Opening Discussion

- Midterm EC.
- Uses of recursive sorts.
What is a Tree?

- You are all familiar with what normal trees look like. In CS we use the term somewhat differently, and more formally.
- To describe trees we need some basic terminology
  - Node - an element of a tree. One node is designated as the “root”
  - Edge - a directed connection from one node to another.
Tree Criteria

- Every node, C, has exactly one incoming edge from another node, P. P is said to be the parent of child node C. Root has 0.
- There is a unique path from the root to any node. The number of edges on that path is called the path length. It is also called the depth of the node.
- A node with no children is called a leaf. The path length from a node to the deepest leaf in the height of that node.
More Terms

- Following the parent-child analogy, children of the same node are called siblings. We also call any node on a path below a given node a descendant and any above an ancestor.

- You might also hear the size of a node referred to as the number of descendants of a node, including itself.

- We can also define a tree as either empty, or a root with zero or more subtrees where the root connects to the roots of those subtrees.
In a general tree, each node can have zero or more children. That is a lot of flexibility. We want a class to represent nodes. To get this flexibility we can use a linked list. Each node has pointers to a first child and the next sibling.

It might be just as easy to have the child member be an Buffer that we put Nodes in. File systems are a good example of this.
Traversals

- As with any data structure one of the things you want to be able to do is to traverse through all the elements.
- Think for a while about how you would do this? There is even a question about the order you traverse them in. Do you want to process a node before you process its children or after? If before we call it a preorder traversal. If after it is a postorder traversal.
Traversals and Recursion

- The simplest way to do a traversal is through recursion. If you want to do it with a loop you have to implement a data structure to store some nodes or have the tree specially set up.
- The traverse function takes a node and calls itself once with each child node. It also does whatever the visit operation is.
- Preorder does a visit before going to children and postorder visits after going to children.
- Breadth first uses a queue, not recursion.
For our first example of a tree, I want to make our formula parser parse to a tree.

If we introduce variables we might evaluate the same formula many times with different values. It is inefficient to do the same string processing over and over.

It is more efficient to parse the string once and build a tree that represents the formula then do the evaluation on that tree.

Let's code this.
Binary Trees

- Sometimes we want to limit how many children a node has. One of the most commonly used trees in programming is the binary tree where no node has more than 2 children.
- The children are often called left and right.
In-order Traversal

• For a binary tree there is an extra type of traversal called an in-order traversal where the node is visited between the recursion down left and right.

• Equations are great examples of trees. We typically write them out in the in-order. We could just as well write them out in post-order or pre-order.
One of the best uses of binary trees is the binary search tree. They make a more efficient implementation of the map ADT.

In this type of tree, we store a key and data in every node and below any node we put lesser key values to the left and greater key values to the right.

We find elements by going down the tree always going left or right. This gives us behavior like a binary search, but the tree is more flexible because adds and removes are quite efficient as well.
Adding and Removing

• The code for both adding and removing from a binary tree begins like a search that keeps track of previous (much like a singly linked list).
  ▪ The add always goes to a leaf and adds the new element to the proper side.
  ▪ The remove replaces the node we are removing with either the greatest node on the left or the smallest node on the right.

• Recursion can be used to make some nice solutions.
I want us to code a BST based mutable map together.
What can go wrong with the type of binary tree that we wrote today to make it perform poorly?