Chapter 7 – Trees

Java Binary Tree Search:

\*\*BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved. \*\*

* Each node only has a maximum of 2 children nodes (can have 0, 1, or 2 children)
* Parent / Root Node: has a middle value
* The lesser value than the parent 🡪 on left (left child)
* The greater value than the parent 🡪 on right (right child)
* Quicker operation than Ordered Arrays & Linked Lists 🡪 Has the benefits of both an ordered array and a linked-list:
* search is as quick as in a sorted array
* insertion or deletion operation are as fast as in linked list
* On average, trees are more efficient if you have to perform many different operations quickly

Each Node has:

* Key
* Name (or other associated value)
* Left child
* Right child
* left & right children can be null

Each Tree has:

* Root (Parent Node)
* Add Node functionality
* Delete Node functionality
* Search / Travers Tree functionality

General BST Algorithm Info:

**Base Case** 🡪 Tree is empty – meaning that this node is the first to be added to the tree

* Simply add the Node

**Recursive Case** 🡪 Tree not empty

\* Must perform BS, tree traversal to find the proper location for the node \*

* Check if new node should be added to the right or left side of the parent
* If the key value is less than the node of focus, traverse left child – else – traverse right child

\*Until we get to a node that we can’t traverse 🡪 then insert node there

Delete Operation:

* Is this the root?

3 Ways to Traverse:

* (1) In order traversal:
* Aims for the smallest value first 🡪 start at first left child (bottom-left most node)
* When null is reached then move up in value
* All nodes are visited in ascending order (by key values)

Recursion is used to go from node to node