CS151 Intro to Data Structures

Tree Traversals

Outline

- HW comments
- Iterator
- Trees:
  - Overview
  - Binary Search Tree
    - Inserting
    - Searching

Announcements

- HW03 (Stacks & Queues) – due Friday 10/27
    - Must include your own Junit tests
    - 10% of grade
    - Have one file that contains all the Unit tests

- Lab 04, 05, 06 due Friday 10/27
    - Lab 06 (last week’s lab) no checkoff, due on Gradescope

Homework Comments

HW00 grades/feedback returned

HW02 grades/feedback returned

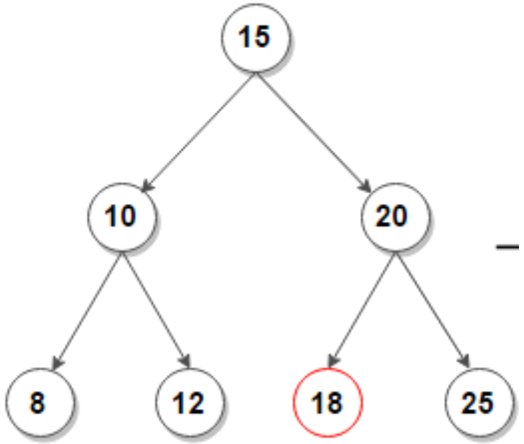
HW02 will be returned this week

Remove

- `boolean remove(E element);`
- returns true if element existed and was removed and false otherwise
- Cases
  - element not in tree
  - element is a leaf
  - element has one child
  - element has two children

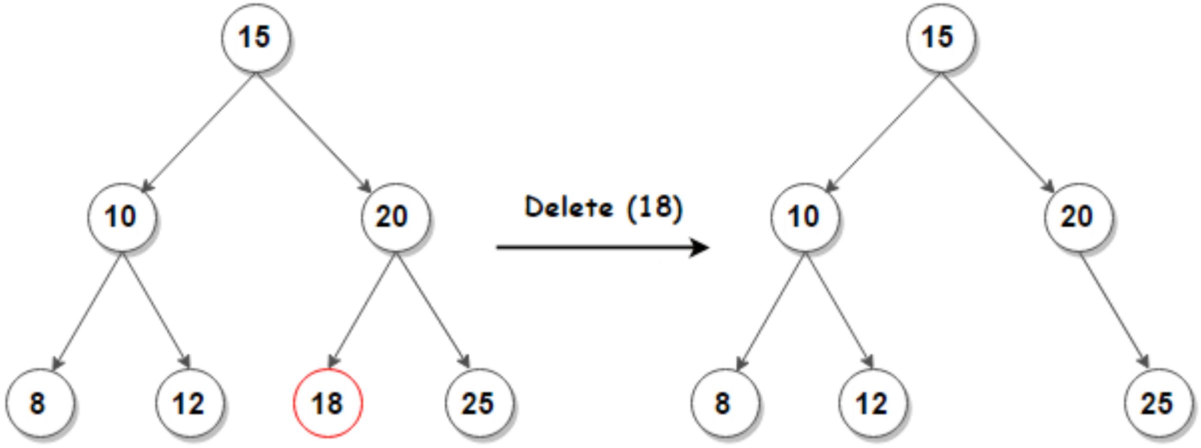
Leaf

- Just delete



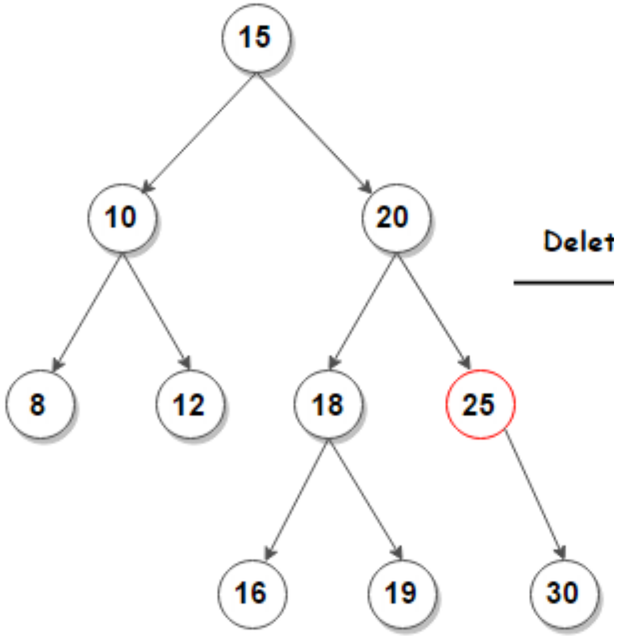
Leaf

- Just delete



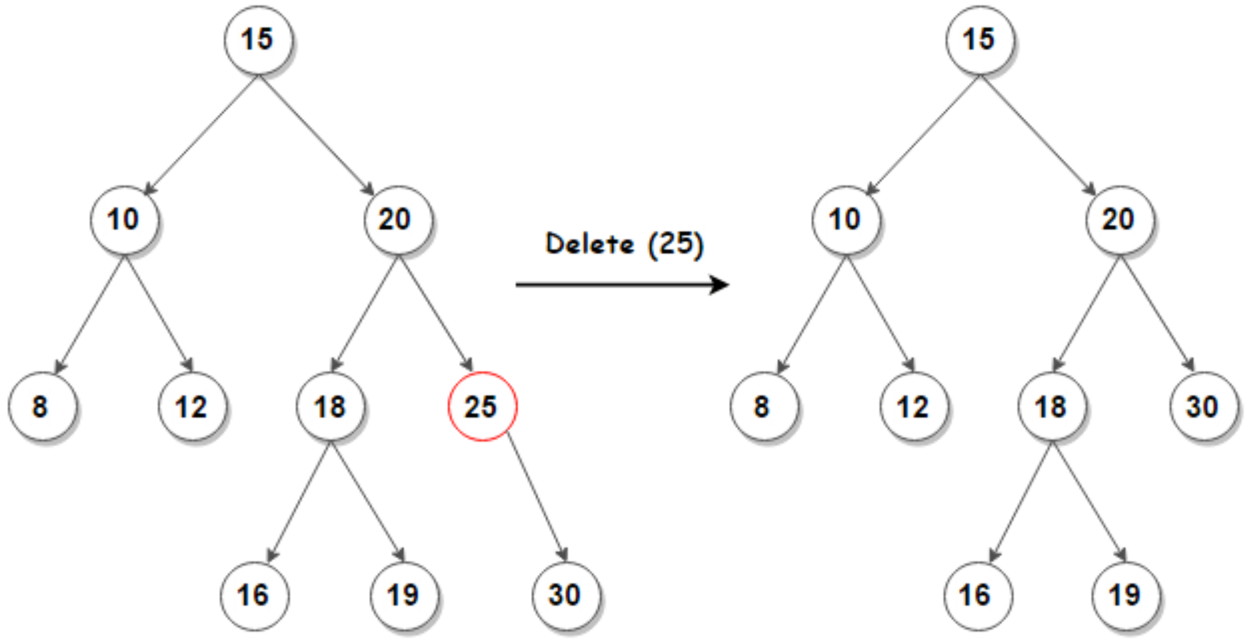
One child

- Replace with child – skip over like in linked list



One child

- Replace with child – skip over like in linked list



Interface

```
public interface BinaryTree<E extends
Comparable<E>> {
  E getRootElement();
  int size();
  boolean isEmpty();
  void insert(E element);
  boolean contains(E element);
  String toStringInOrder();
  String toStringPreOrder();
  String toStringPostOrder();
}
```

Binary Tree Traversals

Traversal visits all nodes in a tree in some order

Inorder:

left subtree, current, right subtree

Preorder:

current, left subtree, right subtree

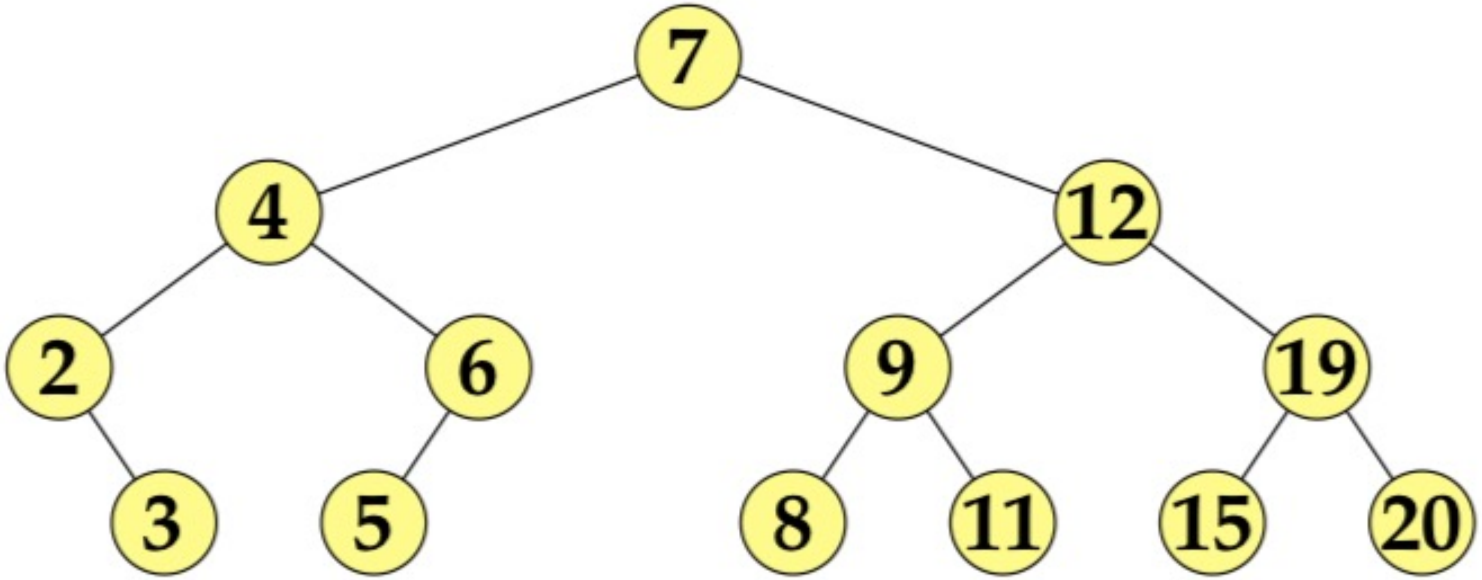
Postorder:

left subtree, right subtree, current

Inorder

What would the in-order traversal be here?

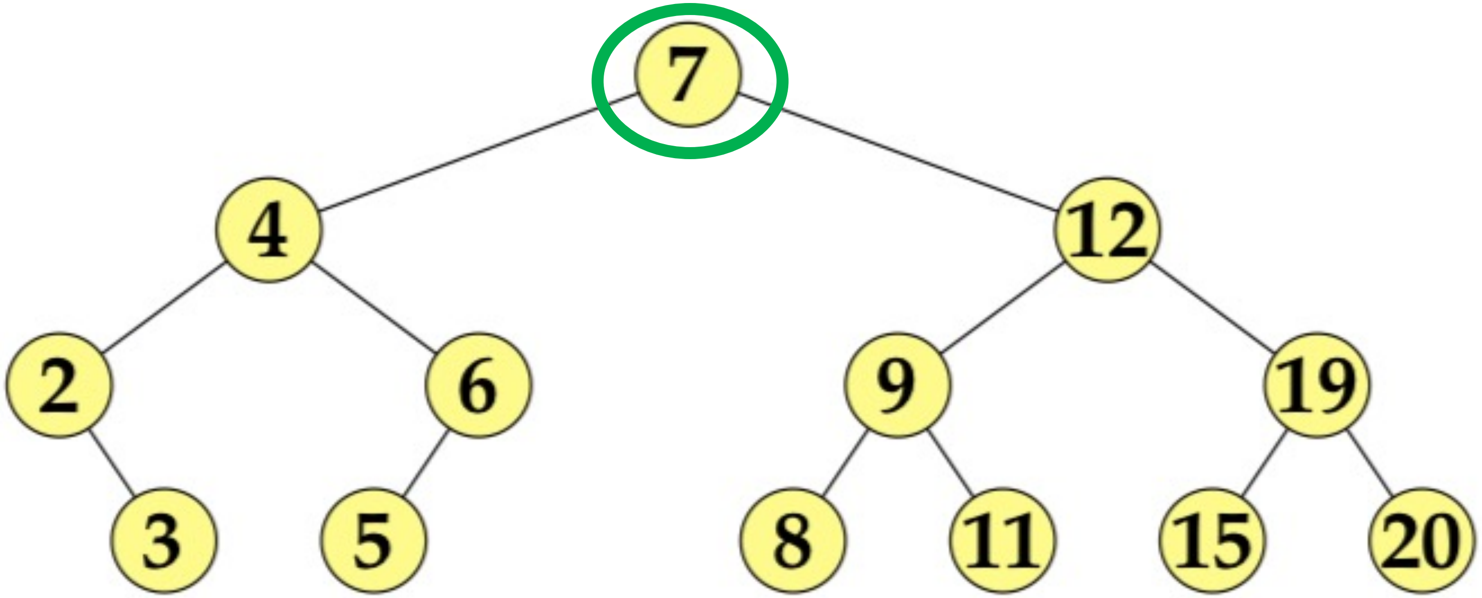
left subtree, current, right subtree



Inorder

What would the in-order traversal be here?

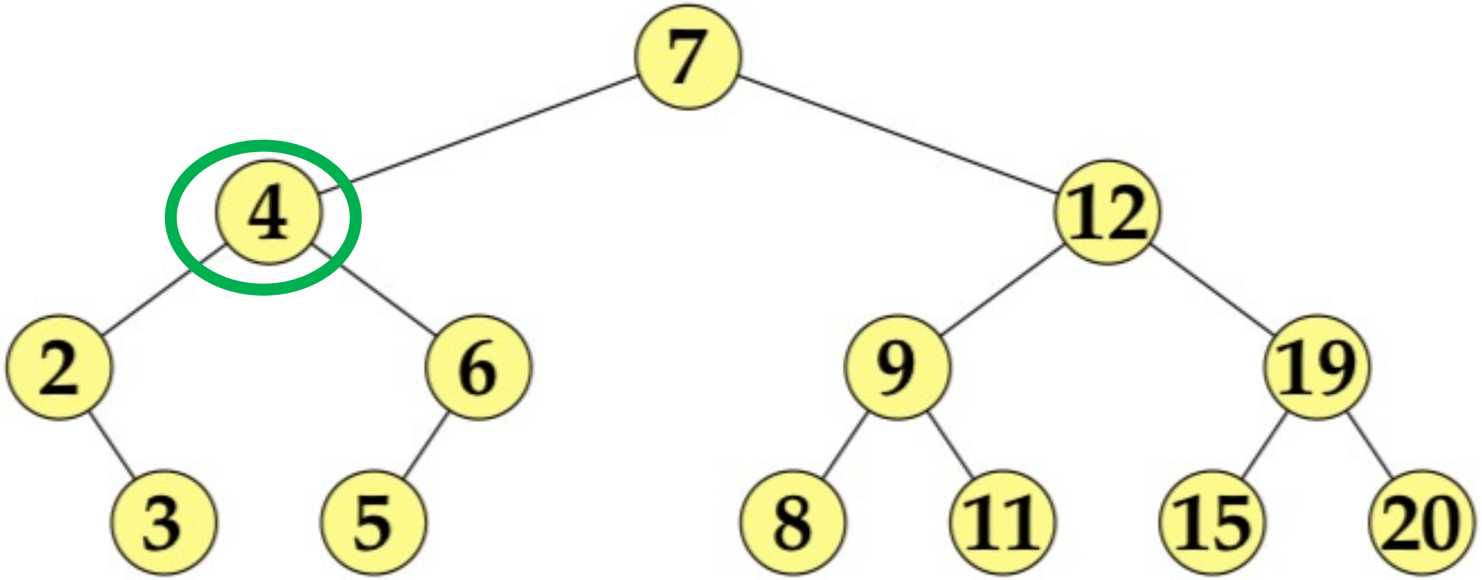
left subtree, current, right subtree



Inorder

What would the in-order traversal be here?

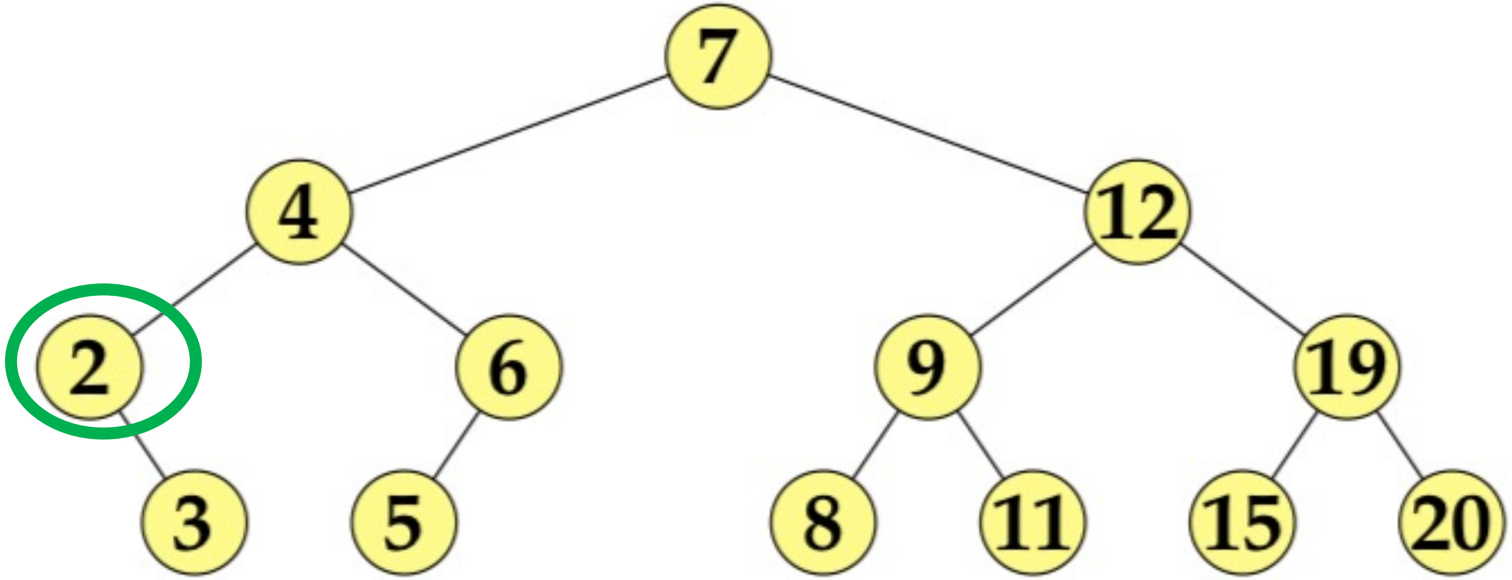
left subtree, current, right subtree



Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

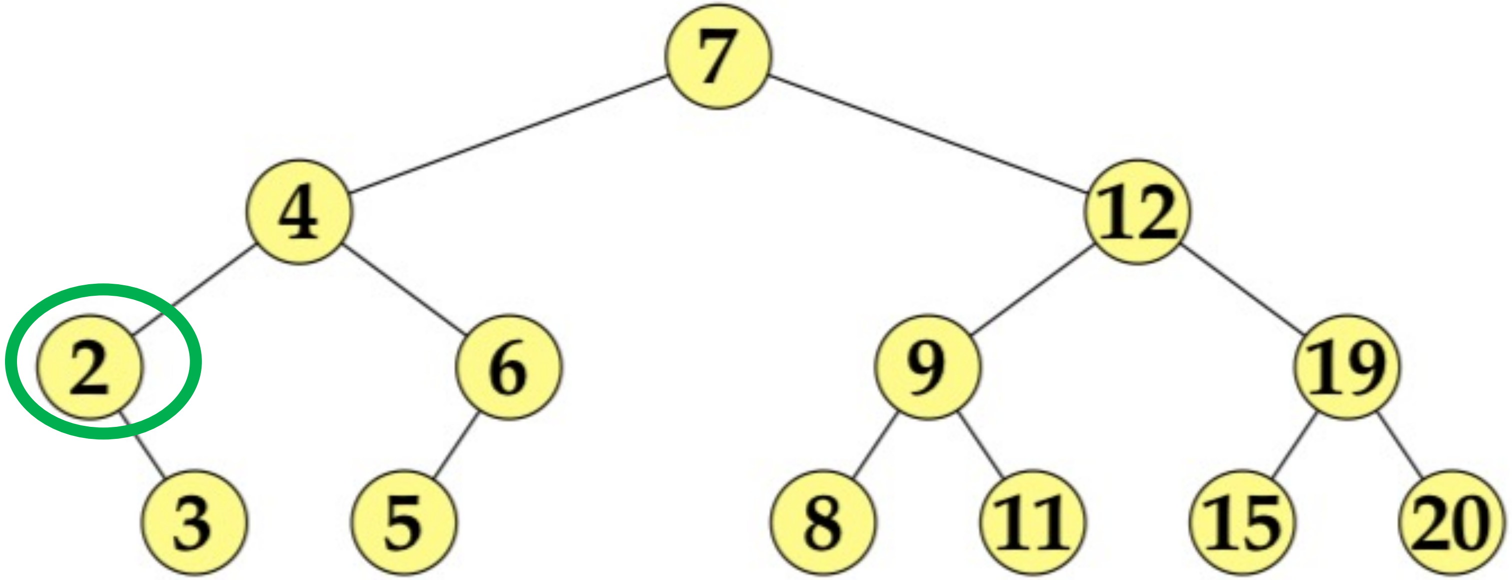


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2,

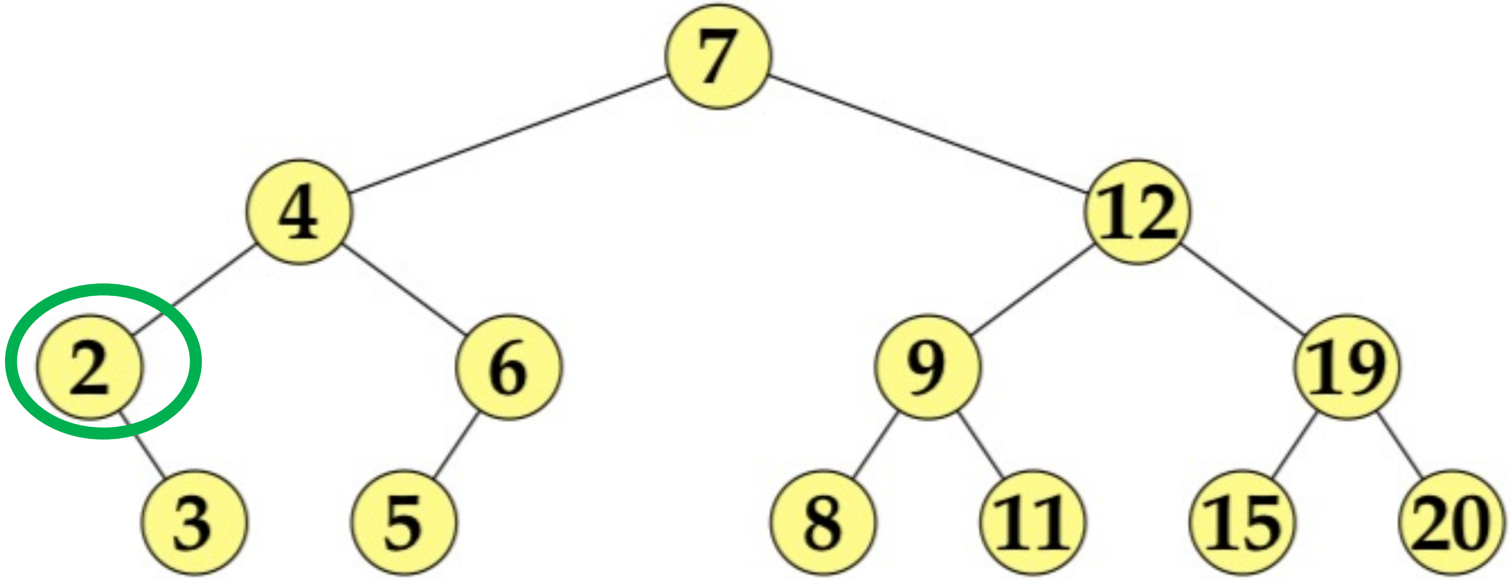


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2,

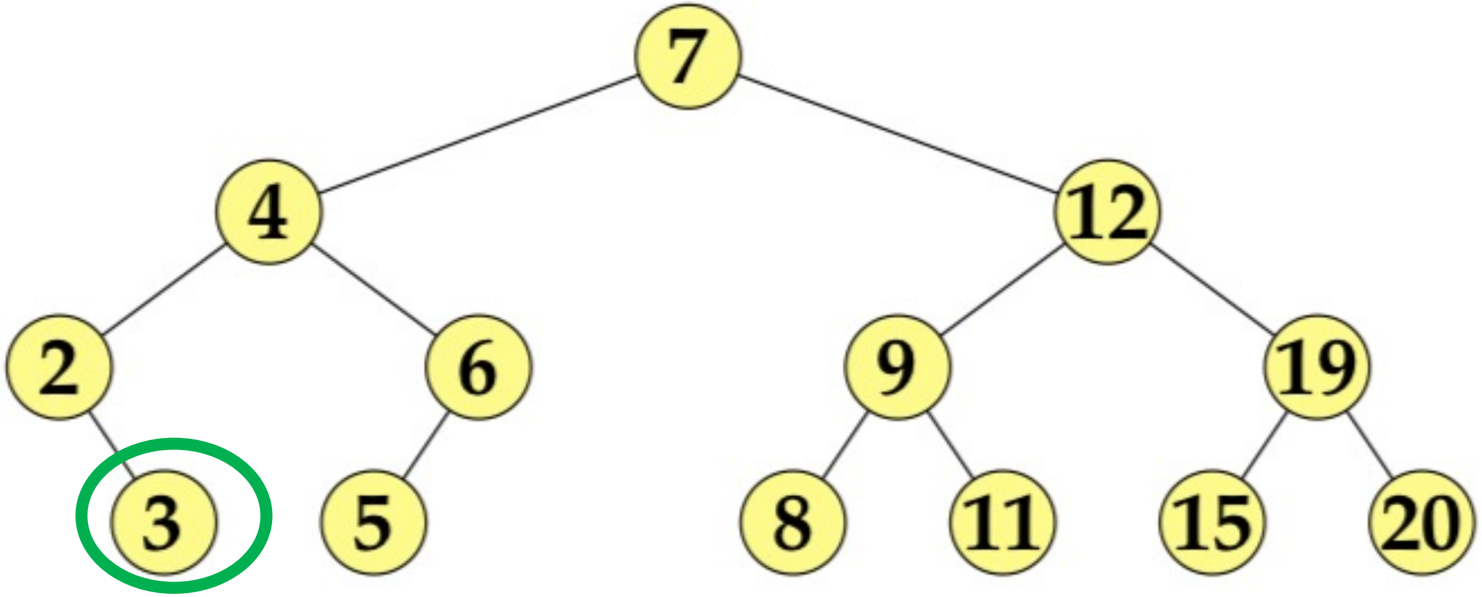


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2,

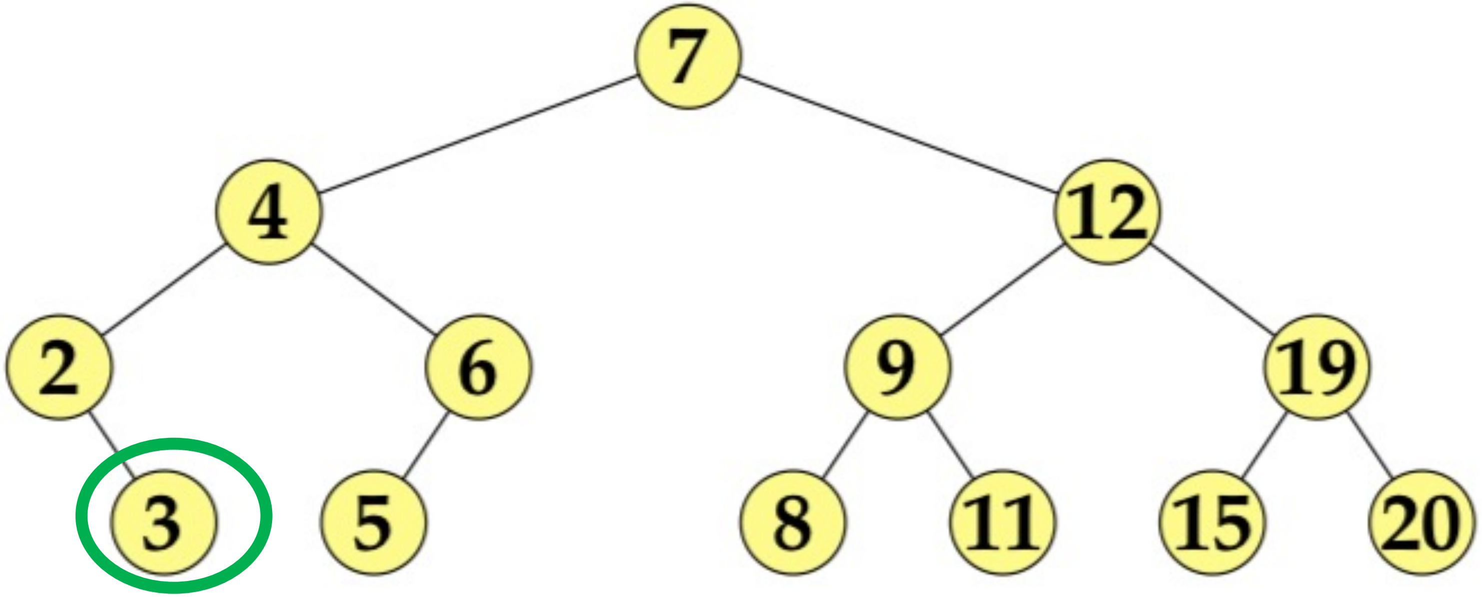


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3

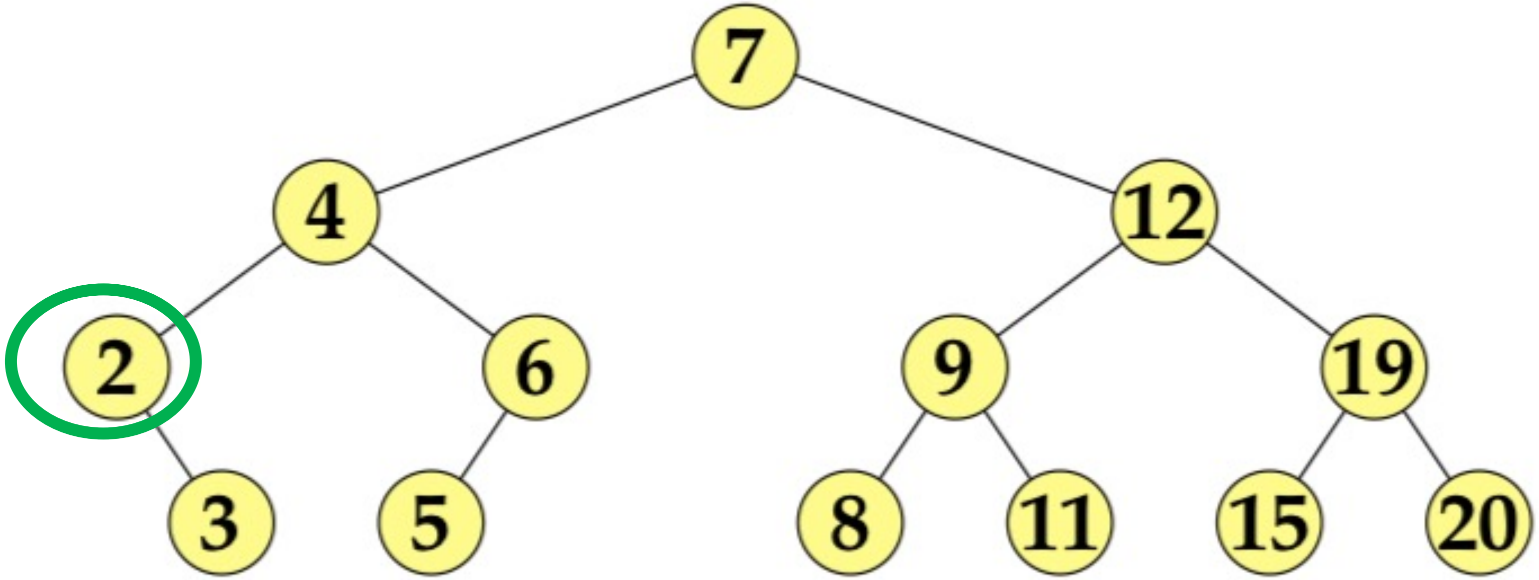


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3

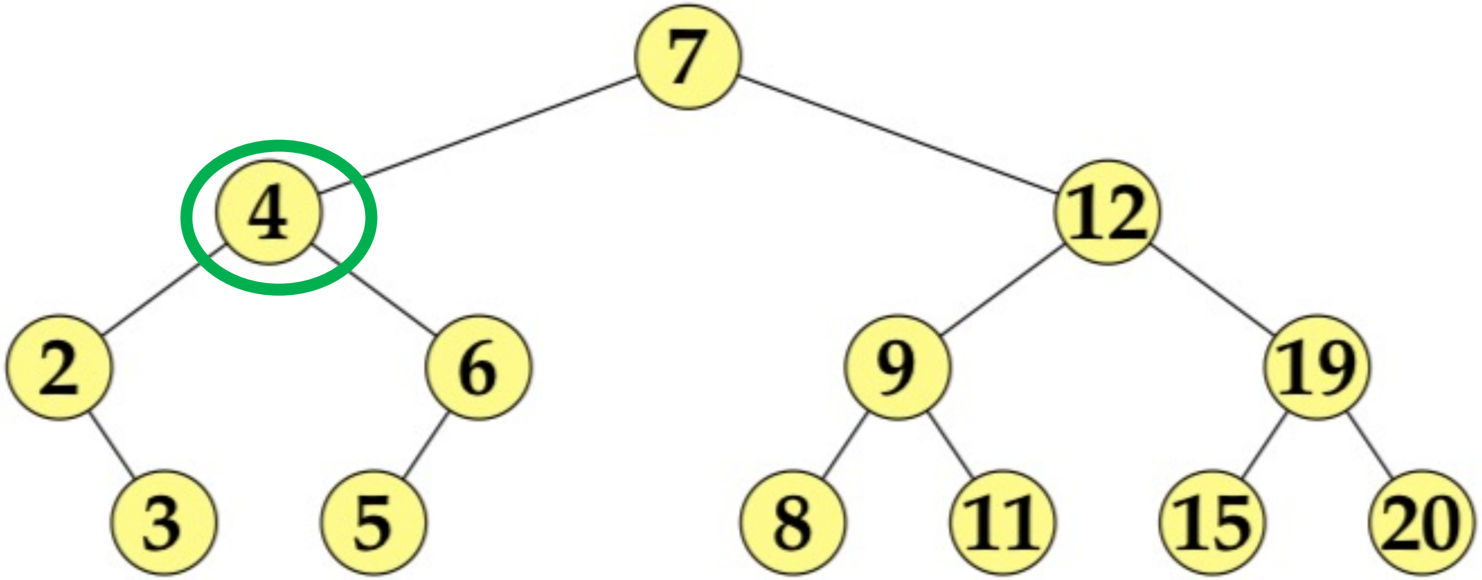


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3

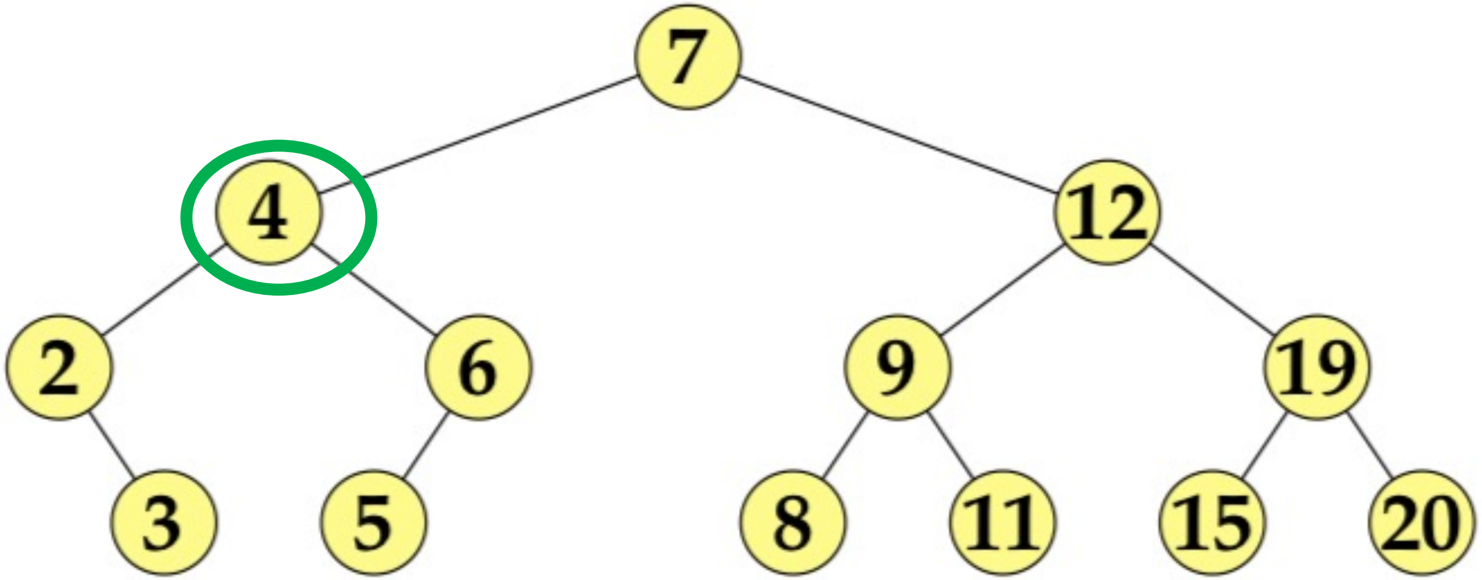


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4

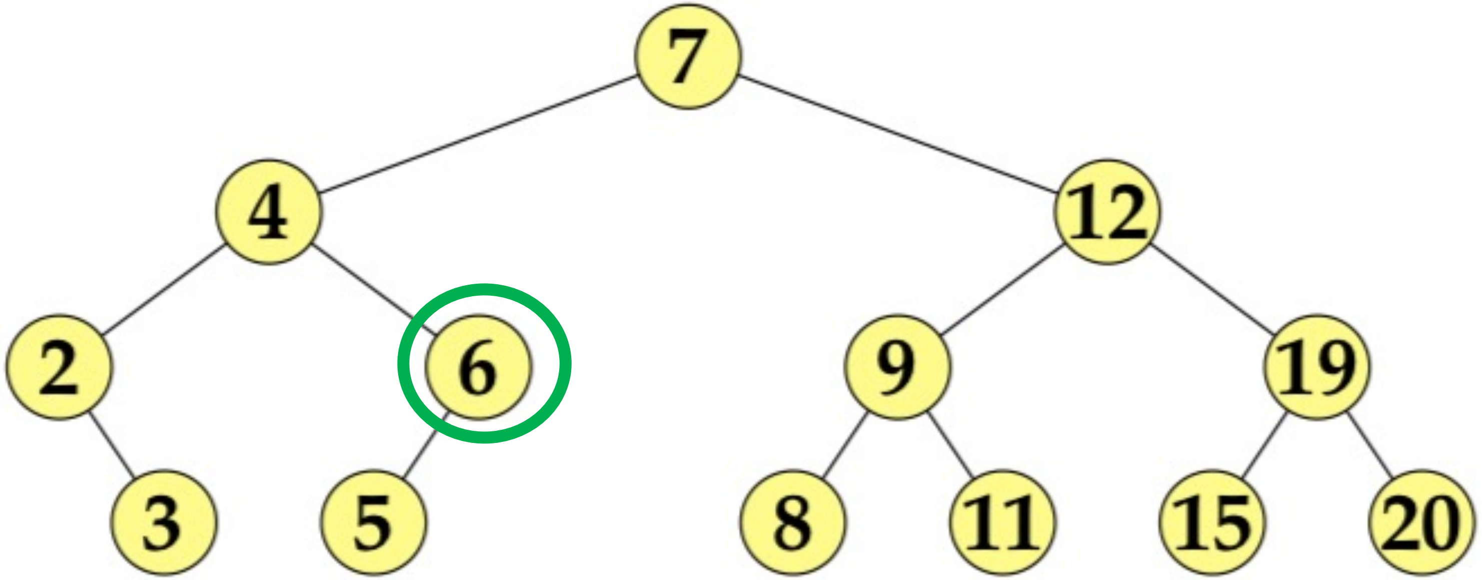


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4

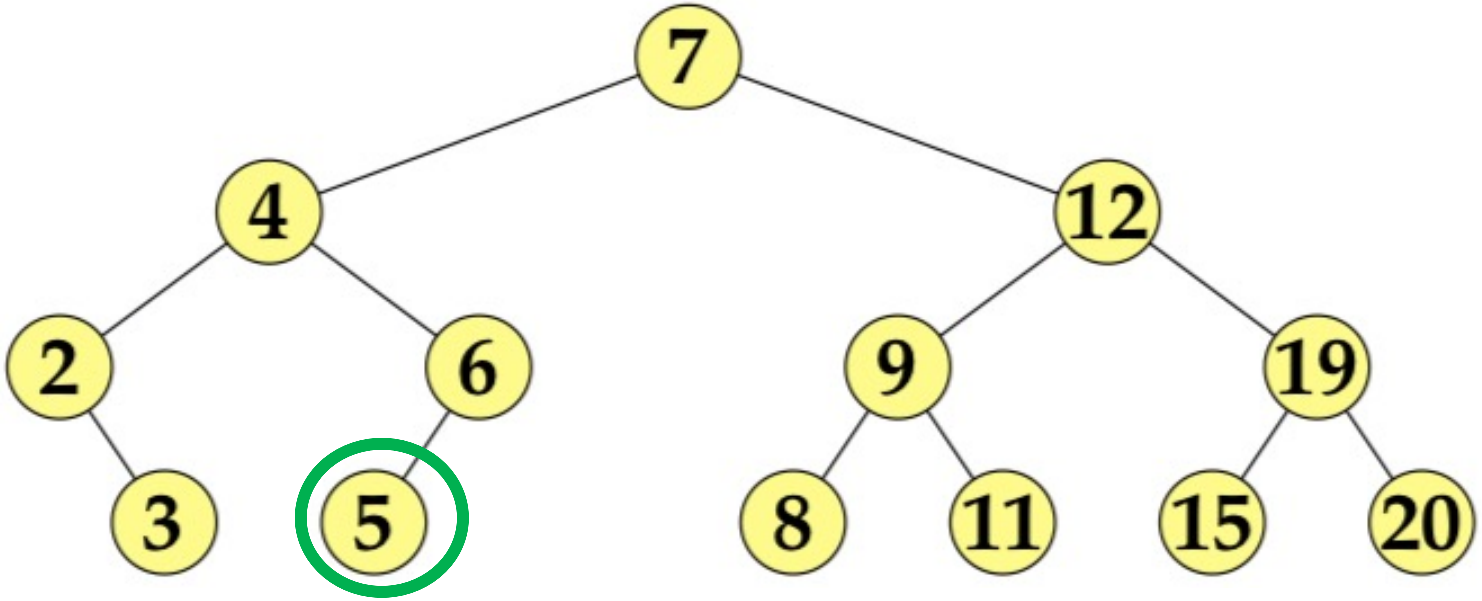


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4

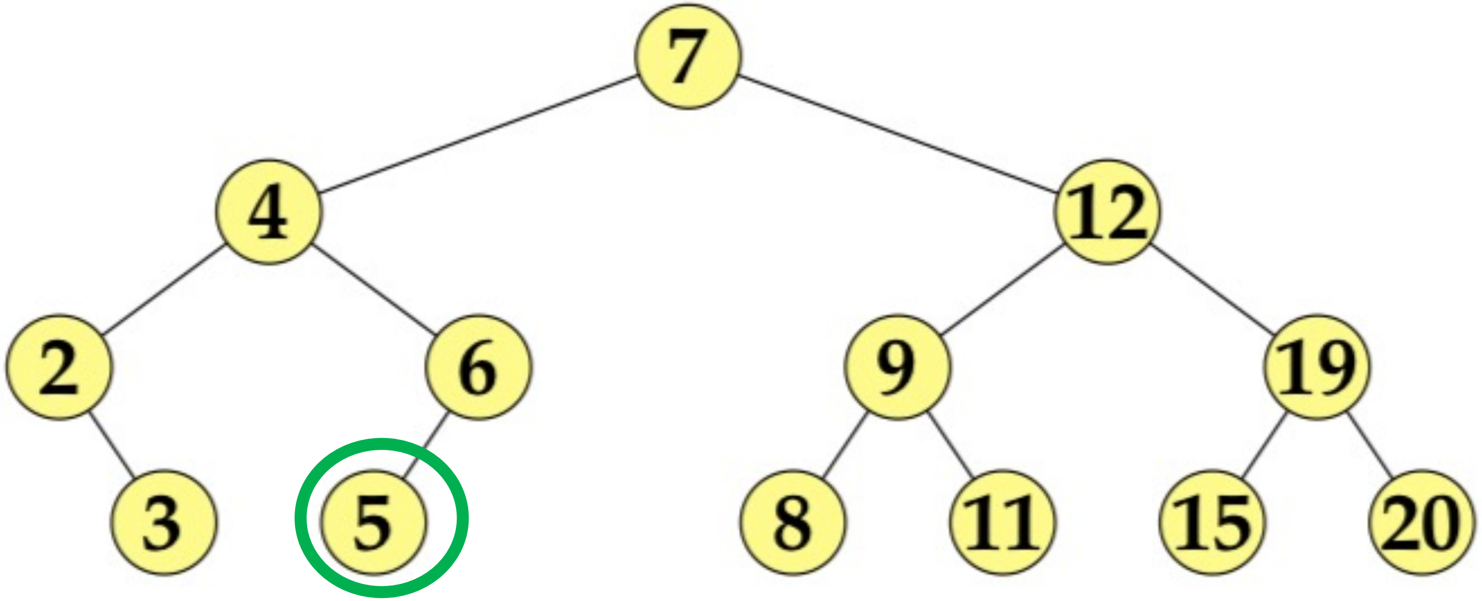


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5

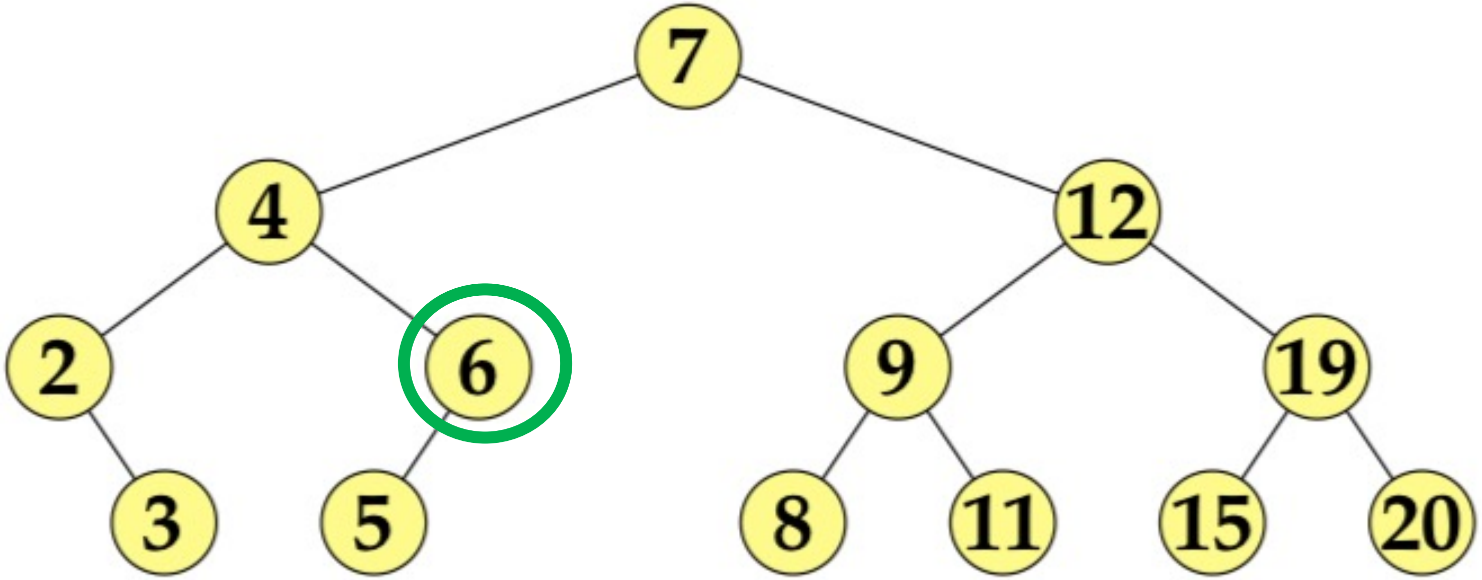


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5

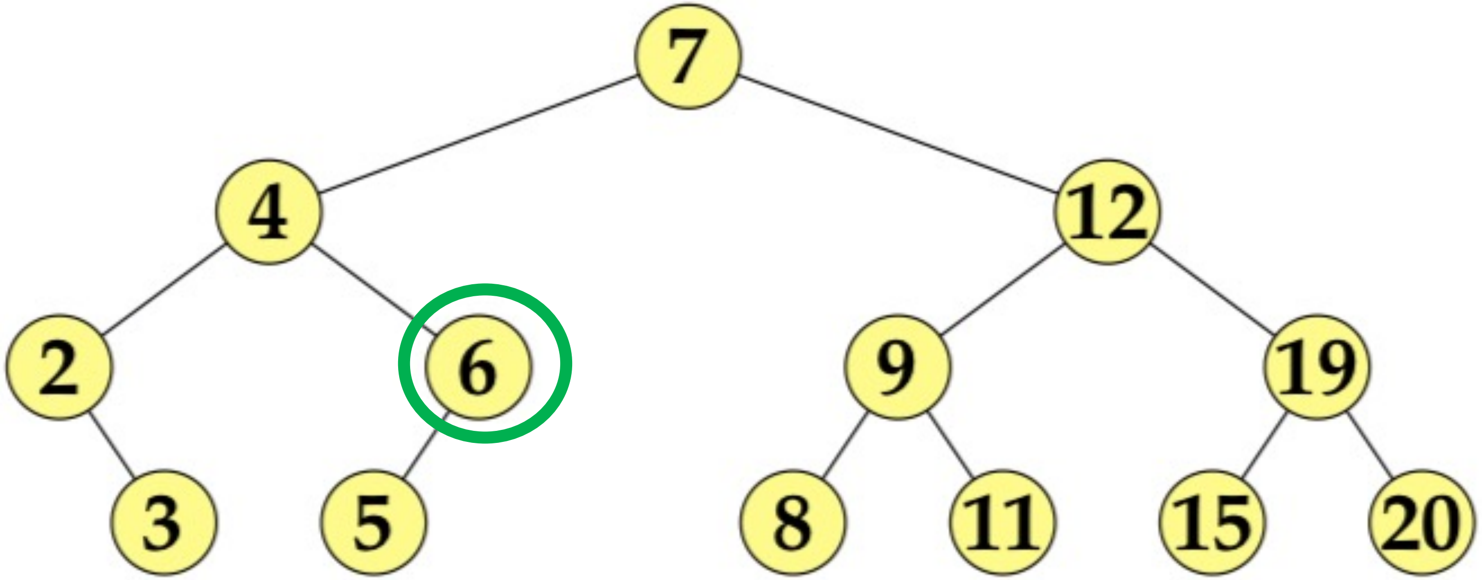


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6

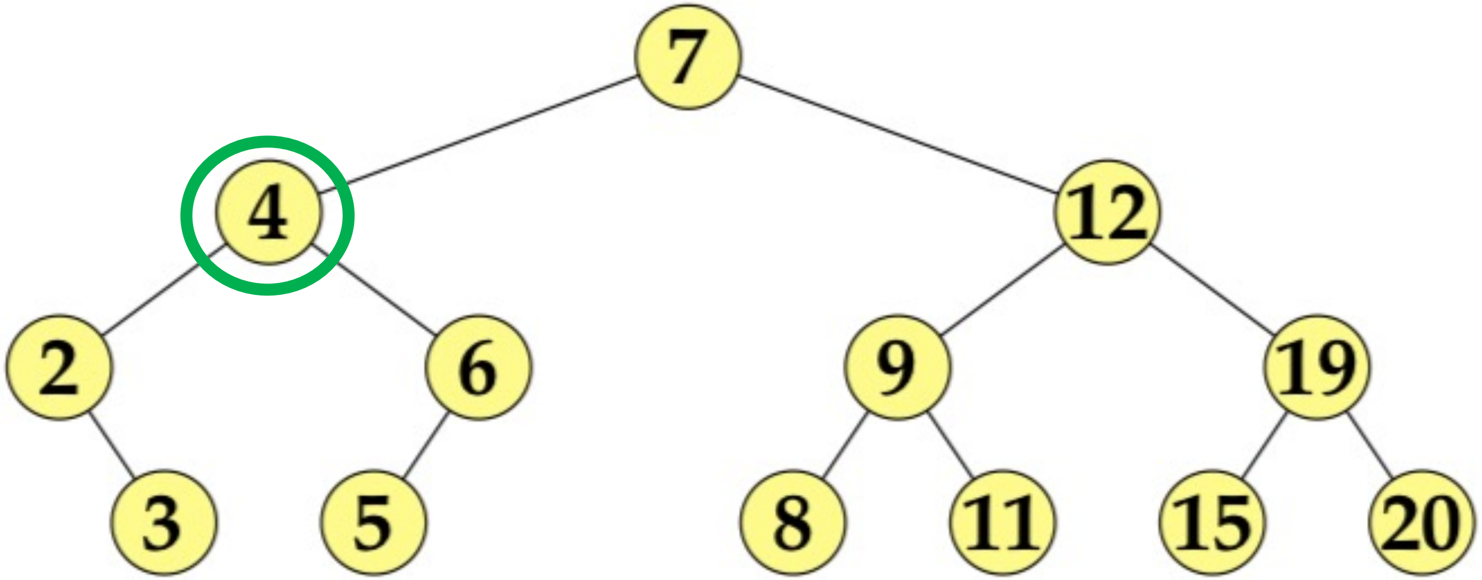


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6

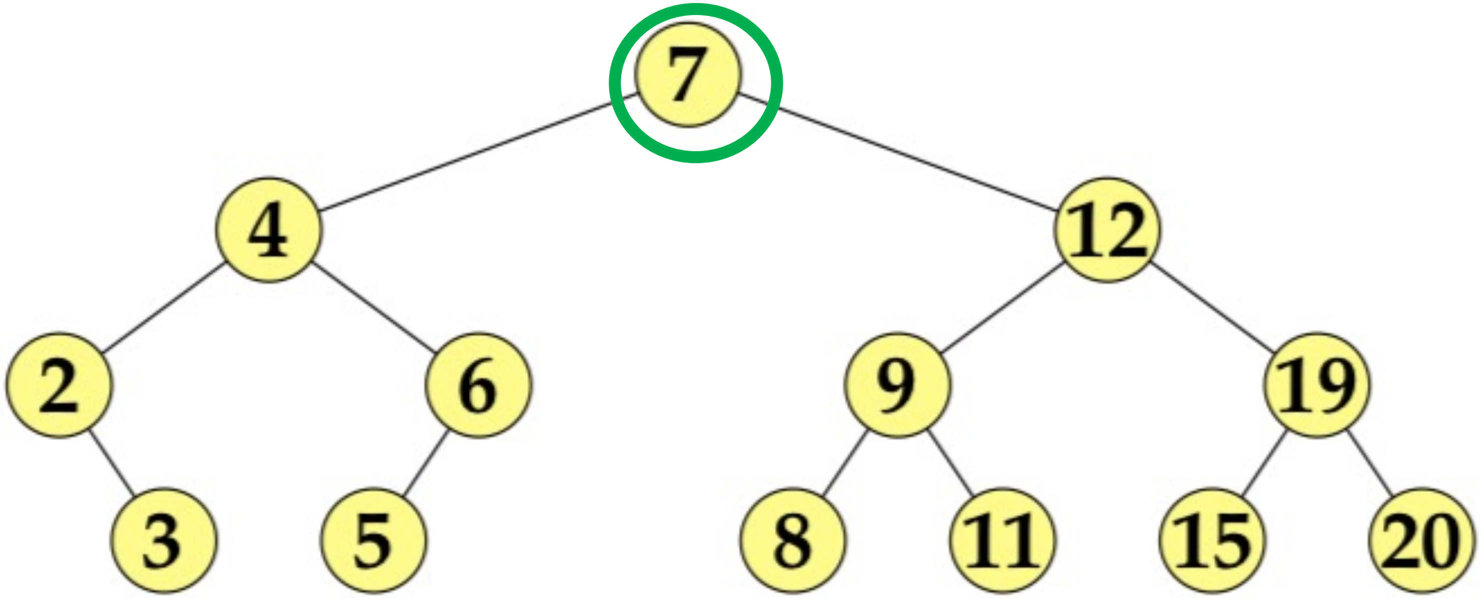


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6

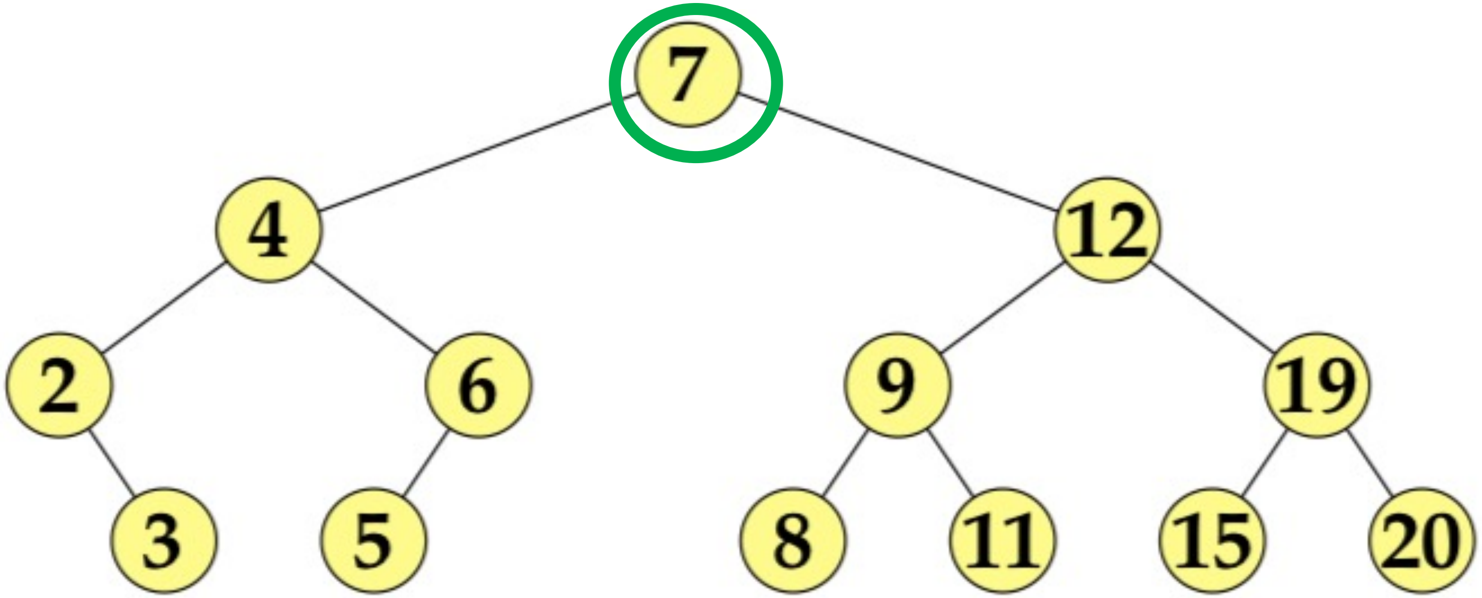


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7

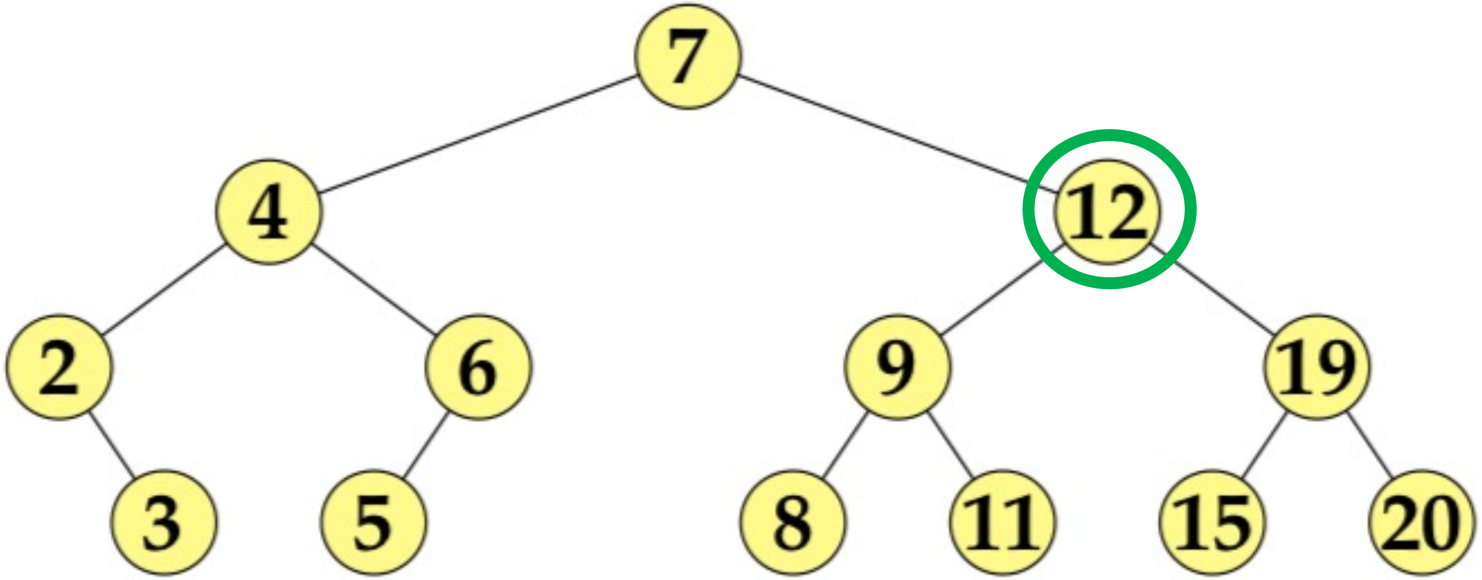


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7

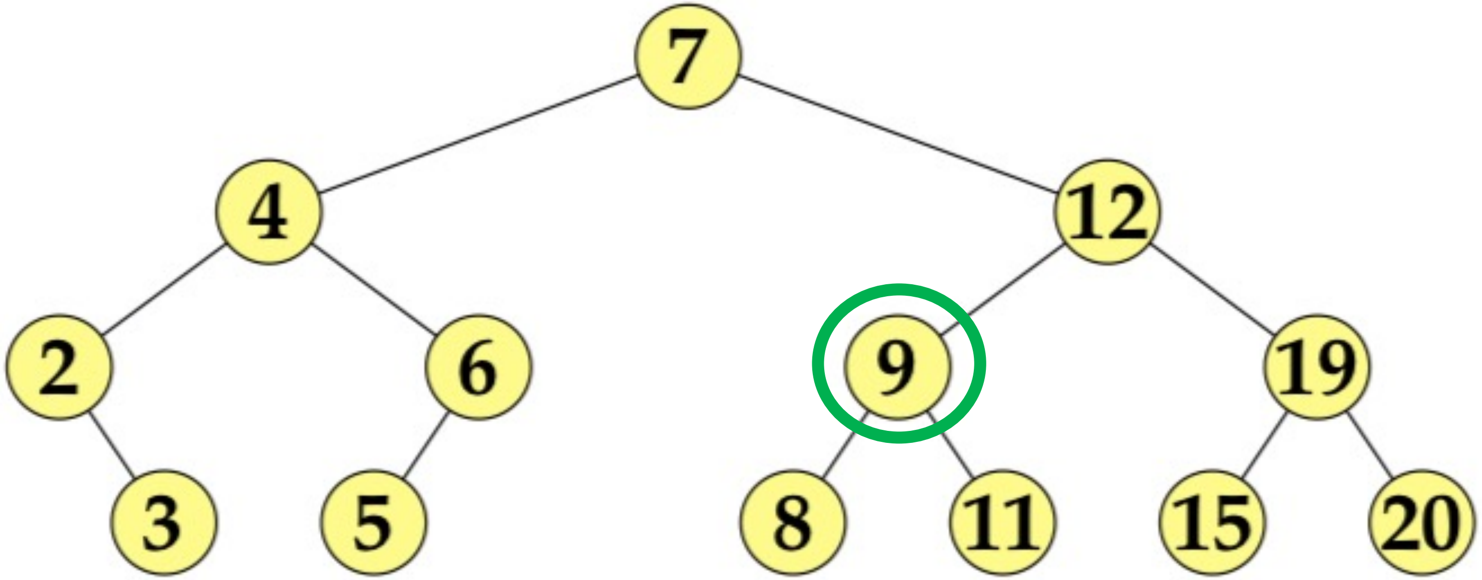


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7

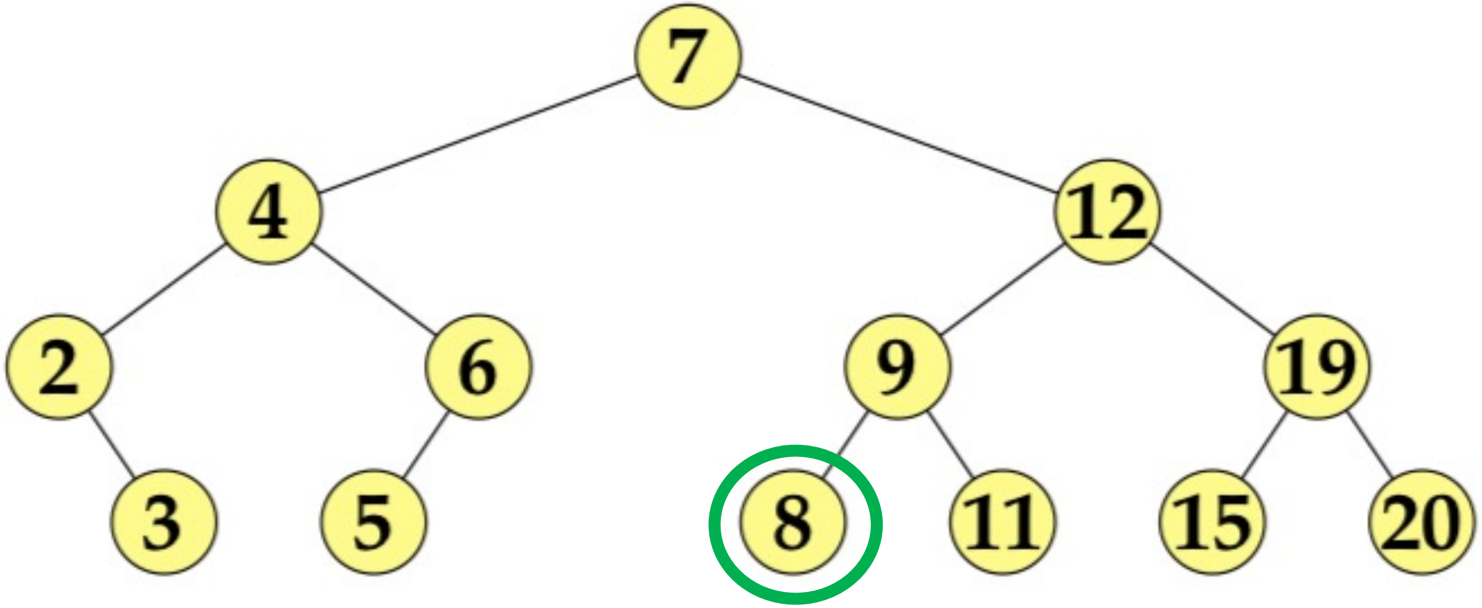


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7

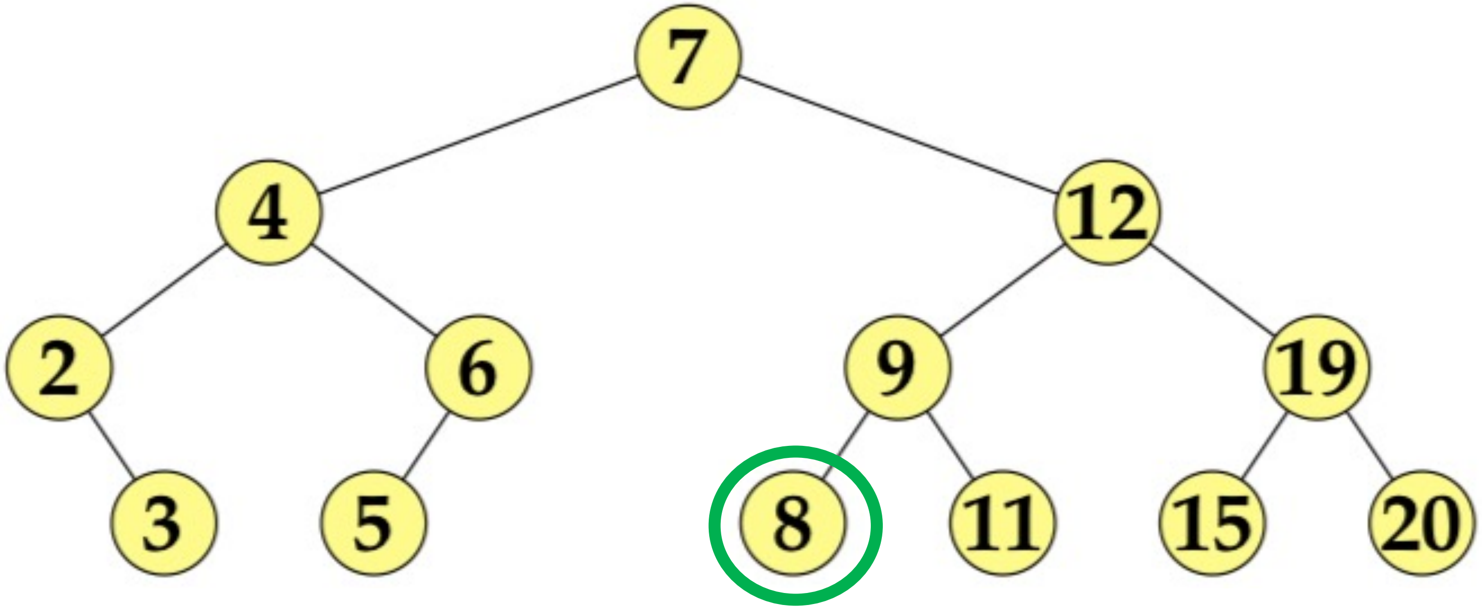


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8

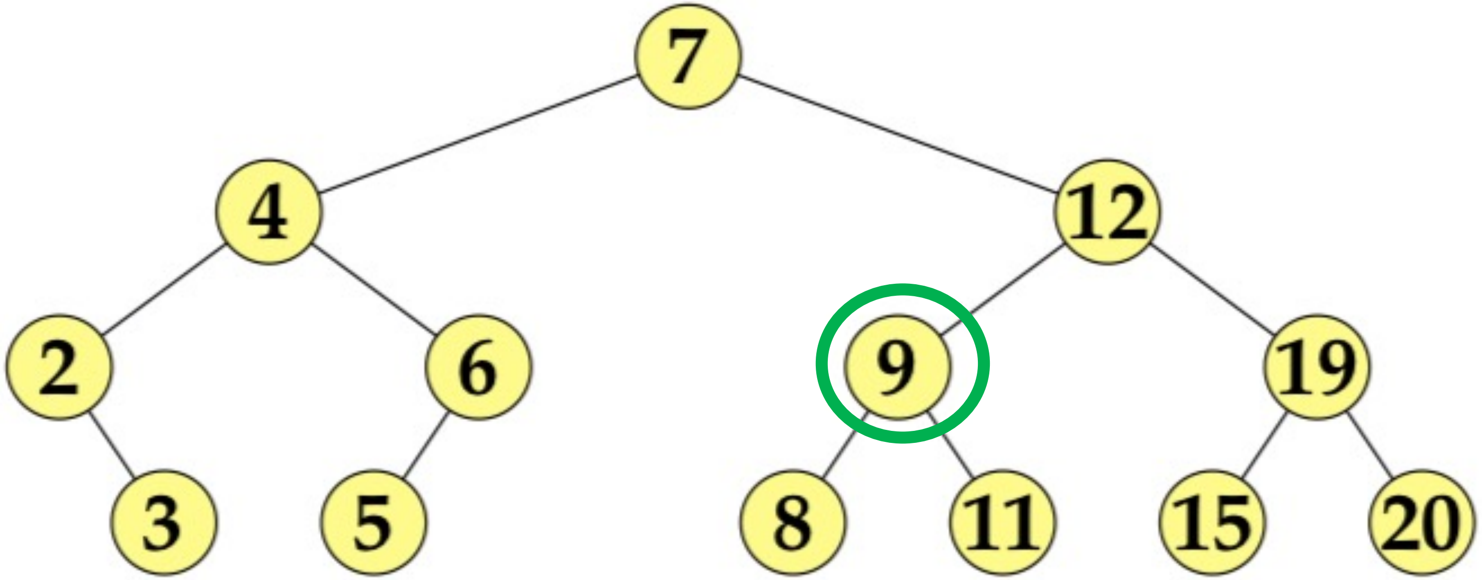


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8

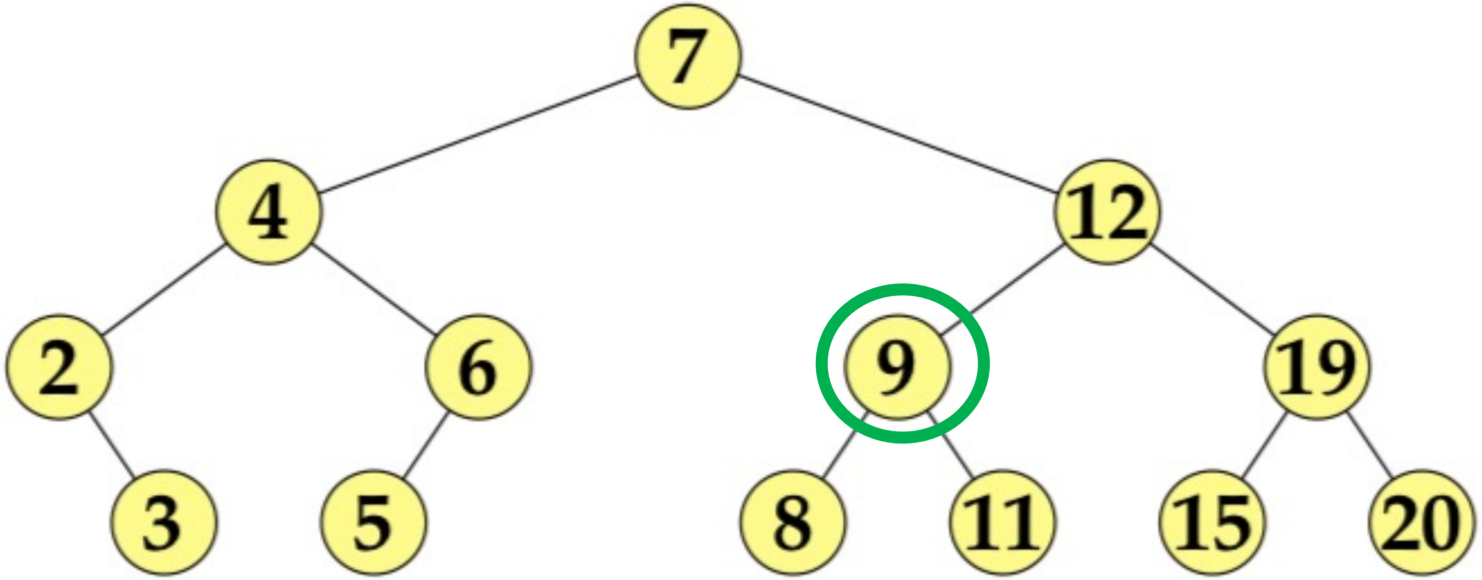


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9

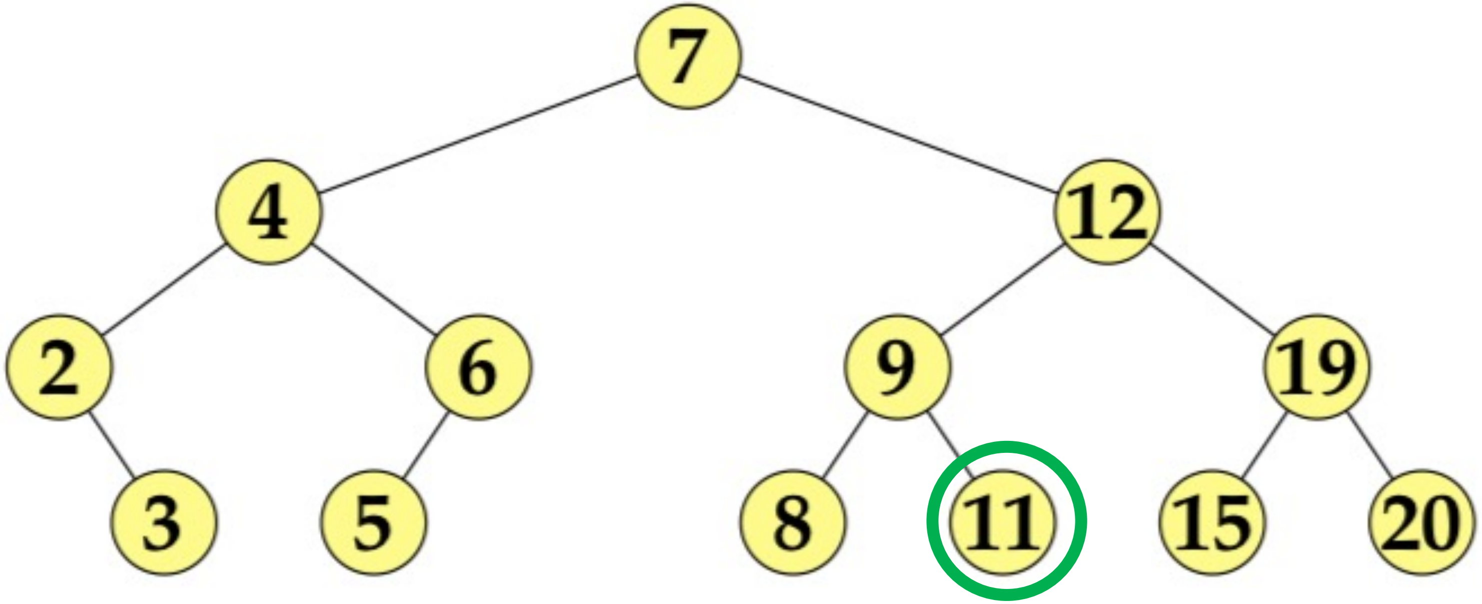


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9

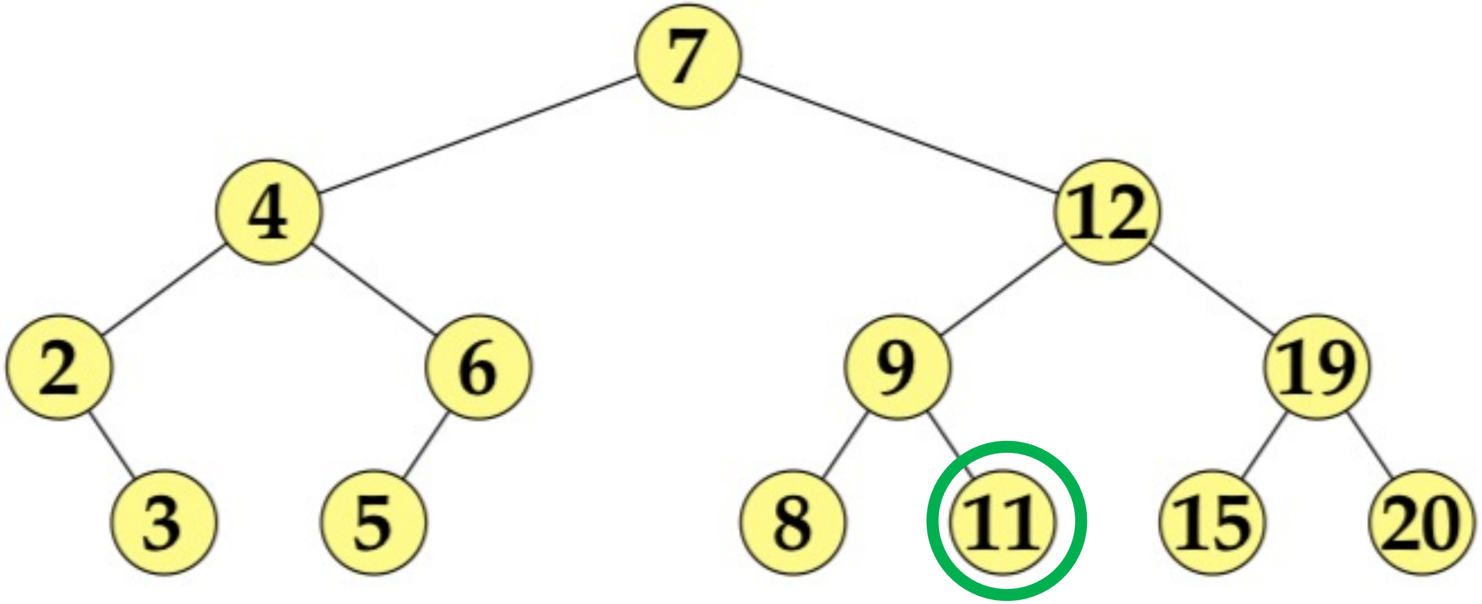


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11

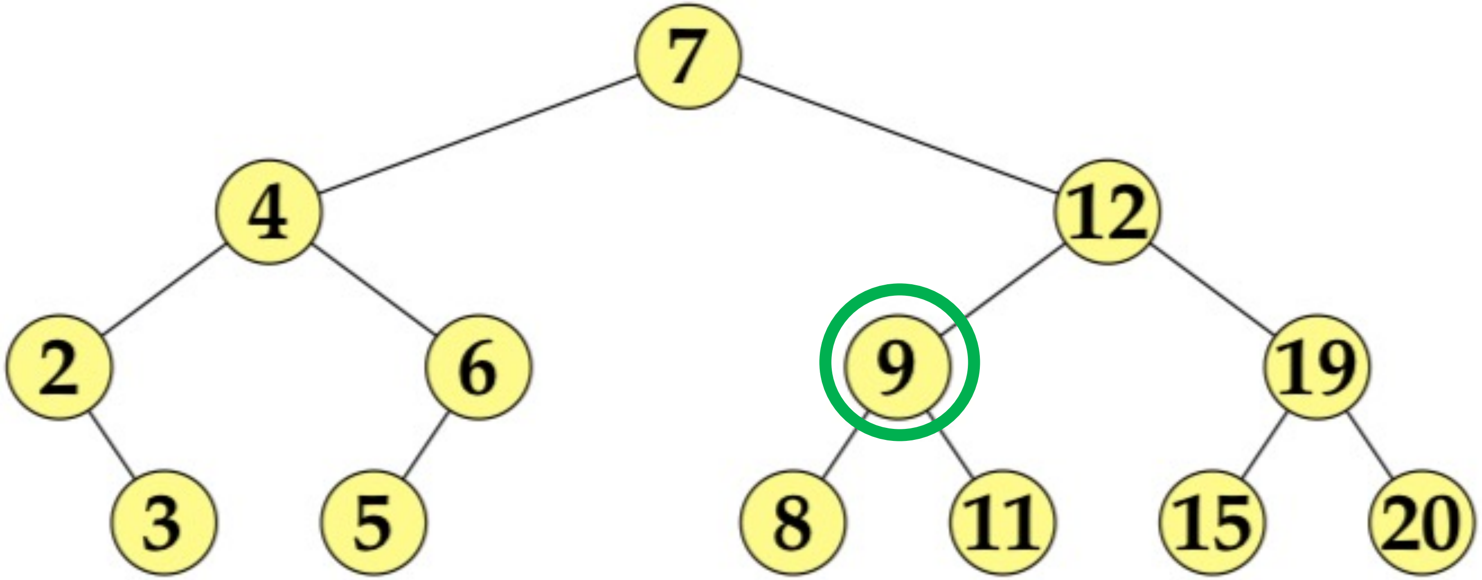


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11

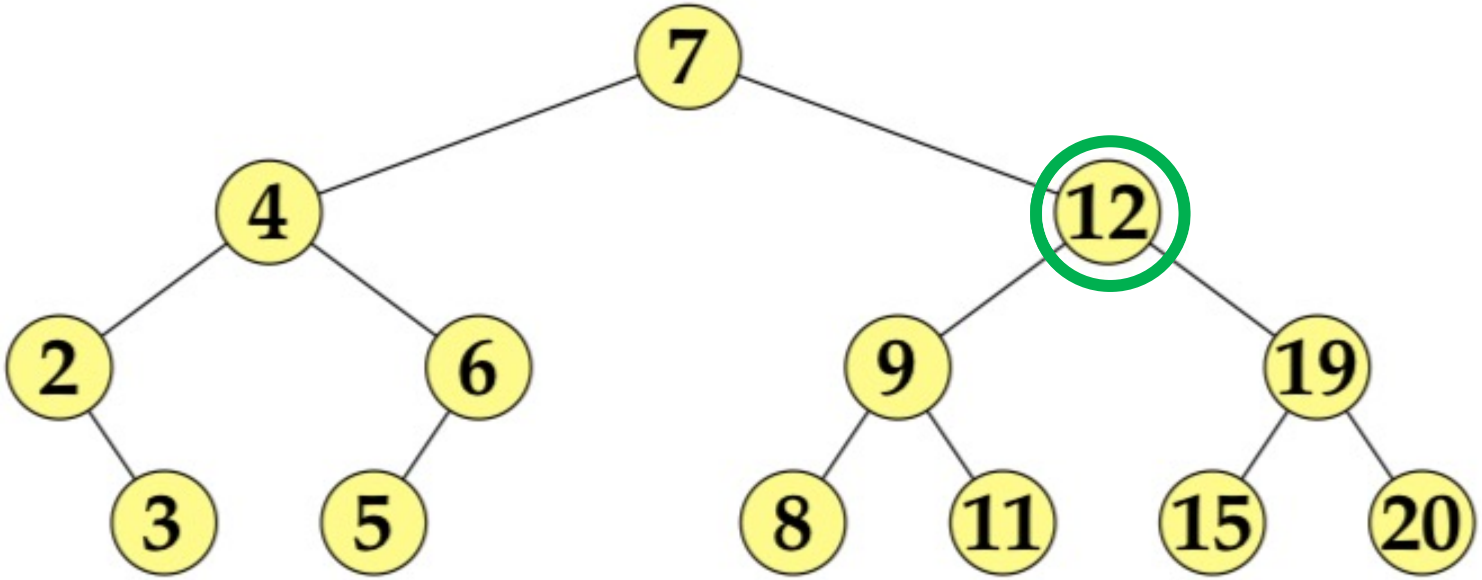


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11

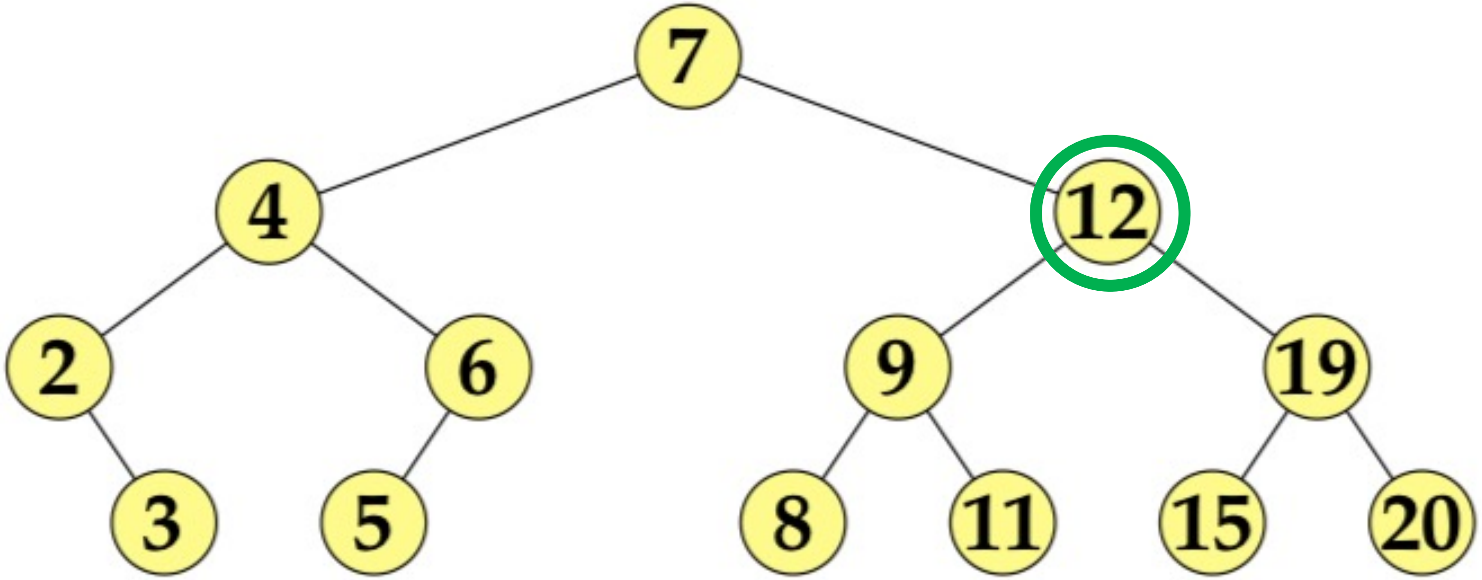


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12

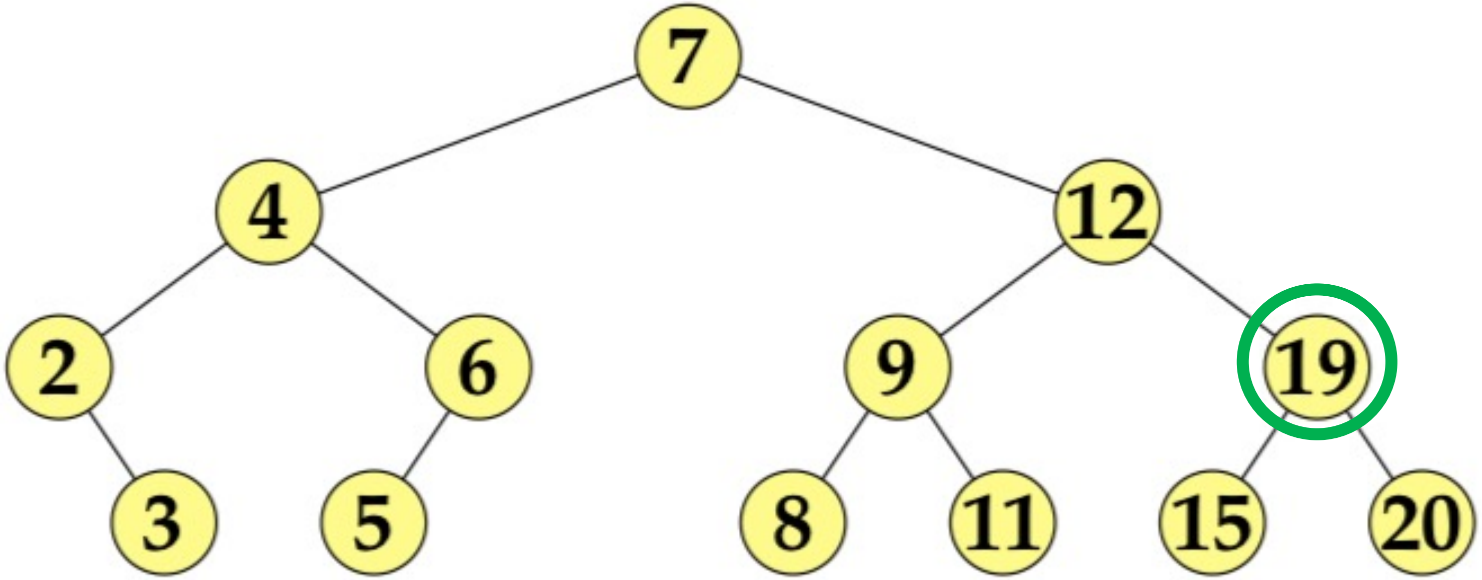


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12

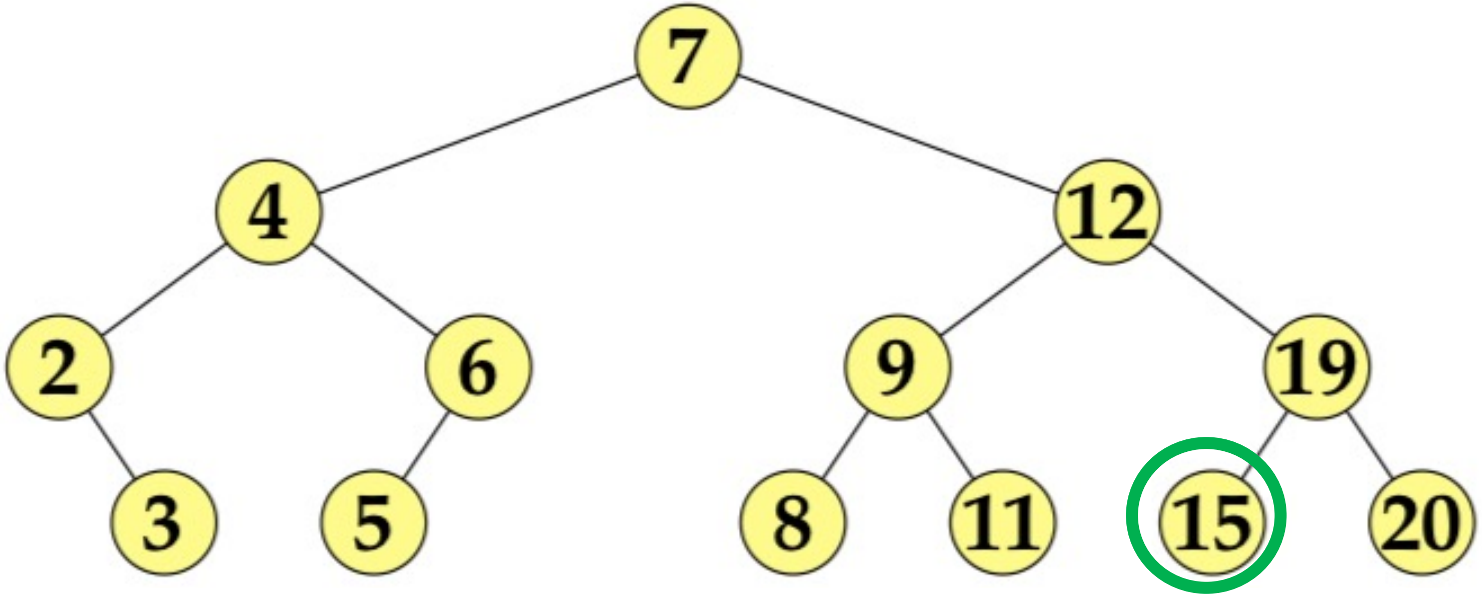


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12

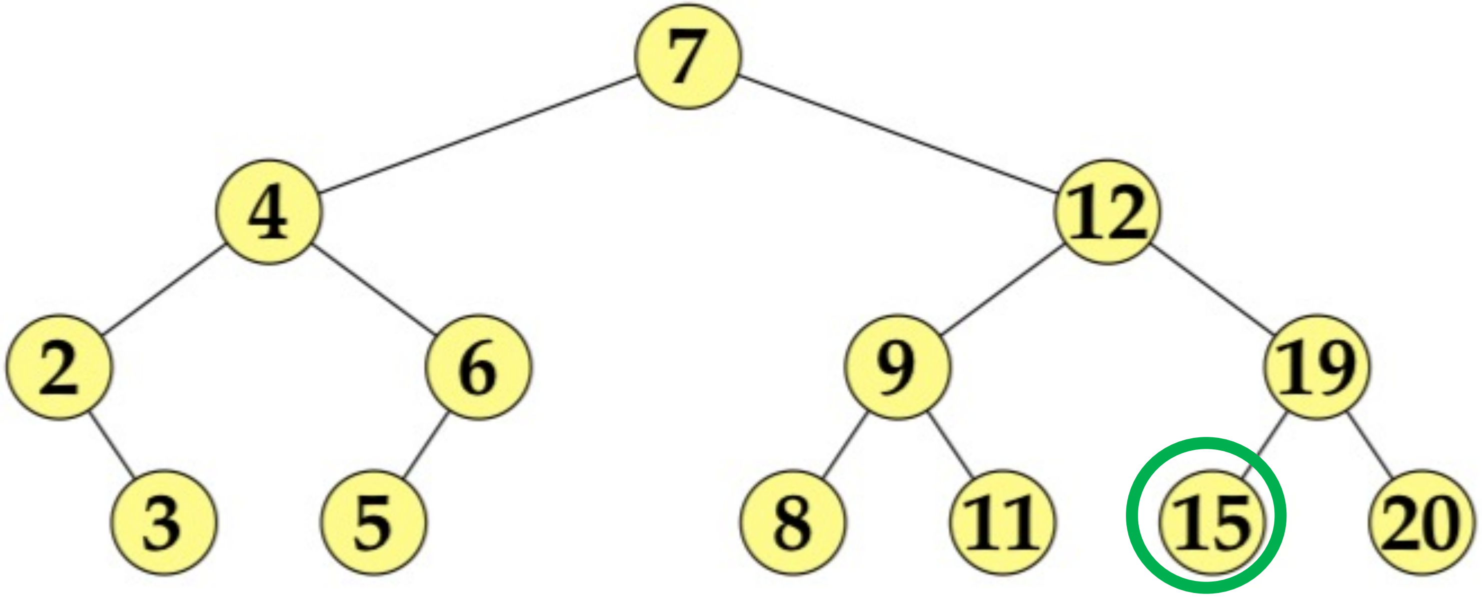


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15

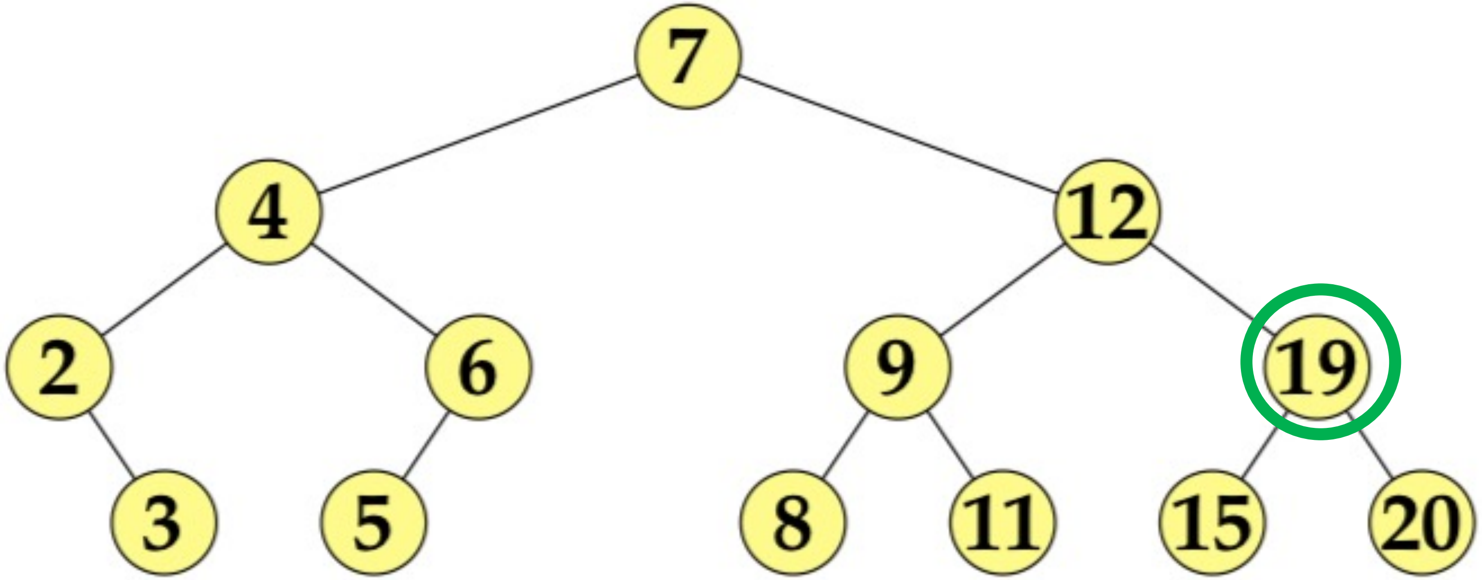


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15

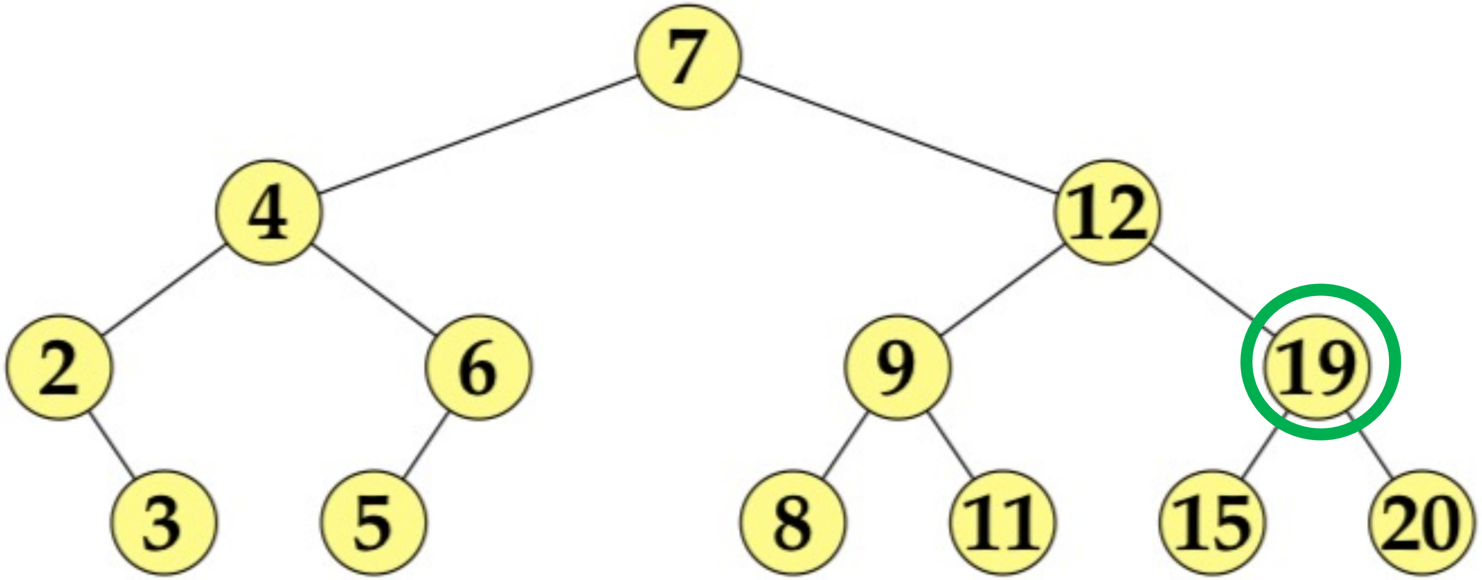


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, *19*

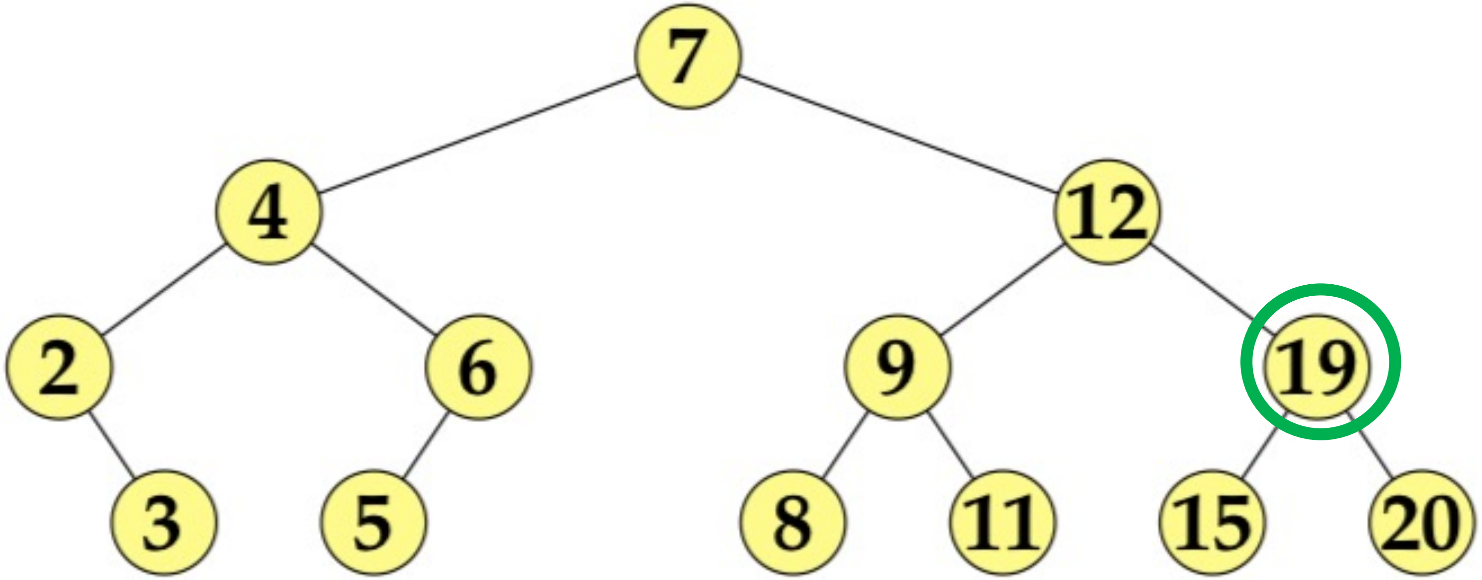


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19

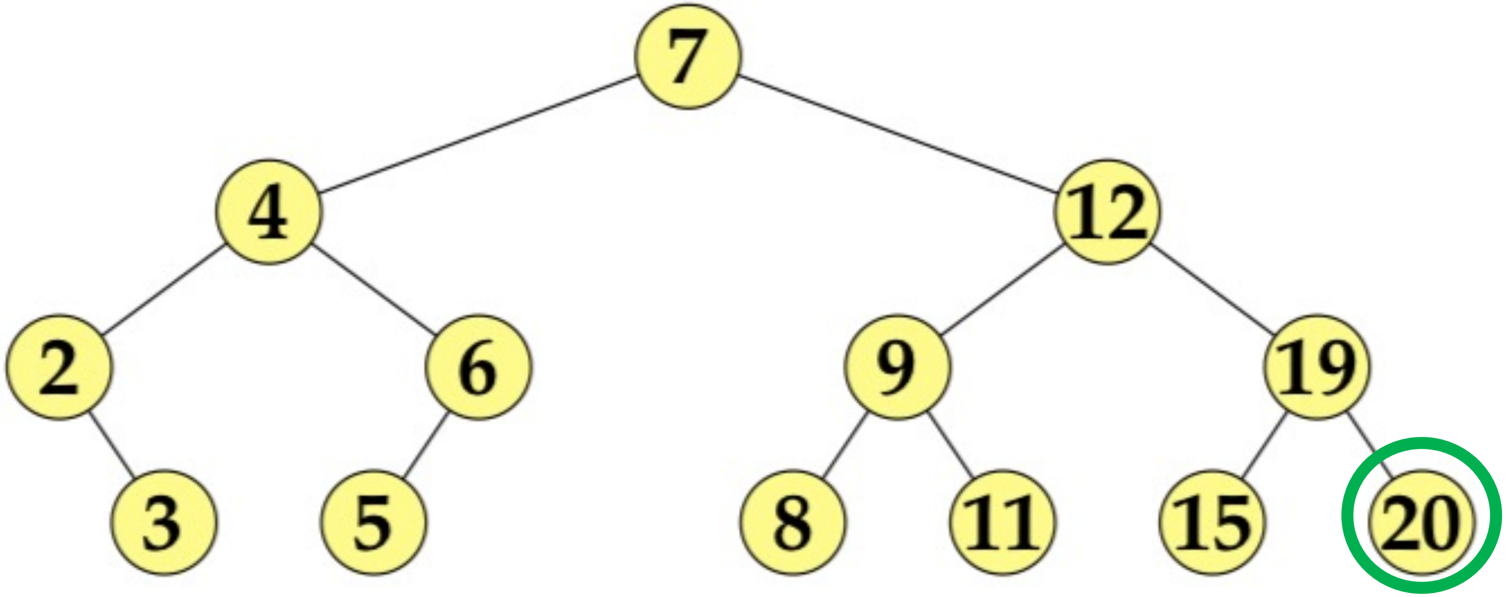


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19

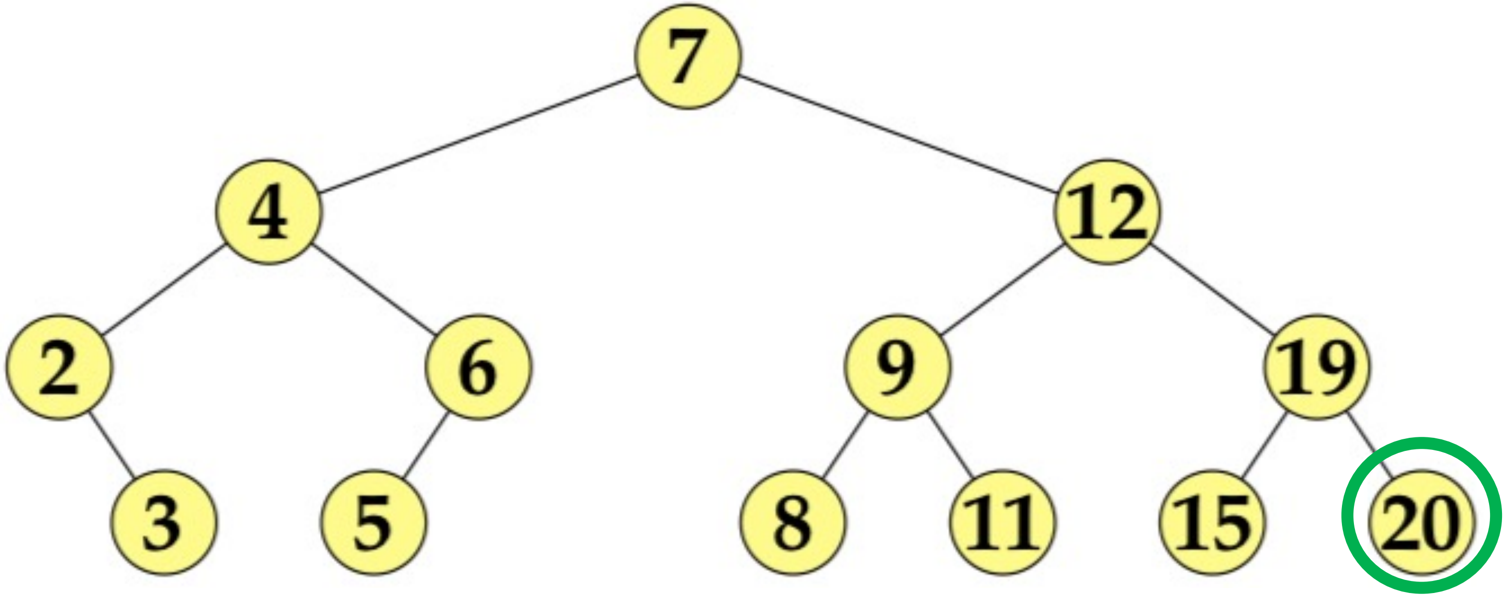


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19, 21

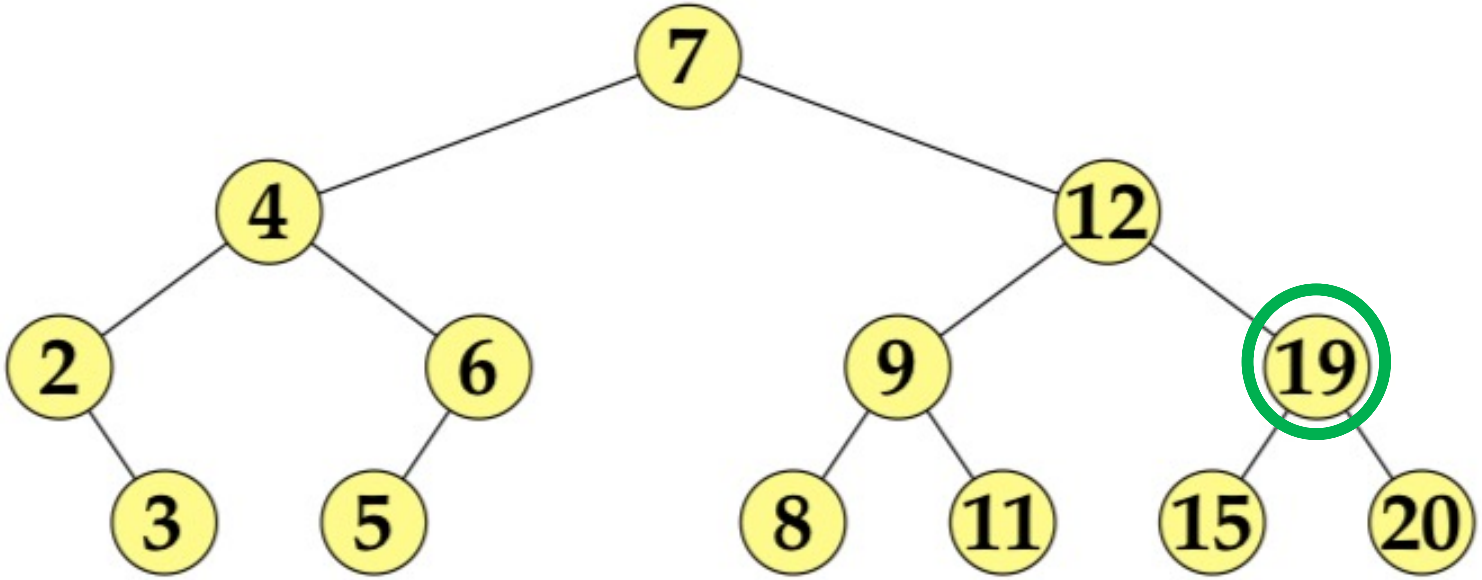


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19, 21

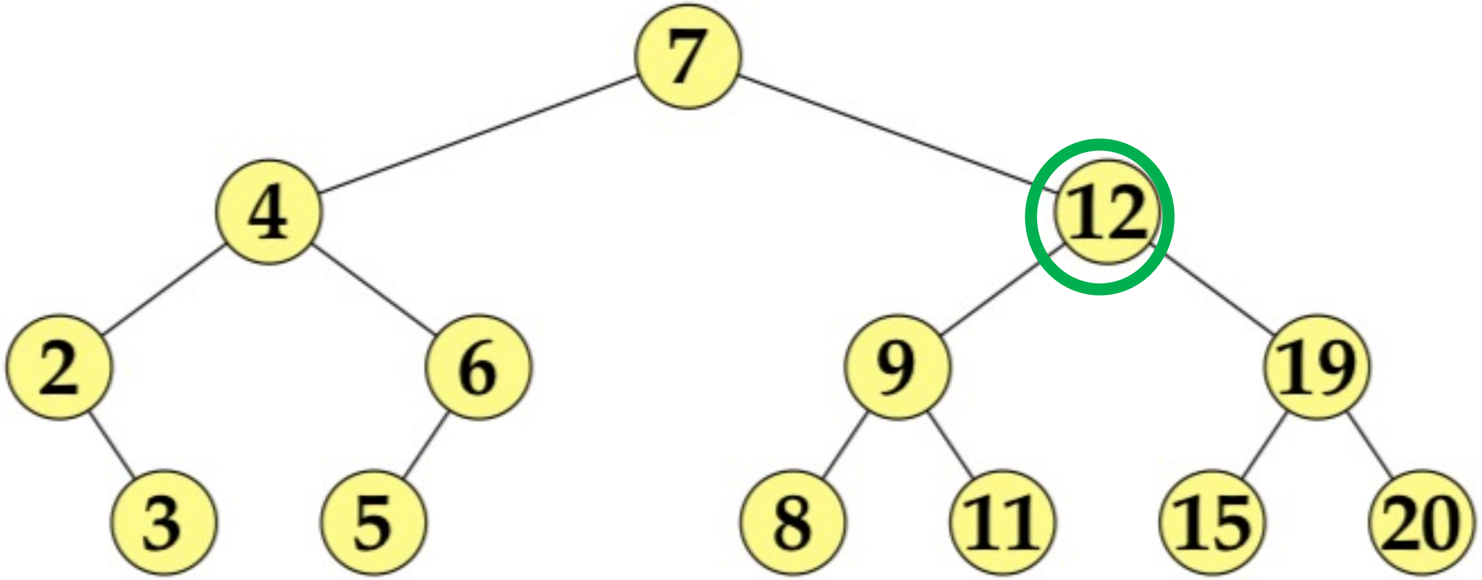


Inorder

What would the in-order traversal be here?

left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19, 21

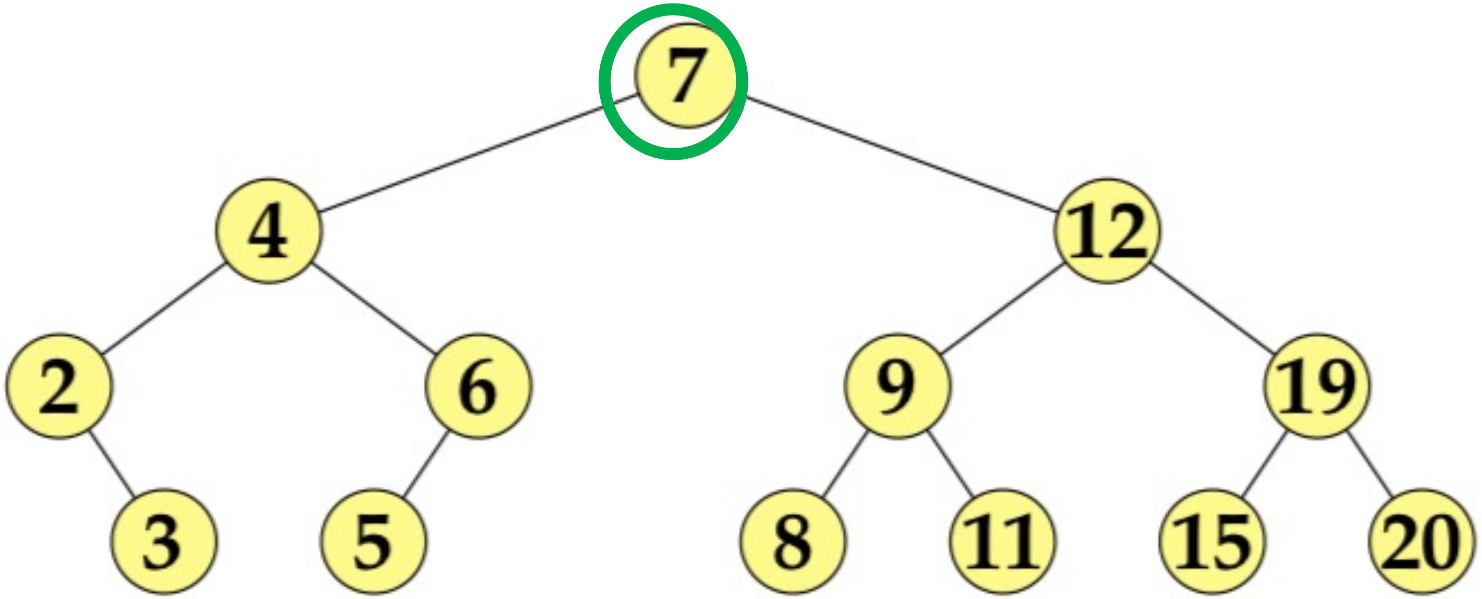


Inorder

What would the in-order traversal be here?

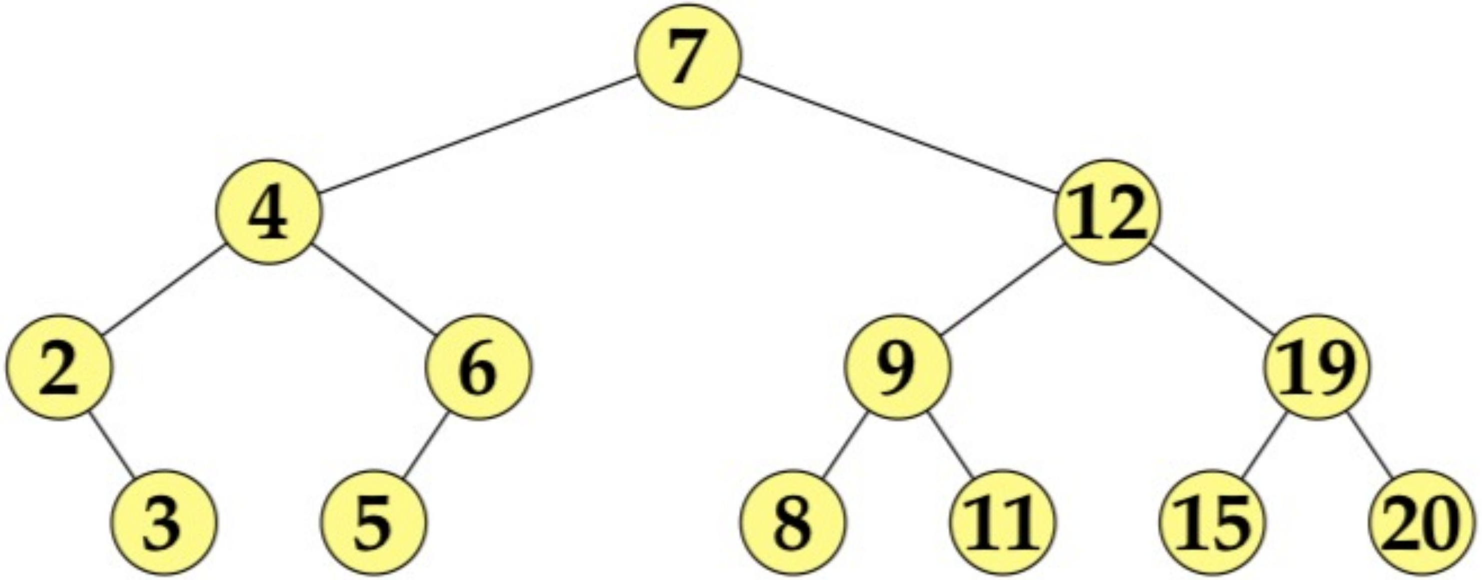
left subtree, current, right subtree

2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 19, 21



Inorder

- 2, 3, 4, 5, 6, 7, 8 , 9, 11, 12, 15, 19, 20

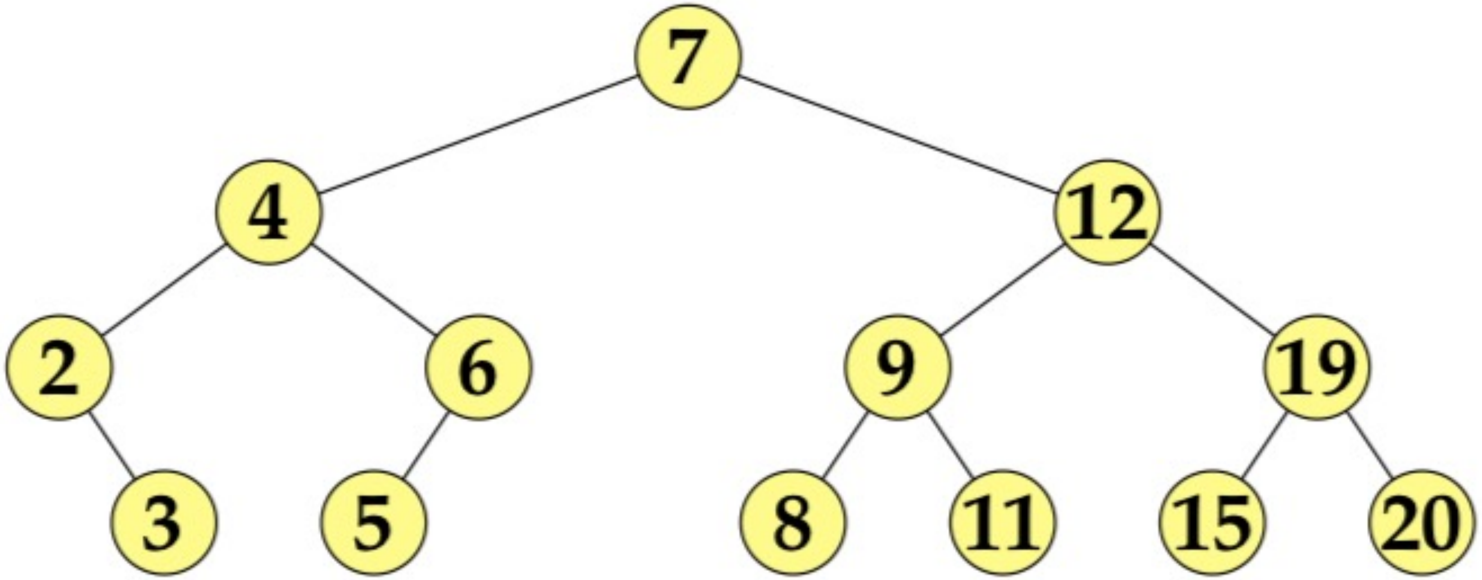


inorder

```
inOrderRec(root):
  if root != null:
    inOrderRec(root.left)
    visit(root)
    inOrderRec(root.right)
```

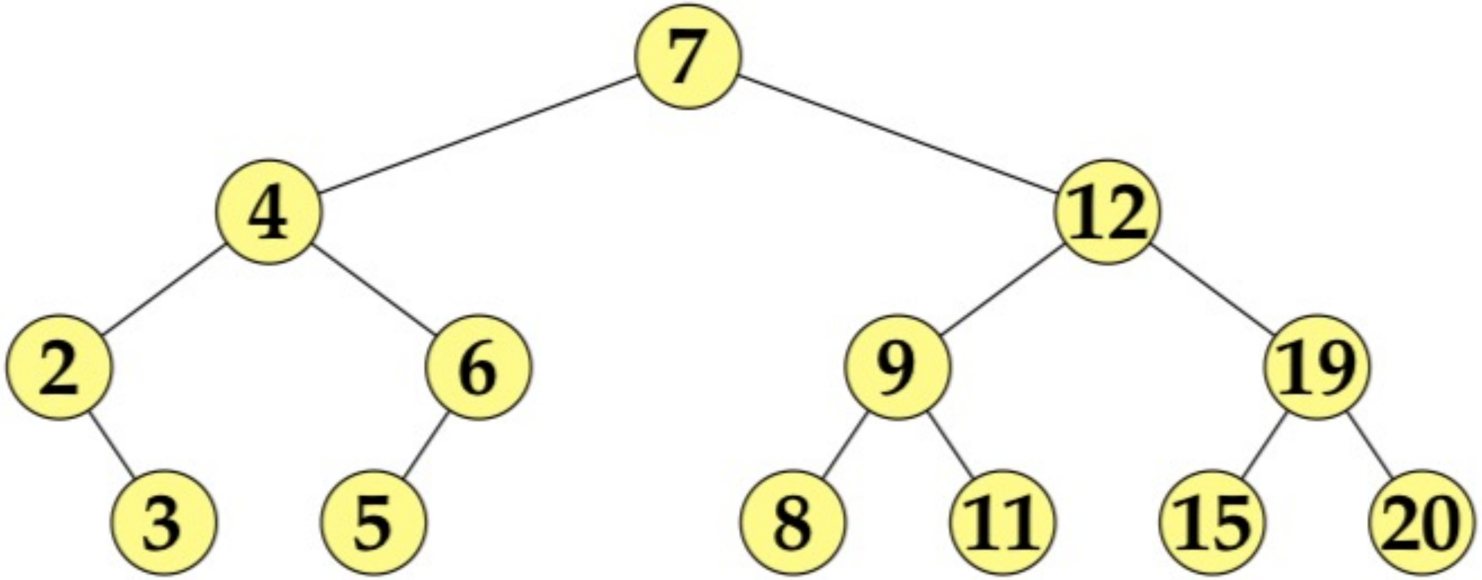
Preorder

Current, left right,



Preorder

- 7, 4, 2, 3, 6, 5, 12, 9, 8, 11, 19, 15, 20



preorder

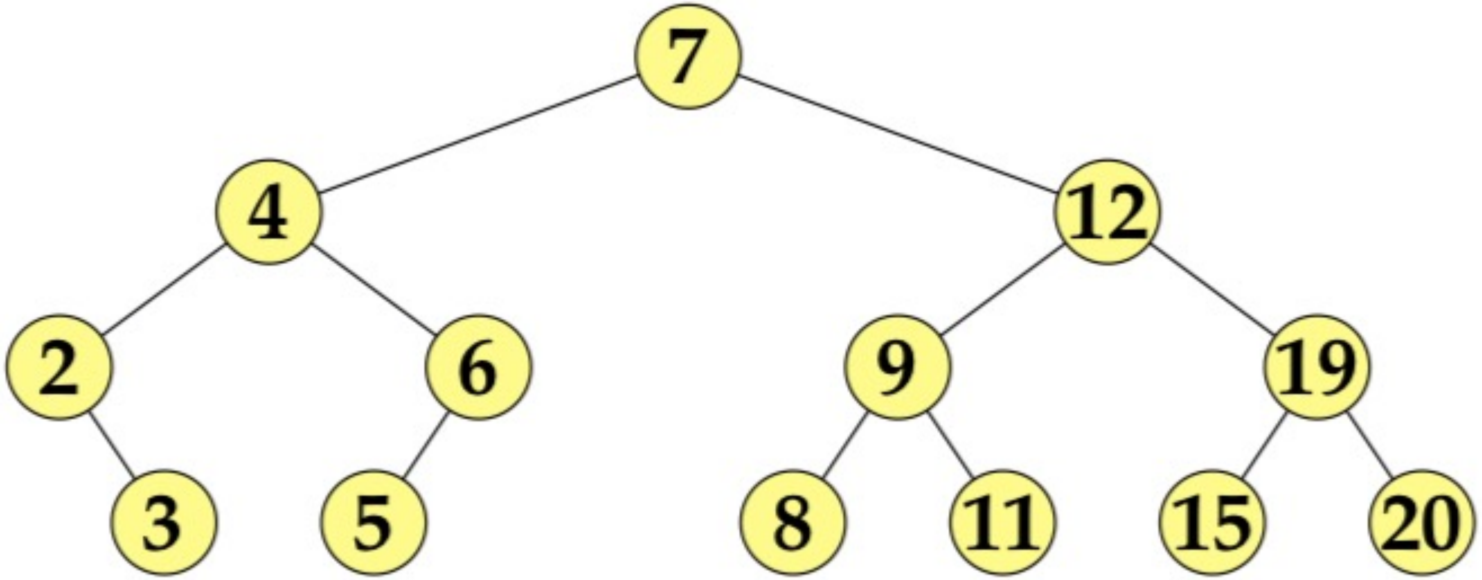
```
inOrderRec(root):
  if root != null:
    inOrderRec(root.left)
    visit(root)
    inOrderRec(root.right)
```

preorder

```
preOrderRec(root):
  if root != null:
    visit(root)
    preOrderRec(root.left)
    preOrderRec(root.right)
```

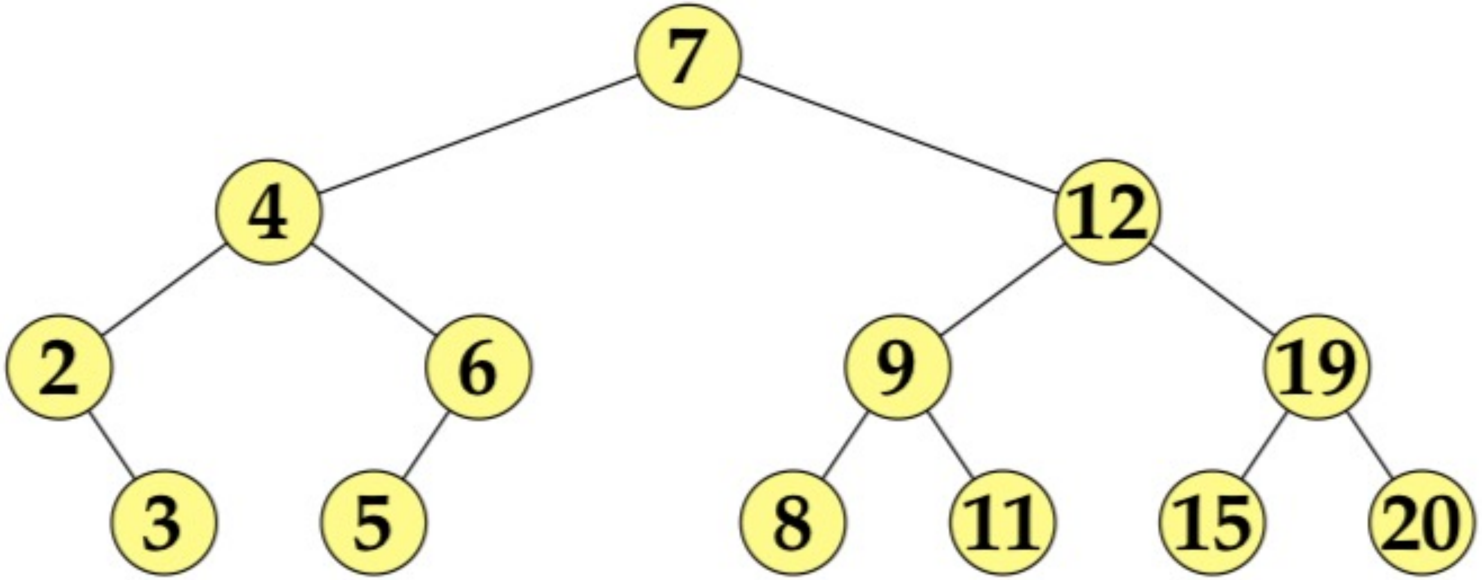
Postorder

Left, right, current



Postorder

- 3, 2, 5, 6, 4, 8, 11, 9, 15, 20, 19, 12, 7



postorder

```
inOrderRec(root):
  if root != null:
    inOrderRec(root.left)
    visit(root)
    inOrderRec(root.right)
```

postorder

```
postOrderRec(root):
  if root != null:
    postOrderRec(root.left)
    postOrderRec(root.right)
    visit(root)
```

Interface

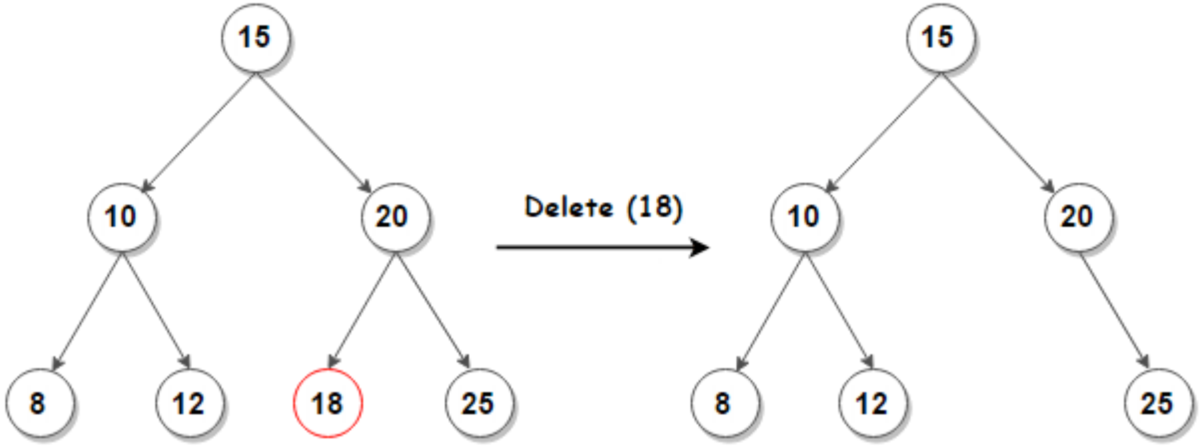
```
public interface BinaryTree<E extends
Comparable<E>> {
  E getRootElement();
  int size();
  boolean isEmpty();
  void insert(E element);
  boolean contains(E element);
  boolean remove(E element);
  String toStringInOrder();
  String toStringPreOrder();
  String toStringPostOrder();
}
```

Remove

- `boolean remove(E element);`
- returns true if element existed and was removed and false otherwise
- Cases
  - element not in tree
  - element is a leaf
  - element has one child
  - element has two children

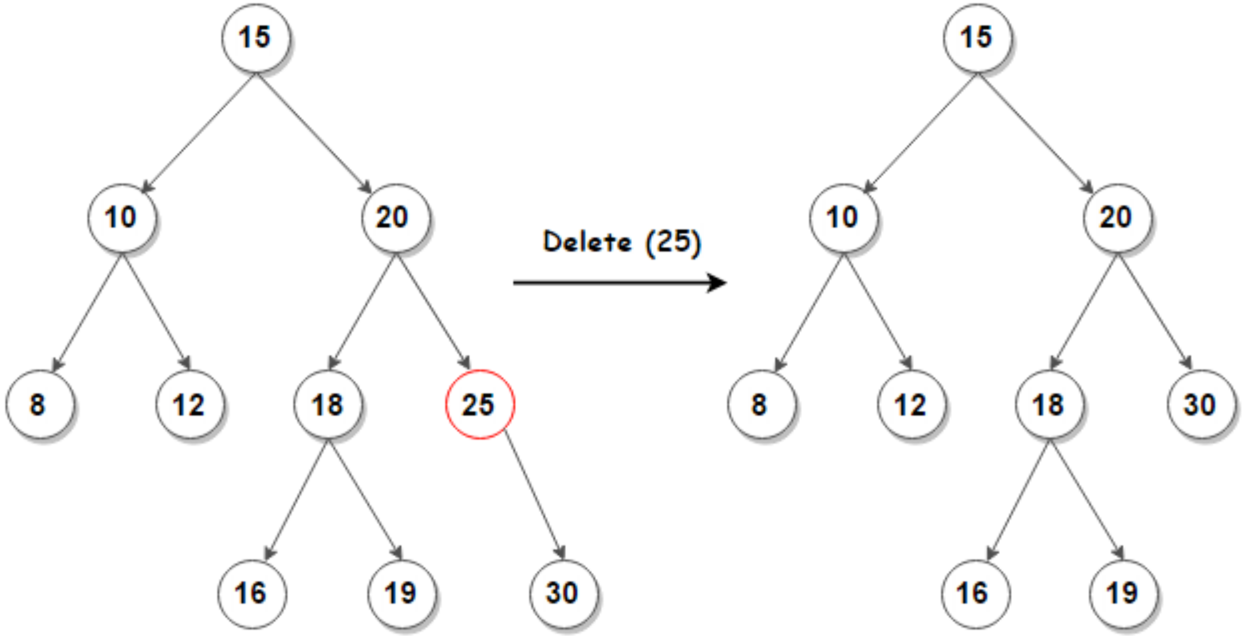
Leaf

- Just delete



One child

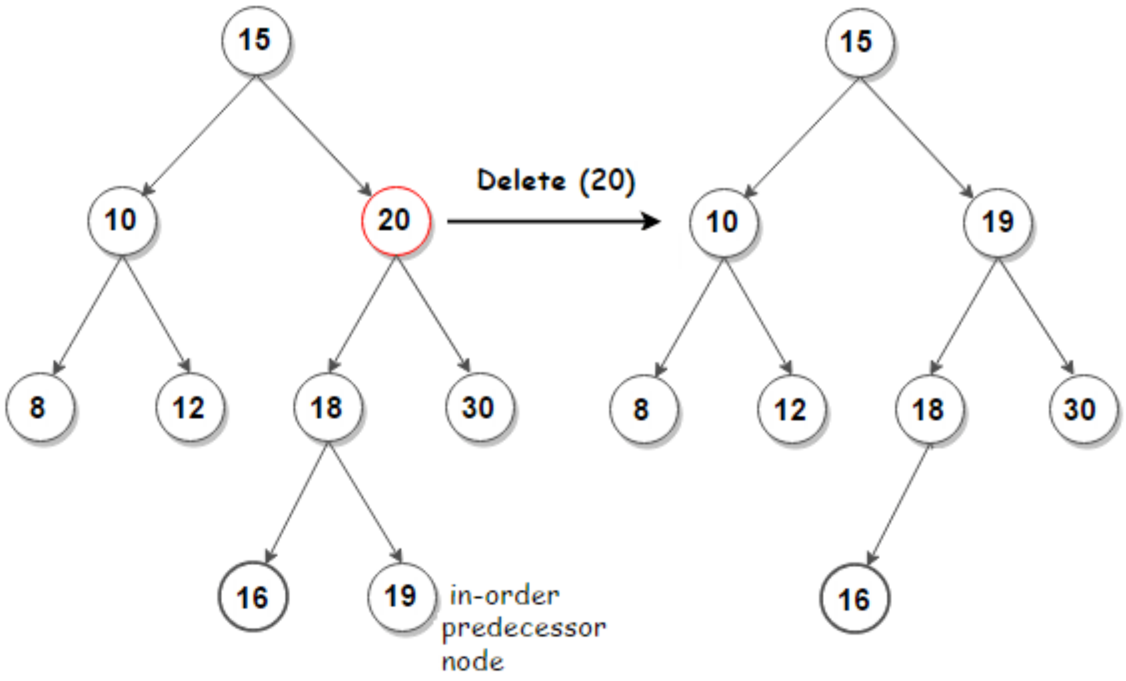
- Replace with child – skip over like in linked list



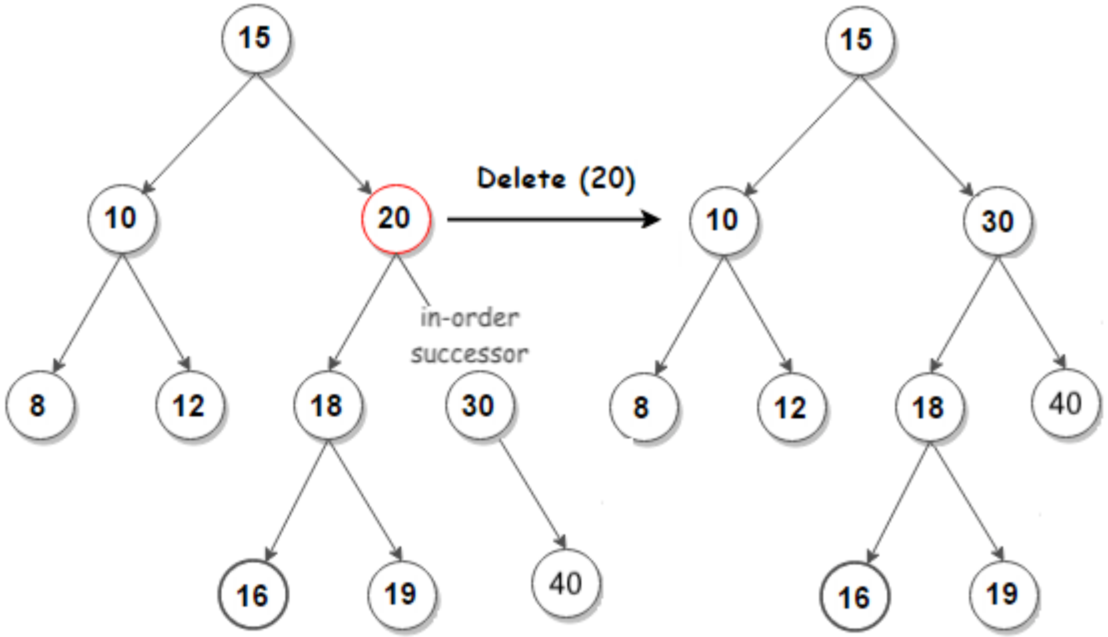
Two Children

- Replace with in-order predecessor or in-order successor
- in-order predecessor
  - rightmost child in left subtree
  - max-value child in left subtree
- in-order successor
  - leftmost child in right subtree
  - min-value child in right subtree

Replace with Predecessor



Replace with Successor



Pseudo code

```
minKey(root):
  if root.left == null:
    return root.key
  else
    return minKey(root.left)
```

```
removeRec(root, key):
  if root == null:
    return null
  if root.key > key:
    root.left = removeRec(root.left, key)
  else if root.key < key:
    root.right = removeRec(root.right, key)
  else
    if root.left == null:
      return root.right
    else if root.right == null:
      return root.left
    else
      root.key = minKey(root.right)
      root.right = removeRec(root.right, root.key)
  return root
```

Performance of BST

	BST balanced	BST worst
search		
insert		
remove		
min/max		

	Unsorted array	Sorted array	Unsorted list	Sorted list
search	$O(n)$	$O(logn)$	$O(n)$	$O(n)$
insert	$O(1)^*$	$O(n)$	$O(1)$	$O(n)$
remove	$O(1)^*$	$O(n)$	$O(1)$	$O(1)$
min/max	$O(n)$	$O(1)$	$O(n)$	$O(1)$

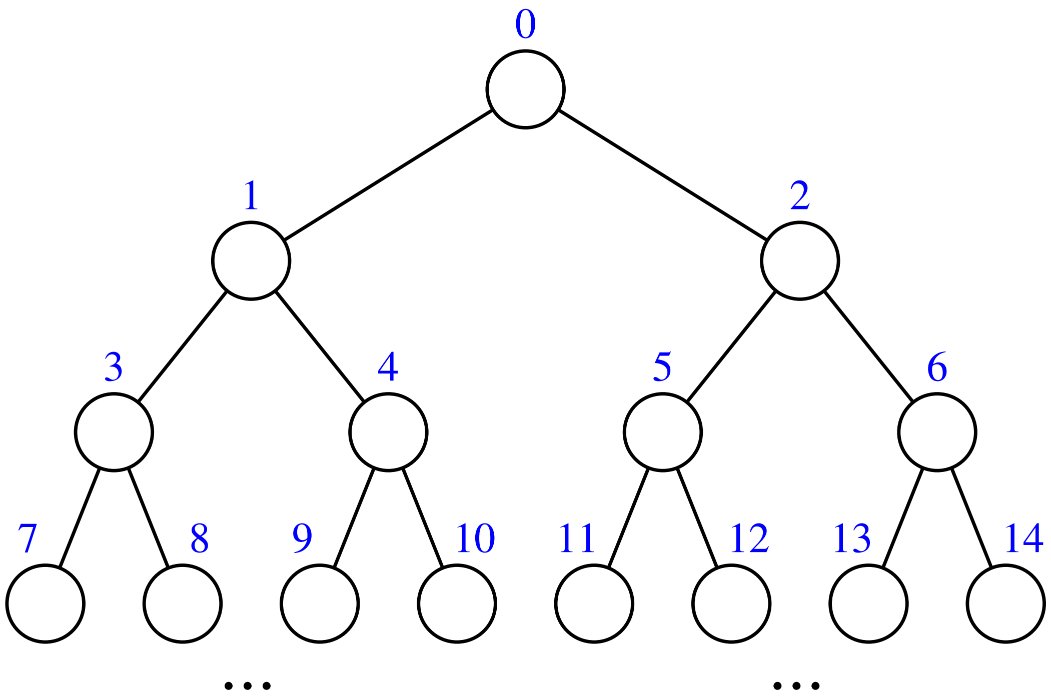
Performance of BST

	BST balanced	BST worst
search	$O(logn)$	$O(n)$
insert	$O(logn)$	$O(n)$
remove	$O(logn)$	$O(n)$
min/max	$O(logn)$	$O(n)$

	Unsorted array	Sorted array	Unsorted list	Sorted list
search	$O(n)$	$O(logn)$	$O(n)$	$O(n)$
insert	$O(1)^*$	$O(n)$	$O(1)$	$O(n)$
remove	$O(1)^*$	$O(n)$	$O(1)$	$O(1)$
min/max	$O(n)$	$O(1)$	$O(n)$	$O(1)$

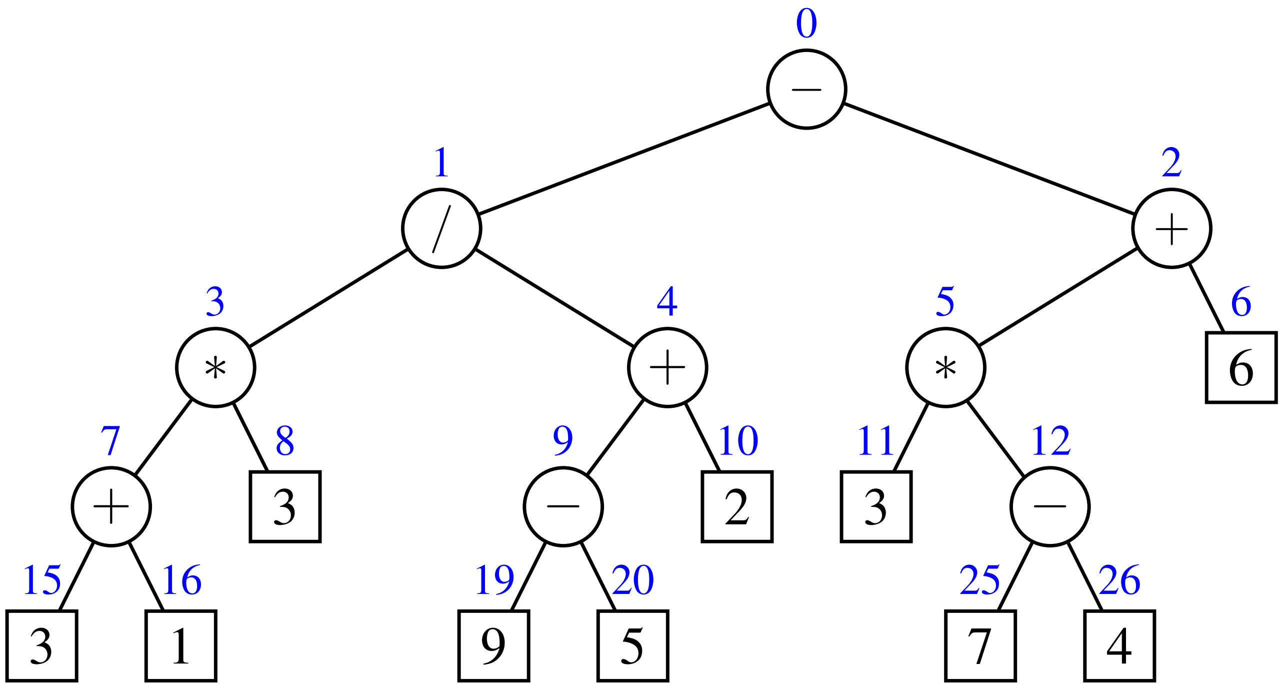
Array-based Binary Tree

- Number nodes level-by-level, left-to-right
- $f(root) = 0$
- $f(l) = 2f(p) + 1$
- $f(r) = 2f(p) + 2$
- Numbering is based on all positions, not just occupied positions



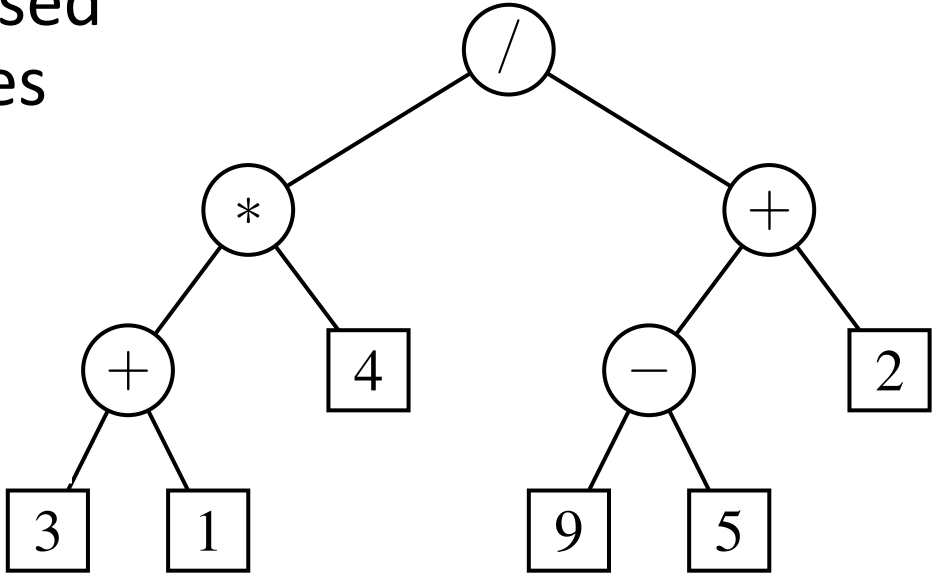
Level-numbering

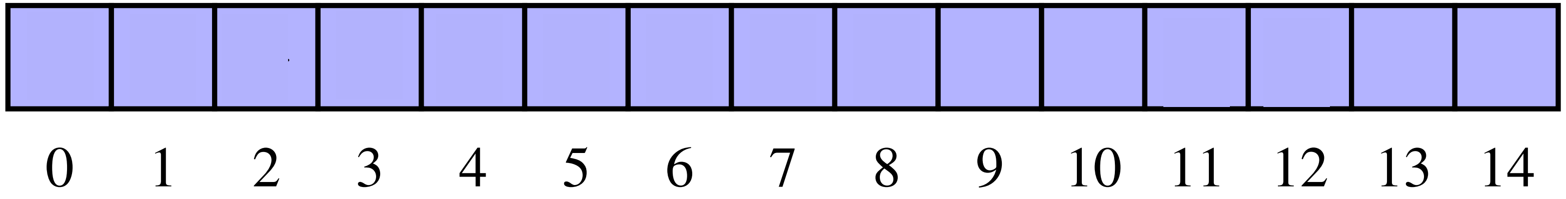
- Numbering is based on all positions, not just occupied positions



Array-based Binary Tree

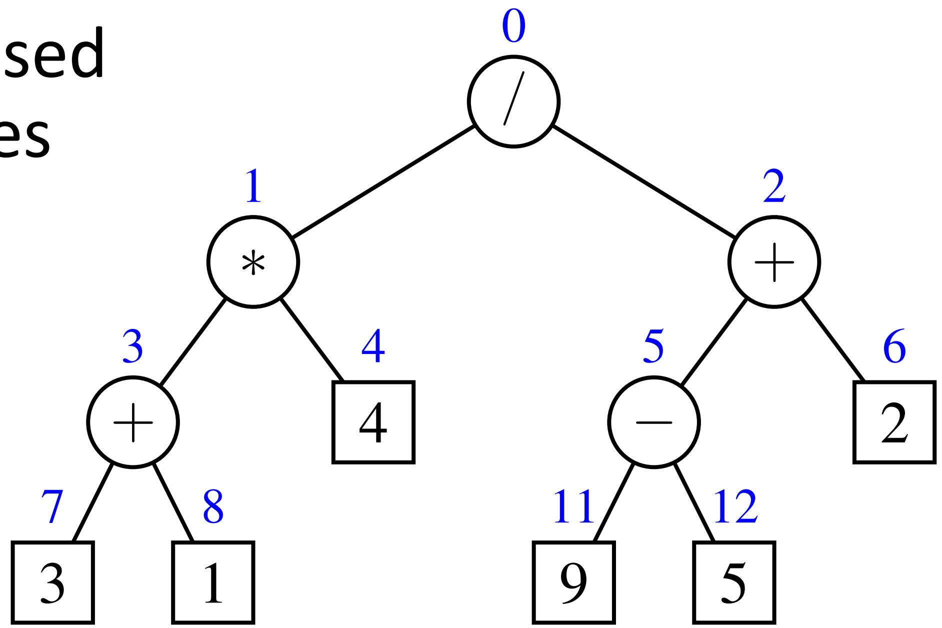
- The numbering can then be used as indices for storing the nodes directly in an array
- $f(root) \ = \ 0$
- $f(l) = \ 2f(p) + 1$
- $f(r) \ = \ 2f(p) + 2$





Array-based Binary Tree

- The numbering can then be used as indices for storing the nodes directly in an array



/	*	+	+	4	−	2	3	1			9	5		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The Textbook’s Way

- `Tree` interface extends `Iterable`
  - positional and iterable
  - accessor, query and general methods
- `AbstractTree` implements `Tree`
  - abstract base class – concrete implementations of some of the methods
- `BinaryTree` interface extends `Tree`

The Textbook's Way

- `AbstractBinaryTree` base class extends `AbstractTree` implements `BinaryTree`

- `LinkedBinaryTree` extends `AbstractBinaryTree`

- 7 classes!