**Solution 2:**

**Java program to find whether given Binary tree is Binary search tree or not:**

class Node

{

int value;

Node left, right;

public Node(int nodeValue)

{

value = nodeValue;

left = null;

right = null;

}

}

public class BinaryTree

{

Node root;

Node prev;

boolean isBinarySearchTree() {

prev = null;

return binarySearch(root);

}

boolean binarySearch(Node node)

{

if (node != null)

{

if (!binarySearch(node.left))

return false;

if (prev != null && node.value <= prev.value )

return false;

prev = node;

return binarySearch(node.right);

}

return true;

}

public static void main(String args[])

{

BinaryTree BST = new BinaryTree();

BST.root = new Node(4);

BST.root.left = new Node(2);

BST.root.right = new Node(5);

BST.root.left.left = new Node(1);

BST.root.left.right = new Node(3);

if (BST.isBinarySearchTree())

System.out.println("Is a Binary Tree");

else

System.out.println("Not a Binary Tree");

}

}

**Time Complexity:**

This program performs In-Order Traversal for the given binary tree using recursion. During the traversal we are keeping track of previously visited nodes and comparing its value with the value of the current node. As long as the previous node’s value is greater or equal to the current node’s value throughout the traversal of the tree, the given tree can be deemed as a binary tree. While verifying, each node is being visited only once hence the time complexity will be O(n)

**T(n) = O(n)**