


Trees

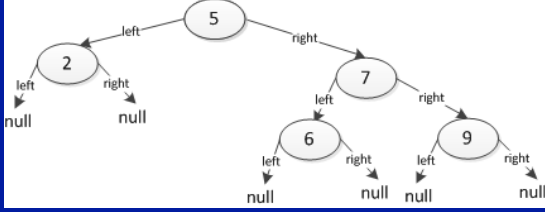
Snarf today's code



Today

- Binary Trees
- Recursion and Trees
- Binary Search Trees
- By the end of class
 - You will be able to articulate what makes binary search trees so powerfully efficient - including understanding the runtime of the mysterious TreeSet

Binary Tree



```

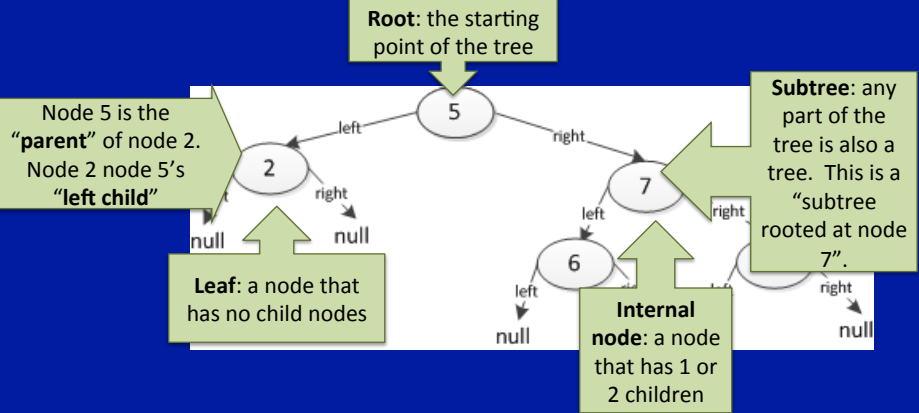
IntTreeNode root = null;

public class IntTreeNode {
    public int myValue;
    public IntTreeNode myLeft; // holds smaller tree nodes
    public IntTreeNode myRight; // holds larger tree nodes

    public IntTreeNode(int val) { myValue = val; }
}

```

Binary Tree



Root: the starting point of the tree

Node 5 is the **“parent”** of node 2. Node 2 node 5’s **“left child”**

Leaf: a node that has no child nodes

Internal node: a node that has 1 or 2 children

Subtree: any part of the tree is also a tree. This is a “subtree rooted at node 7”.

Binary Tree

Depth: distance of a node from the root

```

graph TD
    5((5)) -- left --> 2((2))
    5 -- right --> 7((7))
    2 -- left --> null1[null]
    2 -- right --> null2[null]
    7 -- left --> 6((6))
    7 -- right --> 9((9))
    6 -- left --> null3[null]
    6 -- right --> null4[null]
    9 -- left --> null5[null]
    9 -- right --> null6[null]
  
```

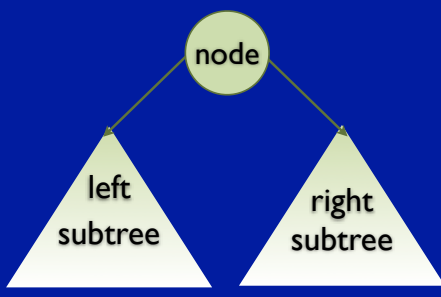
Height: maximum depth of the tree

Today

- Binary Trees
- Recursion and Trees
- Binary Search Trees
- By the end of class
 - You will be able to articulate what makes binary search trees so powerfully efficient - including understanding the runtime of the mysterious TreeSet

Trees and Recursion

- They go together like PB&J!
- Check current node
 - if no
 - check left subtree
 - check right subtree



```
graph TD; node((node)) --- leftSubtree[left subtree]; node --- rightSubtree[right subtree];
```

The diagram shows a central circular node labeled 'node' at the top. Two lines extend downwards from the node to two triangular shapes representing subtrees. The left triangle is labeled 'left subtree' and the right triangle is labeled 'right subtree'.

Trees and Recursion

- Example recursive tree code

```
public int computeTreeThing(TreeNode current) {
    if (we are at the base case) {
        return obviousValue;
    } else {
        int lResult = computeTreeThing(current.left);
        int rResult = computeTreeThing(current.right);
        int result = //combine those values;
        return result;
    }
}
```



Trees and Recursion

- Code
 - countNodes
 - containsNode
 - findMax

```
public int computeTreeThing(TreeNode current) {
    if (we are at the base case) {
        return obviousValue;
    } else {
        int lResult = computeTreeThing(current.left);
        int rResult = computeTreeThing(current.right);
        int result = //combine those values;
        return result;
    }
}
```



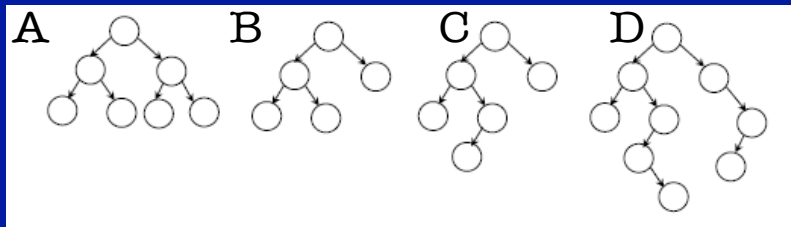
Trees and Recursion

- What is the running time?
 - countNodes
 - containsNode
 - findMax

```
public int computeTreeThing(TreeNode current) {
    if (we are at the base case) {
        return obviousValue;
    } else {
        int lResult = computeTreeThing(current.left);
        int rResult = computeTreeThing(current.right);
        int result = //combine those values;
        return result;
    }
}
```

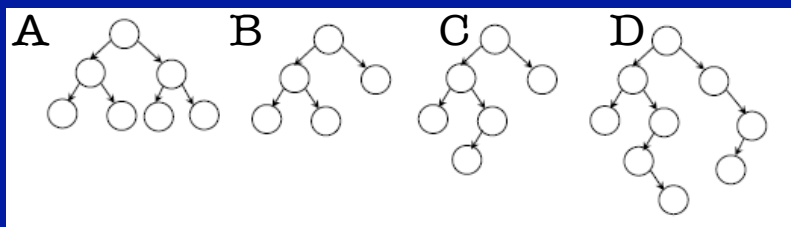
Binary Tree

- A tree is **height-balanced** if
 - left and right subtrees are both height balanced
 - the heights of left and right subtrees do not differ by more than 1



Binary Tree

- What is the height of a **height-balanced** tree?





Today

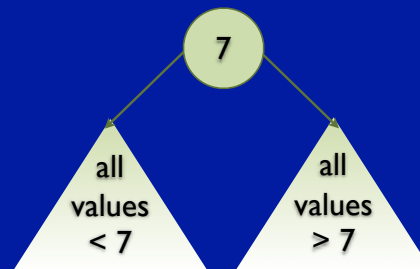
- Binary Trees
- Recursion and Trees
- Binary Search Trees

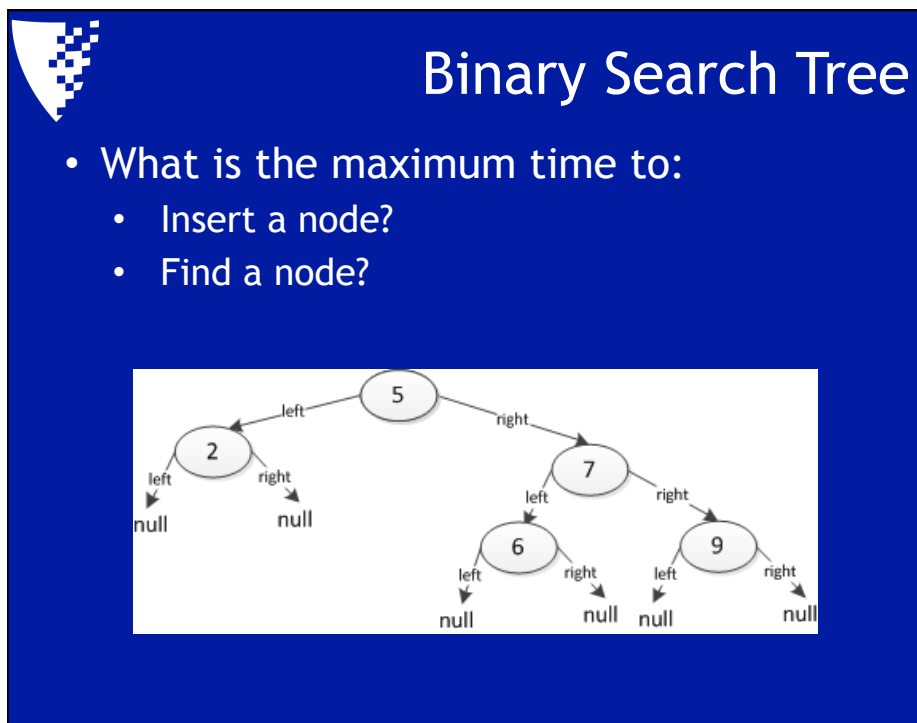
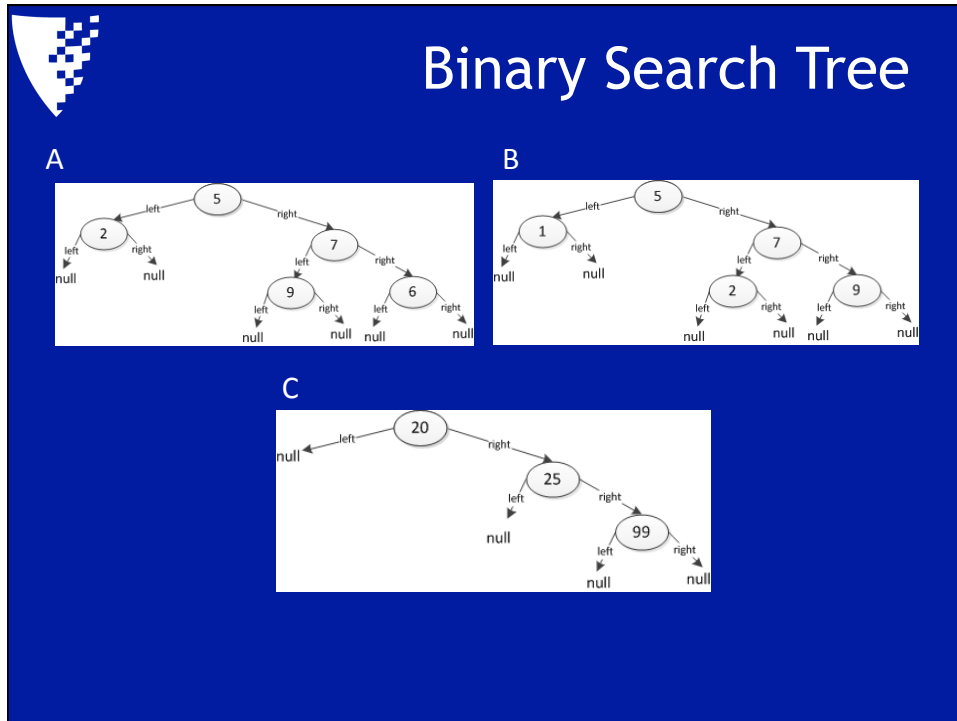
- By the end of class
 - You will be able to articulate what makes binary search trees so powerfully efficient - including understanding the runtime of the mysterious TreeSet



Binary Search Tree

- Each node has a value
- Nodes with values **less than** their parent are in the **left** subtree
- Nodes with values **greater than** their parent are in the **right** subtree







Today

- Binary Trees
- Recursion and Trees
- Binary Search Trees

- By the end of class
 - You will be able to articulate what makes binary search trees so powerfully efficient - including understanding the runtime of the mysterious TreeSet