61A Lecture 21

Announcements

Binary Trees

Binary Tree Class

Binary Tree Class

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

Binary Tree Class

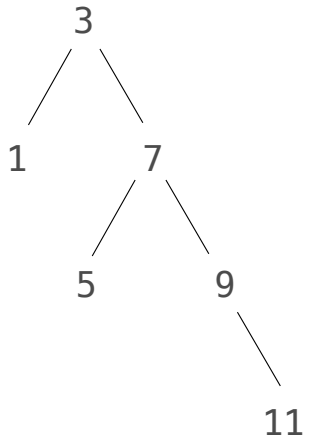
A binary tree is a tree that has a left branch and a right branch

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

Binary Tree Class

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

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

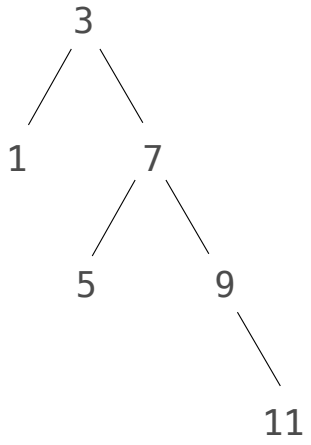


Binary Tree Class

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):
```

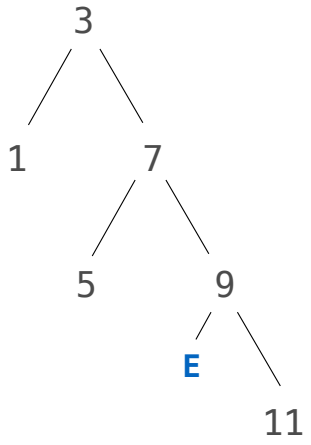


Binary Tree Class

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):
```

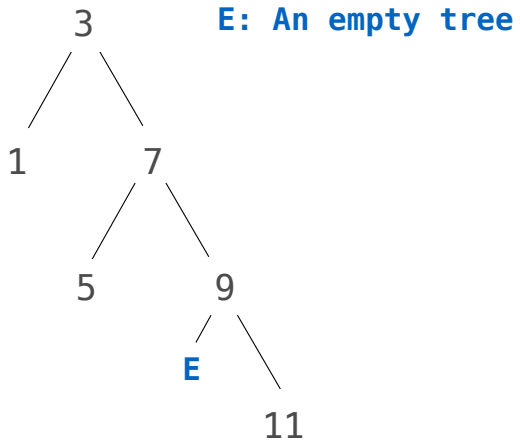


Binary Tree Class

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):
```



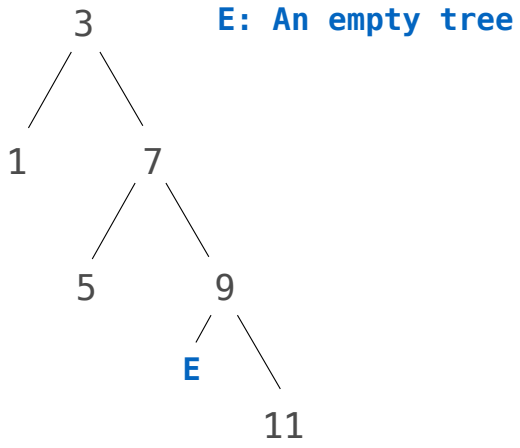
E: An empty tree

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)
```



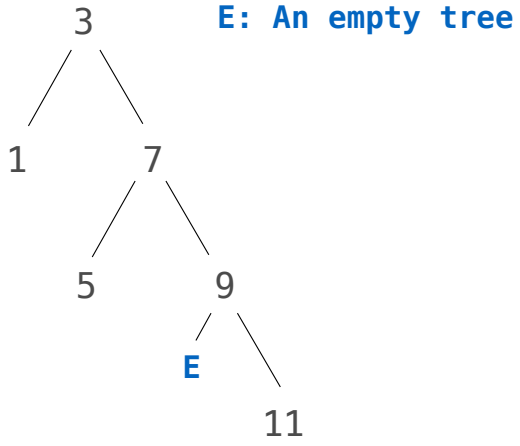
E: An empty tree

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

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



E: An empty tree

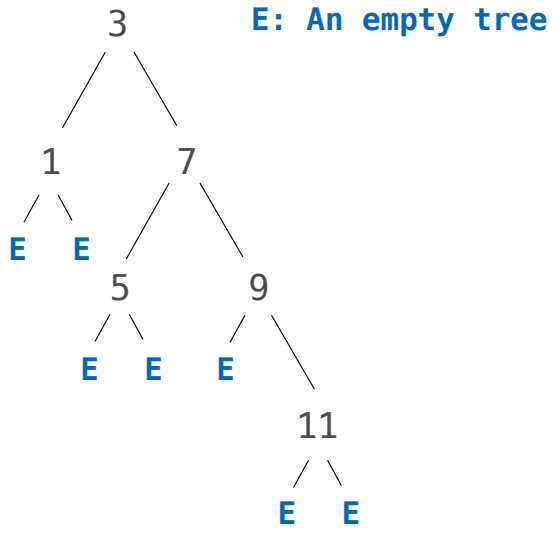
```
class BTree(Tree):
    empty = Tree(None)
```

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



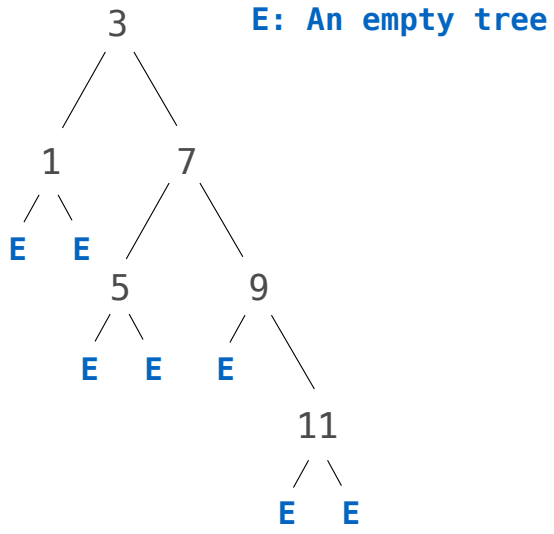

```
class BTree(Tree):
    empty = Tree(None)
```

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



E: An empty tree

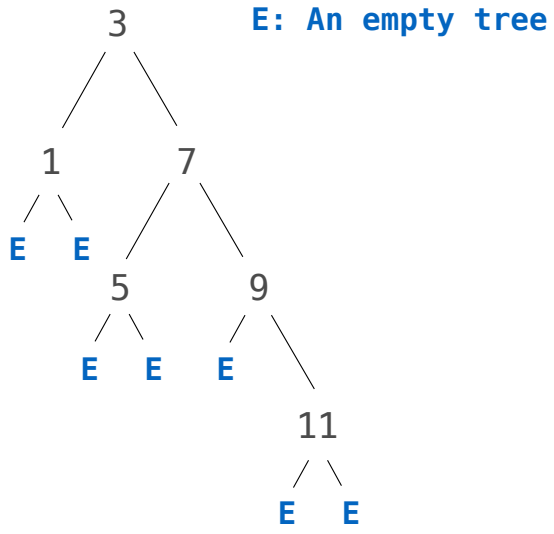
```
class BTree(Tree):
    empty = Tree(None)

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



E: An empty tree

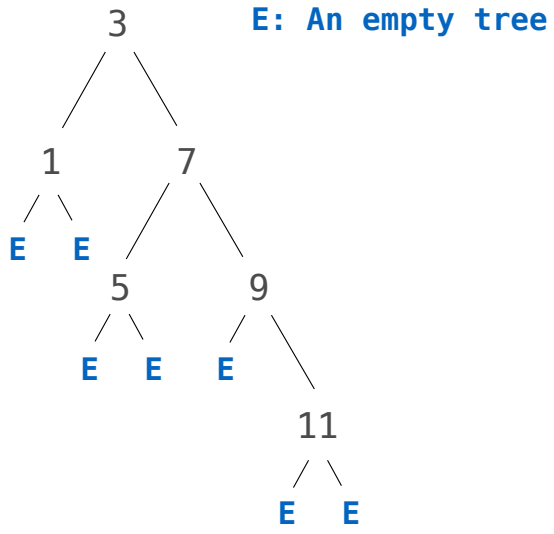
```
class BTree(Tree):
    empty = Tree(None)

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



E: An empty tree

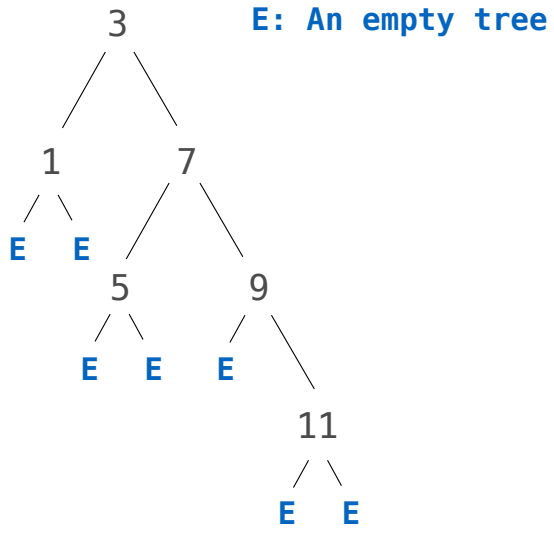
```
class BTree(Tree):
    empty = Tree(None)

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

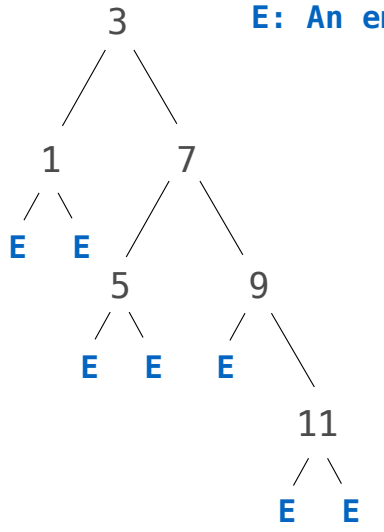
```
class BTree(Tree):
    empty = Tree(None)

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



E: An empty tree

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

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

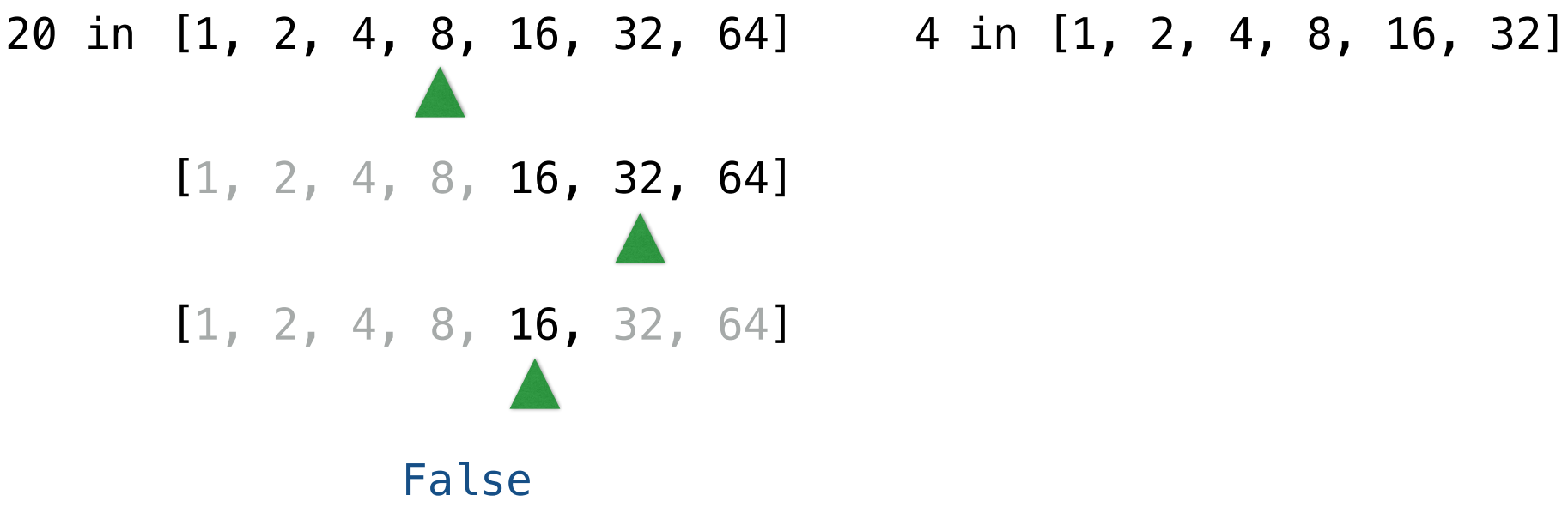
Binary Search

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
```

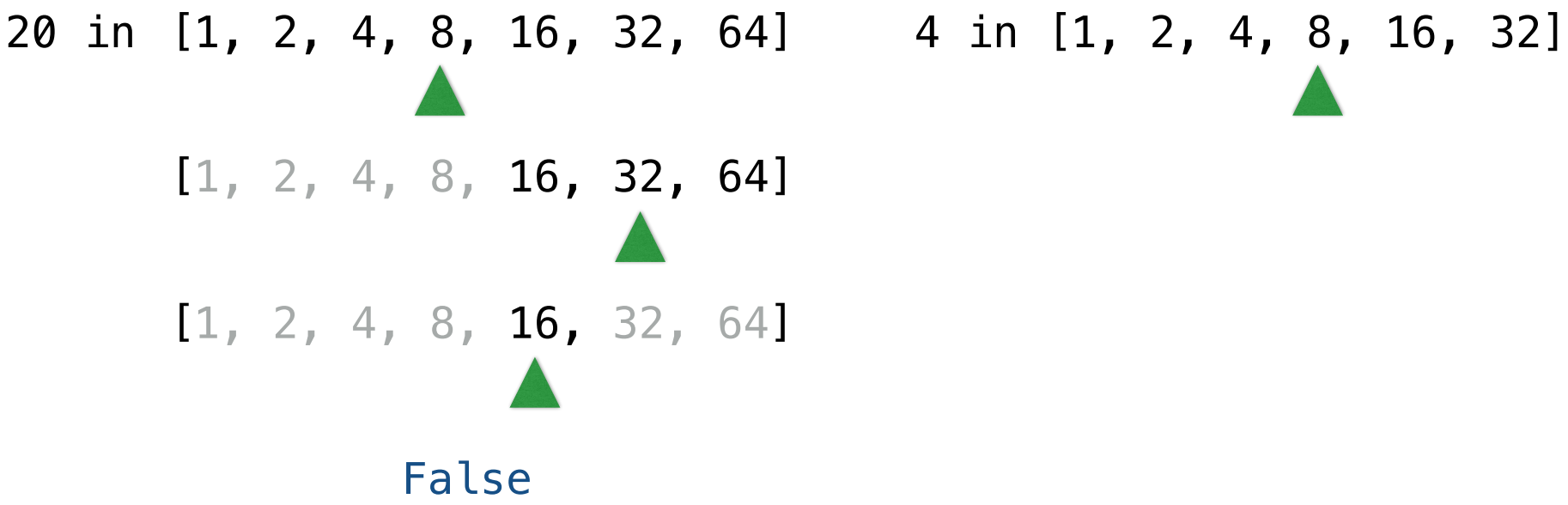
Binary Search

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



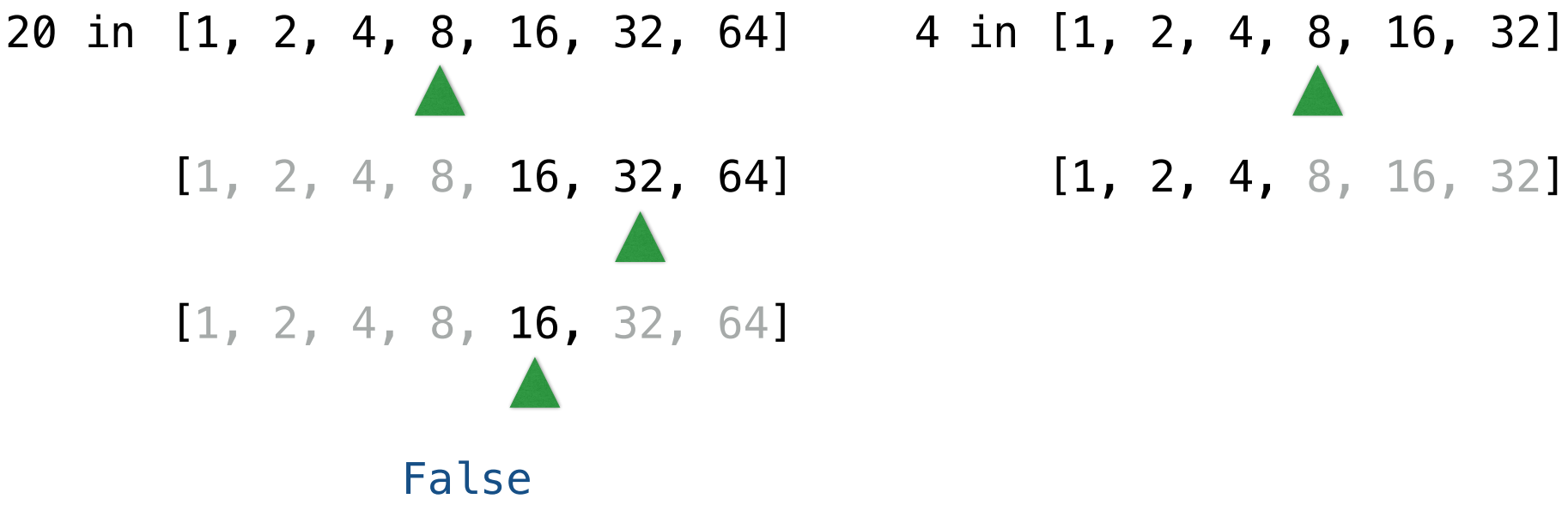
Binary Search

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



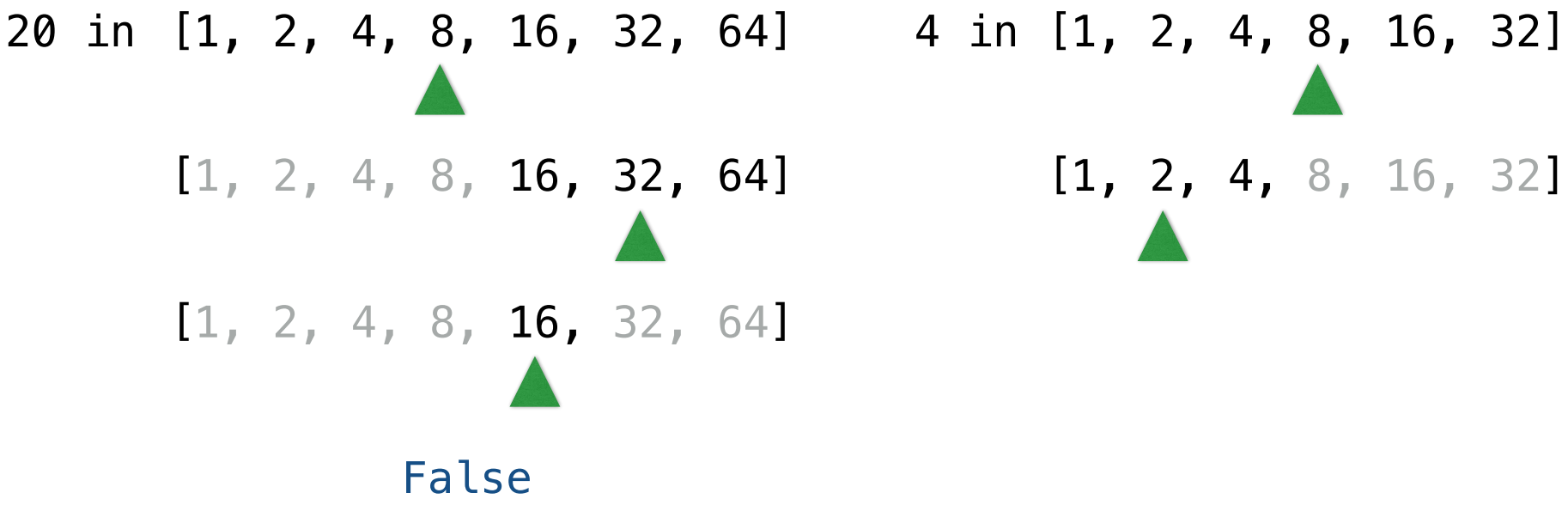
Binary Search

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



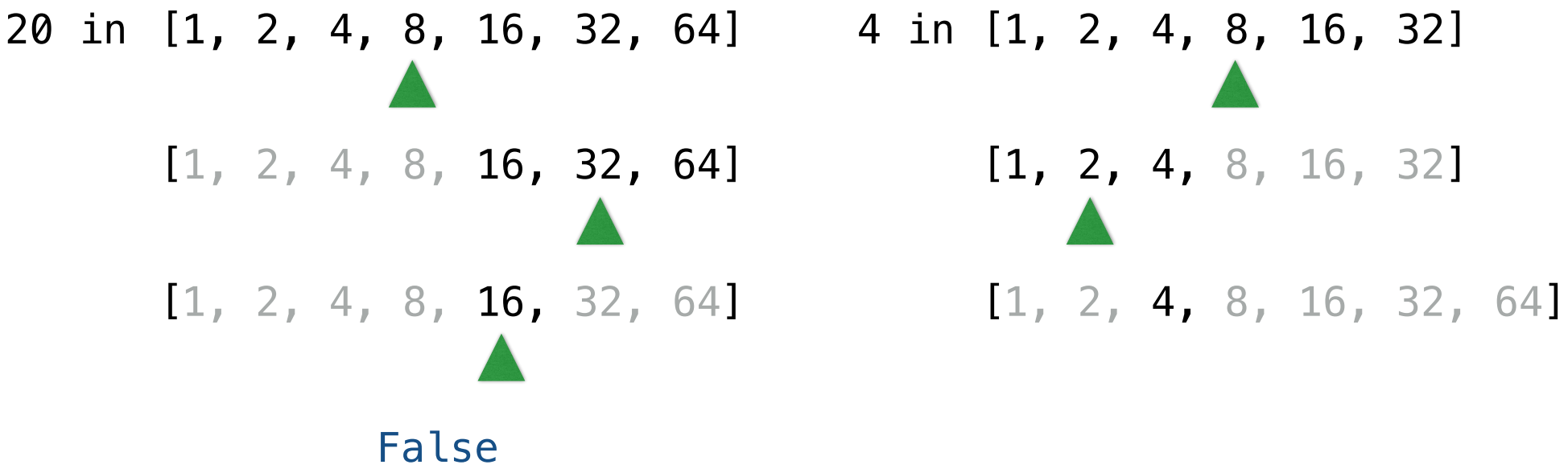
Binary Search

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



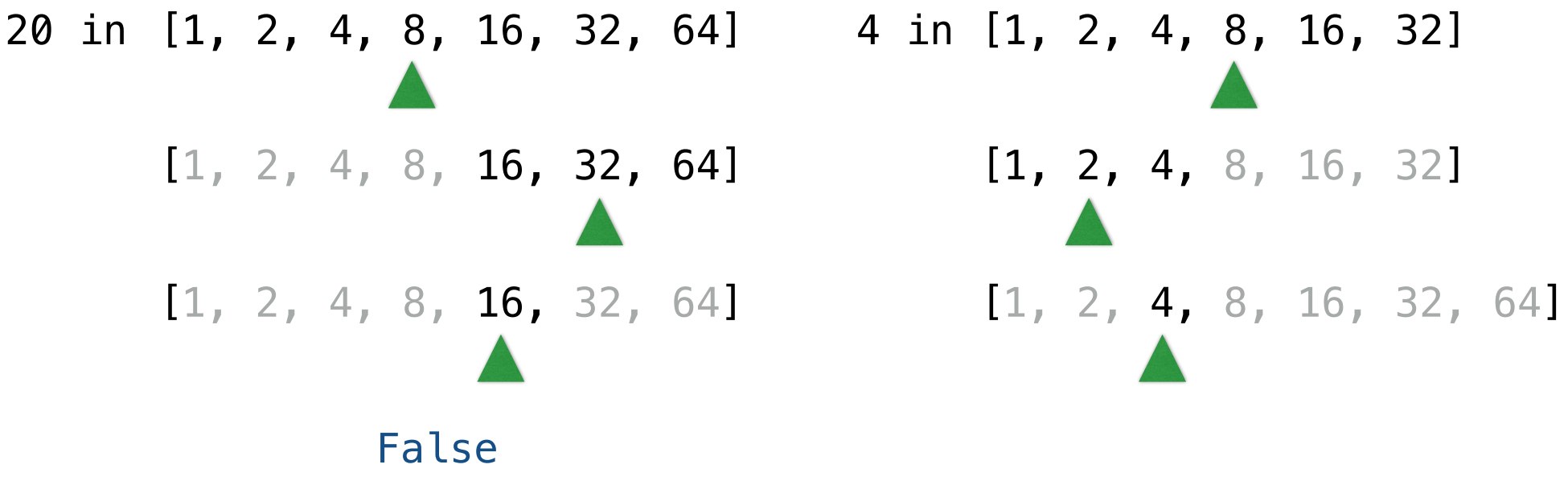
Binary Search

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



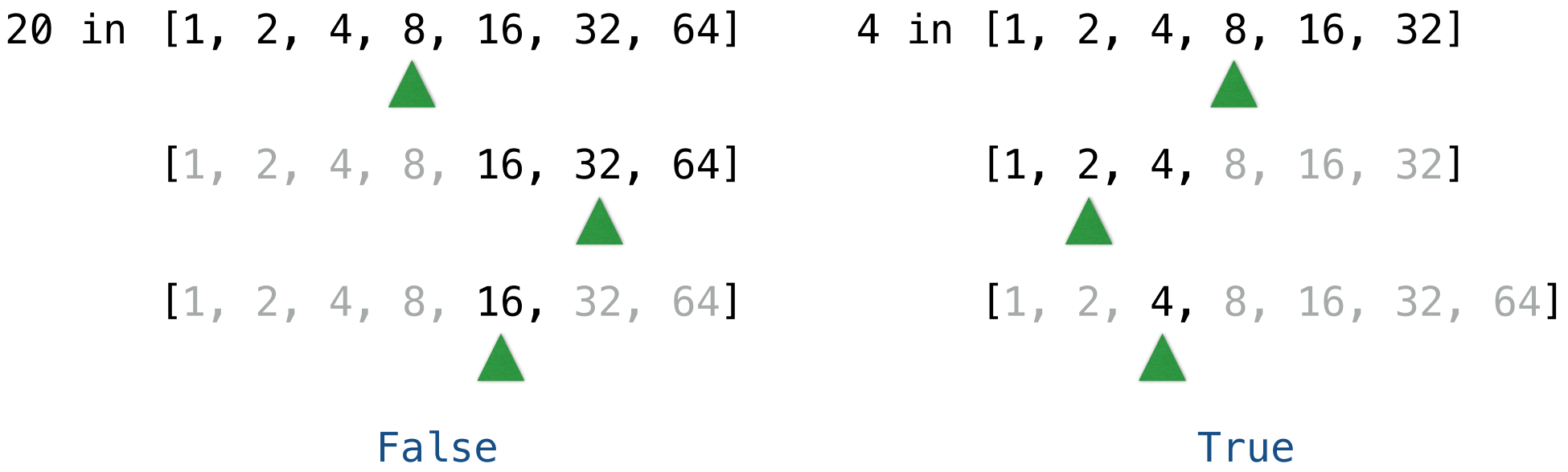
Binary Search

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



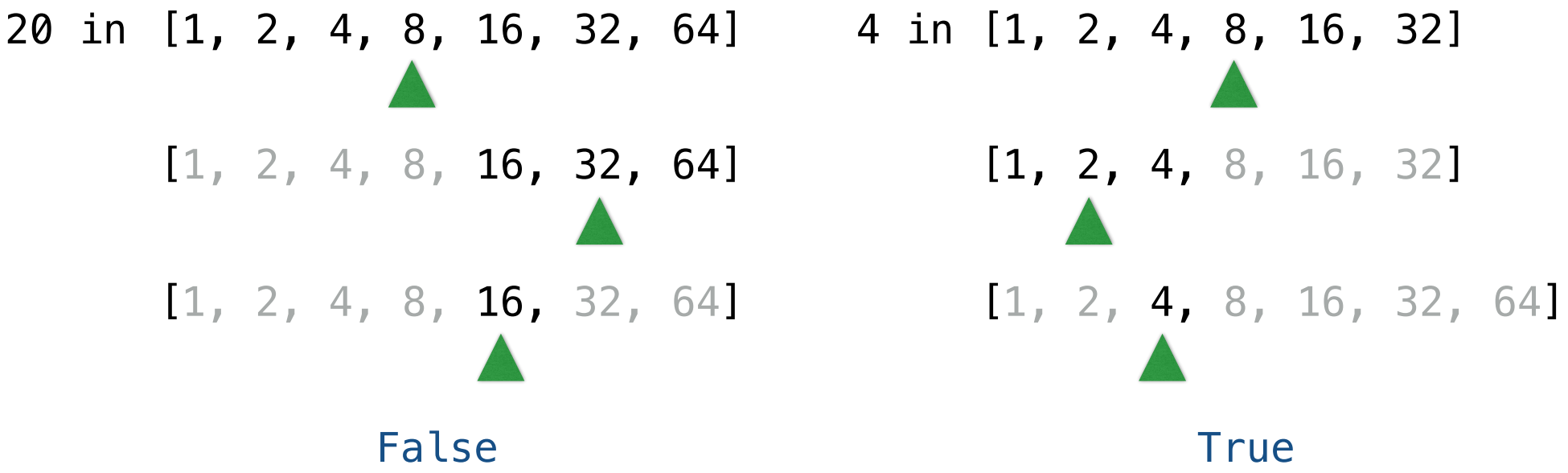
Binary Search

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



Binary Search

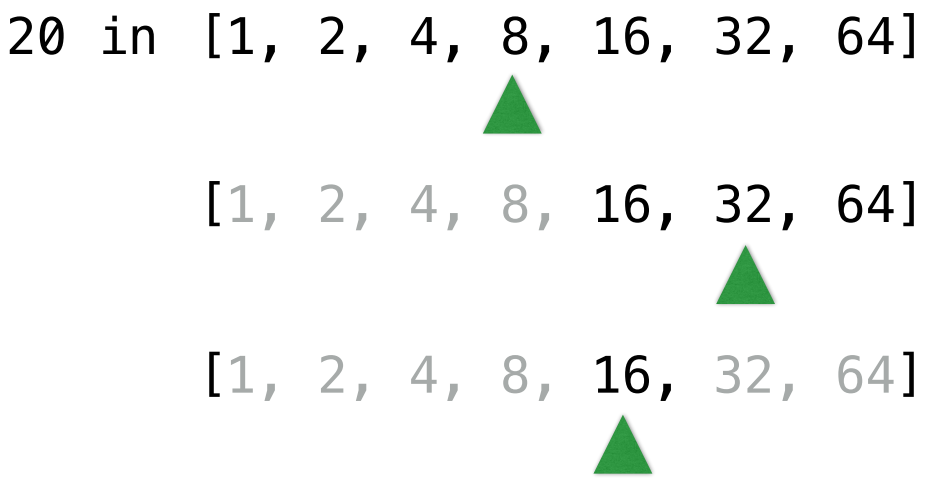
A strategy for finding a value in a sorted list: check the middle and eliminate half

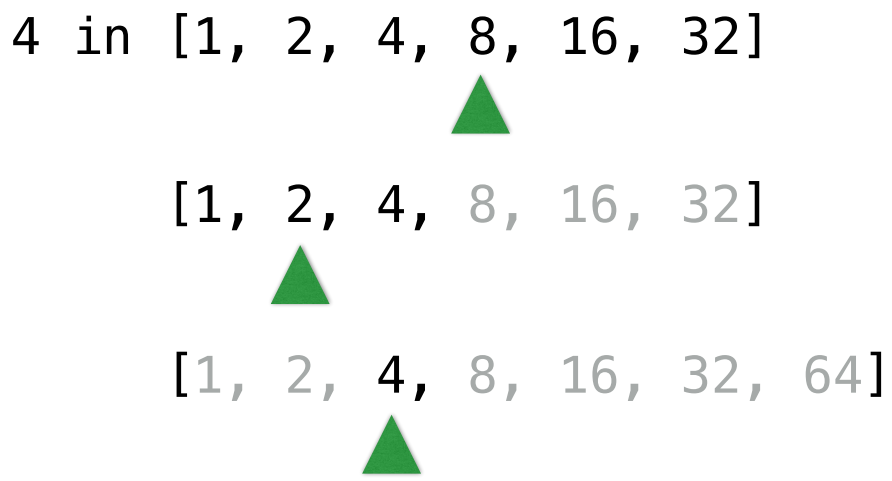


For a sorted list of length n, what Theta expression describes the time required?

Binary Search

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





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 root value is:

Binary Search Trees

A binary search tree is a binary tree where each root value is:

- Larger than all entries in its left branch and

Binary Search Trees

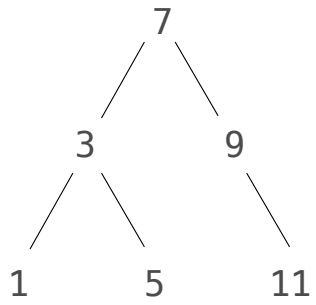
A binary search tree is a binary tree where each root value is:

- Larger than all entries in its left branch and
- Smaller than all entries in its right branch

Binary Search Trees

A binary search tree is a binary tree where each root value is:

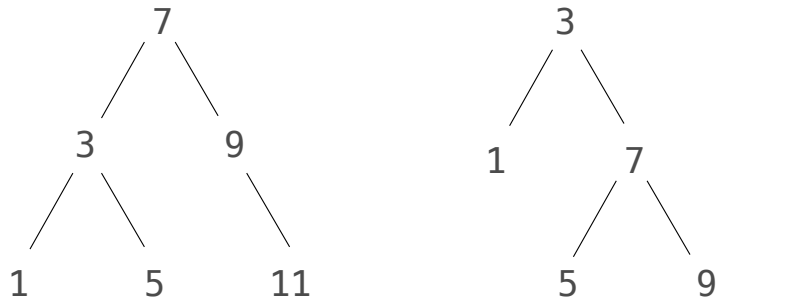
- Larger than all entries in its left branch and
- Smaller than all entries in its right branch



Binary Search Trees

A binary search tree is a binary tree where each root value is:

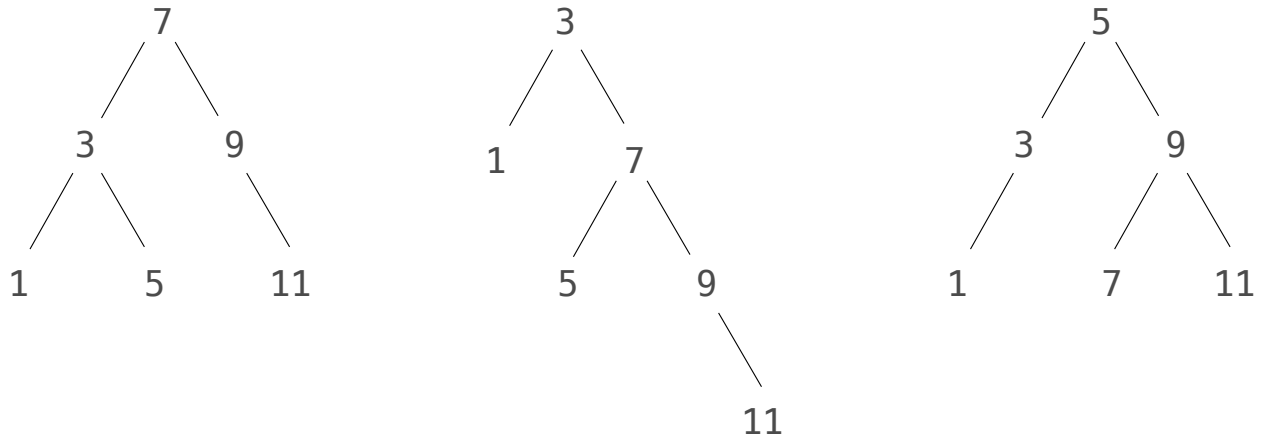
- Larger than all entries in its left branch and
- Smaller than all entries in its right branch



Binary Search Trees

A binary search tree is a binary tree where each root value is:

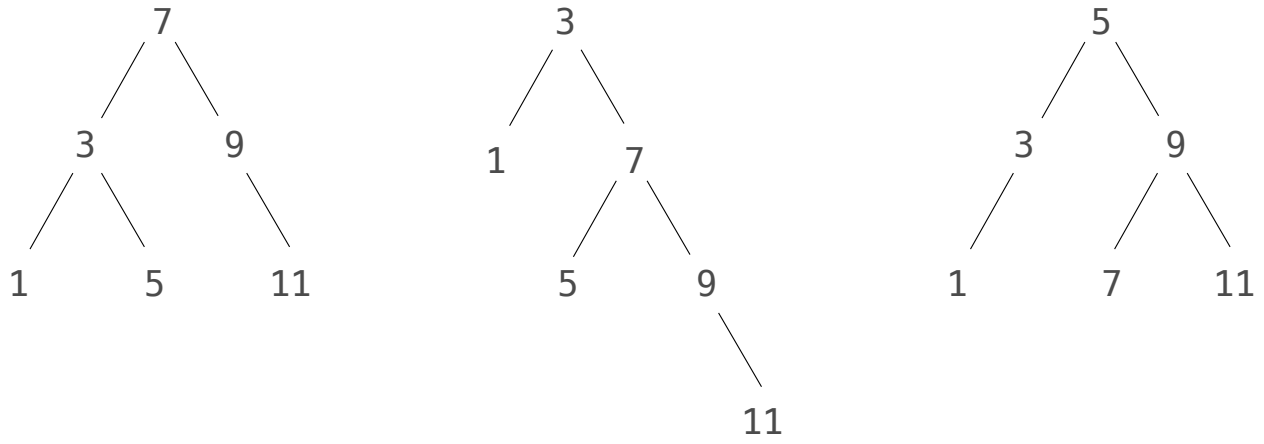
- Larger than all entries in its left branch and
- Smaller than all entries in its right branch



Binary Search Trees

A binary search tree is a binary tree where each root value is:

- Larger than all entries in its left branch and
- Smaller than all entries in its right branch



(Demo)

Discussion Questions

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

Discussion Questions

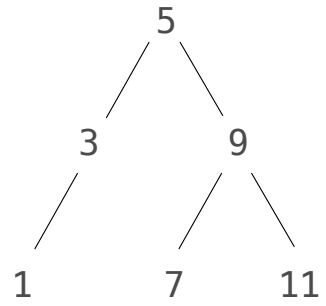
What's the largest element
in a binary search tree?

```
def largest(t):
    if ____________________________:
        return ___________________
    else:
        return ___________________
```

Discussion Questions

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

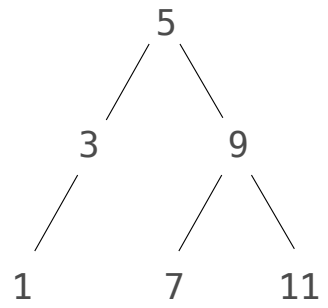
```
def largest(t):
    if ____________________________:
        return ___________________
    else:
        return ___________________
```

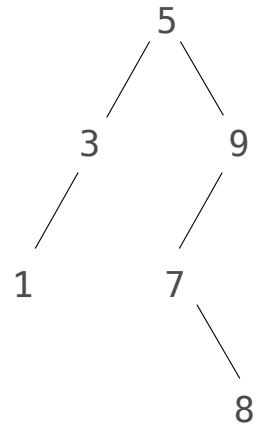


Discussion Questions

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

```
def largest(t):
    if ____________________________:
        return ___________________
    else:
        return ___________________
```

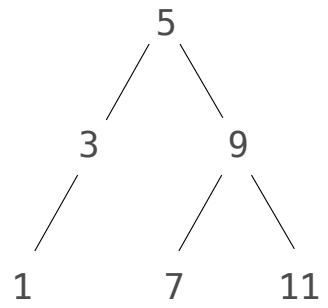


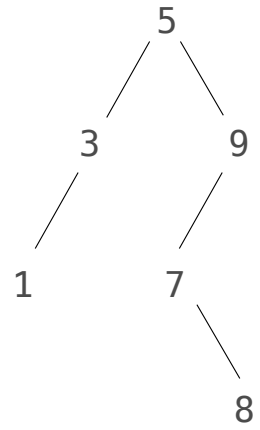


Discussion Questions

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

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

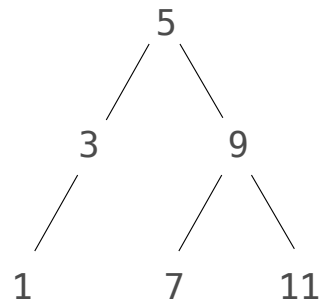


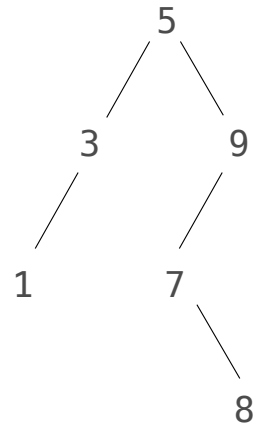


Discussion Questions

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

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



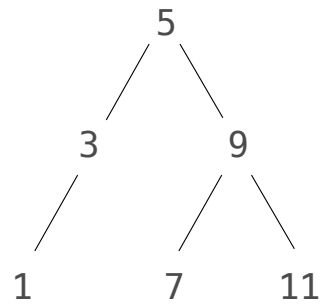


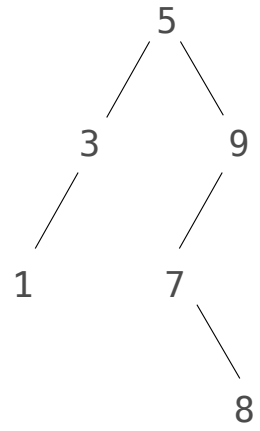
Discussion Questions

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

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

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





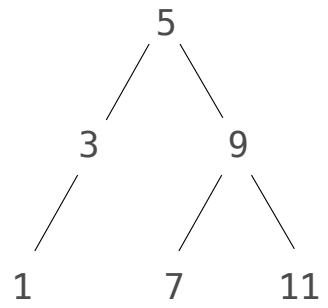
Discussion Questions

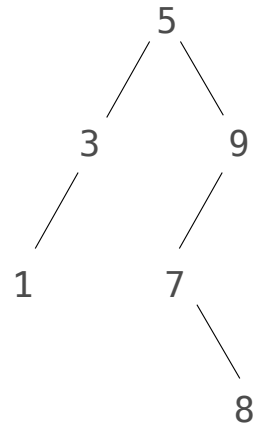
What's the largest element in a binary search tree?

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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 t.root
    elif _____________________:
        return _____________
    else:
        return _____________
```





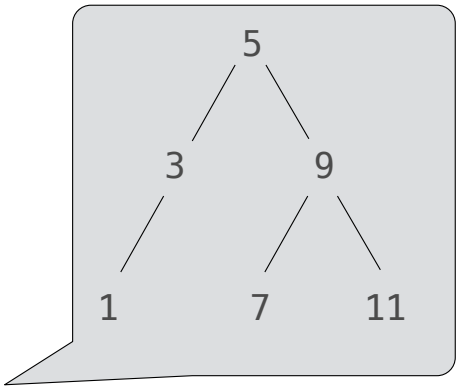
Discussion Questions

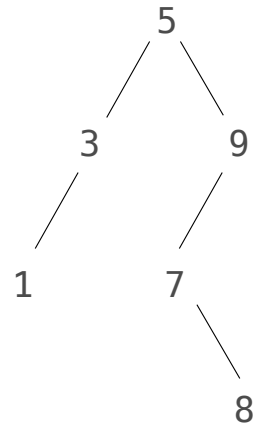
What's the largest element in a binary search tree?

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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 t.root
    elif _____________________:
        return _____________
    else:
        return _____________
```





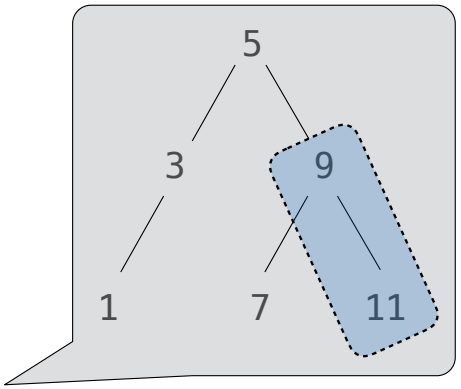
Discussion Questions

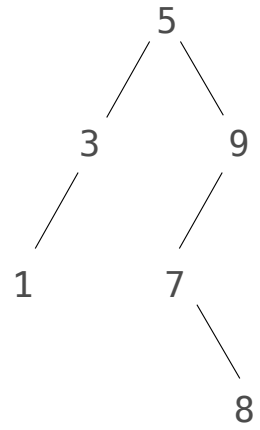
What's the largest element in a binary search tree?

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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 t.root
    elif _____________________:
        return _____________
    else:
        return _____________
```





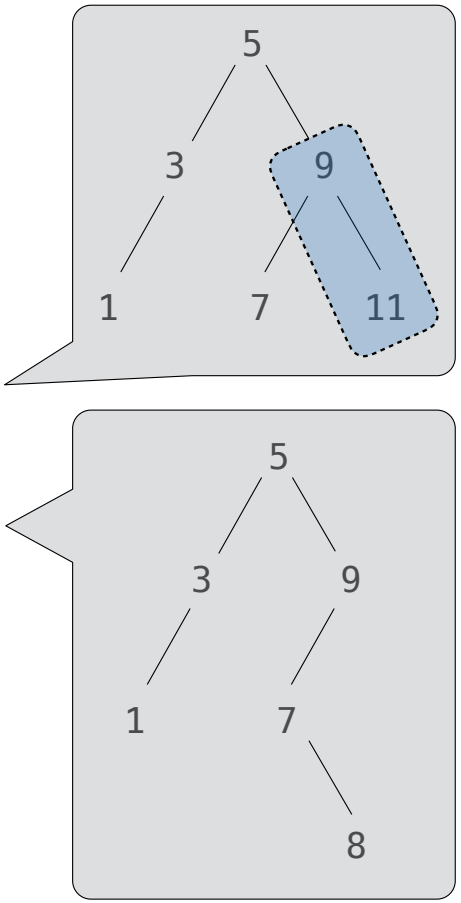
Discussion Questions

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

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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 t.root
    elif _____________________:
        return _____________
    else:
        return _____________
```



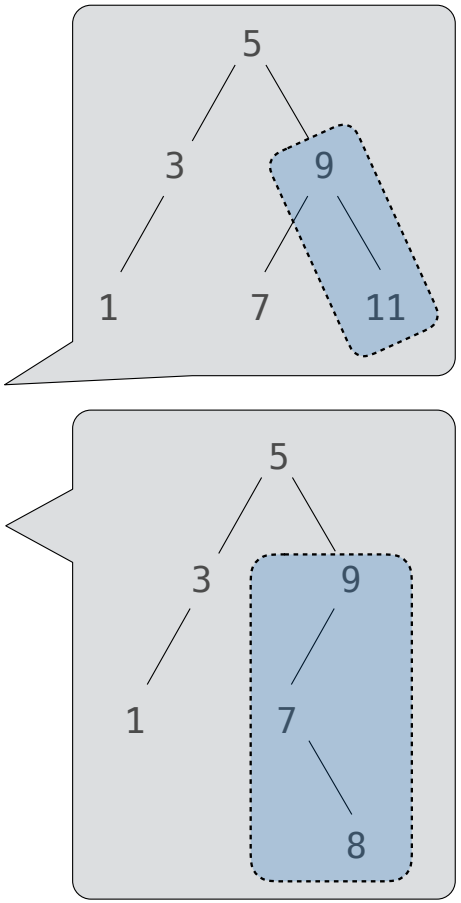
Discussion Questions

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

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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 t.root
    elif _____________________:
        return _____________
    else:
        return _____________
```



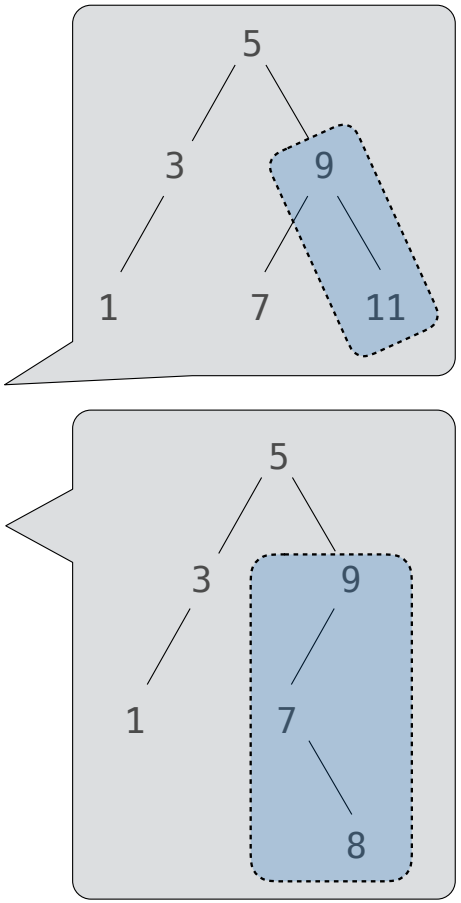
Discussion Questions

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

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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_leaf():
        return t.root
    elif _____:
        return _____
    else:
        return _____
```



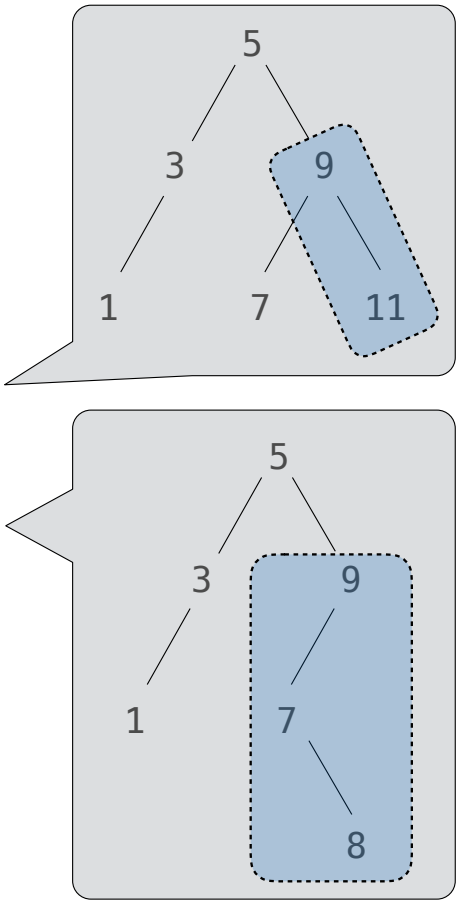
Discussion Questions

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

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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_leaf():
        return t.root
    elif t.right is BTree.empty:
        return _____
    else:
        return _____
```



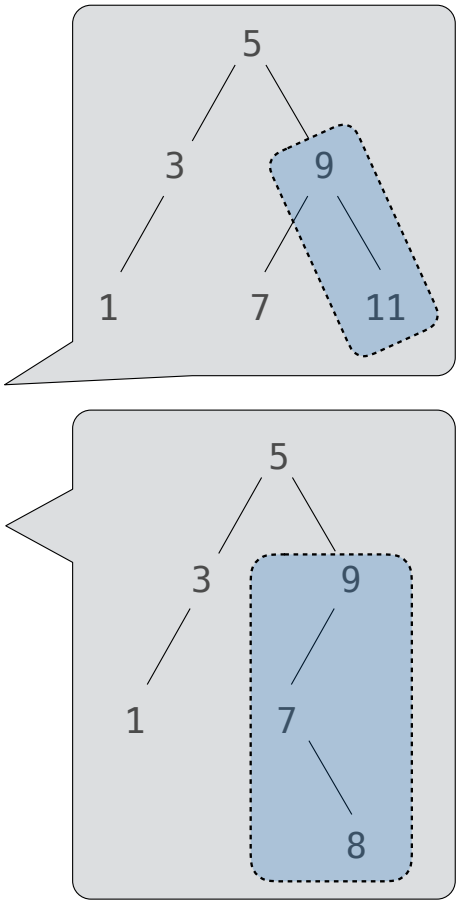
Discussion Questions

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

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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_leaf():
        return t.root
    elif t.right is BTree.empty:
        return largest(t.left)
    else:
        return _____
```



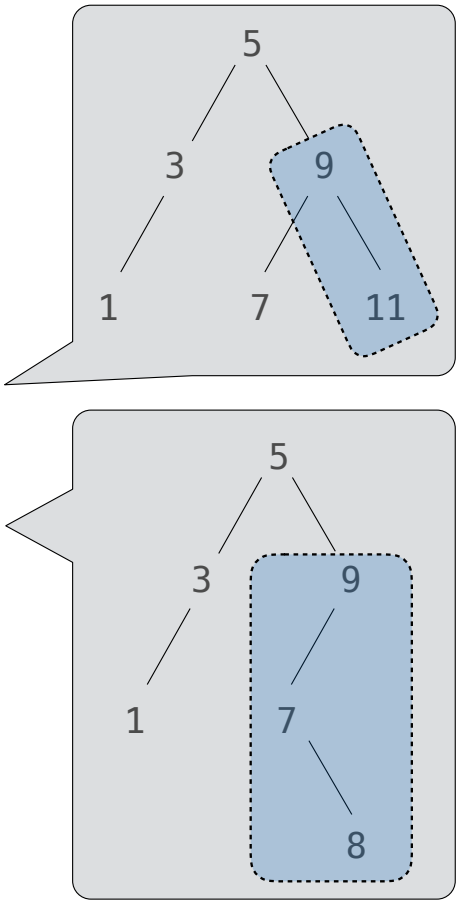
Discussion Questions

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

```
def largest(t):
    if t.right is BTree.empty:
        return t.root
    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_leaf():
        return t.root
    elif t.right is BTree.empty:
        return largest(t.left)
    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 the root, it can only be in either the left or right branch

Membership in Binary Search Trees

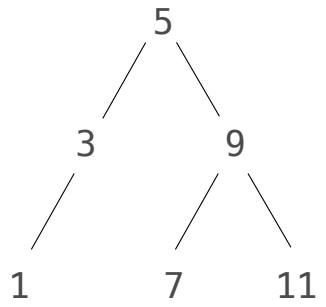
`contains` traverses the tree

- If the element is not 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

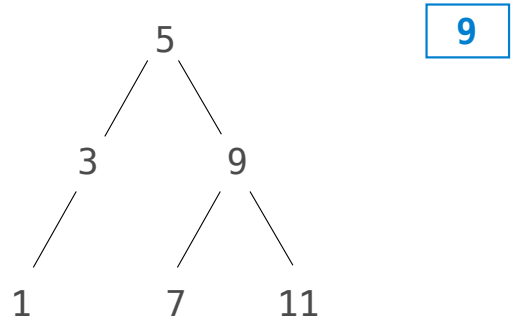
- If the element is not the root, it can only be in either the left or right branch
- By focusing on one branch, we reduce the set by the size of the other branch



Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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



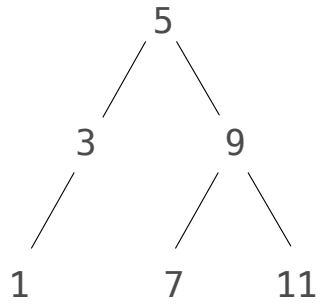
Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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):
```

9

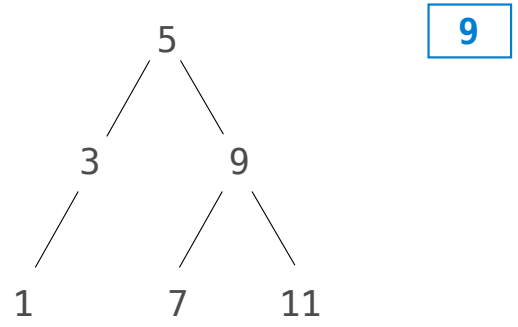


Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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
```

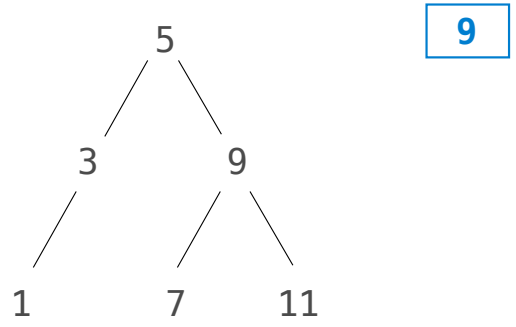


Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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.root == v:
        return True
```

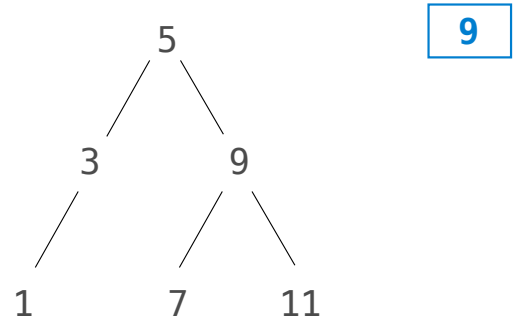


Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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.root == v:
        return True
    elif s.root < v:
        return contains(s.right, v)
```

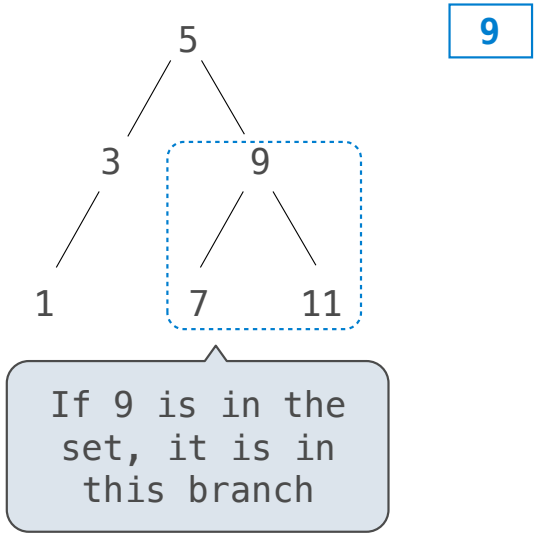


Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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.root == v:
        return True
    elif s.root < v:
        return contains(s.right, v)
```

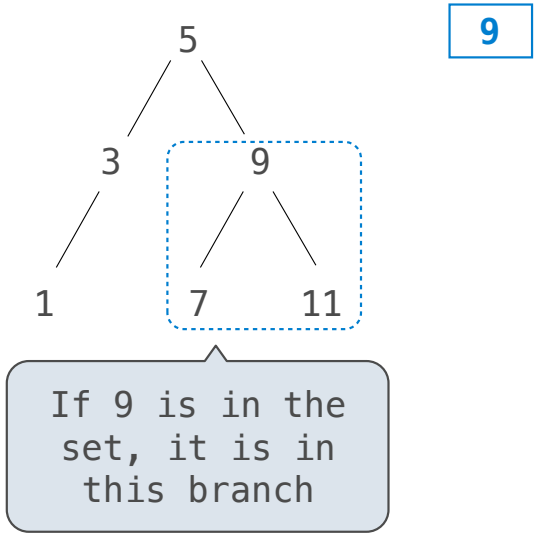


Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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.root == v:
        return True
    elif s.root < v:
        return contains(s.right, v)
    elif s.root > v:
        return contains(s.left, v)
```

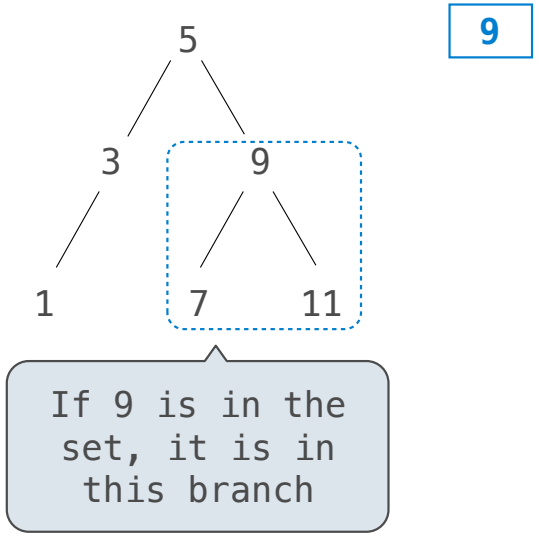


Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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.root == v:
        return True
    elif s.root < v:
        return contains(s.right, v)
    elif s.root > v:
        return contains(s.left, v)
```



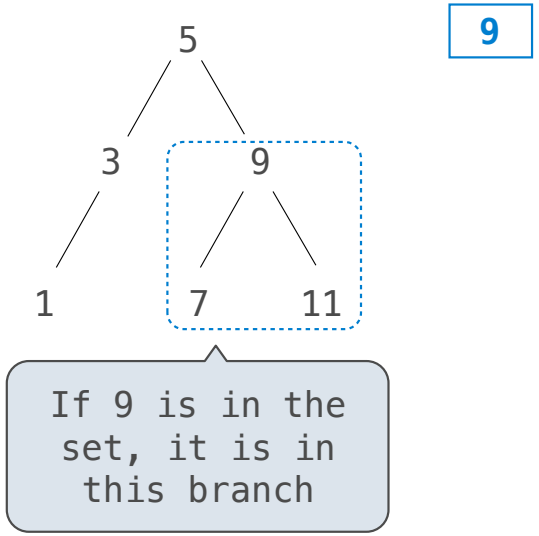
Order of growth?

Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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.root == v:
        return True
    elif s.root < v:
        return contains(s.right, v)
    elif s.root > v:
        return contains(s.left, v)
```



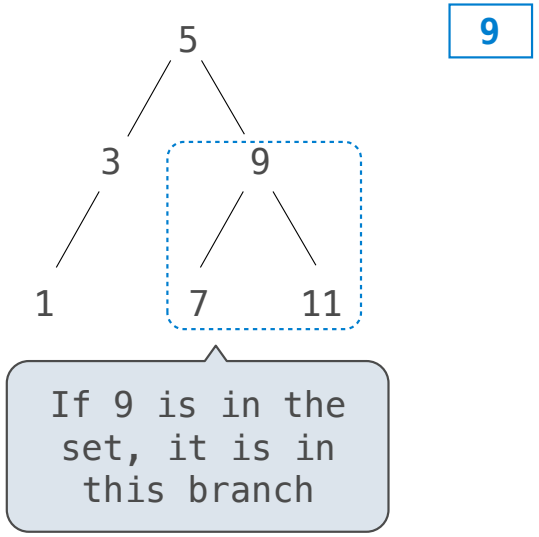
Order of growth? $\Theta(h)$ on average

Membership in Binary Search Trees

contains traverses the tree

- If the element is not 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

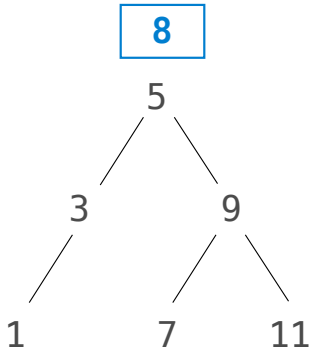
```
def contains(s, v):
    if s is BTree.empty:
        return False
    elif s.root == v:
        return True
    elif s.root < v:
        return contains(s.right, v)
    elif s.root > v:
        return contains(s.left, v)
```



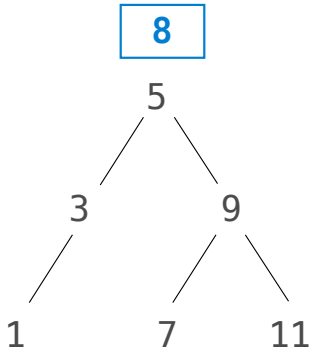
Order of growth? $\Theta(h)$ on average $\Theta(\log n)$ on average for a balanced tree

Adjoining to a Tree Set

Adjoining to a Tree Set

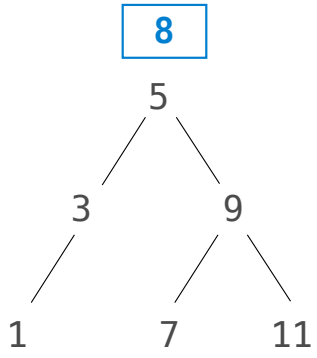


Adjoining to a Tree Set



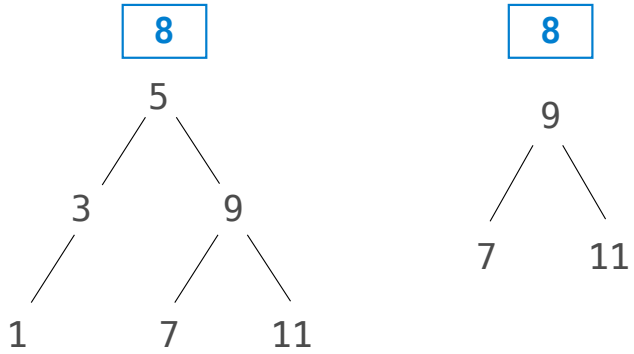
Right!

Adjoining to a Tree Set



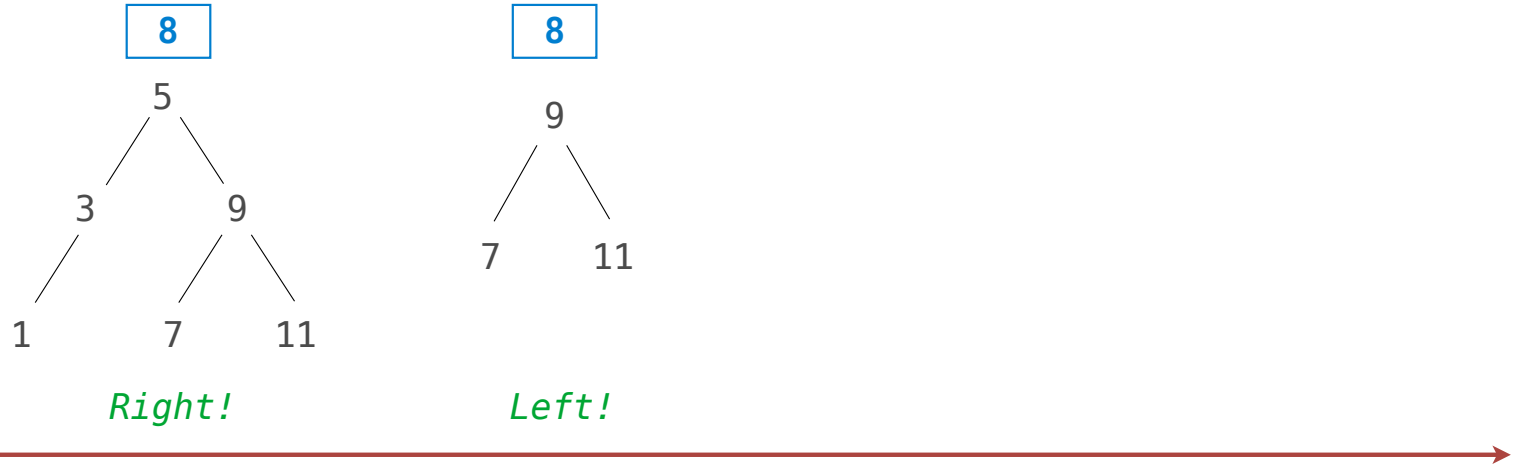
Right!

Adjoining to a Tree Set

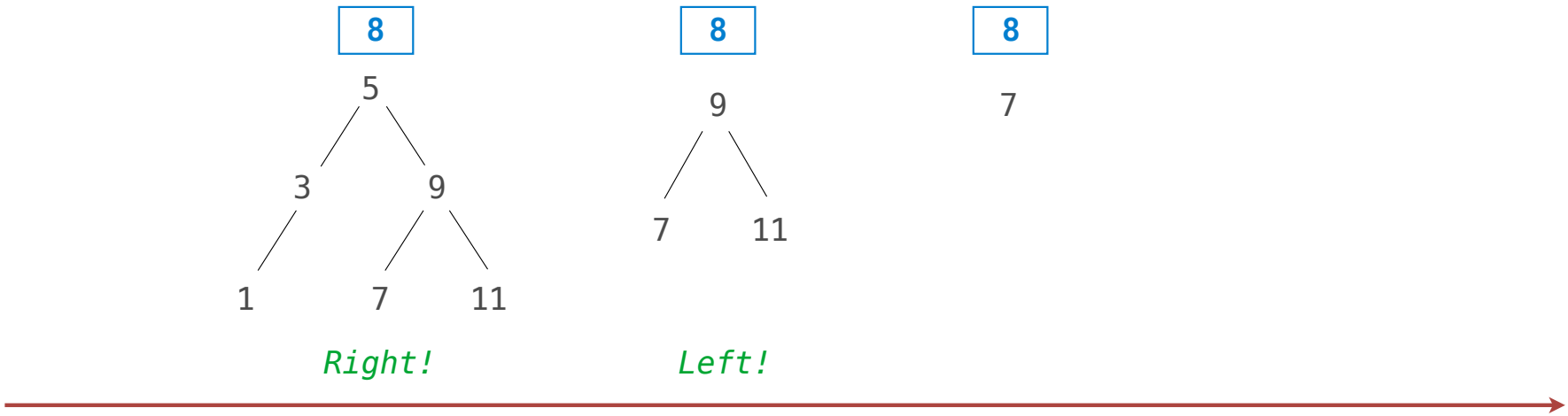


Right!

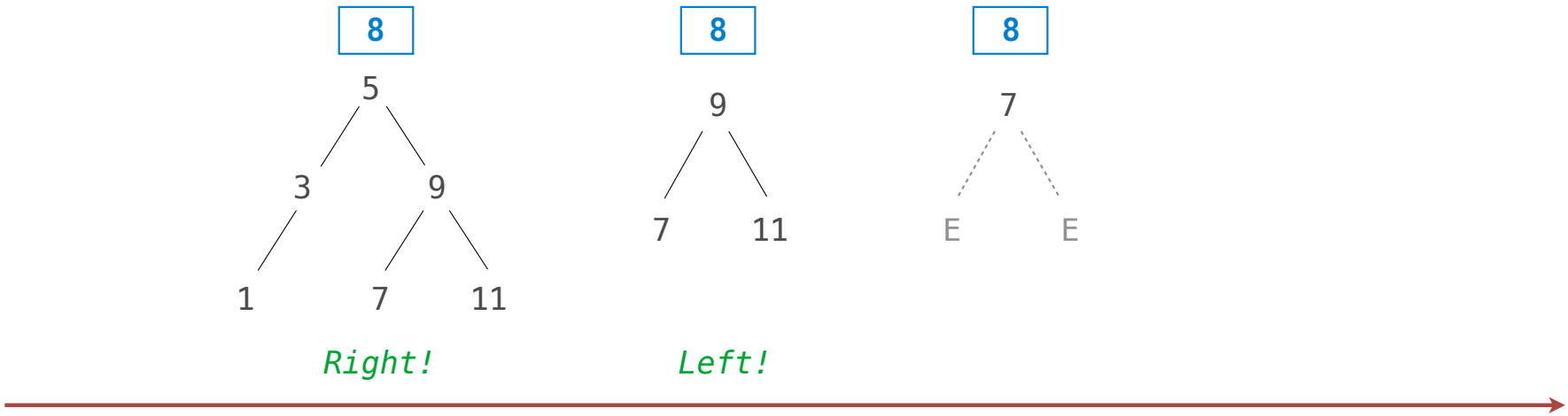
Adjoining to a Tree Set



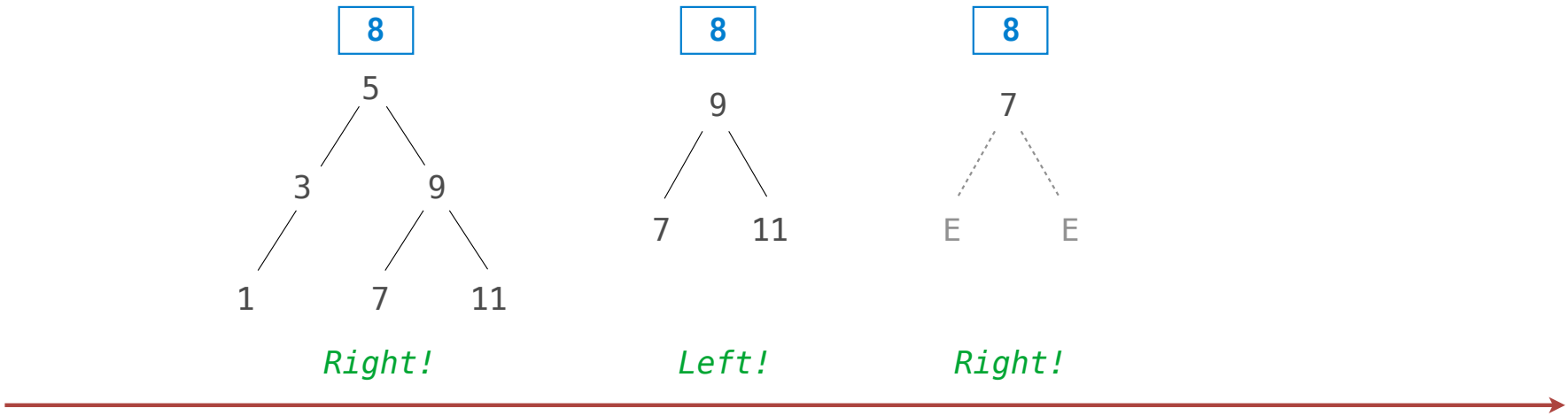
Adjoining to a Tree Set



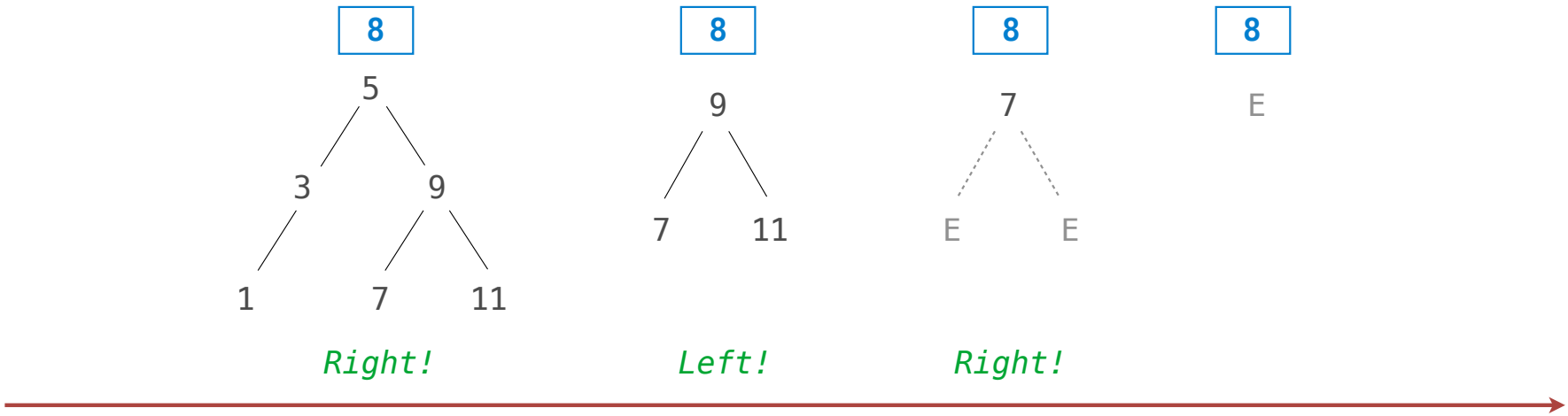
Adjoining to a Tree Set



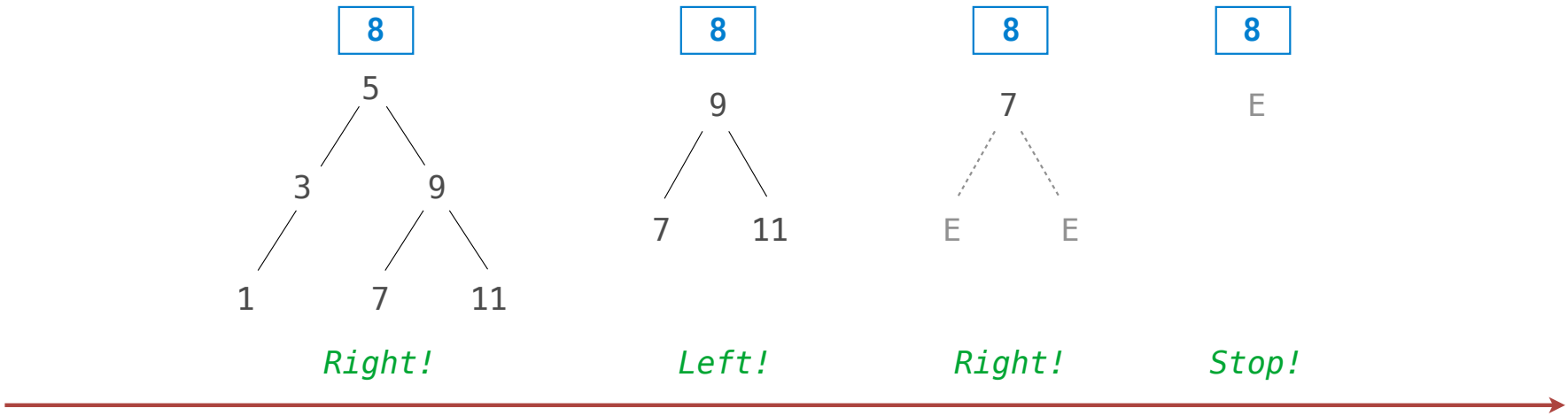
Adjoining to a Tree Set



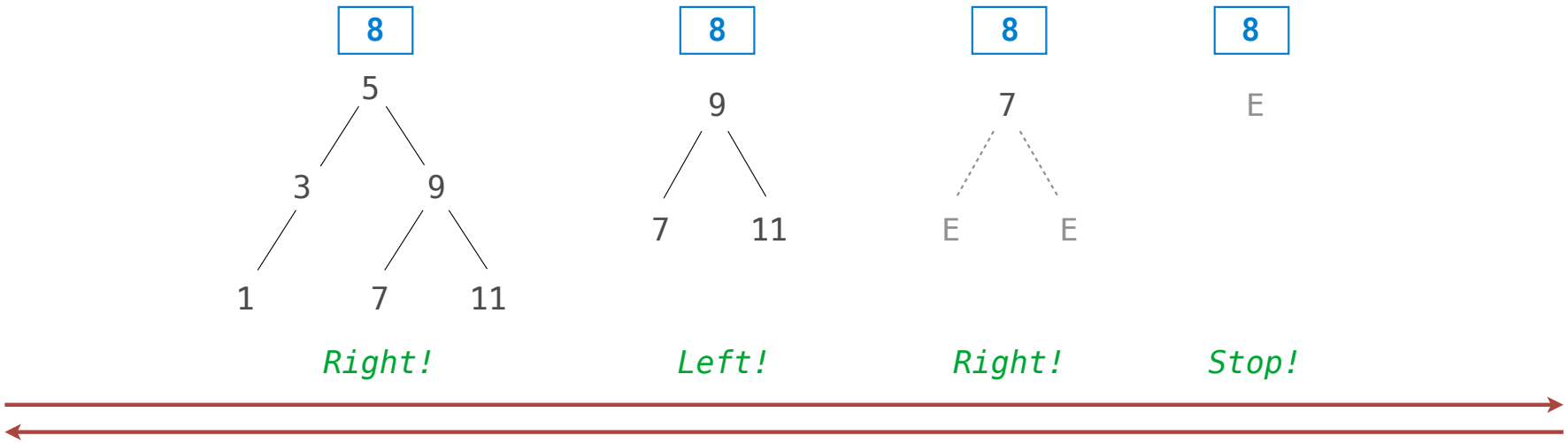
Adjoining to a Tree Set



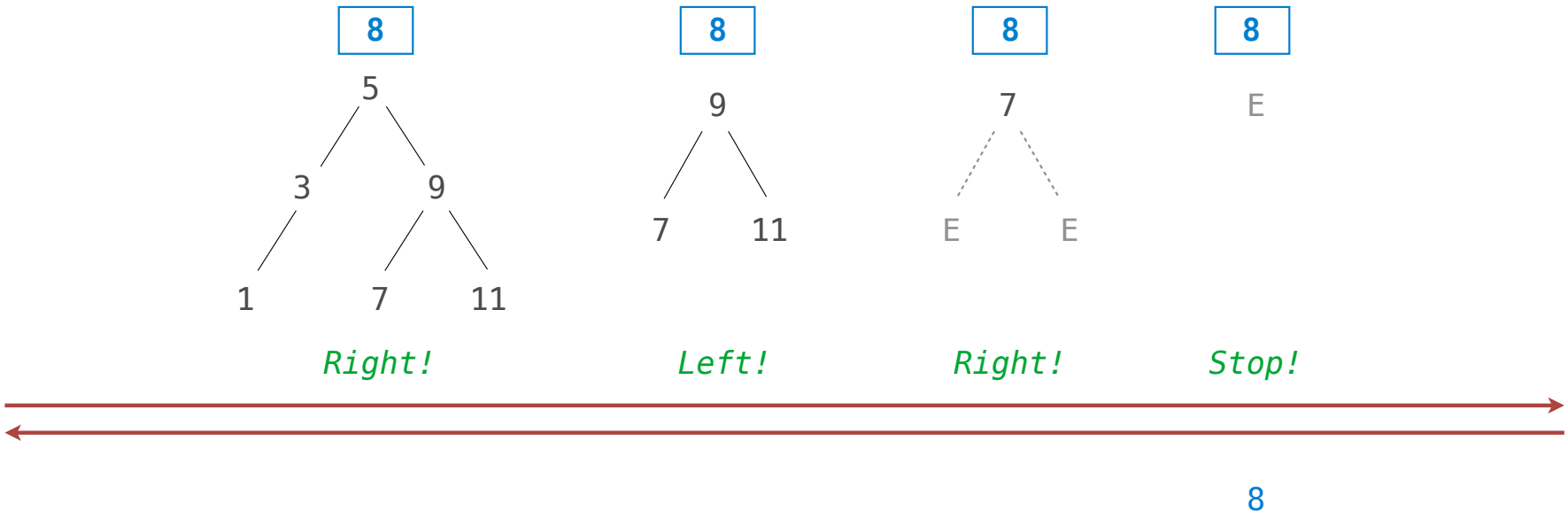
Adjoining to a Tree Set



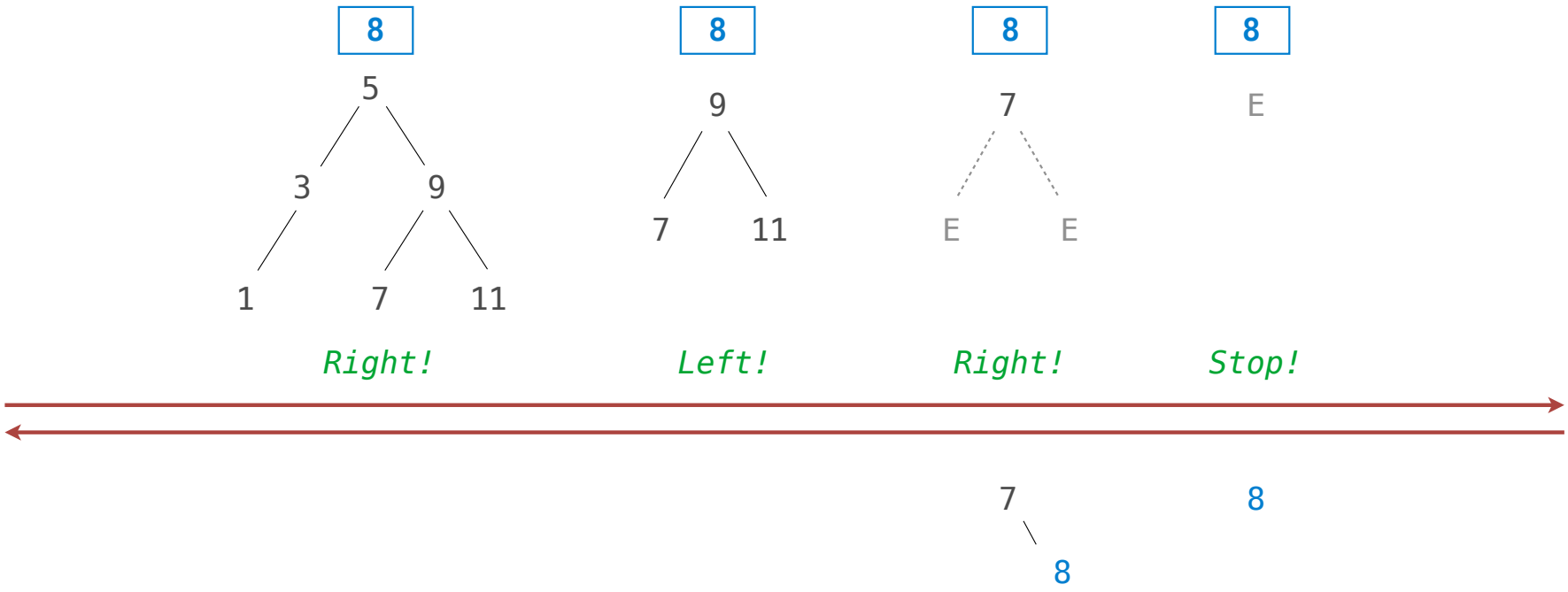
Adjoining to a Tree Set



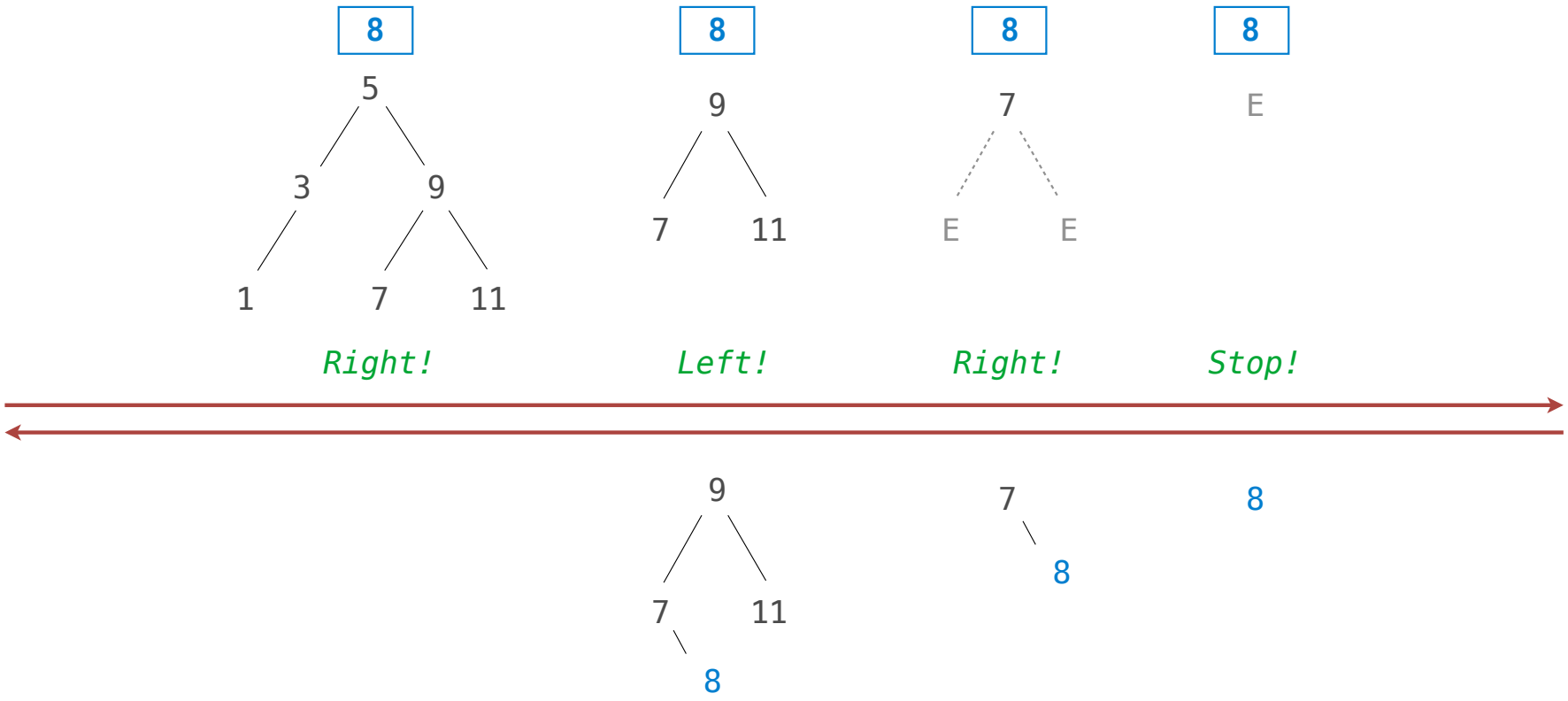
Adjoining to a Tree Set



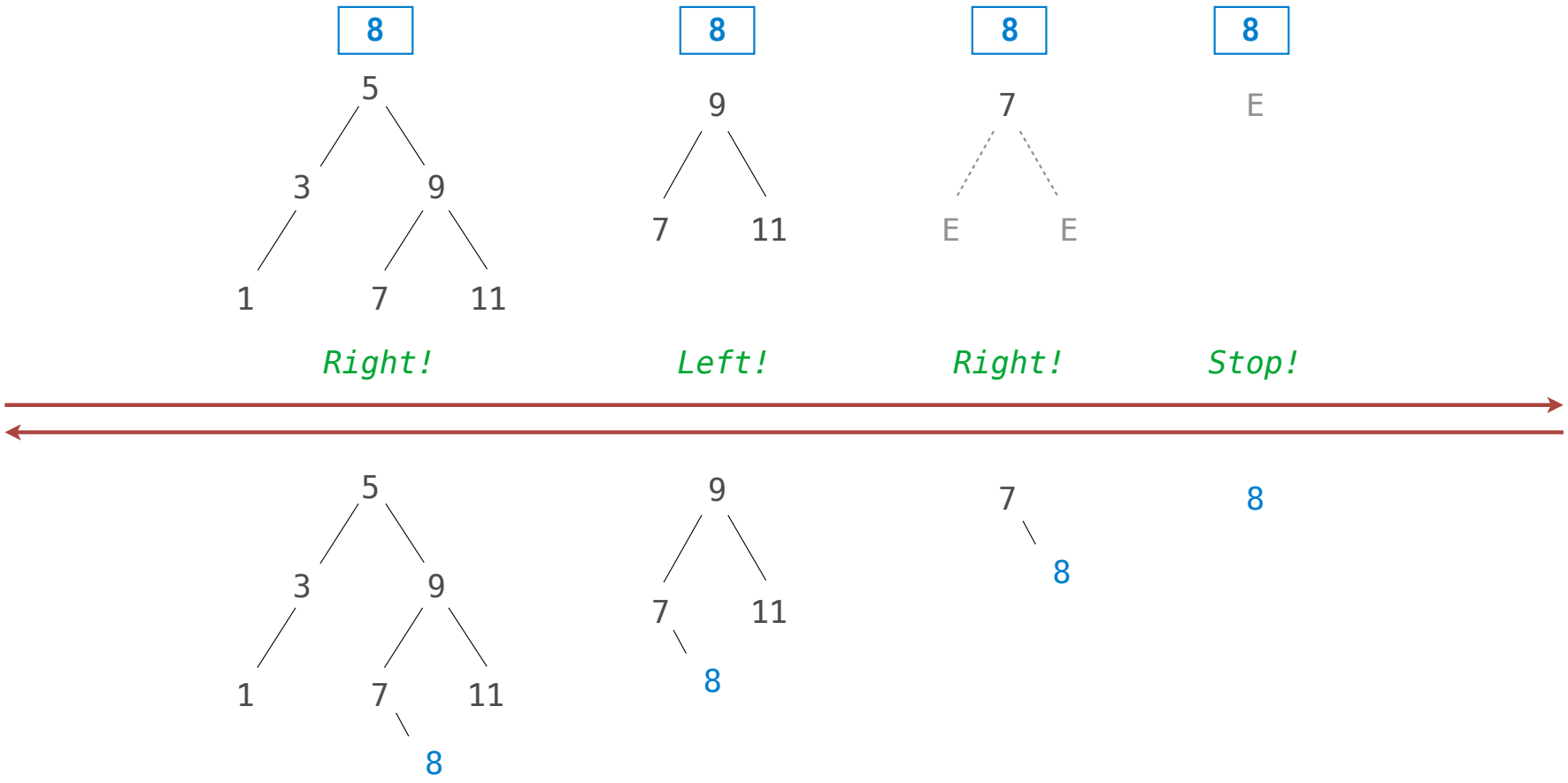
Adjoining to a Tree Set



Adjoining to a Tree Set



Adjoining to a Tree Set



Adjoining to a Tree Set