CSCI 321 Computer Science III Fall 2018

Lecture 6 Activity 1

**Binary Search Tree**

**Part A**: The code attached in this document is a sample code to demonstrate insert operation in binary search tree. Please fill in the missing part for the insert method to make the program work. The expected output should be as follows.

20

30

40

50

60

70

80

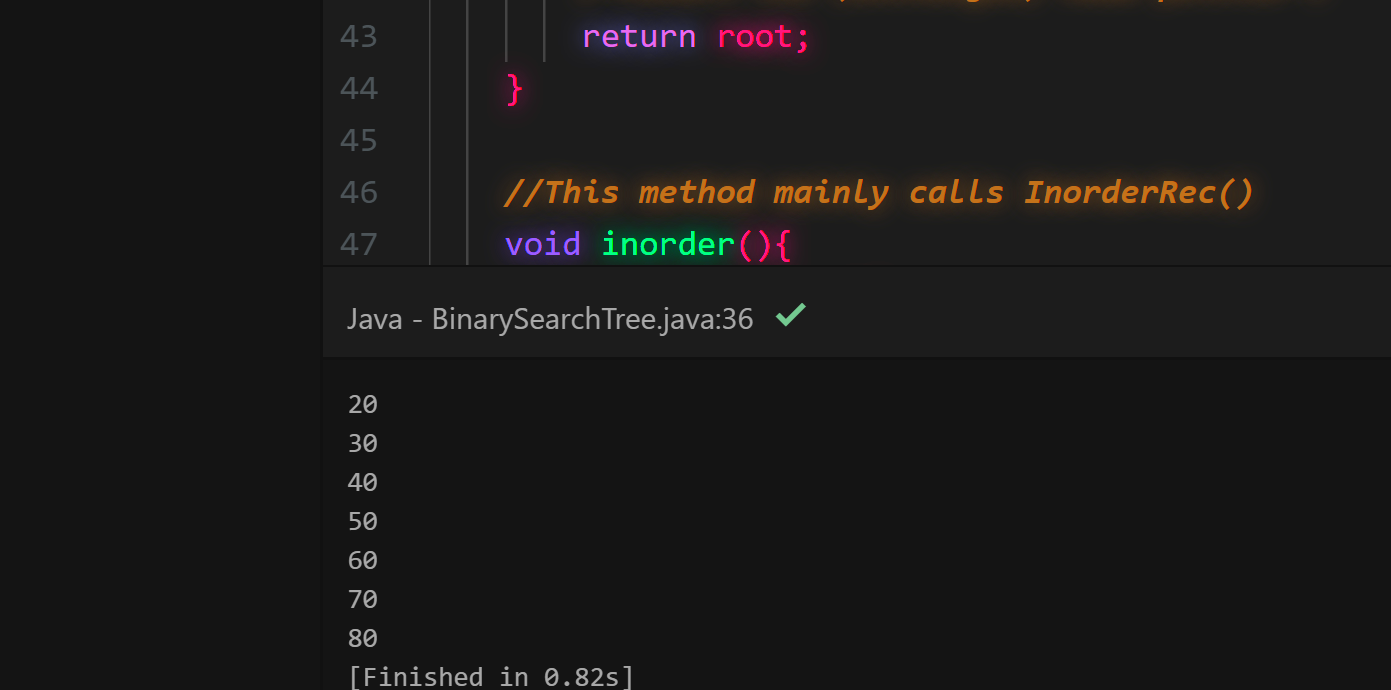
Missing code:

//Otherwise, recur down the tree

if (key < root.key)

root.left = insertRec(root.left, key);

else if (key > root.key)

root.right=insertRec(root.right,key);

**Part B**: Find Lowest Common Ancestor (LCA) of a Binary Search Tree. According to WikiPedia definition , The lowest common ancestor is defined between two nodes v and w as the lowest node in T that has both v and w as descendants (where we allow a node to be a descendant of itself). Write a function (Find Lowest Common Ancestor (LCA) of a Binary Search Tree) within the BinarySearchTree class in Part A and output test results.

Sample input: v: node 20, w: node 40. Expected output: node 30

Sample input: v: node 60, w: node 80. Expected output: node 70

Sample input: v: node 30, w: node 70. Expected output: node 50

// Java program to demonstrate insert operation in binary search tree

class BinarySearchTree {

/\* Class containing left and right child of current node and key value\*/

class Node {

int key;

Node left, right;

public Node(int item) {

key = item;

left = right = null;

}

}

// Root of BST

Node root;

// Constructor

BinarySearchTree() {

root = null;

}

// This method mainly calls insertRec()

void insert(int key) {

root = insertRec(root, key);

}

/\* A recursive function to insert a new key in BST \*/

Node insertRec(Node root, int key) {

/\* If the tree is empty, return a new node \*/

if (root == null) {

root = new Node(key);

return root;

}

/\* Otherwise, recur down the tree \*/

**Add your code here.**

/\* return the (unchanged) node pointer \*/

return root;

}

// This method mainly calls InorderRec()

void inorder() {

inorderRec(root);

}

// A utility function to do inorder traversal of BST

void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

System.out.println(root.key);

inorderRec(root.right);

}

}

// Driver Program to test above functions

public static void main(String[] args) {

BinarySearchTree tree = new BinarySearchTree();

/\* Let us create following BST

tree.insert(50);

tree.insert(30);

tree.insert(20);

tree.insert(40);

tree.insert(70);

tree.insert(60);

tree.insert(80);

// print inorder traversal of the BST

tree.inorder();

}

}

**AVL Tree**

Build an AVL tree with the following values:

15, 20, 24, 10, 13, 7, 30, 36, 25

Perform the following operations:

Remove 24 from the tree

Remove 20 from the tree

Show the whole process of inserting and deleting and also the final structure of the AVL tree.

Please refer to the slide deck “AVL animation exercise.ppt” for more details.