Trees

• Snarf the code for today’s class
• and start looking at the code

Today

• Definition of a binary tree
• and lingo (e.g. “root”, “leaf”, “binary search tree”)
• Write recursive code to manipulate binary trees
• This will be easy and fun!
• 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
```java
IntTreeNode root = null;

public class IntTreeNode {
    public int value;
    public IntTreeNode left; // holds smaller tree nodes
    public IntTreeNode right; // holds larger tree nodes

    public IntTreeNode(int val) { value = val; }
}
```
**More terms**

**Depth:** distance of a node from the root

**Height:** maximum depth of the tree

---

**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
• Which is a binary search tree?

A

B

C

• Height Balanced

• 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
  
  • This matters hugely for efficiency
  
  • Which is NOT height balanced?

A B C D
Today

- Definition of a binary tree
  - and lingo (e.g. "root", "leaf", "binary search tree")
- Write recursive code to manipulate binary trees
  - This will be easy and fun!
- 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

Recursion and Trees

- They go together like PB&J!

- Pseudocode
  - Check the current node
    - if no
      - check the left subtree
      - check the right subtree
Your code

- example:

```java
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;
    }
}
```

Coding exercise

```java
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;
    }
}
```

- Code (as many as you can) countNodes, containsNode, and findMax
- If you get stuck on countNodes raise your hand
- If you finish early, modify your functions to work with a BinaryTree
- Submit your code via Ambient
Today

- Definition of a binary tree
- and lingo (e.g. “root”, “leaf”, “binary search tree”)
- Write recursive code to manipulate binary trees
- This will be easy and fun!
- 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

What is the height of a height-balanced tree?

A. $O(N)$
B. $O(N \ln(N))$
C. $O(\ln(N))$
D. $O(N^2)$

*We can prove this with induction*
In a Binary Search Tree

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

\[ O(\text{Tree height}) \]

Printing a Tree In Order

- Print **Left** subtree
- Print **Root**
- Print **Right** subtree

```java
public void printInOrder(IntTreeNode current){
    if(current == null)
        return;
    printInOrder(current.left);
    System.out.print(current.value + " ");
    printInOrder(current.right);
}
```
• Definition of a binary tree
  • and lingo (e.g. “root”, “leaf”, “binary search tree”)

• Write recursive code to manipulate binary trees
  • This will be easy and fun!

• 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

• Complete the worksheet off from the calendar page for today’s class!