61A Lecture 21
Announcements
Binary Trees
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):

```
  3
 /   \
1     7
 / \
5   9
 /   \
11
```
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.

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

![Binary Tree Diagram]

```
```

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

![Binary Tree Diagram]
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):

```
  3
 /   \
1     7
 /     /\ \
5     9   E
 /     /   /  \
 E     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.

```python
class BTree(Tree):
    # class definition
```

![Binary Tree Diagram]

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)
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)
```

---

![Diagram of a binary tree with nodes 3, 1, 7, 5, 9, 11, and an empty tree at the bottom.](image-url)
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.

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

![Binary Tree Diagram](image)
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.

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

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

![Binary Tree Diagram]
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]

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.

```python
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]
```

E: An empty tree

```
3
  / 
 1 7
 /\ /\ 
E E 5 9
 /\ /\ /
 E E E 
```

```
11
 /\ /
 E E 
```
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.

```python
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)))))
```

E: An empty tree
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.

```python
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]
```

```python
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]
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 \([1, 2, 4, 8, 16, 32, 64]\)

\[\text{[1, 2, 4, 8, 16, 32, 64]}\]

\[\text{[1, 2, 4, 8, 16, 32, 64]}\]

\[\text{[1, 2, 4, 8, 16, 32, 64]}\]
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]

[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]

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

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]

[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]

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]

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

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] → False

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

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

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]

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

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

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

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

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]

[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]  
[1, 2, 4]  
False

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

```
    7
   /|
  /  \
3    9
 /    /
1    5  11
```
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
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
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
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 ______________

```
  5
 / 
3   9
 /   
1    7 11
```
Discussion Questions

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

def largest(t):
    if ____________________:
        return _____________
    else:
        return _________________

![Binary Search Tree Diagram]

```
1 3 5
|
1 3
|
1
```
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?

```python
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?

```python
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?

```python
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?

```python
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?

```python
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?

```python
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?

```python
def second(t):
    if t.is_leaf():
        return None
    elif _____: 
        return _____
    elif _____:
        return t.label
    else:
        return _____
```

```python
5 3 9
1 7 11
```
Discussion Questions

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

```python
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?

```python
def second(t):
    if t.is_leaf():
        return None
    elif _____
        return _____
    elif _____
        return t.label
    else:
        return _____
```

```latex
\begin{center}
\begin{tikzpicture}
  \node (5) {5} child {node (3) {3} child {node (1) {1}}}
  \node (9) {9} child {node (7) {7}}
  \node (11) {11}
\end{tikzpicture}
\end{center}
```
Discussion Questions

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

```python
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?

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

![Binary Search Tree Example](image)
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?

```python
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?

```python
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?

```python
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?

```python
def second(t):
    if t.is_leaf():
        return None
    elif ______ t.right.is_leaf_____:
        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?

```python
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?

```python
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?

```python
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?

```python
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

```
    5
   / \
  3   9
 /   / \ 
1   7   11
```
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

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

```
  5
 /  \\n3    9
 /    /\n1  7  11
```
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

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

![Binary Search Tree Diagram]
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

```python
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

```python
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)
```

---

![Binary Search Tree Diagram](image-url)
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

```python
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)
```

If 9 is in the set, it is in this 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

```python
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

```python
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)
```

If 9 is in the set, it is in this branch

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

```python
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)
    else:
        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

```
8
/  \
5   /
/    /
3    9
/ \\
1 7 11
```
Adjoining to a Tree Set

Right!
Adjoining to a Tree Set

Right!
Adjoining to a Tree Set

Right!
Adjoining to a Tree Set

Right!  

Left!
Adjoining to a Tree Set

Right!  Left!
Adjoining to a Tree Set

Right!  Left!
Adjoining to a Tree Set

```
5
3 7
1 7 11
```

```
9
7 11
```

```
7
E E
```

*Right!*  *Left!*  *Right!*
Adjoining to a Tree Set

Right!  Left!  Right!
Adjoining to a Tree Set

- Right!
- Left!
- Right!
- Stop!
Adjoining to a Tree Set

Right!  Left!  Right!  Stop!
Adjoining to a Tree Set

```
Adjoining to a Tree Set

Right!   Left!   Right!   Stop!
```

```
Right!

Left!

Right!

Stop!
```
Adjoining to a Tree Set

Right!

Left!

Right!

Stop!
Adjoining to a Tree Set

Right!  Left!  Right!  Stop!
Adjoining to a Tree Set

Right!  Left!  Right!  Stop!
Adjoining to a Tree Set

Right!  Left!  Right!  Stop!

(Demo)