Recursive Data Structures: Trees

-- Binary Search Trees --

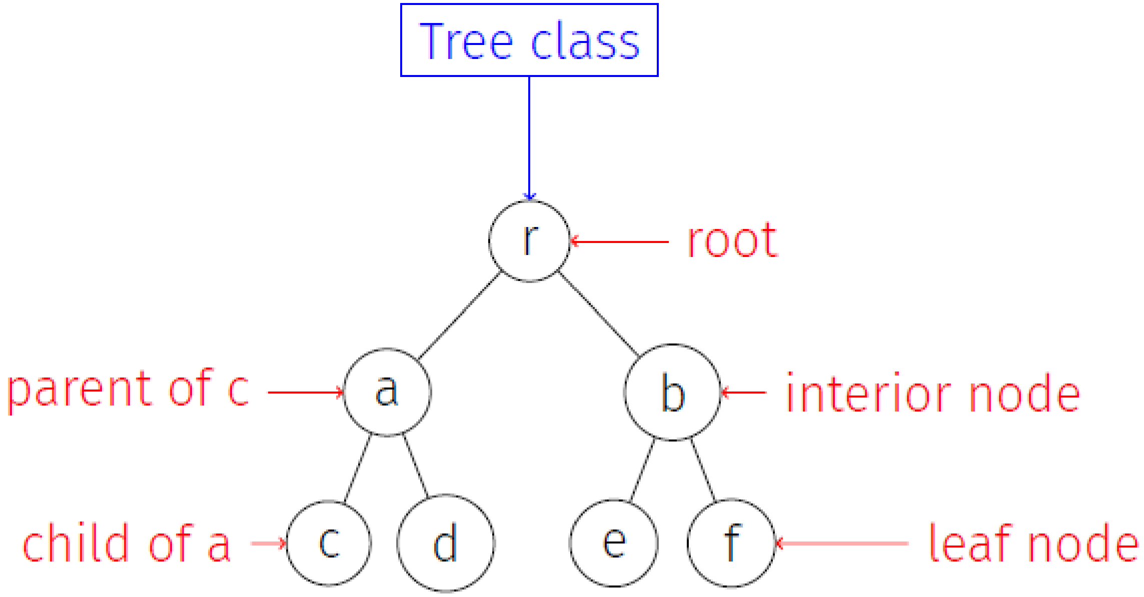
UNIVERSITY OF VIRGINIA
DATA SCIENCE INSTITUTE

Binary Search Trees

- It’s a binary tree, but created in a specific way for the purpose of searching

Trees

- Reminder…



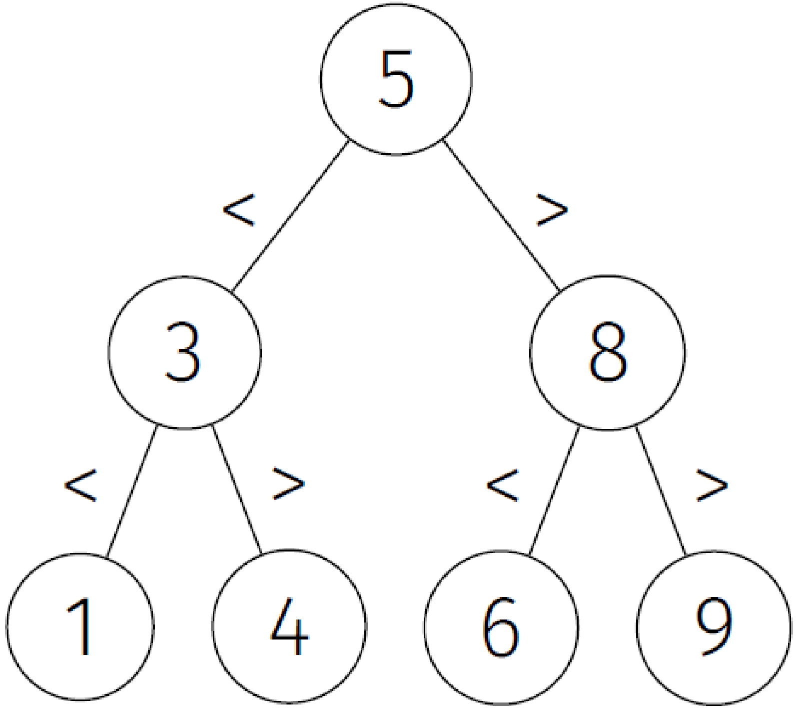
Binary Search Trees: Motivation

- It would be nice to find/search for items quickly
  - Want a fast look up time
  - Want to handle inserts and deletes into list
  - Idea: store items in sorted order
- Lists, like `ArrayList` aren't ideal
  - If not sorted: O(n) lookup (Linear search)
  - If can make use of Binary Search: O(log n) lookup
    - Must pay O(n log n) to sort beforehand
    - If we insert or remove items, **sort** may become invalid!

Is there a way to combine what we've been talking about to get the best of both worlds?

Binary Search Trees

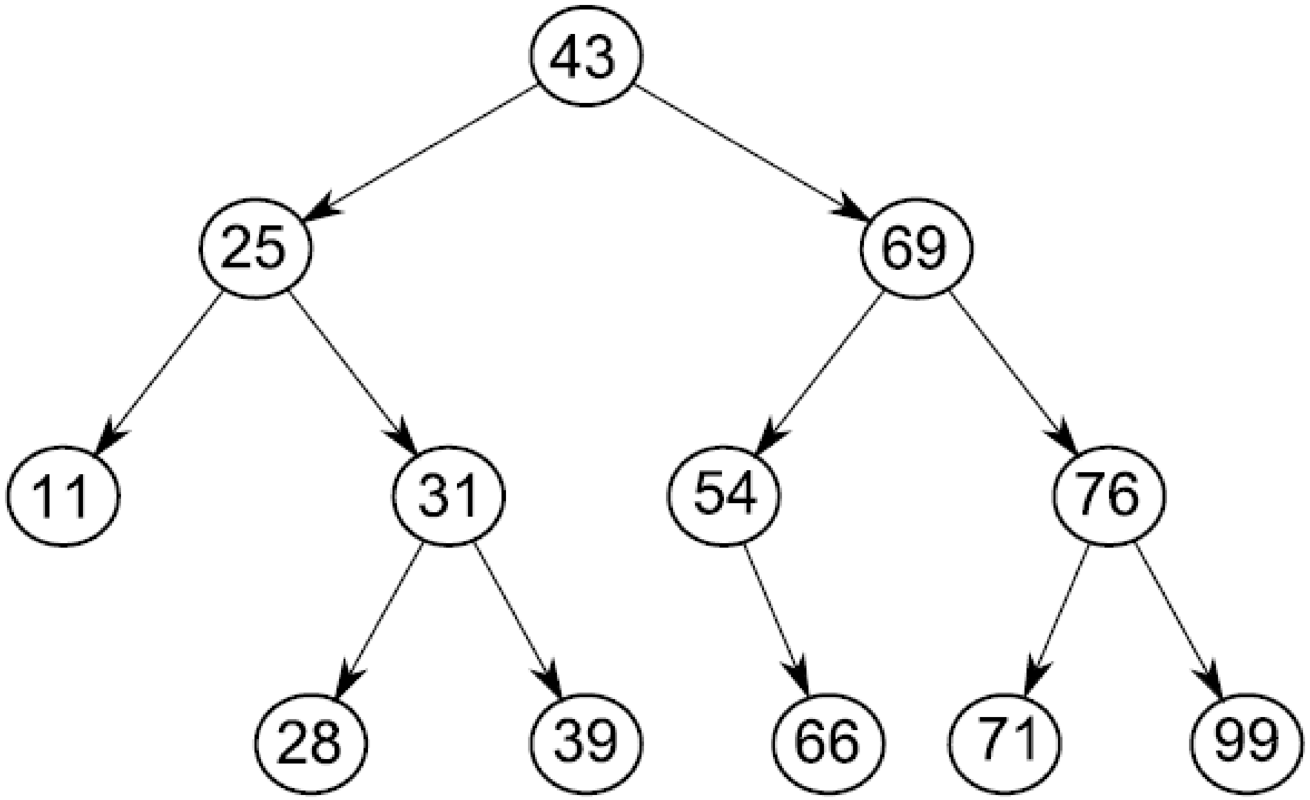
- Binary tree with **comparable** key values
- **Binary search tree property:**
  - every node in the **left** subtree has key whose value is **less** than the value of the root's key value, and
  - every node in the **right** subtree has key whose value is **greater** than the value of the root's key value.



Binary Search Trees: Cool Property

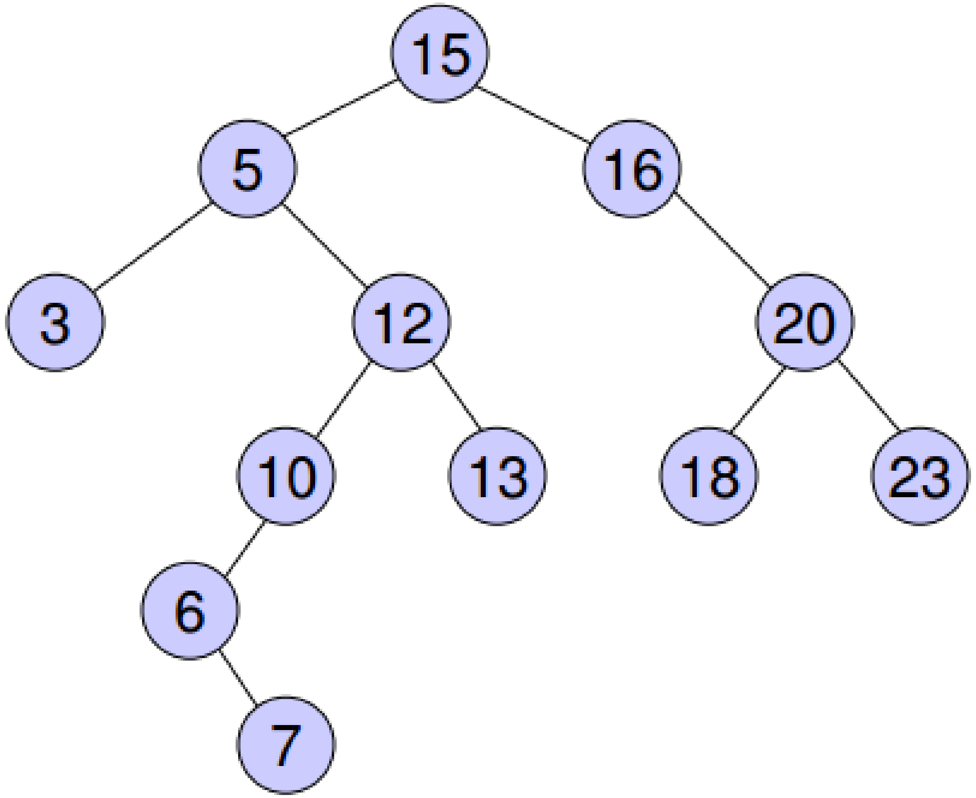
- How could we traverse a BST so that the nodes are visited in **sorted** order?
  - *In-order* traversal: left tree, node, right tree
- It's a very useful property about BSTs
- Consider Java's **TreeSet** and **TreeMap**
  - Built using search trees (not a BST, but one of its better "cousins")
  - Guarantee: search times are **O(lg n)**

Example of a Binary Search Tree



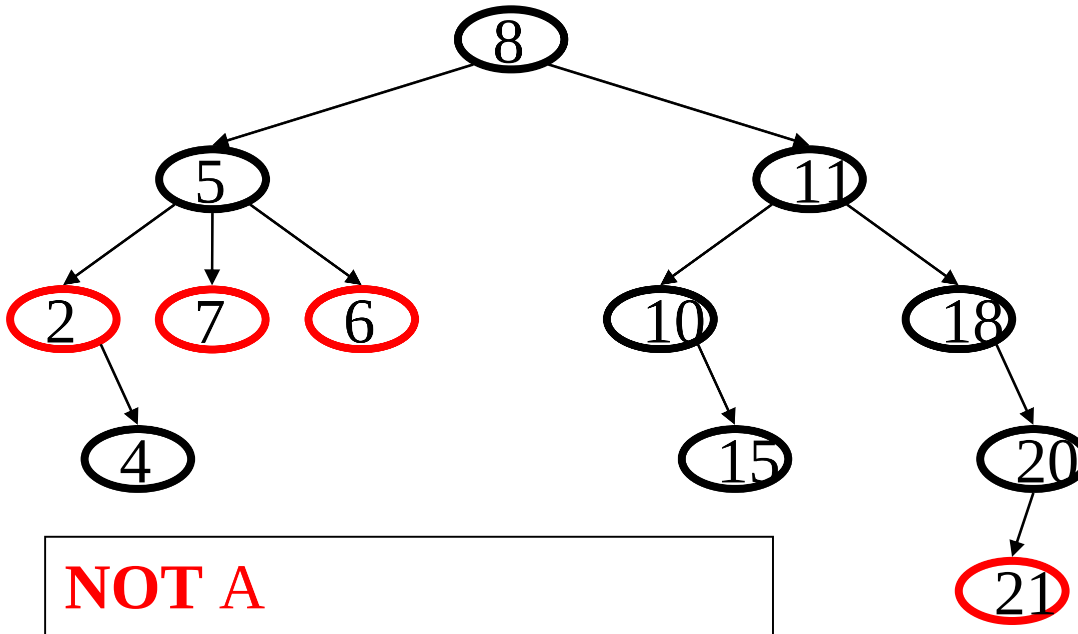
Example of a Binary Search Tree

(we've seen this before!)



Counterexample

Three child nodes

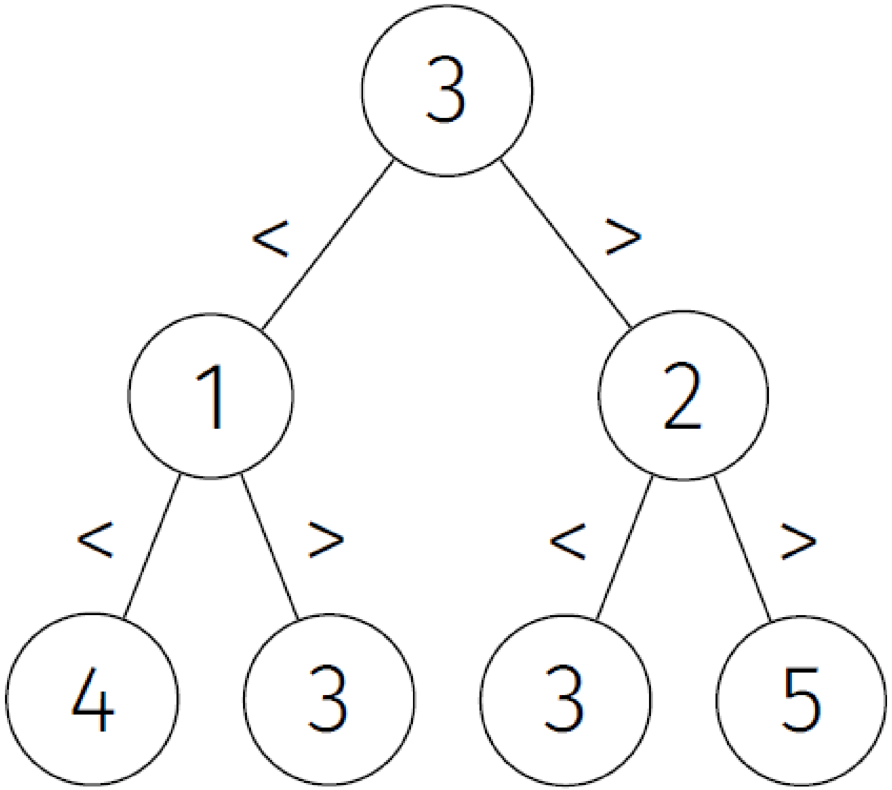


NOT A BINARY SEARCH TREE

Left node not less than root

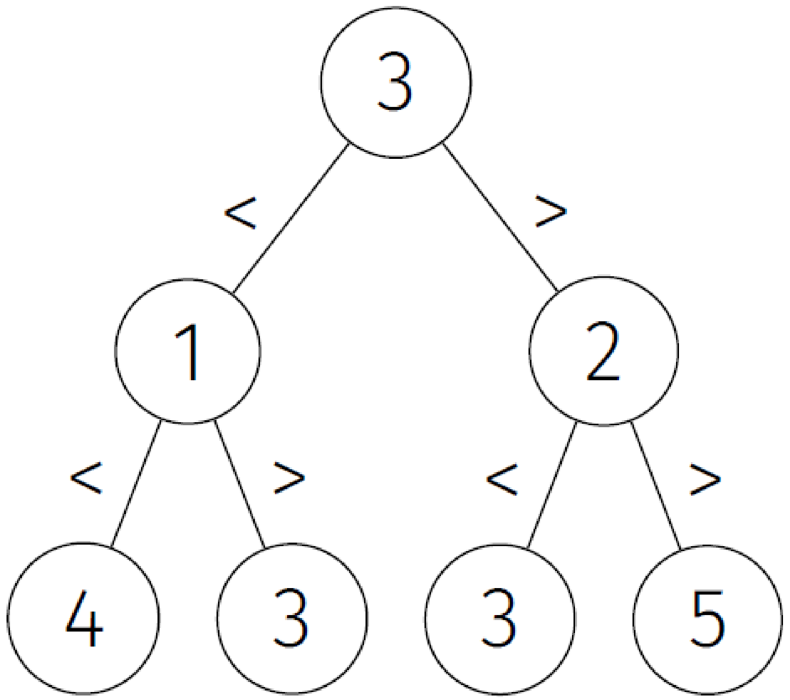
Question!

- Is this a binary search tree?



Question!

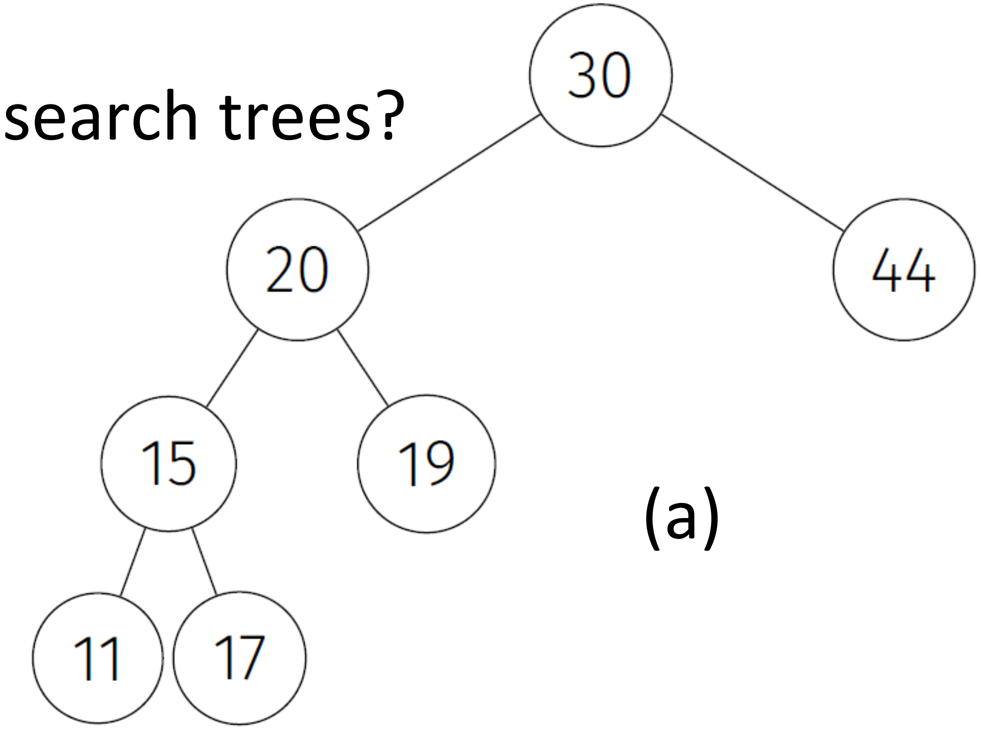
- Is this a binary search tree?



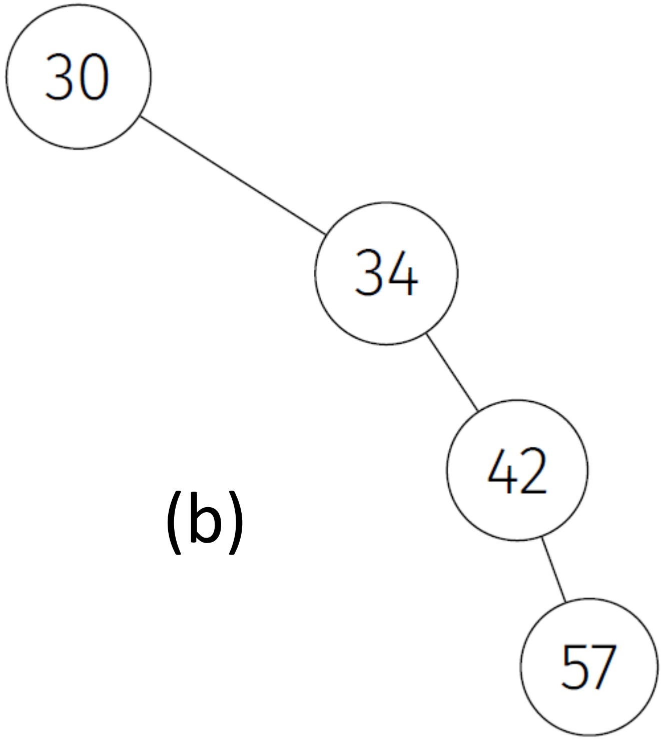
No! Binary search tree property not preserved

Question!

- Are these binary search trees?



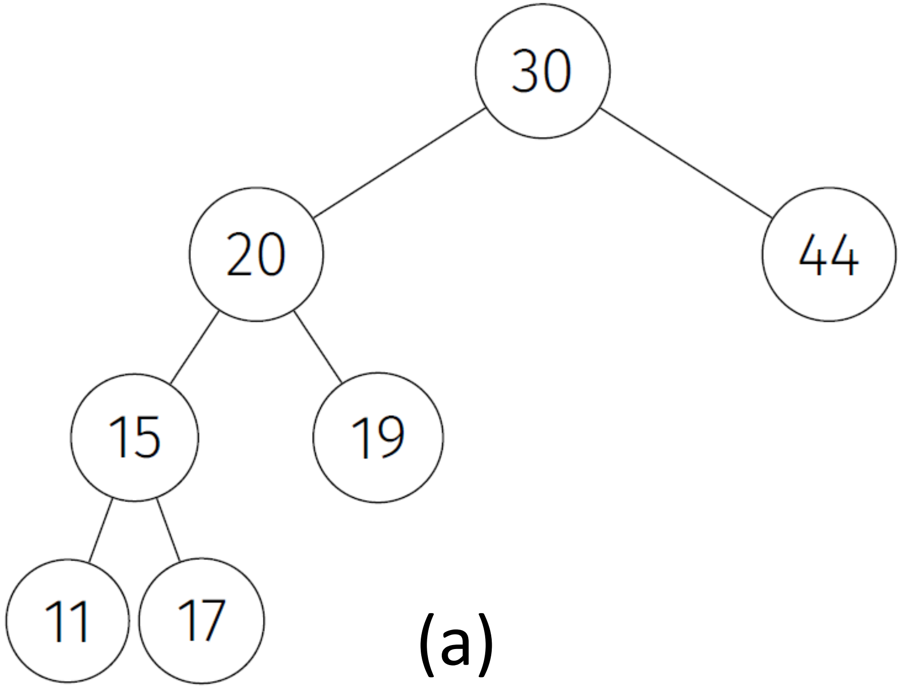
(a)



(b)

Question!

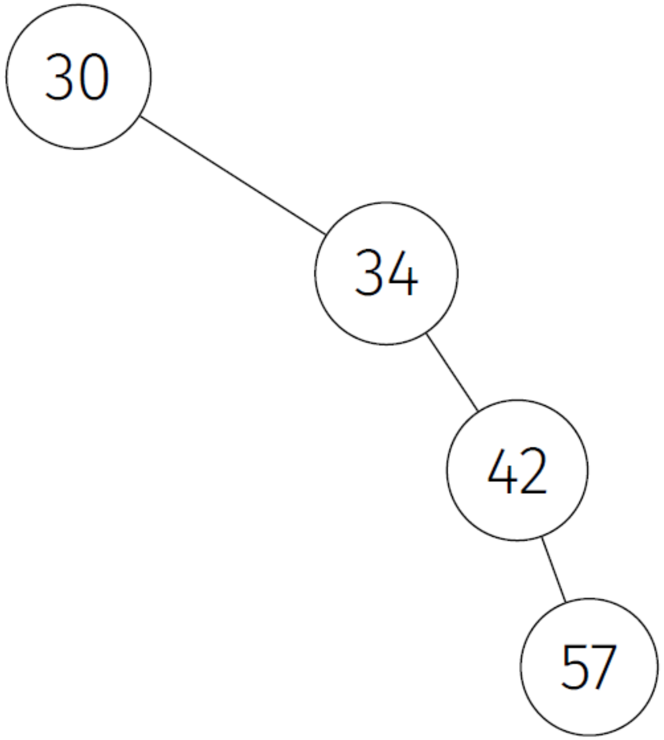
- Are these binary search trees? ***No!*** Binary search tree property not preserved



(a)

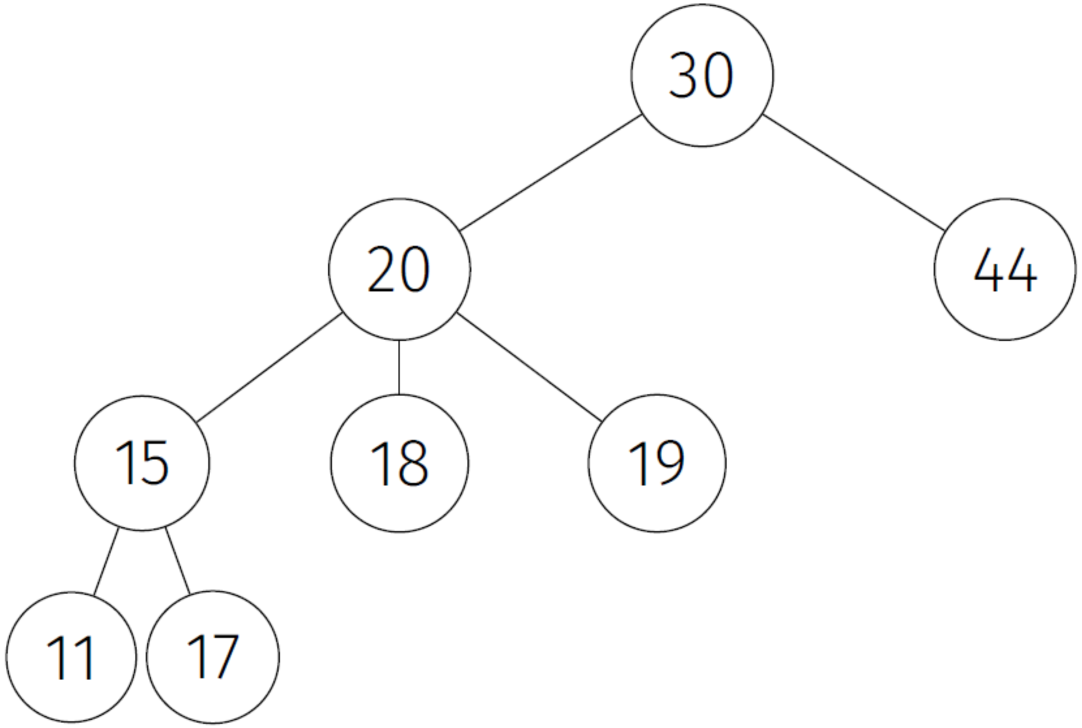
Question!

- Are these binary search trees? ***Yes!***
- However, this tree is unbalanced!
  - O(n) to find 57!
  - This is an ordered list
- A **balanced** binary tree
  - Guarantees height of child subtrees differ by no more than 1
  - Is better! Produces O(log n) runtimes



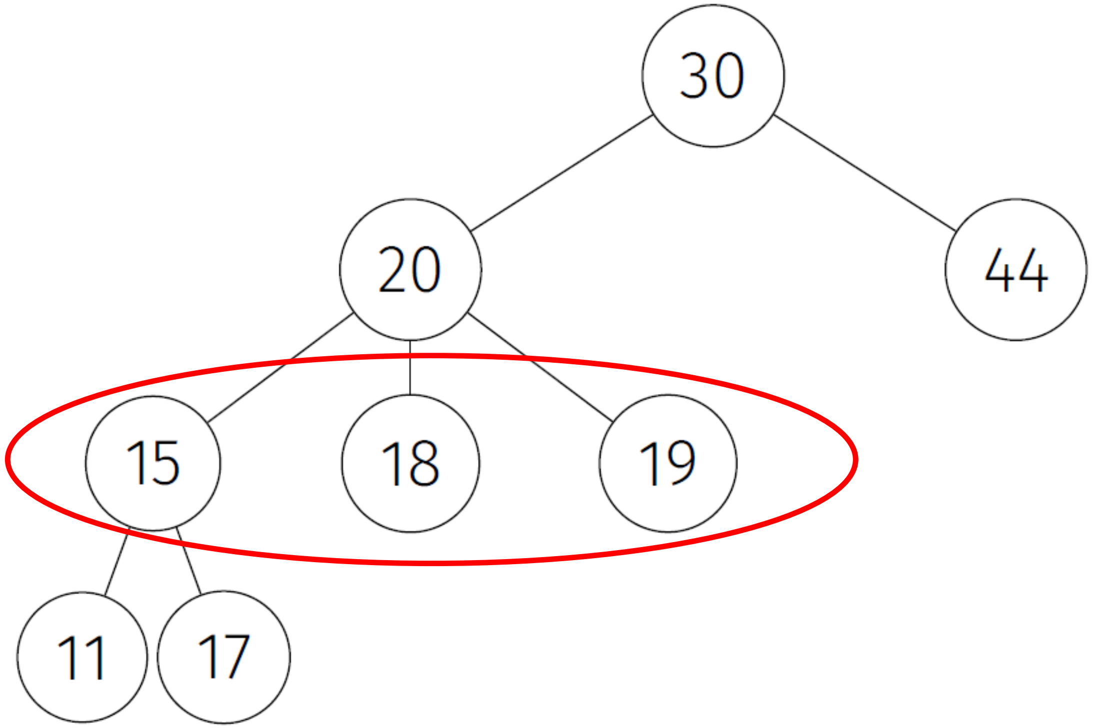
Question!

- Is this a binary search tree?



Question!

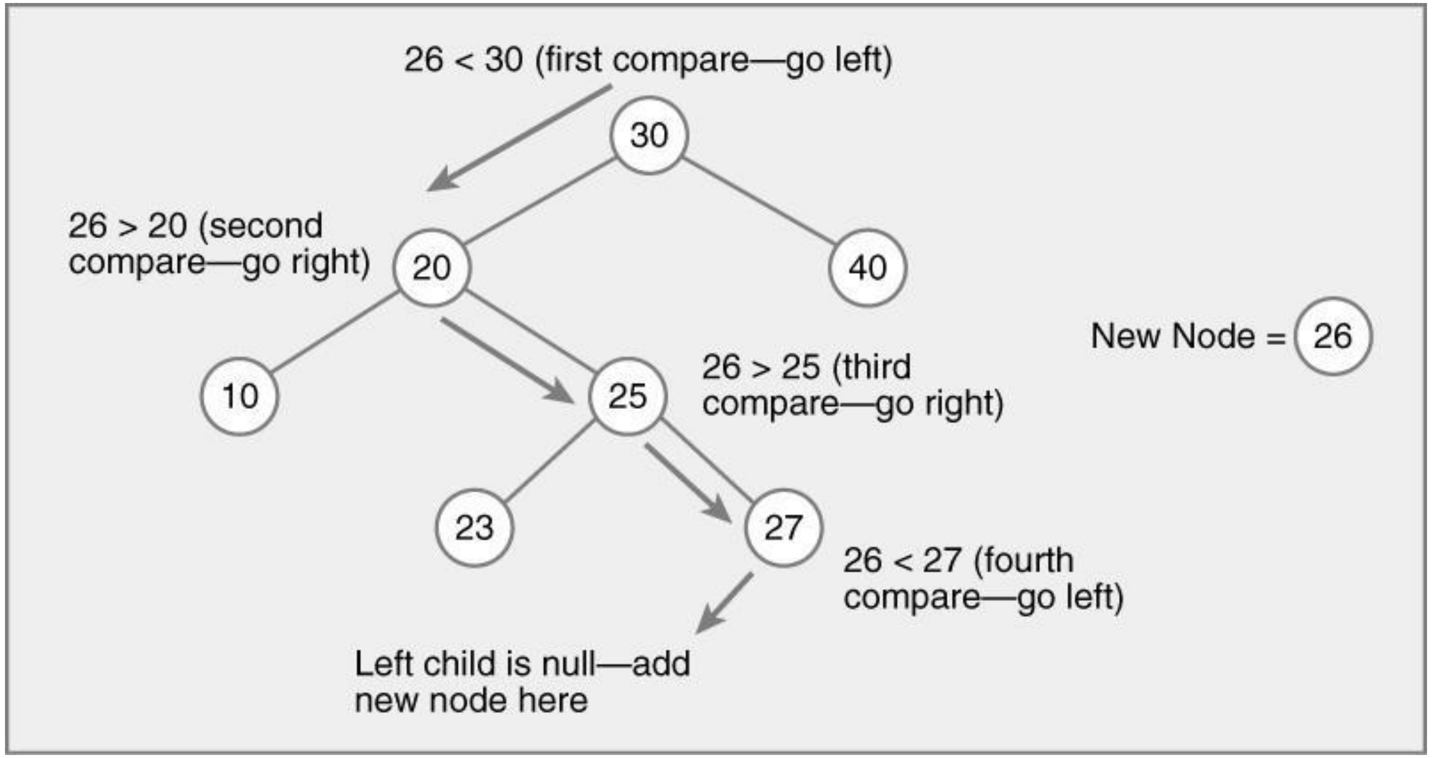
- Is this a binary search tree?



No! It is not even a binary tree!

Find and **Insert** in BST

- **Find**: look for where it should be
- If not there, that's where you **insert**

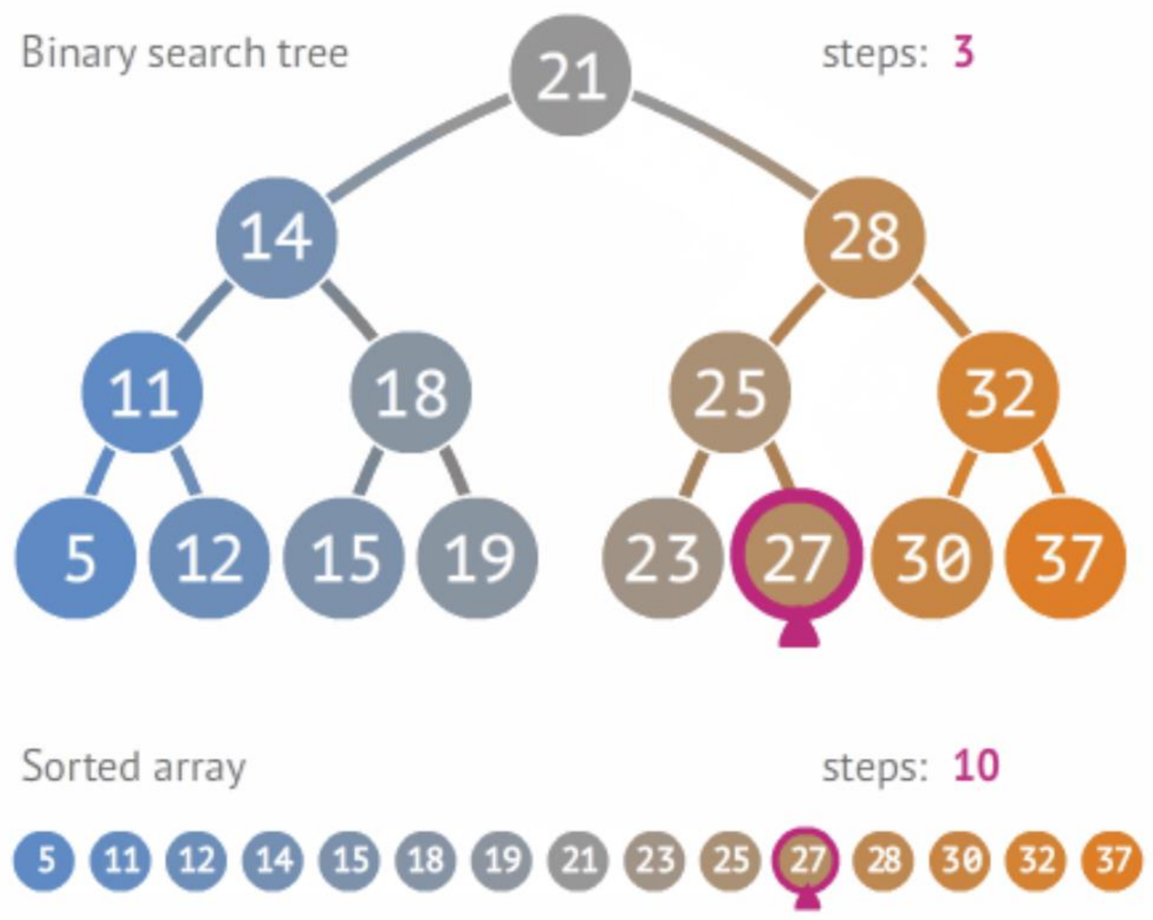


BST Find and Insert

- Find an element in the tree
  - Compare with root, if less traverse left, else traverse right; repeat
  - Stops when found or at a leaf
  - Sounds like **binary search**!
  - Time complexity: **O(log n)**, worst case height of the tree
- Insert a new element into the tree
  - Easy! Do a **find** operation. At the leaf node, add it!
  - Remember: add it to the correct side (left or right)

Binary Search Tree vs Array

- Can find an element much quicker using a BST



Source: penjee.com

Tree Operation: Find

find method: **pseudo-code** [returning True/False] (Python-like)

```
find(self, target):
    while current node exists (isn’t ‘None’):
        if target matches current.data   // found it!
            return True
        else if target > current.data
            current = current.right   // go right
        else
            current = current.left    // go left
    # no next node (either left or right)
    return False   # didn’t find target, so return False
```

Tree Operation: Find

- Recursive code for tree operations is simple, natural, elegant [returning true/false]
- find method: **pseudo-code** (Java-like)

```
boolean find(Comparable target) { //find target
      Node next = null;
      if (this.data matches target) //found it!
            return true
      else if (target's data < this.data)
            next = this.leftChild //look left
      else
            next = this.rightChild //look right
      // 'next' points to left or right subtree
      if (next == null ) return false // no subtree
      return next.find(target) // search on
  }
```

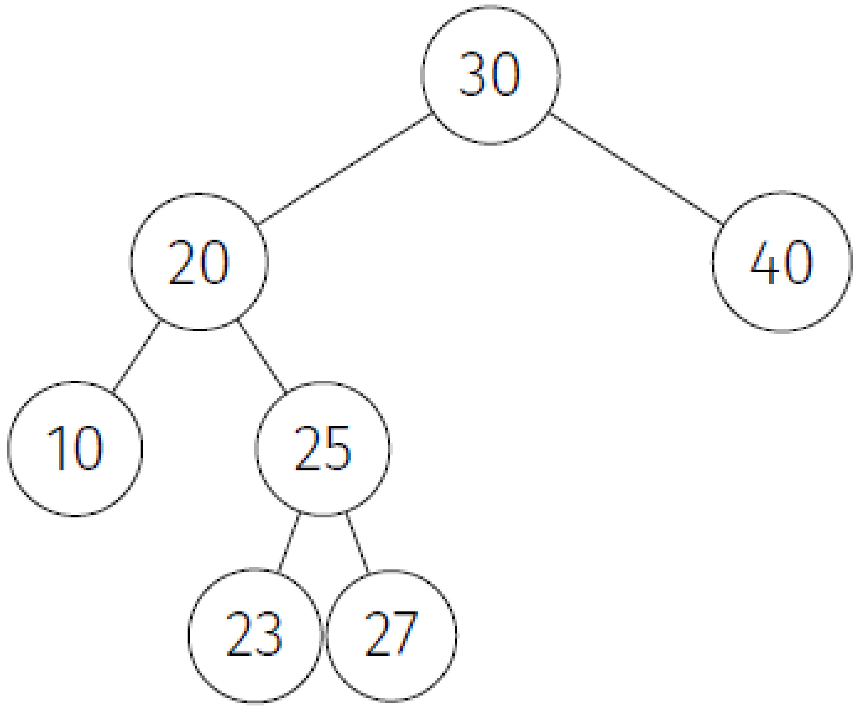
Find and **Insert**

- Where do we insert a new element?

  - Run `find()` method to determine where the element *should have been*
  - Add the new node at that position

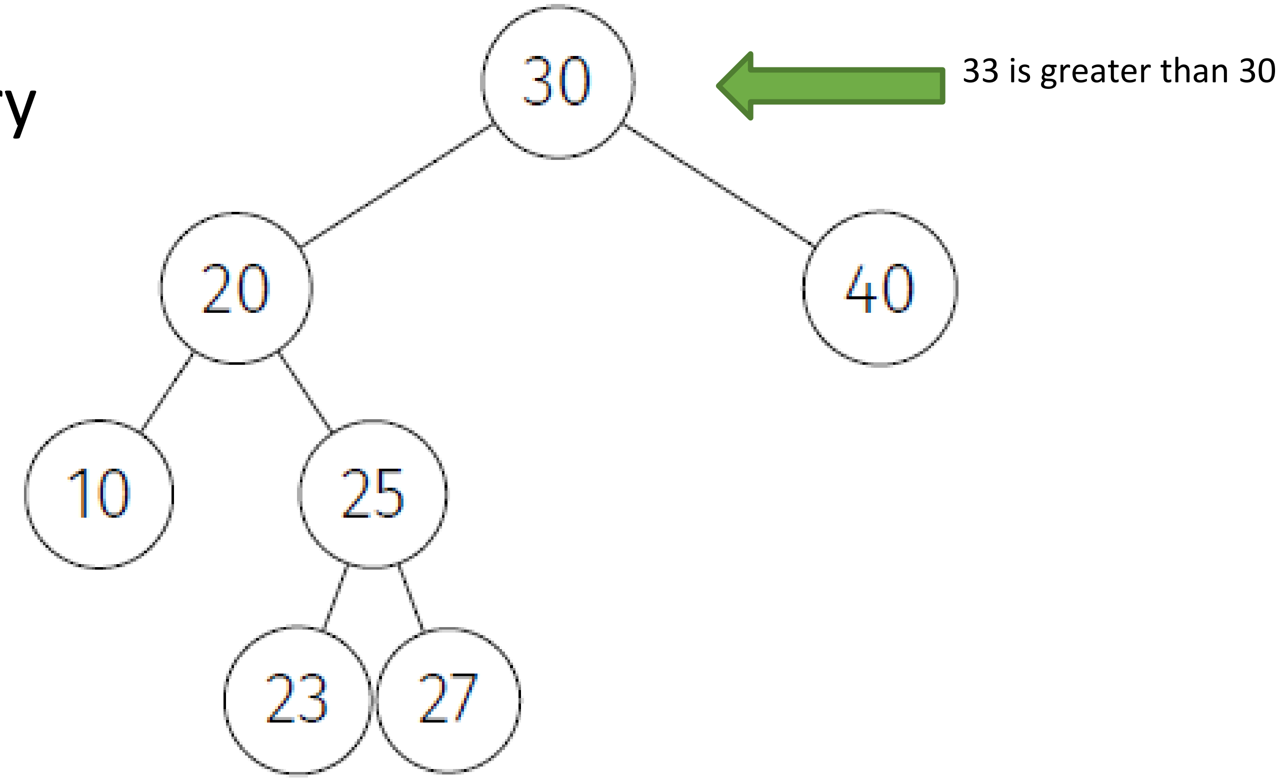
Insert Example

- Insert 33 into the following binary search tree



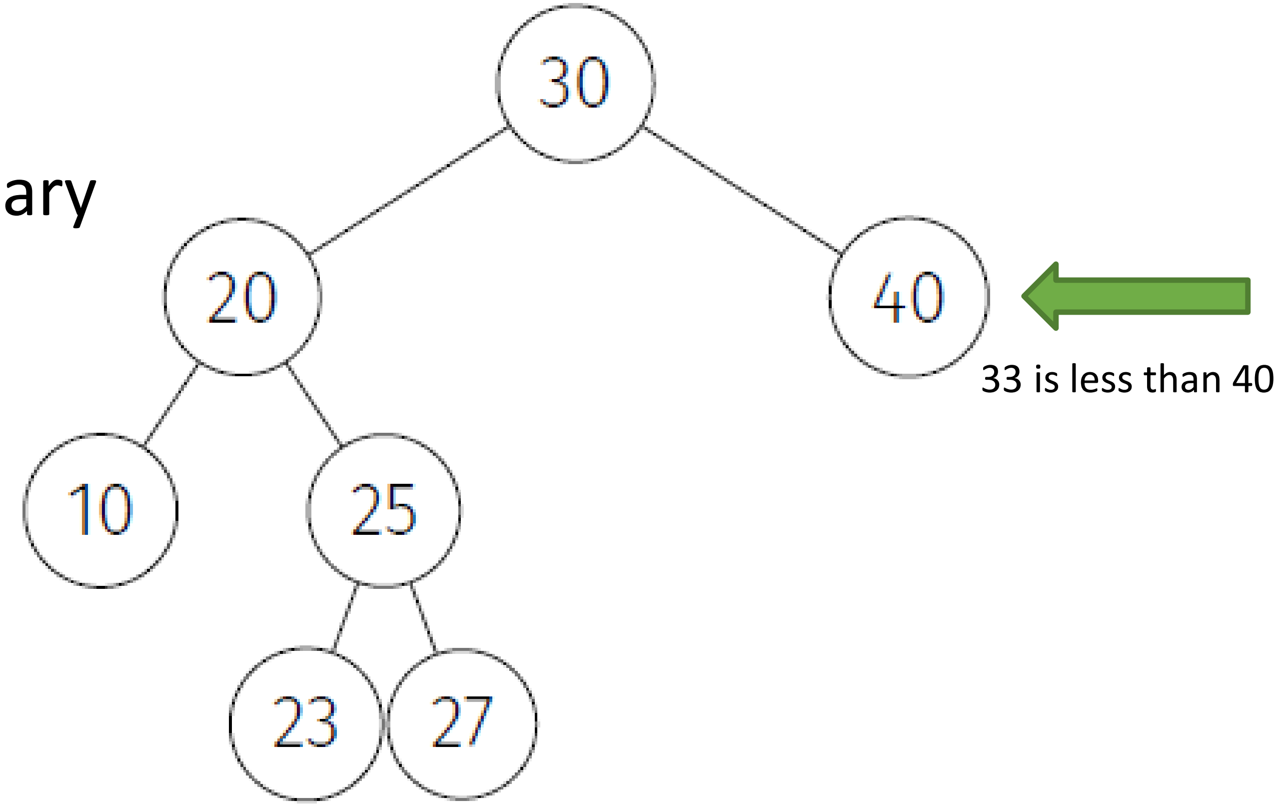
Insert Example

- Insert 33 into the following binary search tree



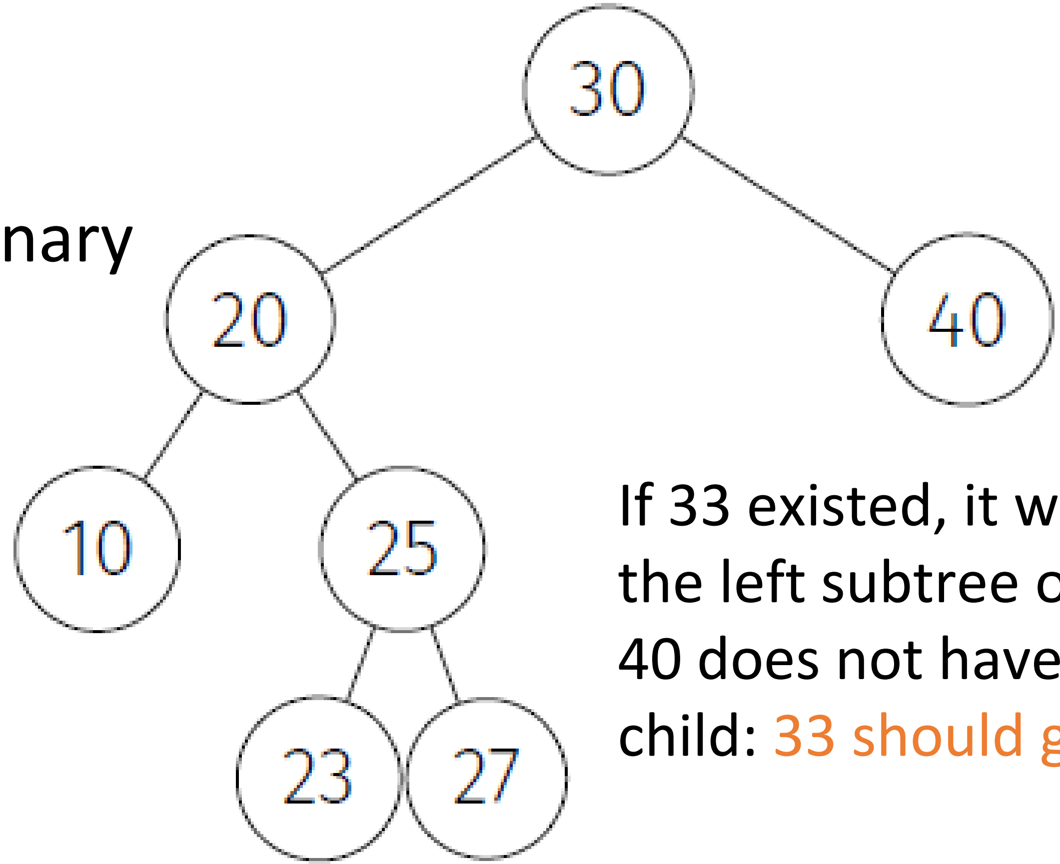
Insert Example

- Insert 33 into the following binary search tree



Insert Example

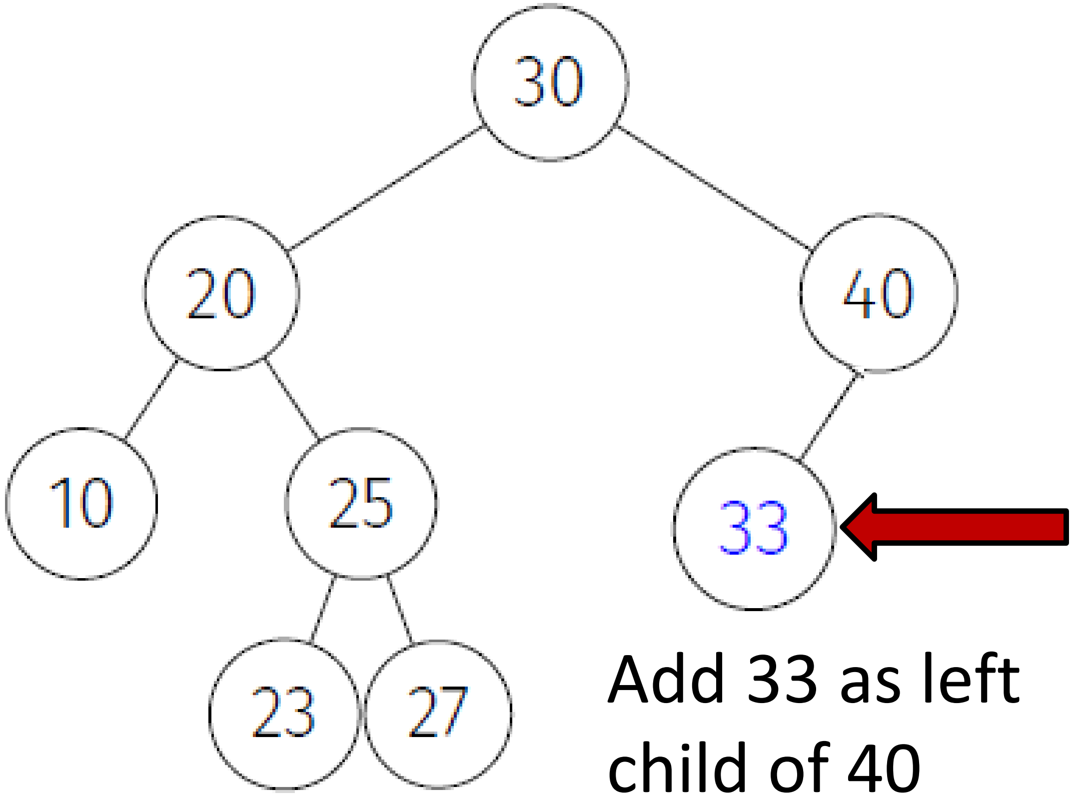
- Insert 33 into the following binary search tree



If 33 existed, it would be in the left subtree of 40. But 40 does not have a left child: 33 should go here!

Insert Example

- Insert 33 into the following binary search tree



Add 33 as left child of 40

Activity –
Write BST "find" method in Python

```
def find(self, target):   # find method: looking for target
"""Return the node for key t if it is in this tree, or None otherwise."""
```

Possible call could be:

myTree.find(t)   or

self.root.find(t) *# call **find()** and search for the target **'t'***

Work with a partner to code a solution to this method in Python.

Deleting from a BST

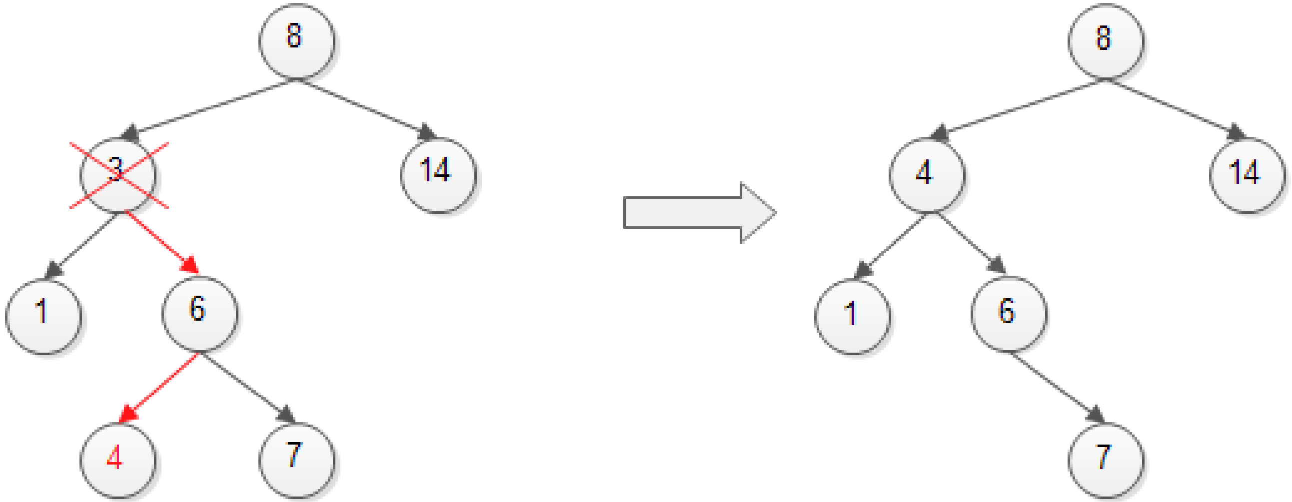
- **Delete** a node from the tree
  - More complicated – we need to select a node as replacement!
- **Removing** a node requires
  - Moving its left and right subtrees
  - If 0 children: delete node
  - If 1 child: replace node with its only child
  - If 2 children: find next largest (or smallest) to fill in
- Answer: not too tough, we'll go over the idea of how to remove nodes next

Deleting from a BST (finding a successor)

- After removing an element from a BST, you have to find a node with which to replace it (it's "Successor")
- Where to find the successor? Well, there are 2 options:
  - The next "largest" element
  - The next "smallest" element
- Where would these exist in the BST?
  - Next largest: in right sub-tree – but where in the sub-tree?
  - Next smallest: in left sub-tree – but where in the sub-tree?

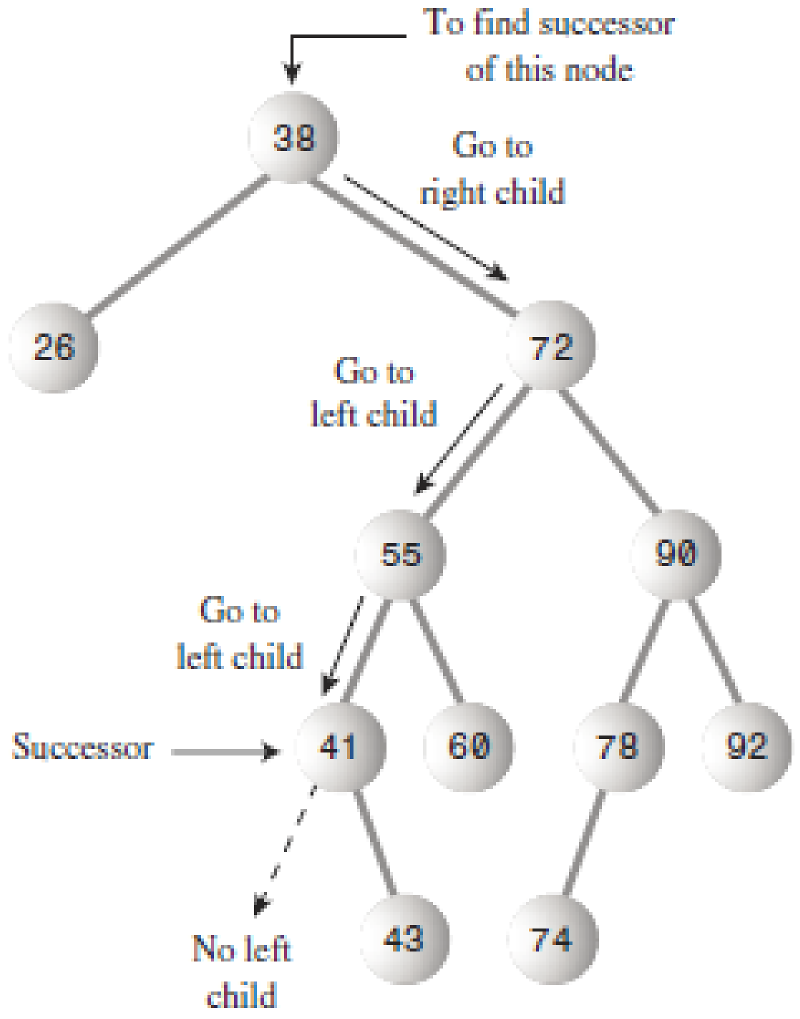
Deleting Quick Overview

- Find successor (of 3) in it's **right** subtree (i.e. node **4**) – finding the **minimum** (*leftmost node*) of right subtree



Find Successor of 38

- *Minimum* of **right** subtree (*leftmost node*)
- Which is the <u>next largest</u> number

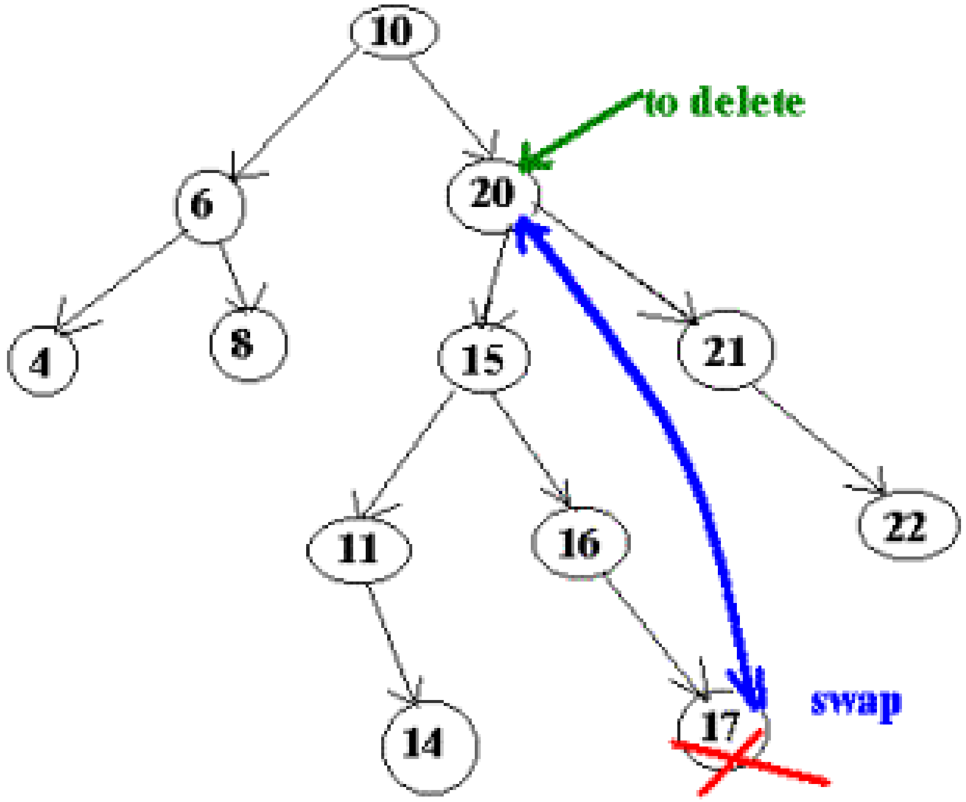


Finding the successor.

Another Example – successor of 20 [1]

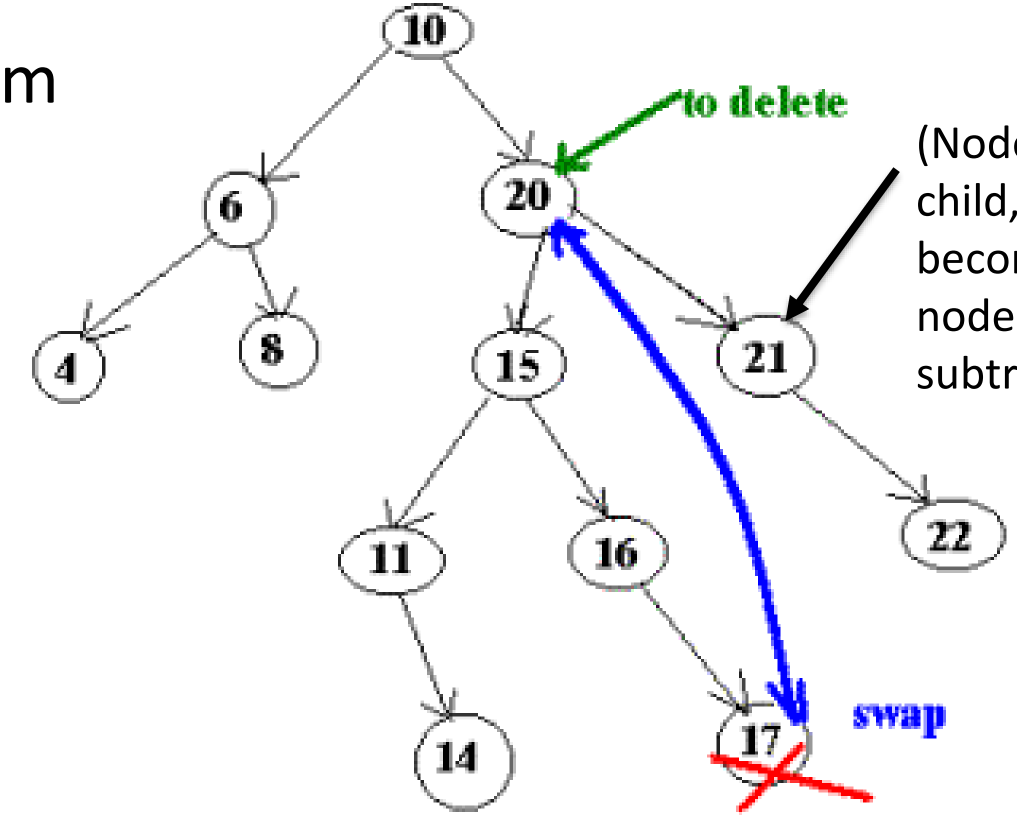
- Largest number –*rightmost*– of **left** subtree (largest number out of left subtree that contains values *smaller* than that node) ➔ node 17

- Which is the next smallest number



Another Example – successor of 20 [2]

- Next largest number (minimum number –*leftmost*– of **right** subtree) ➔ node 21



(Node "21" has no left child, therefore it becomes the leftmost node in the **right** subtree.)

Create a Binary Search Tree

List 1: Leo, Hester, Ressie, Keira, Damian, Victor, Collin, Marci, Ashlie, Willis, Eric, Mya, Elizabeth, Ralph

1. List out the tree created by the add order of **list 1** using **post-order traversal**.
2. If we **removed** the node containing **Damian**, what <u>two</u> values could we replace it with?
3. What if we **removed** the node containing **Ressie**?

List 2: Victor, Ralph, Leo, Mya, Eric, Elizabeth, Hester, Damian, Willis, Collin, Keira, Marci, Ashlie, Ressie

4. List out the tree created by the add order of **list 2** using **post-order traversal**.
5. Compare the tree from **list 1** with the tree from **list 2**, *which do you think would <u>preform better</u> for the add and remove methods?*

In-Class Activity – Binary Search Trees

For each of the following lists (as added to a BST), build a binary search tree. (Comparisons are alphabetical).

- List 1: Leo, Hester, Ressie, Keira, Damian, Victor, Collin, Marci, Ashlie, Willis, Eric, Mya, Elizabeth, Ralph
  1. List out the tree using **post-order traversal**
  2. If we removed **Damian**, what two nodes could replace it?
  3. If we removed **Ressie**, what two nodes could replace it?
- List 2: Victor, Ralph, Leo, Mya, Eric, Elizabeth, Hester, Damian, Willis, Collin, Keira, Marci, Ashlie, Ressie
  4. List out the tree using **post-order traversal**
  5. Compare the two trees. Which do you think would preform better for the add and remove methods?

Submit your answers to Collab.