Binary Trees

Binary Trees is a data storage technique that utilizes a tree-like storage. It starts from a **root node**, the very top node. From there, a binary tree will branch out in **up to two** child nodes, like so:

```
      100
      / \
    50   150
   / \
 25   75
```

In the following tree, 100 is the root node. 50 is 100's child node, and 25 is it's grandchild node. 150 is a **leaf node** because it has no child nodes.

Terminology

- **Height** - The number of edges between the selected node and it's deepest leaf. The height of the root node is the **tree height**
- **SubTree** - A selected node and all it's descendants form a subtree

Advantages

- Efficient search, sort, insert, delete
- Form reflects relationships in data
- Easy to manipulate

Implementations

- Binary Search Trees (BST)
- Heaps

Biinary Search Tree

A **Binary Search Tree** is like a B.Tree except:

- No duplicate notes
- Moving left from a node will have lower numbers. Moving right will have higher numbers.

This is specialized for searching elements. For example, we can do a recursive search to look for an element:

```
Pseudocode
===

Start at root node
Compare node to itemSought
if node == itemSought
  return
if node > itemsought
  search left subtree
else
  earch right subtree
```

Balance

The definition of **balance** can vary. One defenition of a balanced tree is one which has a right height and right height that are no more than 1 apart.

Counting Max Nodes and Comparisons

The **max nodes** a binary tree allows is defined as 2^x - 1, where x is the root height. The max comparisons needed to find a leaf (or not find anything) is defined as x, where x is also the root height.

Efficiency

Searching a balanced BST with n elements takes about log(n) operations. In the worst case with an unbalanced BST, the worst case is n operations. Searching an unordered array with n elements takes about n operations.

Traversal

Traversals is when we visit every node in the tree. There are different kinds of traversals. While reading, use this diagram as visual assistance:

- **Height**: moving top down, visiting all nodes of a level first (Y-S-R-M-W-Q)
- Depth
    - In-order: Starting from smallest node (bottom left), check for left subtree, root node, right subtree (M-S-W-Y-R-Q).