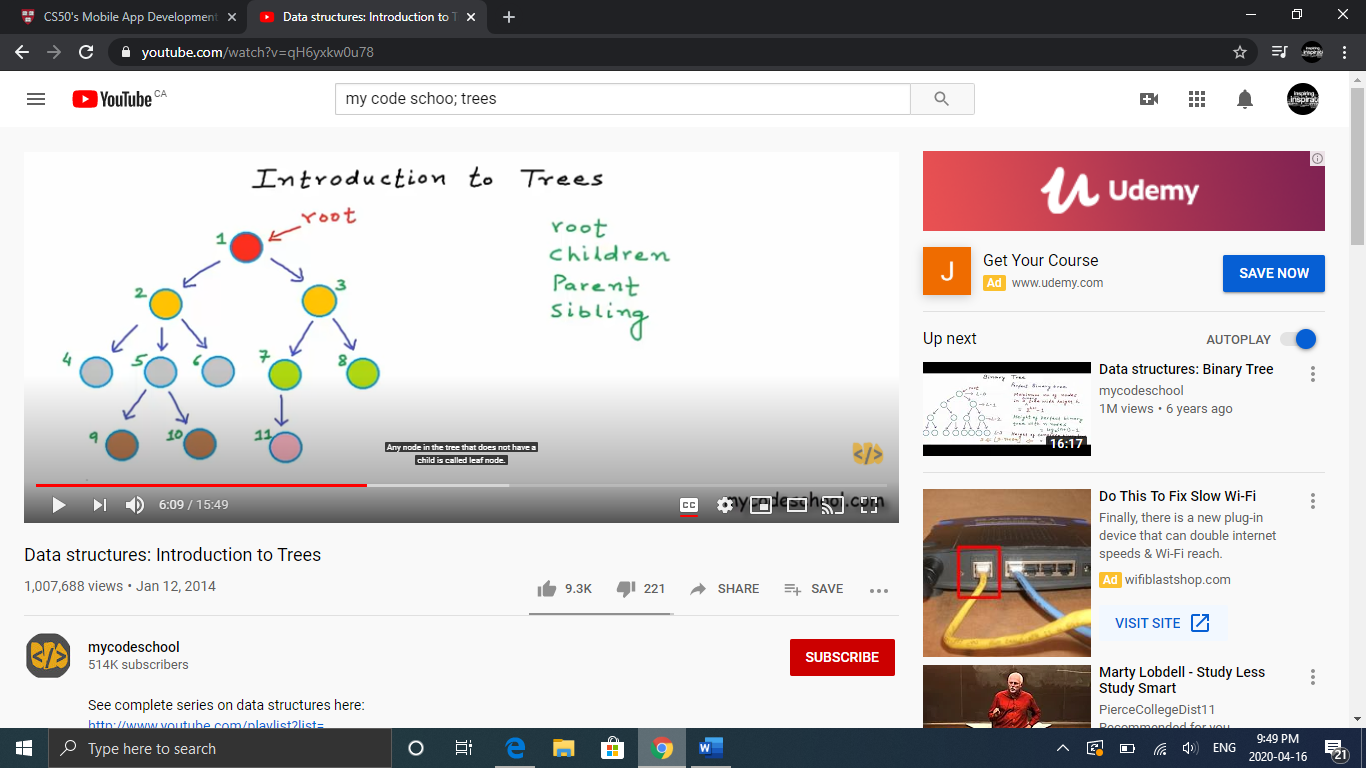
Trees/DFS

* Trees are non-linear data structures
* Trees have a root node, and may have edges to connect to children nodes
* The root is the top most node
* A **child** is a node one edge below another node
* If a node has children nodes it is a **parent** node
* If two nodes have the same **parent** they are **siblings**
* If a node has no children it is a **leaf** node



* If n represents our number of nodes, there are n-1 edges because all nodes are connected except the root (it has no parent)
* The depth of a node x= number of edges from root to x
* The height of node x= number of edges in the longest path from x to leaf
* The height of tree: number of edges in longest path from root to leaf
* The height of the root is 0, and the height of an empty tree is -1
* Many Tree problems require the use of recursion

# Binary Tree

* A Binary Tree is a tree that can have at most 2 children (left and right)
* n-1 is the maximum height of a binary tree where every node has 1 child
* Math.trunc(log2n) is the minimum height of a binary tree of n nodes

# Binary Search Tree

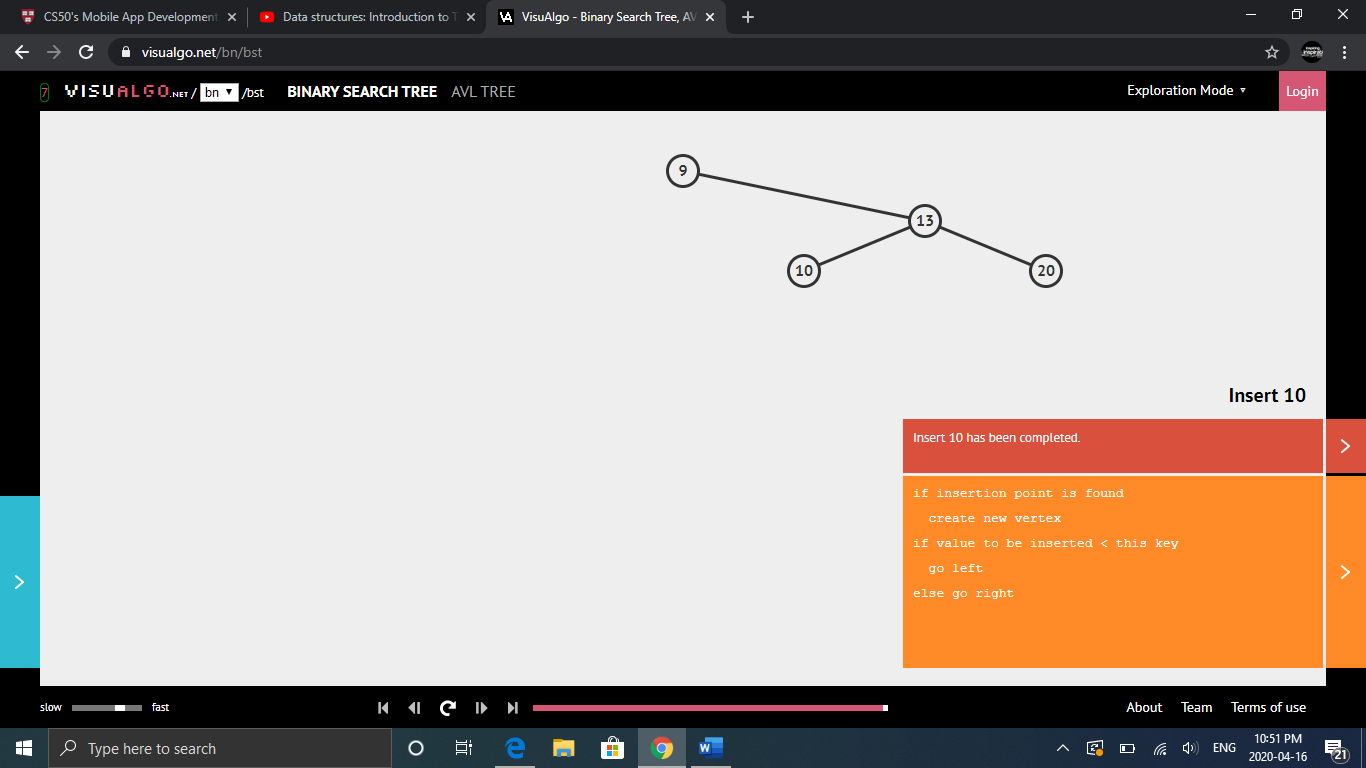
* A binary tree where the value of all nodes in the right subtree of a node is greater than the node’s value, and the value of all nodes in the left subtree of the node is less than the node’s value

**Time Complexity**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Average Case** | **Worst Case** |
| Search | O(log n) | O(n) |
| Insert | O(log n) | O(n) |
| Delete | O(log n) | O(n) |

Array Implementation of Binary Search Tree

* 2i+1, where (i is an index in the array) is the left child of the node at that index
* 2i+2 where (i is an index in the array) is the right child of the node at that index
* int[] BST={9, null, 13, null, null, 10, 20} would look like:



# Depth First Search

* aka DFS is a way to traverse a tree from top down
* There are 3 ways to conduct a depth first search
* Pre-order: Root – Left – Right
* In-order: Left- Root – Right
* Post-order: Left – Right – Root
* Their algorithms are as follows:

preorder(root){

if(root is null) return

print root

preorder(root.left)

preorder(root.right)

}

inorder(root){

if root is null return

inorder(root.left)

print root

Inorder(root.right)

}

postorder(root){

if root is null return

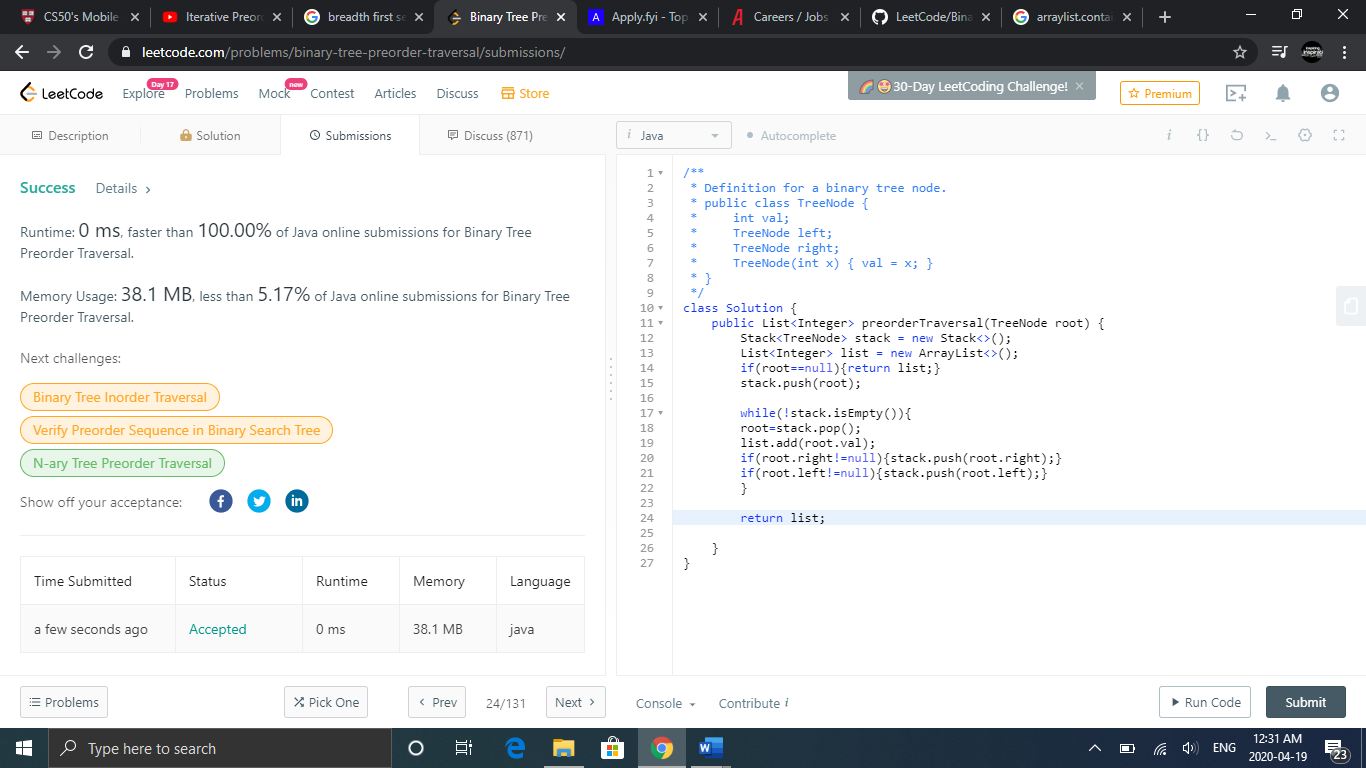
postorder(root.left);

postorder(root.right);

print root

}

## **PreOrder Iteratively**



# Breadth First Search

* A way to traverse a tree by each level
* Can be implemented using a queue
* Implementation for creating a list for each level is as follows:

BFS(root){

Queue<TreeNode> q = new LinkedList<>();

if(root==null){return list;}

q.add(root);

q.add(null);

while(!q.isEmpty()){

TreeNode current = q.remove();

if(current==null){ //if null we have finished traversing a level of tree

q.add(null);

if(q.getFirst()==null){break;} //if null again we have traversed the entire tree

}

else{

List.add(current);

if(current.left!=null){q.add(current.left);}

if(current.right!=null){q.add(current.right);}

}

}

return list;