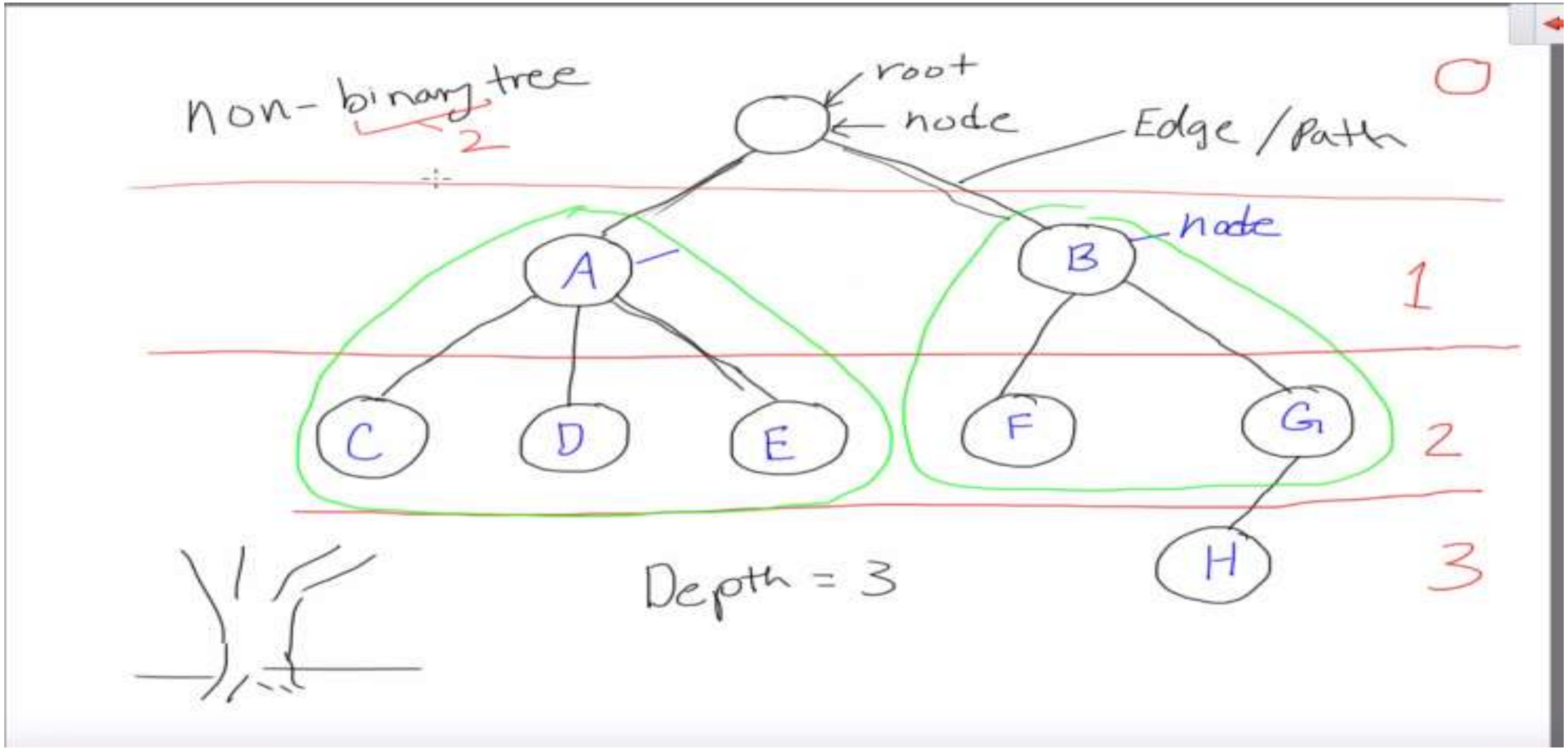
Binary Search Tree

- In computer science, a binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree. A binary tree is a type of data structure for storing data such as numbers in an organized way. Binary search trees allow binary search for fast lookup, addition and removal of data items . The order of nodes in a BST means that each comparison skips about half of the remaining tree, so the whole lookup takes time proportional to the binary logarithm of the number of items stored in the tree. This is much better than the linear time required to find items by key in an (unsorted) array

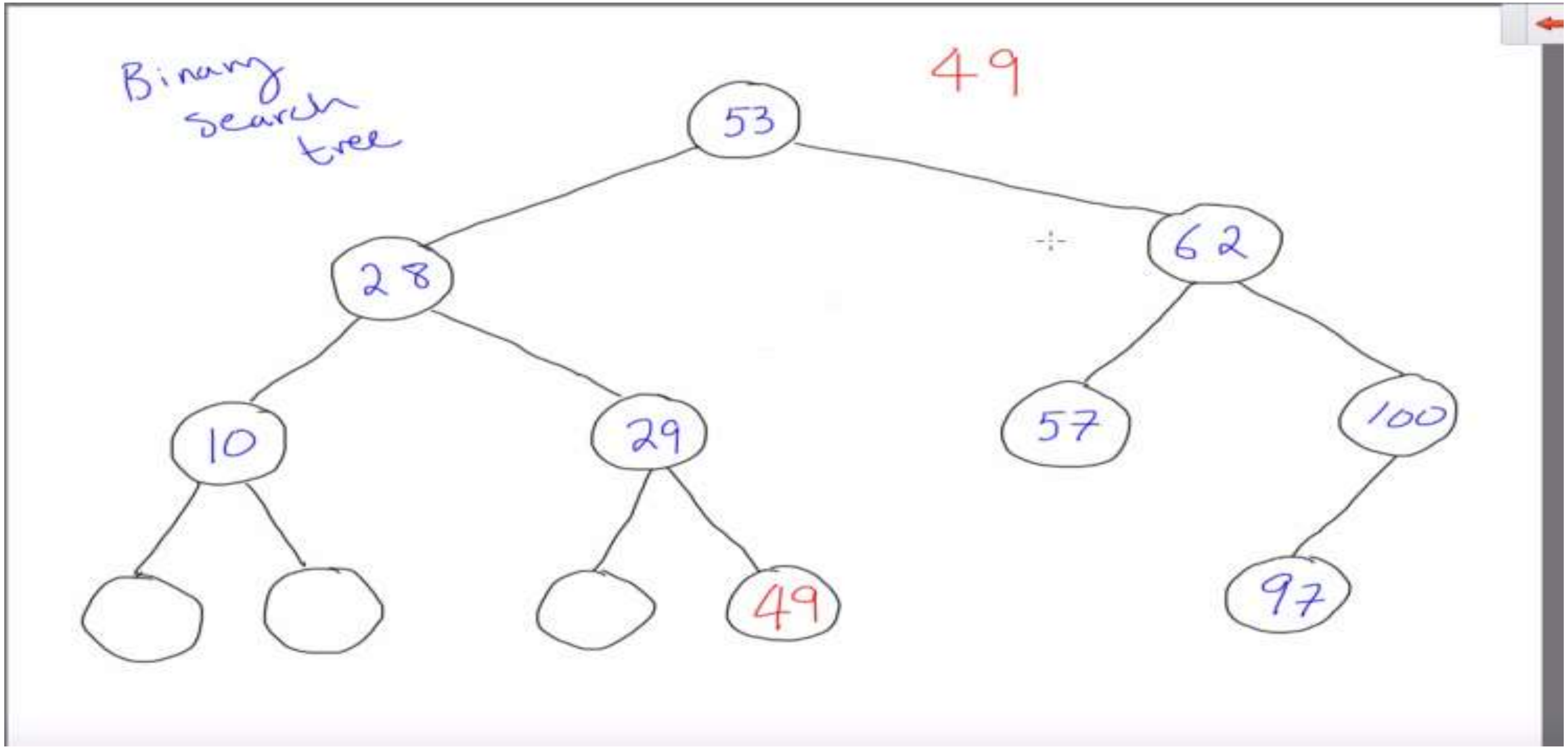
Complexity

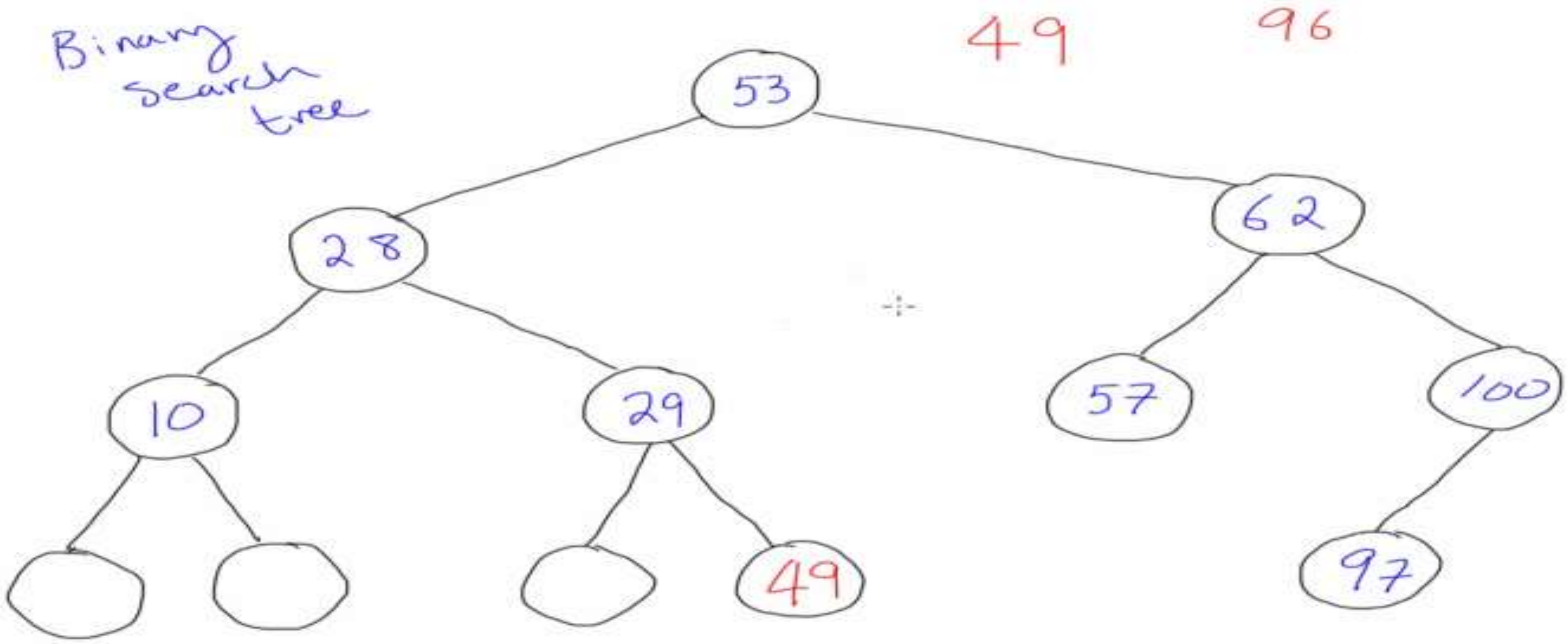
Algorithm	**Average**	**Worst case**
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(*n*)
Insert	O(log *n*)	O(*n*)
Delete	O(log *n*)	O(*n*)

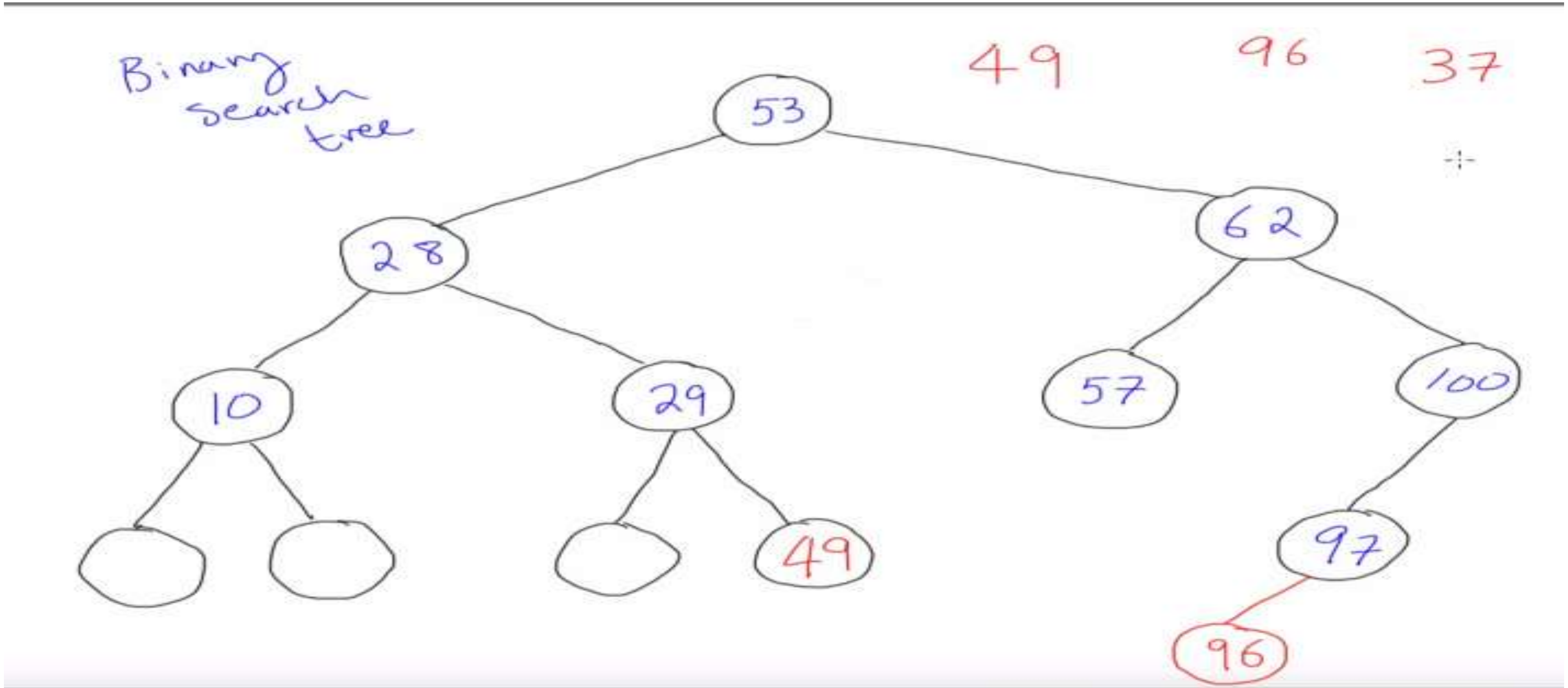
Non-Binary Tree Example

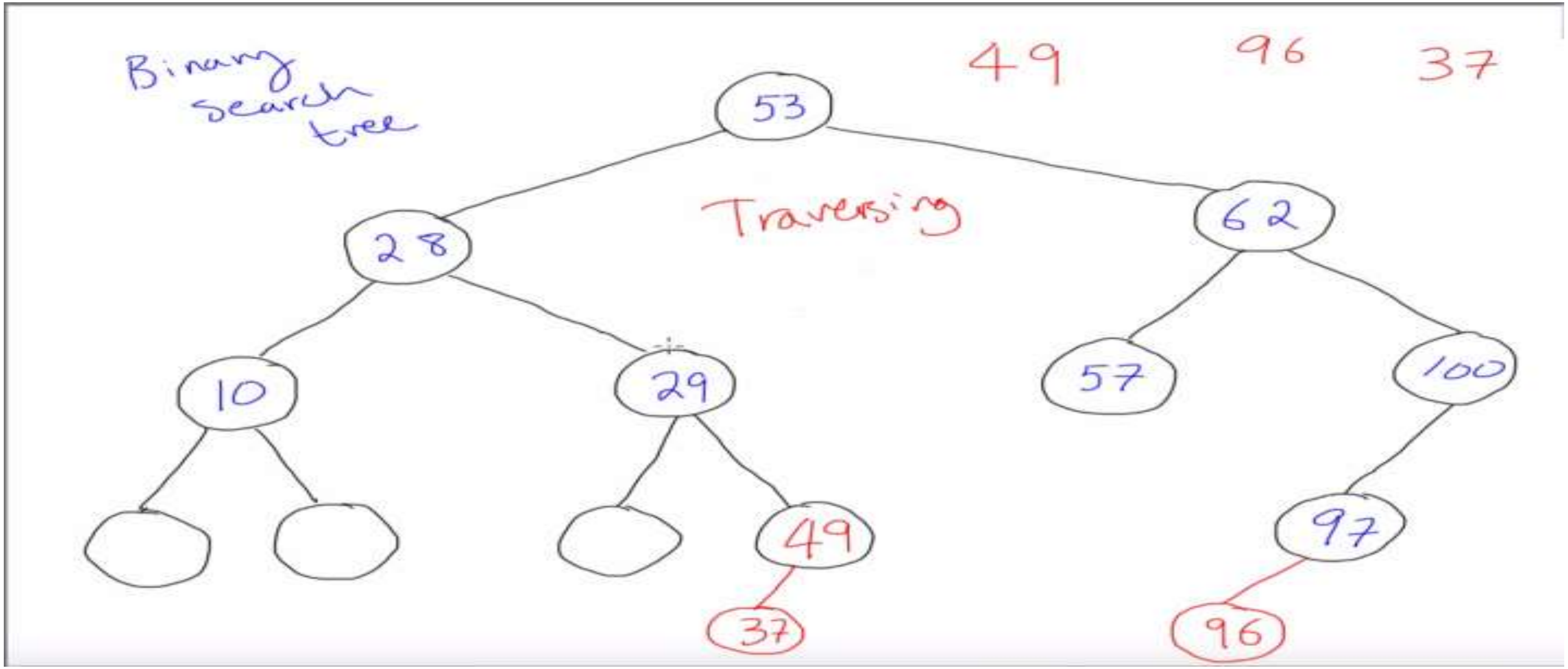


Binary Tree Example

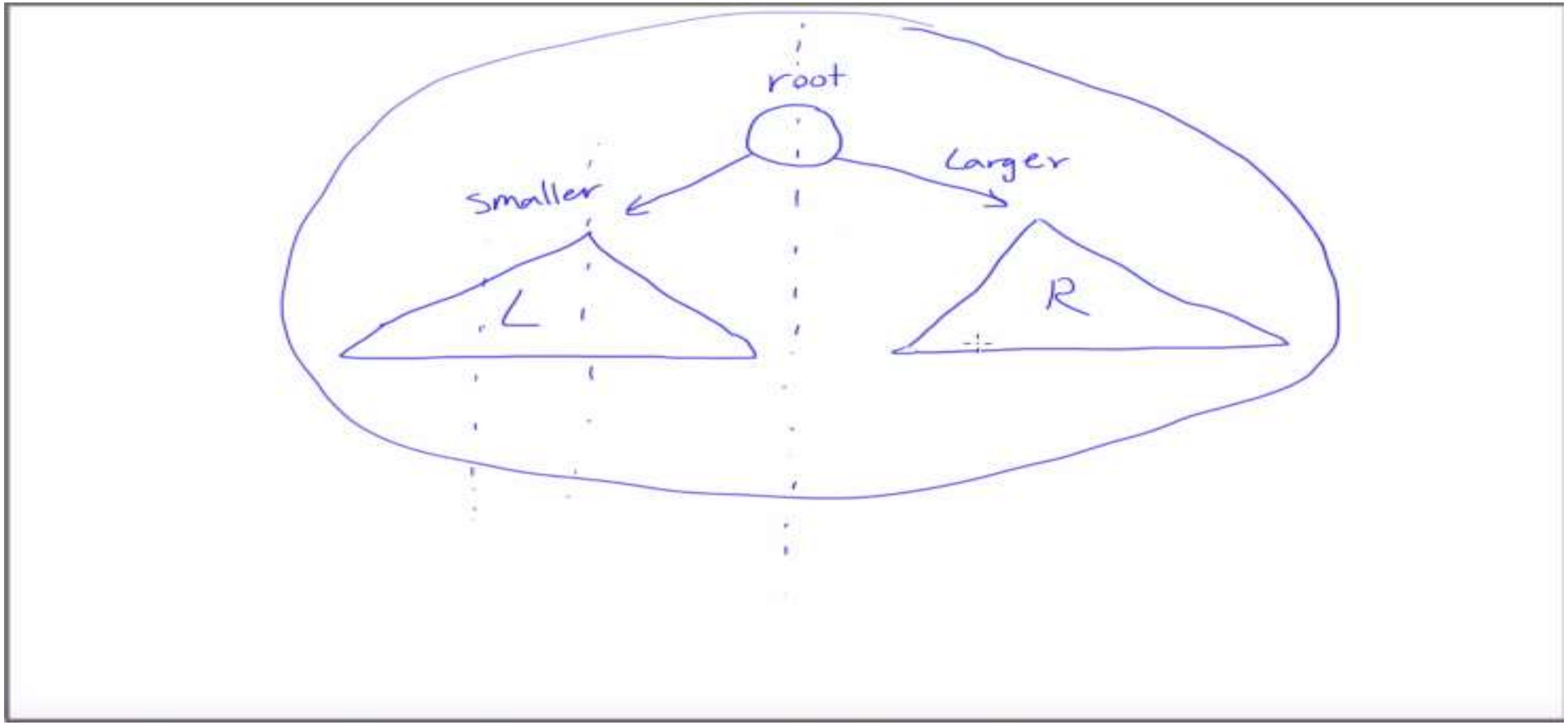


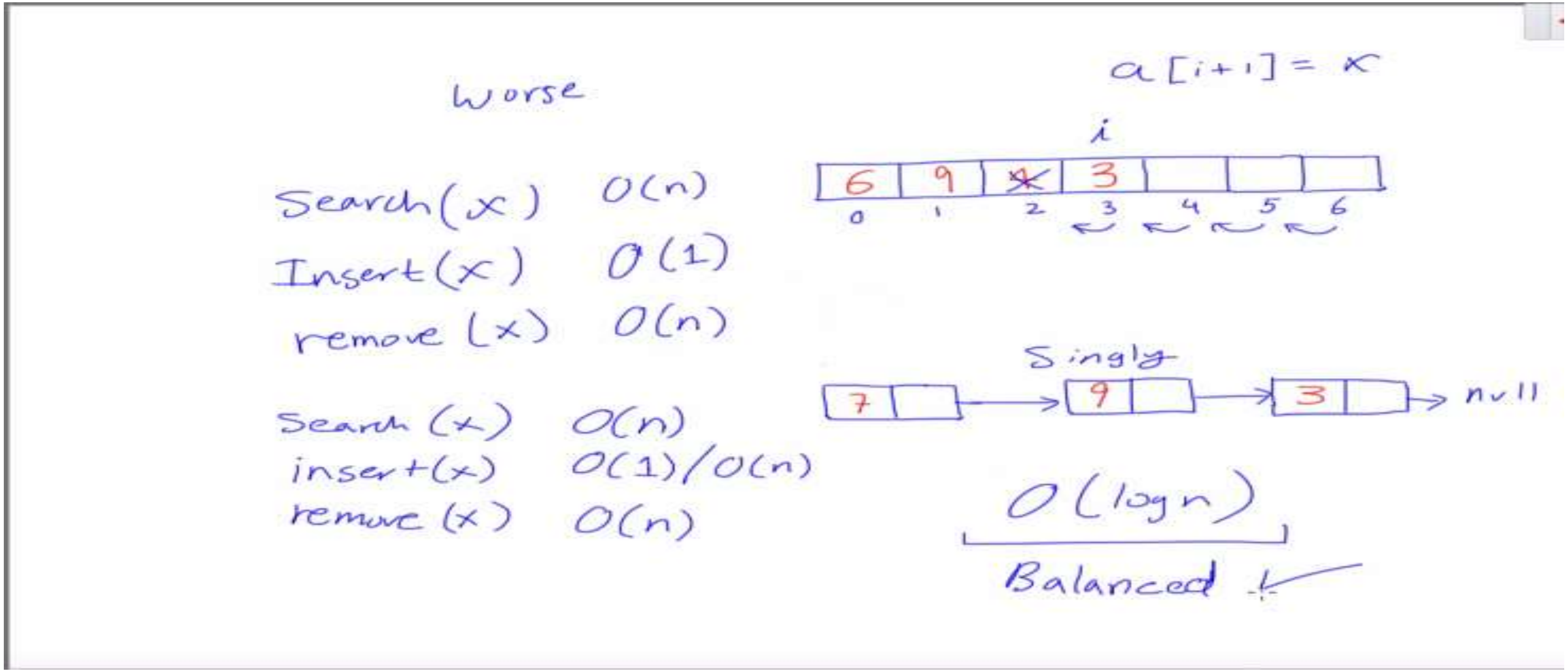

Binary search tree
49
96
53
28
62
10
29
57
100
49
97


Binary search tree
49
96
37
53
28
62
10
29
57
100
49
97
96


Binary search tree
49
96
37
Traversing
53
28
62
10
29
57
100
49
97
37
96

Overview



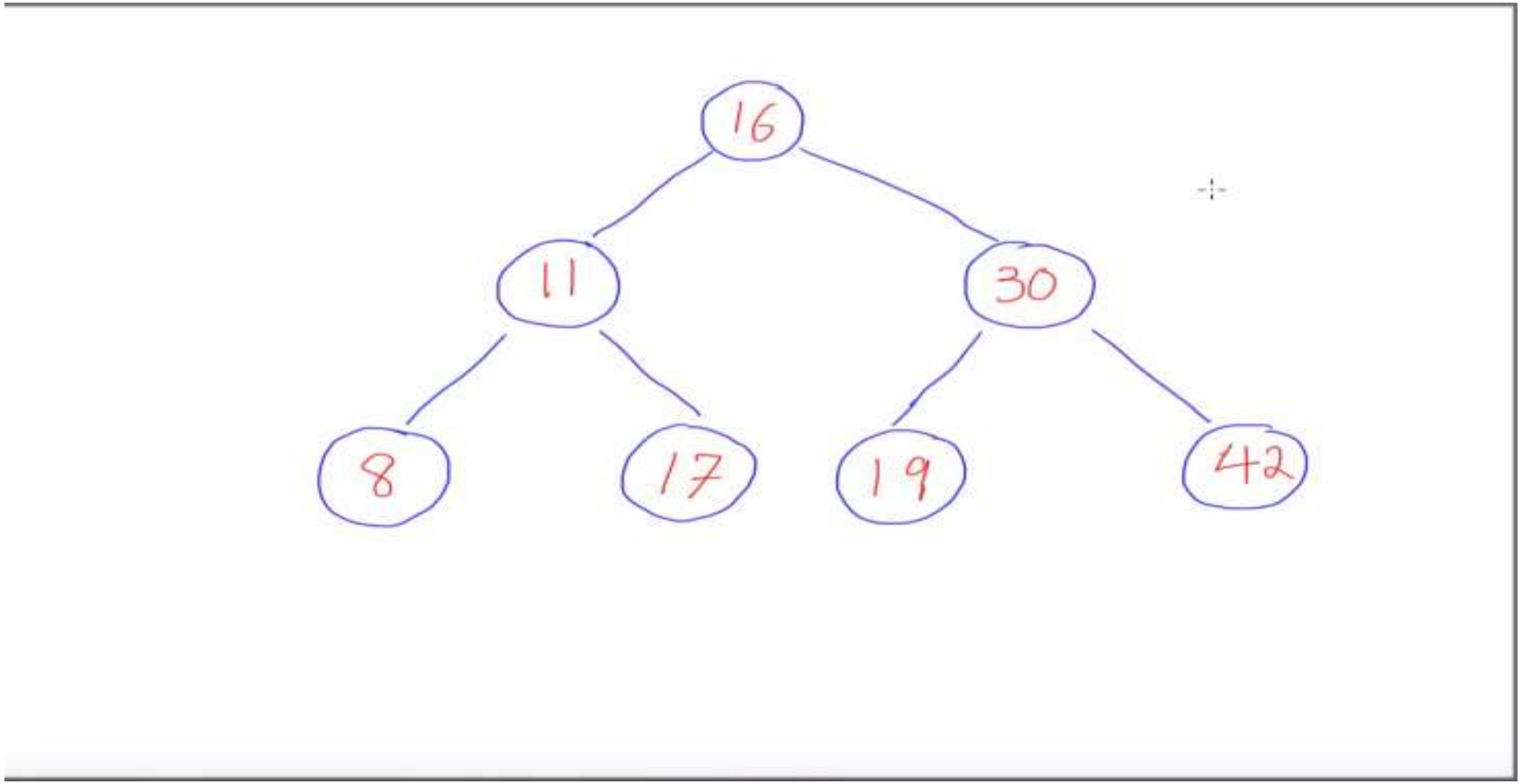


Worse

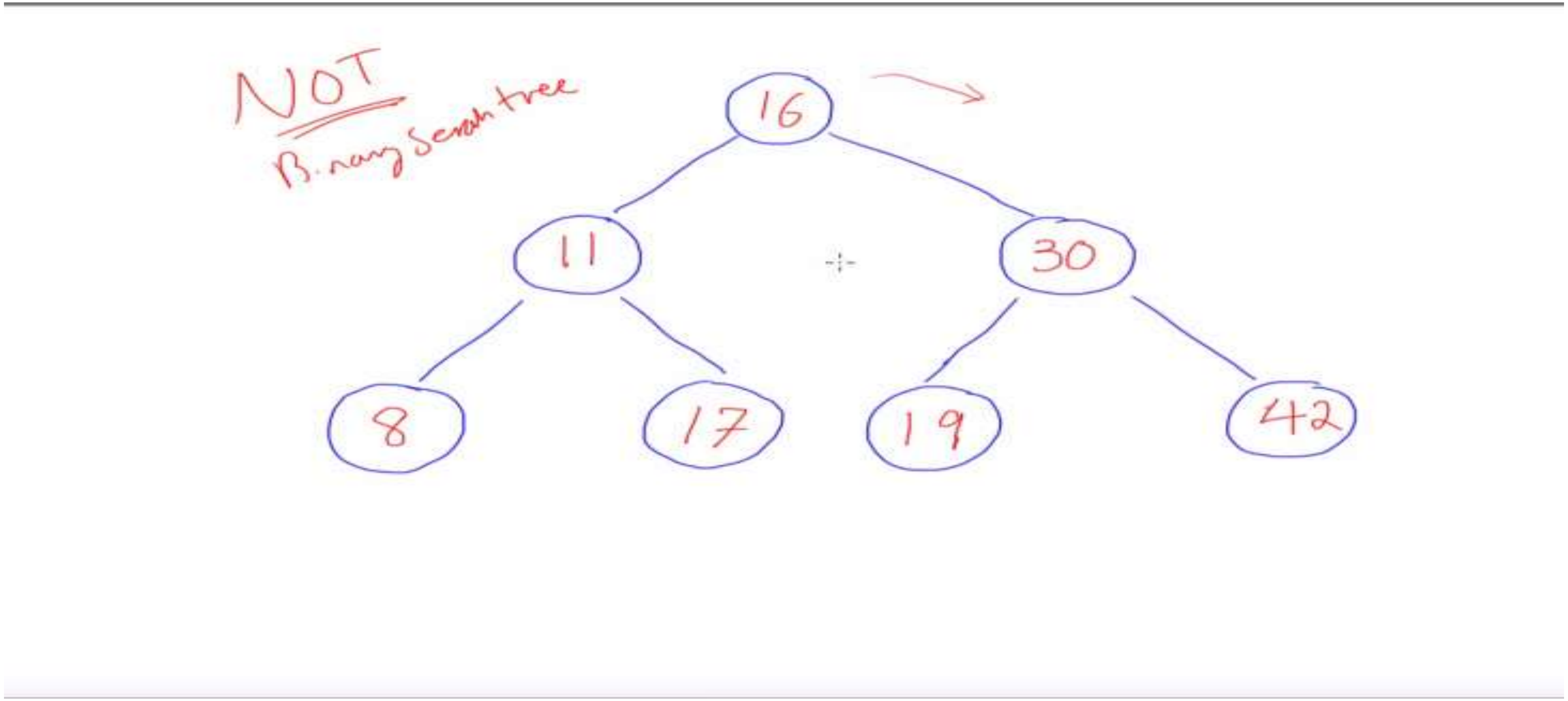
Search(x) O(n)
Insert(x) O(1)
remove (x) O(n)

Search (x) O(n)
insert(x) O(1)/O(n)
remove (x) O(n)

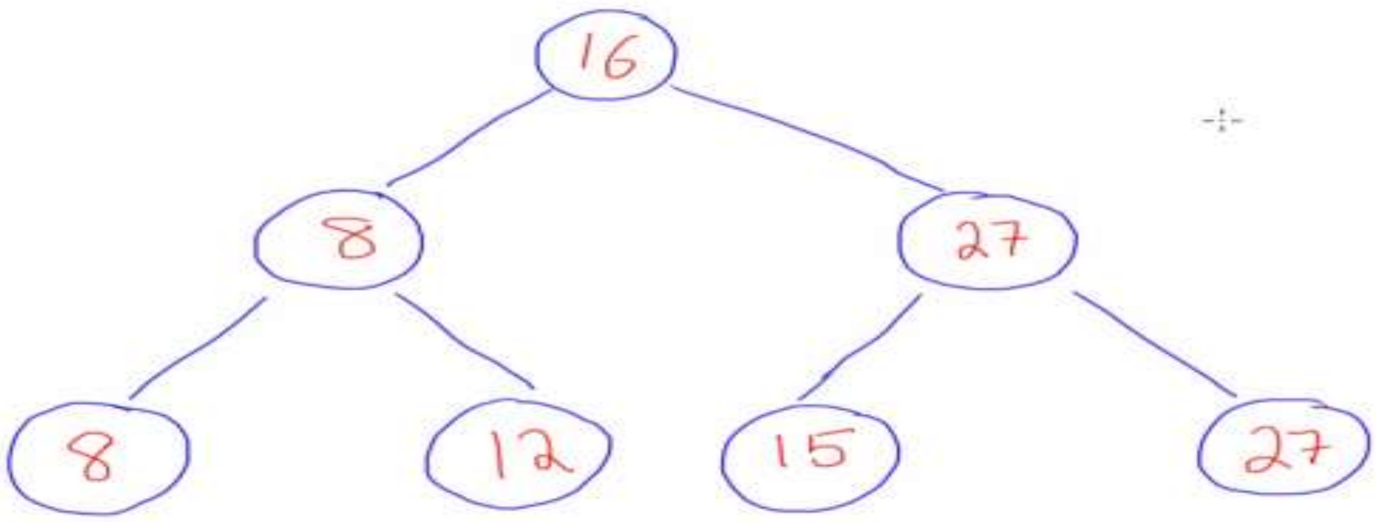
Q) Is it a Binary Search Tree ?



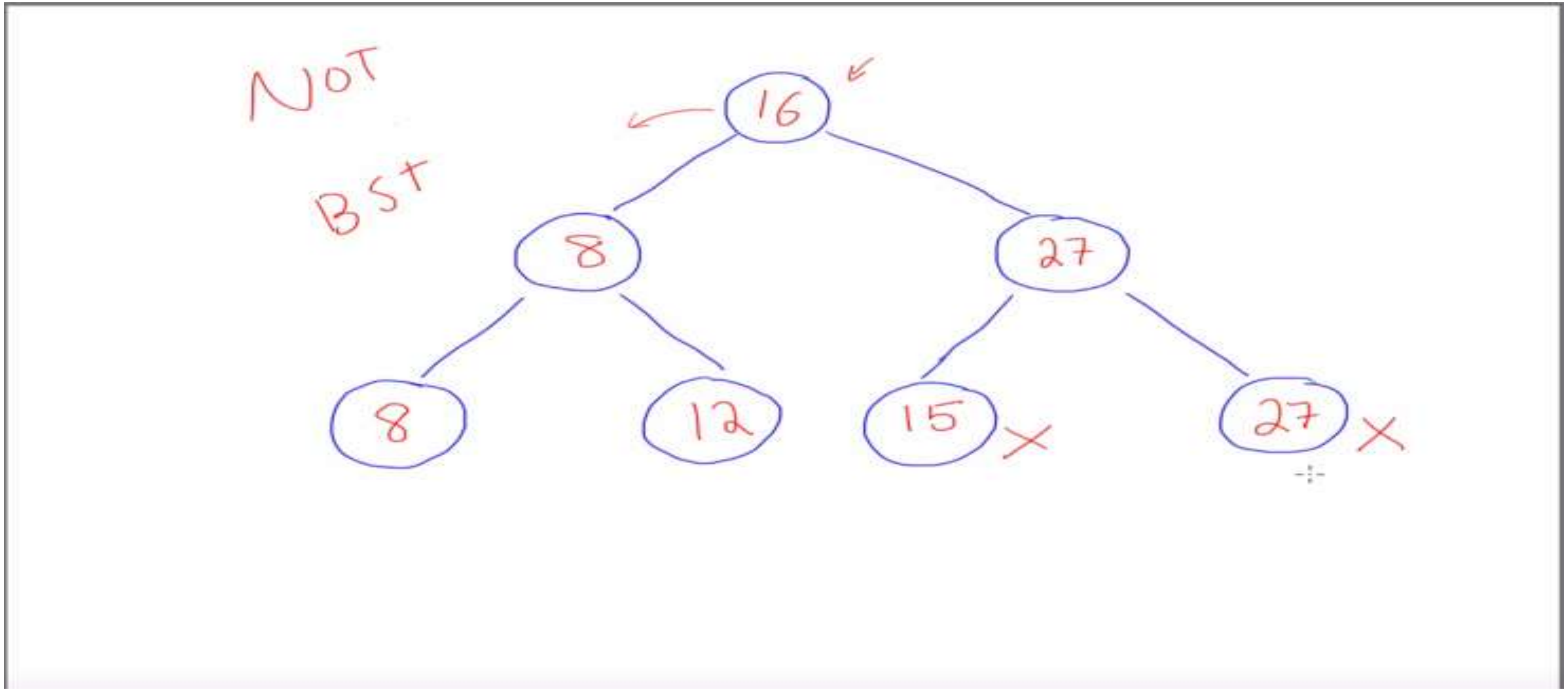
Answer



Q) Is it a Binary Search Tree ?



Answer

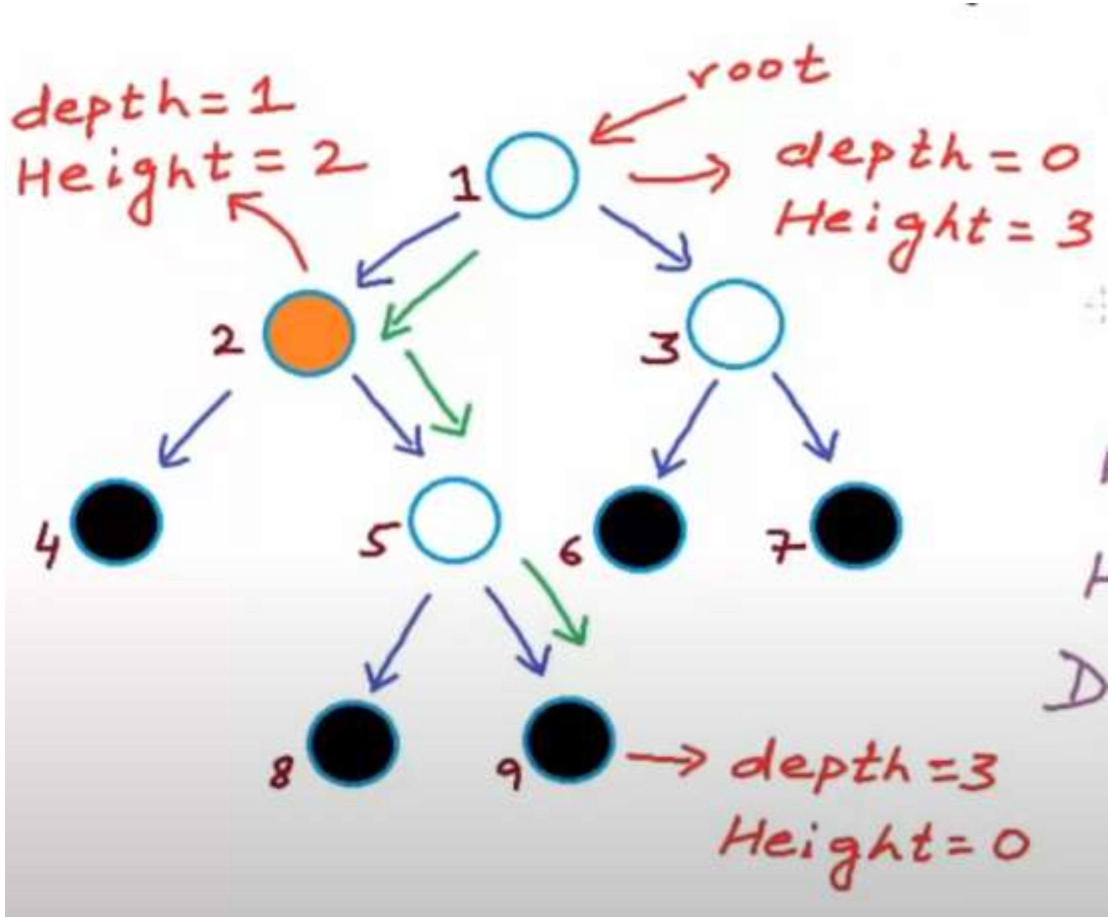


Balanced Binary Search Tree

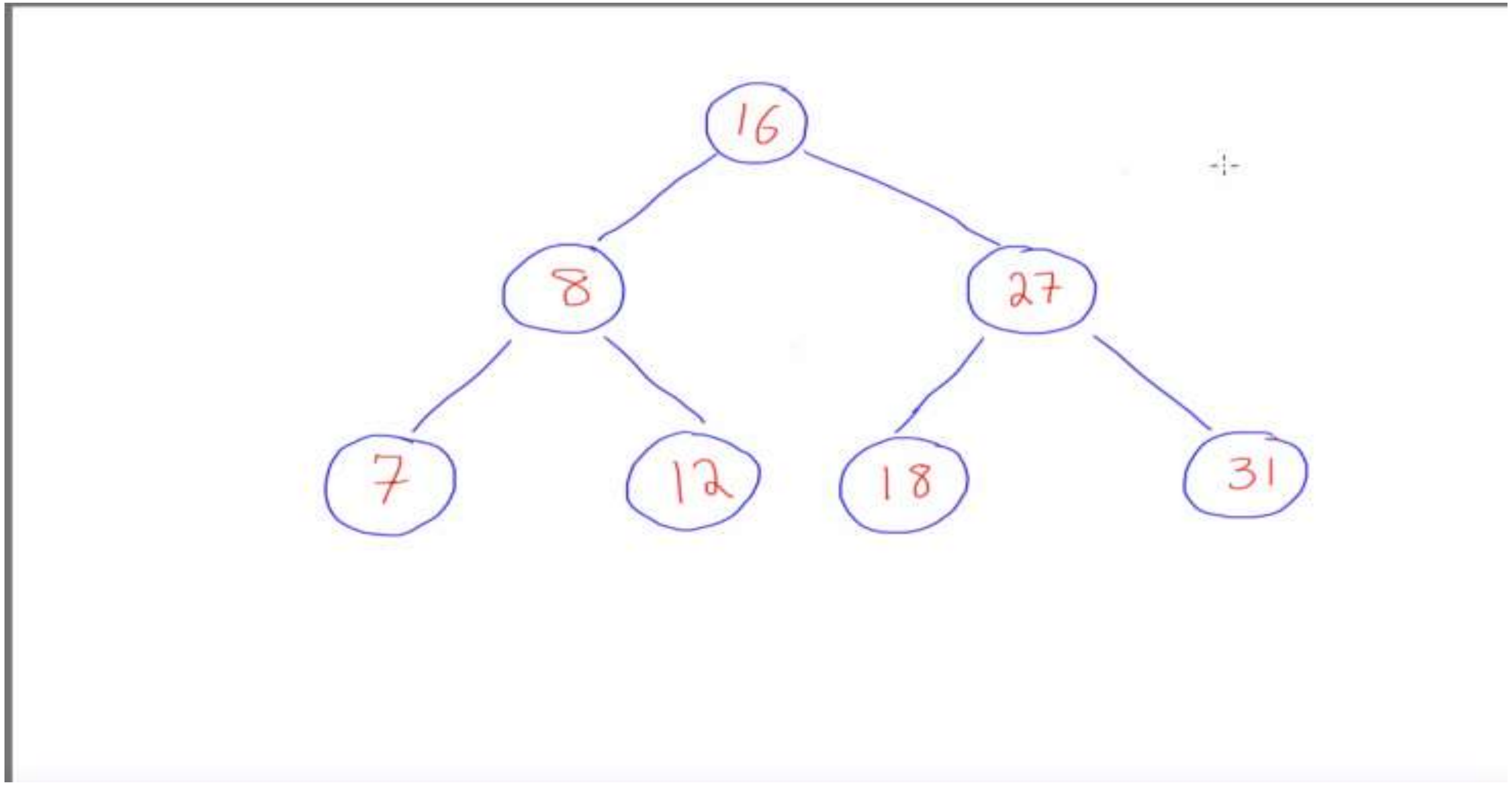
- For all the nodes , the difference between the heights of the left and right subtrees must not be greater than one.

- Height of Binary Search Tree can be calculated by counting number of edges in longest path from the root to the leaf node.
  *Height of Tree = Height of Root
  *Height of Tree with 1 Node (Leaf Node) = 0

- Depth of a Node = No of edges in path from root to that Node.


root
depth=0
Height=3
depth=1
Height=2
1
2
3
4
5
6
7
8
9
depth=3
Height=0

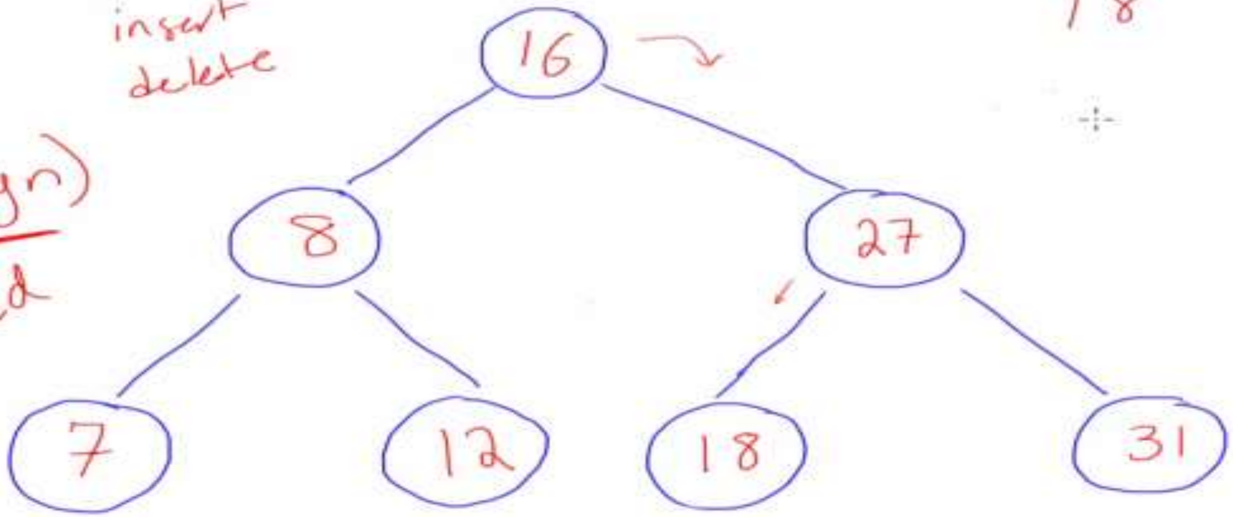
Balanced Binary Search Tree



insert
delete

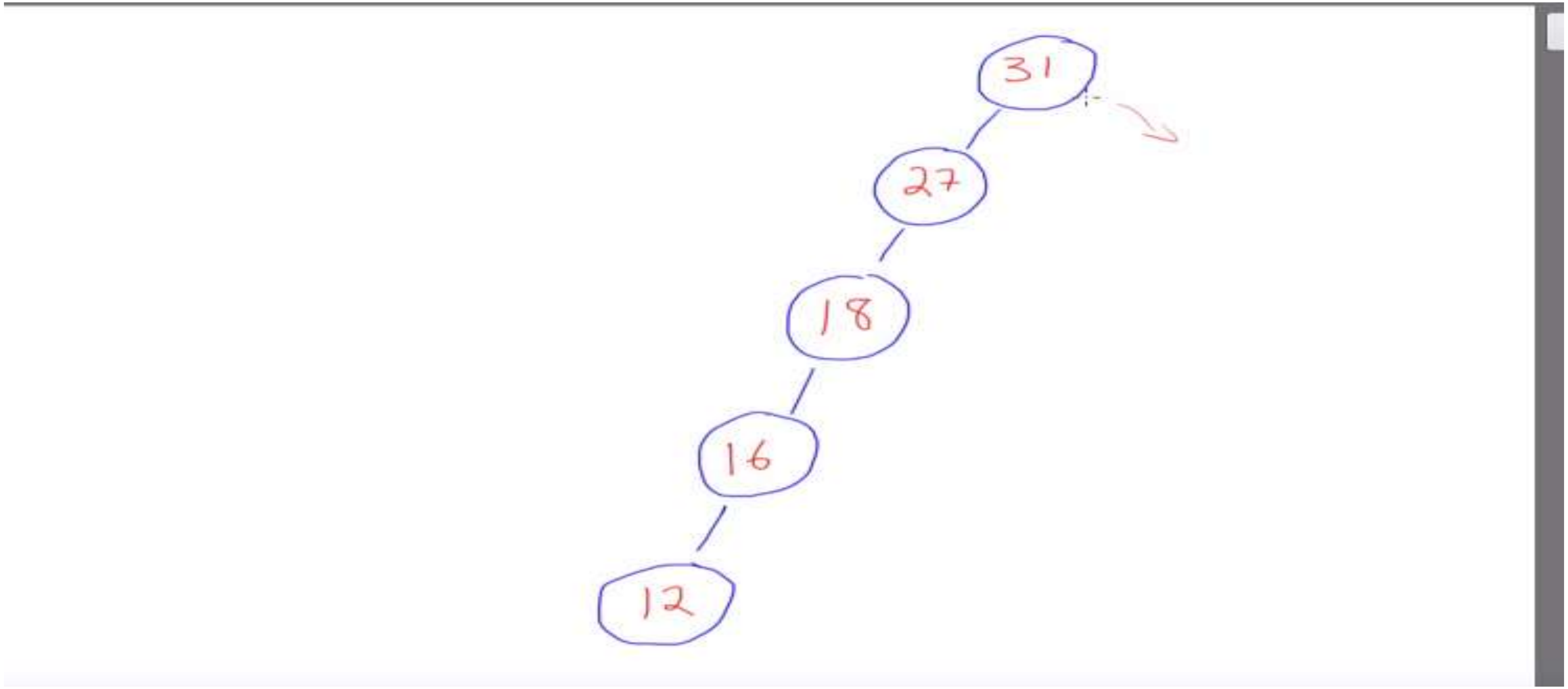
O(log n)

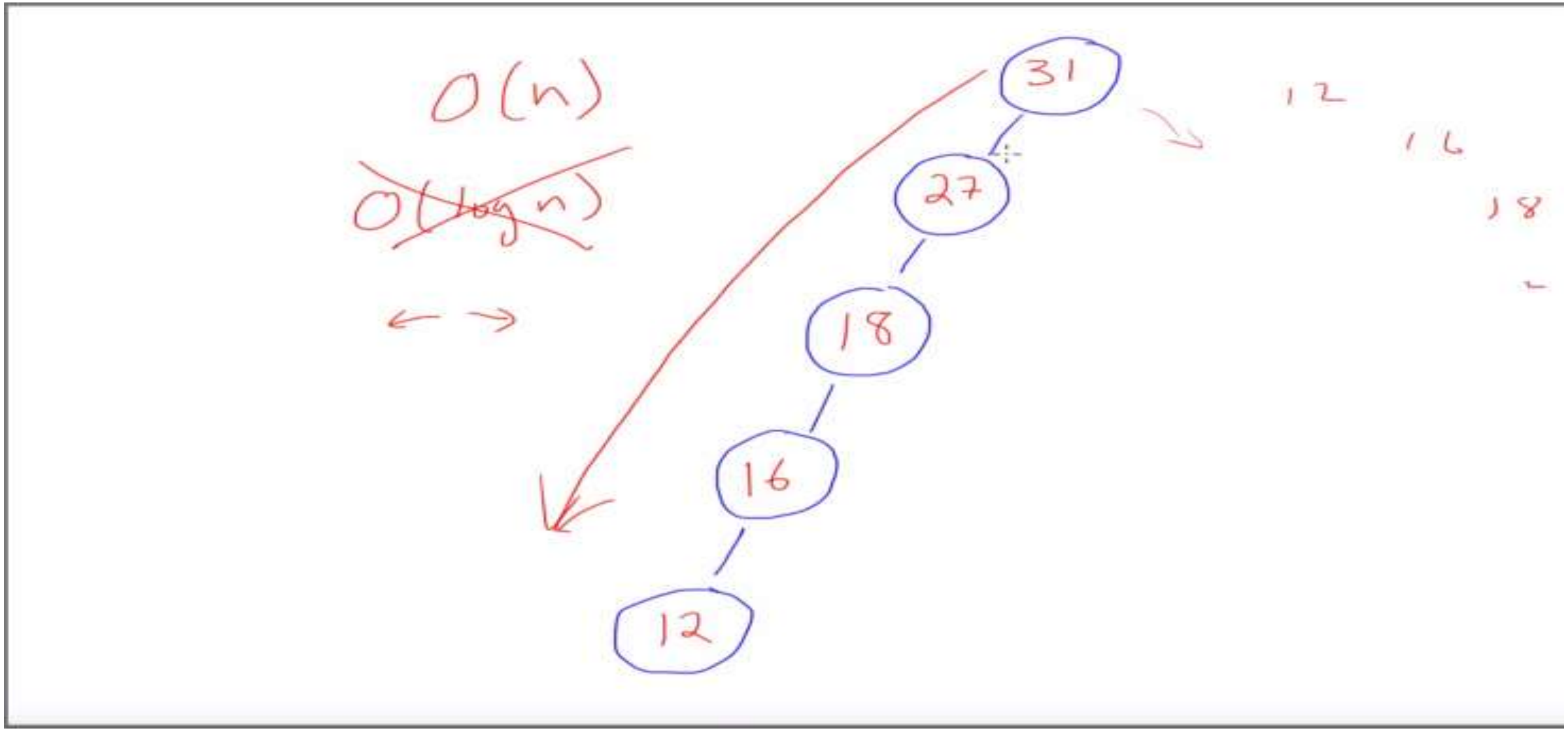
Balanced



18

Unbalanced Binary Tree

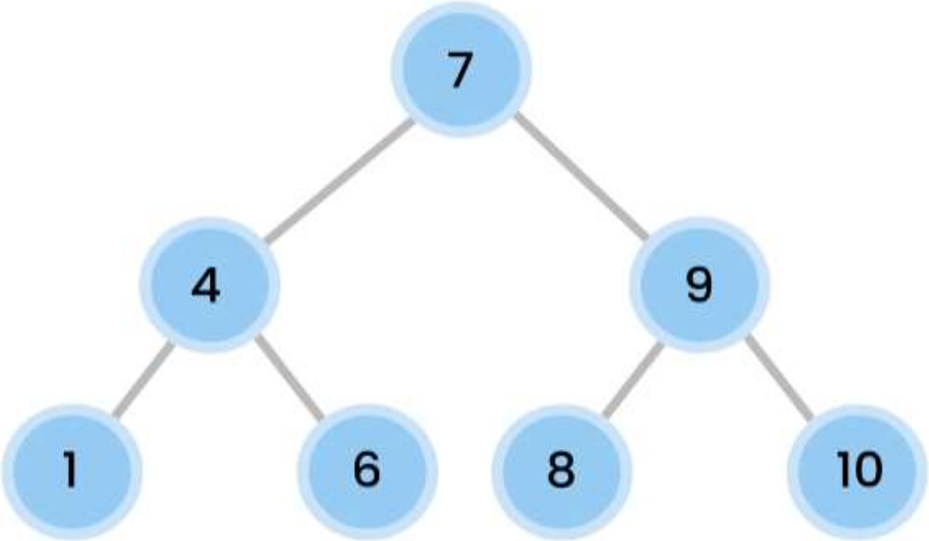


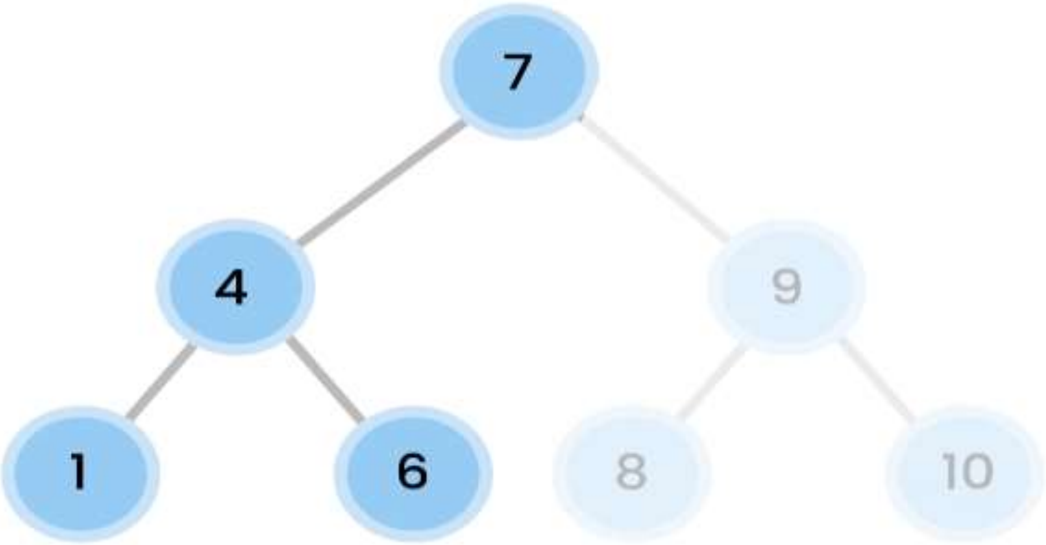


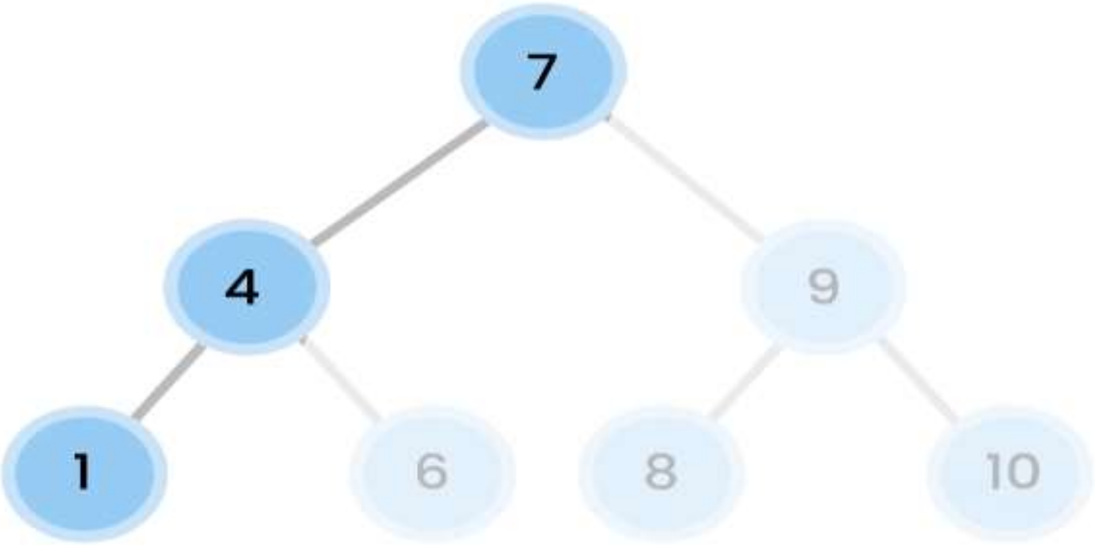
O(n)

~~O(log n)~~

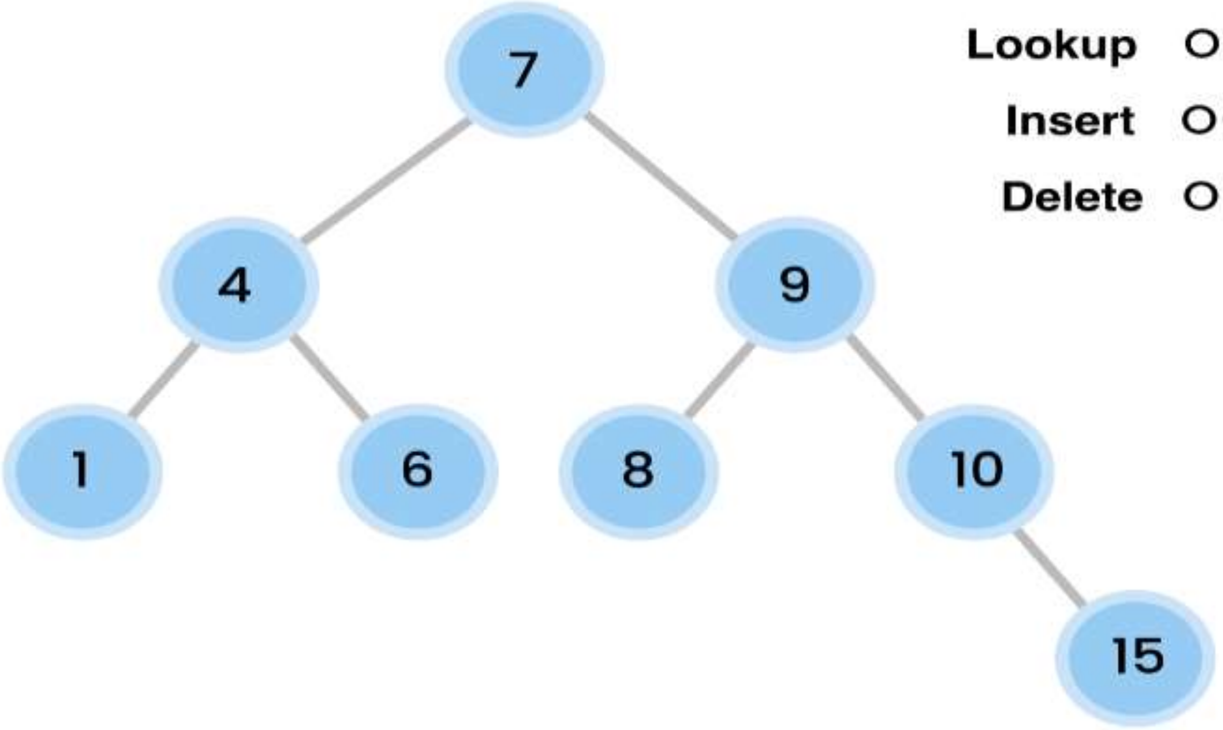
Searching 1




7
4
9
1
6
8
10

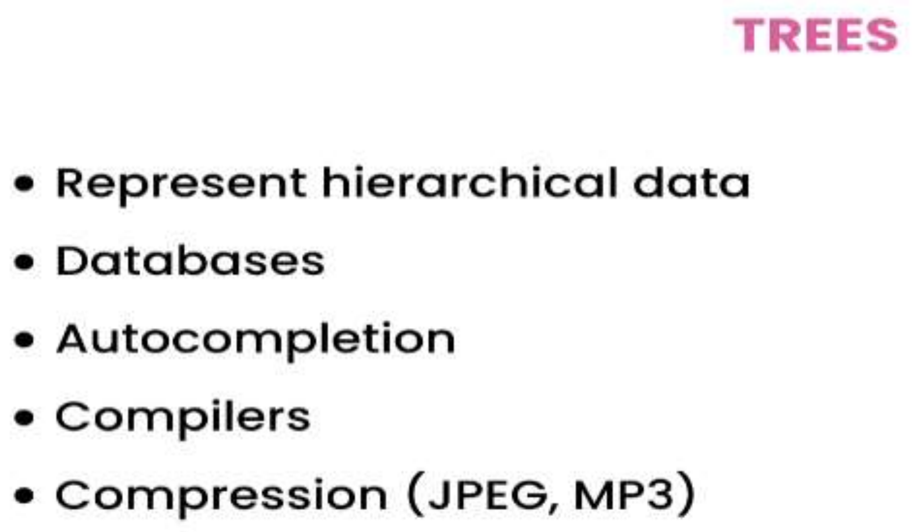

7
4
9
1
6
8
10

Runtime Complexity BST



Lookup	O(log n)
Insert	O(log n)
Delete	O(log n)

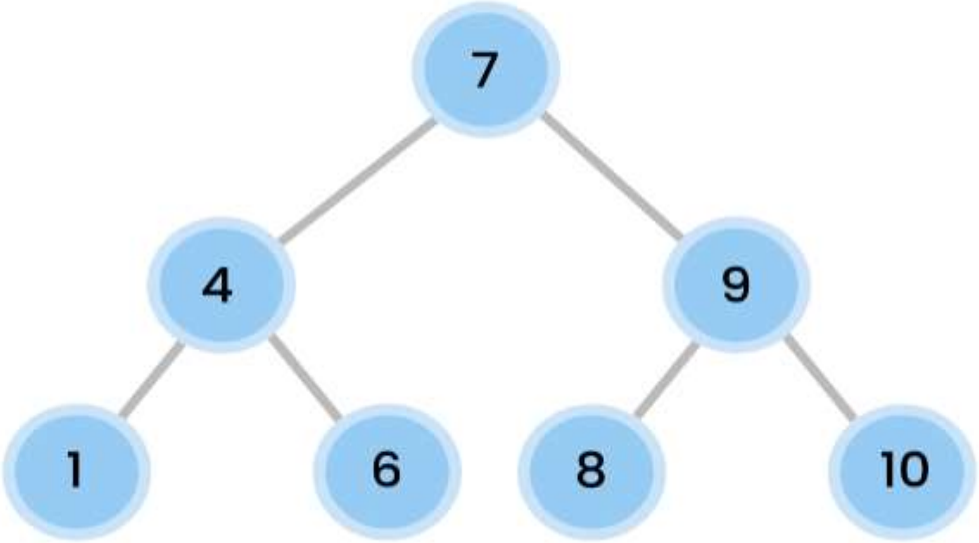
Applications

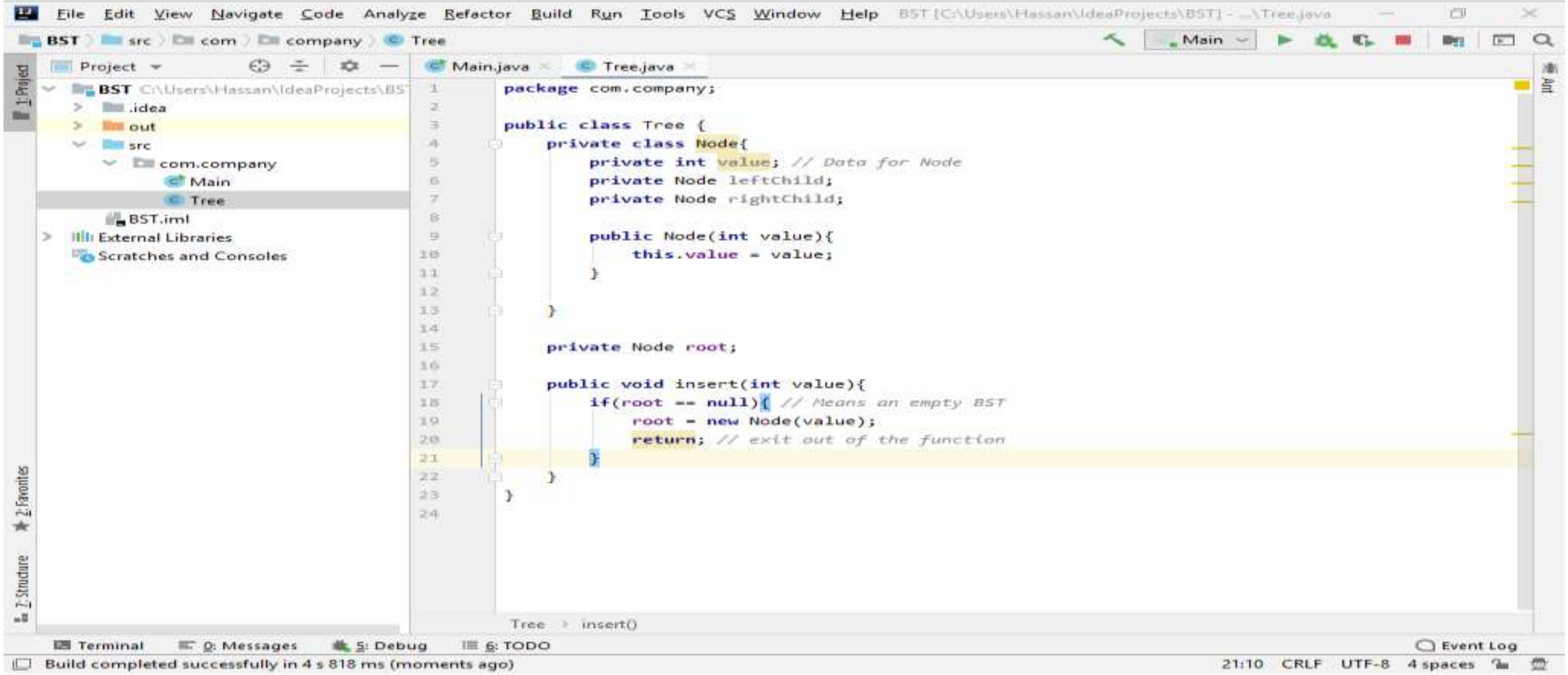


TREES

- Represent hierarchical data
- Databases
- Autocompletion
- Compilers
- Compression (JPEG, MP3)

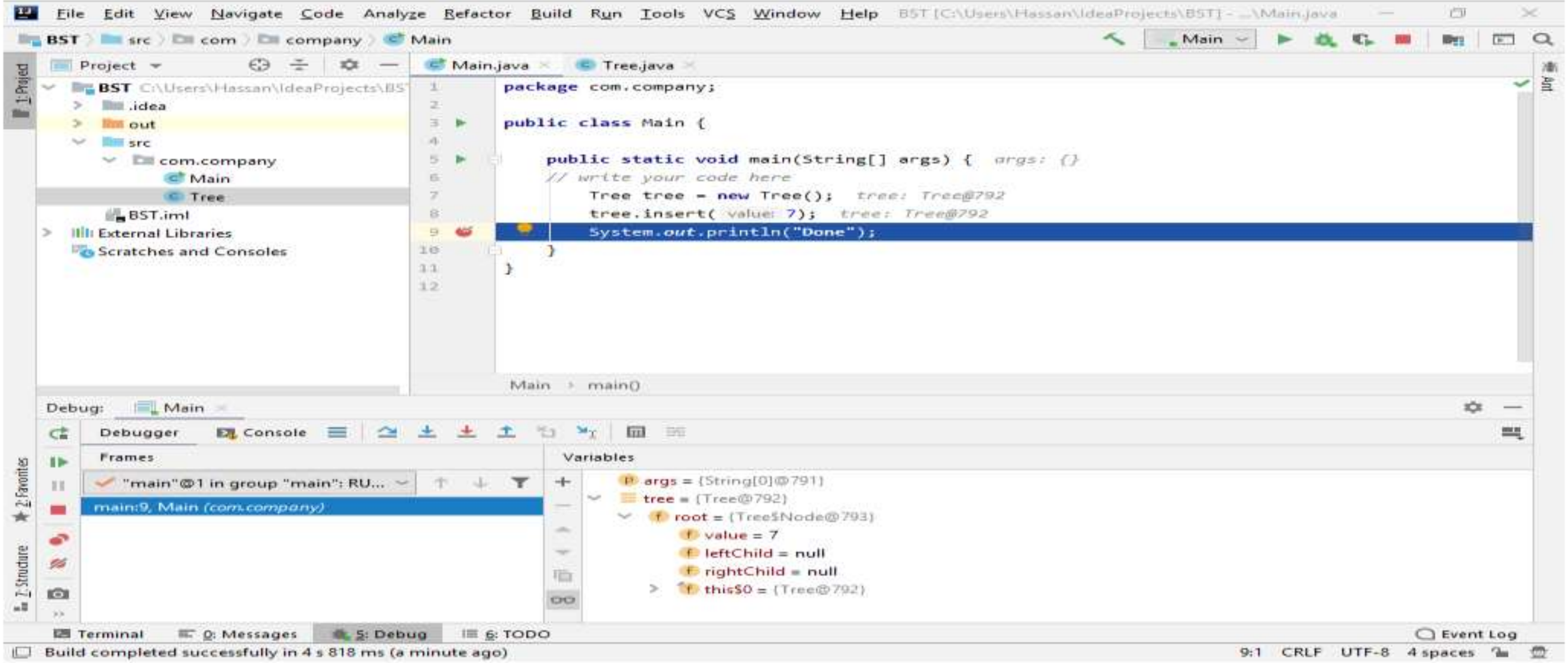
Implementation of Following BST.





```java
package com.company;

public class Tree {
    private class Node{
        private int value; // Data for Node
        private Node leftChild;
        private Node rightChild;

        public Node(int value){
            this.value = value;
        }

    }

    private Node root;

    public void insert(int value){
        if(root == null){ // Means an empty BST
            root = new Node(value);
            return; // exit out of the function
        }
    }
}
```

Build completed successfully in 4 s 818 ms (moments ago)

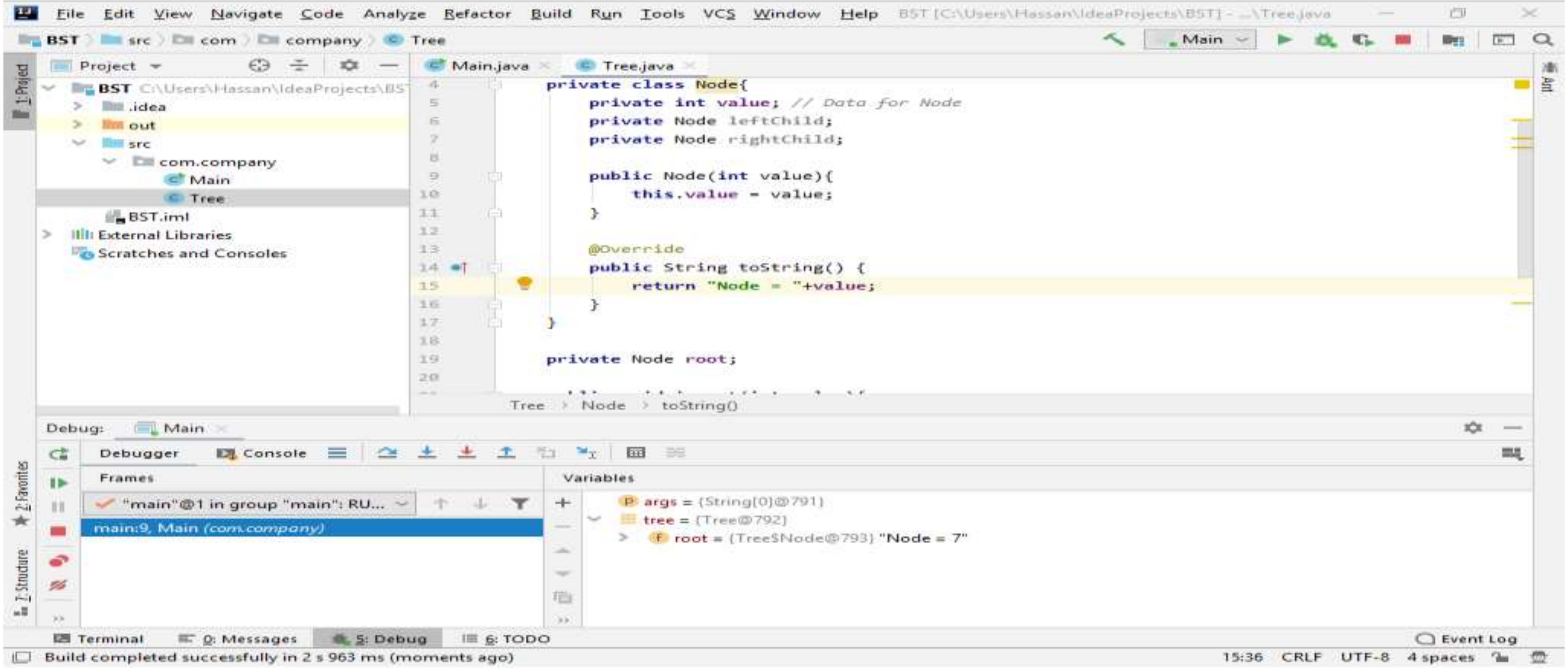


```
package com.company;

public class Main {

    public static void main(String[] args) {  args: {}
    // write your code here
        Tree tree = new Tree();  tree: Tree@792
        tree.insert( value: 7);  tree: Tree@792
        System.out.println("Done");
    }
}
```

```
"main"@1 in group "main": RU...
main:9, Main (com.company)
```

```
args = {String[0]@791}
tree = {Tree@792}
    root = {Tree$Node@793}
        value = 7
        leftChild = null
        rightChild = null
        this$0 = {Tree@792}
```

```
Build completed successfully in 4 s 818 ms (a minute ago)        9:1  CRLF  UTF-8  4 spaces
```



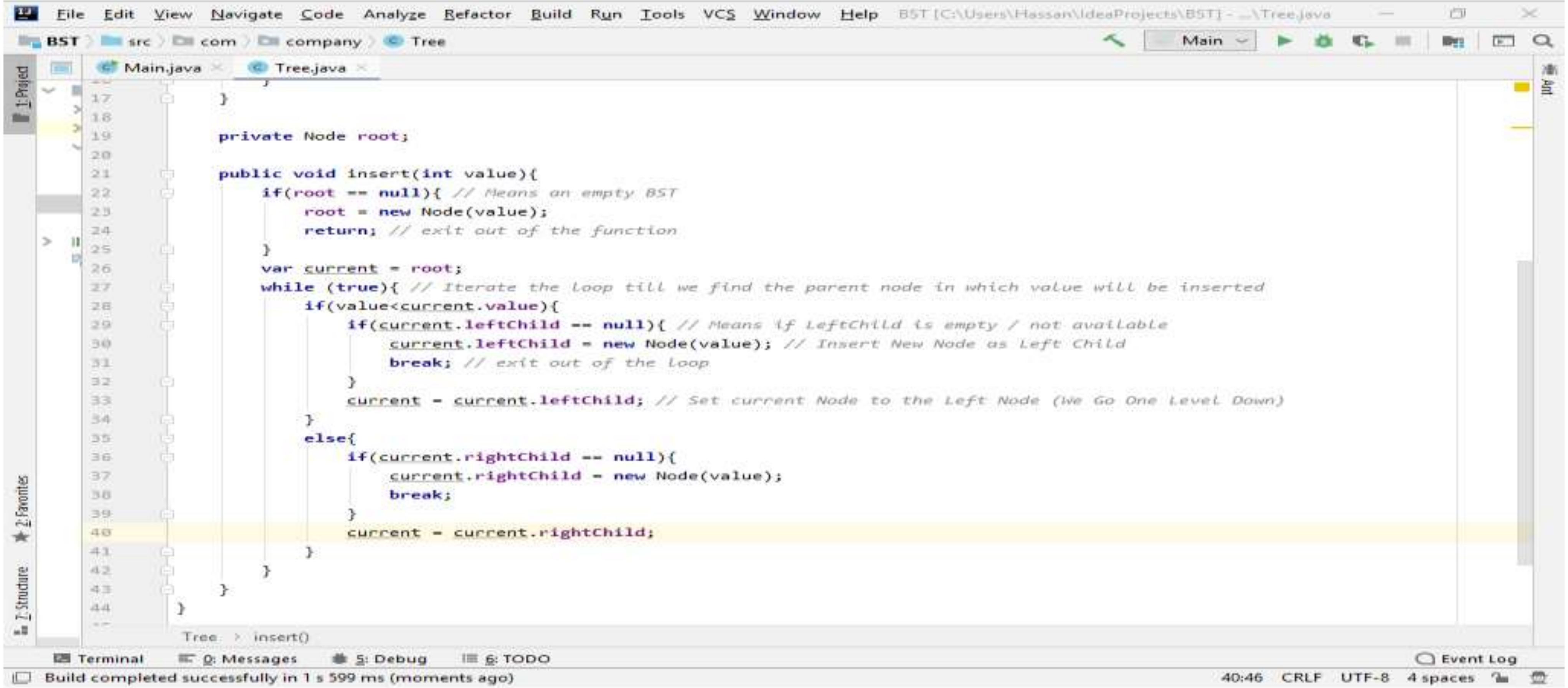
```java
    private class Node{
        private int value; // Data for Node
        private Node leftChild;
        private Node rightChild;

        public Node(int value){
            this.value = value;
        }

        @Override
        public String toString() {
            return "Node = "+value;
        }
    }

    private Node root;
```

```
"main"@1 in group "main": RU...
main:9, Main (com.company)
```

```
args = {String[0]@791}
tree = {Tree@792}
    root = {Tree$Node@793} "Node = 7"
```

Build completed successfully in 2 s 963 ms (moments ago)

Insert Function

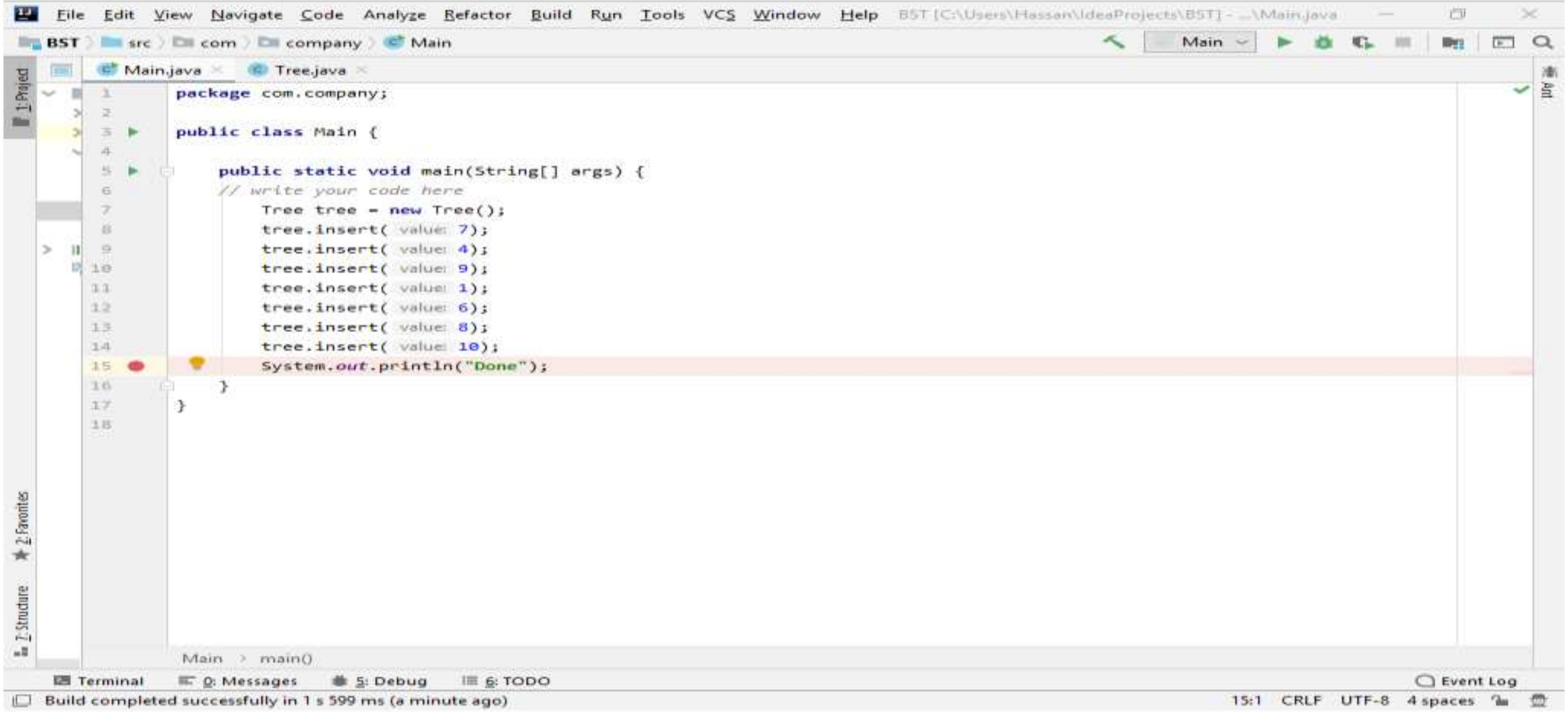


```java
    }

    private Node root;

    public void insert(int value){
        if(root == null){ // Means an empty BST
            root = new Node(value);
            return; // exit out of the function
        }
        var current = root;
        while (true){ // Iterate the Loop till we find the parent node in which value will be inserted
            if(value<current.value){
                if(current.leftChild == null){ // Means if LeftChild is empty / not available
                    current.leftChild = new Node(value); // Insert New Node as Left Child
                    break; // exit out of the Loop
                }
                current = current.leftChild; // Set current Node to the Left Node (We Go One Level Down)
            }
            else{
                if(current.rightChild == null){
                    current.rightChild = new Node(value);
                    break;
                }
                current = current.rightChild;
            }
        }
    }
}
```

Build completed successfully in 1 s 599 ms (moments ago)

40:46 CRLF UTF-8 4 spaces

Test

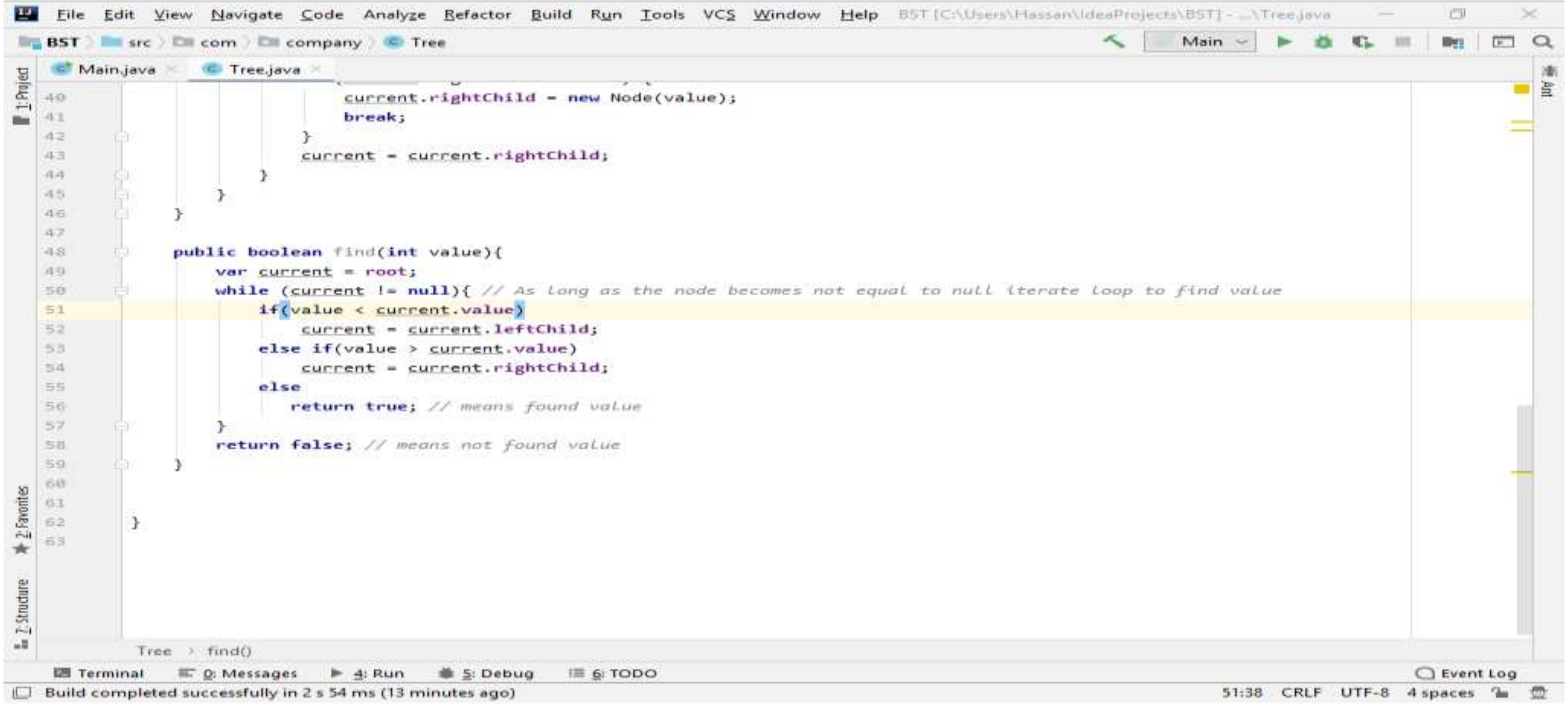


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
BST [C:\Users\Hassan\IdeaProjects\BST] - ...\Main.java
BST > src > com > company > Main
Main
Main.java
Tree.java
1: Project
2: Favorites
7: Structure
Ant
Main > main()
Terminal
0: Messages
5: Debug
6: TODO
Event Log
15:1 CRLF UTF-8 4 spaces

```java
package com.company;

public class Main {

    public static void main(String[] args) {
    // write your code here
        Tree tree = new Tree();
        tree.insert( value: 7);
        tree.insert( value: 4);
        tree.insert( value: 9);
        tree.insert( value: 1);
        tree.insert( value: 6);
        tree.insert( value: 8);
        tree.insert( value: 10);
        System.out.println("Done");
    }
}
```

Build completed successfully in 1 s 599 ms (a minute ago)

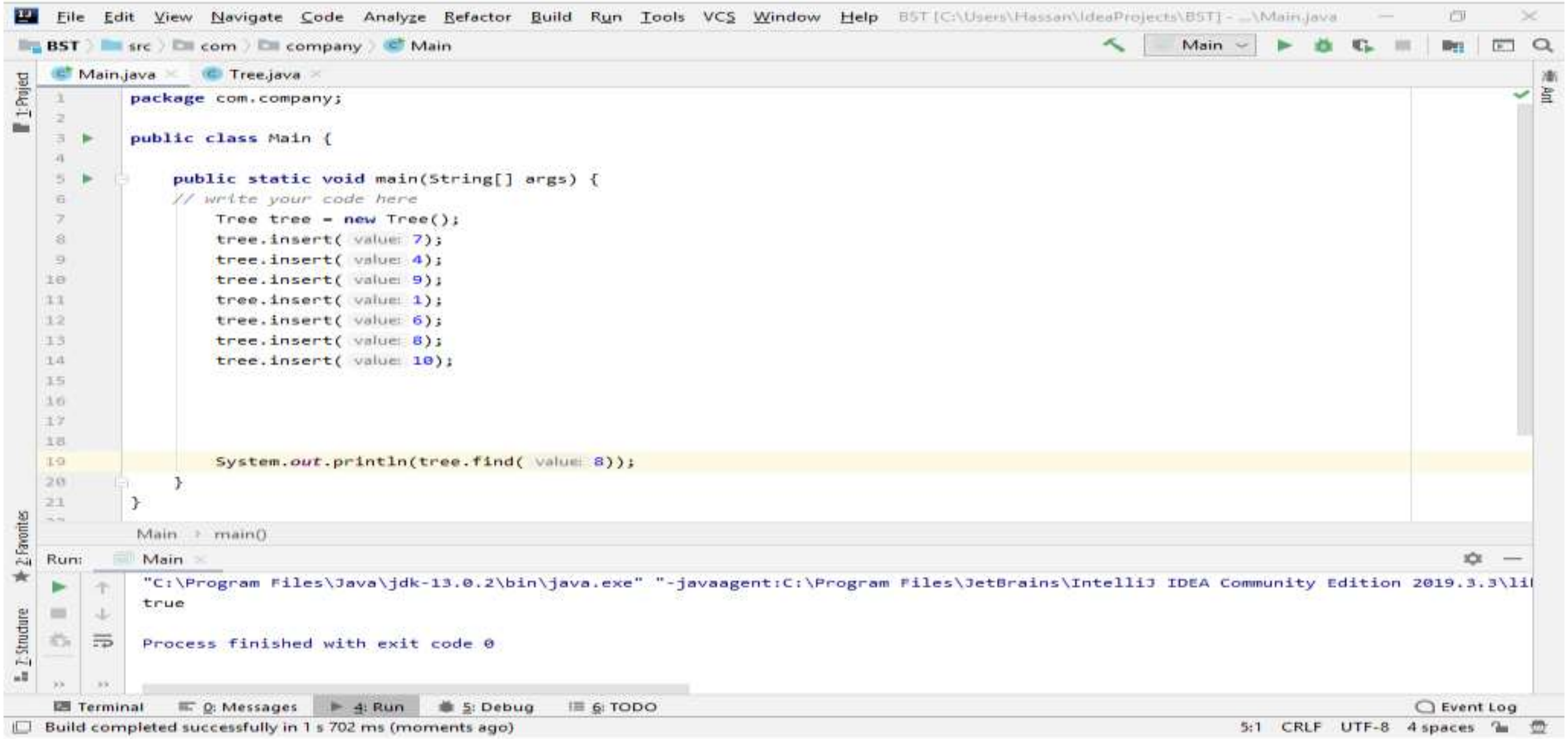
Find function


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
BST [C:\Users\Hassan\IdeaProjects\BST] - ...\Tree.java
BST › src › com › company › Tree
Main
Main.java
Tree.java
1: Project
2: Favorites
Z: Structure
Ant
Tree › find()
Terminal
0: Messages
4: Run
5: Debug
6: TODO
Event Log
51:38 CRLF UTF-8 4 spaces

```java
                        current.rightChild = new Node(value);
                        break;
                    }
                    current = current.rightChild;
                }
            }
        }

        public boolean find(int value){
            var current = root;
            while (current != null){ // As long as the node becomes not equal to null iterate loop to find value
                if(value < current.value)
                    current = current.leftChild;
                else if(value > current.value)
                    current = current.rightChild;
                else
                    return true; // means found value
            }
            return false; // means not found value
        }


    }
```

Build completed successfully in 2 s 54 ms (13 minutes ago)

Test



```
package com.company;

public class Main {

    public static void main(String[] args) {
    // write your code here
        Tree tree = new Tree();
        tree.insert( value: 7);
        tree.insert( value: 4);
        tree.insert( value: 9);
        tree.insert( value: 1);
        tree.insert( value: 6);
        tree.insert( value: 8);
        tree.insert( value: 10);




        System.out.println(tree.find( value: 8));
    }
}
```

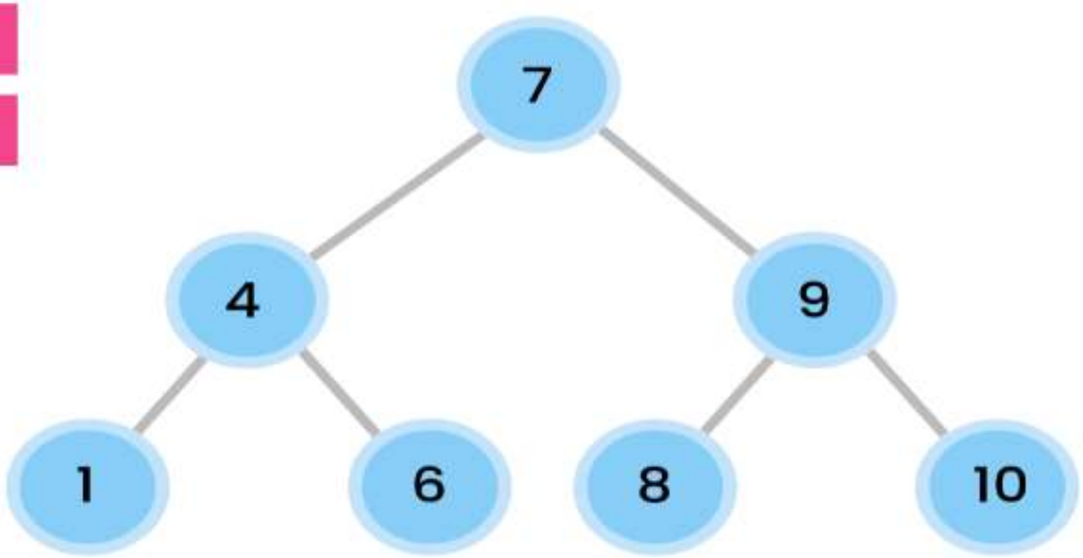
```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\li
true

Process finished with exit code 0
```

Build completed successfully in 1 s 702 ms (moments ago)

Two Types Of Traversal

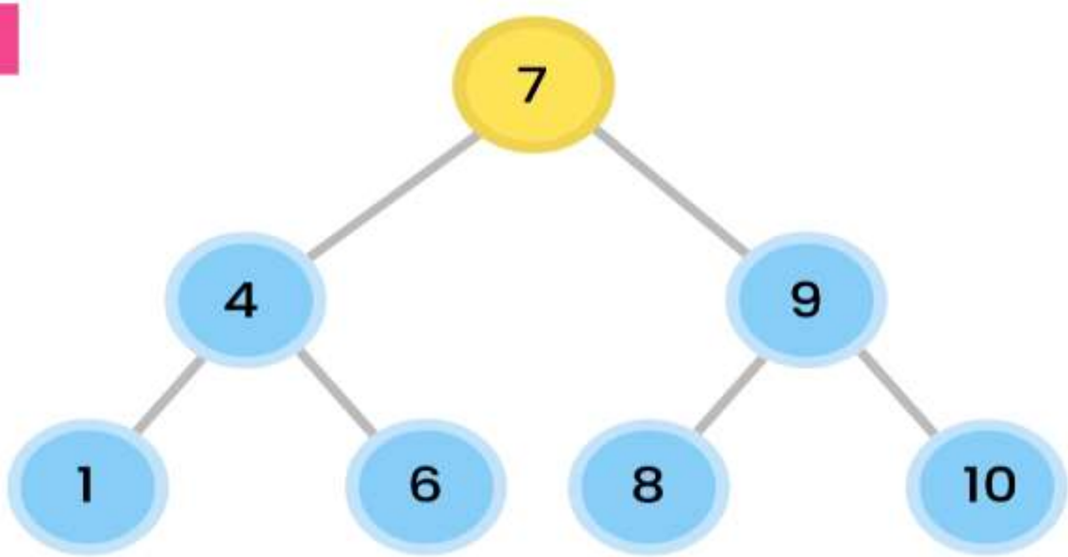
BREADTH FIRST

DEPTH FIRST



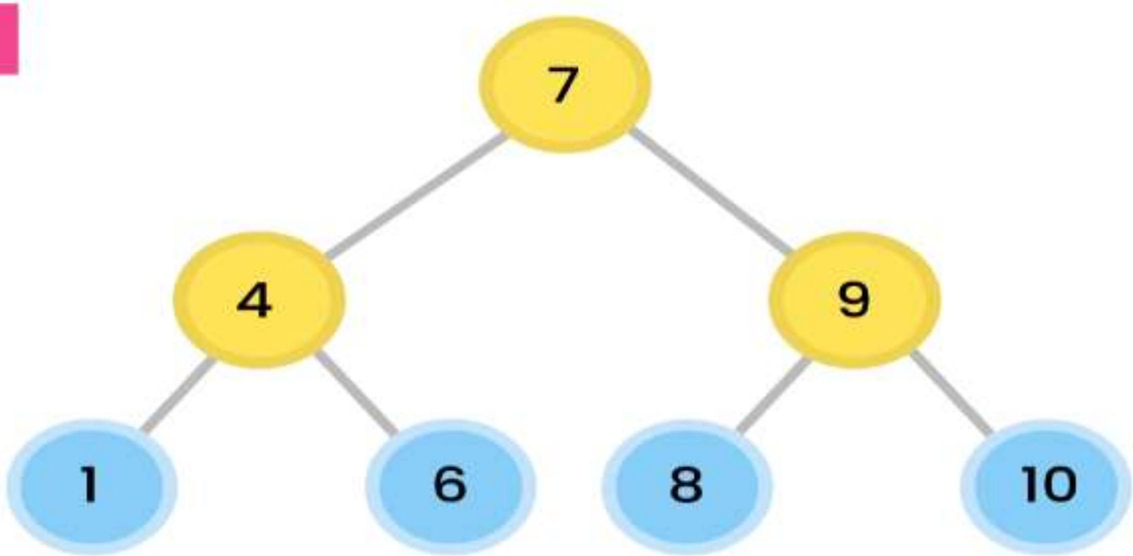
BREADTH FIRST

Level Order



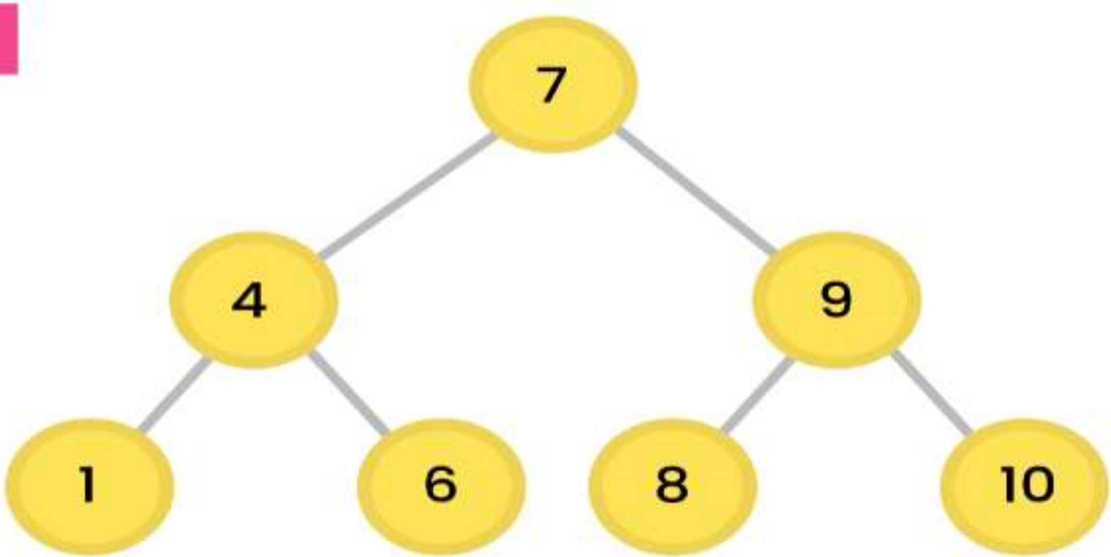
BREADTH FIRST

Level Order



BREADTH FIRST

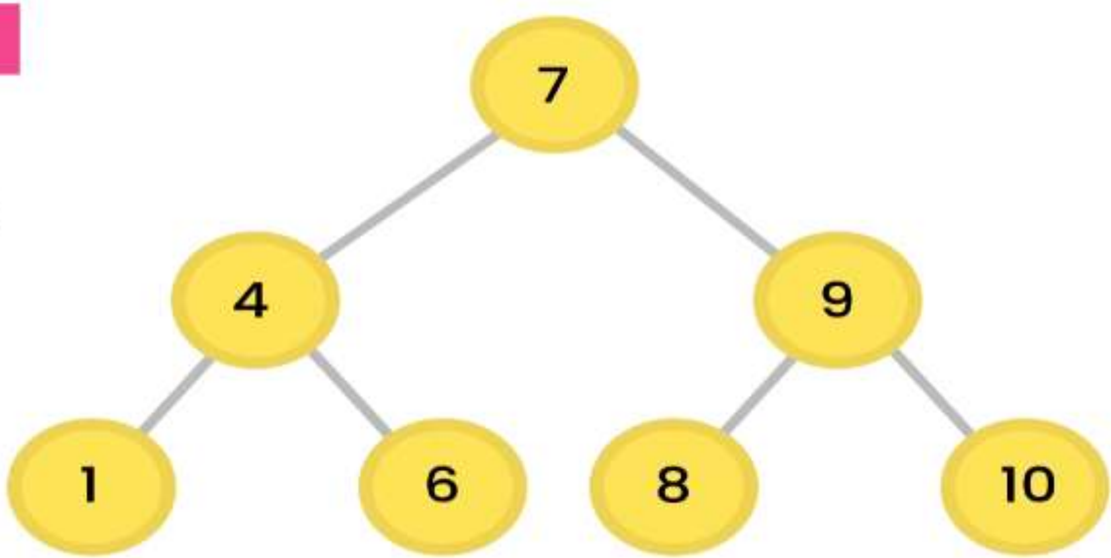
Level Order



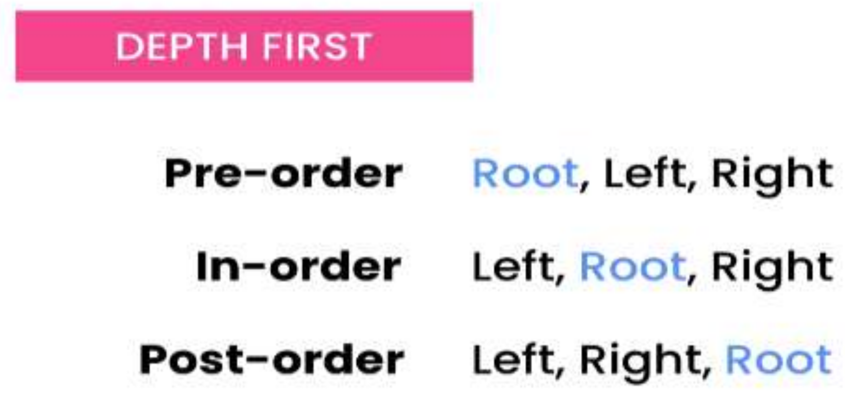
BREADTH FIRST

Level Order

7, 4, 9, 1, 6, 8, 10



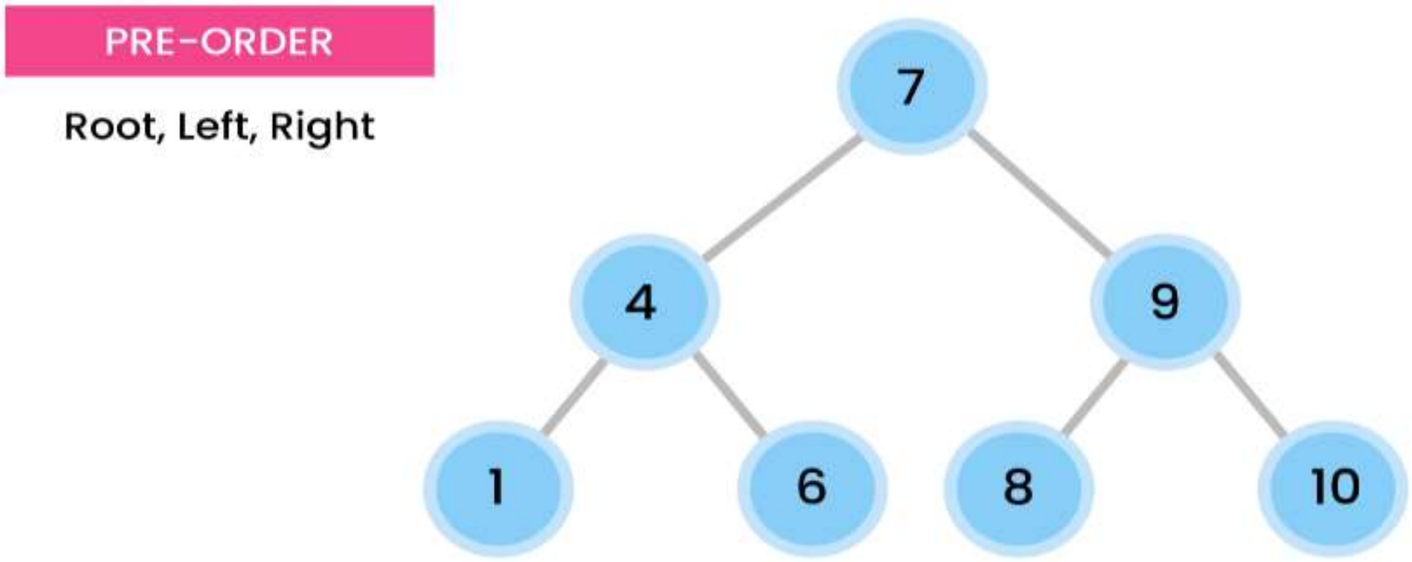
Three Types Of Depth First Traversal



DEPTH FIRST

Pre-order	Root, Left, Right
In-order	Left, Root, Right
Post-order	Left, Right, Root

Pre-Order Traversal

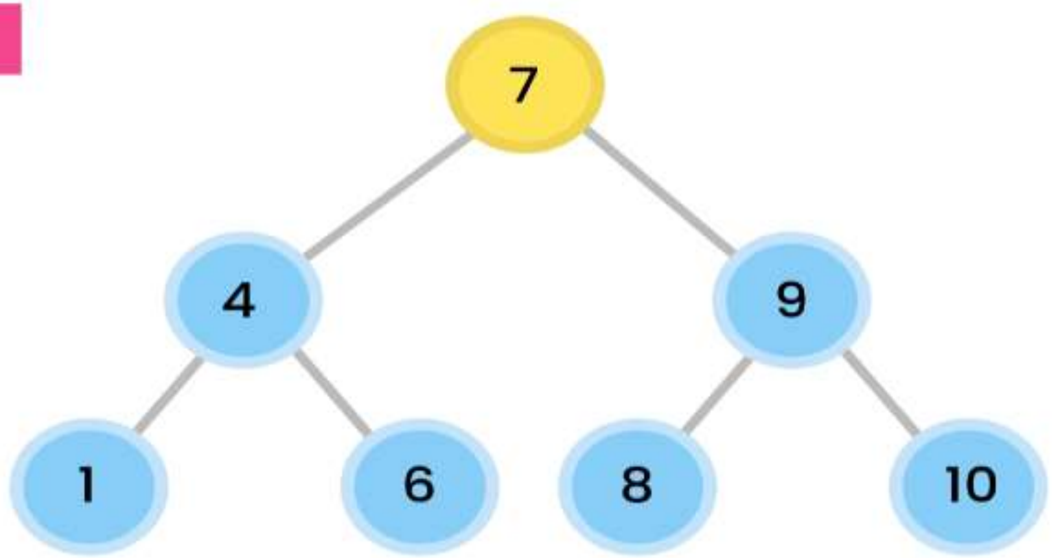


PRE-ORDER

Root, Left, Right

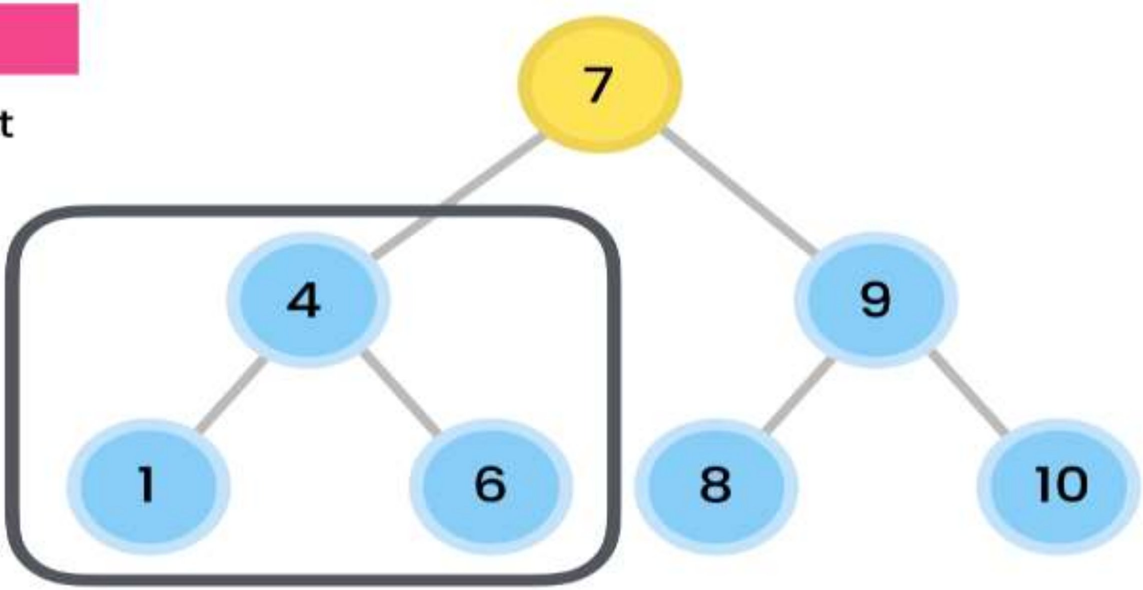
PRE-ORDER

Root, Left, Right



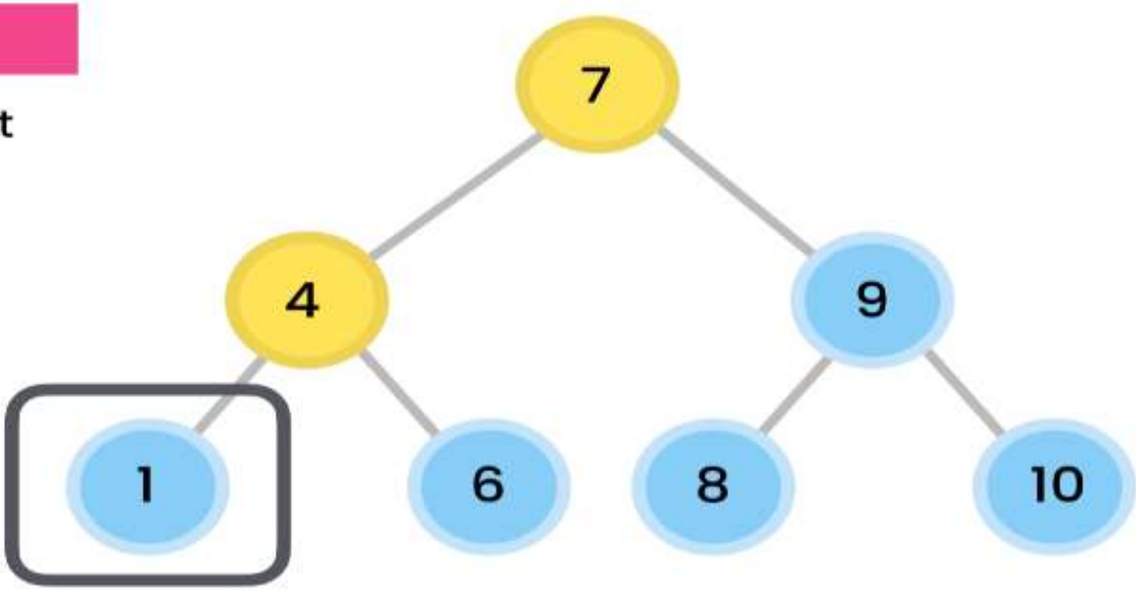
PRE-ORDER

Root, Left, Right



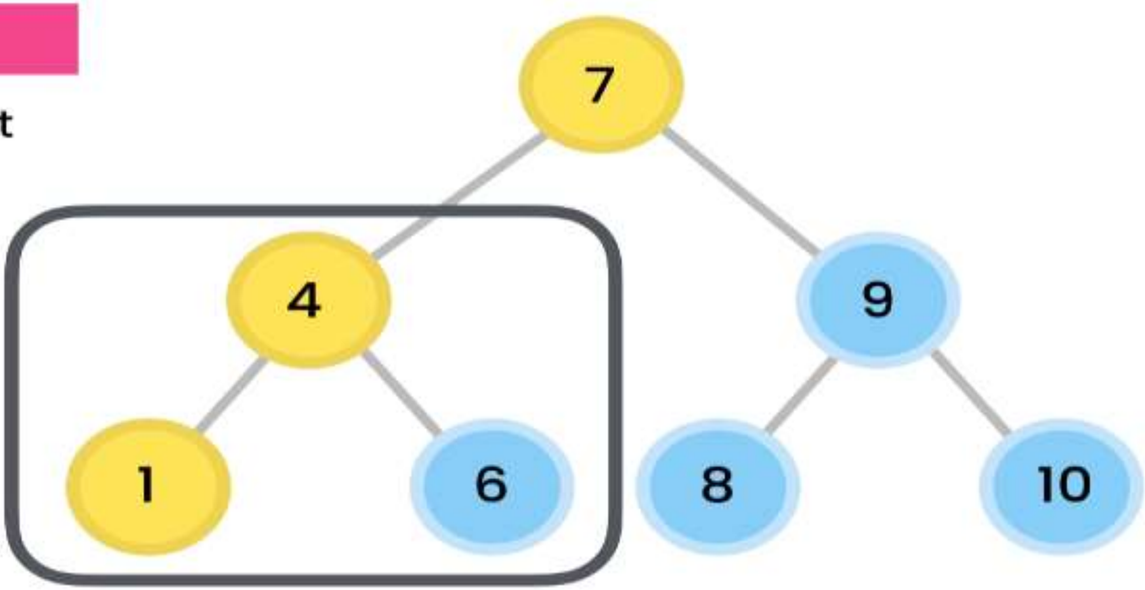
PRE-ORDER

Root, Left, Right



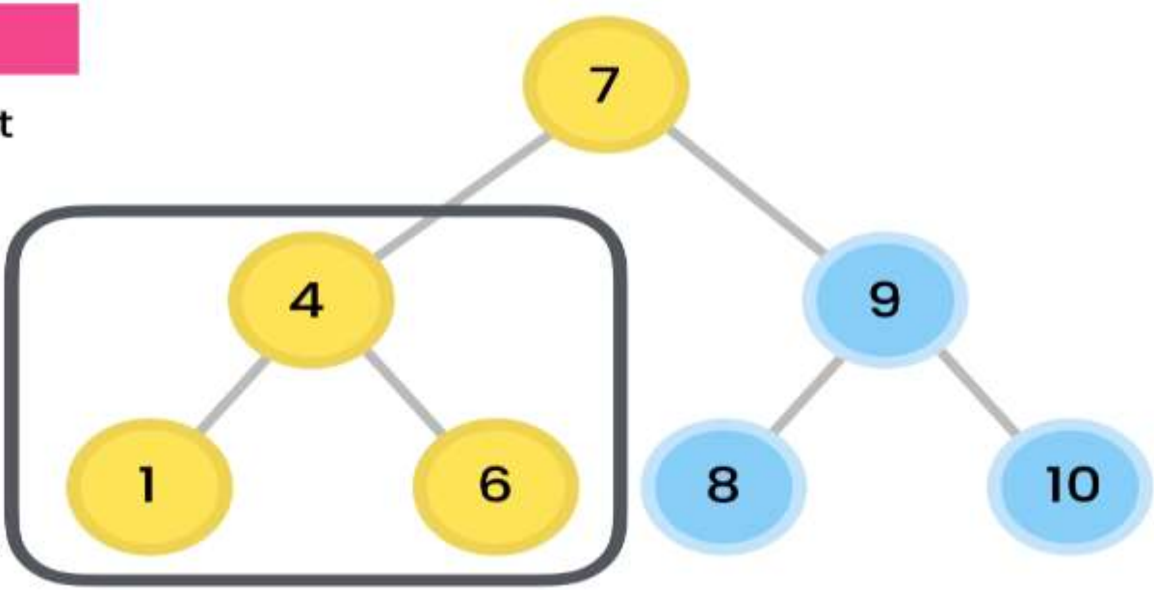
PRE-ORDER

Root, Left, Right



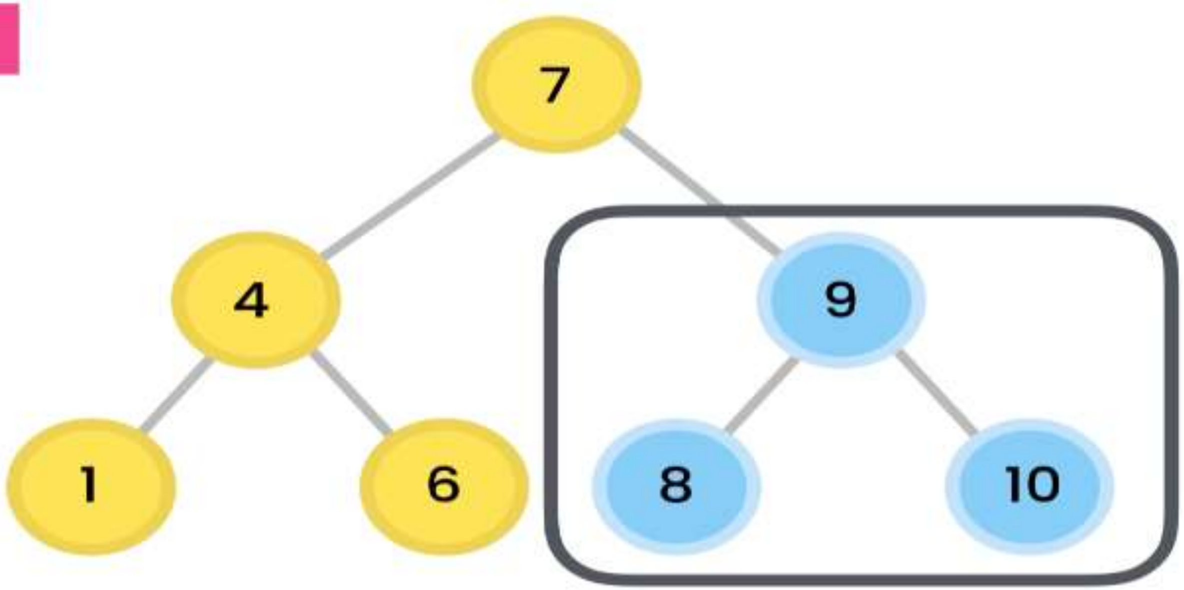
PRE-ORDER

Root, Left, Right



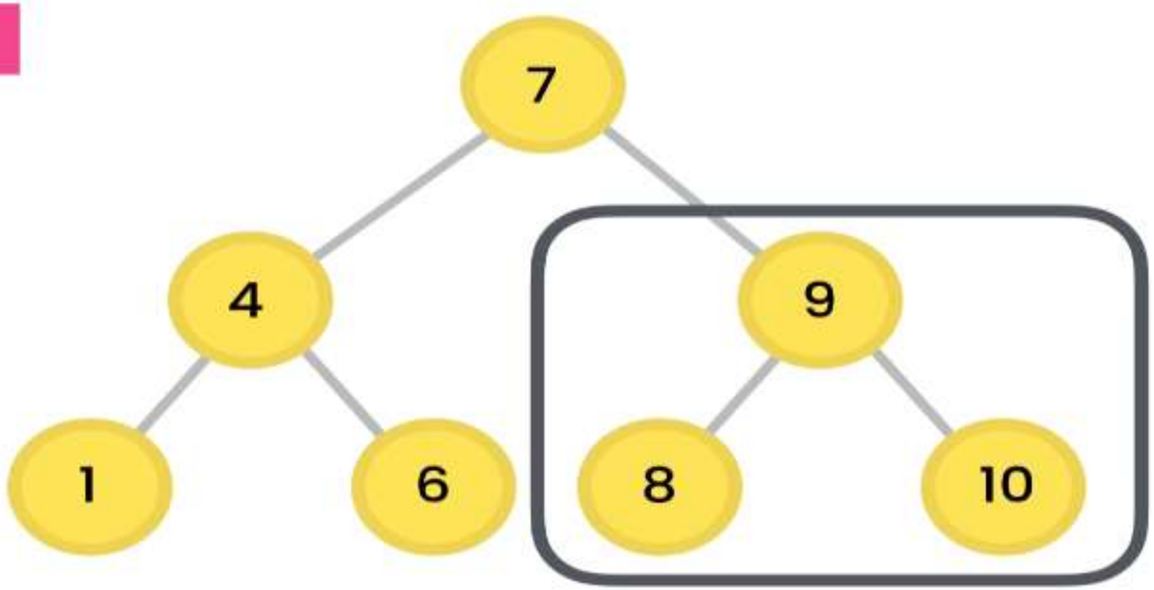
PRE-ORDER

Root, Left, Right



PRE-ORDER

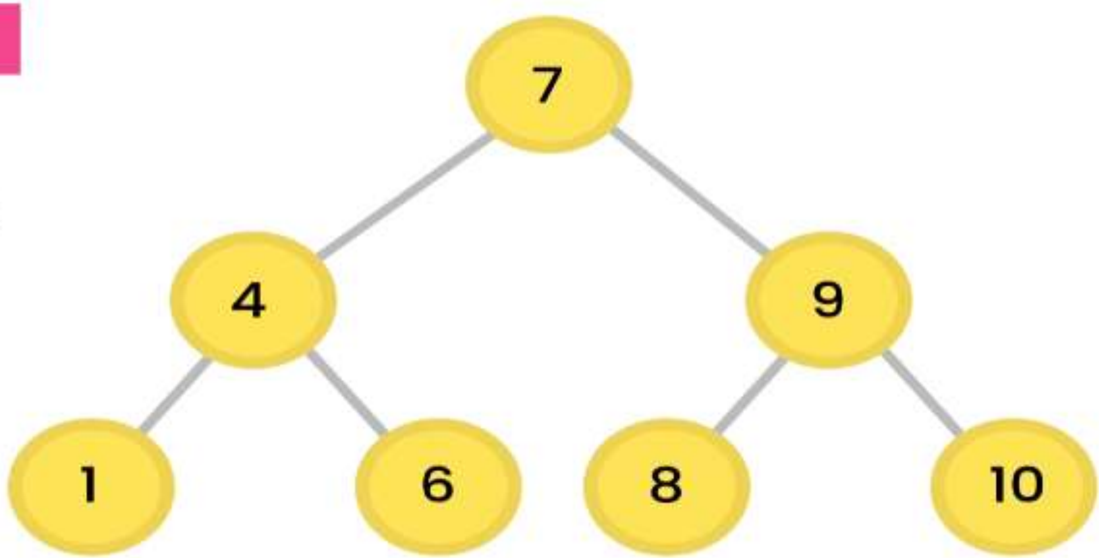
Root, Left, Right



PRE-ORDER

Root, Left, Right

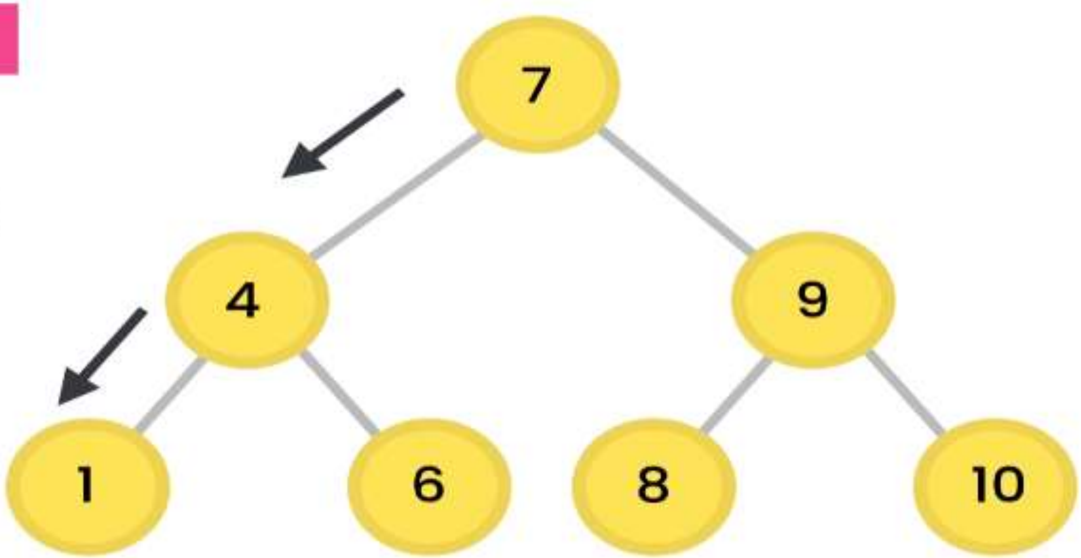
7, 4, 1, 6, 9, 8, 10



PRE-ORDER

Root, Left, Right

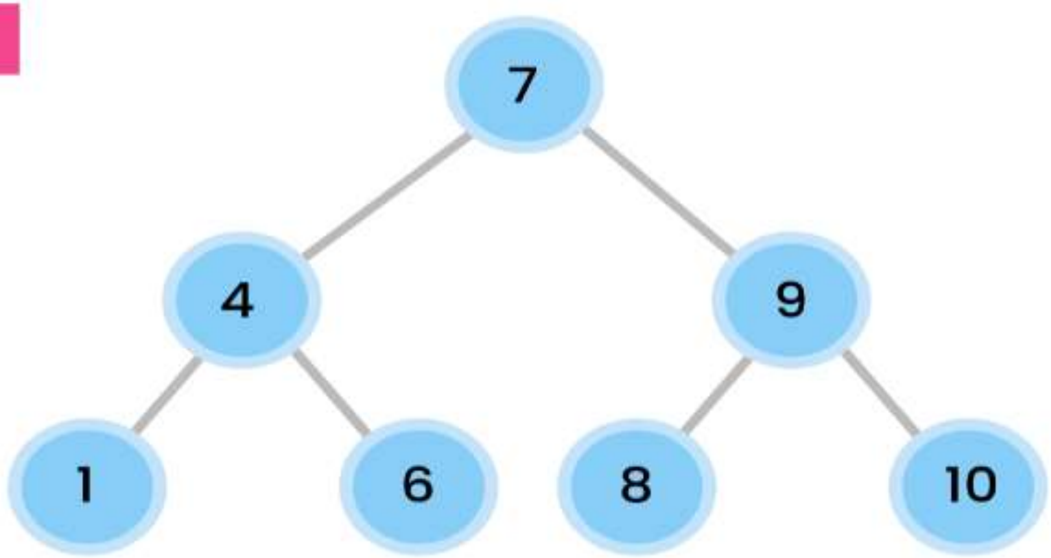
7, 4, 1, 6, 9, 8, 10



In-Order Traversal

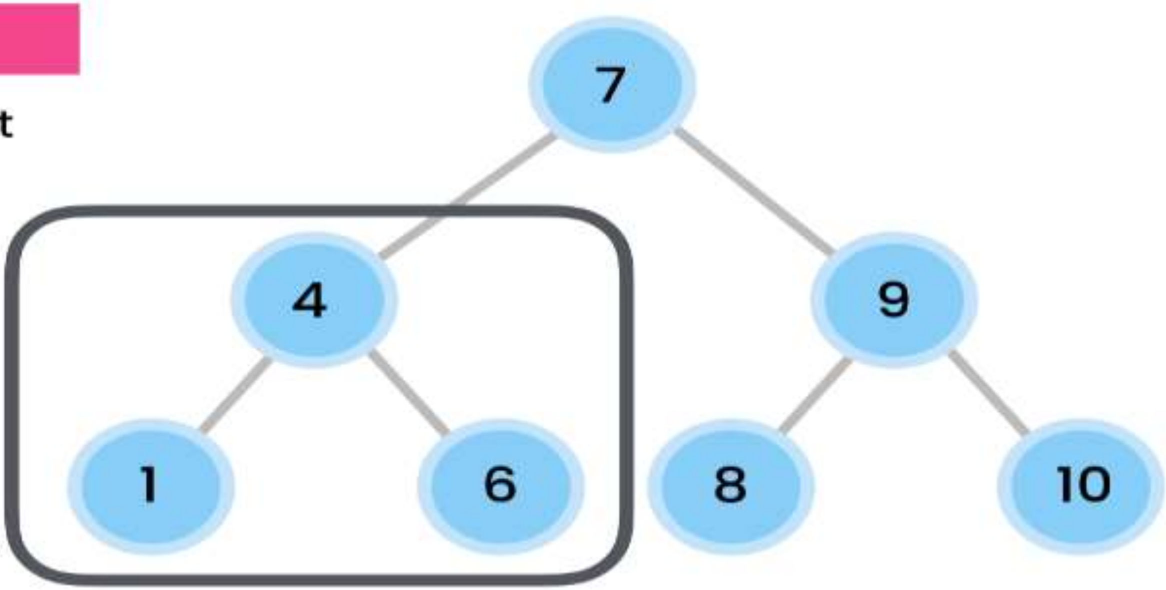
IN-ORDER

Left, Root, Right



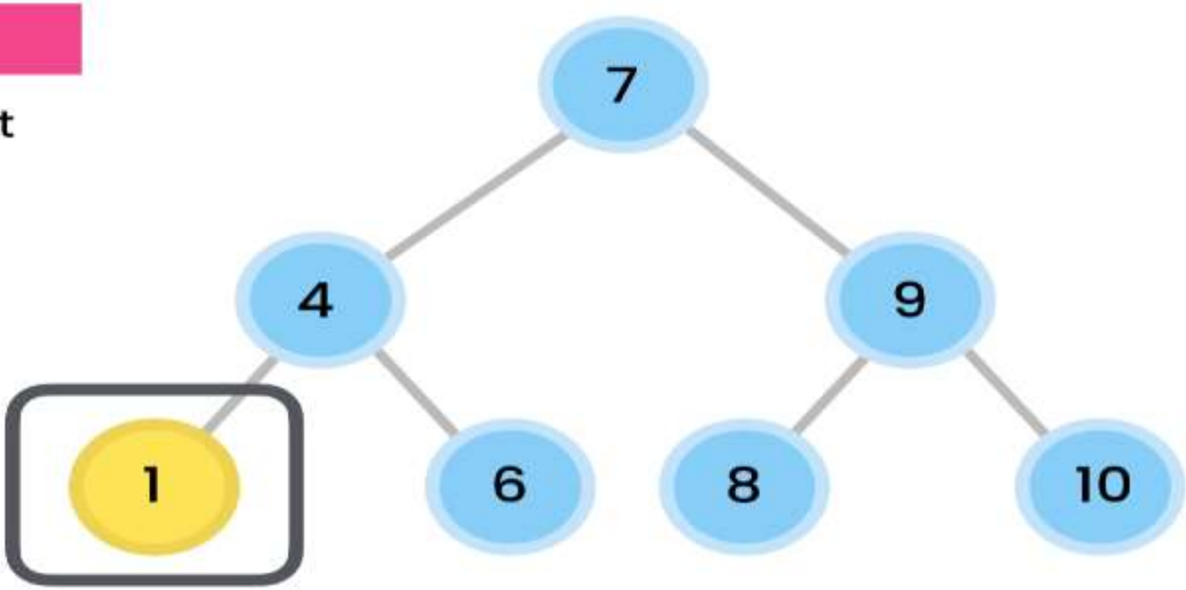
IN-ORDER

Left, Root, Right



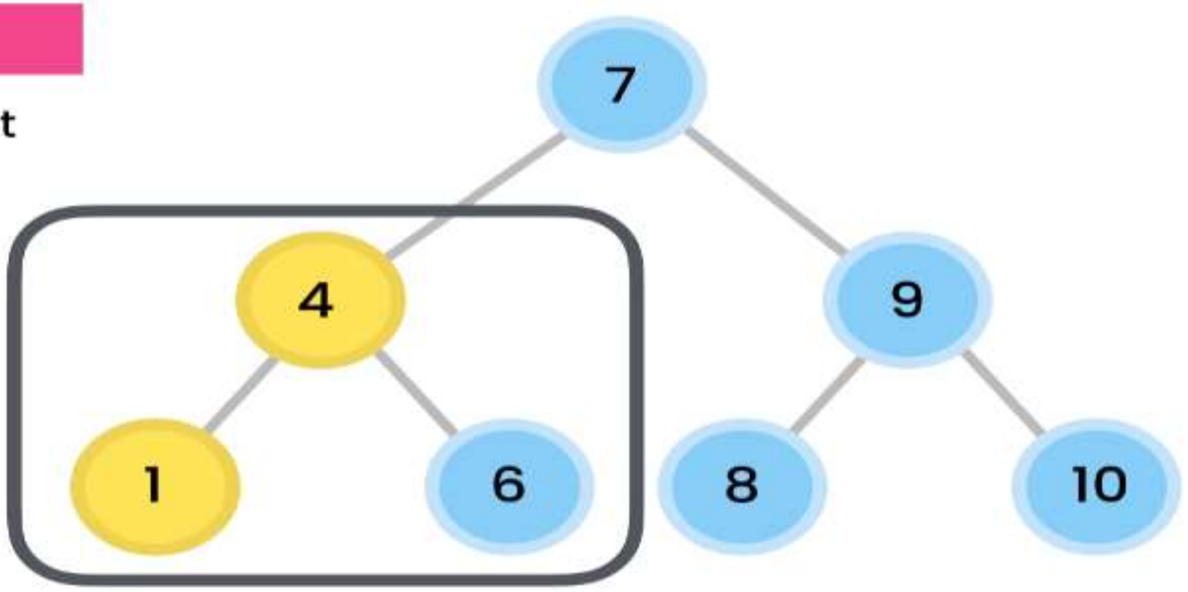
IN-ORDER

Left, Root, Right



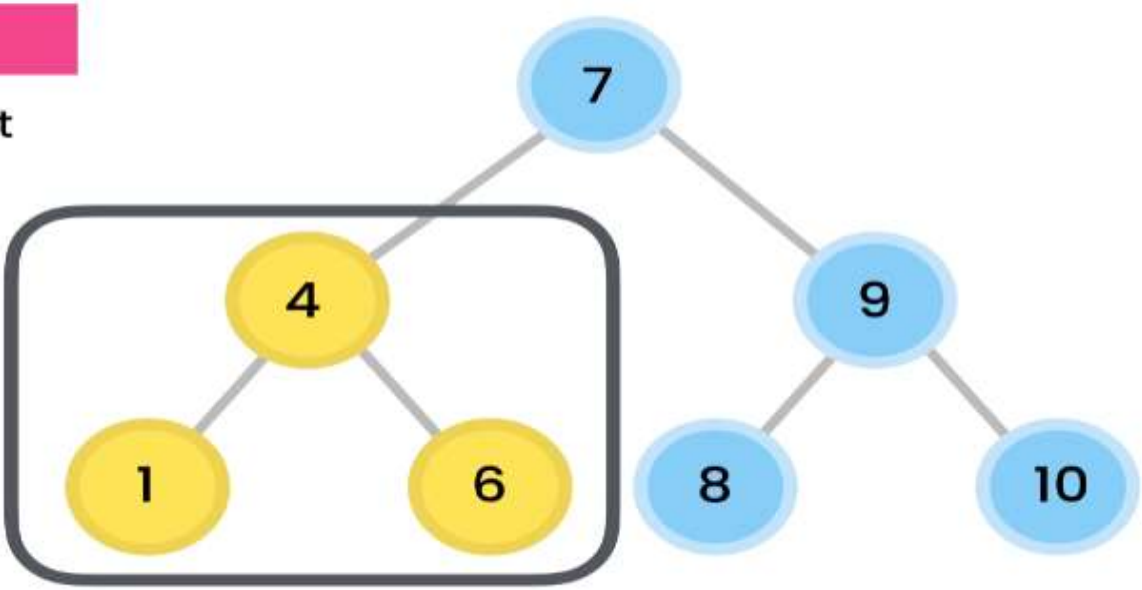
IN-ORDER

Left, Root, Right



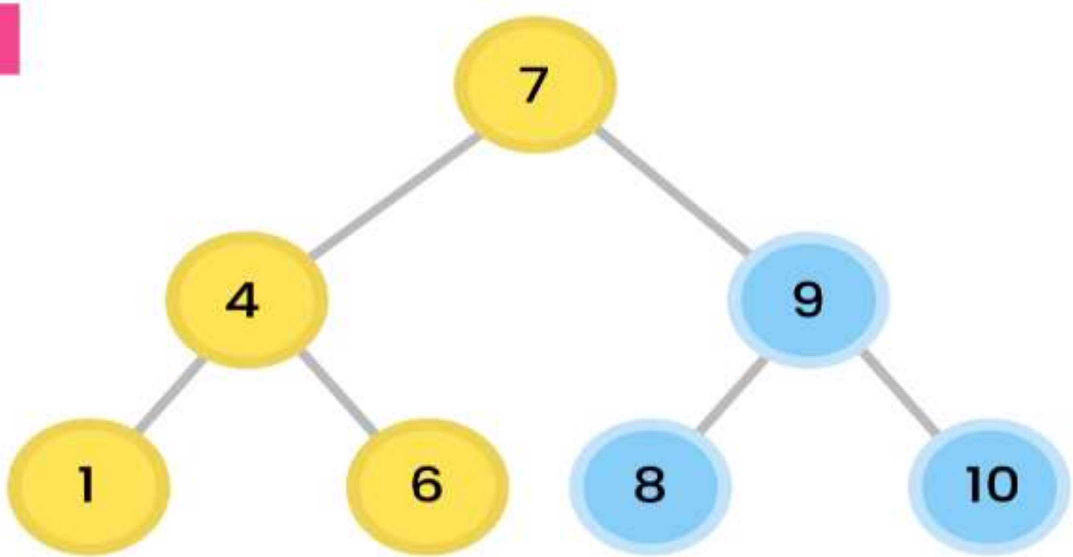
IN-ORDER

Left, Root, Right



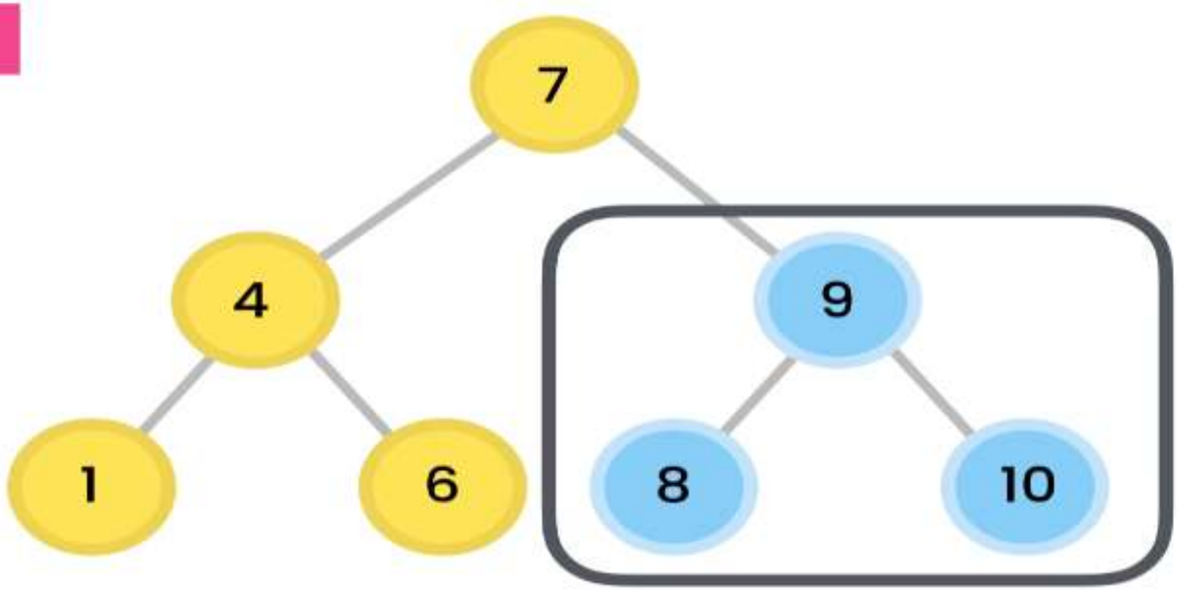
IN-ORDER

Left, Root, Right



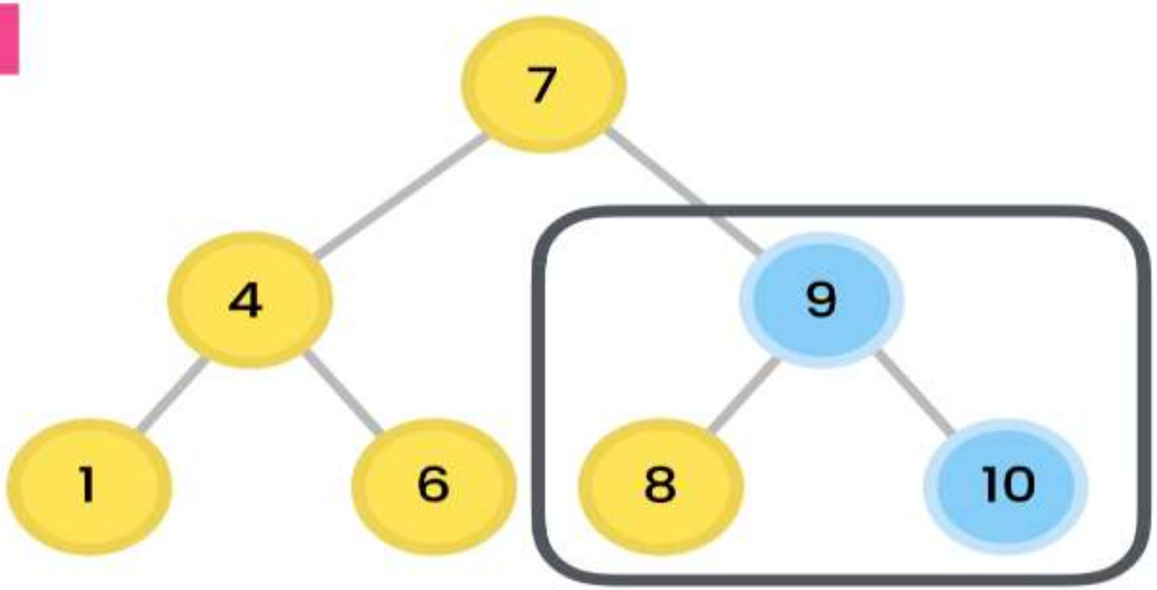
IN-ORDER

Left, Root, Right



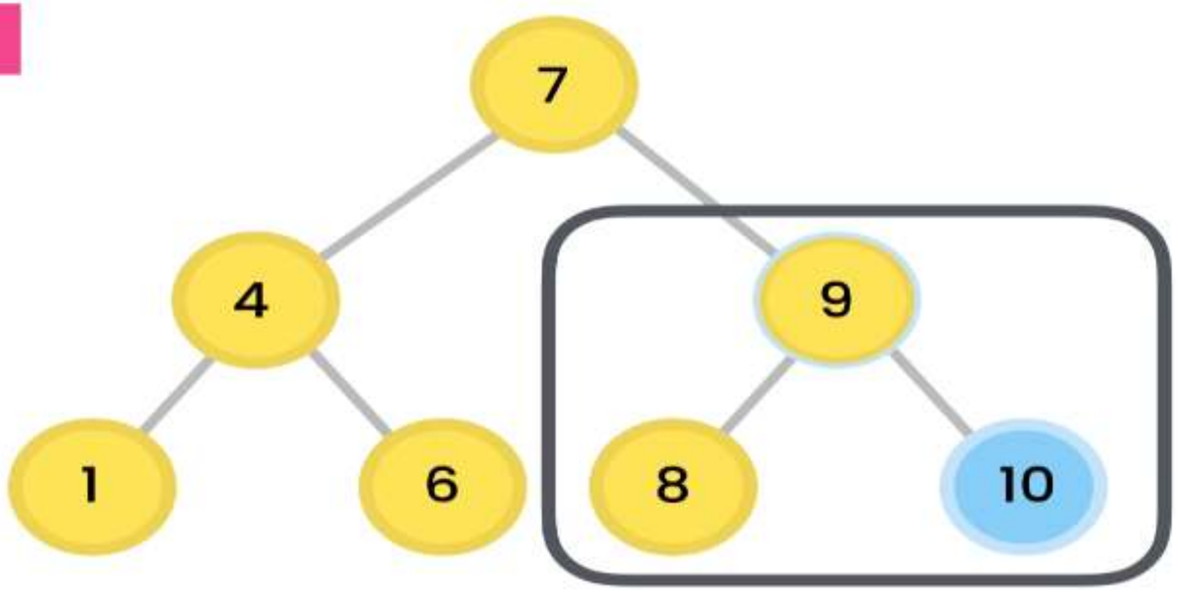
IN-ORDER

Left, Root, Right



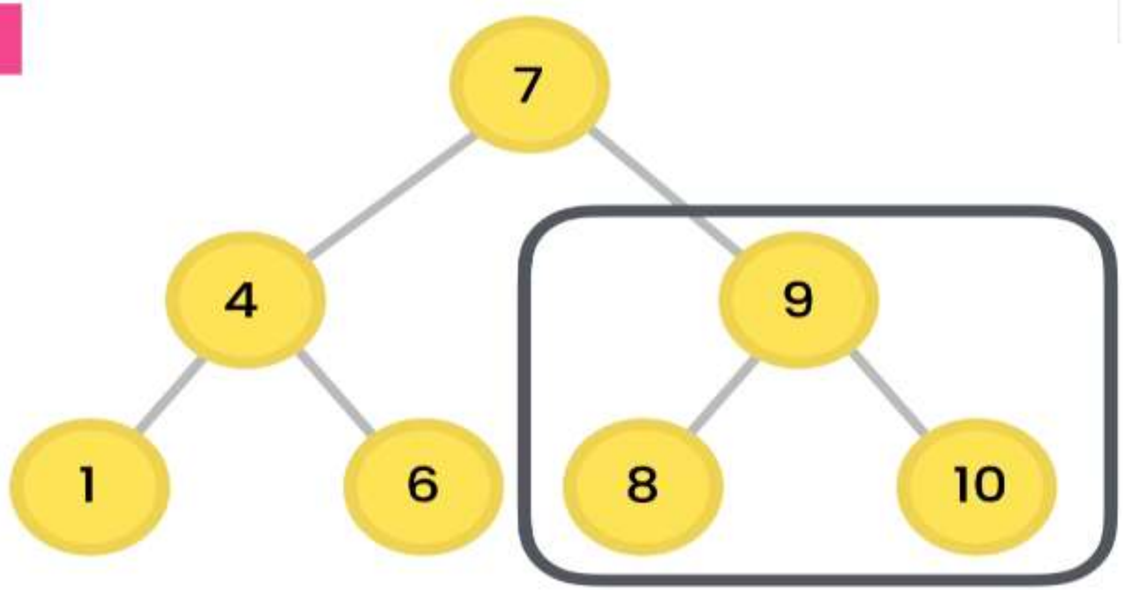
IN-ORDER

Left, Root, Right



IN-ORDER

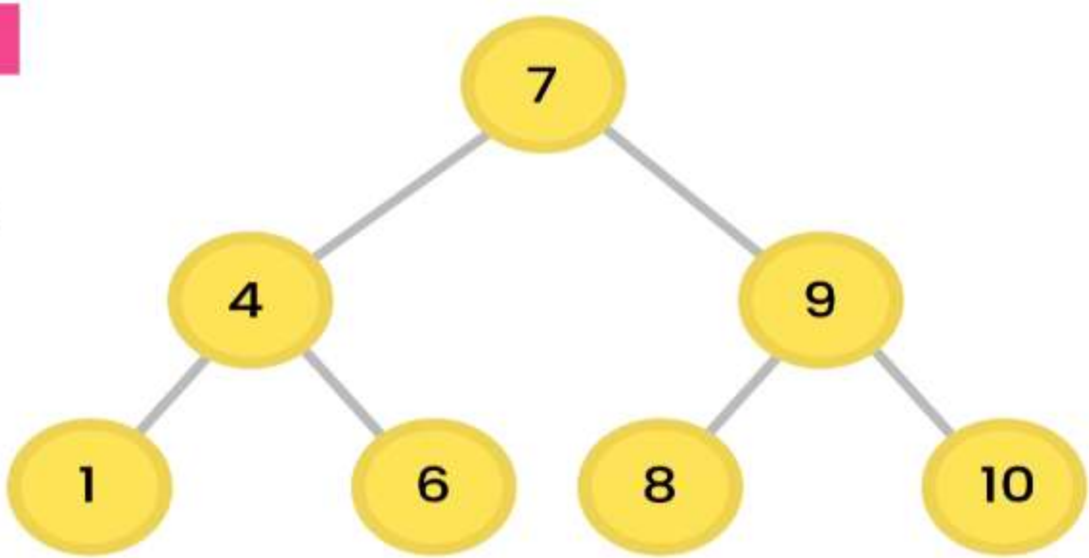
Left, Root, Right



IN-ORDER

Left, Root, Right

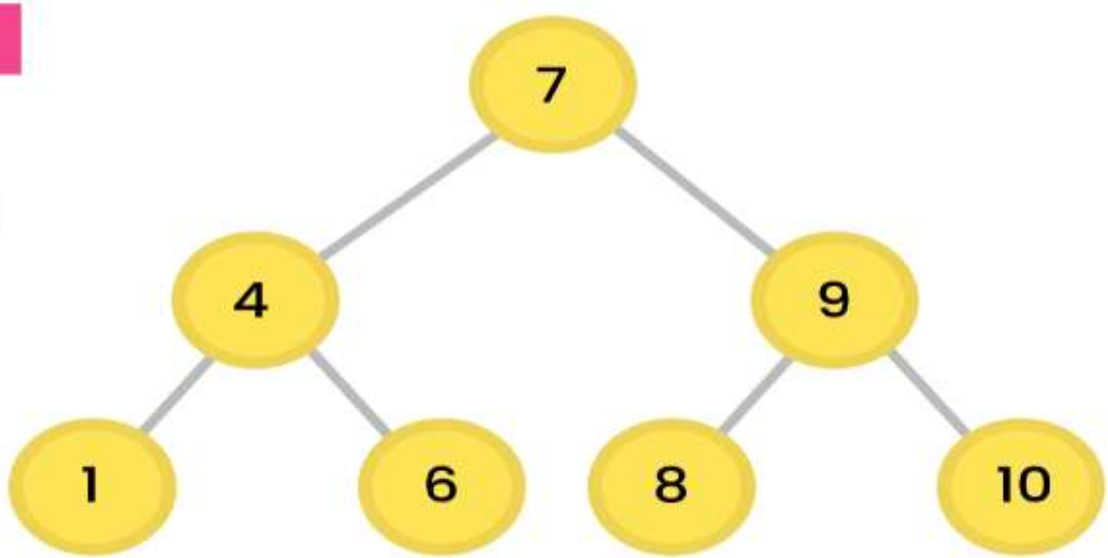
1, 4, 6, 7, 8, 9, 10



IN-ORDER

Right, Root, Left

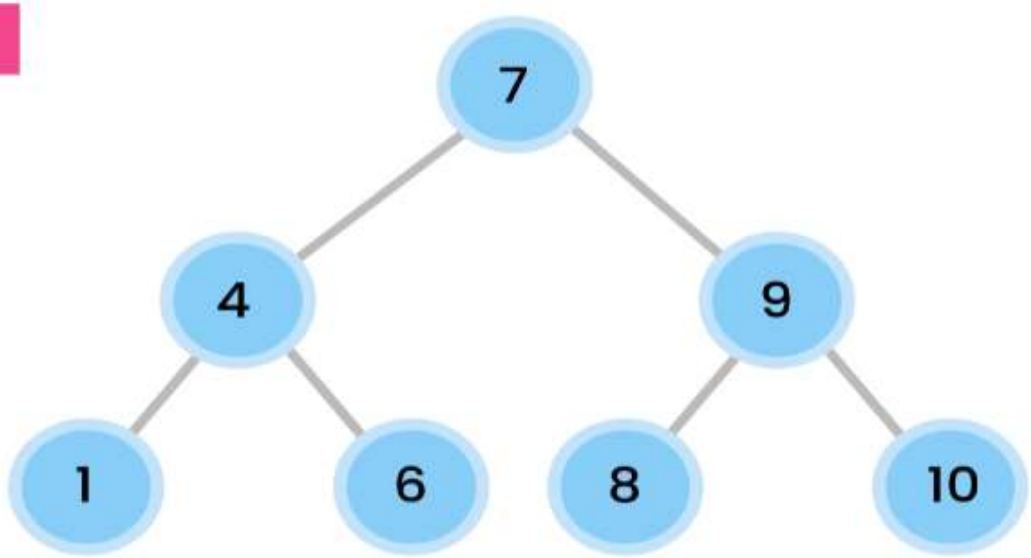
10, 9, 8, 7, 6, 4, 1



Post-Order Traversal

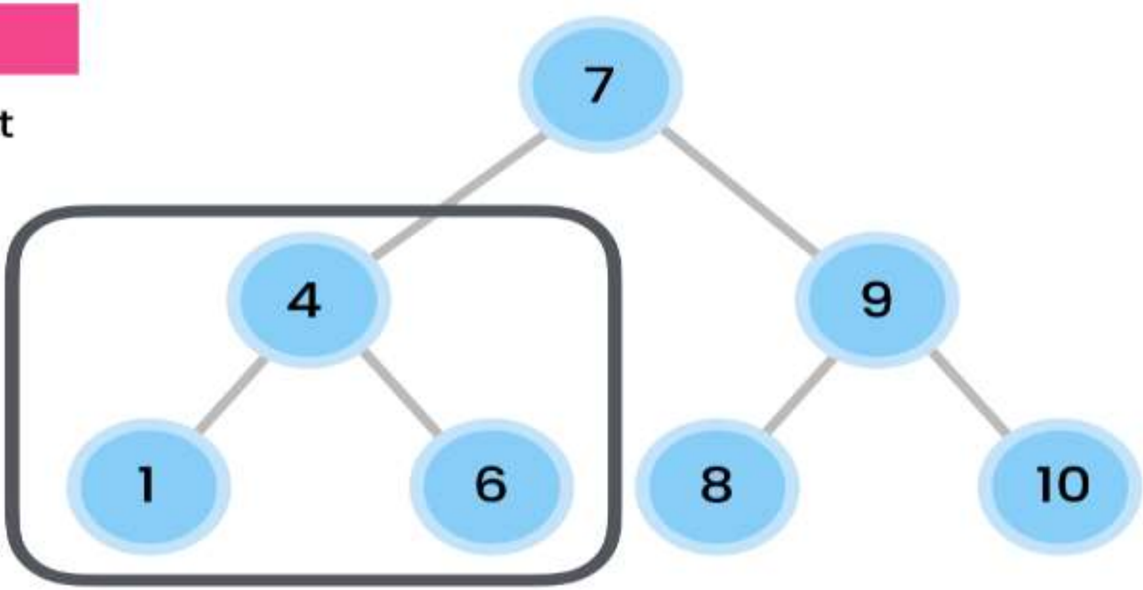
POST-ORDER

Left, Right, Root



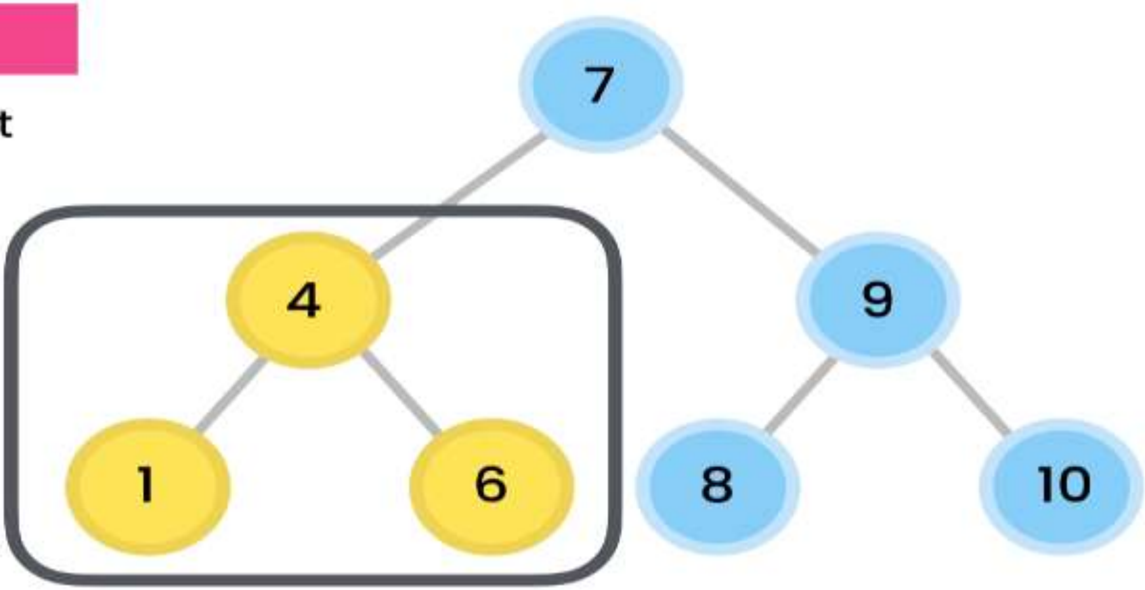
POST-ORDER

Left, Right, Root



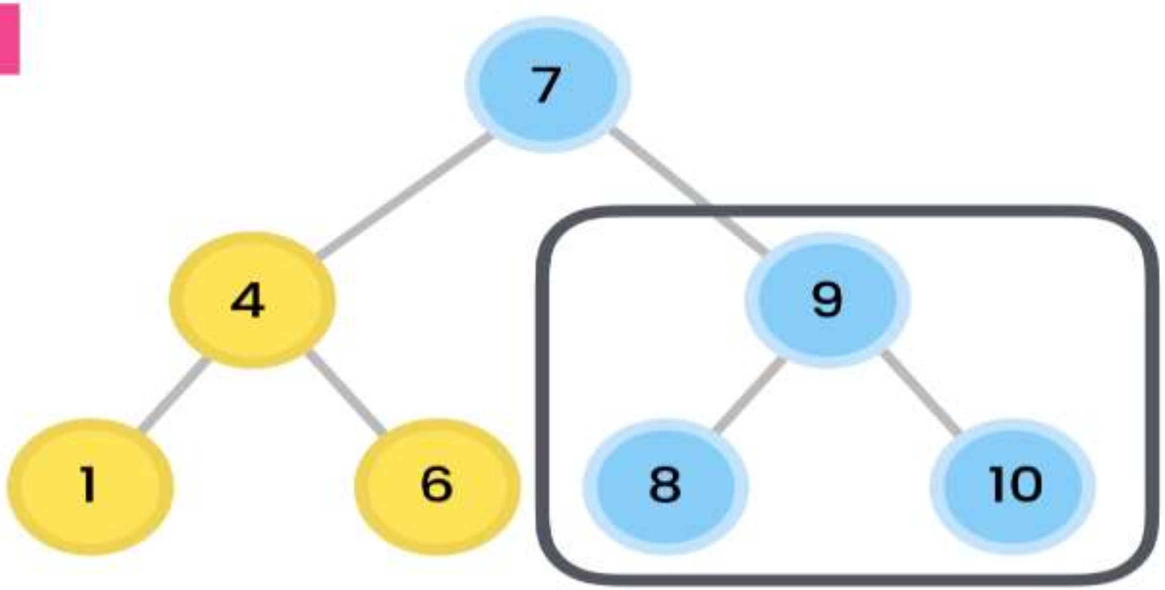
POST-ORDER

Left, Right, Root



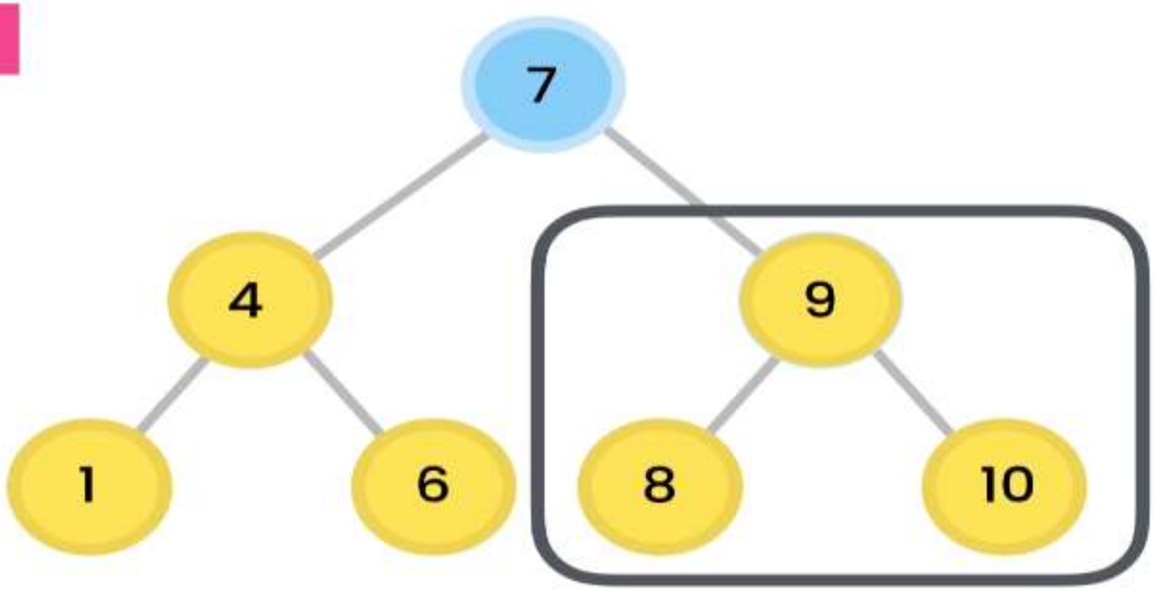
POST-ORDER

Left, Right, Root



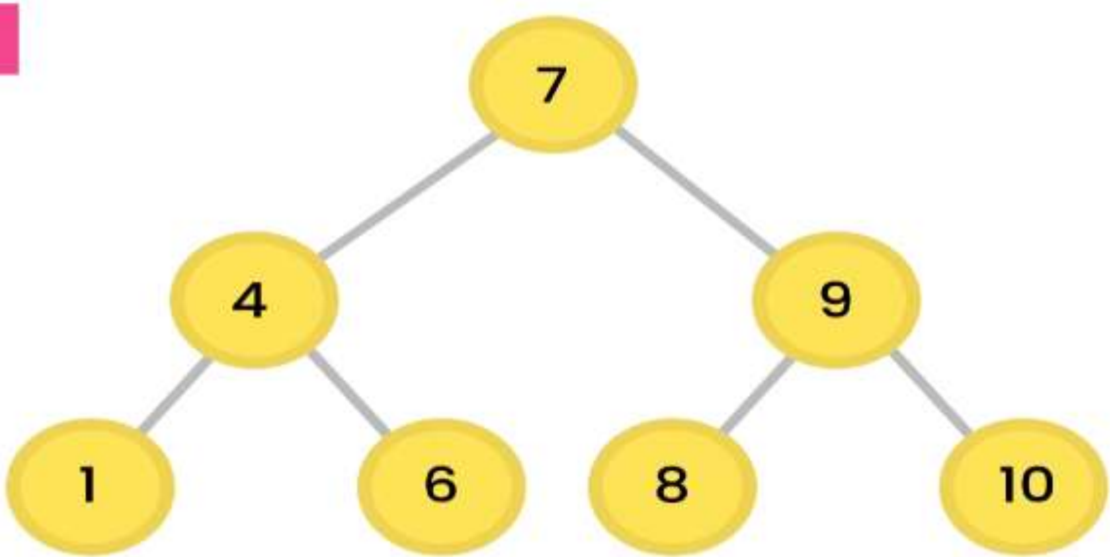
POST-ORDER

Left, Right, Root



POST-ORDER

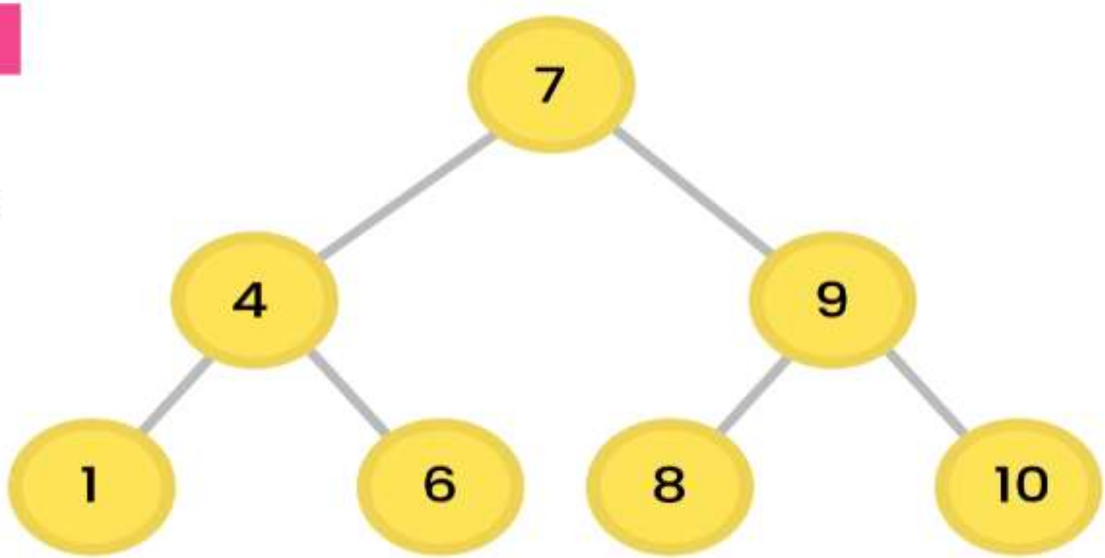
Left, Right, Root



POST-ORDER

Left, Right, Root

1, 6, 4, 8, 10, 9, 7

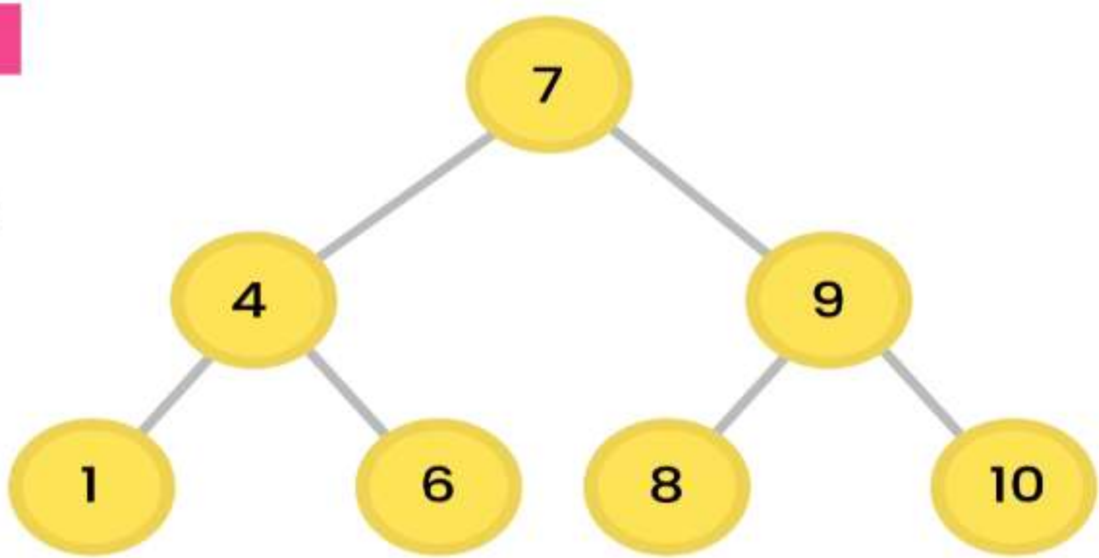


Implementation Of Following Algorithm

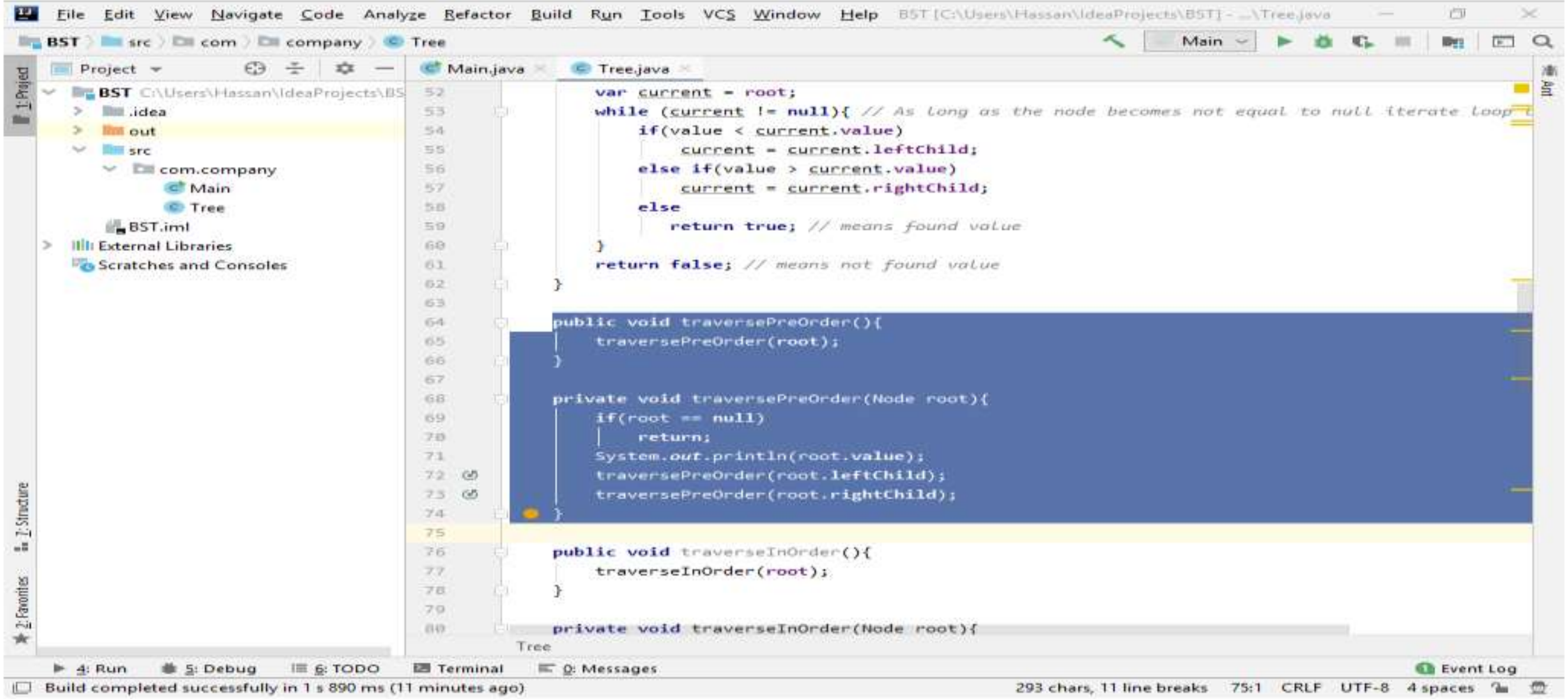
PRE-ORDER

Root, Left, Right

7, 4, 1, 6, 9, 8, 10



Pre-Order Using Recursion

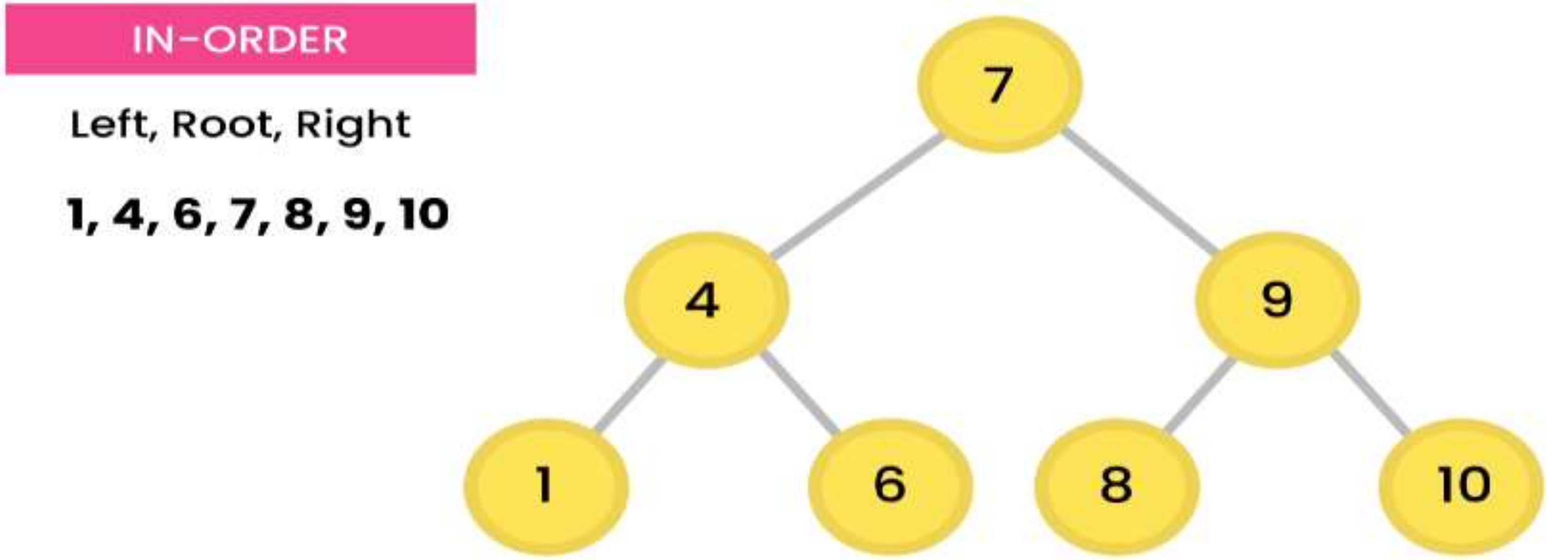


```java
        var current = root;
        while (current != null){ // As long as the node becomes not equal to null iterate loop
            if(value < current.value)
                current = current.leftChild;
            else if(value > current.value)
                current = current.rightChild;
            else
                return true; // means found value
        }
        return false; // means not found value
    }

    public void traversePreOrder(){
        traversePreOrder(root);
    }

    private void traversePreOrder(Node root){
        if(root == null)
            return;
        System.out.println(root.value);
        traversePreOrder(root.leftChild);
        traversePreOrder(root.rightChild);
    }

    public void traverseInOrder(){
        traverseInOrder(root);
    }

    private void traverseInOrder(Node root){
```

Build completed successfully in 1 s 890 ms (11 minutes ago)

293 chars, 11 line breaks 75:1 CRLF UTF-8 4 spaces

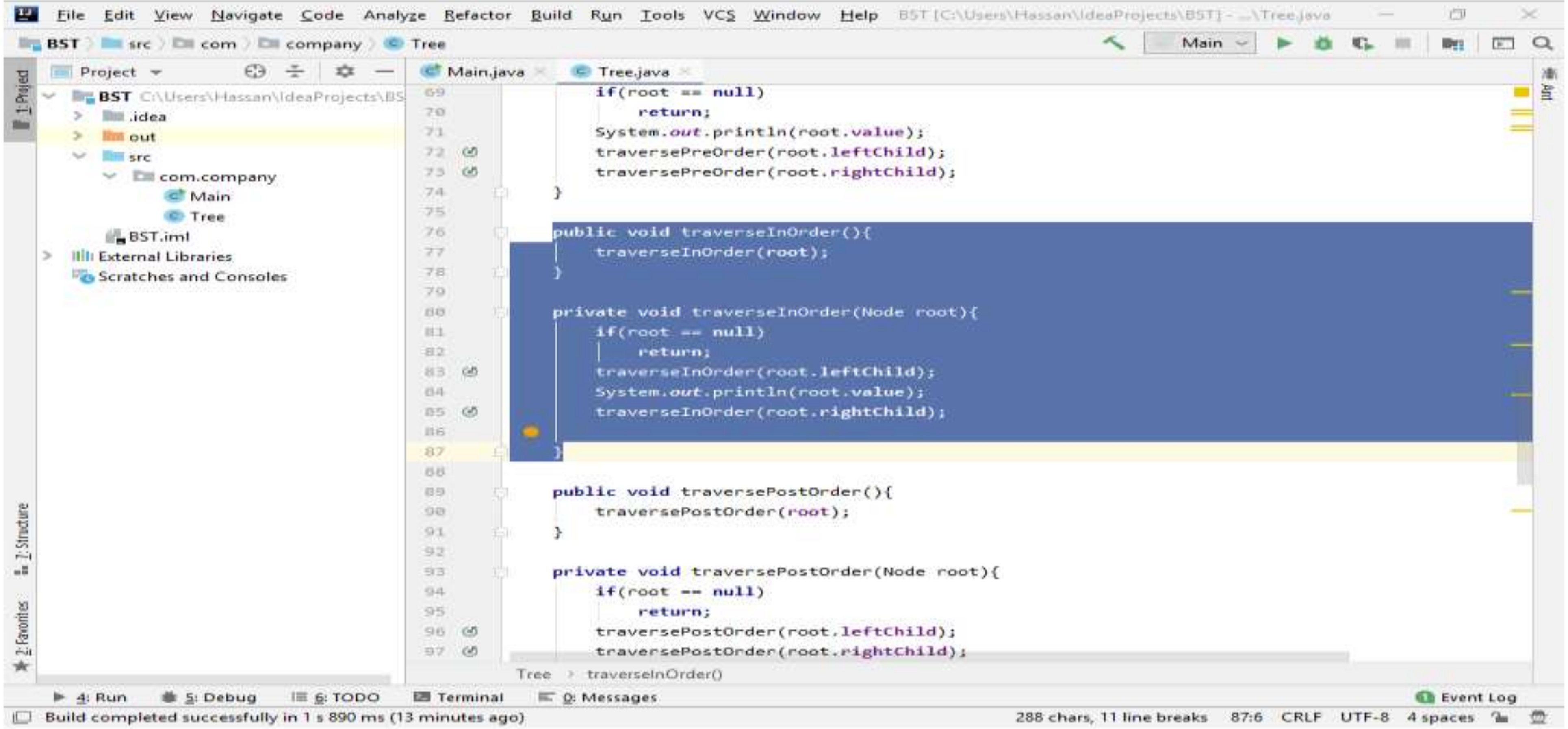
Implementation Of Following Algorithm



IN-ORDER

Left, Root, Right

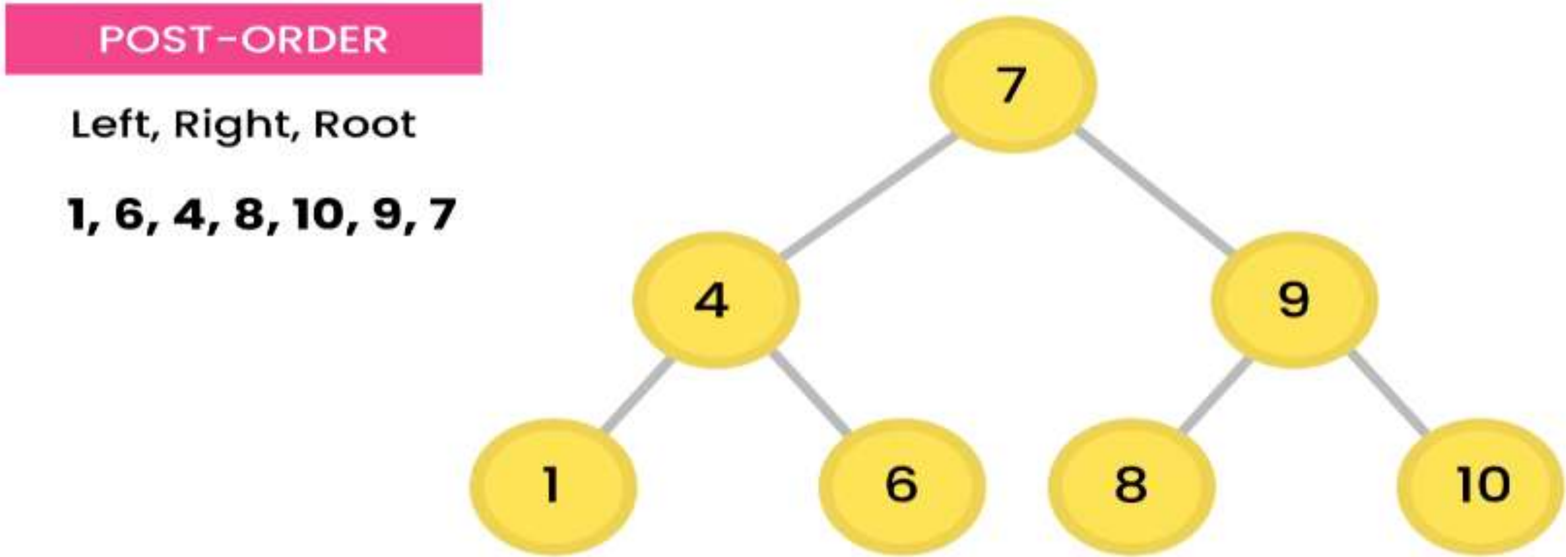
1, 4, 6, 7, 8, 9, 10



```
        if(root == null)
            return;
        System.out.println(root.value);
        traversePreOrder(root.leftChild);
        traversePreOrder(root.rightChild);
    }

    public void traverseInOrder(){
        traverseInOrder(root);
    }

    private void traverseInOrder(Node root){
        if(root == null)
            return;
        traverseInOrder(root.leftChild);
        System.out.println(root.value);
        traverseInOrder(root.rightChild);

    }

    public void traversePostOrder(){
        traversePostOrder(root);
    }

    private void traversePostOrder(Node root){
        if(root == null)
            return;
        traversePostOrder(root.leftChild);
        traversePostOrder(root.rightChild);
```

Build completed successfully in 1 s 890 ms (13 minutes ago)

Implementation Of Following Algorithm

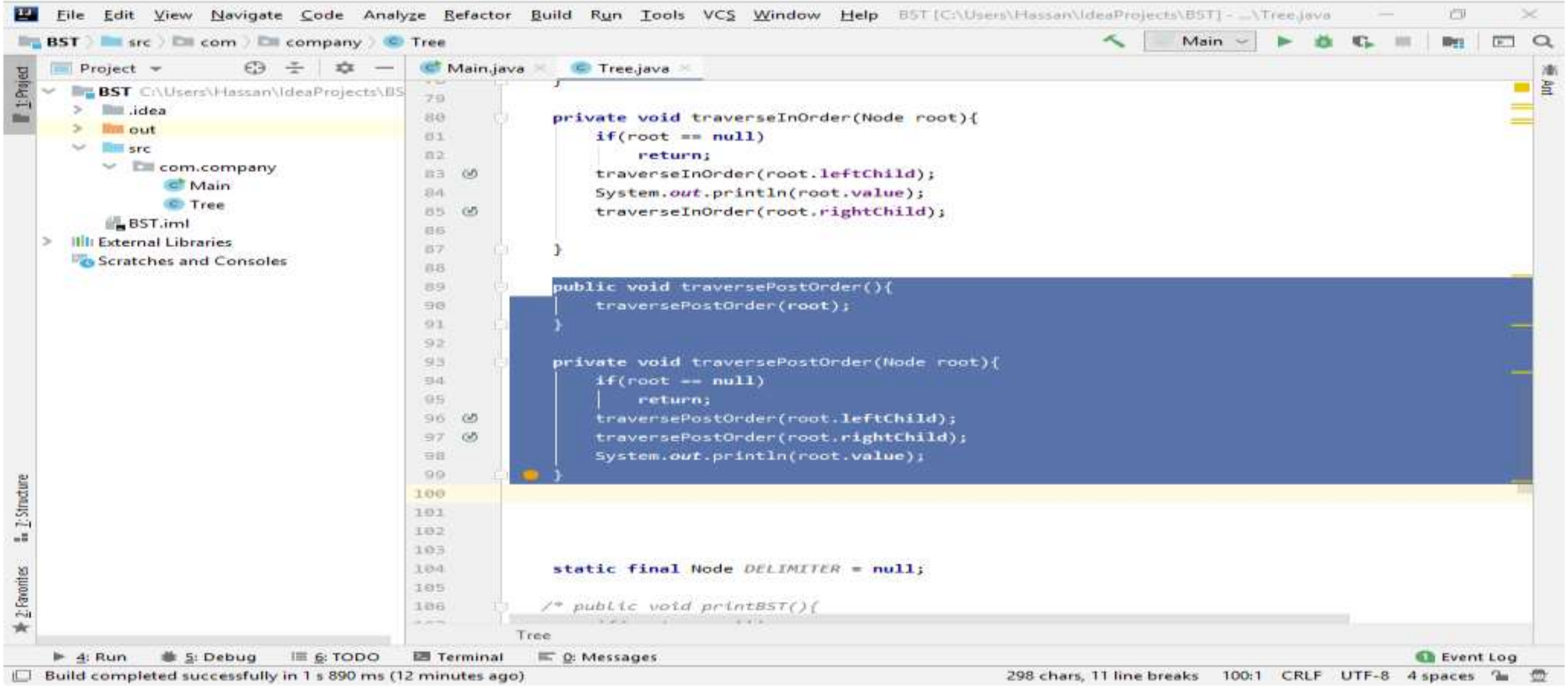


POST-ORDER

Left, Right, Root

1, 6, 4, 8, 10, 9, 7

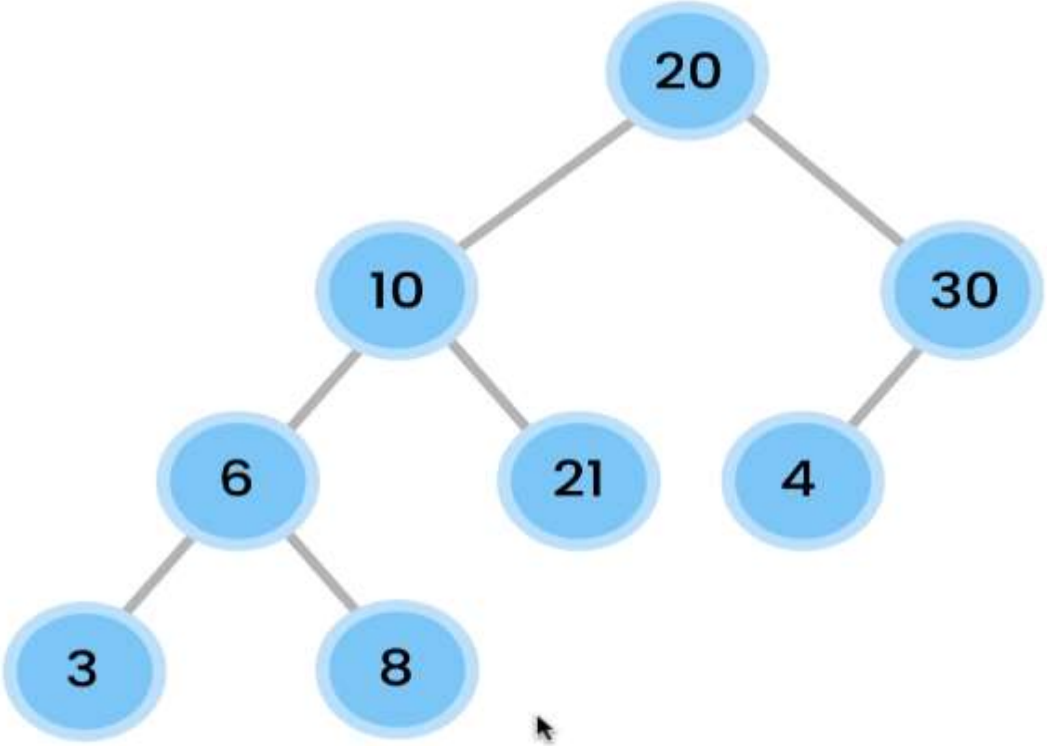
Post-Order Using Recursion



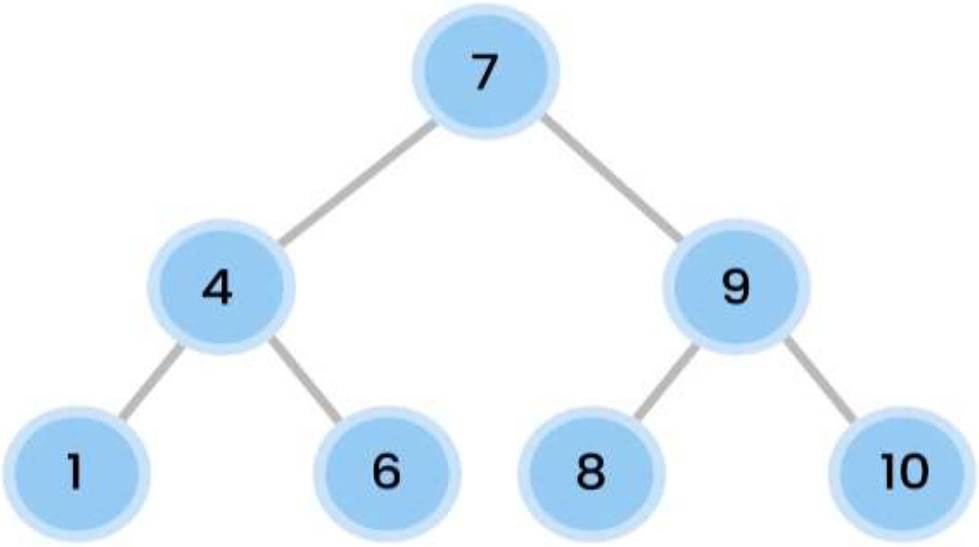
```java
    }

    private void traverseInOrder(Node root){
        if(root == null)
            return;
        traverseInOrder(root.leftChild);
        System.out.println(root.value);
        traverseInOrder(root.rightChild);

    }

    public void traversePostOrder(){
        traversePostOrder(root);
    }

    private void traversePostOrder(Node root){
        if(root == null)
            return;
        traversePostOrder(root.leftChild);
        traversePostOrder(root.rightChild);
        System.out.println(root.value);
    }




    static final Node DELIMITER = null;

  /* public void printBST(){
```

Build completed successfully in 1 s 890 ms (12 minutes ago)

Height & Depth of BT

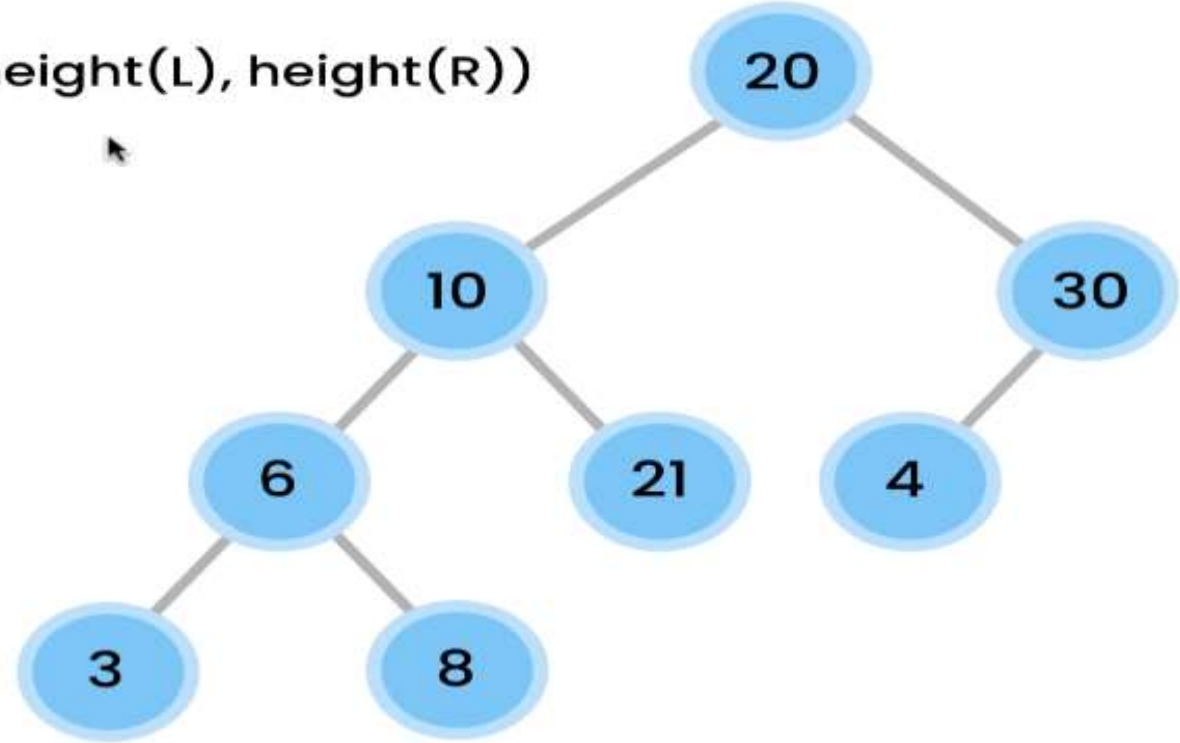


Height & Depth of BT.

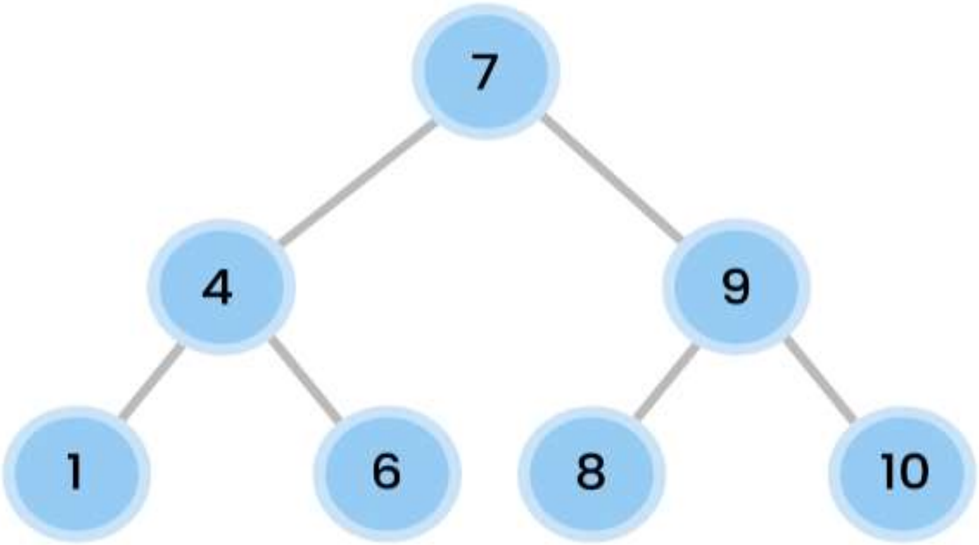


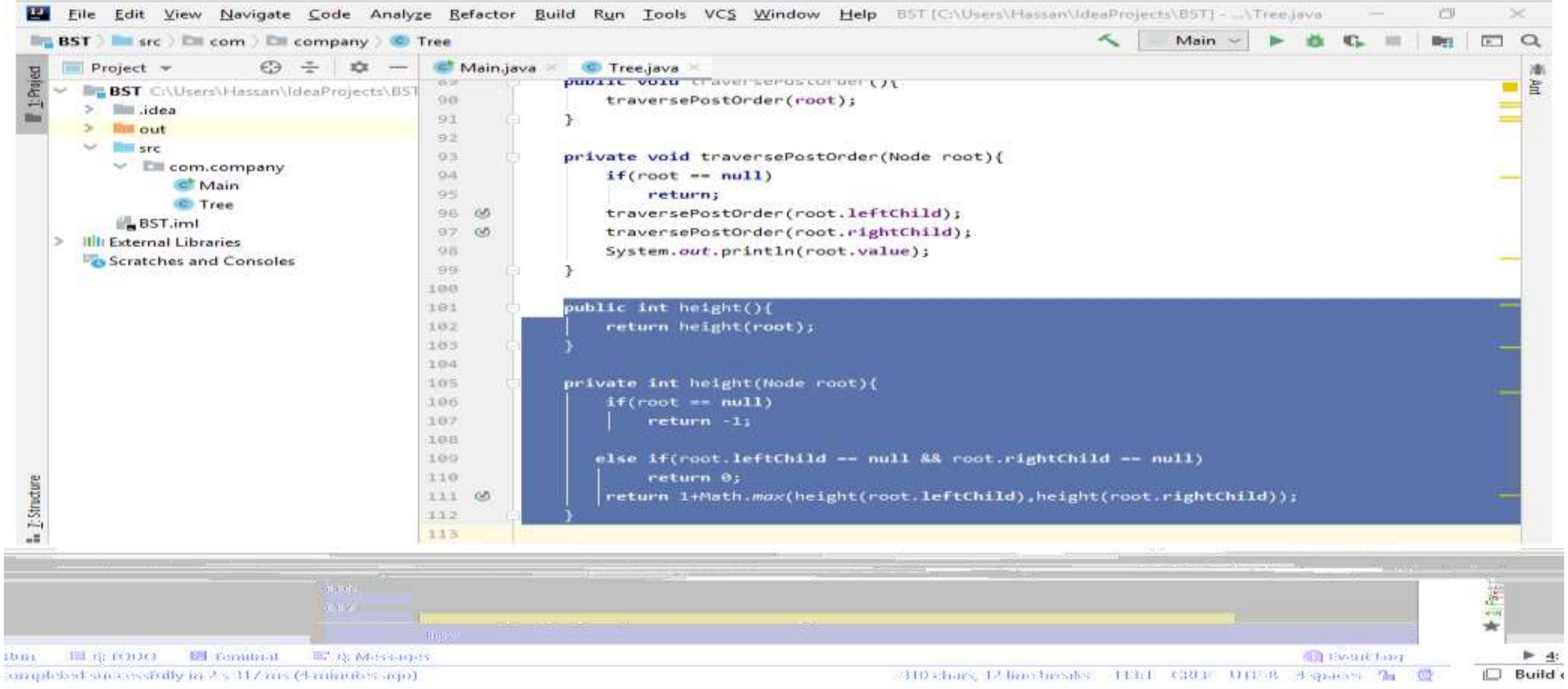
Height Formula (Recursive)

1 + max(height(L), height(R))



Implementation To Find Height Of BST



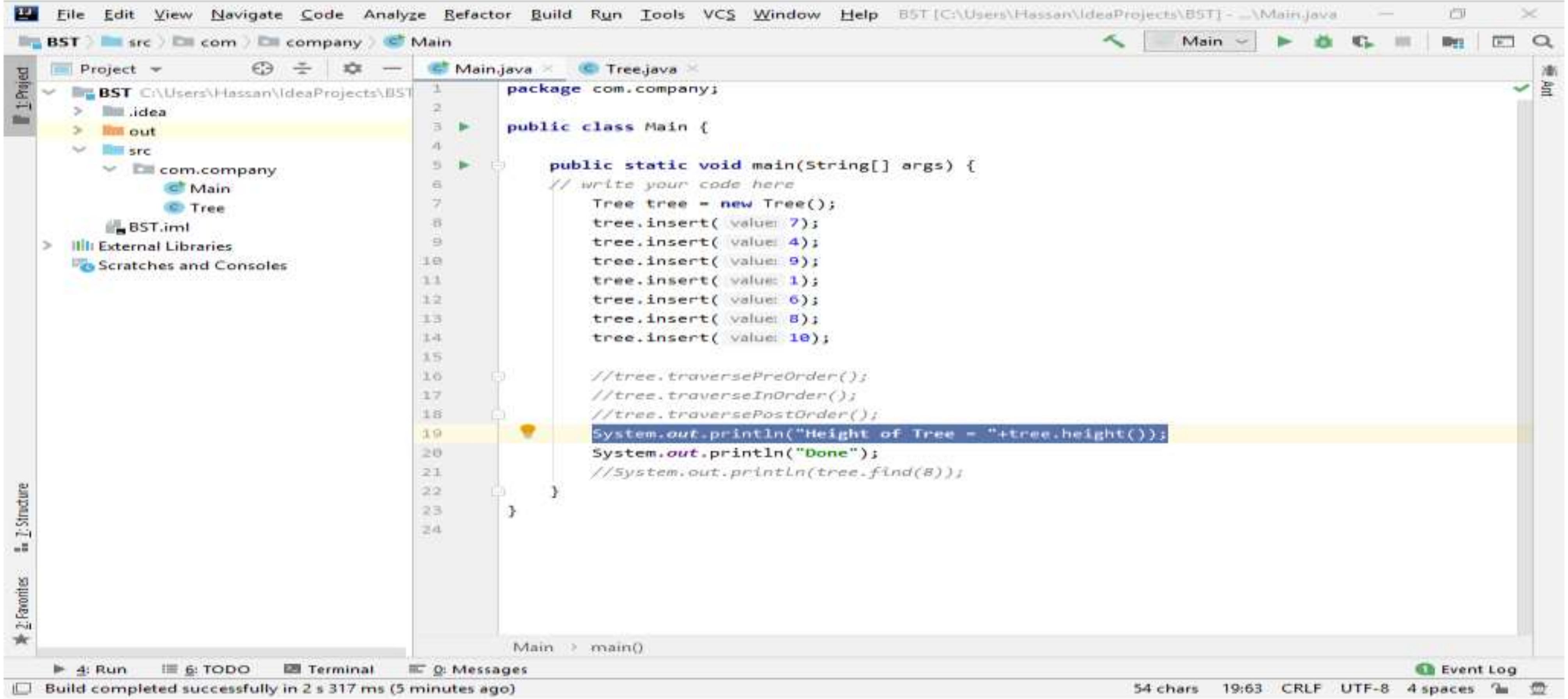


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
BST [C:\Users\Hassan\IdeaProjects\BST] - ...\Tree.java
BST › src › com › company › Tree
Main
Project
BST C:\Users\Hassan\IdeaProjects\BST
.idea
out
src
com.company
Main
Tree
BST.iml
External Libraries
Scratches and Consoles
Main.java
Tree.java
1: Project
7: Structure
Ant
6: TODO
Terminal
0: Messages
Event Log
Build

```java
    public void traversePostOrder(){
        traversePostOrder(root);
    }

    private void traversePostOrder(Node root){
        if(root == null)
            return;
        traversePostOrder(root.leftChild);
        traversePostOrder(root.rightChild);
        System.out.println(root.value);
    }

    public int height(){
        return height(root);
    }

    private int height(Node root){
        if(root == null)
            return -1;

      else if(root.leftChild == null && root.rightChild == null)
            return 0;
        return 1+Math.max(height(root.leftChild),height(root.rightChild));
    }
```

ompleted successfully in 2 s 317 ms (4 minutes ago)

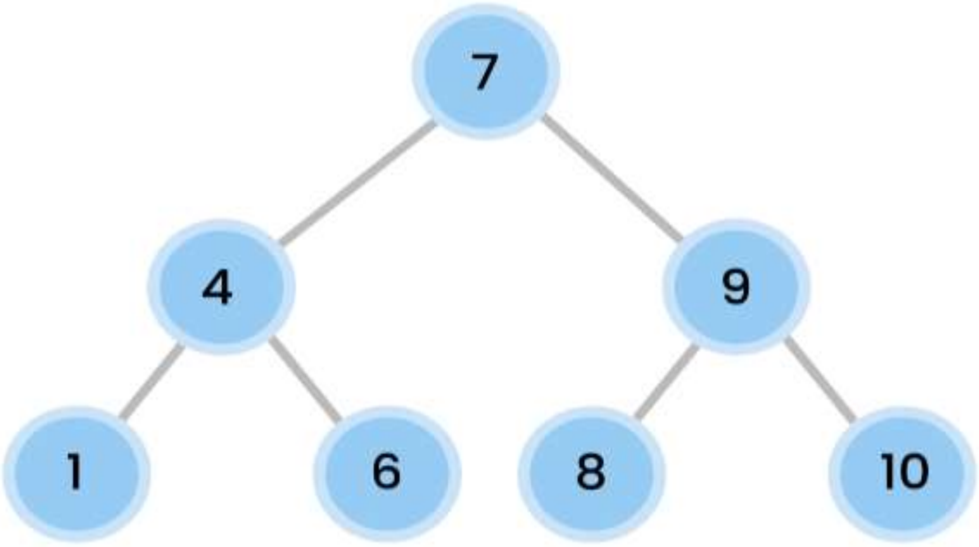
310 chars, 12 line breaks 113:1 CRLF UTF-8 4 spaces

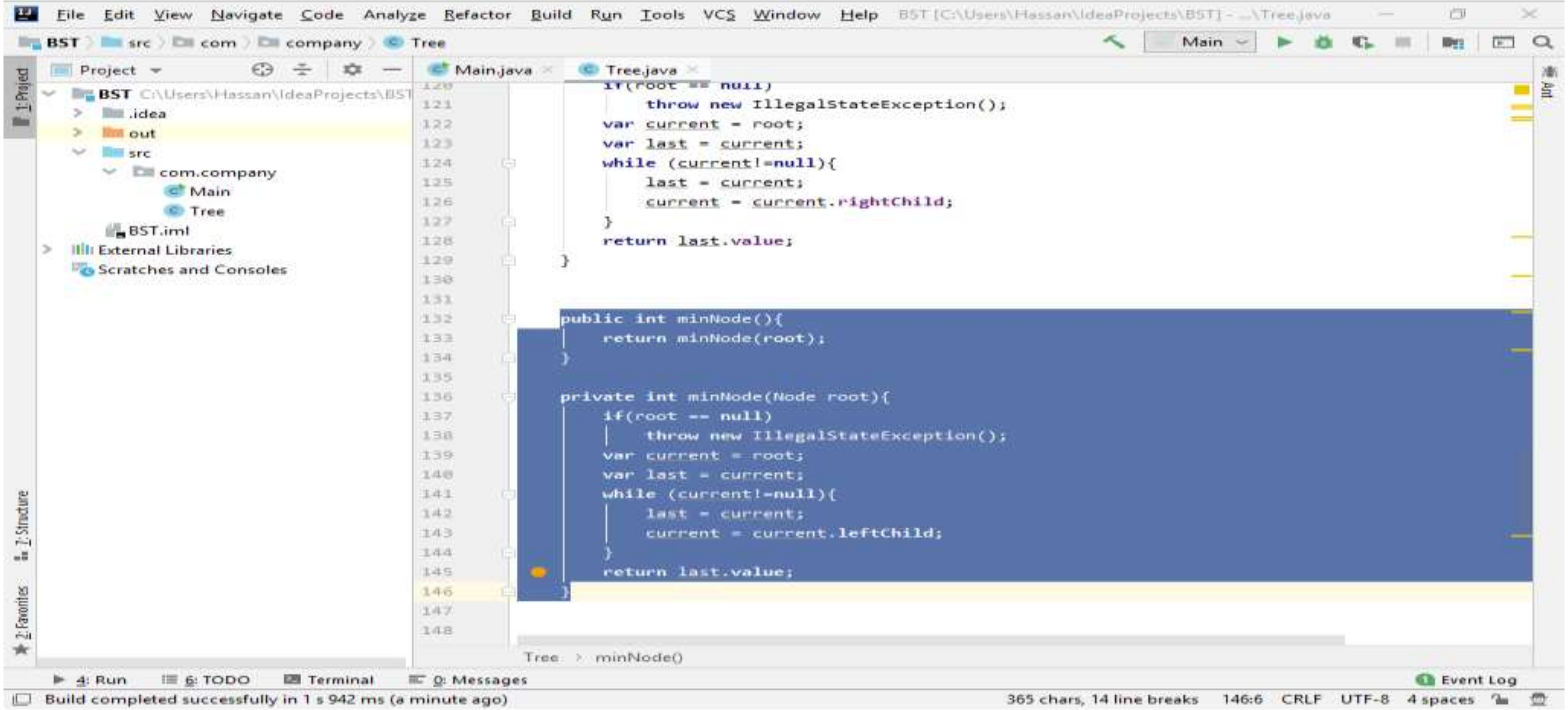


```java
package com.company;

public class Main {

    public static void main(String[] args) {
    // write your code here
        Tree tree = new Tree();
        tree.insert( value: 7);
        tree.insert( value: 4);
        tree.insert( value: 9);
        tree.insert( value: 1);
        tree.insert( value: 6);
        tree.insert( value: 8);
        tree.insert( value: 10);

        //tree.traversePreOrder();
        //tree.traverseInOrder();
        //tree.traversePostOrder();
        System.out.println("Height of Tree = "+tree.height());
        System.out.println("Done");
        //System.out.println(tree.find(8));
    }
}
```

Build completed successfully in 2 s 317 ms (5 minutes ago)

Implementation To Find Min Node Of BST

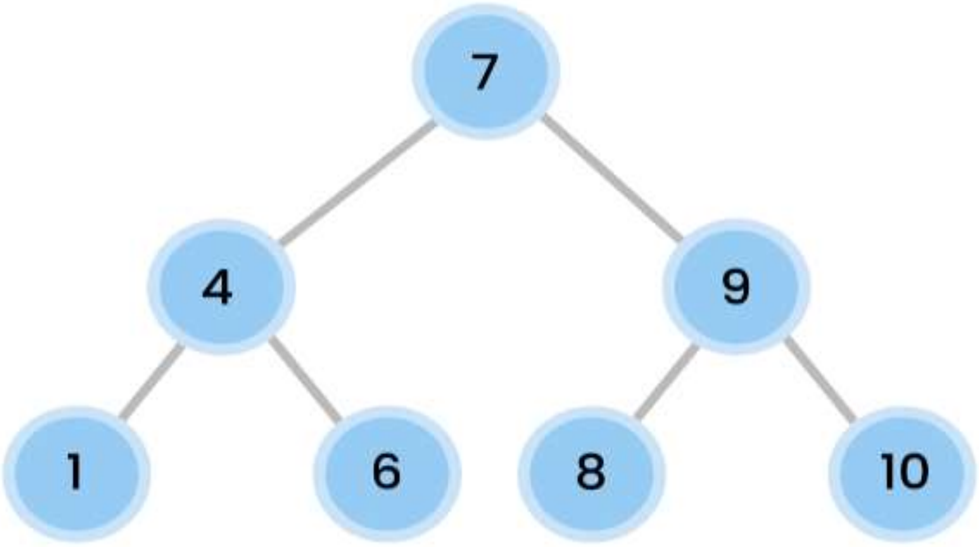


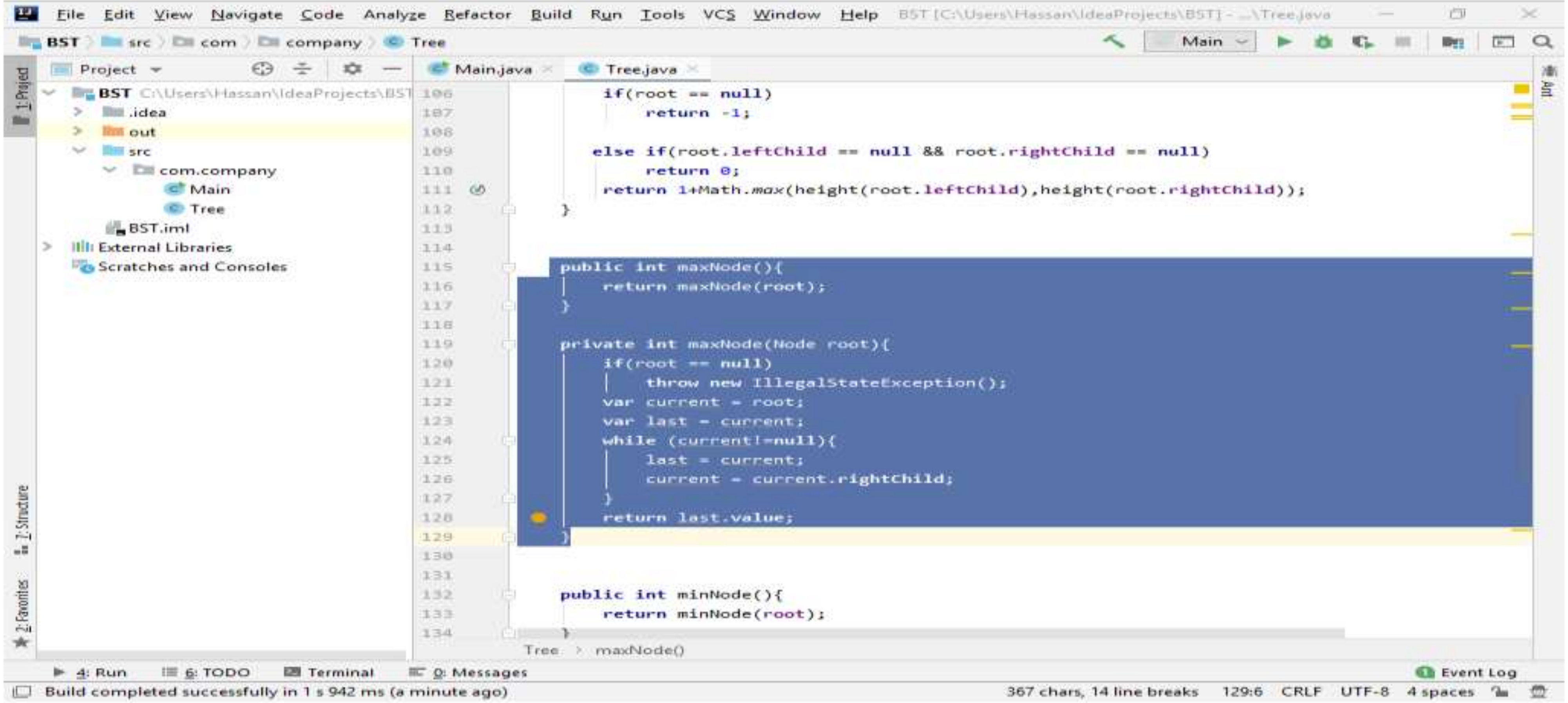

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
BST [C:\Users\Hassan\IdeaProjects\BST] - ...\Tree.java
BST › src › com › company › Tree
Main
Project
Main.java
Tree.java
BST C:\Users\Hassan\IdeaProjects\BST
.idea
out
src
com.company
Main
Tree
BST.iml
External Libraries
Scratches and Consoles
1: Project
7: Structure
2: Favorites
Ant
Tree › minNode()
4: Run
6: TODO
Terminal
0: Messages
Event Log
365 chars, 14 line breaks
146:6
CRLF
UTF-8
4 spaces

```java
        if(root == null)
            throw new IllegalStateException();
        var current = root;
        var last = current;
        while (current!=null){
            last = current;
            current = current.rightChild;
        }
        return last.value;
    }


    public int minNode(){
        return minNode(root);
    }

    private int minNode(Node root){
        if(root == null)
            throw new IllegalStateException();
        var current = root;
        var last = current;
        while (current!=null){
            last = current;
            current = current.leftChild;
        }
        return last.value;
    }
```

Build completed successfully in 1 s 942 ms (a minute ago)

Implementation To Find Max Node Of BST

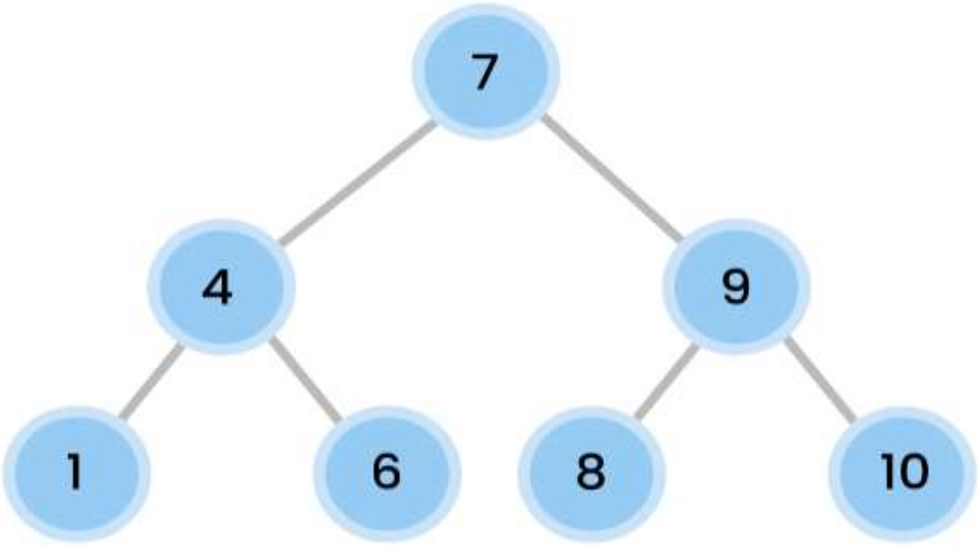


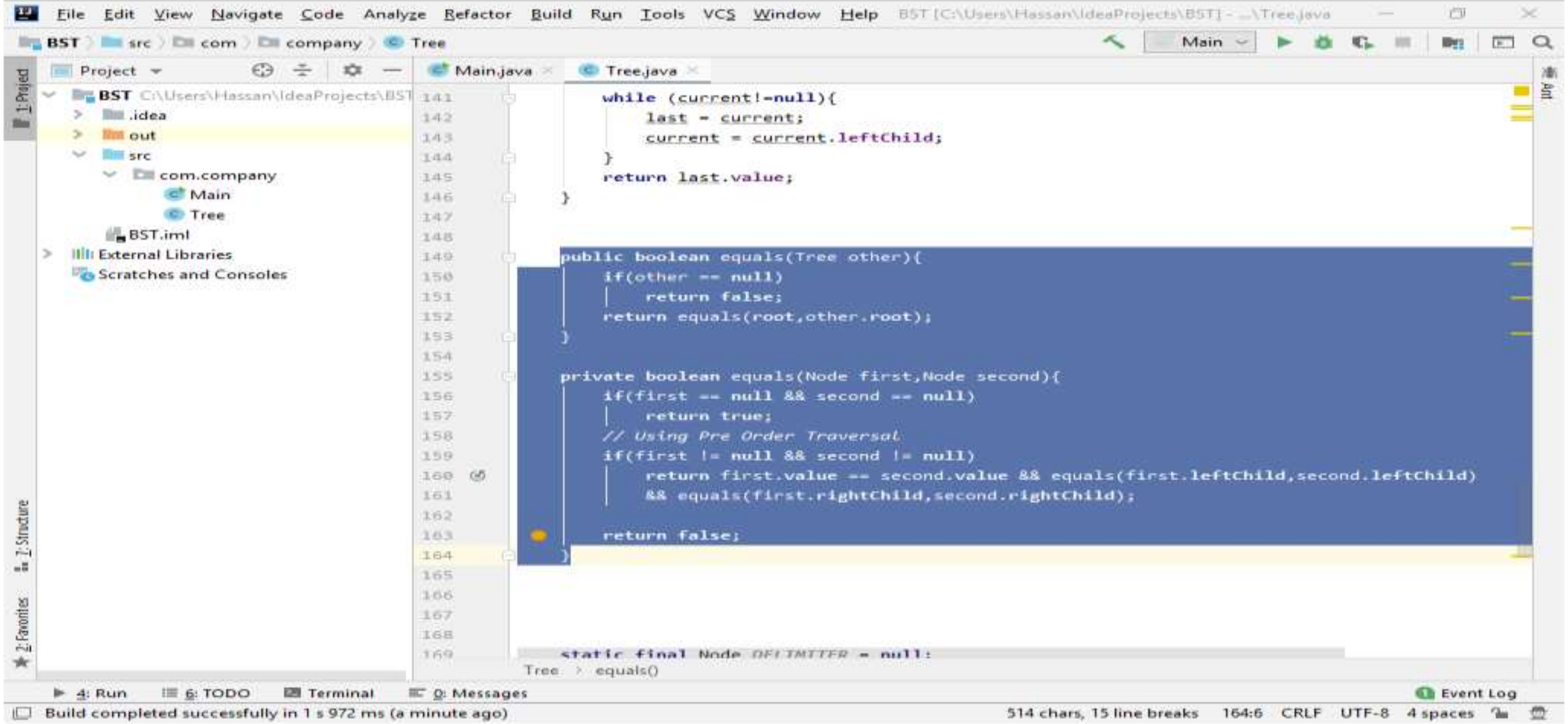


```java
        if(root == null)
            return -1;

       else if(root.leftChild == null && root.rightChild == null)
            return 0;
        return 1+Math.max(height(root.leftChild),height(root.rightChild));
    }


    public int maxNode(){
        return maxNode(root);
    }

    private int maxNode(Node root){
        if(root == null)
            throw new IllegalStateException();
        var current = root;
        var last = current;
        while (current!=null){
            last = current;
            current = current.rightChild;
        }
        return last.value;
    }


    public int minNode(){
        return minNode(root);
    }
```

Build completed successfully in 1 s 942 ms (a minute ago)

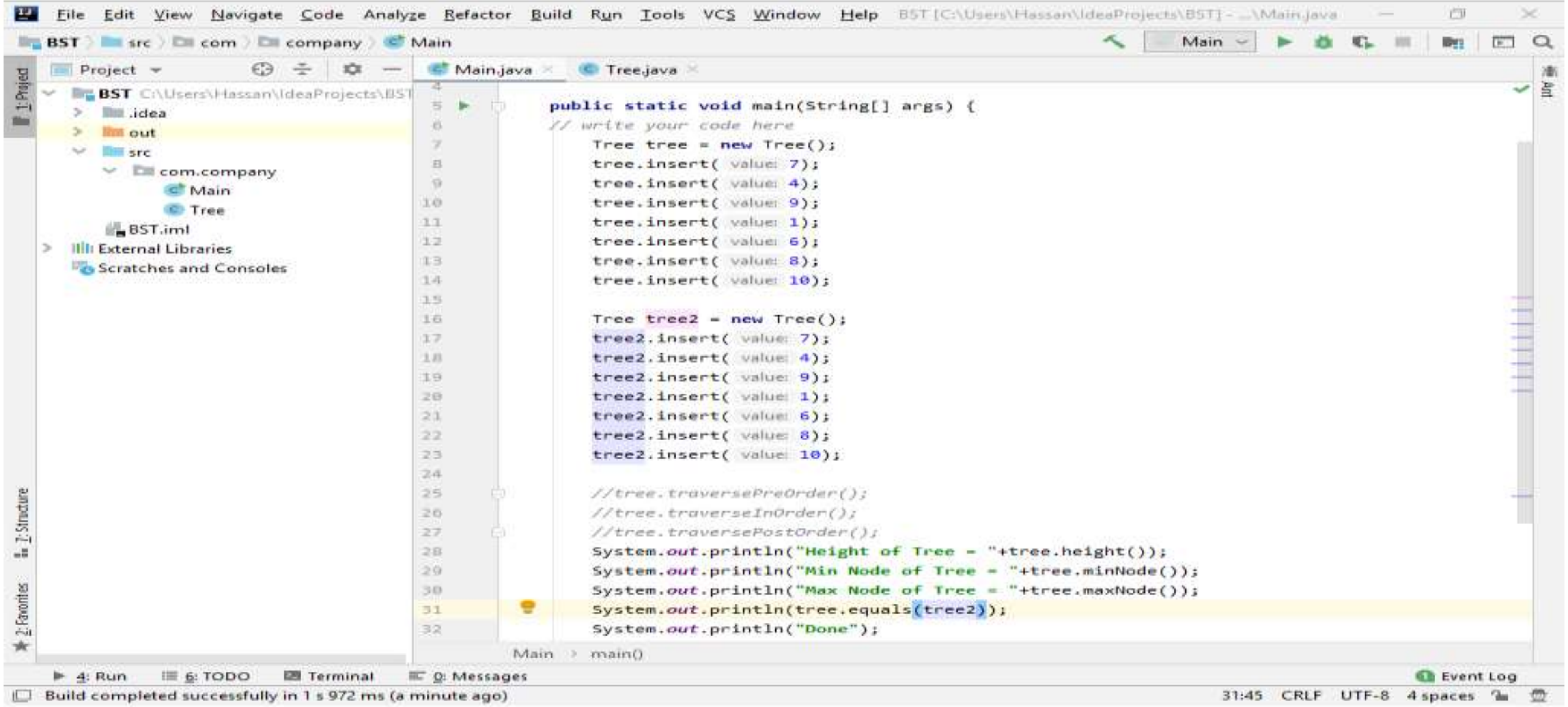
Implementation To Find Equality of BST





```java
        while (current!=null){
            last = current;
            current = current.leftChild;
        }
        return last.value;
    }


    public boolean equals(Tree other){
        if(other == null)
            return false;
        return equals(root,other.root);
    }

    private boolean equals(Node first,Node second){
        if(first == null && second == null)
            return true;
        // Using Pre Order Traversal
        if(first != null && second != null)
            return first.value == second.value && equals(first.leftChild,second.leftChild)
            && equals(first.rightChild,second.rightChild);

        return false;
    }




    static final Node DELIMITER = null;
```

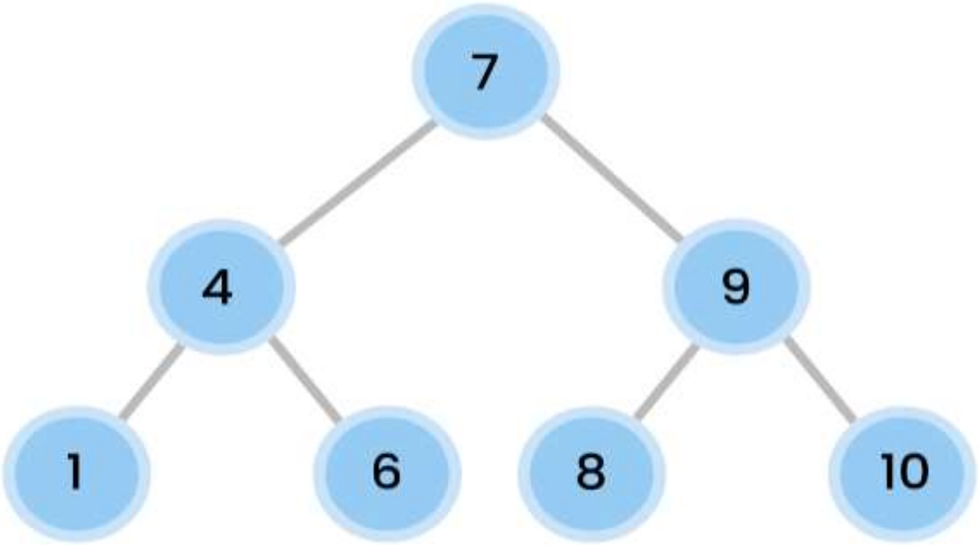
Build completed successfully in 1 s 972 ms (a minute ago)



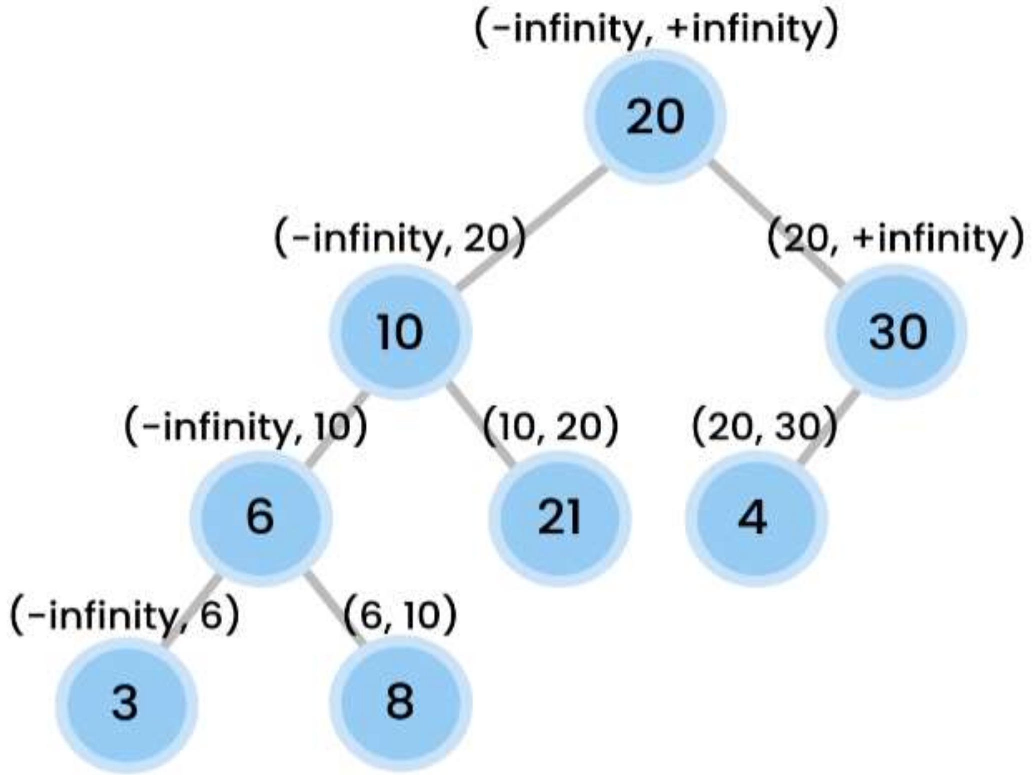
```
    public static void main(String[] args) {
    // write your code here
        Tree tree = new Tree();
        tree.insert( value: 7);
        tree.insert( value: 4);
        tree.insert( value: 9);
        tree.insert( value: 1);
        tree.insert( value: 6);
        tree.insert( value: 8);
        tree.insert( value: 10);

        Tree tree2 = new Tree();
        tree2.insert( value: 7);
        tree2.insert( value: 4);
        tree2.insert( value: 9);
        tree2.insert( value: 1);
        tree2.insert( value: 6);
        tree2.insert( value: 8);
        tree2.insert( value: 10);

        //tree.traversePreOrder();
        //tree.traverseInOrder();
        //tree.traversePostOrder();
        System.out.println("Height of Tree - "+tree.height());
        System.out.println("Min Node of Tree = "+tree.minNode());
        System.out.println("Max Node of Tree = "+tree.maxNode());
        System.out.println(tree.equals(tree2));
        System.out.println("Done");
```

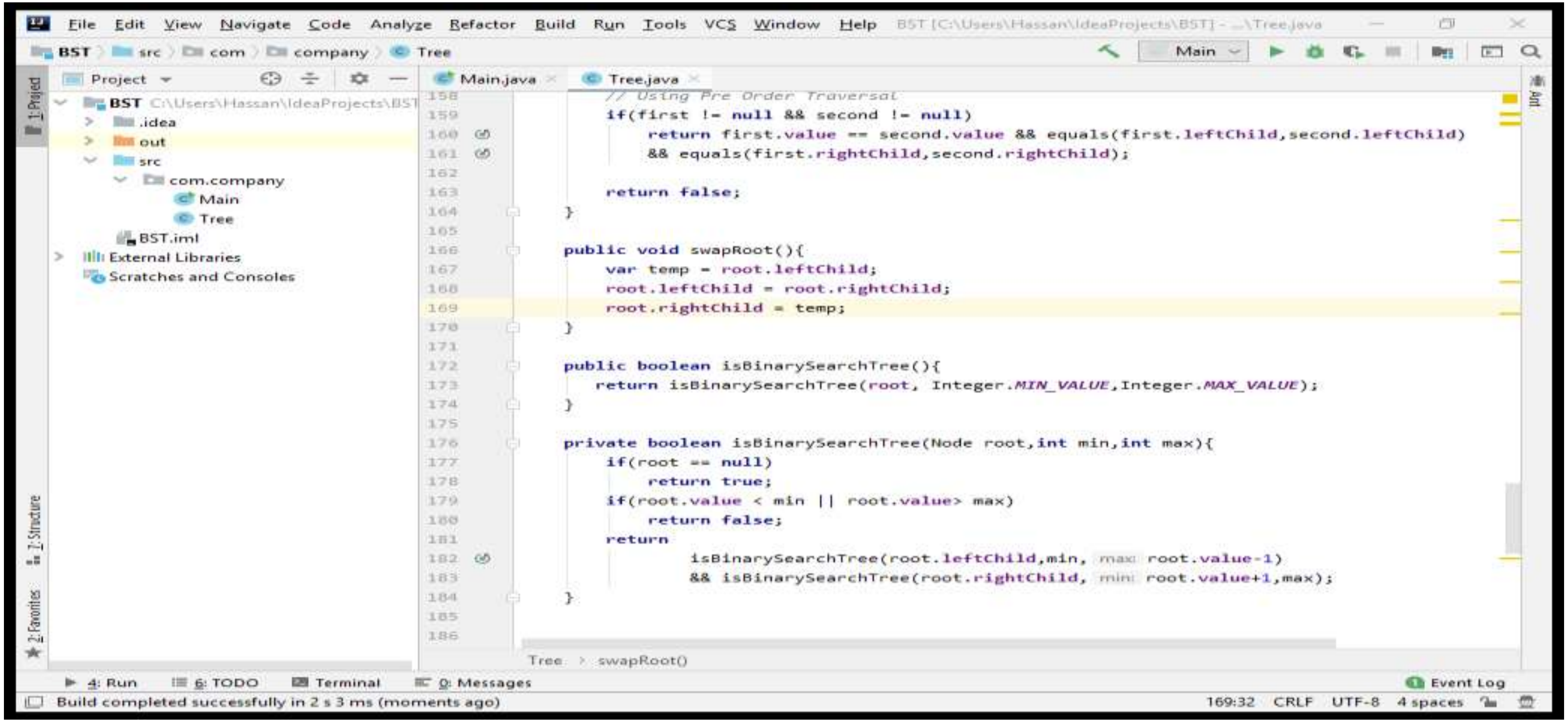
Build completed successfully in 1 s 972 ms (a minute ago)

Validating BST



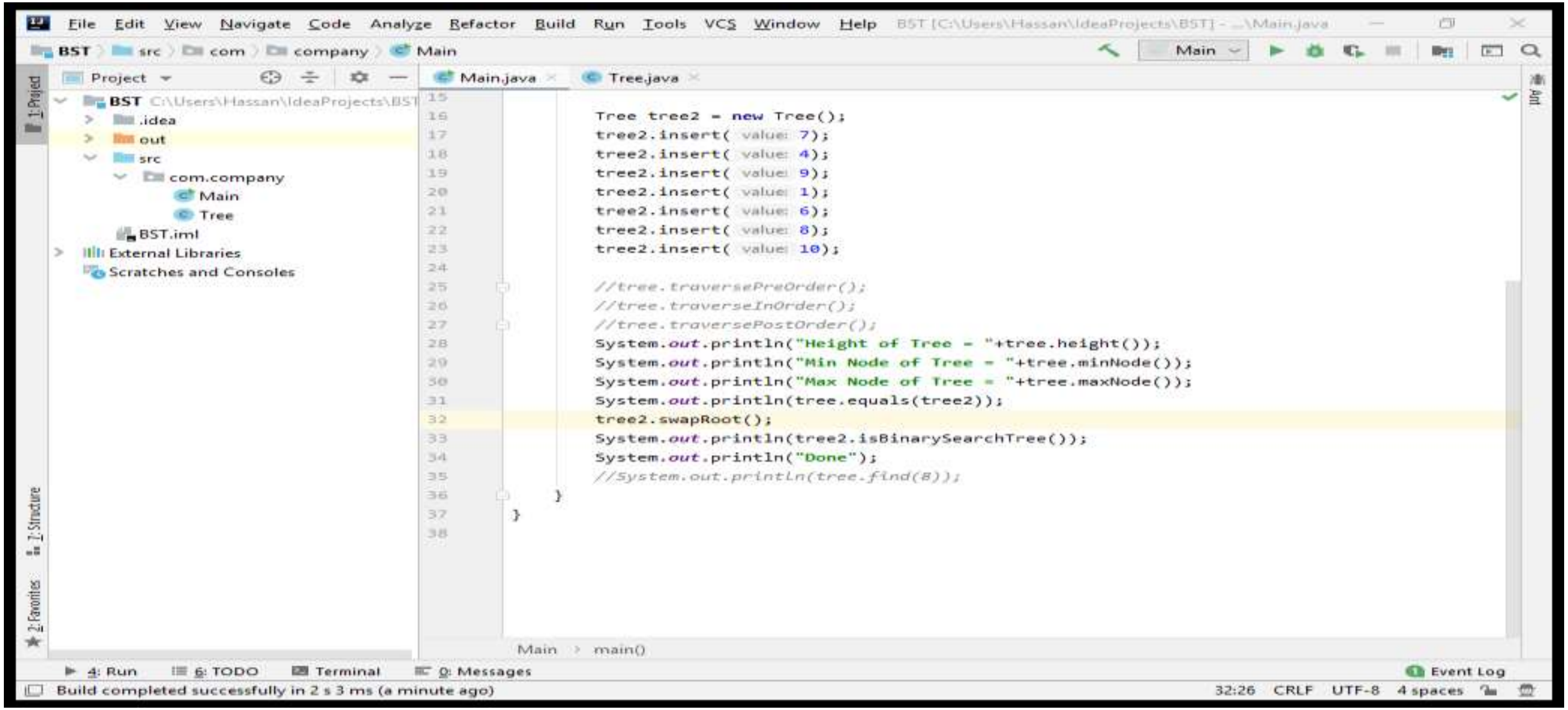
Validating By Checking Range of Nodes





```java
        // Using Pre Order Traversal
        if(first != null && second != null)
            return first.value == second.value && equals(first.leftChild,second.leftChild)
            && equals(first.rightChild,second.rightChild);

        return false;
    }

    public void swapRoot(){
        var temp = root.leftChild;
        root.leftChild = root.rightChild;
        root.rightChild = temp;
    }

    public boolean isBinarySearchTree(){
       return isBinarySearchTree(root, Integer.MIN_VALUE,Integer.MAX_VALUE);
    }

    private boolean isBinarySearchTree(Node root,int min,int max){
        if(root == null)
            return true;
        if(root.value < min || root.value> max)
            return false;
        return
                isBinarySearchTree(root.leftChild,min, max: root.value-1)
                && isBinarySearchTree(root.rightChild, min: root.value+1,max);
    }
```

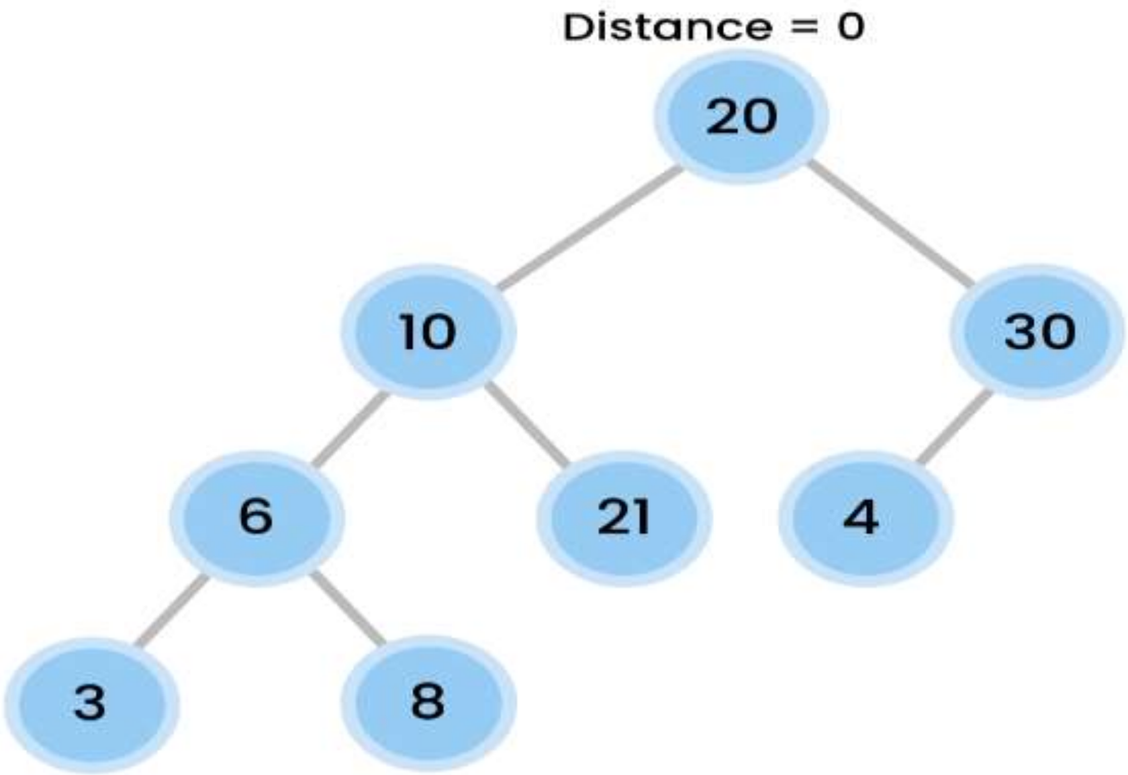
Build completed successfully in 2 s 3 ms (moments ago)



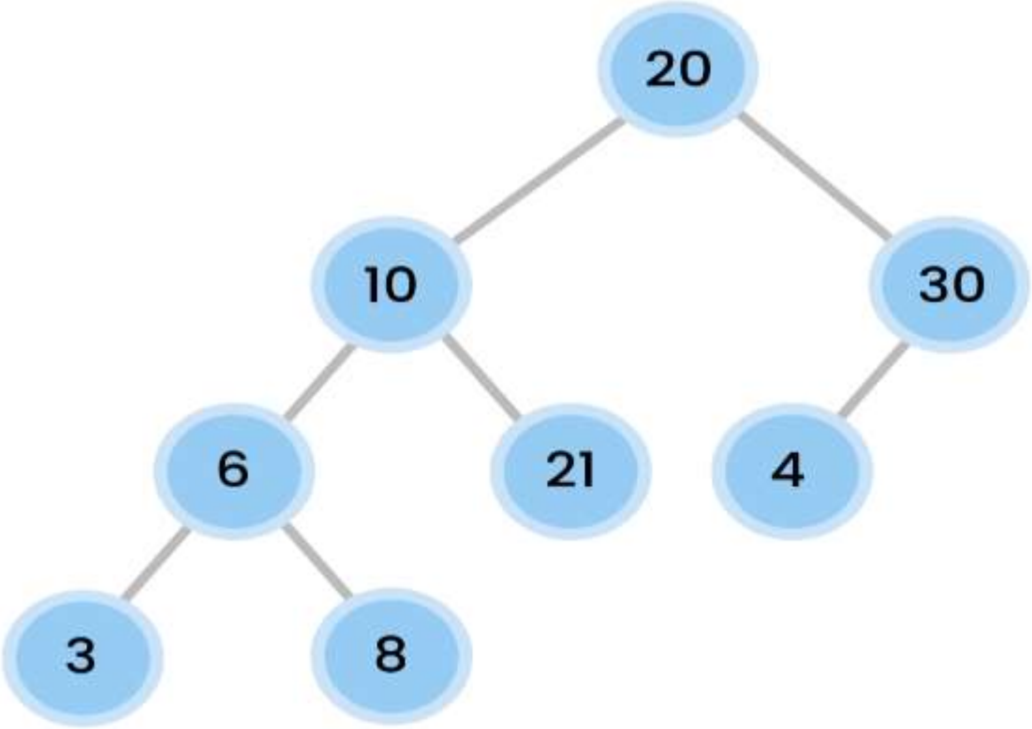
```java
        Tree tree2 = new Tree();
        tree2.insert( value: 7);
        tree2.insert( value: 4);
        tree2.insert( value: 9);
        tree2.insert( value: 1);
        tree2.insert( value: 6);
        tree2.insert( value: 8);
        tree2.insert( value: 10);

        //tree.traversePreOrder();
        //tree.traverseInOrder();
        //tree.traversePostOrder();
        System.out.println("Height of Tree = "+tree.height());
        System.out.println("Min Node of Tree = "+tree.minNode());
        System.out.println("Max Node of Tree = "+tree.maxNode());
        System.out.println(tree.equals(tree2));
        tree2.swapRoot();
        System.out.println(tree2.isBinarySearchTree());
        System.out.println("Done");
        //System.out.println(tree.find(8));
    }
}
```

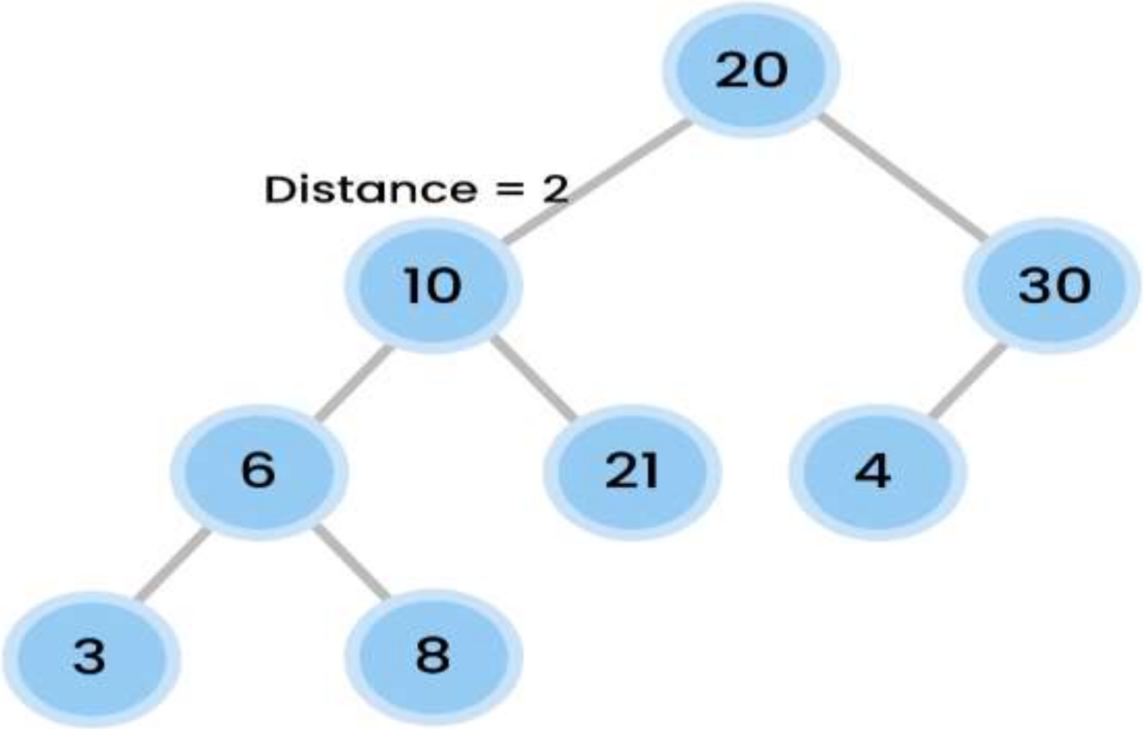
Build completed successfully in 2 s 3 ms (a minute ago)

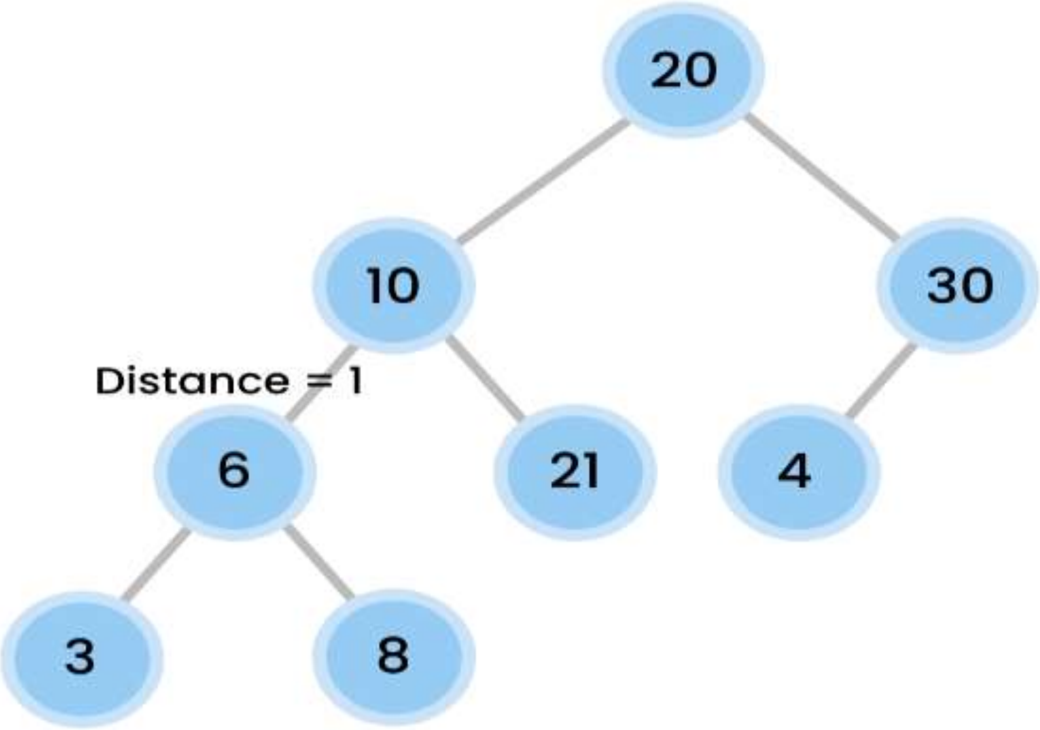
Nodes at K Distance from the Root Node

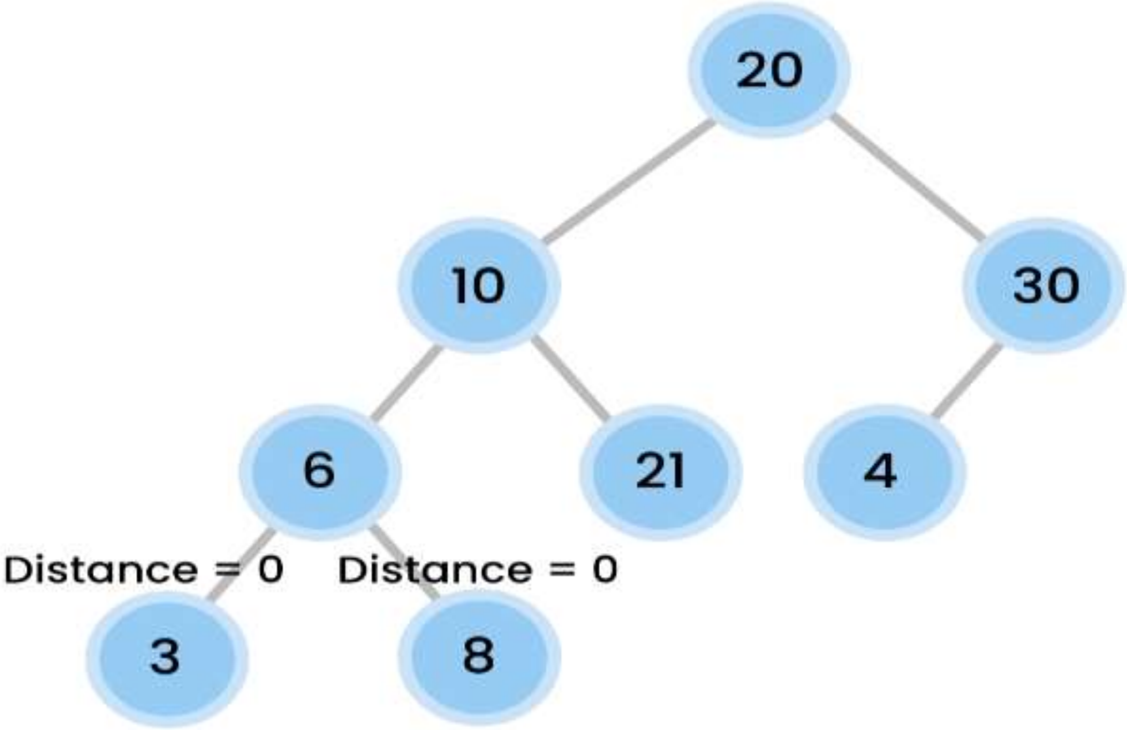


Distance = 3

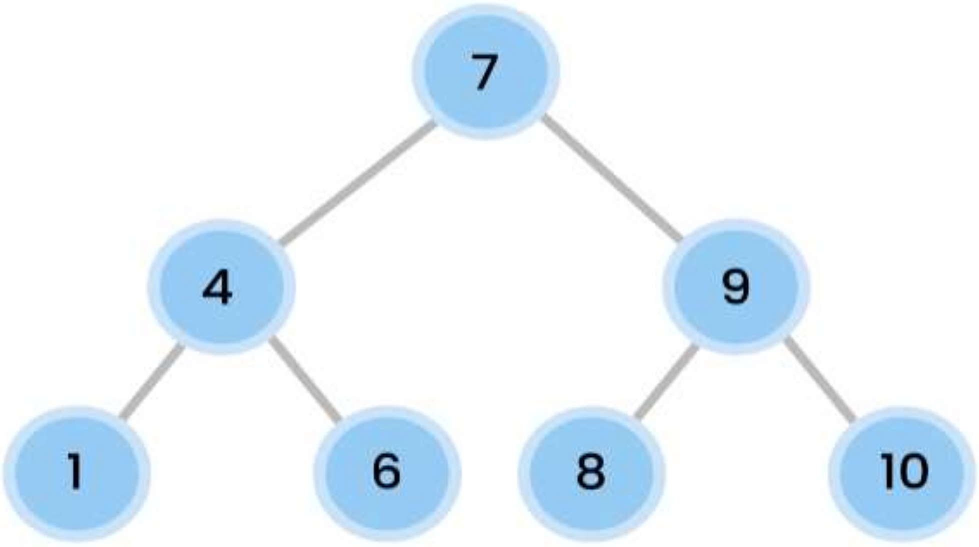


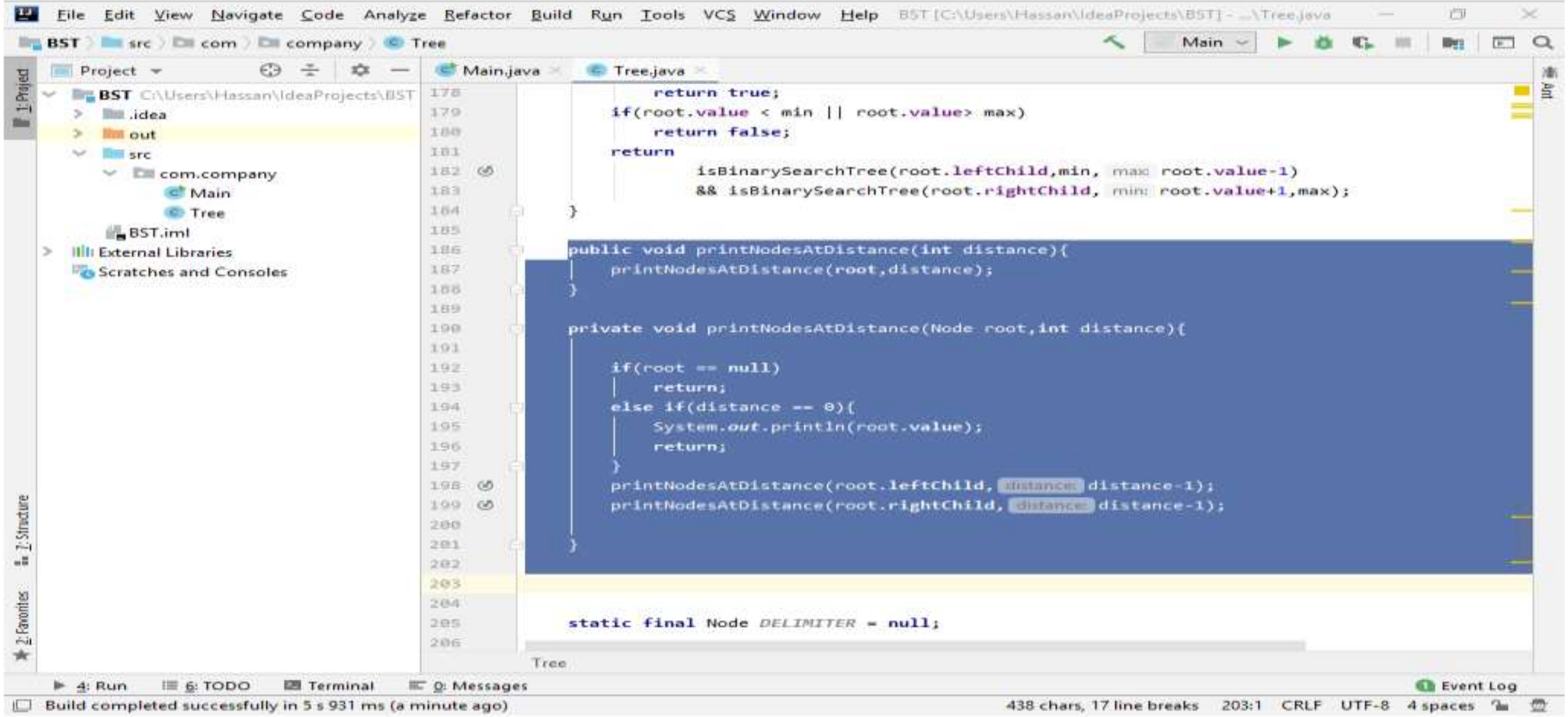

20
Distance = 2
10
30
6
21
4
3
8


20
10
30
Distance = 1
6
21
4
3
8


20
10
30
6
21
4
Distance = 0
Distance = 0
3
8

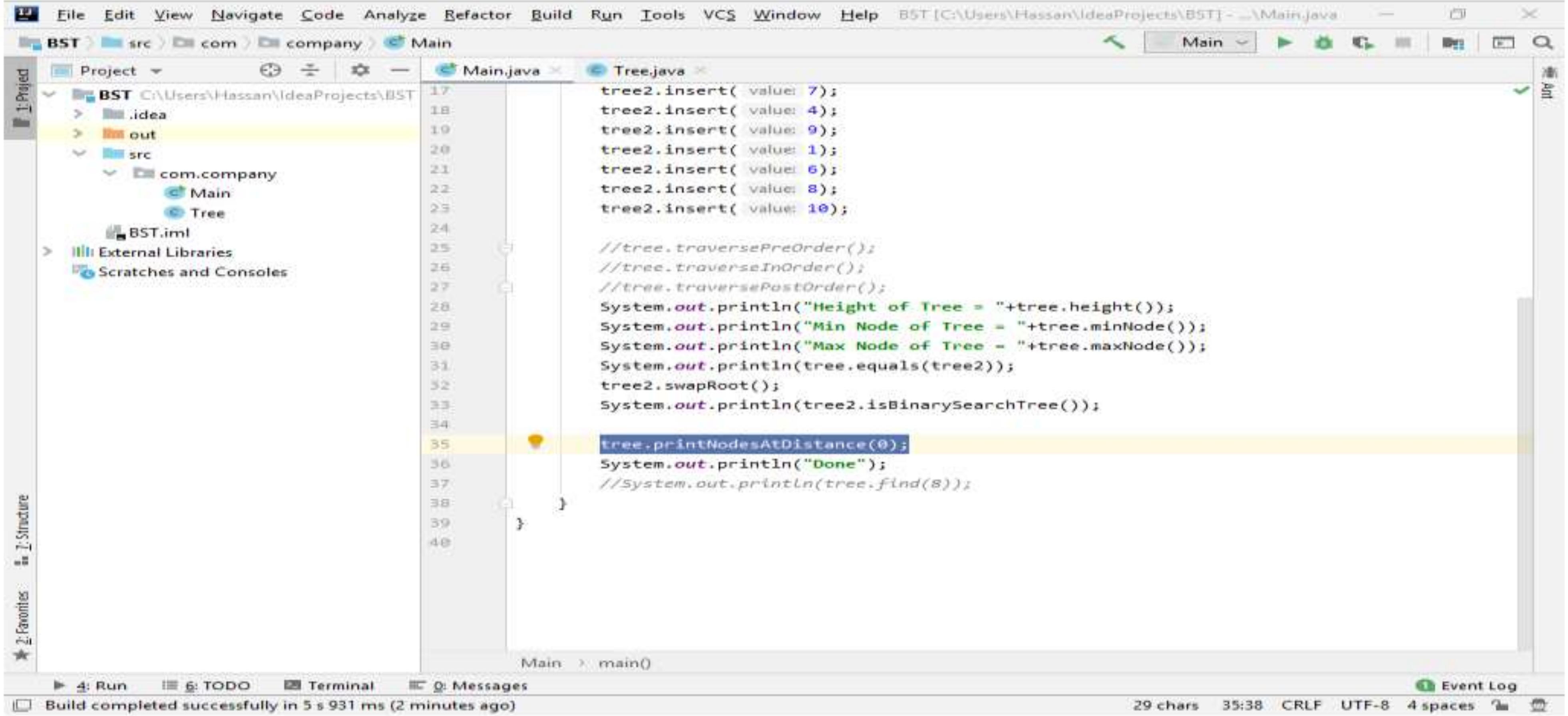
Implementation To Find K Distance of Nodes In Following BST





```java
            return true;
        if(root.value < min || root.value> max)
            return false;
        return
                isBinarySearchTree(root.leftChild,min, max: root.value-1)
                && isBinarySearchTree(root.rightChild, min: root.value+1,max);
    }

    public void printNodesAtDistance(int distance){
        printNodesAtDistance(root,distance);
    }

    private void printNodesAtDistance(Node root,int distance){

        if(root == null)
            return;
        else if(distance == 0){
            System.out.println(root.value);
            return;
        }
        printNodesAtDistance(root.leftChild, distance: distance-1);
        printNodesAtDistance(root.rightChild, distance: distance-1);

    }


    static final Node DELIMITER = null;
```

Build completed successfully in 5 s 931 ms (a minute ago)



```java
        tree2.insert( value: 7);
        tree2.insert( value: 4);
        tree2.insert( value: 9);
        tree2.insert( value: 1);
        tree2.insert( value: 6);
        tree2.insert( value: 8);
        tree2.insert( value: 10);

        //tree.traversePreOrder();
        //tree.traverseInOrder();
        //tree.traversePostOrder();
        System.out.println("Height of Tree = "+tree.height());
        System.out.println("Min Node of Tree = "+tree.minNode());
        System.out.println("Max Node of Tree = "+tree.maxNode());
        System.out.println(tree.equals(tree2));
        tree2.swapRoot();
        System.out.println(tree2.isBinarySearchTree());

        tree.printNodesAtDistance(0);
        System.out.println("Done");
        //System.out.println(tree.find(8));
    }
}
```

Build completed successfully in 5 s 931 ms (2 minutes ago)