**Binary Tree**

*Pseudocode for BST:*

insert:

* Create a new Node
* Starting at the root:
  + Check if there is a root, if not – the root is the newly created node
  + If there is a root, check if the value of the new node is greater than or less than the value of the root
  + If it is greater
    - Check to see if there is a node to the right
      * If there is, move to that node and repeat these steps
      * If there is not, add that node as the right property
  + If it is less
    - Check to see if there is a node to the left
      * If there is, move to that node and repeat these steps
      * If there is not, add that node as the left property

find:

* Starting at the root:
  + Check if there is a root, if not – we’re done searching
  + If there was a root, check if the value of the new node is the value we are looking for. If we found, we’re done
  + If not, check if the value of the new node is greater than or less than the value of the root
  + If it is greater
    - Check to see if there is a node to the right
      * If there is, move to that node and repeat these steps
      * If there is not, we’re done searching
  + If it is less
    - Check to see if there is a node to the left
      * If there is, move to that node and repeat these steps
      * If there is not, we’re done searching

*Full Code:*

class Node{

constructor(val) {

this.val = val;

this.left = null;

this.right = null;

}

}

class BST {

constructor() {

this.root = null;

}

insert(val) {

var newNode = new Node(val);

if (this.root == null) {

this.root = newNode;

return this;

} else {

var current = this.root;

while(true) {

if (val == current.val) {

return undefined;

}

if (val < current.val) {

if (current.left == null) {

current.left = newNode;

return this;

} else {

current = current.left;

}

} else if (val > current.val) {

if (current.right == null) {

current.right = newNode;

return this;

} else {

current = current.right;

}

}

}

}

}

find(val) {

if (this.root === null) return false;

var current = this.root;

var found = false;

while(current && !found) {

if (val < current.val) {

current = current.left;

} else if (val > current.val) {

current = current.right;

} else {

found = true;

}

}

if (!found) return undefined;

return current;

}

contain(val) {

if (this.root === null) return false;

var current = this.root;

var found = false;

while(current && !found) {

if (val < current.val) {

current = current.left;

} else if (val > current.val) {

current = current.right;

} else {

return true;

}

}

return false;

}

}

*Big O:*

Insertion – O(logN)

Searching – O(logN)