Java 2 Binary Trees

EVERYTHING HERE ONLY APPLIES TO BINARY TREES!!!

- **How to implement:**
  - Can do this with array based or node-and-link based

Node :

```
class BTN<T>
{
	T element;
	BTN left;
	BTN right;
}
```

- This implementation matches our conceptual picture of what a tree looks like.
- For now we stick with this for the sake of memory

Array:

- Store the root at index 0
- For a node stored at index i
  - Left child at 2i + 1
  - Right child at 2i + 2
  - Parent at (i - 1)/2
- Convenient but bad with memory unless its full or complete.

Recursive Definition

- A binary tree is a tree that is either empty or is is a single node that has two binary tree as its left and right subtrees.

Computing Height

- Think recursively…
- Base Case
  - Empty, so Height is zero (Some define the height of an empty tree as -1. This makes no intuitive sense; our way is better.
- Recursive Case
  - The node (N_ contributes 1 to the height
  - Calculate the height of the left Subtree
  - Calculate the height of the right subtree
  - Add 1 to each and see which one is bigger

```
public int height(BTN n) {
	if (n == null) {
		return 0;
	}
	else {
		int lh = height(n.getLeft());
		int rh = height(n.getRight());
		return 1 + max(lh, rh);
	}
}
```

Search a binary tree

- Search the tree for a particular element. Return true if the value is found false otherwise.

```
public boolean search(BTN n, T target) {
```

```
        if (n == null) {
                return false;
        }

        else {
                if (n.getElement().equals(target))
                        return true;
                else {
                        //Some recursive call
                }
        }
}
```

Binary Tree Traversal

Base Case - Empty (nothing to traverse)
Recursive Case

1. Visit the Node(N)
2. Traverse the left subtree
3. Traverse the right subtree //This ordering is called Preordering

```
public void preorder(BTN n) {
        if (n != null) {
                vist(n);
                preorder(n.getLeft());
                preorder(n.getRight());
        }
}
```

Three Major Order of Tree

- **Preorder**
  - **NLR**
- **PostOrder**
  - **LRN**
- **Inorder**
  - **LNR**
- Always handle the Left subtree before the right subtree

Level Order Traversal

- Preorder, inorder, and postorder are all **depth-firs**t strategies. A **breadth-first** strategy would visit the node level by level.