**Course: Data Structures and Algorithms**

**Lab 07**

**Operations in a BST**

**Task 1:**

**class BinarySearchTree{  
Node root;  
BinarySearchTree(){  
  
}**

***public class BST {***

***static class Node {***

***int data;***

***Node left;***

***Node right;***

***Node(int data) {***

***this.data = data;***

***this.left = null;***

***this.right = null;***

***}***

***}***

**//insertion: insert values in a binary search tree here**

***public static Node insertNode(Node root, int value) {***

***if (root == null) {***

***root = new Node(value);***

***return root;***

***}***

***if (root.data > value) {***

***root.left = insertNode(root.left, value);***

***} else {***

***root.right = insertNode(root.right, value);***

***}***

***return root;***

***}***

//Node left(node): returns left child of node

***public Node left(Node node) {***

***return node.left;***

***}***

//Node right(node): returns right child of node  
 ***public Node right(Node node) {***

***return node.right;***

***}***

//boolean hasLeft(node): tells if a node has left child or not

***public boolean hasLeft(Node node) {***

***return node.left != null;***

***}***

//boolean hasRight(node): tells if a node has right child or not

***public boolean hasRight(Node node) {***

***return node.right != null;***

***}***

//method to return parent of a given Node

Node parent(Node node,Node root)

***public Node parent(Node node, Node root) {***

***if (root == null || root == node) {***

***return null;***

***}***

***if (root.left == node || root.right == node) {***

***return root;***

***}***

***if (node.data < root.data) {***

***return parent(node, root.left);***

***} else {***

***return parent(node, root.right);***

***}***

***}***

//Node sibling(Node node): returns sibling of given node

***public Node sibling(Node node, Node root) {***

***Node parent = parent(node, root);***

***if (parent == null) {***

***return null;***

***}***

***if (parent.left == node) {***

***return parent.right;***

***} else {***

***return parent.left;***

***}***

***}***

//implement the following algorithms

//preorder  
Recursive\_PreOrder(Tree node)

***public static void recursive\_preorder(Node root){***

***if(root!=null){***

***System.out.println(root.data);***

***recursive\_postorder(root.left);***

***recursive\_postorder(root.right);***

}

}

If node is not NULL

print(node)

Recursive\_PreOrder(node.left)

Recursive\_PreOrder(node.right)

End If

//postorder   
 Recursive\_PostOrder(Tree node)

***public static void recursive\_postorder(Node root){***

***if(root!=null){***

***recursive\_postorder(root.left);***

***recursive\_postorder(root.right);***

***System.out.println(root.data);***

***}***

}

If node is not NULL

Recursive\_PostOrder(node.left)

Recursive\_PostOrder(node.right)

print(node)

End If

//inorder

***public static void recursive\_inorder(Node root){***

***if(root!=null){***

***recursive\_inorder(root.left);***

***System.out.println(root.data);***

***recursive\_inorder(root.right);***

}

}

 Recursive\_InOrder(Tree node)

If node is not NULL

Recursive\_InOrder(node.left)

print(node)

Recursive\_InOrder(node.right)

End If

//Node search

***public Node search(Node root, int value) {***

***if (root == null || root.data == value) {***

***return root;***

***}***

***if (value < root.data) {***

***return search(root.left, value);***

***} else {***

***return search(root.right, value);***

***}***

***}***

//Node deletion:

***public Node delete(Node root, int value) {***

***if (root == null) {***

***return root;***

***}***

***if (value < root.data) {***

***root.left = delete(root.left, value);***

***} else if (value > root.data) {***

***root.right = delete(root.right, value);***

***} else {***

***if (root.left == null) {***

***return root.right;***

***} else if (root.right == null) {***

***return root.left;***

***}***

***Node minNode = findMin(root.right);***

***root.data = minNode.data;***

***root.right = delete(root.right, minNode.data);***

***}***

***return root;***

***}***

***private Node findMin(Node node) {***

***while (node.left != null) {***

***node = node.left;***

***}***

***return node;***

***}***

//#################  
public class Main {  
// the user will be asked what type of binary tree he wants to build, then the values for the tree will be taken from the user. User can add as much values as he wants to add. then the choice of operation will be asked from the user.  
}

***public static void main(String[] args) {***

***int value[]={1,2,3,4,5,6,7,8,9};***

***Node root = null;***

***for(int i = 0 ;i<=value.length;i++){***

***root = insertNode(root,value[i]);***

***}***

***}***