LAB 11

PRACTICE QUESTIONS

Program 1: Write a code for the construction of Binary Search Tree.

**package** PRACTICE;

**public** **class** EX\_1 {

**static** **class** Node{

**public** **int** data;

**public** Node left;

**public** Node right;

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

**this**.data = data;

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

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

}

}

**public** Node root;

**public** EX\_1() {

**this**.root = **null**;

}

**public** **boolean** isBSTornot() {

**return** isBSTornot(**this**.root, Integer.***MAX\_VALUE***, Integer.***MAX\_VALUE***);

}

**private** **boolean** isBSTornot(Node root, **int** minValue, **int** maxValue) {

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

**return** **true**;

}

**if**(root.data >= minValue && root.data <= maxValue && isBSTornot(root.left,minValue, root.data)

&& isBSTornot(root.right, root.data, maxValue)) {

**return** **true**;

}

**return** **false**;

}

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

EX\_1 bt = **new** EX\_1();

bt.root= **new** Node(100);

bt.root.left= **new** Node(90);

bt.root.right= **new** Node(110);

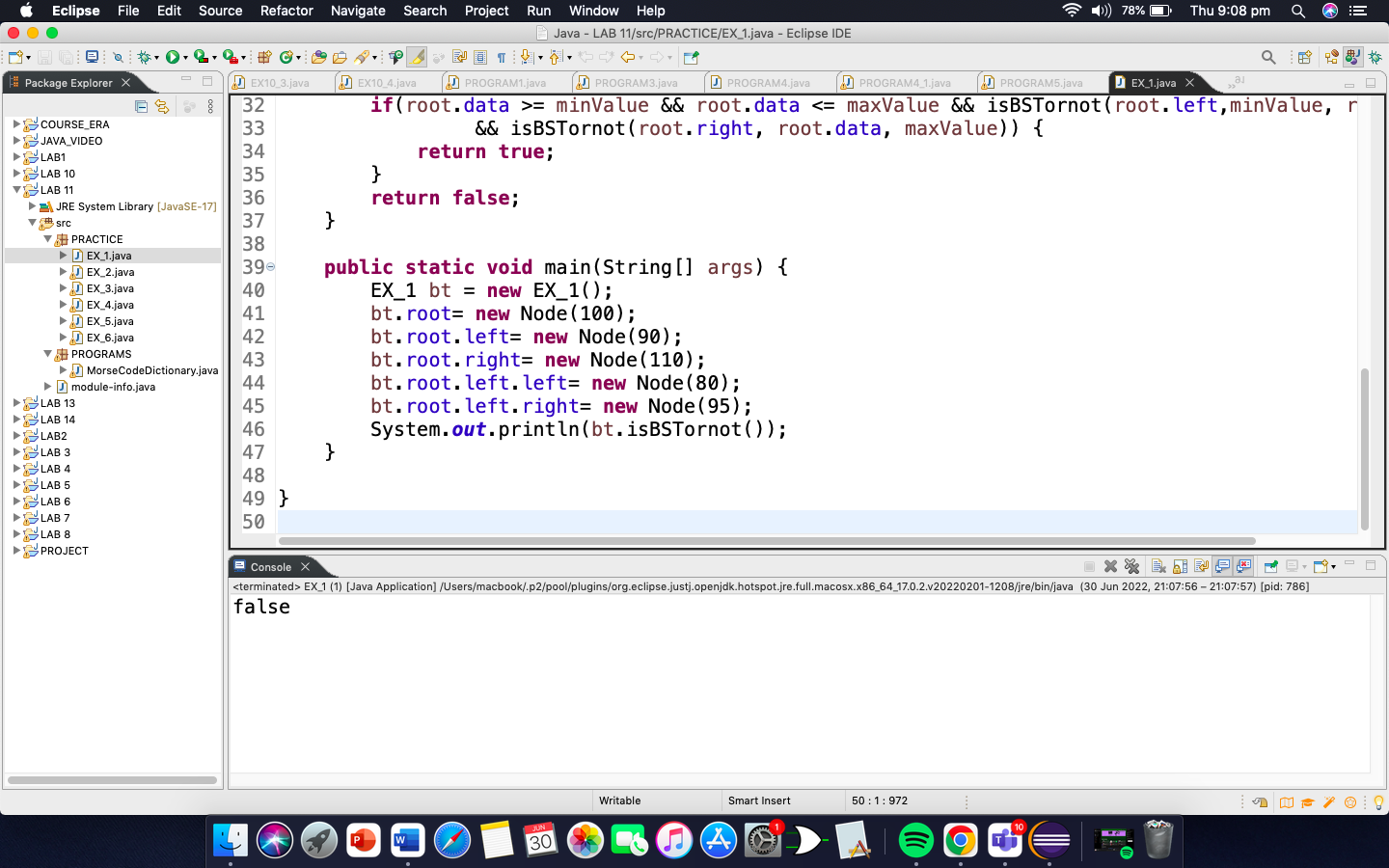
bt.root.left.left= **new** Node(80);

bt.root.left.right= **new** Node(95);

System.***out***.println(bt.isBSTornot());

}

}



Program 2: Write a code to insert a value in BST.

**package** PRACTICE;

**import** PRACTICE.EX\_1.Node;

**public** **class** EX\_2 {

**static** **class** Node{

**public** **int** data;

**public** Node left;

**public** Node right;

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

**this**.data = data;

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

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

}

}

**public** Node root;

**public** EX\_2() {

**this**.root = **null**;

}

**public** **void** insert(**int** newData) {

**this**.root = insert(root, newData);

}

**public** Node insert(Node root, **int** newData) {

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

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

**return** root;

}

**else** **if**(root.data >= newData) {

root.left = insert(root.left, newData);

}

**else** {

root.right = insert(root.right, newData);

}

**return** root;

}

**public** **void** preOrder() {

preOrder(root);

}

**public** **void** preOrder(Node root) {

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

**return**;

}

System.***out***.print(root.data + " ");

preOrder(root.left);

preOrder(root.right);

}

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

**long** start4 = System.*currentTimeMillis*();

EX\_2 bst = **new** EX\_2();

bst.insert(2);

bst.insert(4);

bst.insert(1);

bst.insert(3);

bst.insert(5);

bst.preOrder();

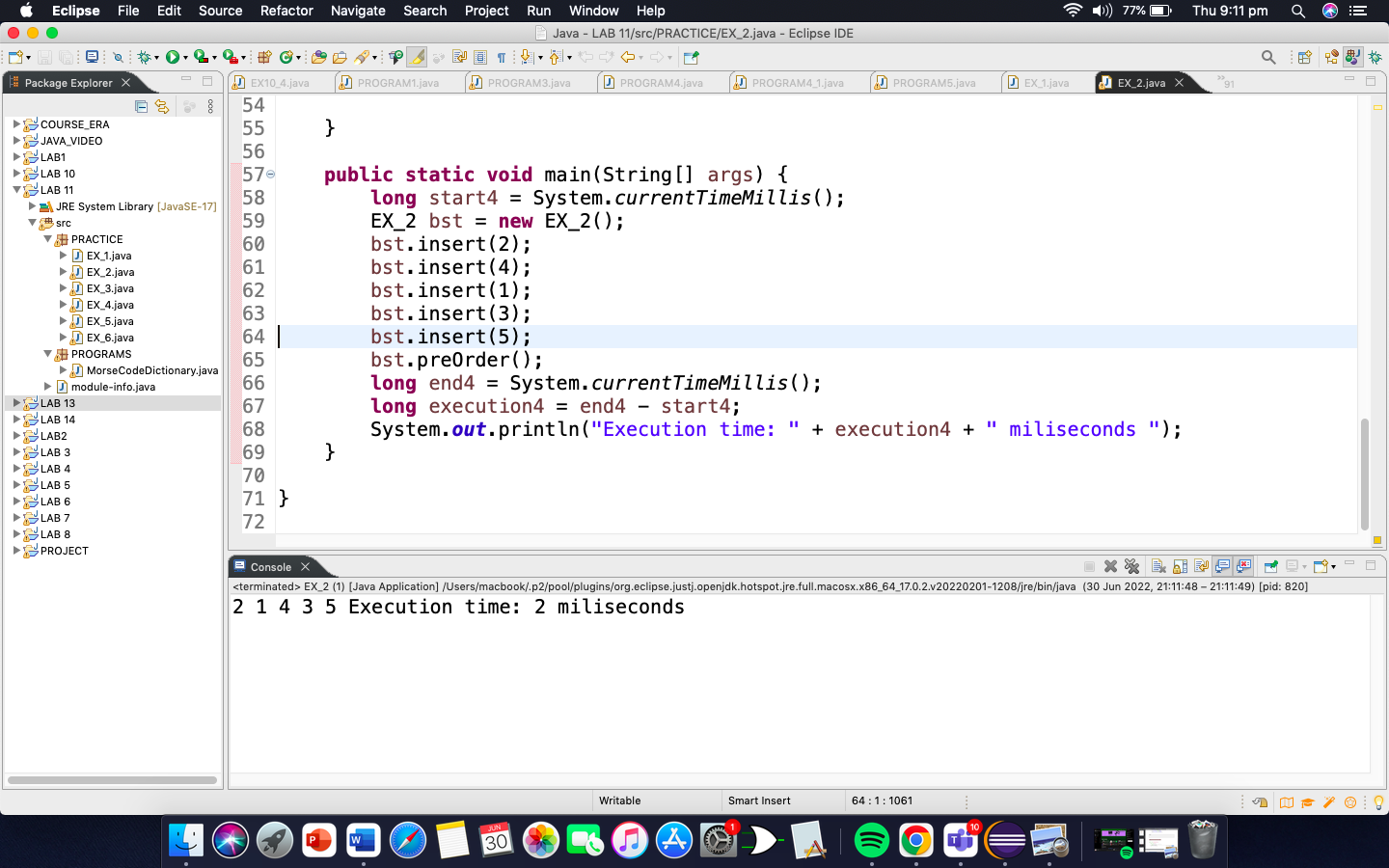
**long** end4 = System.*currentTimeMillis*();

**long** execution4 = end4 - start4;

System.***out***.println("Execution time: " + execution4 + " miliseconds ");

}

}



Program3: Write a code to delete a value in BST.

**package** PRACTICE;

**import** PRACTICE.EX\_2.Node;

**public** **class** EX\_3 {

**static** **class** Node{

**public** **int** data;

**public** Node left;

**public** Node right;

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

**this**.data = data;

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

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

}

}

**public** Node root;

**public** EX\_3() {

**this**.root = **null**;

}

**public** **void** insert(**int** newData) {

**this**.root = insert(root, newData);

}

**public** Node insert(Node root, **int** newData) {

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

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

**return** root;

}

**else** **if**(root.data >= newData) {

root.left = insert(root.left, newData);

}

**else** {

root.right = insert(root.right, newData);

}

**return** root;

}

**public** **void** deleteNode(Node node) {

deleteNode(**this**.root, node);

}

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

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

**return** **null**;

}

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

root.left = deleteNode(root.left, node);

}

**else** **if**(node.data>root.data) {

root.right = deleteNode(root.right,node);

}

**else** **if**(node.data==root.data) {

**if**(root.left != **null** && root.right !=**null**) {

**int** lmax = findmaxData(root.left);

root.data = lmax;

root.left = deleteNode(root.left, **new** Node(lmax));

**return** root;

}

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

**return** root.left;

}

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

**return** root.right;

}

**else** {

**return** **null**;

}

}

**return** root;

}

**public** **int** findmaxData(Node root) {

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

**return** findmaxData(root.right);

}**else** {

**return** root.data;

}

}

**public** **void** preOrder() {

preOrder(root);

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

}

**public** **void** preOrder(Node node) {

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

System.***out***.print(node.data + " ");

preOrder(node.left);

preOrder(node.right);

}

}

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

**long** start4 = System.*currentTimeMillis*();

// Creating the object of BinarySearchTree class

EX\_3 bst = **new** EX\_3(); // call the method insert

bst.insert(8);

bst.insert(5);

bst.insert(9);

bst.insert(3);

bst.insert(7);

bst.preOrder();

bst.deleteNode(**new** Node(9));

bst.preOrder();

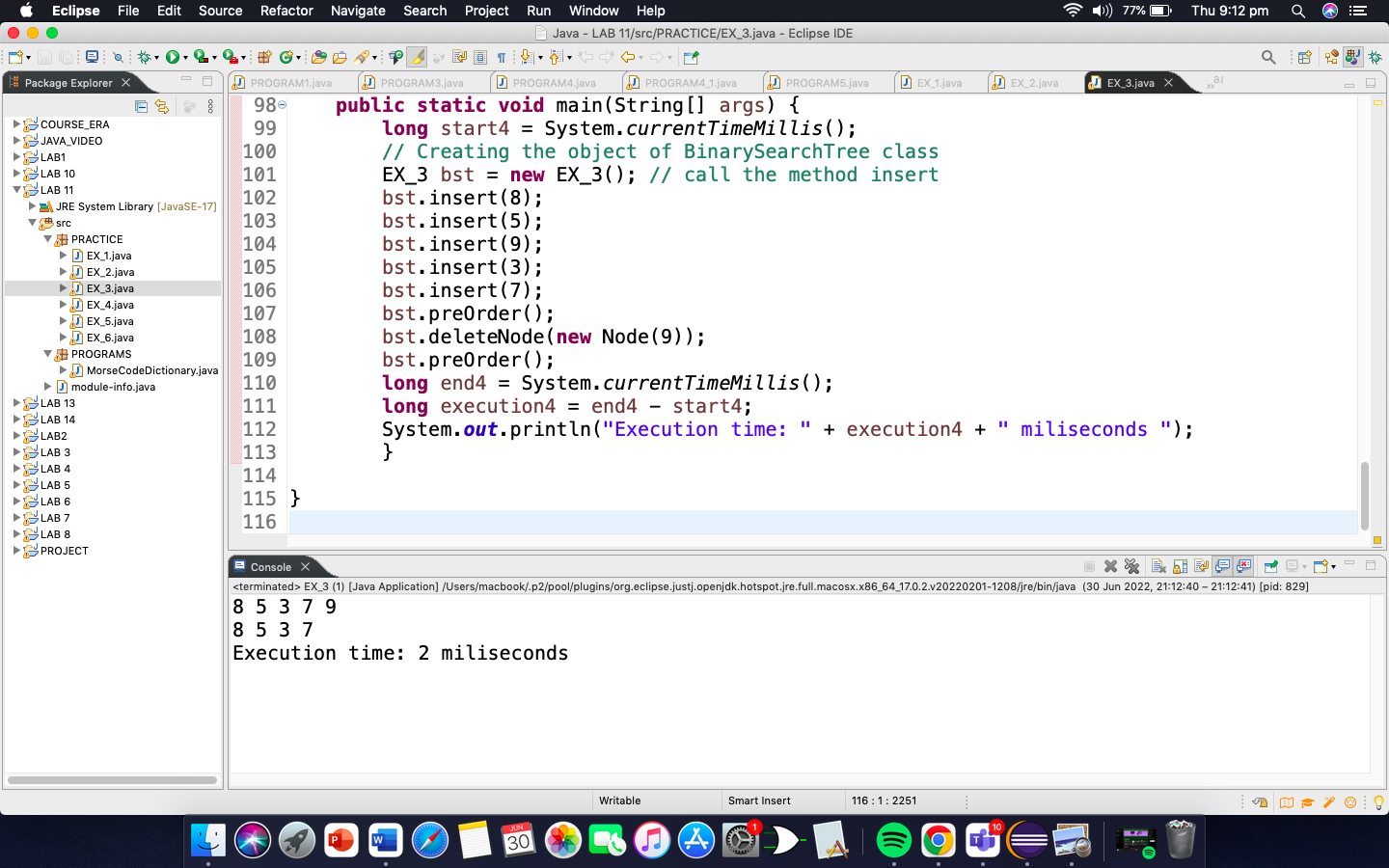
**long** end4 = System.*currentTimeMillis*();

**long** execution4 = end4 - start4;

System.***out***.println("Execution time: " + execution4 + " miliseconds ");

}

}



Program4: Write a code to search specific value from BST.

**package** PRACTICE;

**import** PRACTICE.EX\_3.Node;

**public** **class** EX\_4 {

**static** **class** Node{

**public** **int** data;

**public** Node left;

**public** Node right;

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

**this**.data = data;

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

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

}

}

**public** Node root;

**public** EX\_4() {

**this**.root = **null**;

}

**public** **void** insert(**int** newData) {

**this**.root = insert(root, newData);

}

**public** Node insert(Node root, **int** newData) {

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

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

**return** root;

}

**else** **if**(root.data >= newData) {

root.left = insert(root.left, newData);

}

**else** {

root.right = insert(root.right, newData);

}

**return** root;

}

**public** **int** findMaximum() {

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

**return** -1;

}

Node current = **this**.root;

**while**(current.right!=**null**) {

current = current.right;

}

**return** (current.data);

}

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

EX\_4 bst = **new** EX\_4();

**long** start4 = System.*currentTimeMillis*();

bst.insert(8);

bst.insert(5);

bst.insert(3);

bst.insert(7);

bst.insert(9); System.***out***.println(bst.findMaximum());

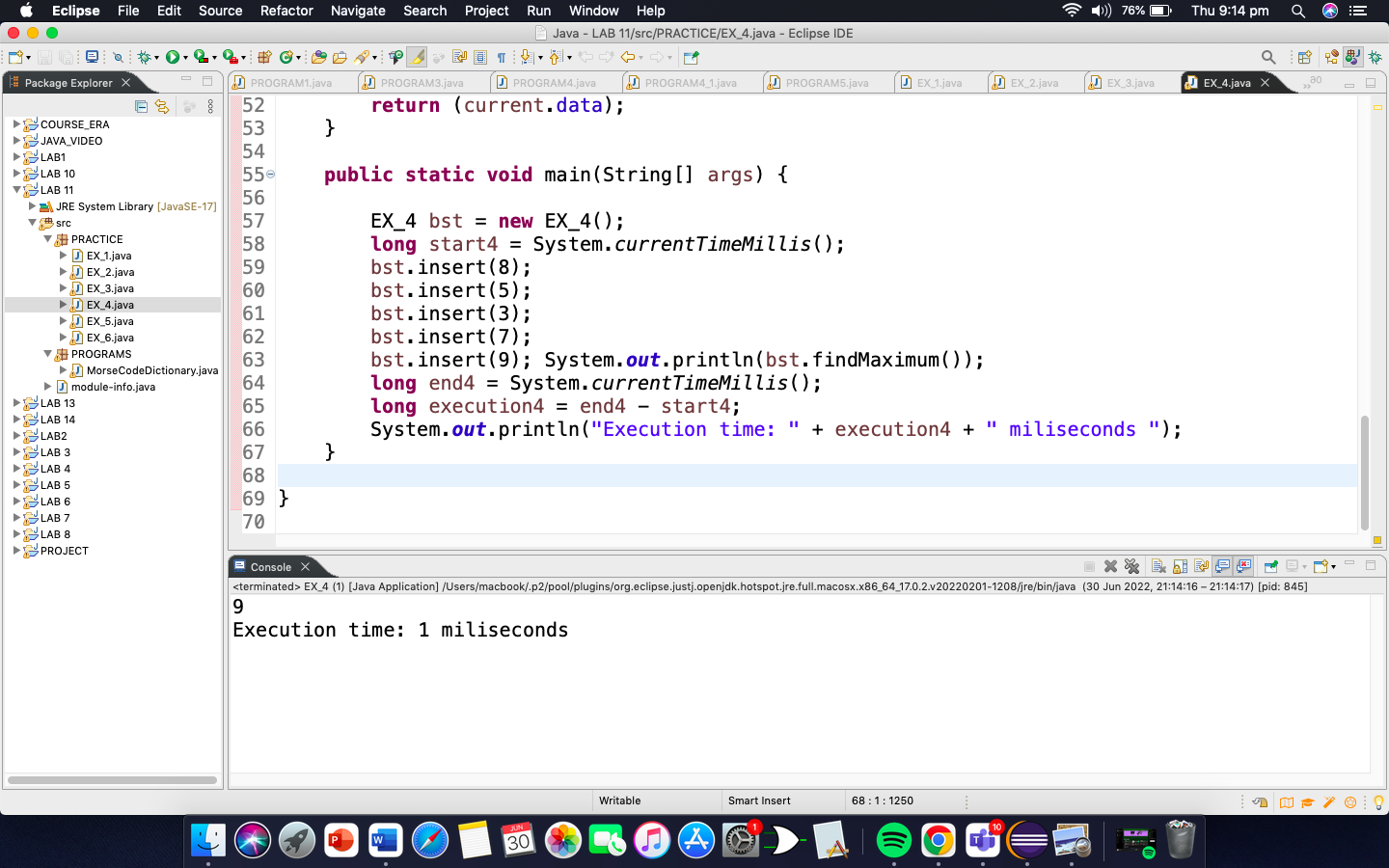
**long** end4 = System.*currentTimeMillis*();

**long** execution4 = end4 - start4;

System.***out***.println("Execution time: " + execution4 + " miliseconds ");

}

}



Program5: Write a code to search maximum value from BST.

**package** PRACTICE;

**import** PRACTICE.EX\_4.Node;

**public** **class** EX\_5 {

**static** **class** Node{

**public** **int** data;

**public** Node left;

**public** Node right;

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

**this**.data = data;

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

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

}

}

**public** Node root;

**public** EX\_5() {

**this**.root = **null**;

}

**public** **void** insert(**int** newData) {

**this**.root = insert(root, newData);

}

**public** Node insert(Node root, **int** newData) {

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

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

**return** root;

}

**else** **if**(root.data >= newData) {

root.left = insert(root.left, newData);

}

**else** {

root.right = insert(root.right, newData);

}

**return** root;

}

**public** **boolean** search(**int** data) {

**return** search(**this**.root, data);

}

**public** **boolean** search(Node root, **int** data) {

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

**return** **false**;

}

**else** **if**(root.data==data) {

**return** **true**;

}

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

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

}

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

}

**public** **void** preOrder() {

preOrder(root);

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

}

**public** **void** preOrder(Node root) {

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

**return**;

}

System.***out***.print(root.data + " ");

preOrder(root.left);

preOrder(root.right);

}

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

**long** start4 = System.*currentTimeMillis*();

EX\_5 bst = **new** EX\_5();

bst.insert(8);

bst.insert(5);

bst.insert(9);

bst.insert(3);

bst.insert(7);

bst.preOrder();

System.***out***.println(bst.search(7));

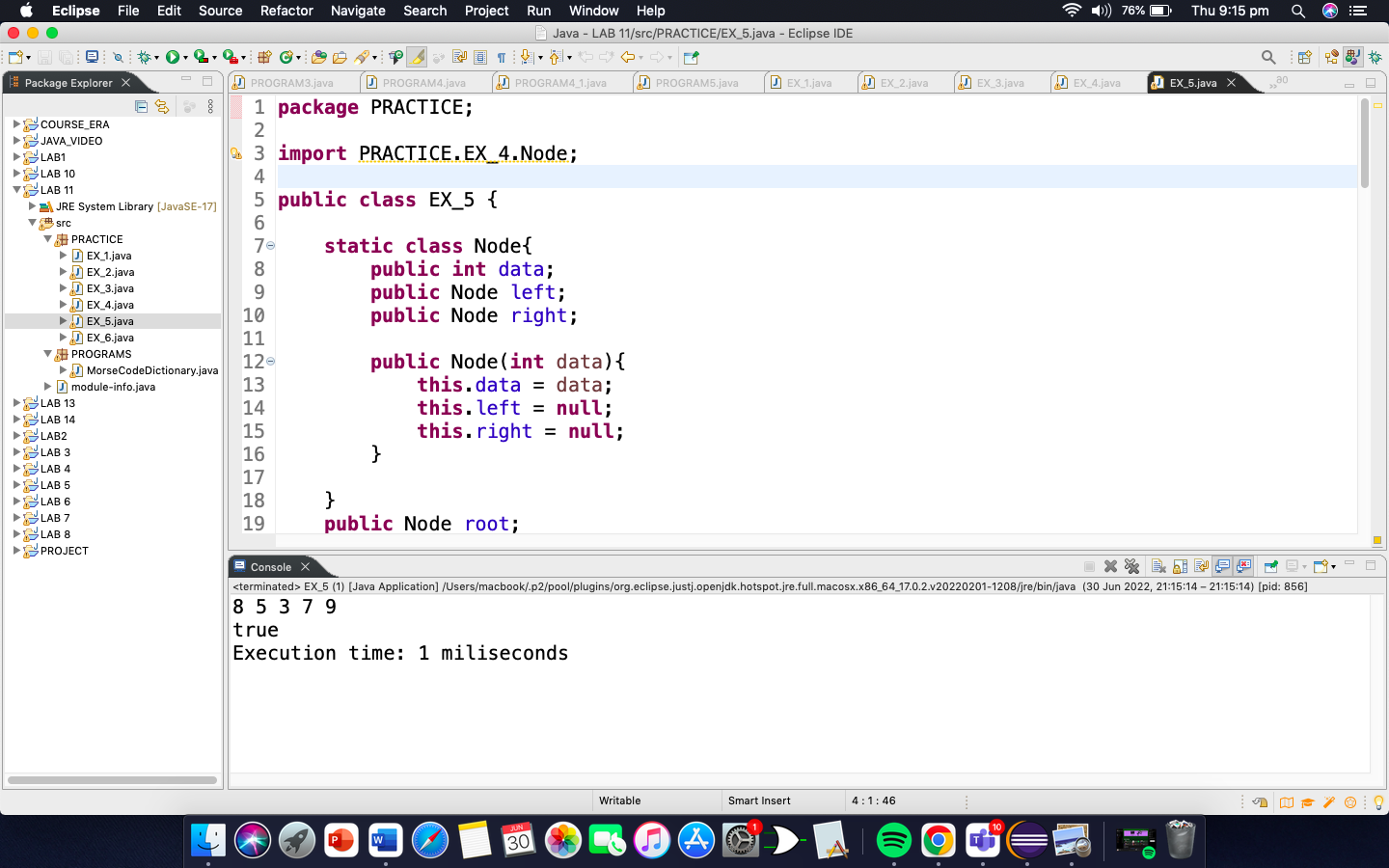
**long** end4 = System.*currentTimeMillis*();

**long** execution4 = end4 - start4;

System.***out***.println("Execution time: " + execution4 + " miliseconds ");

}

}



Program6: Write a code to search minimum value from BST.

**package** PRACTICE;

**import** PRACTICE.EX\_4.Node;

**public** **class** EX\_6 {

**static** **class** Node{

**public** **int** data;

**public** Node left;

**public** Node right;

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

**this**.data = data;

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

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

}

}

**public** Node root;

**public** EX\_6() {

**this**.root = **null**;

}

**public** **void** insert(**int** newData) {

**this**.root = insert(root, newData);

}

**public** Node insert(Node root, **int** newData) {

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

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

**return** root;

}

**else** **if**(root.data >= newData) {

root.left = insert(root.left, newData);

}

**else** {

root.right = insert(root.right, newData);

}

**return** root;

}

**public** **int** findMinimum() {

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

**return** -1;

}

Node current = **this**.root;

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

current = current.left;

}

**return** (current.data);

}

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

**long** start4 = System.*currentTimeMillis*();

EX\_6 bst = **new** EX\_6();

bst.insert(8);

bst.insert(5);

bst.insert(3);

bst.insert(7);

bst.insert(9);

System.***out***.println(bst.findMinimum());

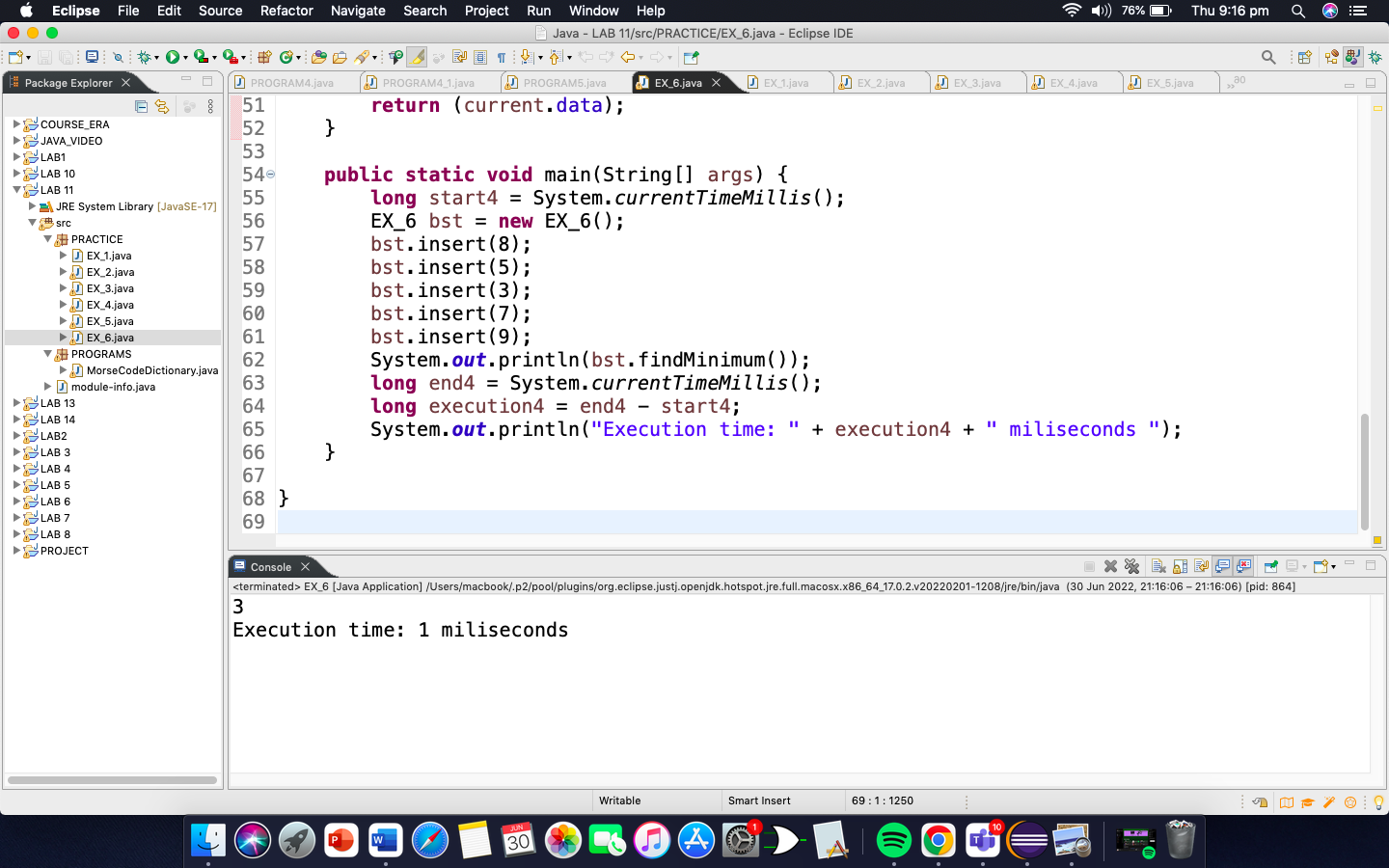
**long** end4 = System.*currentTimeMillis*();

**long** execution4 = end4 - start4;

System.***out***.println("Execution time: " + execution4 + " miliseconds ");

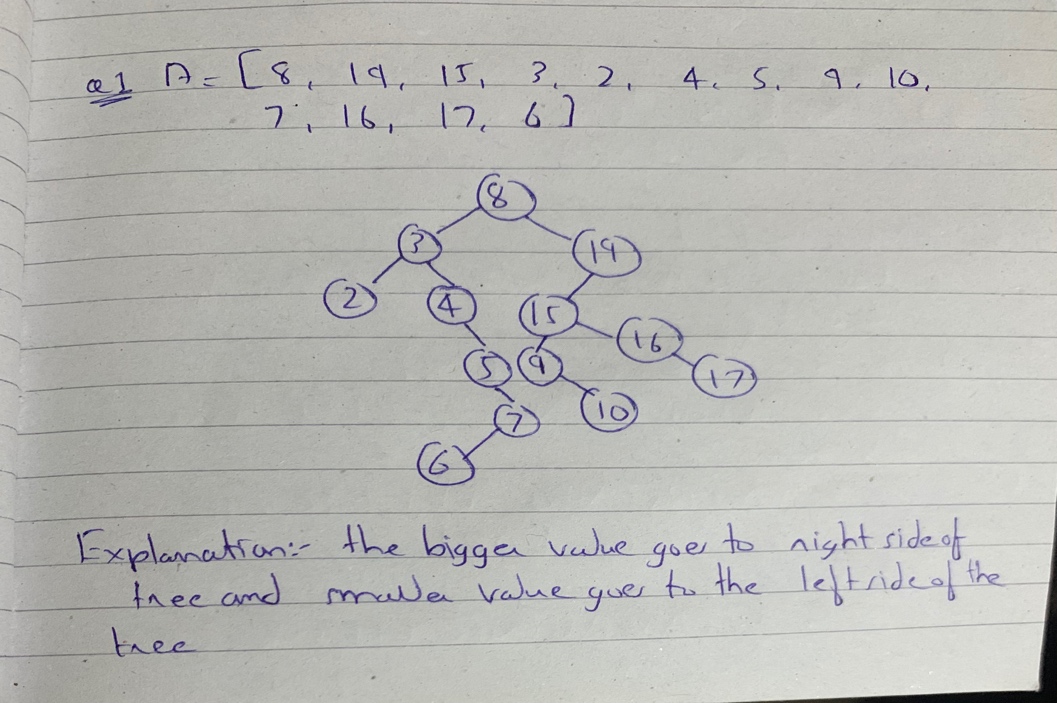
}

}

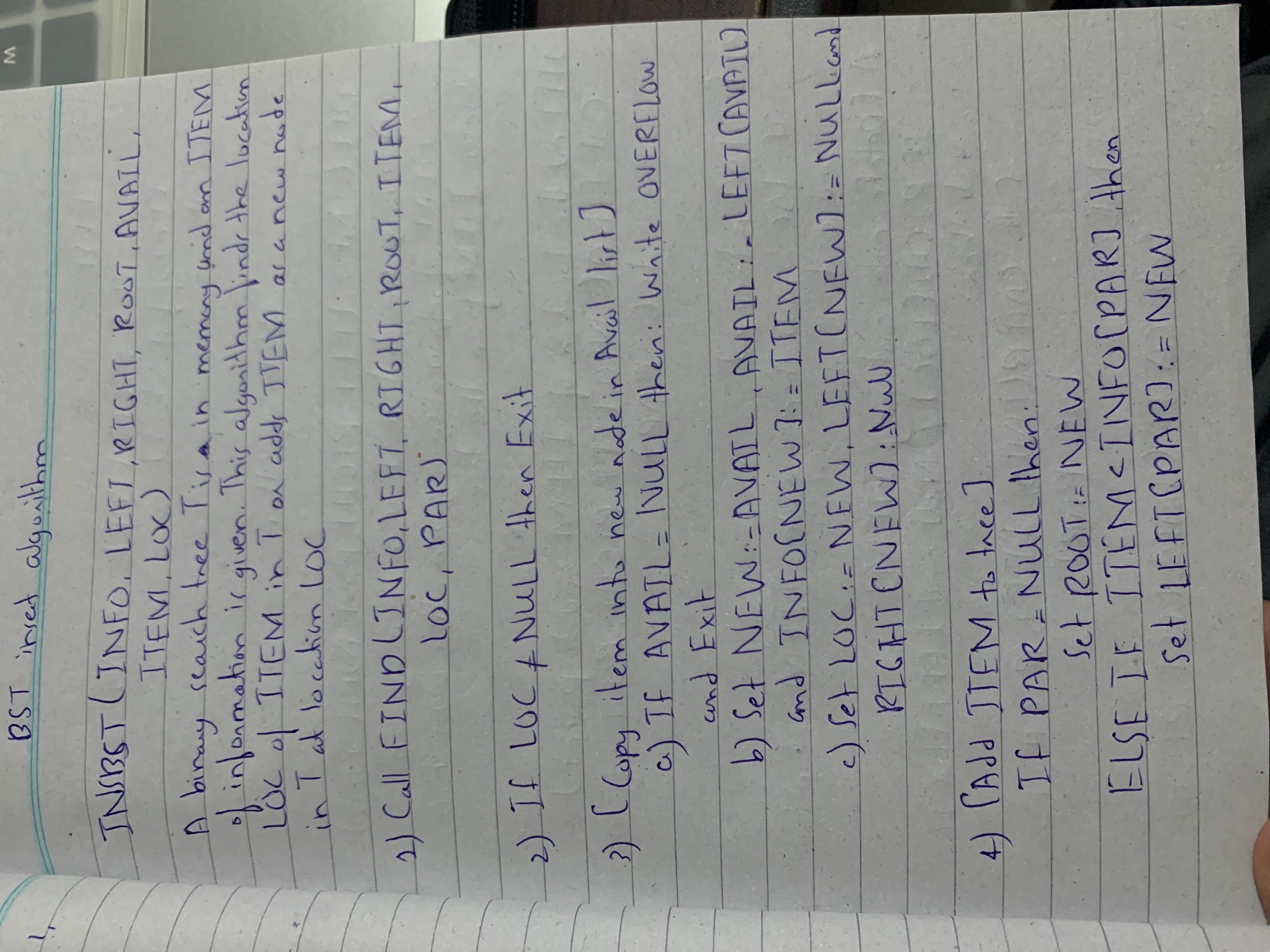


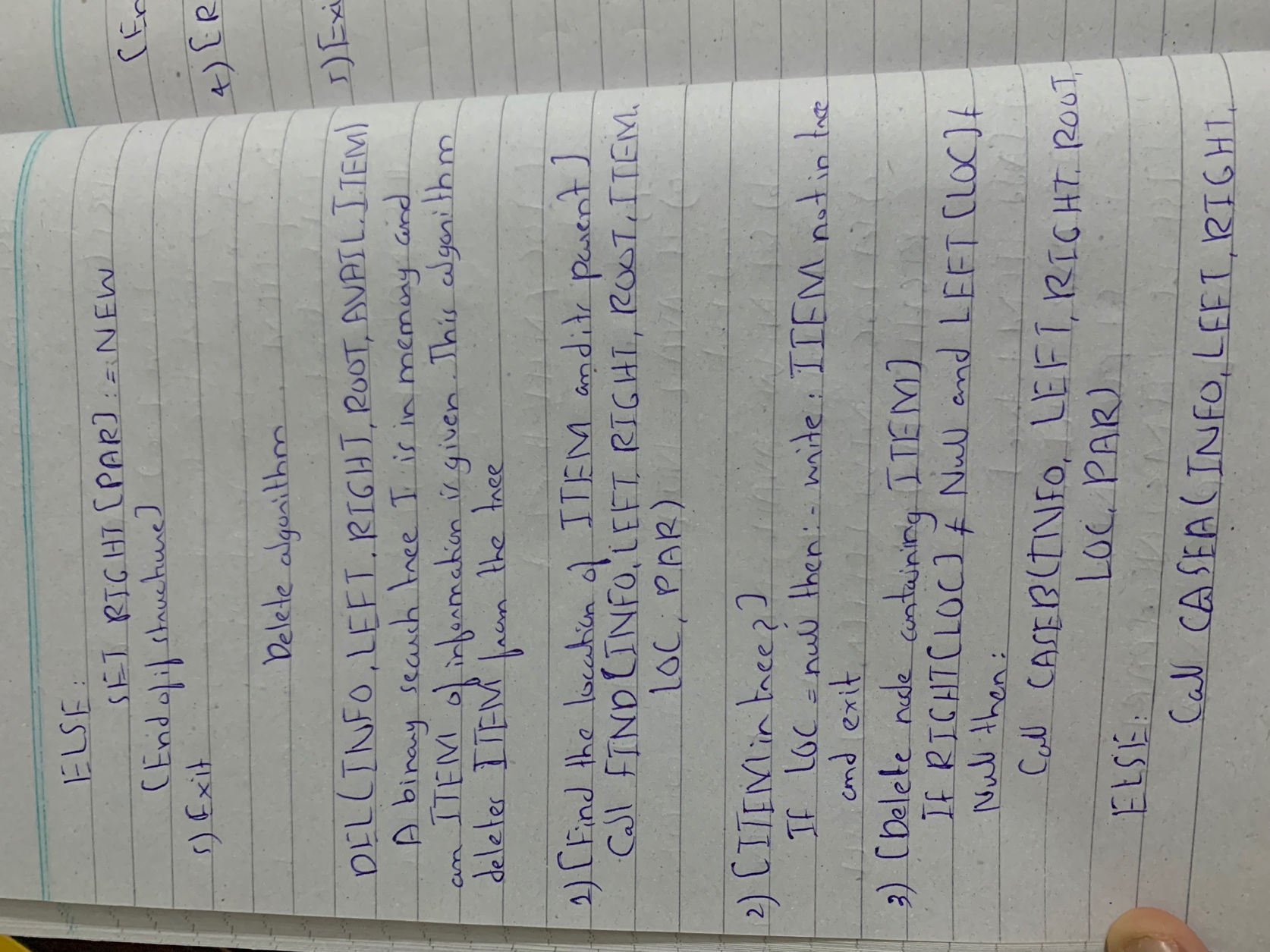
PROGRAMMING EXERCISE

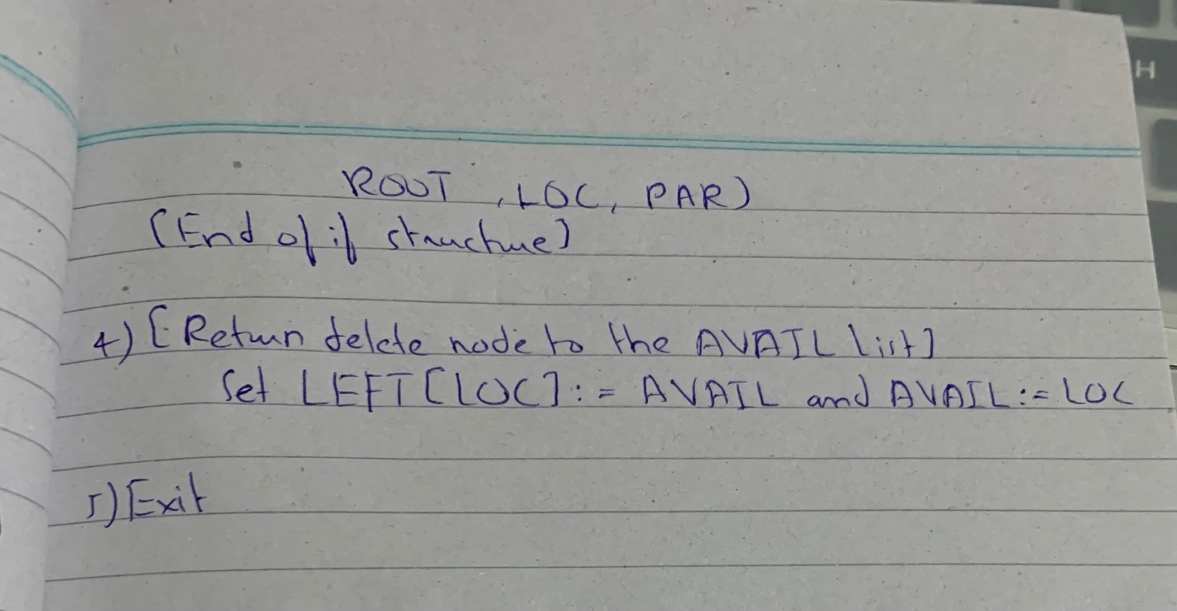
1. You have been given an unsorted array of 13 numbers. What will be the outcome of BST if you process it? Explain the working procedure.



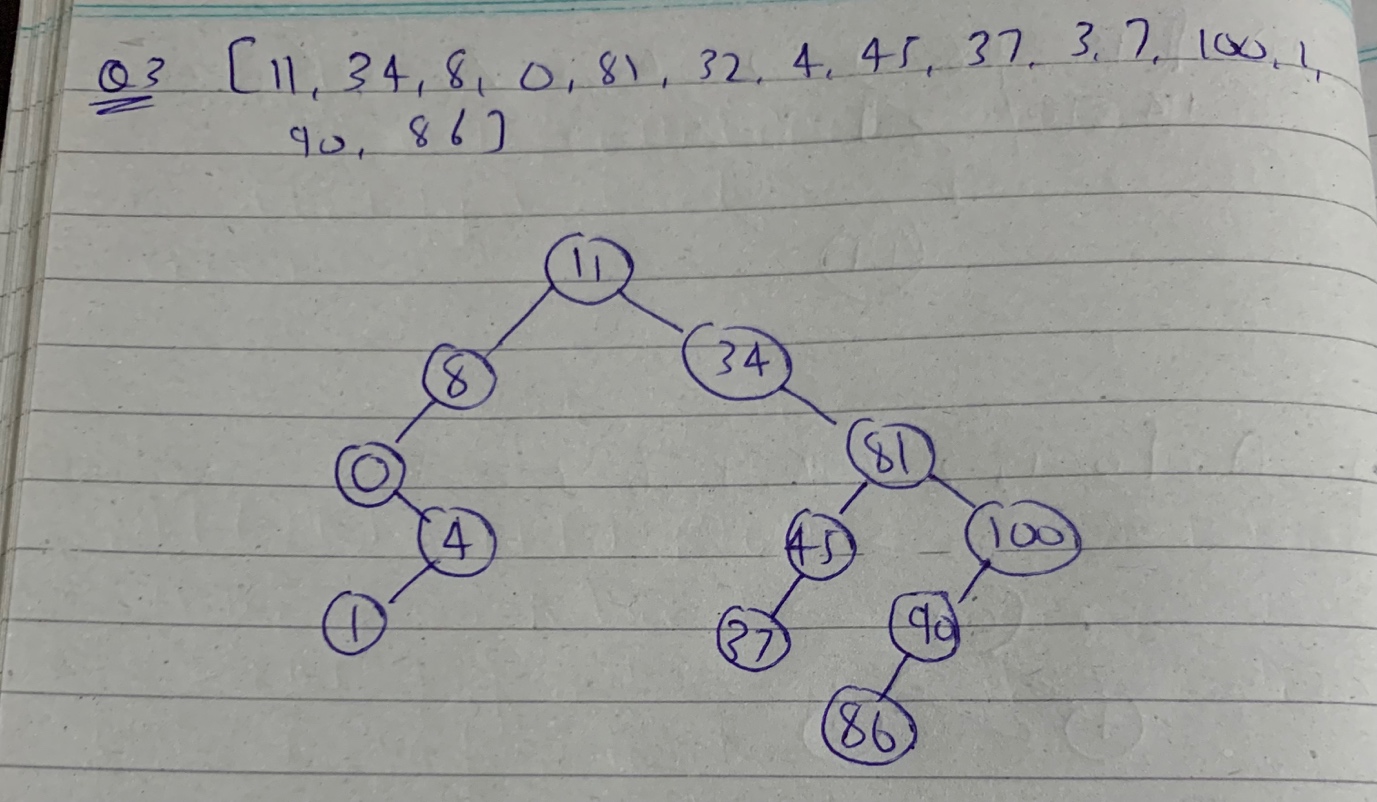
1. Write algorithm of Insert and delete specific elements from the BST.







1. What will be the graph if you have the data [11, 34, 8, 0, 81, 32, 4, 45, 37,3, 7, 100, 1, 90, 86]?



1. Write the complexity of the BST and its operations (insert, delete, search, traversing, min, max)

and calculate time of execution on your machine.

Complexity of BST is O(logn) and execution of every operation is given above with the output.

1. Implement the Morse Code using BST. You will get the following graph if you successfully implement the code

**package** PROGRAMS;

**import** java.util.\*;

**public** **class** MorseCodeDictionary {

MorseCodeNode root;

Map<String, String> alpha\_dictionary = **new** HashMap<String, String>() {{

put("a", ".-");

put("b", "-...");

put("c", "-.-.");

put("d", "-..");

put("e", ".");

put("f", "..-.");

put("g", "--.");

put("h", "....");

put("i", "..");

put("j", ".---");

put("k", "-.-");

put("l", ".-..");

put("m", "--");

put("n", "-.");

put("o", "---");

put("p", ".--.");

put("q", "--.-");

put("r", ".-.");

put("s", "...");

put("t", "-");

put("u", "..-");

put("v", "...-");

put("w", ".--");

put("x", "-..-");

put("y", "-.--");

put("z", "--..");

}};

**public** **class** MorseCodeNode {

**private** String letter;

**private** MorseCodeNode dot;

**private** MorseCodeNode dash;

**public** MorseCodeNode() {

letter = **null**;

dot = **null**;

dash = **null**;

}

**public** String getLetter() {

**return** letter;

}

**public** **void** setLetter(String l) {

letter = l;

}

**public** MorseCodeNode getDot() {

**return** dot;

}

**public** **void** setDot(MorseCodeNode n) {

dot = n;

}

**public** MorseCodeNode getDash() {

**return** dash;

}

**public** **void** setDash(MorseCodeNode n) {

dash = n;

}

**public** String toString() {

**return**

((letter == **null**) ? "nul" : " " + letter + " ") + " " +

"@(" + Integer.*toHexString*(**this**.hashCode()) + ") " +

".(" + ((dot == **null**) ? "null" : Integer.*toHexString*(dot.hashCode())) + ") " +

"-(" + ((dash == **null**) ? "null" : Integer.*toHexString*(dash.hashCode())) + ")";

}

**public** **void** toStringBuffer(StringBuffer s) {

/\*

Print out to graphviz / TD format / mermaid markdown

\*/

String h = " " + Integer.*toHexString*(**this**.hashCode());

s.append(

h + ((letter == **null**) ? "( )" : "(" + letter + ")") + "\n"

);

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

s.append(h + " -->|.| " +

Integer.*toHexString*(getDot().hashCode()) + "\n"

);

}

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

s.append(h + " -->|-| " +

Integer.*toHexString*(getDash().hashCode()) + "\n"

);

}

**if**(getDot() != **null**) { getDot().toStringBuffer(s); }

**if**(getDash() != **null**) { getDash().toStringBuffer(s); }

}

}

**public** MorseCodeDictionary () {

root = **new** MorseCodeNode();

Iterator i = alpha\_dictionary.entrySet().iterator();

**while** (i.hasNext()) {

Map.Entry<String,String> x = (Map.Entry<String,String>)i.next();

System.***out***.println("adding '" + x.getKey() + "' - '"+x.getValue() + "'");

add(x.getKey(), x.getValue());

}

}

**private** **void** initializeNodes() {

}

**private** **void** add(String character, String signal) {

MorseCodeNode current = root;

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

String c = signal.substring(i, i+1);

**if** (c.equals(".")) {

**if** (current.getDot() == **null**) {

current.setDot(**new** MorseCodeNode());

}

current = current.getDot();

} **else** **if** (c.equals("-")) {

**if** (current.getDash() == **null**) {

current.setDash(**new** MorseCodeNode());

}

current = current.getDash();

} **else** {

System.***out***.println("Error - unknown character '" + c + "' encountered");

**return**;

}

}

current.setLetter(character);

}

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

MorseCodeDictionary mcd = **new** MorseCodeDictionary();

MorseCodeNode m;

ArrayList<String> teststrings = **new** ArrayList<String>(Arrays.*asList*(

".-",

".",

"..",

"...",

"....",

".....",

"-----",

"......"

));

}

}

