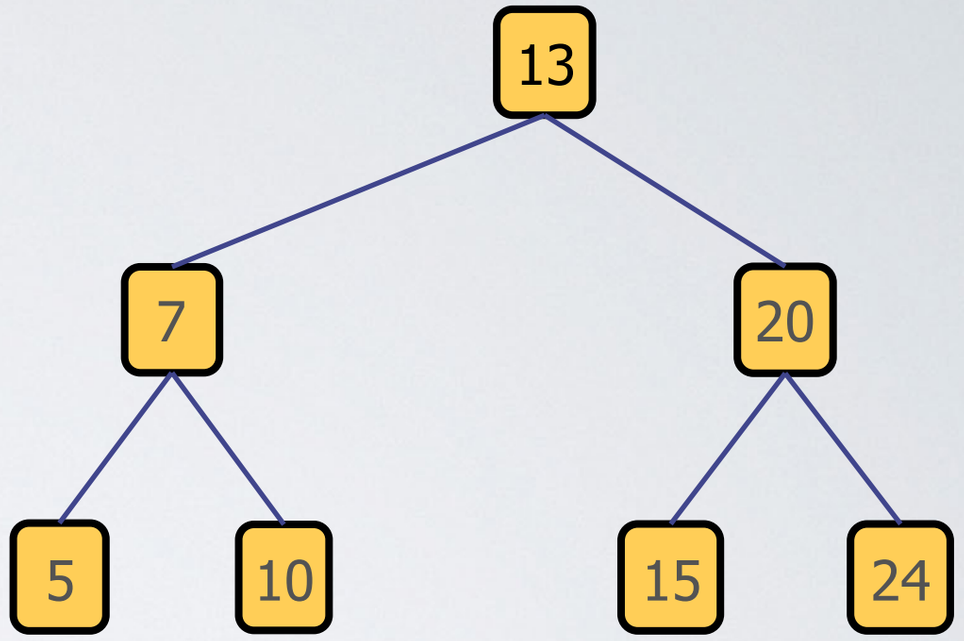
Binary Search Tree — Remove( )

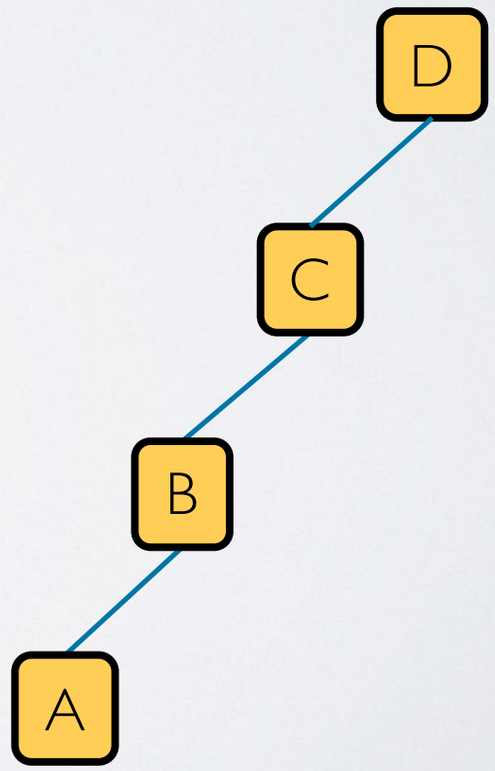
Activity #1

0 min

Binary Search Tree Analysis

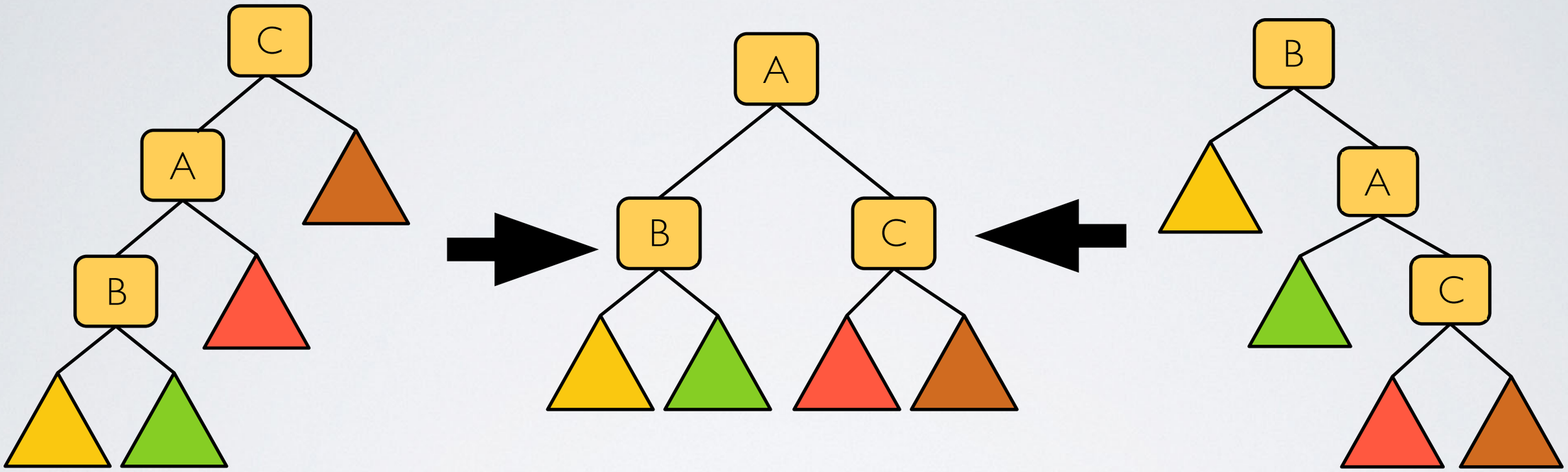
- How fast are BST operations?
  - Given a tree, what is the worst-case node to find/remove?
- What is the best-case tree?
  - a balanced tree
- What is the worst-case tree?
  - a completely unbalanced tree





Binary Search Trees — Rotations

- We can re-balance unbalanced trees w/ tree rotations



- In-order traversal of all 3 trees is

- so BST order is preserved

Beyond CS16, But good to know