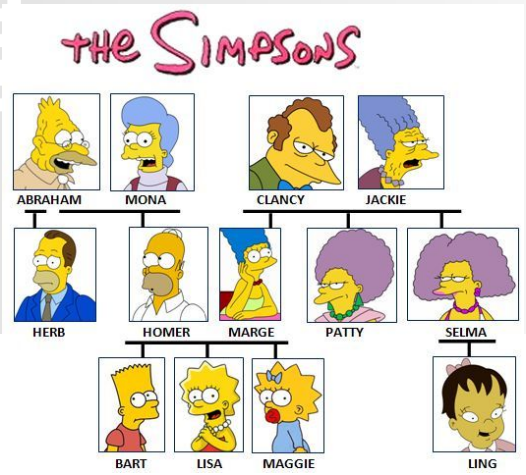


Trees

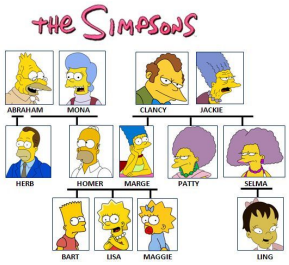
Snarf today's code

Announcements

- DNA – Due October 23
- APT Set 5 – Due October 28

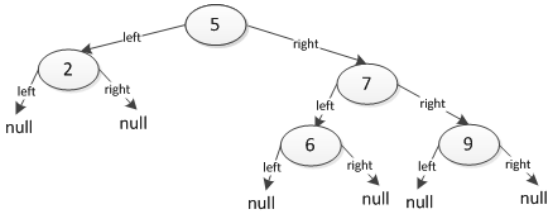
Today

- 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

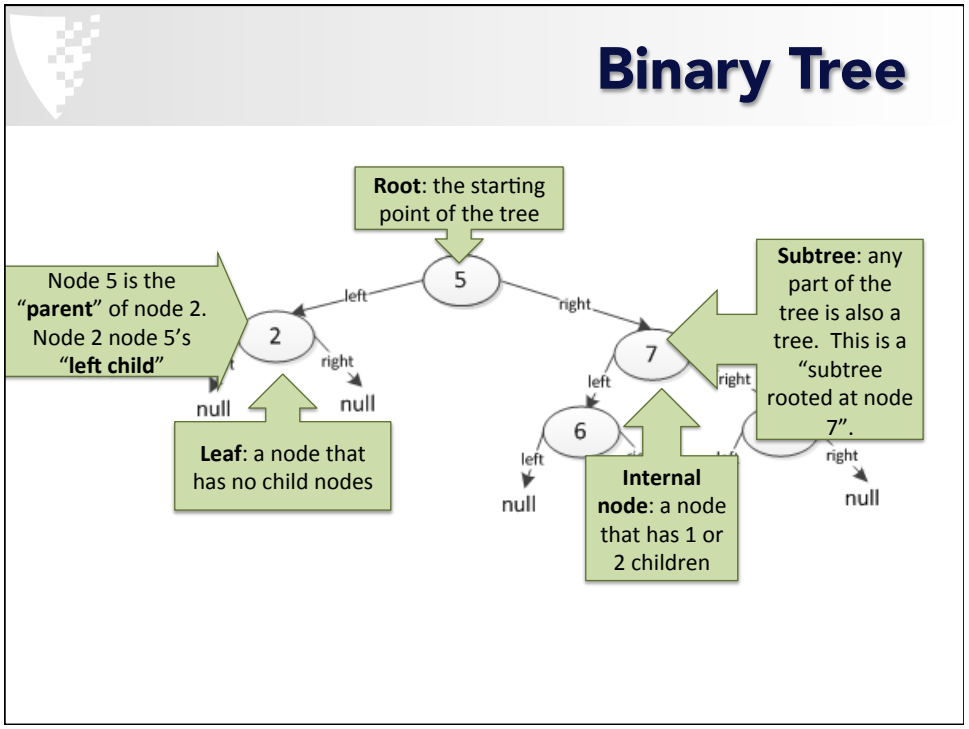
Binary Tree

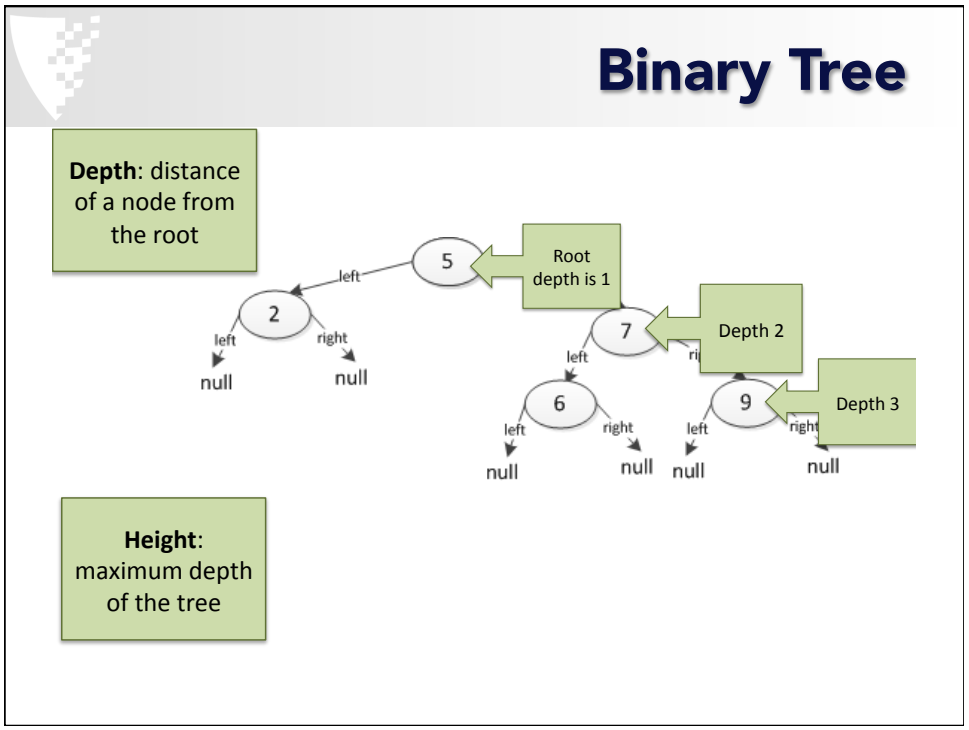



```
IntTreeNode root = null;

  public class IntTreeNode {
    public int myValue;
    public IntTreeNode myLeft;
    public IntTreeNode myRight;

    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"
Subtree: any part of the tree is also a tree. This is a "subtree rooted at node 7".
Leaf: a node that has no child nodes
Internal node: a node that has 1 or 2 children
5
2
7
6
left
right
null


Binary Tree
Depth: distance of a node from the root
Root depth is 1
Depth 2
Depth 3
Height: maximum depth of the tree
5
2
7
6
9
left
right
null

Today

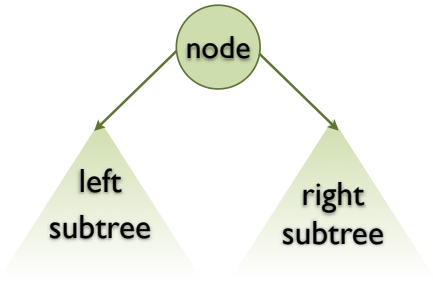
- Binary Trees
- Recursion and Trees
- Binary Search Trees

- 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



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;
}
```

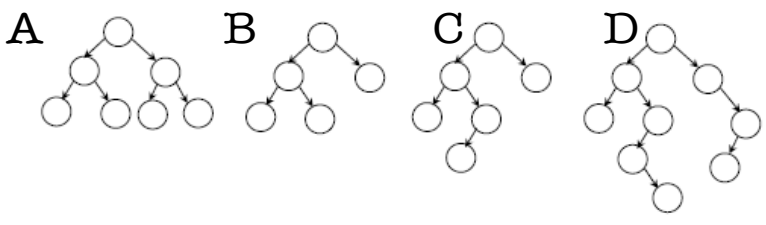
Trees and Recursion

- 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;
}
```

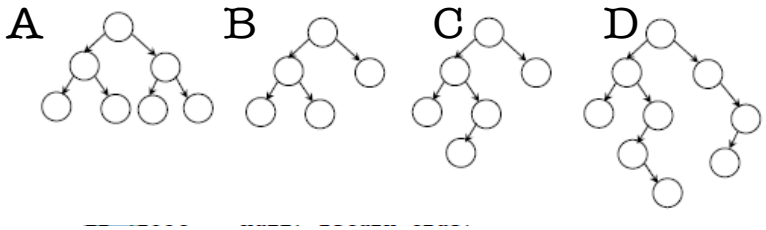
Binary Tree

- 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?



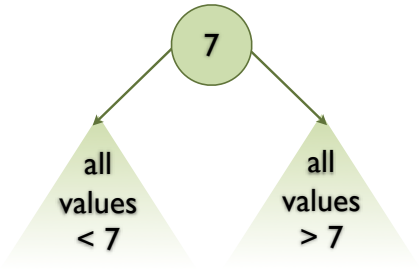
Today

- 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

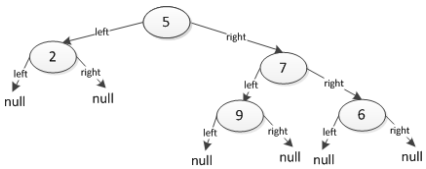
Binary Search Tree

- 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

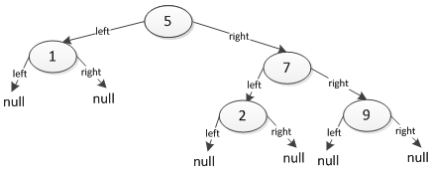


Binary Search Tree

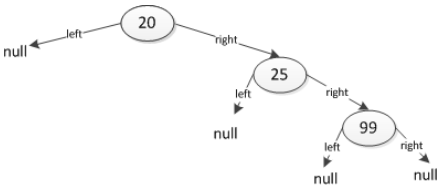
A



B

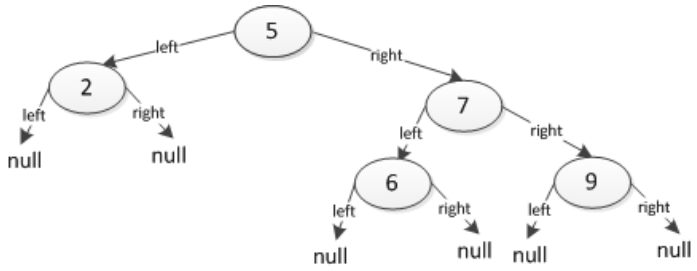


C



Binary Search Tree

- What is the maximum time to:
  - Insert a node?
  - Find a node?



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

In Class Questions

- http://goo.gl/TavW6D