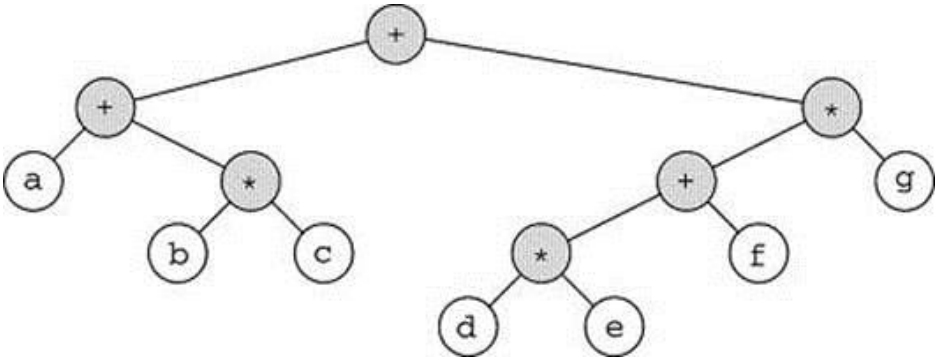
CIS 22C-03Y TTh 1:30-3:20 PM
Class Notes for 05-09-2017

MIDTERM will be on THURSDAY May 18 (whole class time)! It's closed book, closed computer, closed device, you may bring 3 sheets of your own notes (front & back for a total of 6 pages), may be typed or handwritten, pen + pencil w/ eraser!

Review for the midterm is on Catalyst under Week 5! We will review some this Today, some next Tuesday!

==========
ANSWERS TO LAB EXERCISE 7.1:

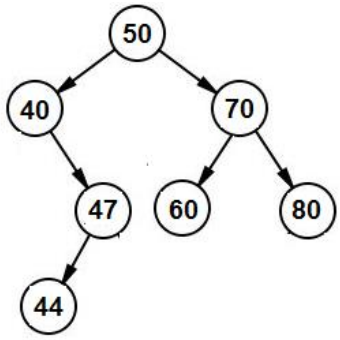


Answer the following questions about the Binary Tree shown above.

1. What is the height of the complete tree 4
2. What is the height of node b? 0
3. What is the height of the node on the left with *? 1
4. Which are the leaf nodes? a, b, c, d, e, f, g
5. What is the depth of node f? 3
6. If the preorder function was called for the above tree (starting at the top root), and the visit function just displays the node's value (item), what would be displayed?
   ++a*bc*+*defg
   logically: +(+a(*bc))(*(+(*de)f)g)

==========
ANSWERS TO LAB EXERCISE 7.2:
Tree 1:

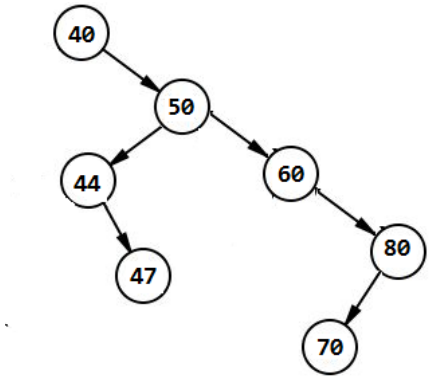
50 40 47 70 80 60 44



height = 3
leaf nodes: 44, 60, 80

Tree 2:

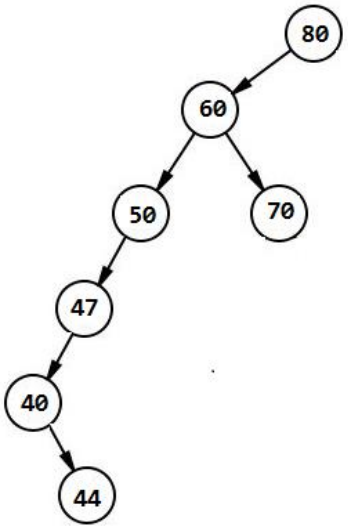
40 50 44 47 60 80 70



height = 4
leaf nodes: 47, 70

Tree 3:

80 60 70 50 47 40 44



height = 5
leaf nodes: 44, 70

QUESTIONS:

- What are the heights of each root of each tree?
- In each tree, which are the leaf nodes?
- If the order of tree insertion was in sorted order (smallest to largest), how would you describe the tree?

  one line of nodes going down to the right

==========

Lesson 7: BST code

RECALL BinaryNode (NOT a nested inner class this time) and BinaryTree (page 2)

See more (but not completed) BST.java class in the Binary Tree Code folder under Week 5 inCatalyst:

```
public class BST<E extends Comparable<E>>
    extends BinaryTree<E>
```

```
{

    private boolean foundNode; // helper variable

    /** Create a default binary tree */
    public BST() {
    }


    /** Create a binary tree from an array of objects */
    public BST(E[] objects)
    {
    for (int i = 0; i < objects.length; i++)
        insert(objects[i]);
    }

    @Override /** Returns true if the element is in the tree */
    public boolean contains(E e)
    {
    BinaryNode<E> current = root; // Start from the root

    while (current != null)
    {
        if (e.compareTo(current.getData()) < 0)
        {
            current = current.getLeftChild();
        }
        else if (e.compareTo(current.getData()) > 0)
        {
                current = current.getRightChild();
        }
        else // element matches current.getData()
            return true; // Element is found
    } // end while

    return false;
    }

    @Override
    /**
     * Returns the data of the Node that equals the parameter, null if not found.
     * */
     public E getEntry(E e)
     {
        BinaryNode<E> foundNode = _findNode(root, e); // YOU WRITE FOR LAB EX. 7.3
        if( foundNode != null)
            return foundNode.getData();

        return null;
    }


    @Override
    /** Insert element o into the binary tree
     * Return true if the element is inserted successfully */
    public boolean insert(E e)
    {
        if (root == null)
            root = new BinaryNode<E>(e); // Create a new root
```

```
        else
        {

            // FILL IN HERE FOR HW#4**********************

            size++;
        }
        return true; // Element inserted successfully
    }


    @Override
    /** Delete an element from the binary tree.
     * Return true if the element is deleted successfully
     * Return false if the element is not in the tree */
    public boolean delete(E e)
    {
        foundNode = false; // initialize boolean instance variable
        root = _delete(root, e); //call private method to do actual deletion

        if( foundNode )
        {
            size--;// Element deleted successfully
        }
        return foundNode;
    }

    // Private recursive method that returns an updated "root" node from where current
node is
    private BinaryNode<E> _delete( BinaryNode<E> node, E e )
    {
        if( node==null )
        {
            return null;
        }
        if ( e.compareTo(node.getData()) < 0 )
             node.setLeftChild( _delete(node.getLeftChild(), e) );//recursive call
        else
            if( e.compareTo(node.getData()) > 0 )
                node.setRightChild( _delete(node.getRightChild(), e) );//recursive call
            else
            {
                foundNode = true;
                node = _deleteNode( node );
            }
        return node;
    } // end _delete

    // Private method that either "moves up" the left or right child, OR
    //    replaces the data in the nodeToDelete with the data in the rightmost child of
    //    the nodeToDelete's left child, then "removes" that node
    private BinaryNode<E> _deleteNode( BinaryNode<E> nodeToDelete )
    {
        if( nodeToDelete.isLeaf() ) // node to delete has no children
        {
           return null;
        }
        if( !nodeToDelete.hasLeftChild() ) // node to delete has no LEFT child
        {
```

```
            return nodeToDelete.getRightChild();
        }
        if( !nodeToDelete.hasRightChild() ) // node to delete has no RIGHT child
        {
            return nodeToDelete.getLeftChild();
        }
        // must have left and right children
        // Locate the rightmost node in the left subtree of
        // the node to delete and also its parent
        BinaryNode<E> parentOfRightMost = nodeToDelete;
        BinaryNode<E> rightMost = nodeToDelete.getLeftChild();

        while (rightMost.getRightChild() != null) {
            parentOfRightMost = rightMost;
            rightMost = rightMost.getRightChild(); // Keep going to the right
        }

        // Replace the element in nodeToDelete by the element in rightMost
        nodeToDelete.setData( rightMost.getData() ); // don't really delete the node, just
change the data

        // Eliminate rightmost node
        if (parentOfRightMost.getRightChild() == rightMost)
            parentOfRightMost.setRightChild(
                                rightMost.getLeftChild() );
        else
            // Special case:
                //     nodeToDelete's leftChild has no rightChild
            parentOfRightMost.setLeftChild(
                               rightMost.getLeftChild() );

        return nodeToDelete;
    } // end private _deleteNode


} // end class BST
```

==========

SEE PROG. HW#3 Questions (due this Sunday!)

DON'T CHANGE ANY EXISTING CODE (UNLESS SPECIFIED) WITHOUT THE INSTRUCTOR'S PERMISSION OR YOU MAY GET POINTS OFF!!!!!

(HW#4 will be posted in Catalyst early next week)

==========

REVIEW FOR THE MIDTERM (PART 1):

Describe what a stack is and its operations.
AN ANSWER: A stack is a Last-In-First-Out data structure.
Operations:
push(item) that adds the item to the top of the list
peek() which returns the top of the stack
pop() which removes the top of the stack

Describe what a queue is and its operations.
AN ANSWER: A queue is a First-In-First-Out data structure.
Operations:
enqueue(item): adds an item to the end of the queue

peekFront(): returns the items in the front of the queue
dequeue(): removes and item from the front of the queue

Describe the difference between array implementations and linked list implementations of stacks (the stack methods in StackInterface of the textbook)

AN ANSWER:
ArrayStack has limited size (UNLESS you dynamically allocate the array), but uses an array, so is more efficient
LinkedStack may add as many until the program memory runs out, but is less efficient

(ANSWERS TO REVIEW QUESTIONS NOT SHOWN HERE WILL BE GIVEN ON TUESDAY)

Determine the worst case Big-Oh of the following method:

```
public static <E> void insertionSort(E theArray[], int n)
{
   for (int unsorted = 1; unsorted < n; unsorted++)
   {
      E nextItem = theArray[unsorted];
      int loc = unsorted;
      while ((loc > 0) && (theArray[loc - 1] > nextItem))
      {
         theArray[loc] = theArray[loc - 1];
         loc--;
      }  // end while

      theArray[loc] = nextItem; // Insert nextItem into sorted region
   }  // end for
}  // end insertionSort
```

ANSWER: O(n^2)

Answer the questions below about the following method:

```
public static int gcd(int a, int b)
{
   if (a % b == 0)  // line 1
      return b;     // line 2
   else                // line3
      return gcd(b, a % b); // line4
}  // end gcd
```

- what is the base case of the method? ANSWER: line 1
- what is the recursive statement? ANSWER: line 4
- could you write the above iteratively (using a loop and no recursion)? ANSWER: YES
- If you called it: gcd(40, 16), what would the 2nd call have for a & b?? ANSWER: 16 and 8

Using the following method from the SortedList (concrete LinkedList class in which the items are inserted in order), answer the questions below.

```
    private Node getPreviousNode(T anEntry)//DIFFERENT FROM HW#3!
    {
        Node prev, curr;

        prev = null;
        curr = firstNode;
        while( curr != null && curr.getData().compareTo(anEntry) < 0 )
        {
```

```
            prev = curr;
            curr = curr.getNextNode();
        }
        return prev;
    }
```

- if the list is empty, what will happen? ANSWER: returns null
- if the "anEntry" item equals the firstNode's item, what is returned? ANSWER: returns null
  if the "anEntry" item equals the 2nd node's item, what is returned? ANSWER: returns firstNode
- if the "anEntry" item > than all of the list's items, describe what is returned
  ANSWER: reference to the last node

What is a binary tree?

ANSWER: a tree where each node has at most 2 children.

Know the terminology: height of a subtree, depth of a tree (in the textbook, it's like the level-1), leaf nodes

REST of the REVIEW for the Midterm will be discussed on Tuesday! You should try to answer the rest on your own first!

==========

IN-CLASS EXERCISE (CUT-OFF TIME IS FRIDAY 11:55 PM):

For the LinkedQueueDummy<T> class shown in the Review for the Midterm, fill in the blanks (or indicate which lines should be deleted) as specified on the review.

Submit ONLY in the TEXT BOX (don't upload any files), but you should SAVE AS a TEXT file, then copy and paste the text into the text box (don't copy and paste parts of the RTF file)!

When done, work on Lab Exercise 7.3 OR HW#3!!!

==========

TO DO THE REST of THIS WEEK AND FOR TUESDAY:

- Turn in IN-CLASS Review EXERCISE (due Saturday NIGHT)
- Work on the exercise due BEFORE CLASS ON TUESDAY (7.3)
- Work on HW#3 (due TUESDAY NIGHT)!!!
- start studying for the midterm (TRY to do the REST of the review problems on your own!), write some your 3 sheets of notes!

==========