

# CSSE 230 Day 9

More simple BinaryTree methods  
Tree Traversals

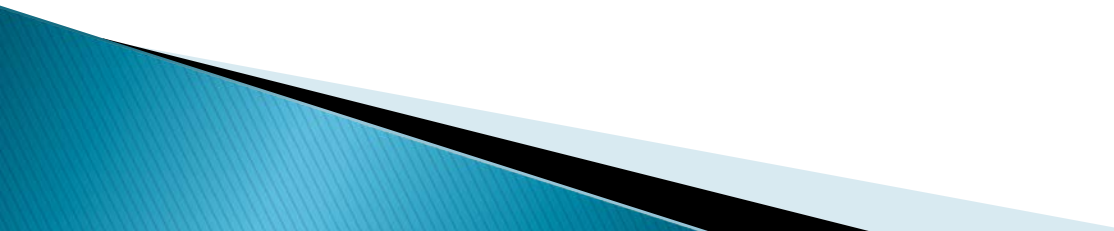
# Reminders / Announcements

- ▶ Hardy/Colorize programs due **Tuesday, Mar 26.**
- ▶ Exam 1 – Also **Tuesday, Mar 26 7:00 PM:**  
**O269 (Sec 1), O267 (Sec 2)**
  - Coverage:
    - Everything from reading and lectures, Sessions 1–9
    - Programs through Hardy + Colorize
    - Written assignments 1–3
  - Allowed resources:
    - Written part: One side of one 8.5 x 11 sheet of paper
    - Programming part:
      - Textbook
      - Eclipse (including programs in your repositories)
      - Course web pages and materials on ANGEL
      - Java API documentation
    - A previous 230 Exam 1 is available on ANGEL

**No devices  
with  
headphones  
or earbuds  
are allowed**

# Questions?

# Agenda

- ▶ Another induction example
  - ▶ Binary Tree Traversals
  - ▶ Implementing Binary Trees (continued)
  - ▶ Hardy/Colorize work time
- 

# Another induction proof example

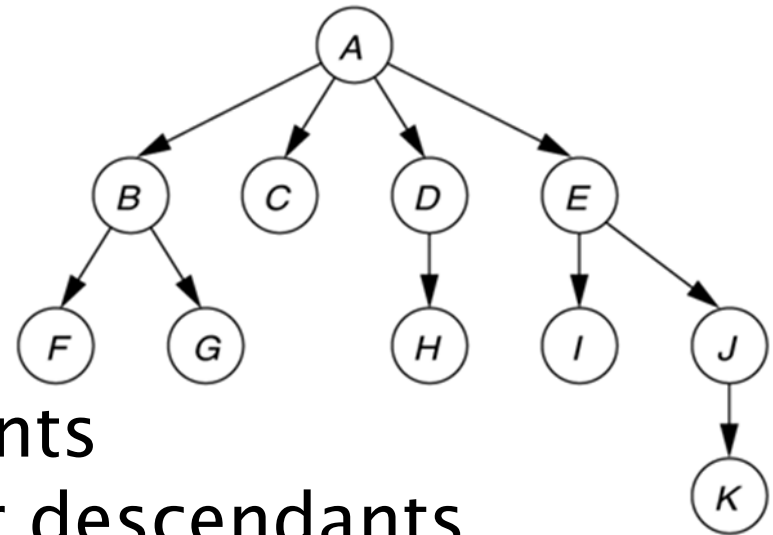
Show by induction that  $2n + 1 < n^2$  for all integers  $n \geq 3$

There are other ways that we could show this (using calculus, for example)

But for now the goal is to have another example that can illustrate how to do proofs by induction

# Recap: Tree Terminology

- ▶ Parent
- ▶ Child
- ▶ Grandparent
- ▶ Sibling
- ▶ Ancestors and descendants
- ▶ Proper ancestors, proper descendants
- ▶ Subtree
- ▶ Leaf, interior node
- ▶ Depth and height of a node
- ▶ Height of a tree



# Binary tree traversals

- ▶ PreOrder (top-down, depth-first)
  - root, left, right
- ▶ PostOrder (bottom-up)
  - left, right, root
- ▶ InOrder (left-to-right, if tree is spread out)
  - Left, root, right
- ▶ LevelOrder (breadth-first)
  - Level-by-level, left-to-right within each level

If the tree has  $N$  nodes, what's the (worst-case) big-Oh run-time of each traversal? big-Oh space used?

```
// Print tree rooted at current node using preorder
public void printPreOrder( ) {
    System.out.println( element );           // Node
    if( left != null )
        left.printPreOrder( );               // Left
    if( right != null )
        right.printPreOrder( );              // Right
}

// Print tree rooted at current node using postorder
public void printPostOrder( ) {
    if( left != null )
        left.printPostOrder( );              // Left
    if( right != null )
        right.printPostOrder( );             // Right
    System.out.println( element );           // Node
}

// Print tree rooted at current node using inorder
public void printInOrder( ) {
    if( left != null )
        left.printInOrder( );                // Left
    System.out.println( element );           // Node
    if( right != null )
        right.printInOrder( );               // Right
}
```



# Growing Trees

- » Let's continue implementing a *BinaryTree*<*T*> class including methods *size()*, *height()*, *duplicate()*, and *contains(T)*.

# Binary Tree Iterators

- » What if we want to iterate over the elements in the nodes of the tree one-at-a-time instead of just printing all of them?

# Hardy/Colorize Work time

»» The assistants and I will be  
available for help