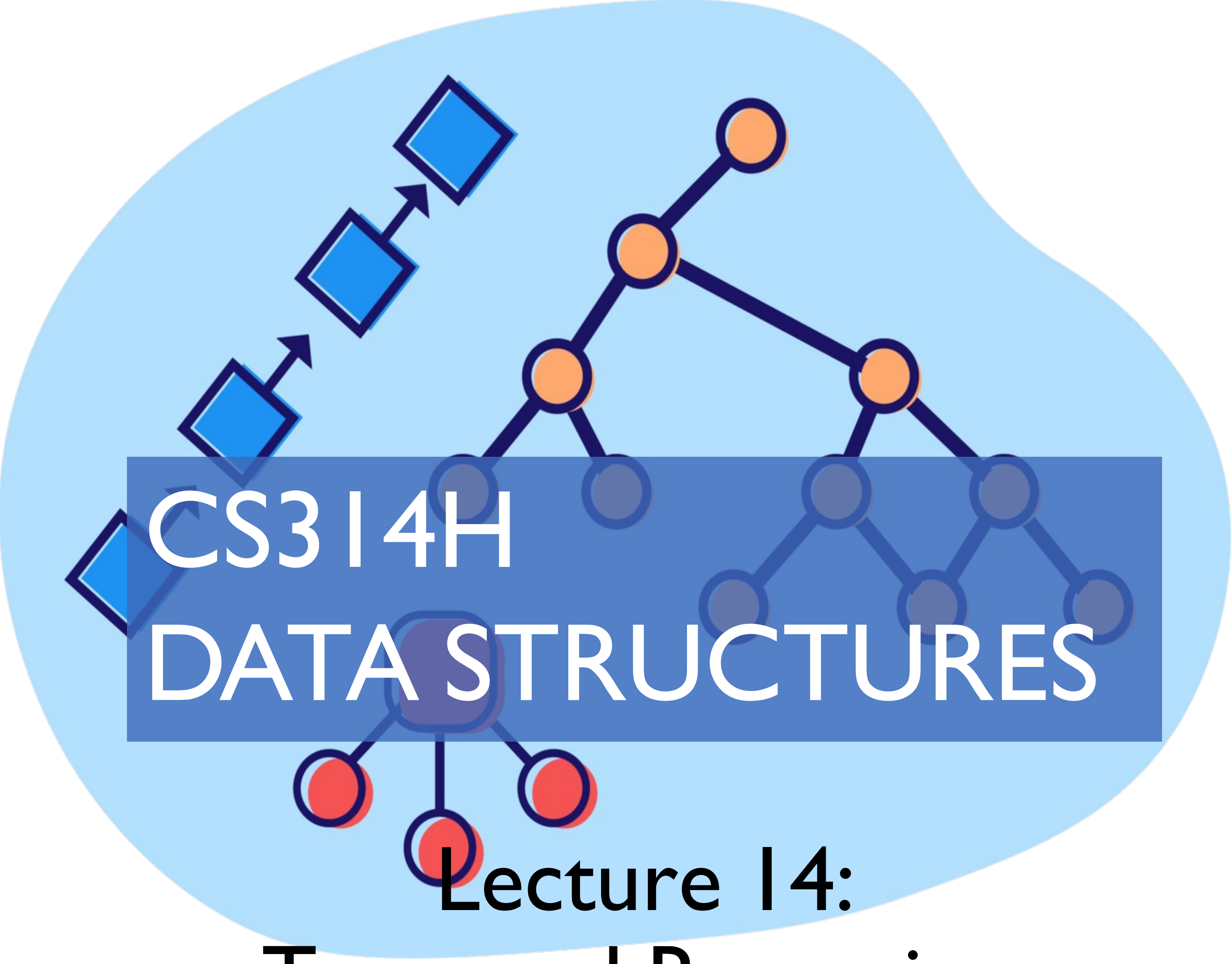


CS314H DATA STRUCTURES

Lecture 14: Trees and Recursion

Mikyung Han

Please, interrupt and ask questions AT ANY TIME !

Reminders

- Read Ch 18, Ch 19.1 – 19.4
- CritterFest Oct 5:30 PM – 6:30 PM @ JGB 2.218
  - Extra credit towards Exam 1

Outline

1. Administrative
2. Trees Intro

Trees

- What is a tree?
- Why trees could be useful?

Tree Definition 1

A connected collection of n nodes connected by n - 1 undirected edges

Does it allow cycle?

Tree Definition 2

A tree is a root connected to 0 or more subtrees that do not include the root.

Tree Definition 3 (Our definition)

A set of nodes

1. one node is distinguished as root r
2. Every node c except r has exactly one parent node p connected by an edge
3. A unique path exists from r to each node.

- How does this definition include Tree Definition 1?
  - N nodes connected by n-1 edges?
- Could there be cycle?

More definitions

- Root: A distinguished node that has no parent
- Leaf node/external node: A node with no children
- Internal nodes: non-leaf nodes
- Parent: If node c is reachable via node p, p is the parent, c is the child
- Siblings: The nodes nodes that have the same parent
- Ancestors/Descendants
- Directed vs undirected edges
- Depth of a node: the length of the path from the root to that node
- Height of a node: the length of the path from that node to is deepest descendent (must be a leaf)

What is Binary Tree?

- Tree whose node has at most 2 children
- Why binary tree is useful?
- It is versatile: We can encode ANY tree into a binary tree

Recap: Abstract Data Type (ADT) vs. Data Structure

- **ADT defines a set of values and legal operations on the values.**
  - A circular queue, the next element after the last element is the first element
  - A binary tree has left/right child

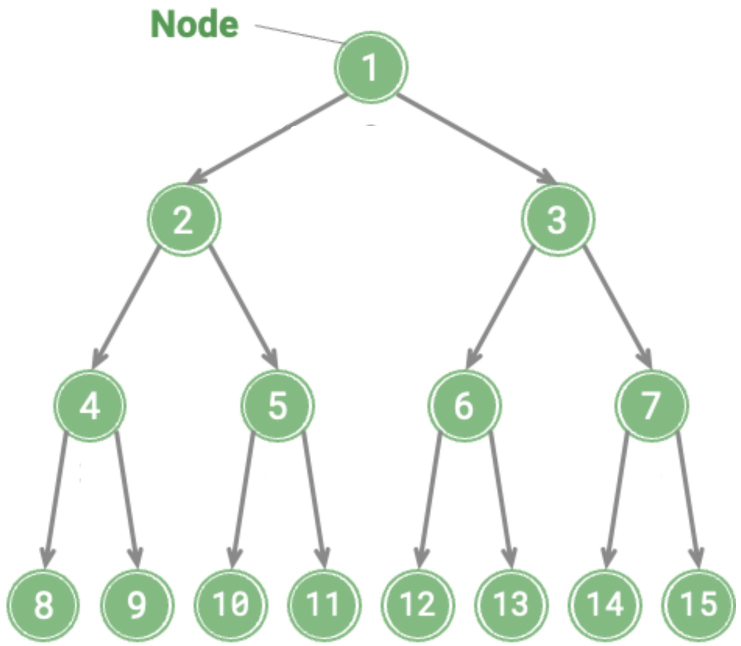
- **Data structure is an implementation of ADT**
  - A circular queue can be implemented with an array
  - A circular queue can be implemented with a linked list
  - A binary tree can be implemented with an array
  - A binary tree can be implemented with a node with references to left/right child

So, when we say "Tree" we are talking about ADT

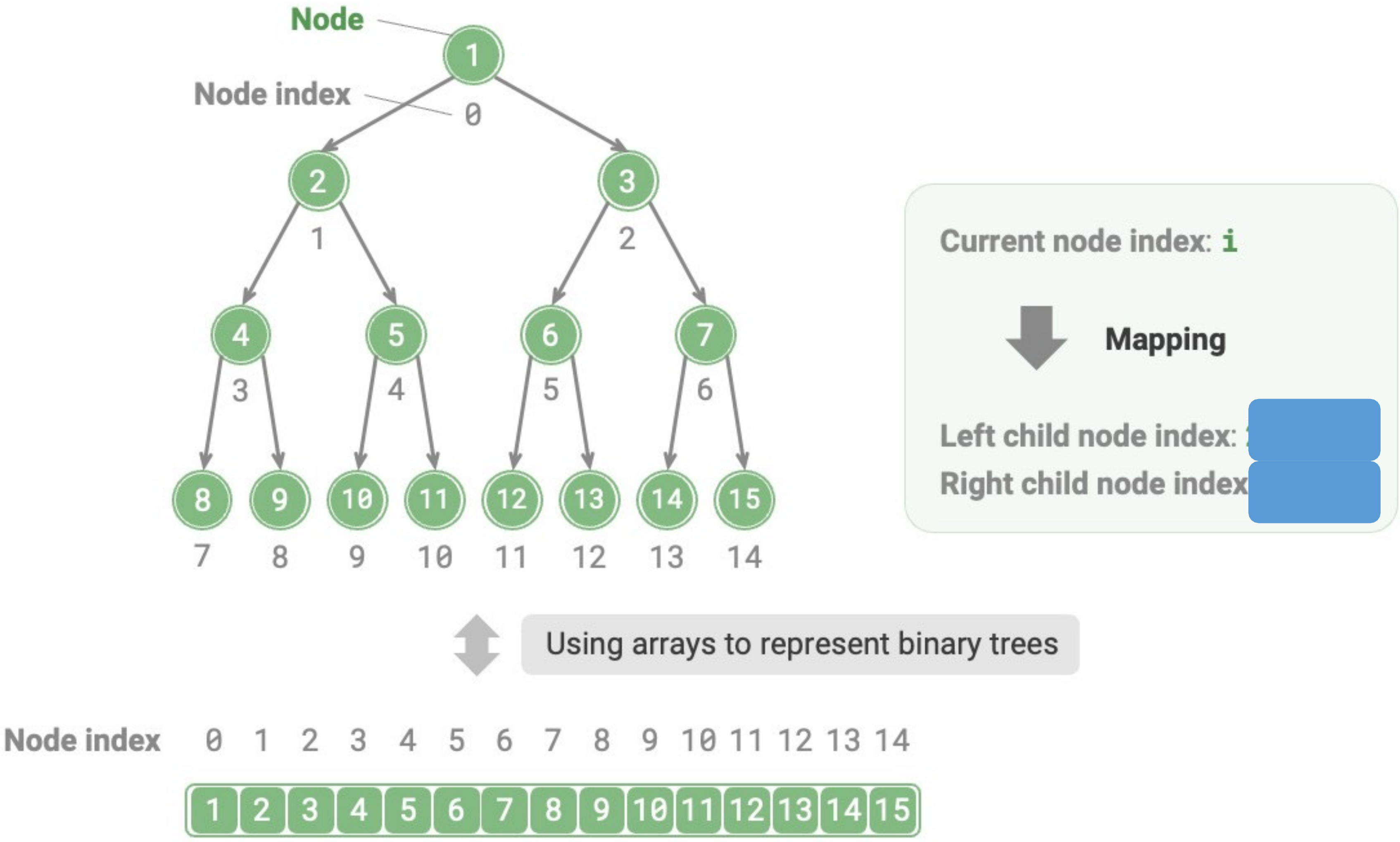
Let’s talk about data structure

- How to implement tree?

Use node with left/right reference



Use array


Node
Node index
1
0
2
1
3
2
4
3
5
4
6
5
7
6
8 9 10 11 12 13 14 15
7 8 9 10 11 12 13 14
Current node index: i
Mapping
Left child node index:
Right child node index
Using arrays to represent binary trees
Node index 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pros and Cons

- A binary tree with an array
- A binary tree with a node with references to left/right

Code

- **Using array**
  - Let's take a look!
- **Using tree node with left/right references**
  - Code it up!
  - Level order insert

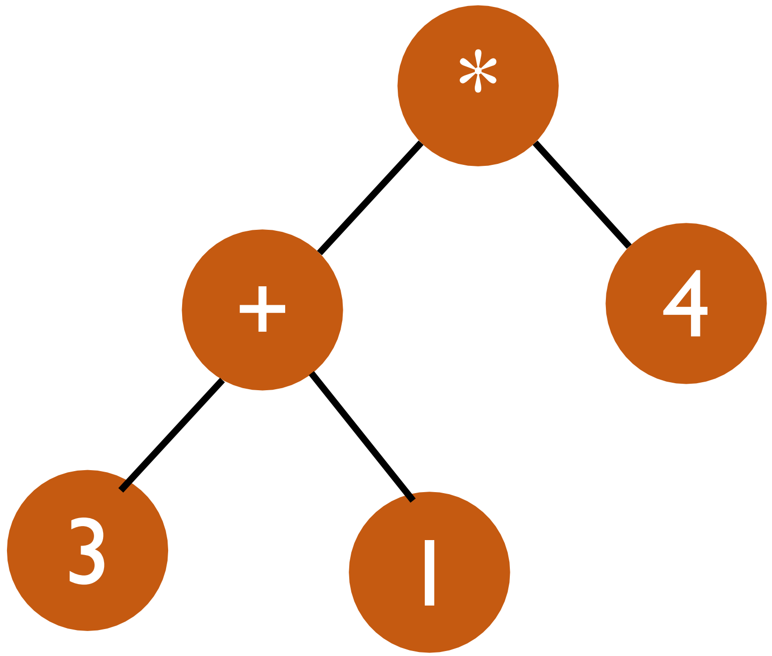
Outline

1. Administrative
2. Trees Intro
3. Binary Tree Traversal

Consider expression (3 + 1) * 4 in Java

- How should complier represent it?

1) In-order traversal
2) Pre-order traversal
3) Post-order traversal



How to implement tree traversal?

- Use recursion!

```
        10
   /          \
 20            30
   \           / \
   40         60  90
   /        / \     \
50        70 80      100
```

Pseudo code

```
preorder(node)
    if(node == null) return;
    visit(node)
    preorder(node.left)//recurse on left subtree
    preorder(node.right) //recurse on right subtree

inorder(node)
    if(node == null) return;
    inorder(node.left)//recurse on left subtree
    visit(node)
    inorder(node.right)//recurse on right subtree


postorder(node)
    if(node == null) return;
    postorder(node.left)//recurse on left subtree
    postorder(node.right)//recurse on right subtree
    visit(node)
```

Can we implement tree traversal without recursion?

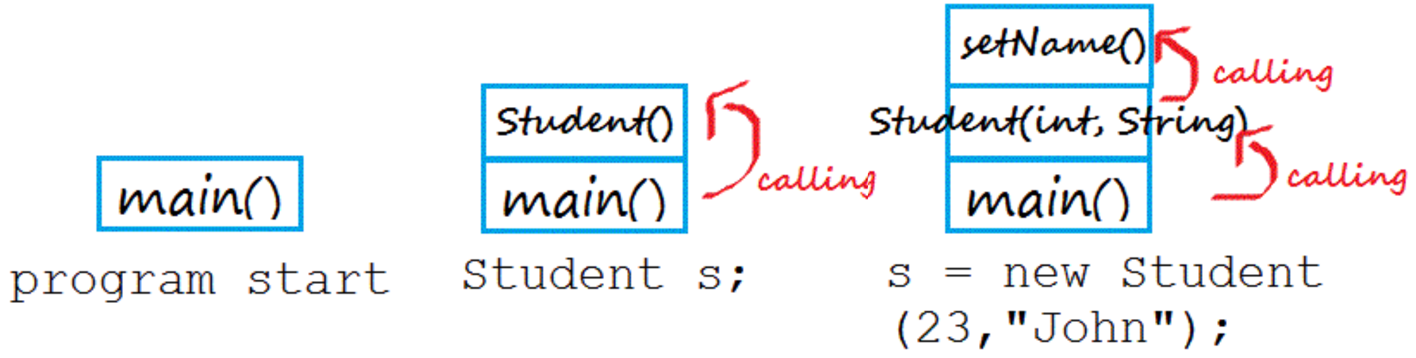
- Which data structure are we implicitly using when using recursion?
  - A call stack!
- What happens whenever you invoke a method?
  - Push the callers' local variables onto stack and jump to the called method
- What happens whenever you return from the method?
  - You pop those local variables back

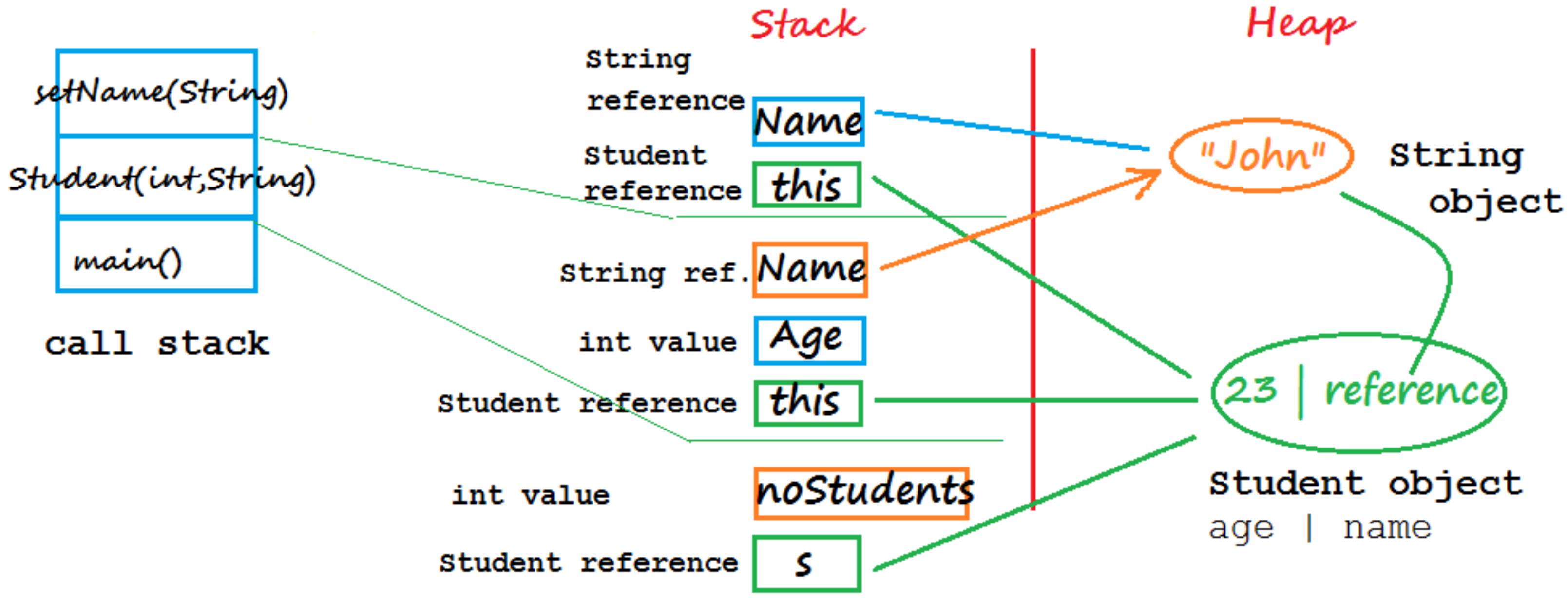
Call stack and heap

```
class Student{
    int age;                  //instance variable
    String name;     //instance variable

    public Student()
    {
        this.age = 0;
        name = "Anonymous";
    }
    public Student(int Age, String Name)
    {
        this.age = Age;
        setName(Name);
    }
    public void setName(String Name)
    {
        this.name = Name;
    }
}

public class Main{
        public static void main(String[] args) {
            Student s;                 //local variable
            s = new Student(23,"Jonh");
            int noStudents = 1;        //local variable
        }
}
```

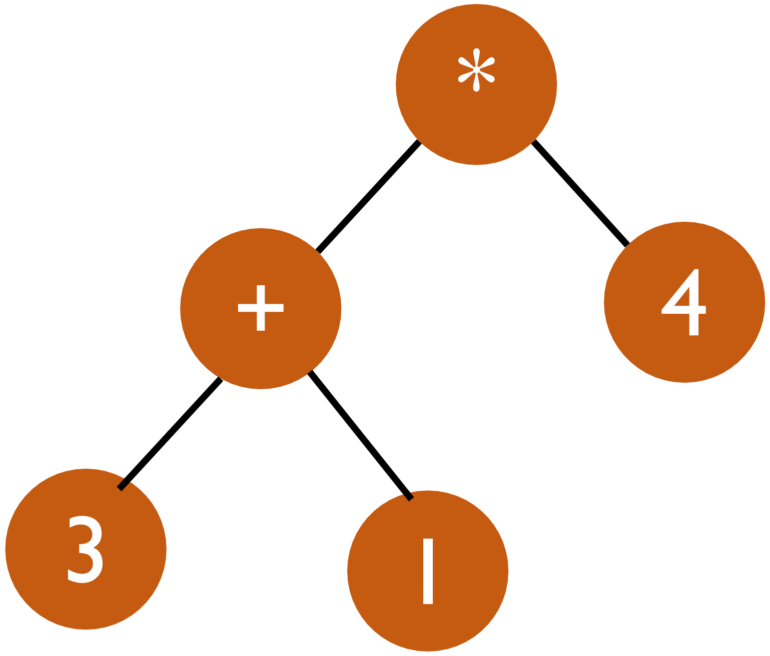




Can we convert ANY recursive implementation to non-recursive one?

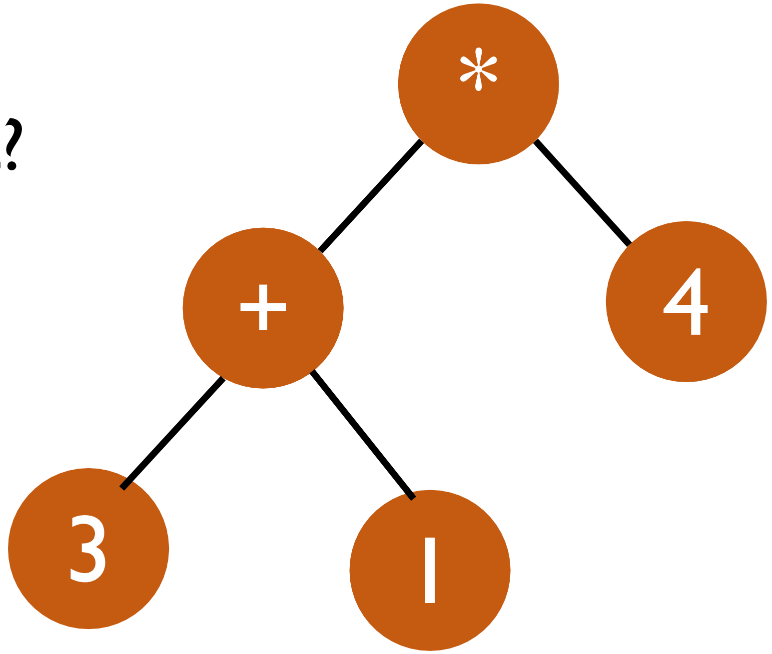
Non recursive pre-order traversal

- What are we using stack for in this case?
  - To remember nodes that we need to traverse
- Push left first or right first?
  - If we push left first, then left will be traversed later
  - Push right first, so that left will be traversed first



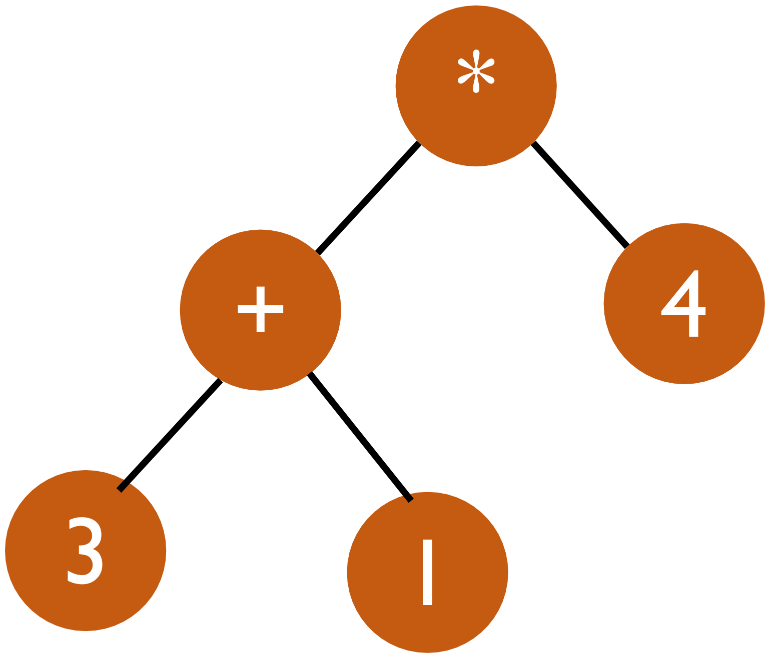
Non recursive in-order traversal

- How many times should we visit each node?
  - Twice
- What should happen on the first visit?
- What should happen on the second visit?



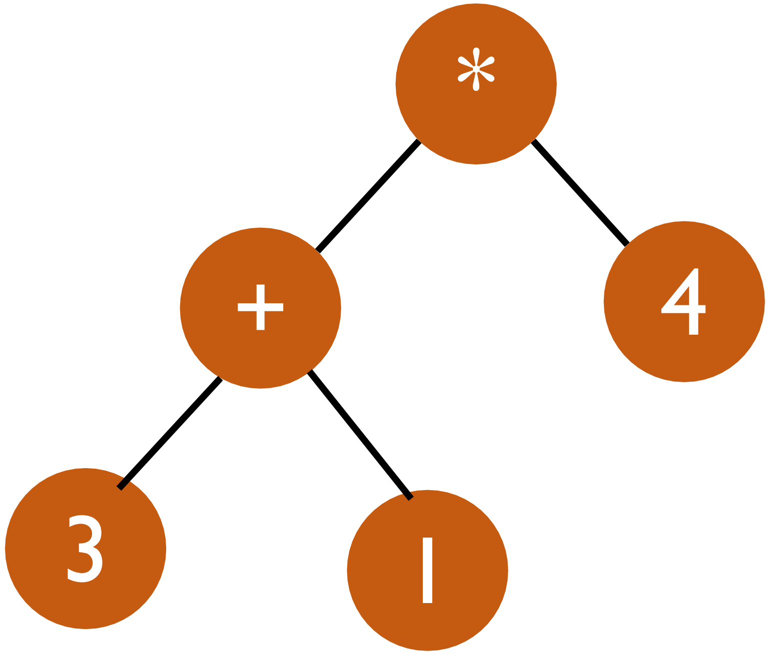
Non recursive post-order traversal

- How many times are we visiting each node?
  - Twice
- What happens on the first visit?
- What happens on the second visit?



How about level-order traversal?

- Which existing code to modify to make level-order?
- What are the 2 things to modify?



Is tree traversal unique given a binary tree?

- Practice question: Assume keys are unique.
  Given pre-order traversal and in-order traversal, construct a binary tree
  - int[] preorder = {10, 5, 3, 2, 7, 20, 15, 17, 25};
  - int[] inorder = {2, 3, 5, 7, 10, 15, 17, 20, 25};

Outline

1. Administrative
2. Trees Intro
3. Binary Tree Traversal
4. Binary Search Trees

Suppose we need a dictionary that inserts, finds, deletes items efficiently

- How can we do this with binary tree?

Binary Search Tree

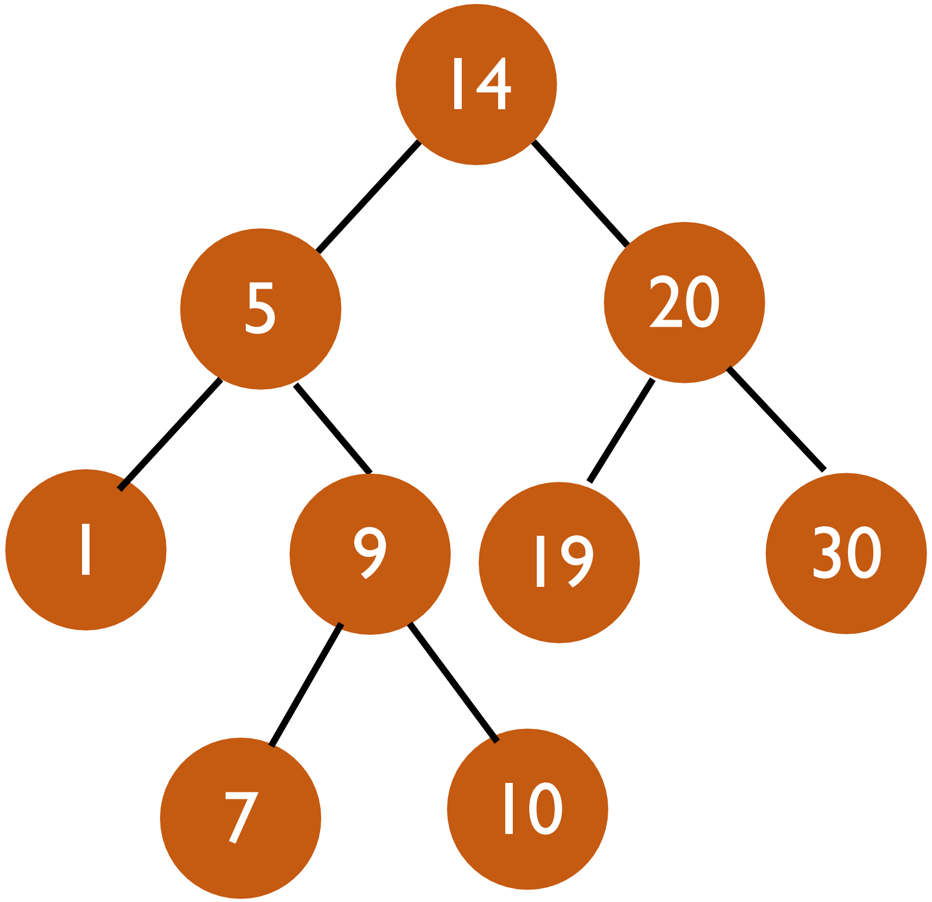
BST is a binary tree with BST property

BST Property
For every node x,
- all keys in x's left subtree have smaller values than the key in x
- all keys in x's right subtree have larger values than the key in x

Does it say anything about the tree being balanced?

Find on BST

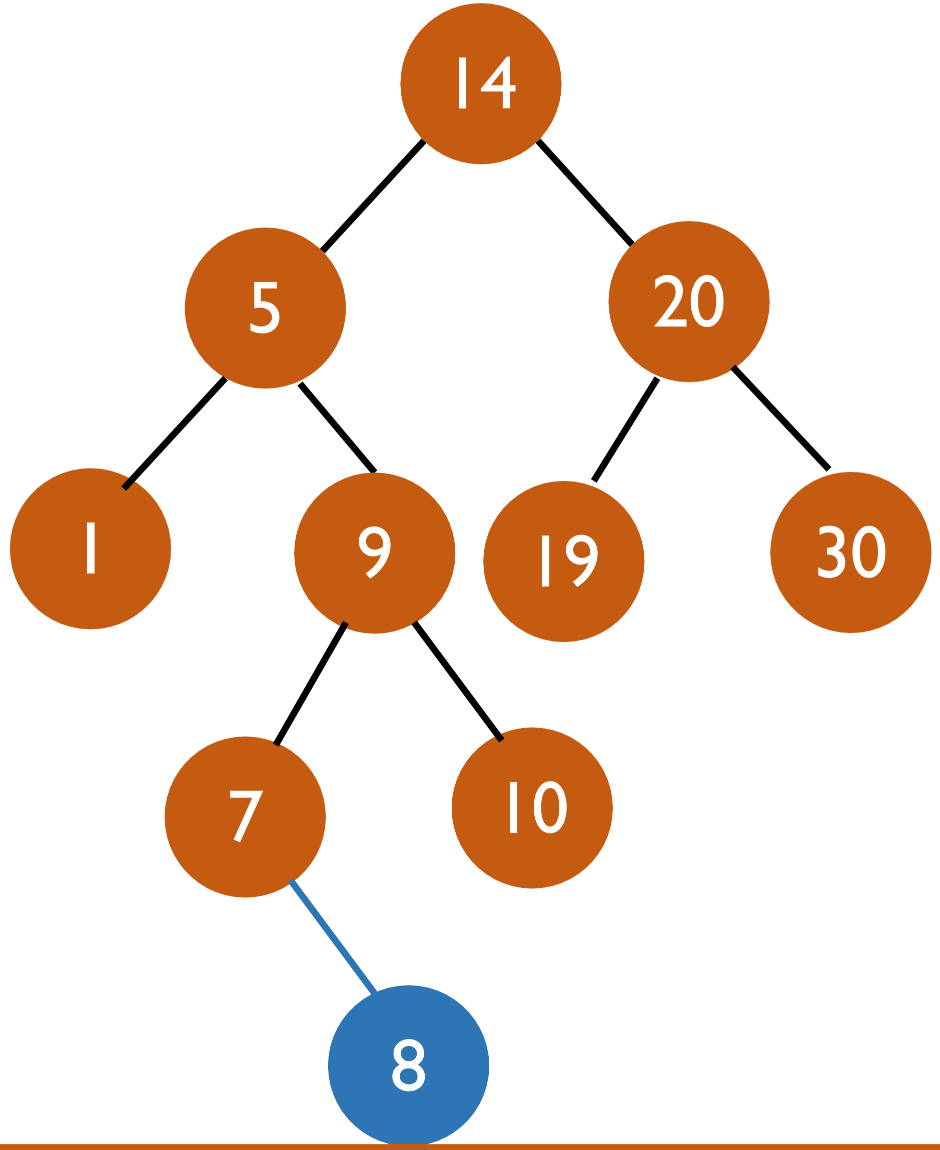
```
Find 8
Find 10
```



What is the time complexity of find?

Insert on BST

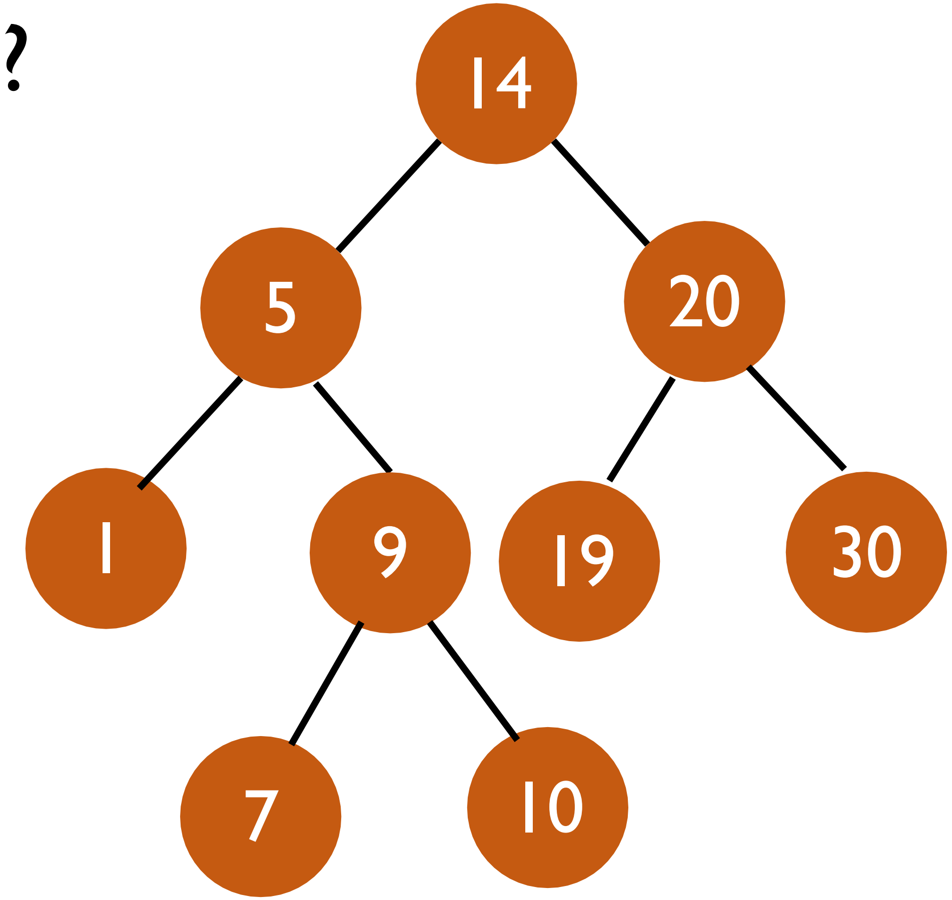
`Insert 8`



What is the time complexity of insert?

Duplicate on BST?

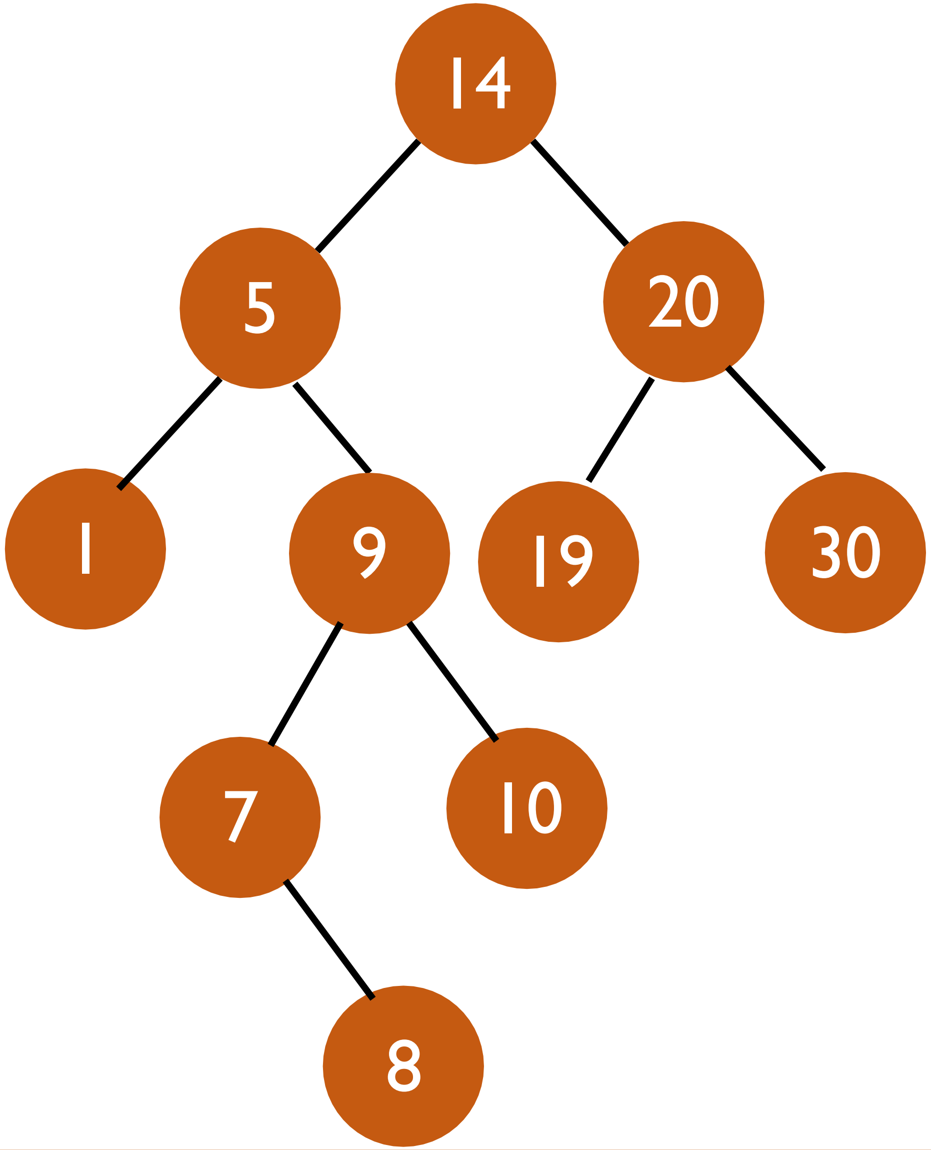
`Insert 10`



Flag an error, ignore, or allow duplicate – all possible options

Delete on BST

Delete 19



Case 1: Deleting on a leaf node – simple!

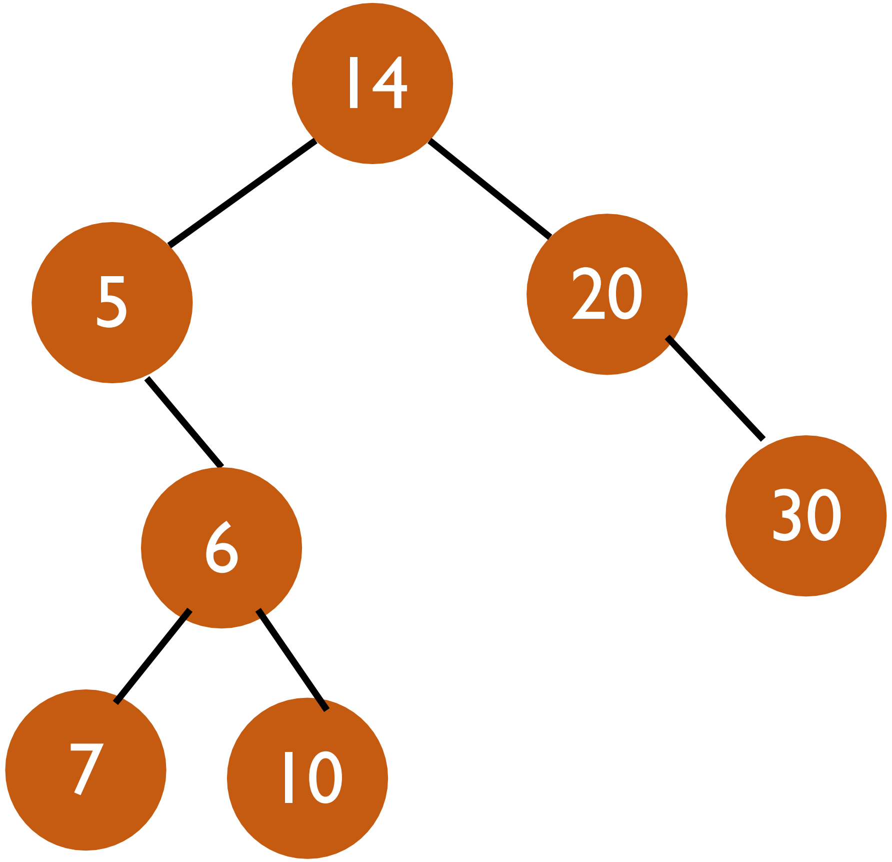
Deleting an internal node

Case 2: when the internal node has either left or right subtree

Case 3: when the internal node has both left and right subtree

Case 2: Deleting an internal node with either left/right subtree

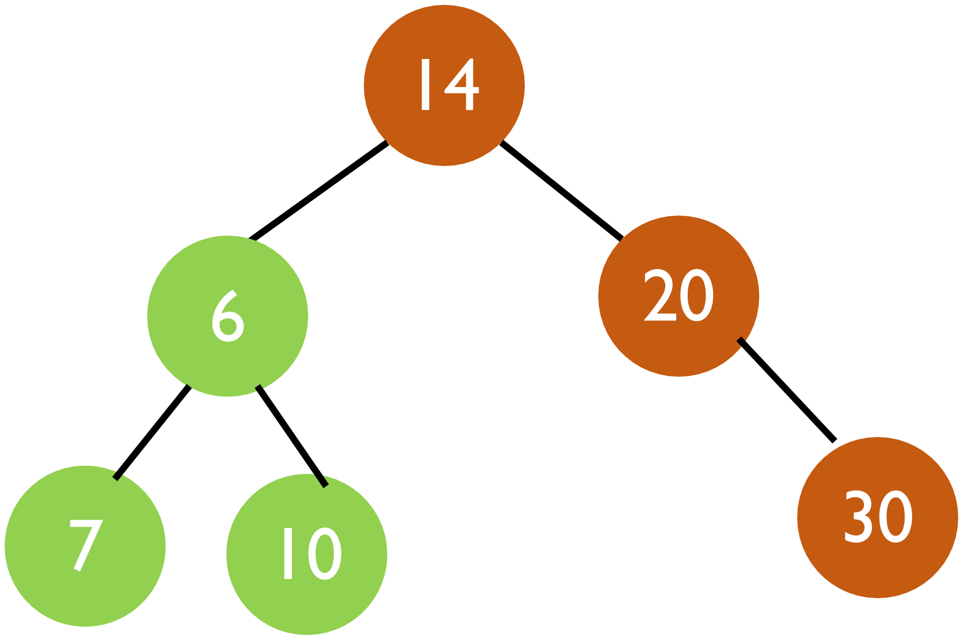
`Delete 5`



Which node should replace 5 to preserve BST property?

Case 2: Deleting an internal node with either left/right subtree

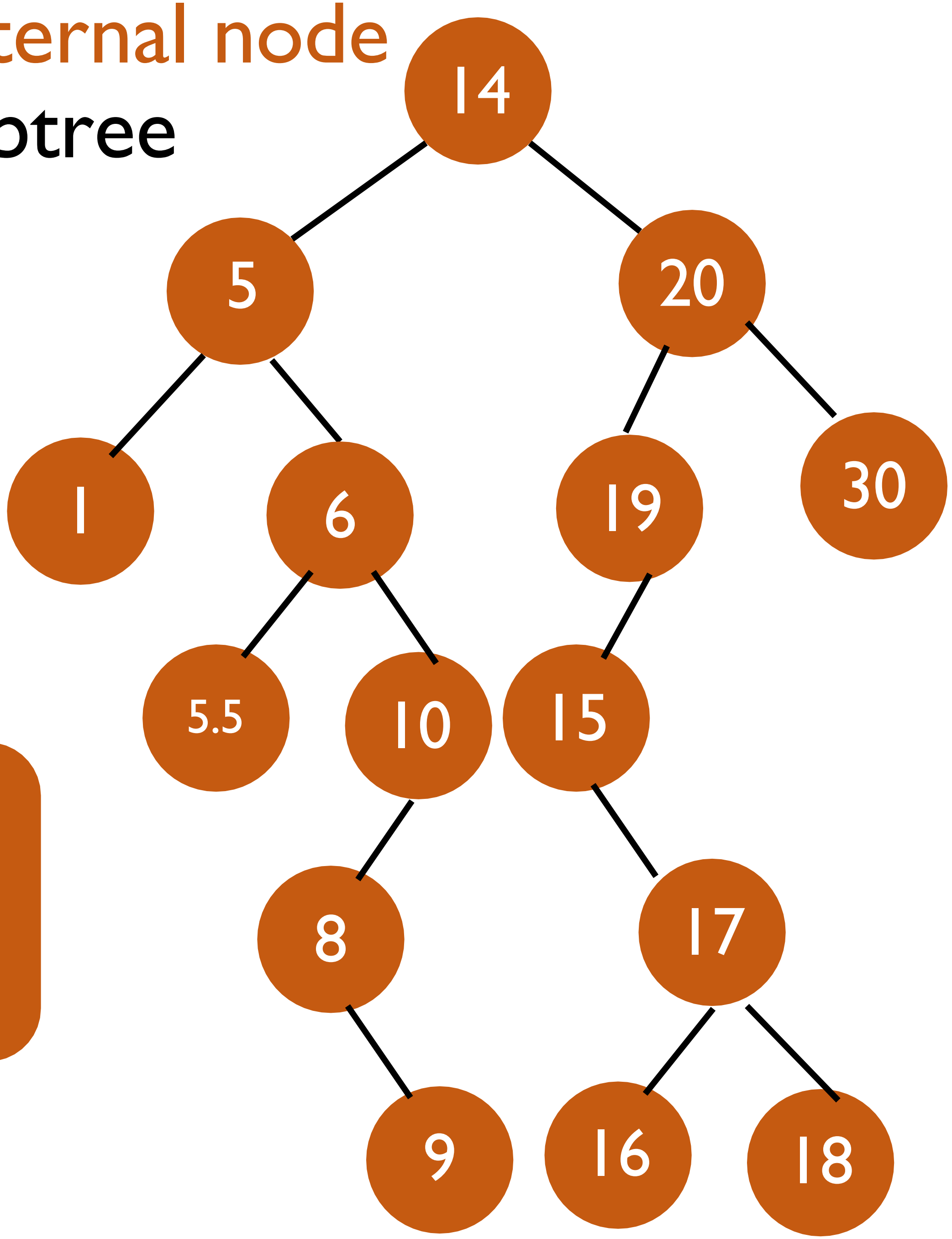
```
Delete 5
```



Its subtree can replace the internal node. Why?

Case 3: Deleting an internal node with both left/right subtree

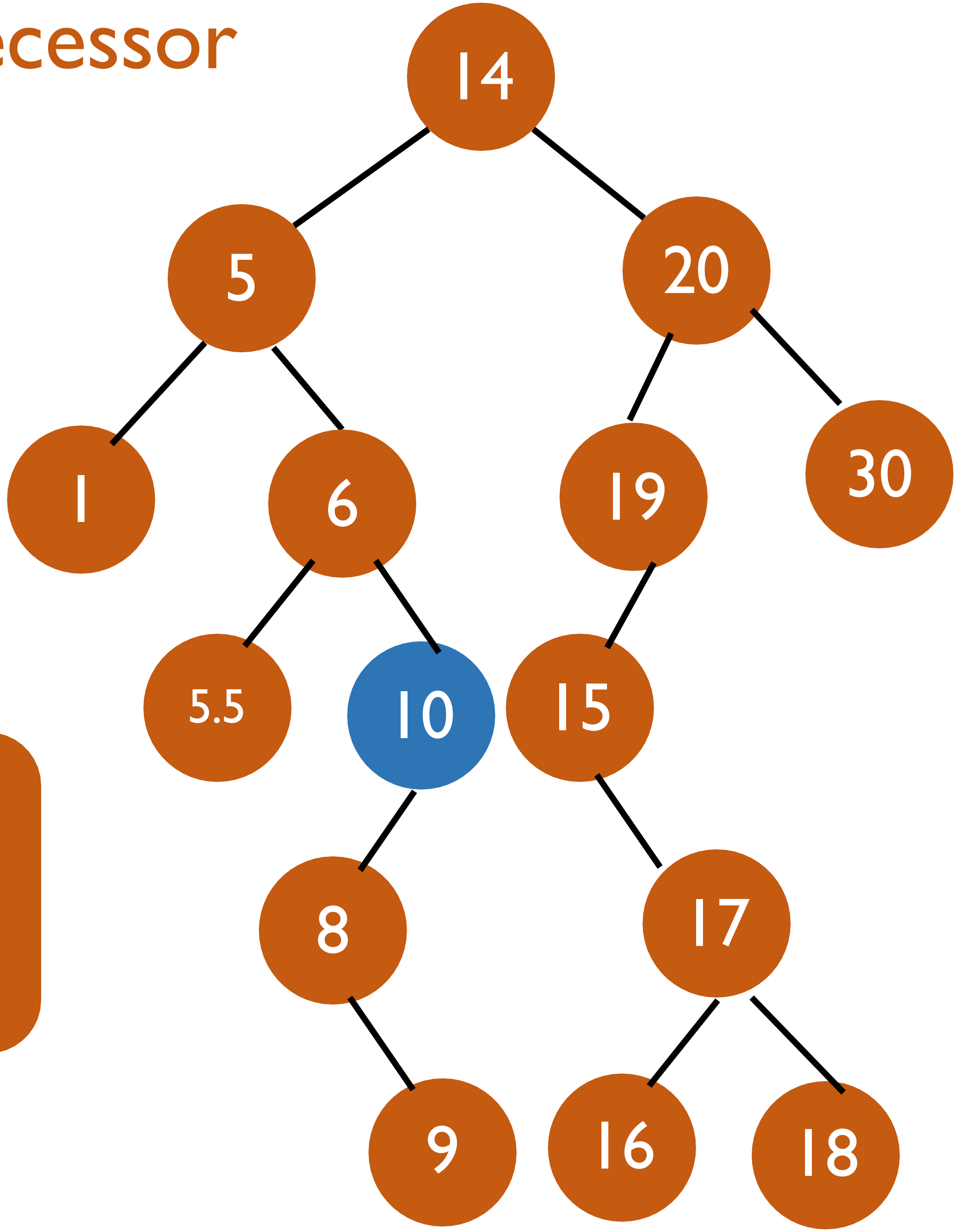
`Delete 14`



Which node should replace 14 to preserve BST property?

Replace with predecessor

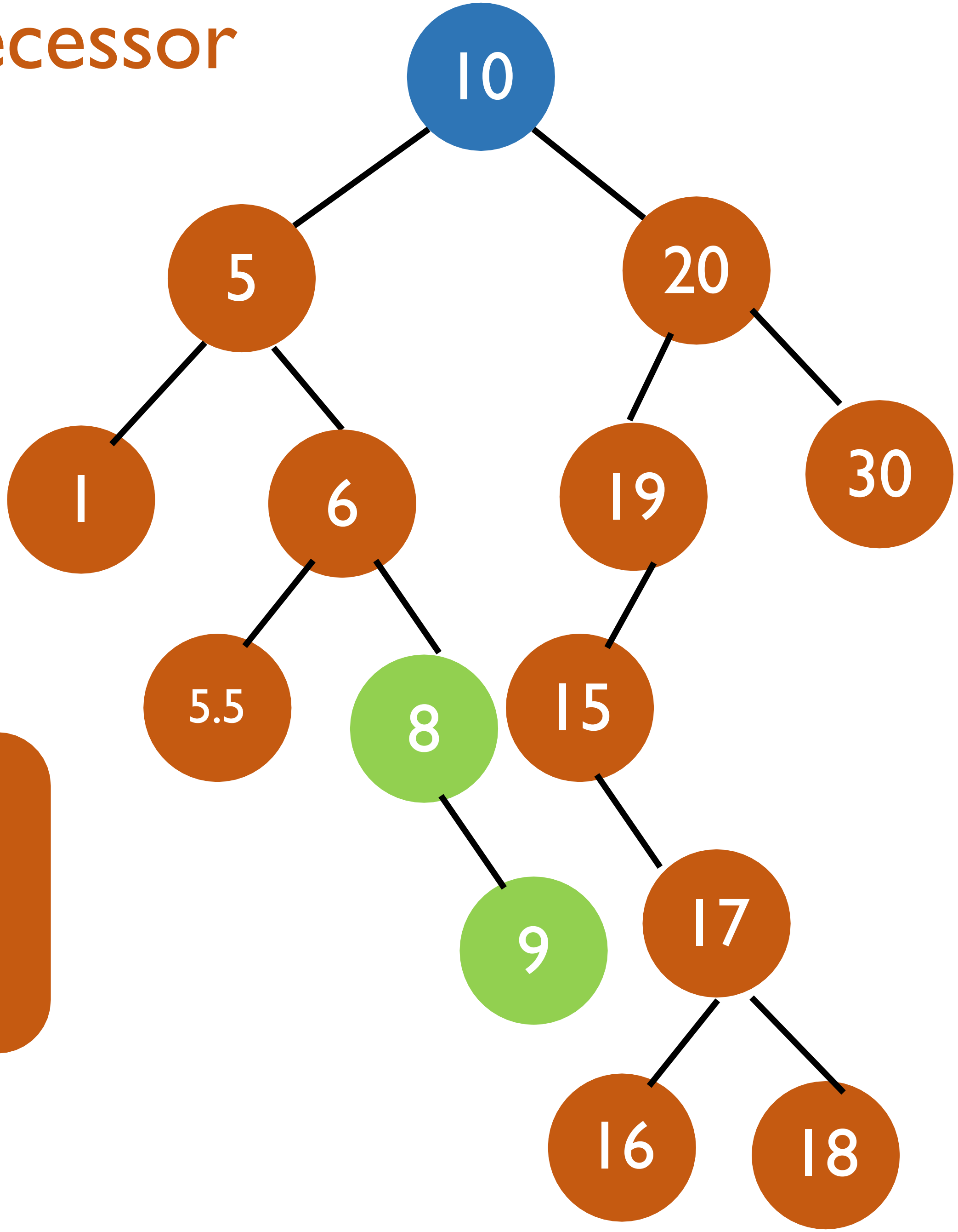
`Delete 14`



Replace with the largest key in the left subtree

Replace with predecessor

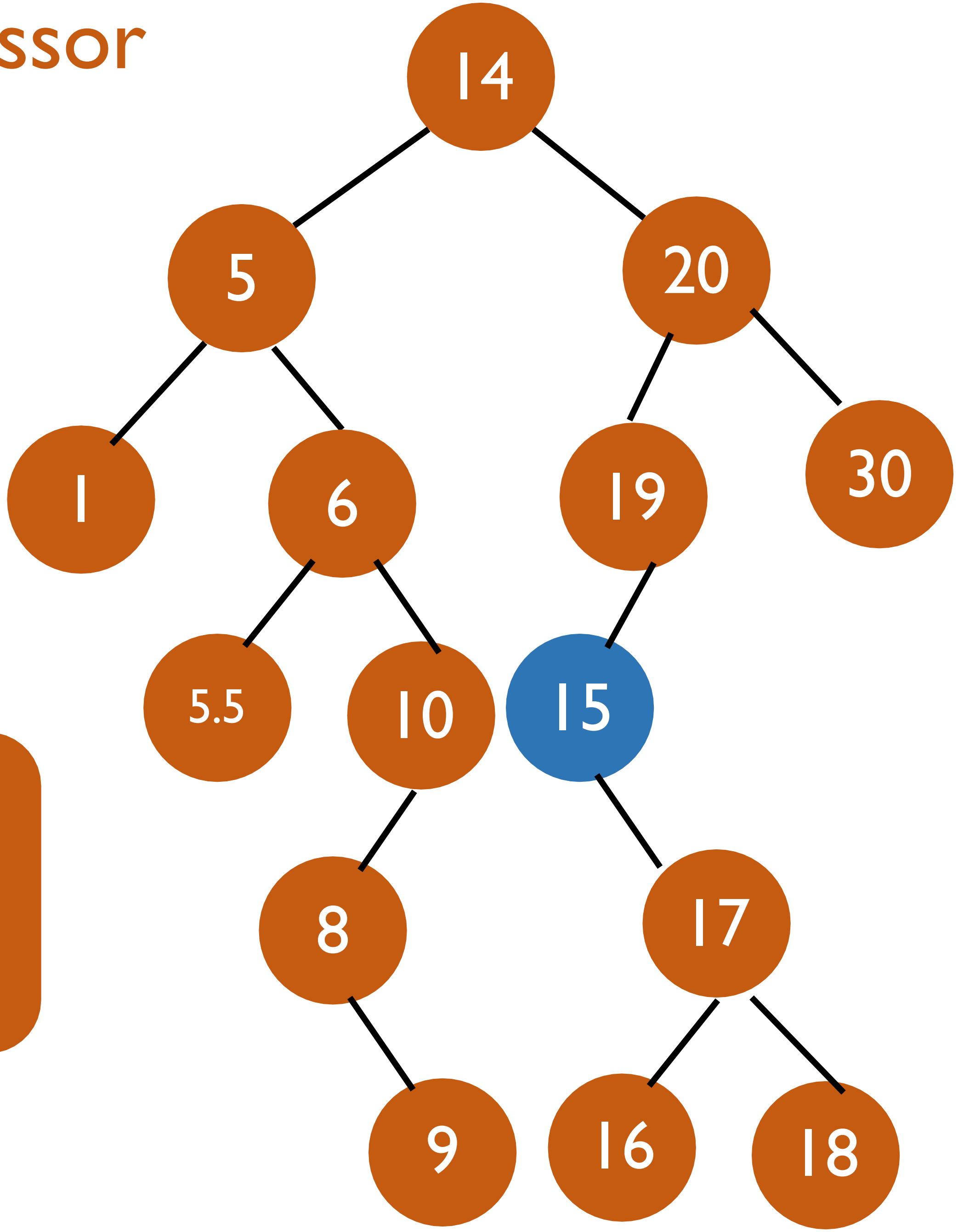
Delete 14



Replacing will always result in case 1 or case 2

Replace with successor

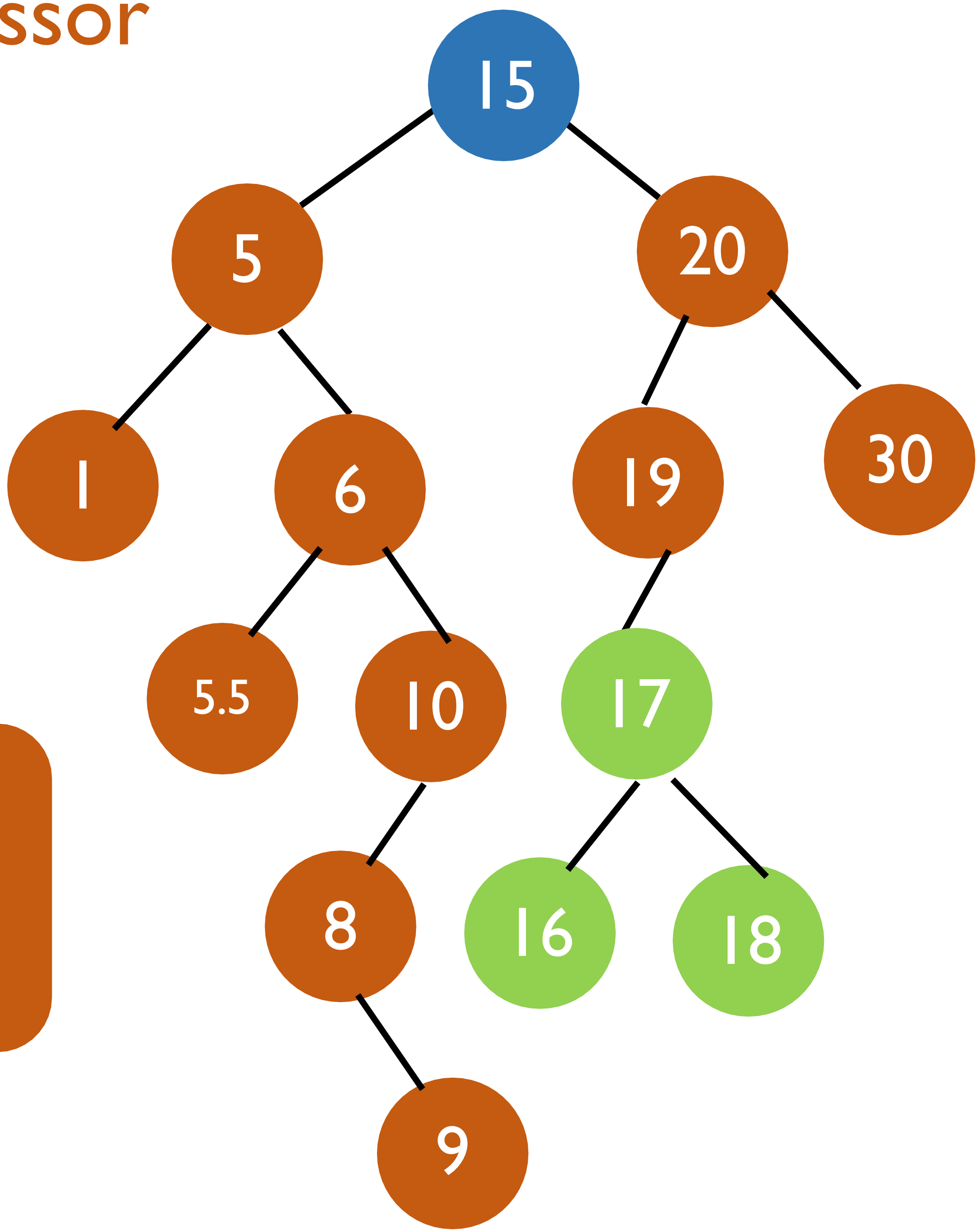
Delete 14



Replace with the smallest key in the right subtree

Replace with successor

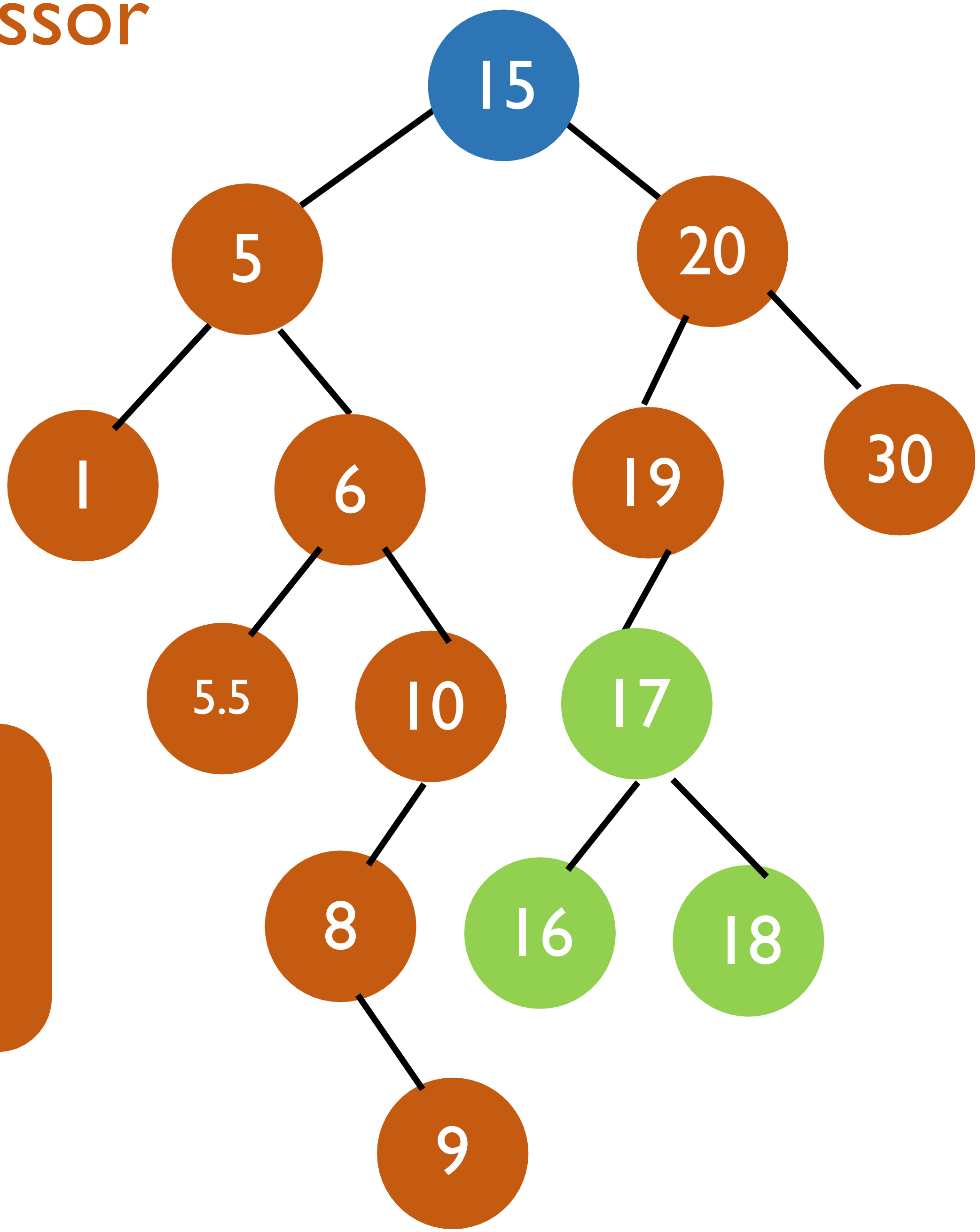
Delete 14



Replacing
will always result in
case 1 or case 2

Replace with successor

`Delete 14`



Replacing
will always result in
case 1 or case 2

Outline

1. Administrative
2. Trees Intro
3. Binary Tree Traversal
4. Binary Search Tree
5. Binary Tree Analysis

Complete Binary Tree

- **All the levels of the tree are filled completely except the lowest level**
  - At the lowest level, nodes are filled from as left as possible
- **Which is NOT a complete binary tree?**
  - Num nodes at level k?
  - Min num nodes at the lowest level?

```
     1
    / \
   2   3
  / \ /
 4  5 6
```

```
        10
       /  \
     20    30
    /  \  / \
   40   50 60 70
  / \  /
 80 90 100
```

```
   1
  / \
 2   3
  \
   4
```

Complete Binary Tree

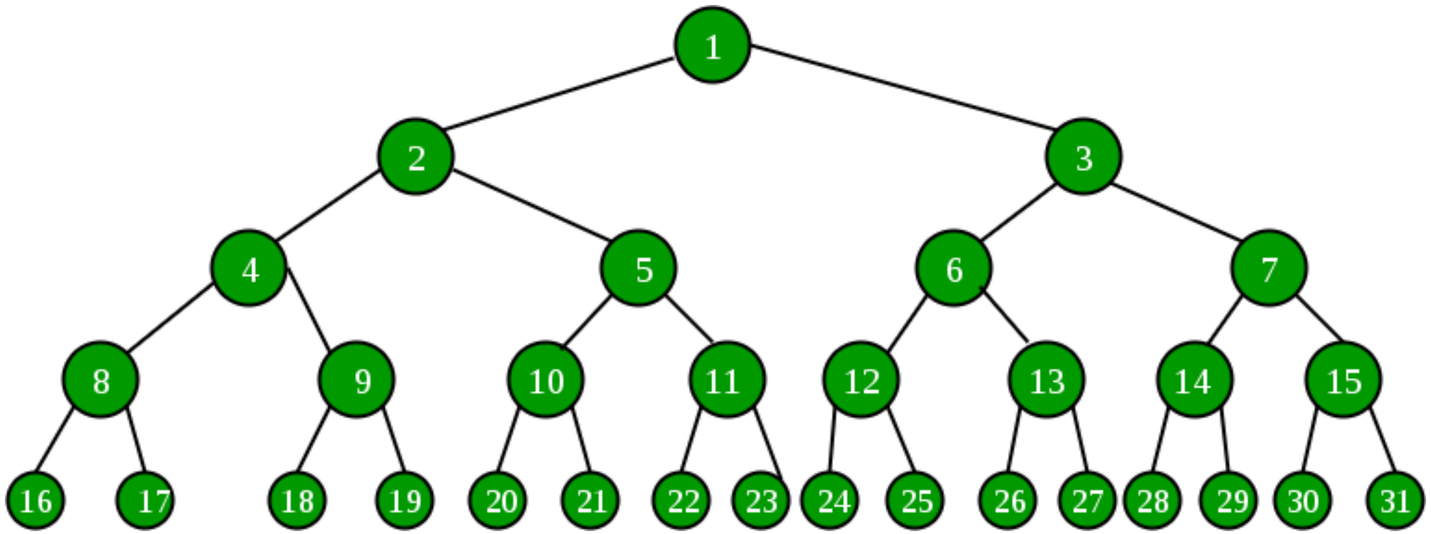
- All the levels of the tree are filled completely except the lowest level
  - At the lowest level, nodes are filled from as left as possible
- Num nodes at level k?
- Min/max num nodes at the lowest level?

Complete Binary Tree

- **All the levels of the tree are filled completely except the lowest level**
  - At the lowest level, nodes are filled from as left as possible
- **Num nodes at level k?**
  - Root is at level 0
  - 2^k
- **Min/max num nodes at the lowest level?**
  - Let k be the number of levels
  - Min: 1
  - Max: 2^{k-1}

Perfect Binary Tree

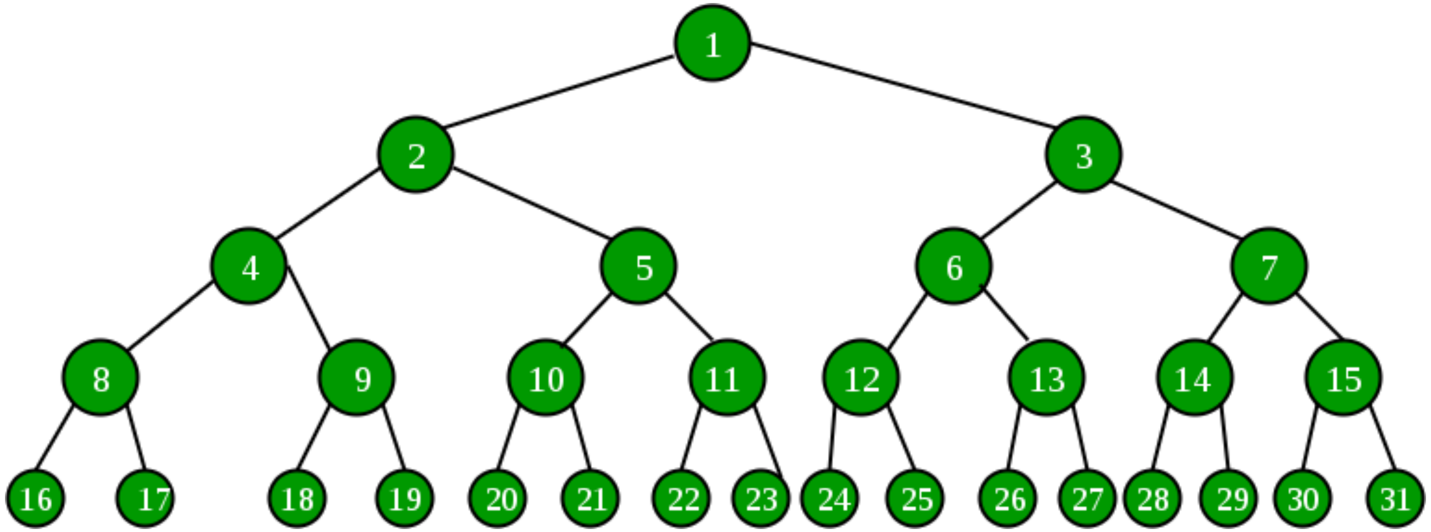
- All the levels of the tree are filled completely including the lowest level



- Total number of nodes given there are total k levels?
- How many leaf nodes?
- How many internal nodes?

Perfect Binary Tree

- All the levels of the tree are filled completely including the lowest level



Let k be the number of levels

- Total 2^k-1 nodes
- 2^{k-1} leaf nodes
- 2^{k-1}-1 internal nodes