**Breadth First Search**

*Definition:*

* Visit every node on the same level before looking at the child (10,6,15,3,8,20)

🡪 10

🡪 6 🡪 15

🡪 3 8 🡪 20

*Pseudocode for Breadth First Search:*

* Create a queue (this can be an array) and a variable to store the values of nodes visited
* Place the root node in the queue
* Loop as long as there is anything in the queue
  + Dequeue a node from the queue and push the value of the node into the variable that stores the nodes
  + if there is a left property on the node dequeued – add it to the queue
  + If there is a right property on the node dequeued – add it to the queue
* Return the variable that stores the values

*Solution for Breadth First Search:*

function BFS() {

var node = this.root;

var data = [];

var queue = [];

queue.push(node);

while(queue.length) {

node = queue.shift();

data.push(node.val);

if (node.left) {

queue.push(node.left);

}

if (node.right) {

queue.push(node.right);

}

}

return data;

}

**Depth First Search**

*Definition:*

* Visit vertically down to the end of the tree before visiting siblings nodes

*Definition for Depth First Search Pre-Order:*

* Visit the node then visit the node’s entire left side then traverse the right

*Pseudocode for Depth First Search Pre-Order:*

* Create a variable to store the values of nodes visited
* Store the root of BST in a variable called current
* Write a helper function which accepts a node
  + Push the value of the node to the variable that stores the values
  + If the node has a left property, call the helper function with the left property on the node
  + If the node has a right property, call the helper function with the right property on the node
* Invoke the function with the current variable
* Return the array of values

*Solution for Depth First Search Pre-Order:*

function DFS\_Pre() {

var data = [];

var current = this.root;

function traverse(node) {

data.push(node.val);

if (node.left) {

traverse(node.left);

}

if (node.right) {

traverse(node.right);

}

}

traverse(current);

return data;

}

*Definition for Depth First Search Post-Order:*

* Traverse the tree, the left and right, then we visit the node

*Pseudocode for Depth First Search Post-Order:*

* Create a variable to store the values of nodes visited
* Store the root of BST in a variable called current
* Write a helper function which accepts a node
  + If the node has a left property, call the helper function with the left property on the node
  + If the node has a right property, call the helper function with the right property on the node
  + Push the value of the node to the variable that stores the values
* Invoke the function with the current variable
* Return the array of values

*Solution for Depth First Search Post-Order:*

function DFS\_Post() {

var data = [];

var current = this.root;

function traverse(node) {

if (node.left) {

traverse(node.left);

}

if (node.right) {

traverse(node.right);

}

data.push(node.val);

}

traverse(current);

return data;

}

*Definition for Depth First Search In-Order:*

* Traverse the entire left side, then visit the node then traverse entire right

*Pseudocode for Depth First Search In-Order:*

* Create a variable to store the values of nodes visited
* Store the root of BST in a variable called current
* Write a helper function which accepts a node
  + If the node has a left property, call the helper function with the left property on the node
  + Push the value of the node to the variable that stores the values
  + If the node has a right property, call the helper function with the right property on the node
* Invoke the function with the current variable
* Return the array of values

*Solution for Depth First Search In-Order:*

function DFS\_In() {

var data = [];

var current = this.root;

function traverse(node) {

if (node.left) {

traverse(node.left);

}

data.push(node.val);

if (node.right) {

traverse(node.right);

}

}

traverse(current);

return data;

}