Binary Trees

Types of binary trees:

- A **rooted** binary tree has a root node, and every node has at most two children.
- A **full** binary tree is a rooted BT in which all interior nodes have either 0 or 2 children.
- A **perfect** binary tree is a tree structure in which all interior nodes have two children *and* all leaves have the same depth or level.
- A **balanced** binary tree has the minimum possible maximum depth for the leaf nodes.
- A **degenerate** tree is where each parent node has only one associated child node, effectively reducing the tree to a linked list.

An example in Java:

```
class Node {
    int data;
    Node left;
    Node right;
}
```

Or in C:

```
typedef struct Node
{
    int data;
    struct Node left;
    struct Node right;
} Node;
```

Or in Haskell:

```
data Tree a = Null | Node a (Tree a) (Tree a)
```

Finding a particular node in a tree

In Java:

```
Node search(int key) {
    Node c = root;
    while (true) {
        if (key < c.data)
            c = c.left;
        else if (key > c.data)
            c = c.right;
        else return c;
    }
}
```

And a recursive implementation in C++:

```
Node *Tree::search(int key, Node *c)
{
    if (c != NULL) {
        if (key < c->data)
            return search(key, c->l);
        else if (key > c->data)
            return search(key, c->r);
        else
            return c;
    }
}
```

Inserting a value into a tree

In Haskell:

```
insertT :: (Ord a) => a -> Tree a -> Tree a
insertT x Null = Node x Null Null
insertT x (Node a l r)
    | x < a     = Node a (insertT x l) r
    | x > a     = Node a l (insertT x r)
    | otherwise = Node a l r
```

In C++:

```
void Tree::insert(int key, Node *c)
{
    if (key < c->data) {
        if (c->l != NULL)
            insert(key, c->l);
        else {
            c->l = new Node;
            c->l->data = key;
            c->l->l = NULL;
            c->l->r = NULL;
        }
    }
    else if (key > c->data) {
        if (c->r != NULL) {
            insert(key, c->r);
        }
        else {
            c->r = new Node;
            c->r->data = key;
            c->r->l = NULL;
            c->r->r = NULL;
        }
    }
}
```

Convert BT to ordered list

Depth first in-order traversal in Java:

```
void inOrder(Node root) {
    if (root != null) {
        inOrder(root.left);
        System.out.println(root.data);
        inOrder(root.right);
    }
}
```

Much the same in any language.