61A Lecture 21

Announcements

Binary Trees

Binary Tree Class

Binary Tree Class

```
class BTree(Tree):
```

Binary Tree Class

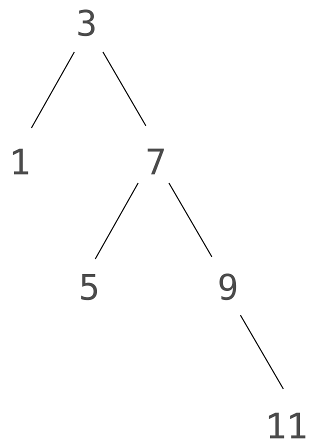
A binary tree is a tree that has a left branch and a right branch

```
class BTree(Tree):
```

Binary Tree Class

A binary tree is a tree that has
a left branch and a right branch

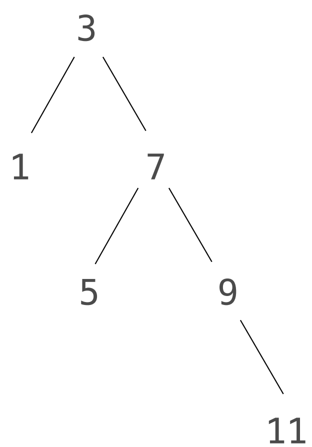
```
class BTree(Tree):
```



Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

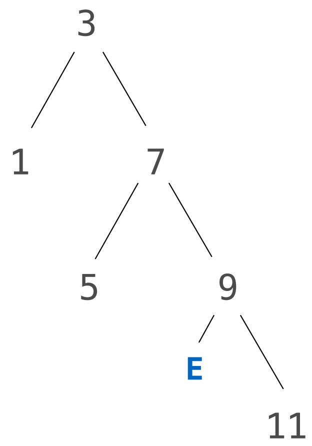



```
class BTree(Tree):
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree



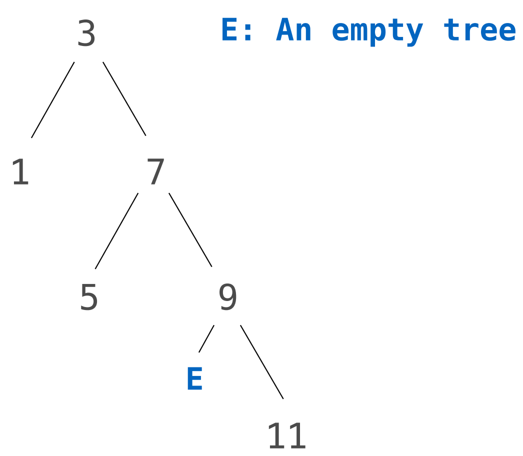

```
class BTree(Tree):
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

```
class BTree(Tree):
```

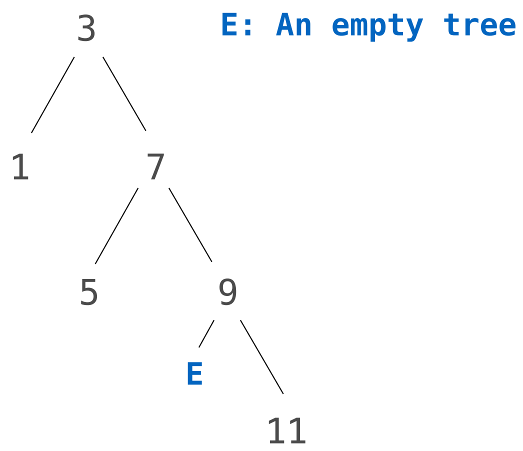


Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

```
class BTree(Tree):
    empty = Tree(None)
```

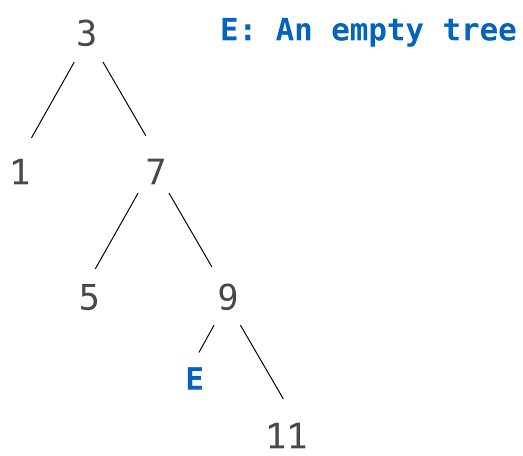


Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches



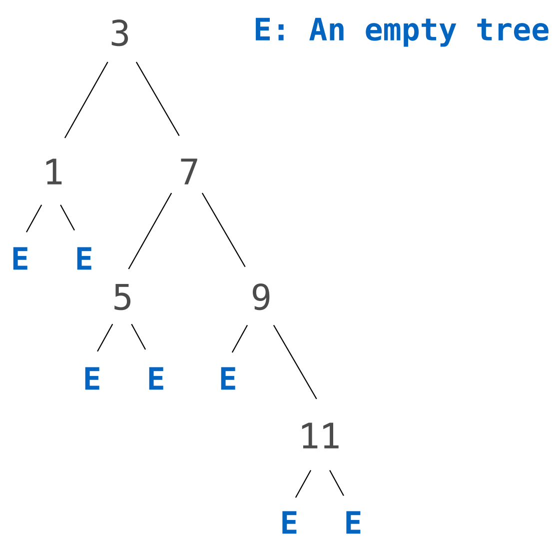

```
class BTree(Tree):
    empty = Tree(None)
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches



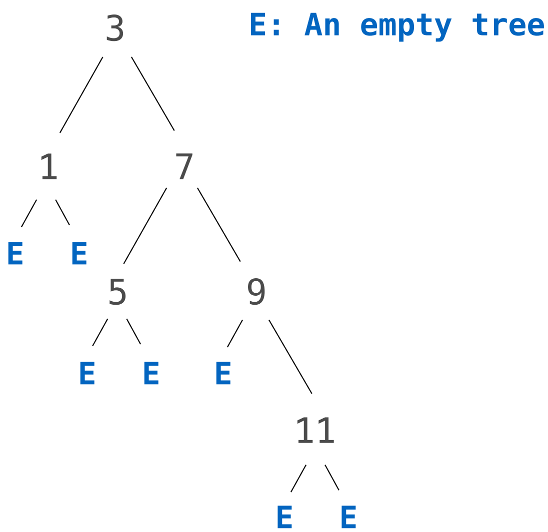

```
class BTree(Tree):
    empty = Tree(None)
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches



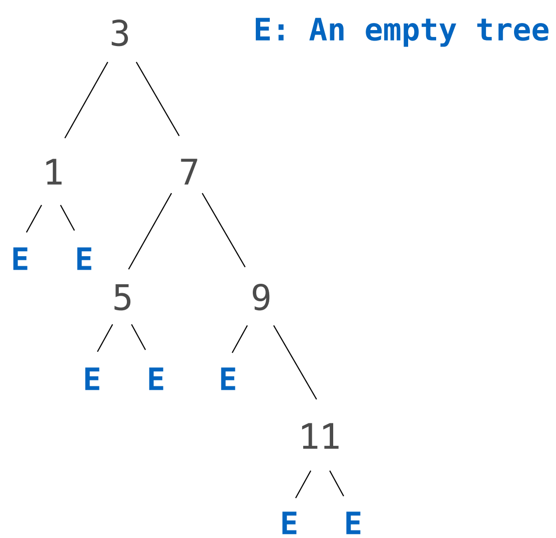

```
class BTree(Tree):
    empty = Tree(None)

    def __init__(self, label, left=empty, right=empty):
        Tree.__init__(self, label, [left, right])
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches



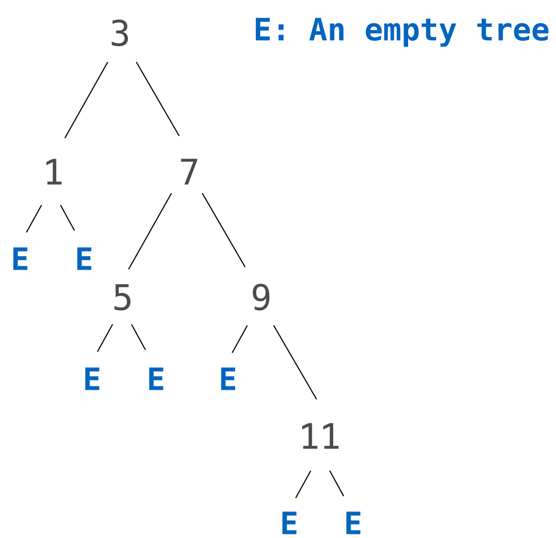

```
class BTree(Tree):
    empty = Tree(None)

    def __init__(self, label, left=empty, right=empty):
        Tree.__init__(self, label, [left, right])

    @property
    def left(self):
        return self.branches[0]
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches



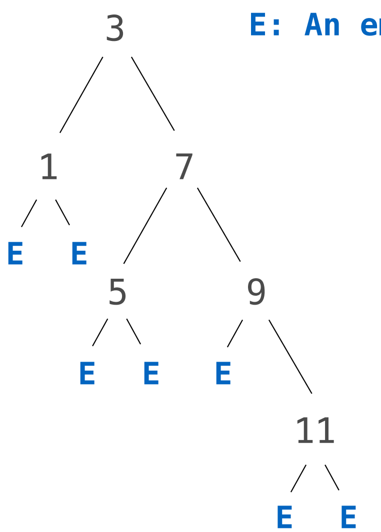

```
class BTree(Tree):
    empty = Tree(None)

    def __init__(self, label, left=empty, right=empty):
        Tree.__init__(self, label, [left, right])

    @property
    def left(self):
        return self.branches[0]

    @property
    def right(self):
        return self.branches[1]
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches



E: An empty tree

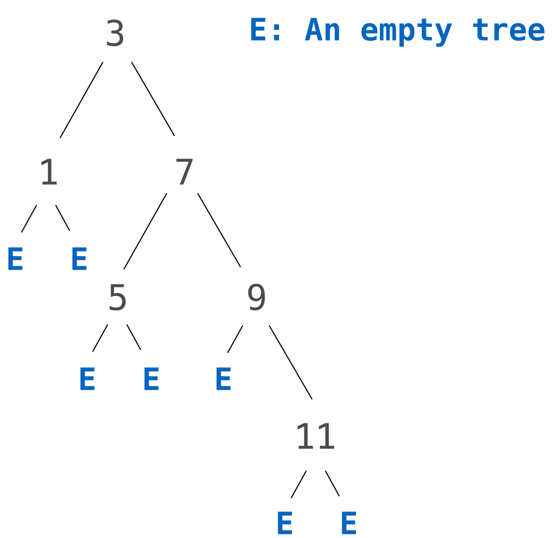
```
class BTree(Tree):
    empty = Tree(None)

    def __init__(self, label, left=empty, right=empty):
        Tree.__init__(self, label, [left, right])

    @property
    def left(self):
        return self.branches[0]

    @property
    def right(self):
        return self.branches[1]


t = BTree(3, BTree(1),
             BTree(7, BTree(5),
                      BTree(9, BTree.empty,
                               BTree(11))))
```

Binary Tree Class

A binary tree is a tree that has a left branch and a right branch

Idea: Fill the place of a missing left branch with an empty tree

Idea 2: An instance of BTree always has *exactly* two branches




```
class BTree(Tree):
    empty = Tree(None)

    def __init__(self, label, left=empty, right=empty):
        Tree.__init__(self, label, [left, right])

    @property
    def left(self):
        return self.branches[0]

    @property
    def right(self):
        return self.branches[1]


t = BTree(3, BTree(1),
             BTree(7, BTree(5),
                      BTree(9, BTree.empty,
                               BTree(11))))
```

(Demo)

Binary Search Trees

Binary Search

```
A strategy for finding a value in a sorted list: check the middle and eliminate half
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

            False
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]        4 in [1, 2, 4, 8, 16, 32]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

           False
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]        4 in [1, 2, 4, 8, 16, 32]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

            False
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

            False
```

```
4 in [1, 2, 4, 8, 16, 32]

     [1, 2, 4, 8, 16, 32]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

False
```

```
4 in [1, 2, 4, 8, 16, 32]

     [1, 2, 4, 8, 16, 32]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

            False
```

```
4 in [1, 2, 4, 8, 16, 32]

     [1, 2, 4, 8, 16, 32]

     [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]
      [1, 2, 4, 8, 16, 32, 64]
      [1, 2, 4, 8, 16, 32, 64]
      False
```

```
4 in [1, 2, 4, 8, 16, 32]
     [1, 2, 4, 8, 16, 32]
     [1, 2, 4, 8, 16, 32, 64]
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]        4 in [1, 2, 4, 8, 16, 32]

      [1, 2, 4, 8, 16, 32, 64]             [1, 2, 4, 8, 16, 32]

      [1, 2, 4, 8, 16, 32, 64]             [1, 2, 4, 8, 16, 32, 64]

            False                                 True
```

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]        4 in [1, 2, 4, 8, 16, 32]

      [1, 2, 4, 8, 16, 32, 64]             [1, 2, 4, 8, 16, 32]

      [1, 2, 4, 8, 16, 32, 64]             [1, 2, 4, 8, 16, 32, 64]

            False                                  True
```

For a sorted list of length n, what Theta expression describes the time required?

Binary Search

A strategy for finding a value in a sorted list: check the middle and eliminate half

```
20 in [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

      [1, 2, 4, 8, 16, 32, 64]

             False
```

```
4 in [1, 2, 4, 8, 16, 32]

     [1, 2, 4, 8, 16, 32]

     [1, 2, 4, 8, 16, 32, 64]

             True
```

For a sorted list of length n, what Theta expression describes the time required? $\Theta(\log n)$

Binary Search Trees

Binary Search Trees

A binary search tree is a binary tree where each node’s label is:

Binary Search Trees

A binary search tree is a binary tree where each node's label is:

- Larger than all node labels in its left branch and

Binary Search Trees

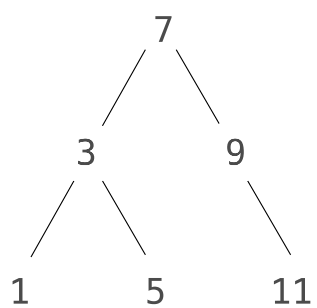
A binary search tree is a binary tree where each node's label is:

- Larger than all node labels in its left branch and
- Smaller than all node labels in its right branch

Binary Search Trees

A binary search tree is a binary tree where each node's label is:

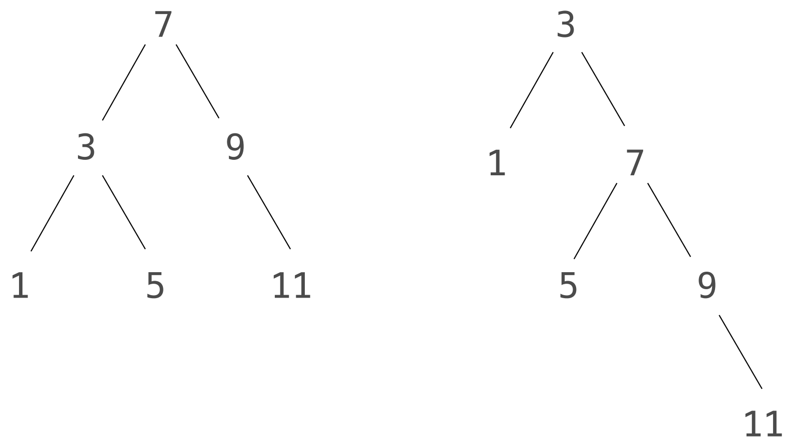
- Larger than all node labels in its left branch and
- Smaller than all node labels in its right branch



Binary Search Trees

A binary search tree is a binary tree where each node's label is:

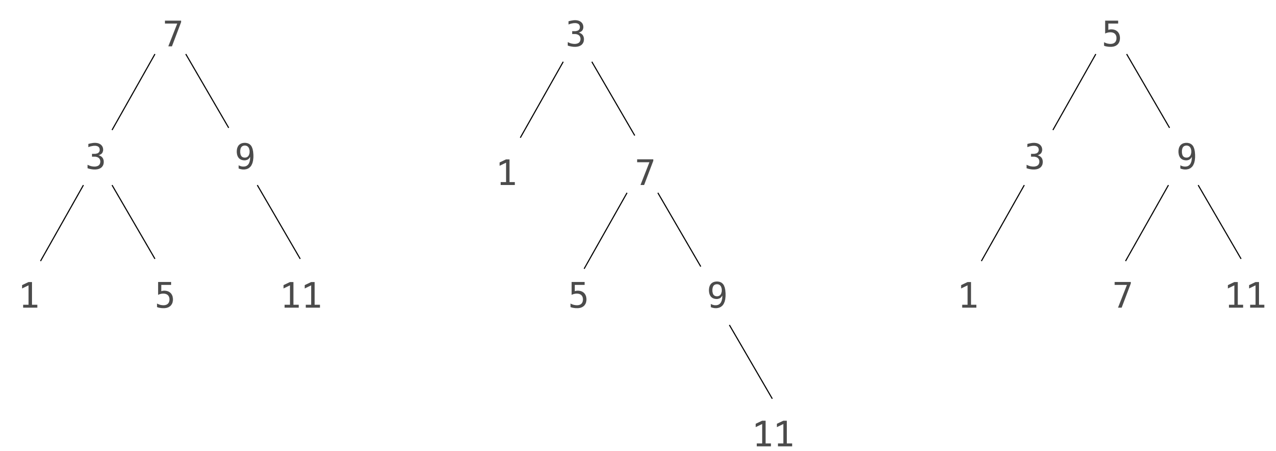
- Larger than all node labels in its left branch and
- Smaller than all node labels in its right branch



Binary Search Trees

A binary search tree is a binary tree where each node's label is:

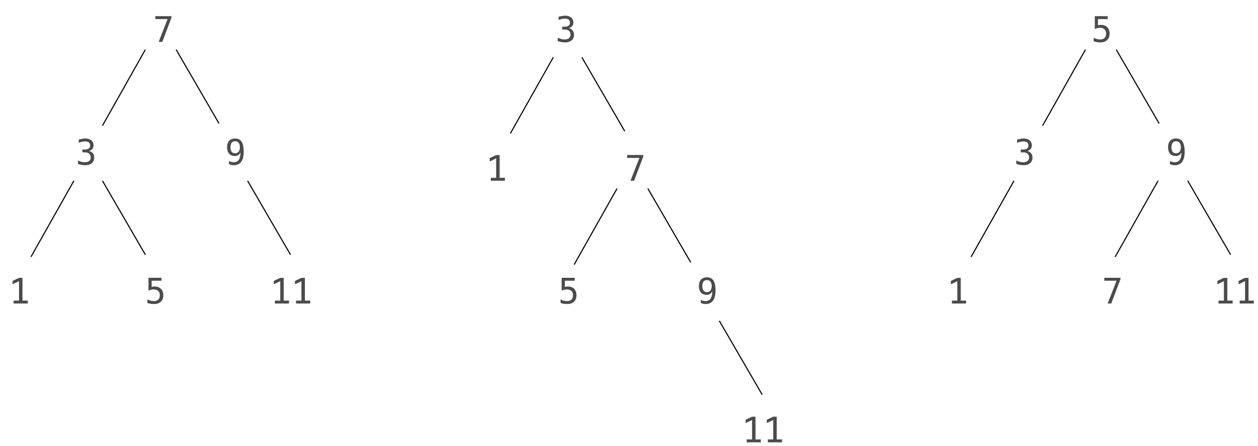
- Larger than all node labels in its left branch and
- Smaller than all node labels in its right branch



Binary Search Trees

A binary search tree is a binary tree where each node's label is:

- Larger than all node labels in its left branch and
- Smaller than all node labels in its right branch



(Demo)

Discussion Questions

```
What's the largest element
in a binary search tree?
```

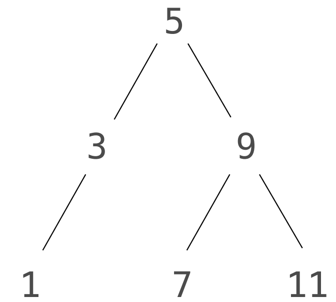
Discussion Questions

```
What's the largest element
in a binary search tree?

def largest(t):
    if ____________________:
        return ______________
    else:
        return ______________
```

Discussion Questions

What's the largest element
in a binary search tree?

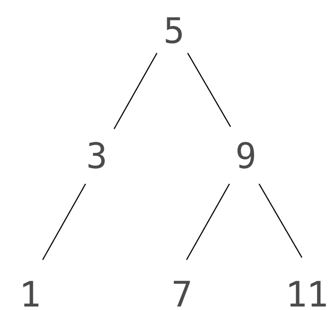
```
def largest(t):

    if ______________________:

        return _______________

    else:

        return _______________
```



Discussion Questions

What's the largest element
in a binary search tree?

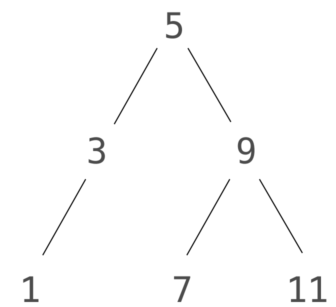
```
def largest(t):
    if  t.right is BTree.empty :
        return      t.label
    else:
        return ________________
```



Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return      t.label
    else:
        return  largest(t.right)
```

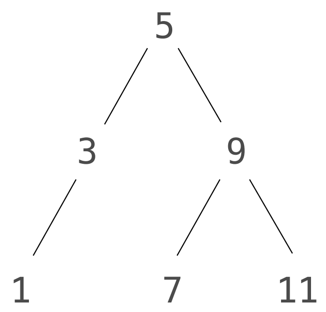


Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return  t.label
    else:
        return  largest(t.right)
```

What's the second largest element in a binary search tree?



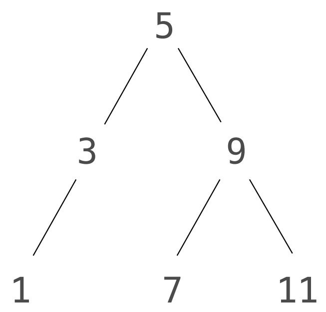
Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return  t.label
    else:
        return  largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif ________________________:
        return __________________
    elif ________________________:
        return t.label
    else:
        return __________________
```



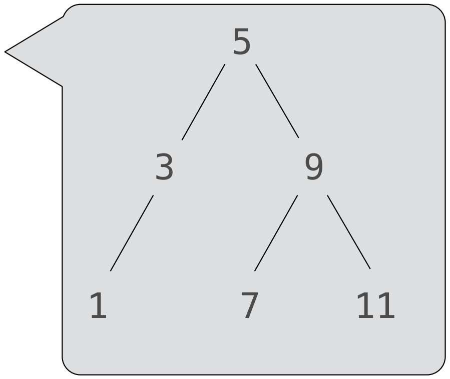
Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return      t.label
    else:
        return largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif ________________________:
        return __________________
    elif ________________________:
        return t.label
    else:
        return __________________
```



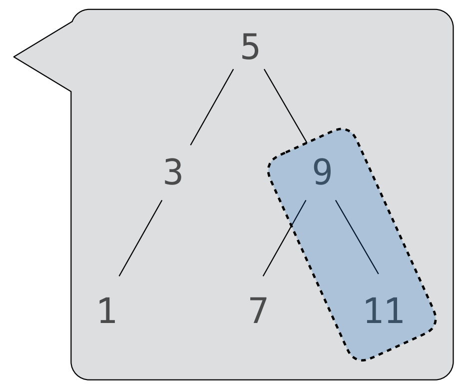
Discussion Questions

What's the largest element
in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return      t.label
    else:
        return  largest(t.right)
```

What's the second largest element
in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif ______________________:
        return ________________
    elif ______________________:
        return t.label
    else:
        return ________________
```



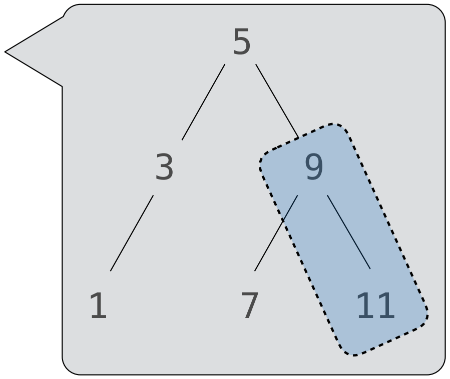
Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if __t.right is BTree.empty__:
        return ____t.label____
    else:
        return _largest(t.right)_
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif ______________________:
        return ________________
    elif __t.right.is_leaf()__:
        return t.label
    else:
        return ________________
```



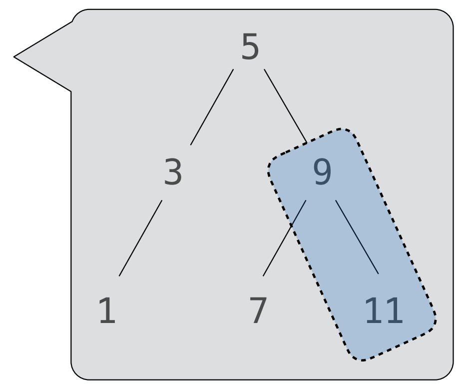
Discussion Questions

What's the largest element
in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return     t.label
    else:
        return  largest(t.right)
```

What's the second largest element
in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif  t.right is BTree.empty :
        return ________________
    elif  t.right.is_leaf() :
        return t.label
    else:
        return ________________
```



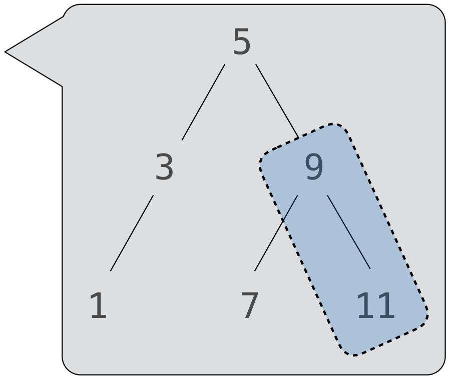
Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return  t.label
    else:
        return  largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif  t.right is BTree.empty :
        return  largest(t.left)
    elif  t.right.is_leaf() :
        return t.label
    else:
        return ____________________
```



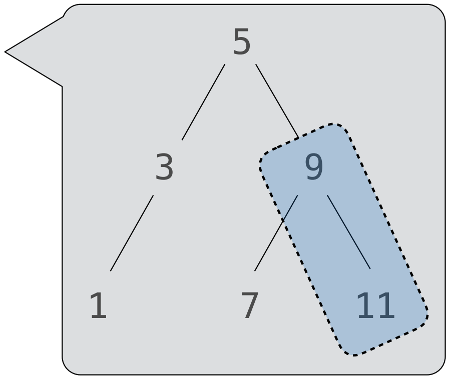
Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return      t.label
    else:
        return  largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif t.right is BTree.empty :
        return  largest(t.left)
    elif    t.right.is_leaf()    :
        return t.label
    else:
        return  second(t.right)
```



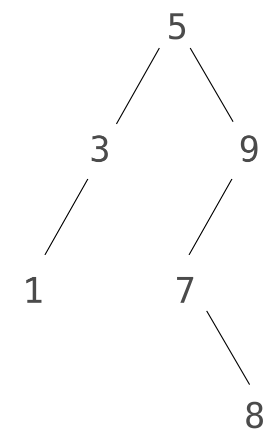
Discussion Questions

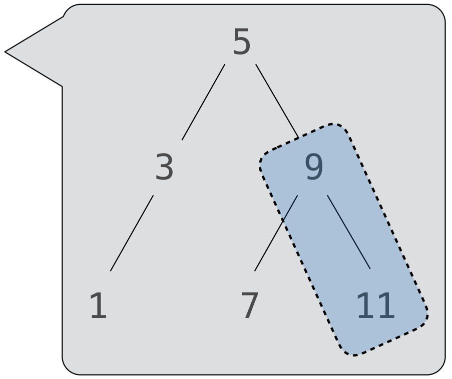
What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return  t.label
    else:
        return  largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif  t.right is BTree.empty :
        return  largest(t.left)
    elif  t.right.is_leaf() :
        return t.label
    else:
        return  second(t.right)
```





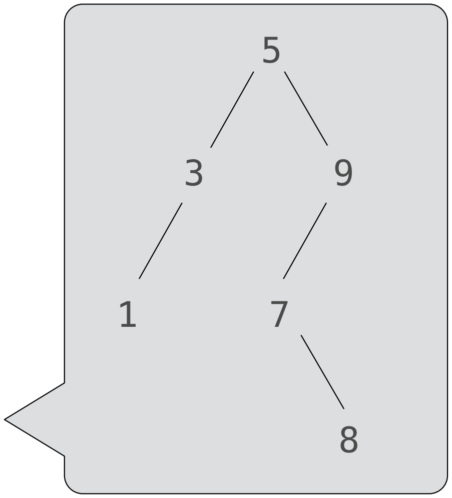
Discussion Questions

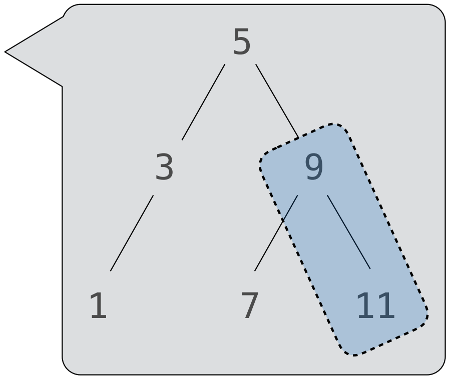
What's the largest element in a binary search tree?

```
def largest(t):
    if  t.right is BTree.empty :
        return  t.label
    else:
        return  largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif  t.right is BTree.empty :
        return  largest(t.left)
    elif  t.right.is_leaf() :
        return t.label
    else:
        return  second(t.right)
```





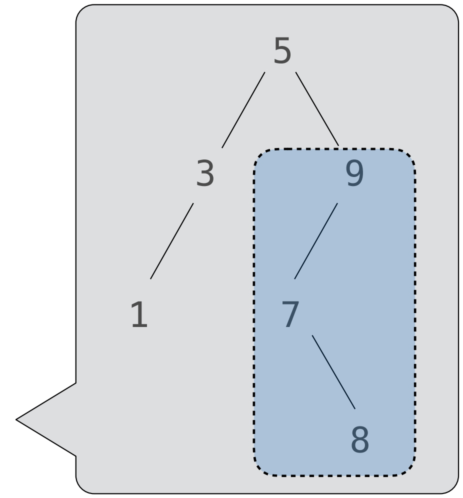
Discussion Questions

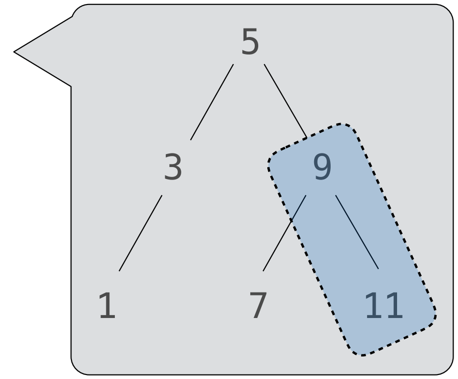
What's the largest element in a binary search tree?

```
def largest(t):
    if _t.right is BTree.empty_:
        return ____t.label____
    else:
        return _largest(t.right)_
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif _t.right is BTree.empty_:
        return _largest(t.left)_
    elif __t.right.is_leaf()__:
        return t.label
    else:
        return _second(t.right)_
```





Sets as Binary Search Trees

Membership in Binary Search Trees

Membership in Binary Search Trees

`contains` traverses the tree

Membership in Binary Search Trees

`contains` `traverses the tree`

- `If the element is not at the root, it can only be in either the left or right branch`

Membership in Binary Search Trees

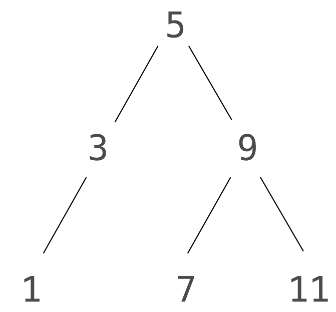
`contains` traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

Membership in Binary Search Trees

`contains` traverses the tree

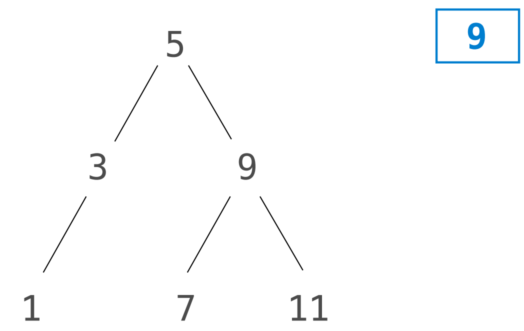
- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch



Membership in Binary Search Trees

`contains` traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

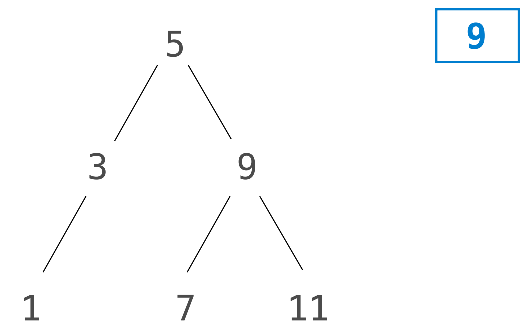


Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
```

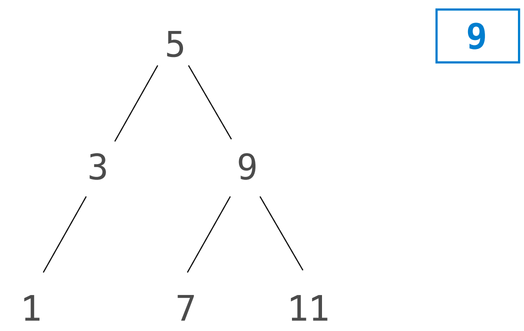


Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
```

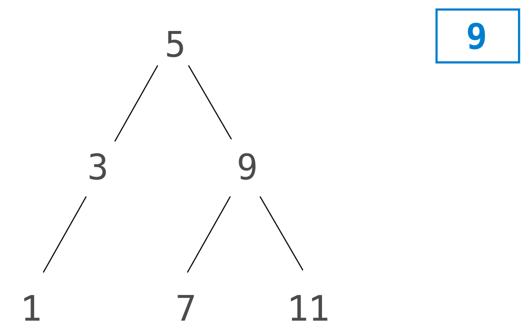


Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
```

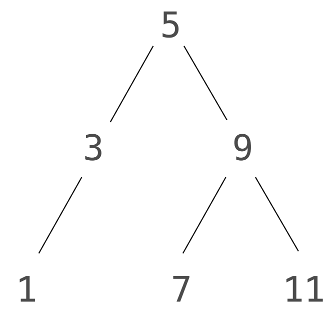


Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
    elif s.label < v:
        return contains(s.right, v)
```



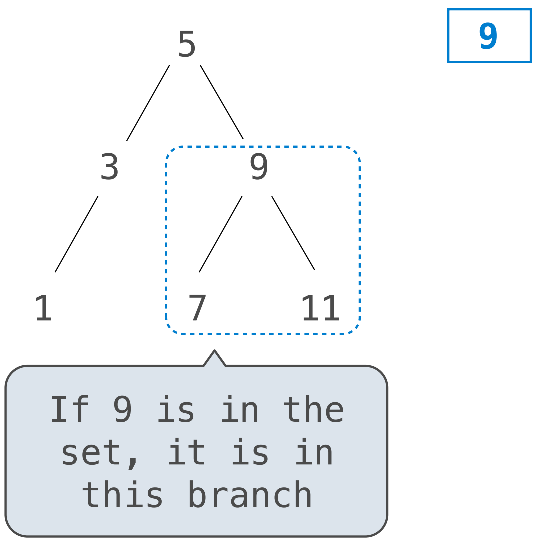
9

Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
    elif s.label < v:
        return contains(s.right, v)
```

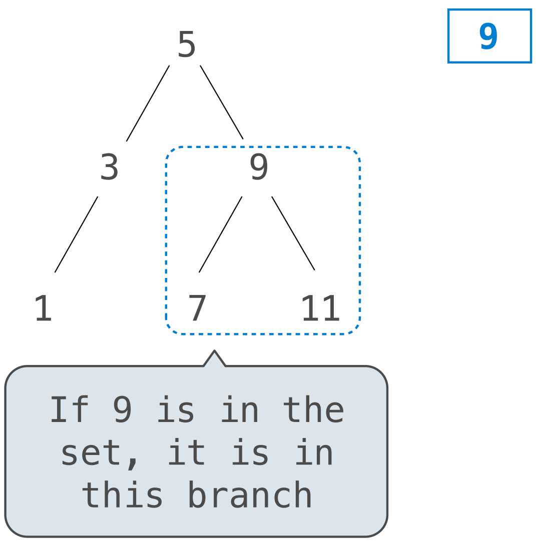


Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
    elif s.label < v:
        return contains(s.right, v)
    elif s.label > v:
        return contains(s.left, v)
```

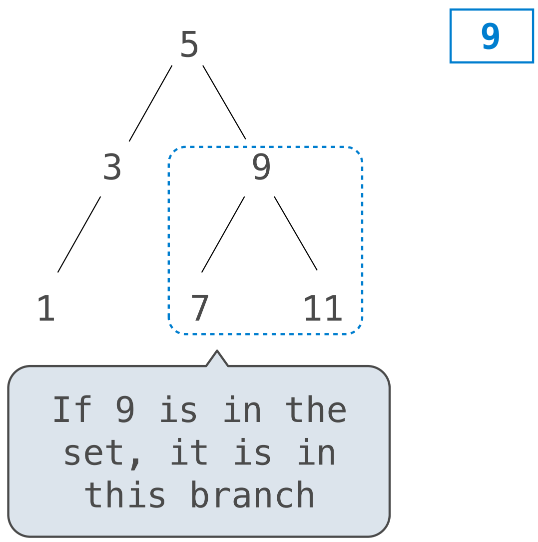


Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
    elif s.label < v:
        return contains(s.right, v)
    elif s.label > v:
        return contains(s.left, v)
```



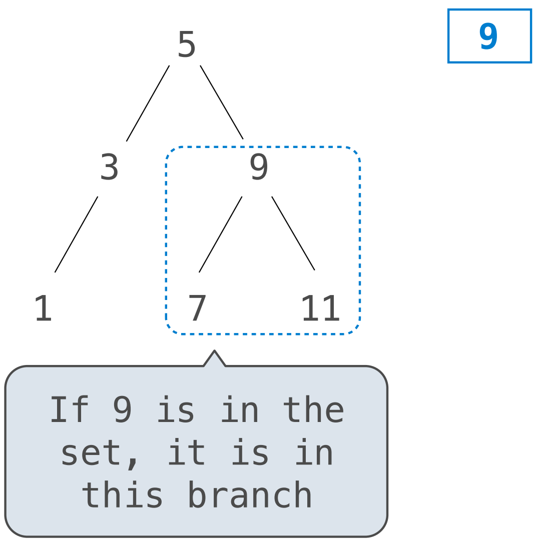
Order of growth?

Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
    elif s.label < v:
        return contains(s.right, v)
    elif s.label > v:
        return contains(s.left, v)
```



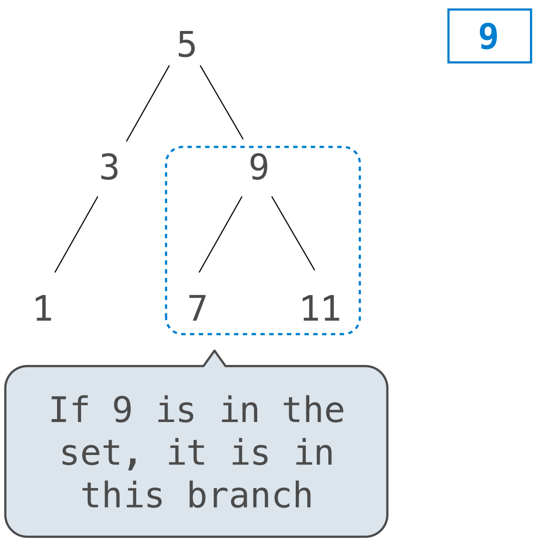
Order of growth? $\Theta(h)$ on average

Membership in Binary Search Trees

contains traverses the tree

- If the element is not at the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch

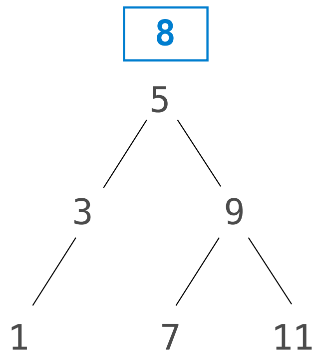
```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.label == v:
        return True
    elif s.label < v:
        return contains(s.right, v)
    elif s.label > v:
        return contains(s.left, v)
```



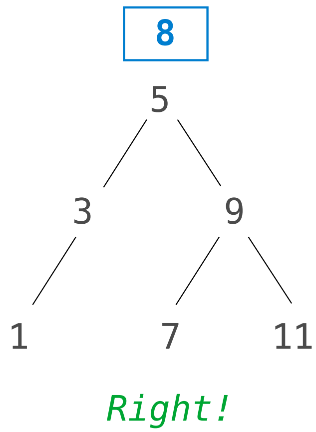
Order of growth? $\Theta(h)$ on average $\Theta(\log n)$ on average for a balanced tree

Adjoining to a Tree Set

Adjoining to a Tree Set

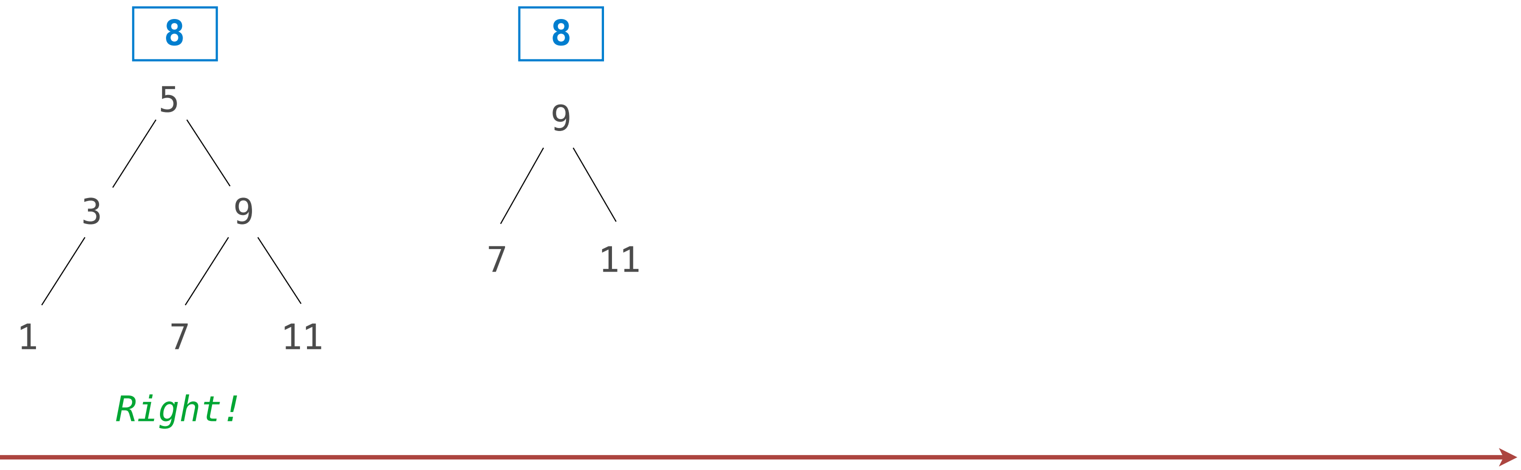


Adjoining to a Tree Set

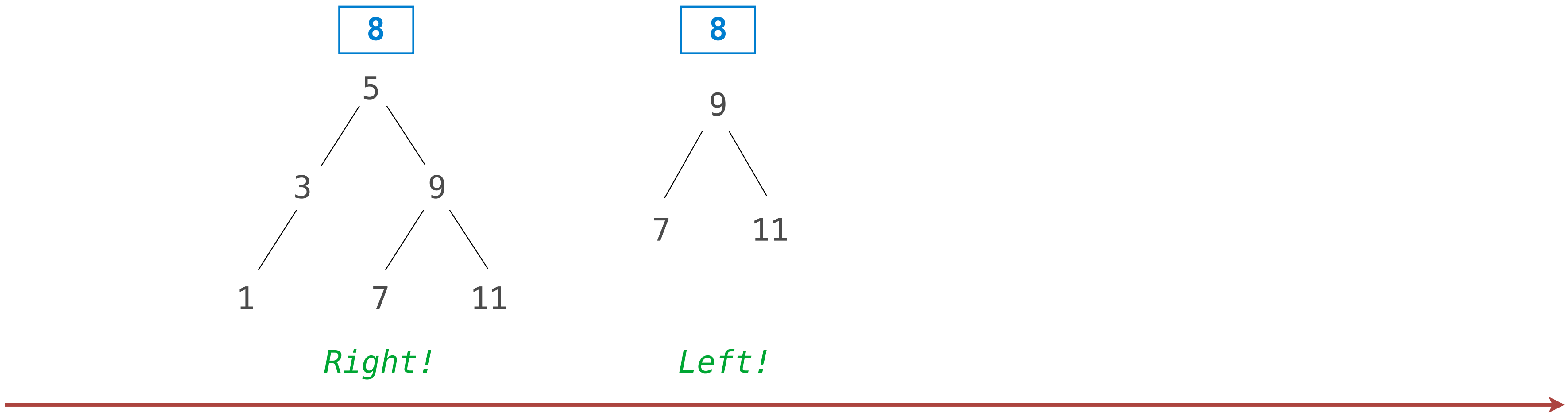


Adjoining to a Tree Set

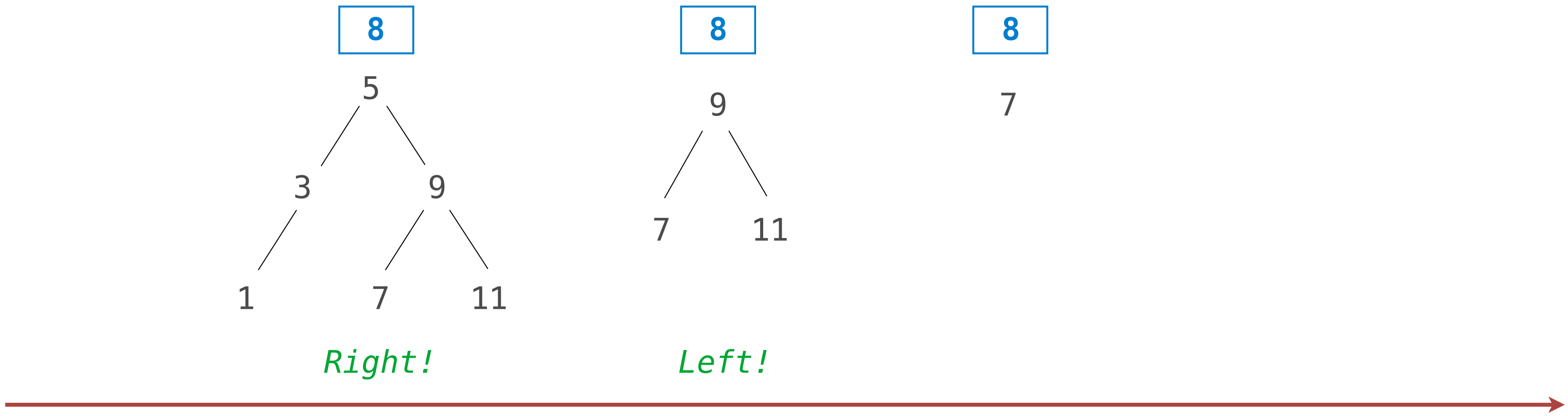
Adjoining to a Tree Set



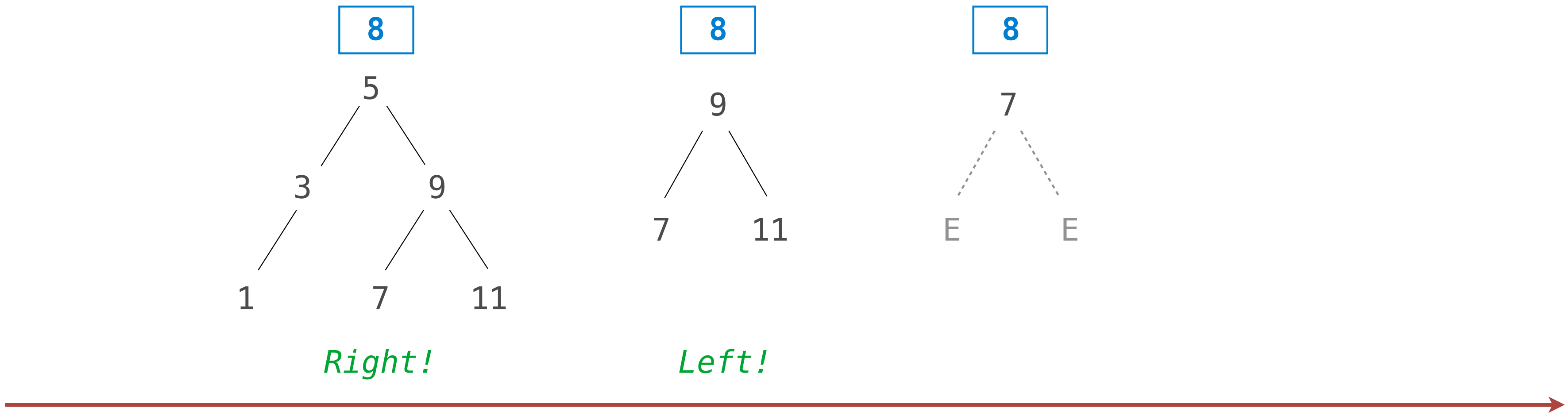
Adjoining to a Tree Set



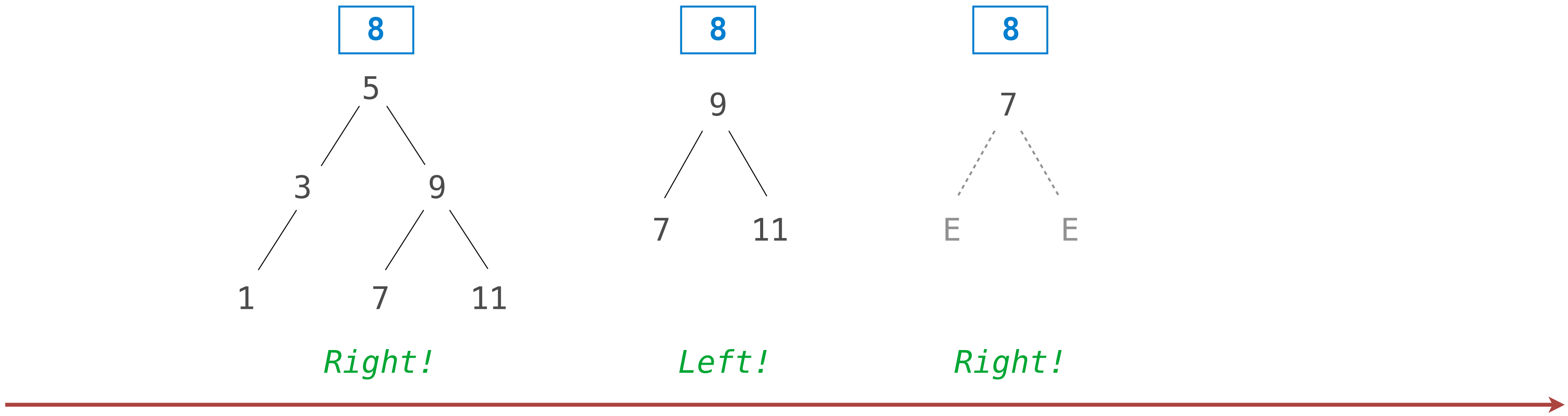
Adjoining to a Tree Set



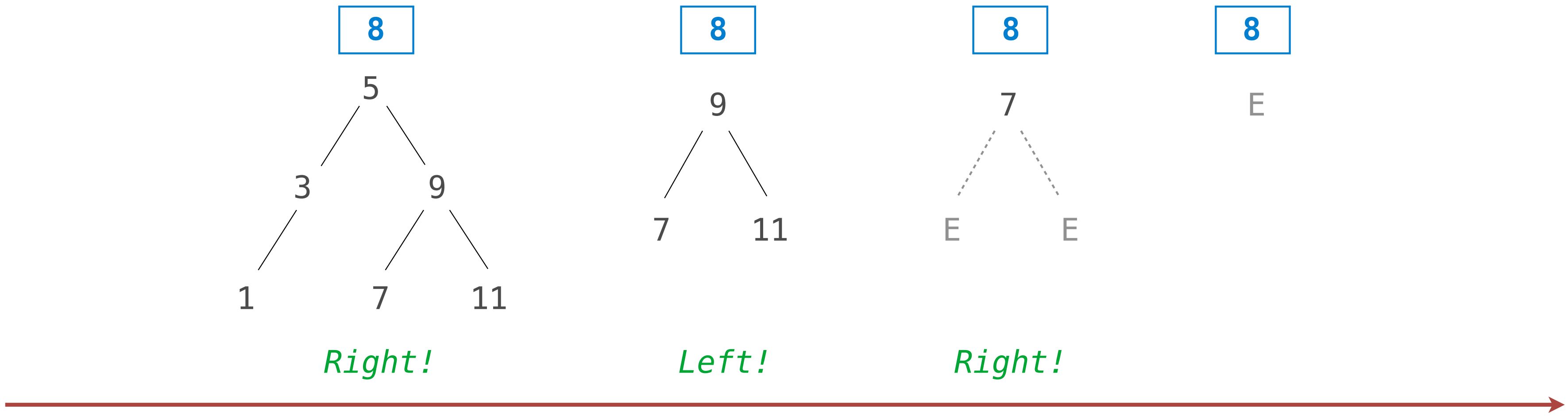
Adjoining to a Tree Set



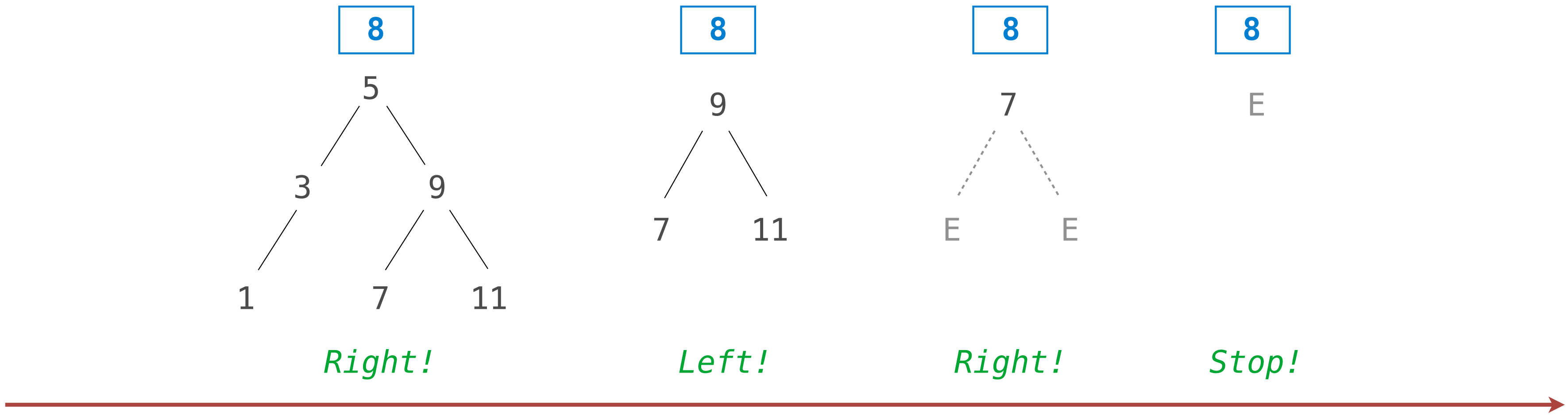
Adjoining to a Tree Set



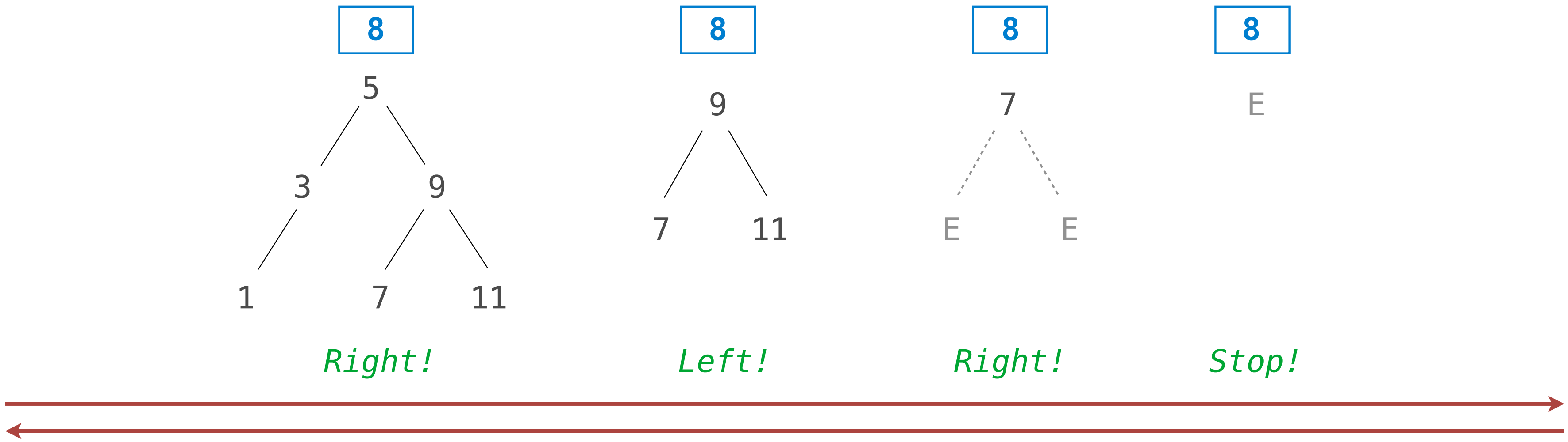
Adjoining to a Tree Set



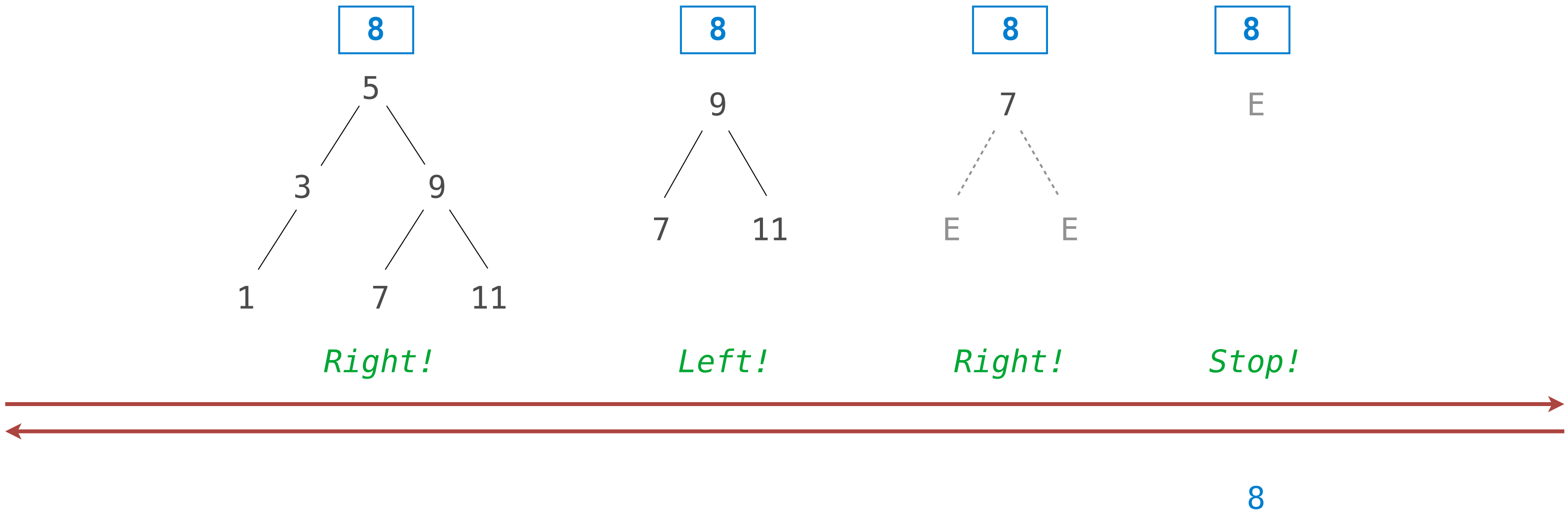
Adjoining to a Tree Set



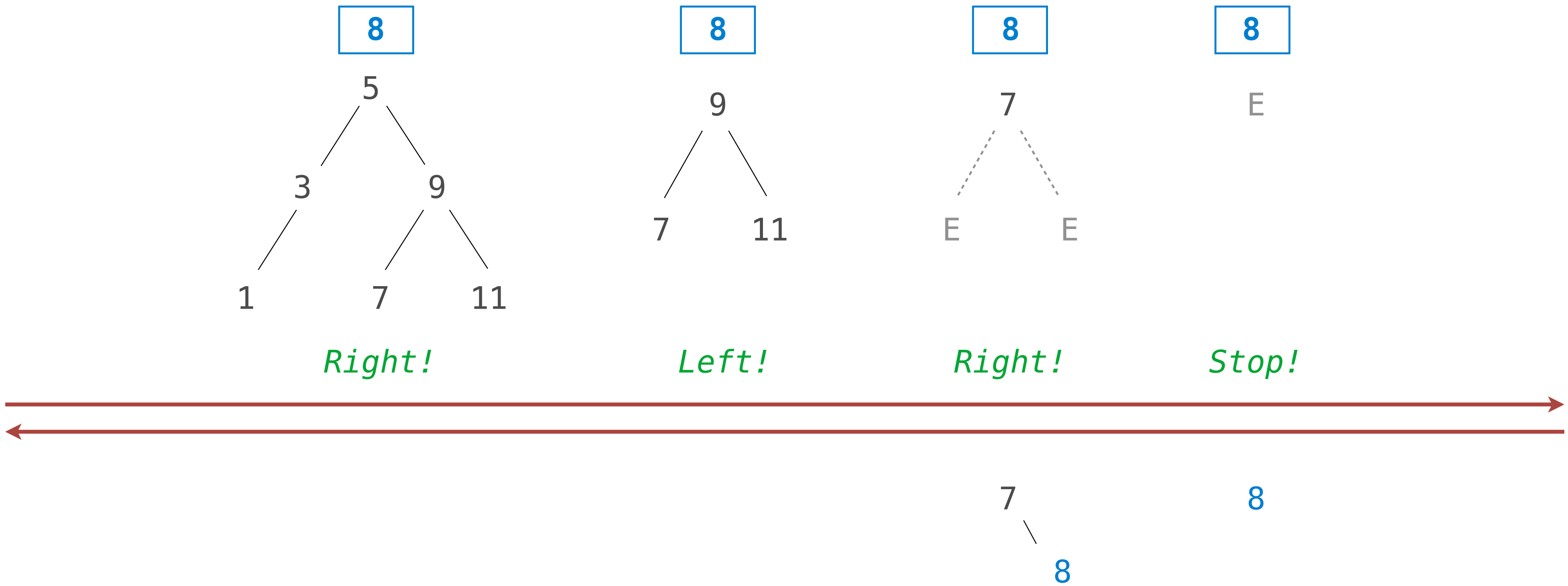
Adjoining to a Tree Set



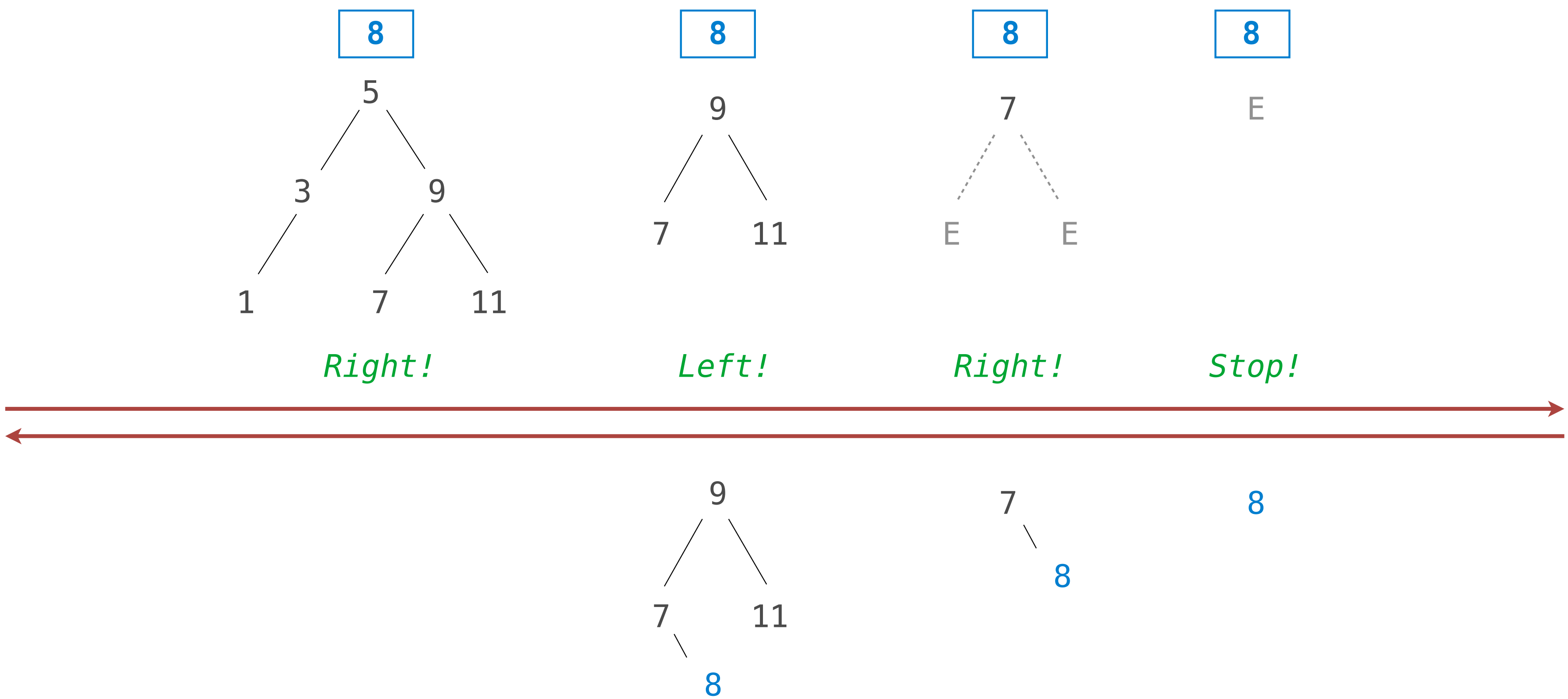
Adjoining to a Tree Set



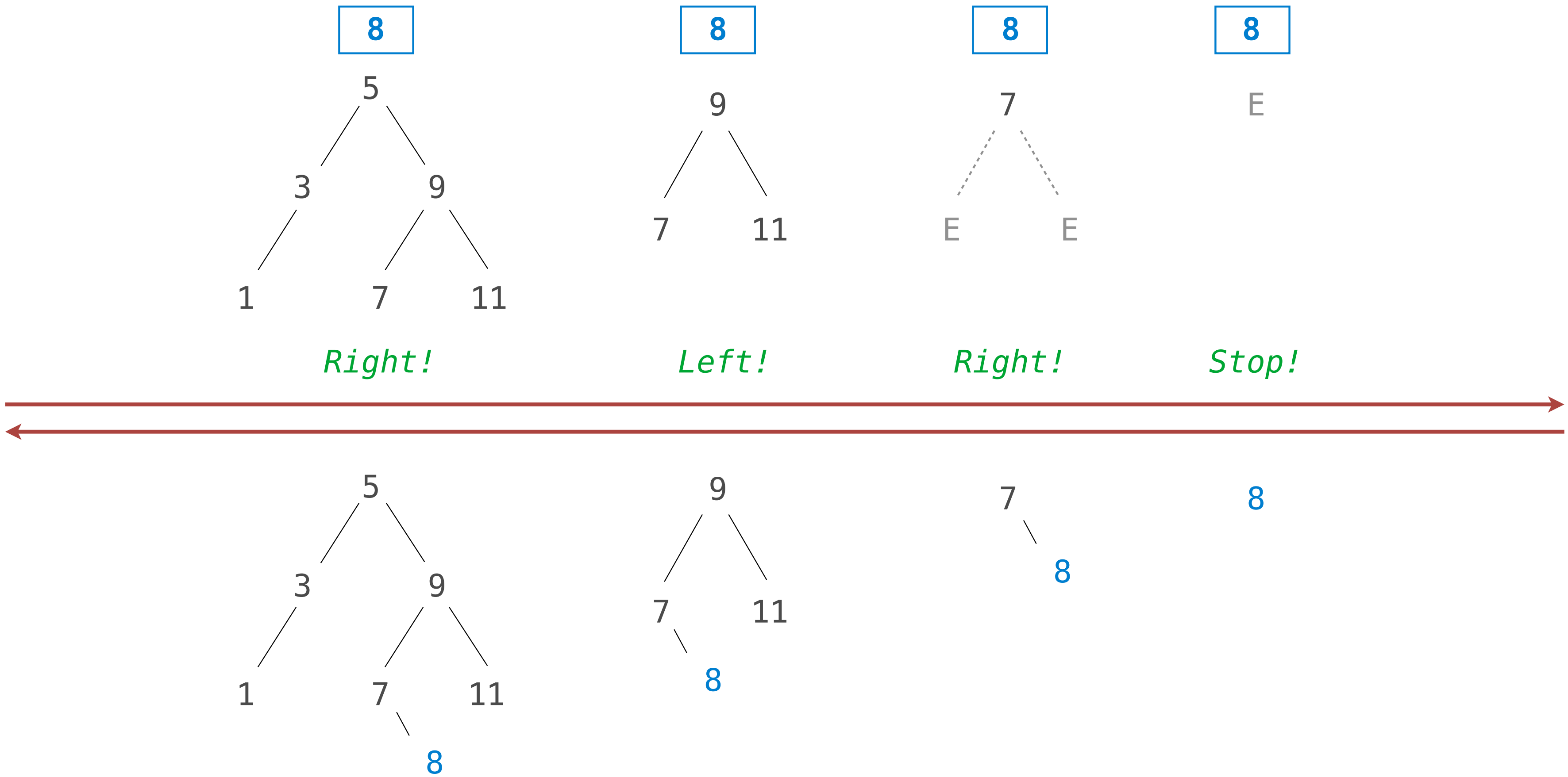
Adjoining to a Tree Set



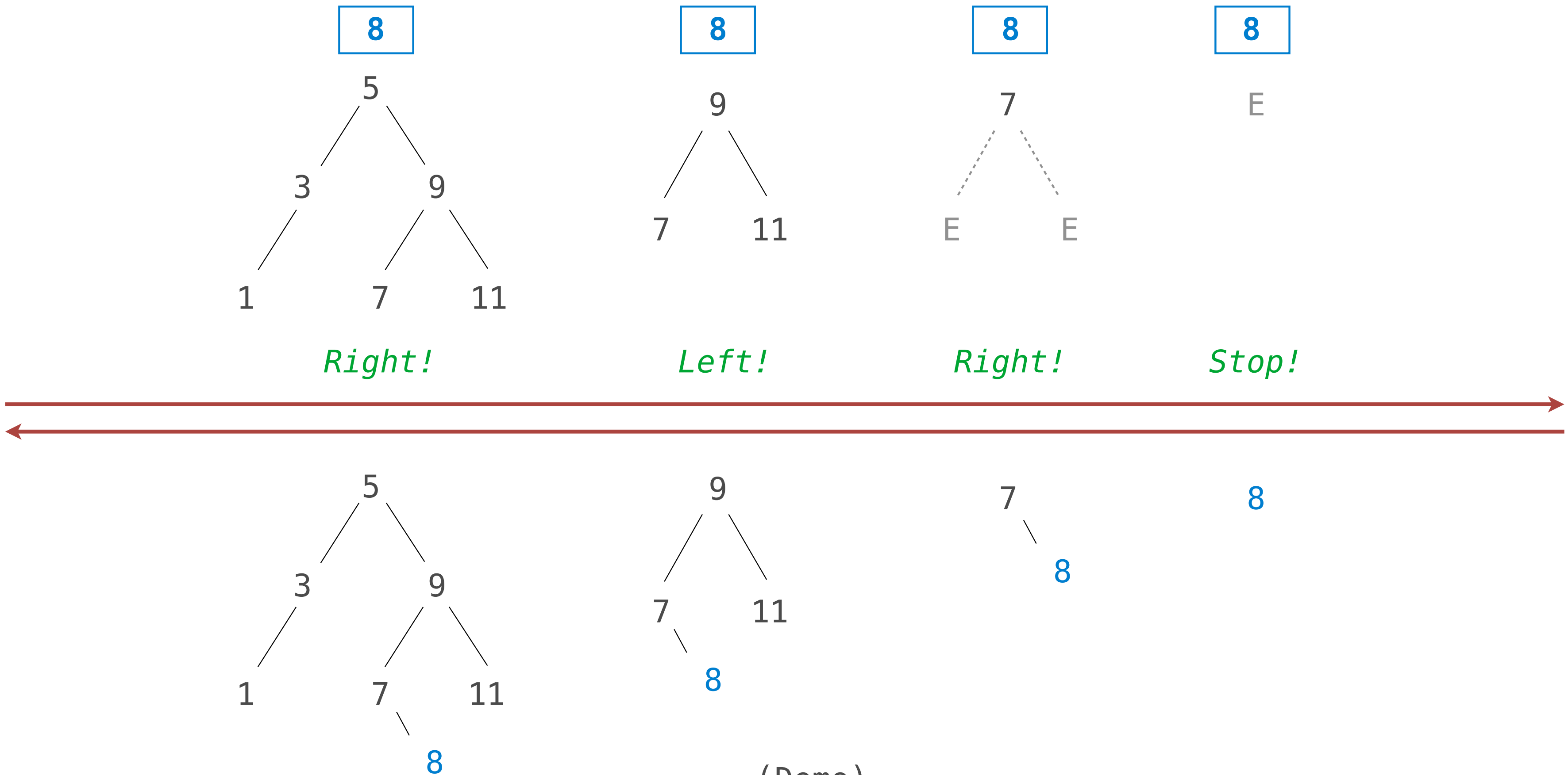
Adjoining to a Tree Set



Adjoining to a Tree Set



Adjoining to a Tree Set



(Demo)