CS162 ASSIGNMENT 12

NAME:

ARCHIT AGRAWAL

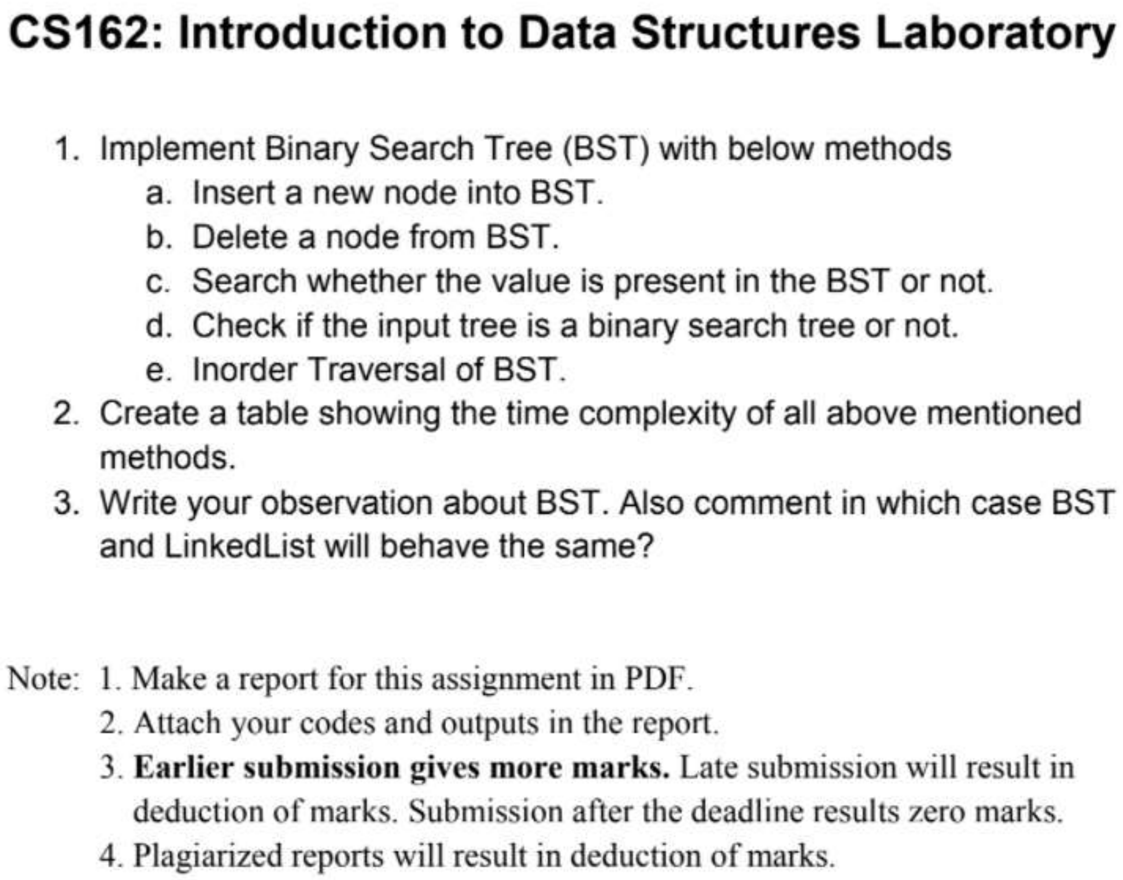
ROLL NO. :

202052307

SECTION:

A

Question



CS162: Introduction to Data Structures Laboratory

1. Implement Binary Search Tree (BST) with below methods
    a. Insert a new node into BST.
    b. Delete a node from BST.
    c. Search whether the value is present in the BST or not.
    d. Check if the input tree is a binary search tree or not.
    e. Inorder Traversal of BST.
2. Create a table showing the time complexity of all above mentioned methods.
3. Write your observation about BST. Also comment in which case BST and LinkedList will behave the same?

Note: 1. Make a report for this assignment in PDF.
2. Attach your codes and outputs in the report.
3. **Earlier submission gives more marks.** Late submission will result in deduction of marks. Submission after the deadline results zero marks.
4. Plagiarized reports will result in deduction of marks.

CODE

```
//Archit Agrawal
//202052307

/*
This code contains the following classes:
    1. Node class ---> contains the implementation of Node
    2. BinaryTree class ---> implementation and methods for Binary Tree
    3. BinarySearchTree class ---
> implementation and methods for Binary Search tree
    4. Main Class -----
> It is just used for input/output purposes and to demonstrate the methods use
d in Binary Search Tree class

*/

import java.io.*;
import java.util.*;

class Node {
    int data;
    Node left;
    Node right;

    public Node(int element){
        this.data = element;
        this.left = null;
        this.right = null;
    }
}

class BinaryTree{

    Node root;

    public BinaryTree(){
        root = null;
    }

    public BinaryTree(int key){
        root = new Node(key);
    }
```

```java
    public Node constructLevelOrder(int []arr, Node root, int index){
        if(index < arr.length){
            Node temp =  new Node(arr[index]);
            root = temp;

            root.left = constructLevelOrder(arr, root.left, 2 * index + 1);
            root.right = constructLevelOrder(arr, root.right, 2 * index + 2);
        }

        return root;
    }

    public void preOrderTraversal(Node node){
        if(node == null) return;
        //pre-order is node-left-right(NLR) traversal
        System.out.print(node.data + " ");
        preOrderTraversal(node.left);
        preOrderTraversal(node.right);
    }

    public void inOrderTraversal(Node node){
        if(node == null) return;
        //in-order is left-node-right(LNR) traversal
        inOrderTraversal(node.left);
        System.out.print(node.data + " ");
        inOrderTraversal(node.right);
    }

    public void postOrderTraversal(Node node){
        if(node == null) return;
        //post-order is left-right-node(LRN) traversal
        postOrderTraversal(node.left);
        postOrderTraversal(node.right);
        System.out.print(node.data + " ");
    }

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

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

    public void postOrderTraversal(){
        postOrderTraversal(root);
    }
```

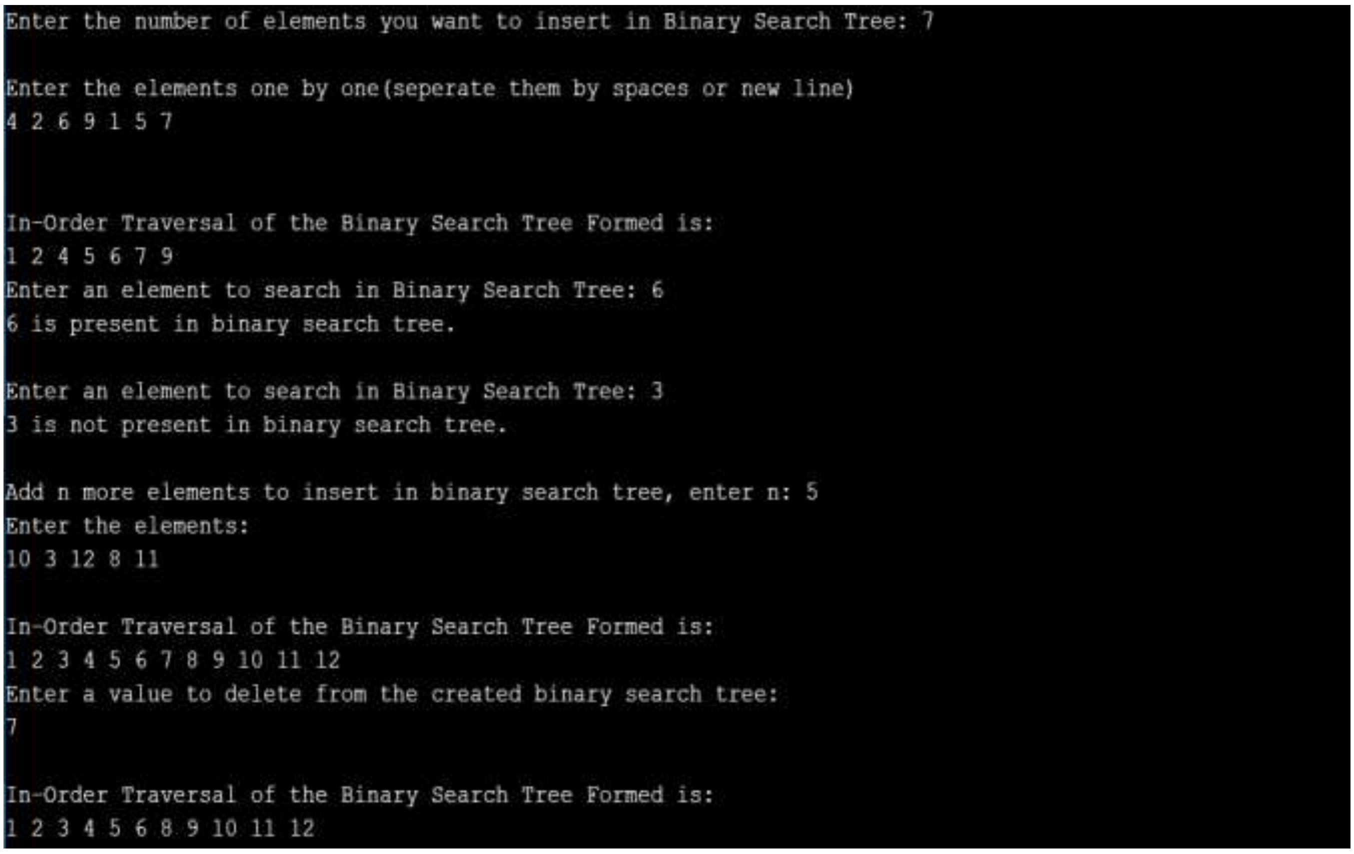
```java
}


class BinarySearchTree{
    Node root;

    public BinarySearchTree(){
        root = null;
    }

    public Node insertNode(Node root, int element){

        if(root == null){
            root = new Node(element);
            return root;
        }

        if(element < root.data){
            root.left = insertNode(root.left, element);
        } else if(element > root.data){
            root.right = insertNode(root.right, element);
        }

        return root;
    }

    public void insert(int element){
        root = insertNode(root, element);
    }

    public void inOrderTraversal(Node root){

        if(root == null) return;
        inOrderTraversal(root.left);
        System.out.print(root.data + " ");
        inOrderTraversal(root.right);
    }

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

    public Node deleteNode(Node root, int element){

        if(root == null){
            return root;
        }
        if(element < root.data){
            root.left = deleteNode(root.left, element);
        } else if(element > root.data){
            root.right = deleteNode(root.right, element);
        } else {
            if(root.left == null){
                return root.right;
            } else if(root.right == null){
                return root.left;
            }

            root.data = inOrderSuccessor(root.right);

            root.right = deleteNode(root.right, root.data);
        }

        return root;
    }

    public void delete(int element){
        root = deleteNode(root, element);
    }

    public int inOrderSuccessor(Node root){
        int min = root.data;
        while(root.left != null){

            root = root.left;
            min = root.data;
        }

        return min;
    }

    public Node search(Node root, int element){

        if(root == null || root.data == element){
            return root;
        }

        if(element < root.data){
            return search(root.left, element);
        } else {
            return search(root.right, element);
        }
    }
```

```java
public Node search(int element){
    Node temp = search(root, element);
    return temp;
}

public int maximumValue(Node root){

    if(root == null) return Integer.MIN_VALUE;

    int value = root.data;
    int leftValue = maximumValue(root.left);
    int rightValue = maximumValue(root.right);

    if(leftValue > value) value = leftValue;
    if(rightValue > value) value = rightValue;

    return value;
}

public int minimumValue(Node root){

    if(root == null) return Integer.MAX_VALUE;

    int value = root.data;
    int leftValue = minimumValue(root.left);
    int rightValue = minimumValue(root.right);

    if(leftValue < value) value = leftValue;
    if(rightValue < value) value = rightValue;

    return value;
}

public boolean isBST(Node root){

    if(root == null){
        return true;
    }

    if(root.left != null && maximumValue(root.left) > root.data){
        return false;
    }

    if(root.right != null && minimumValue(root.right) < root.data){
        return false;
    }
```

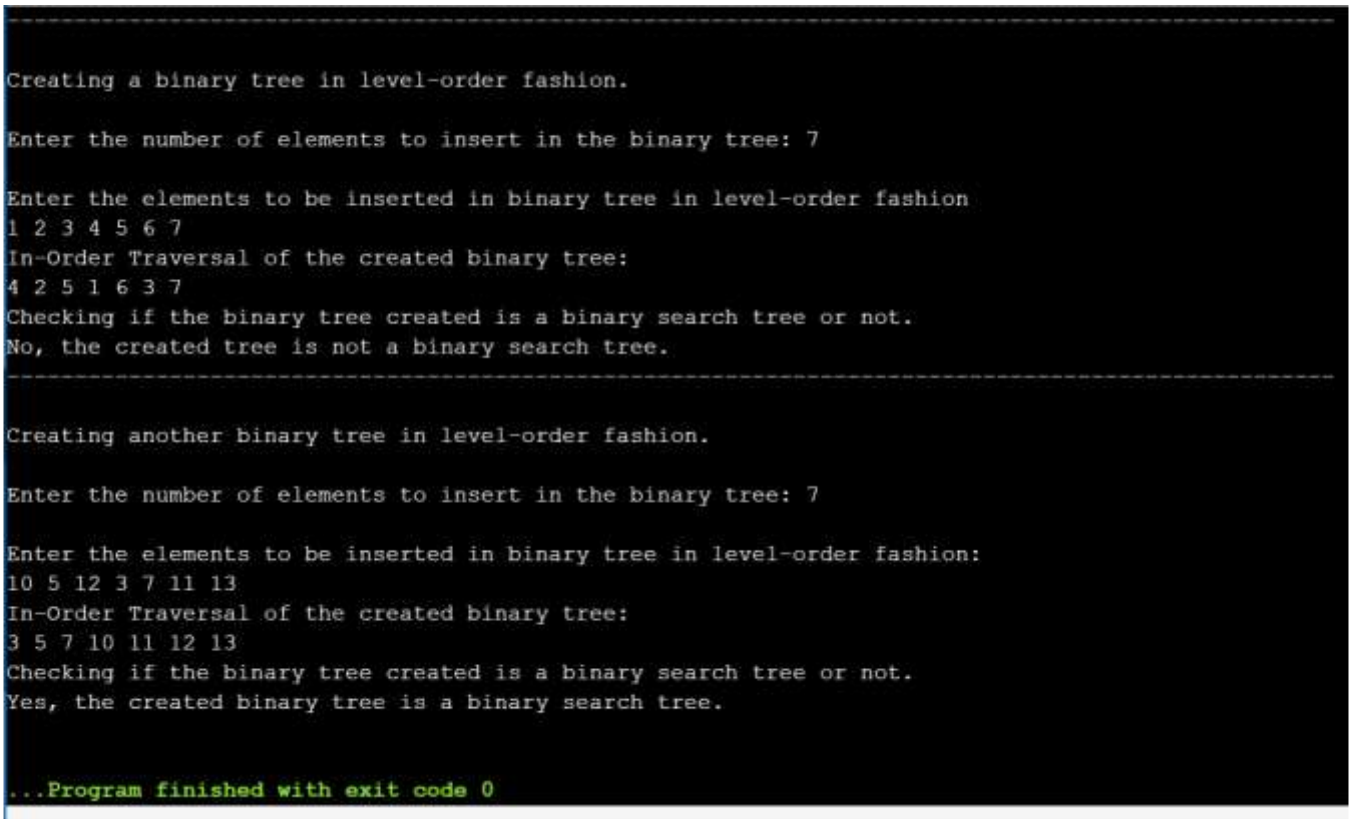
```
        if(!(isBST(root.left) && isBST(root.right))){
            return false;
        }

        return true;
    }

}

public class Main{

    public static void main(String []args){

        Scanner sc = new Scanner(System.in);
        BinarySearchTree bst = new BinarySearchTree();

        System.out.print("Enter the number of elements you want to insert in B
inary Search Tree: ");
        int n = sc.nextInt();
        System.out.println();
        System.out.println("Enter the elements one by one(seperate them by spa
ces or new line)");

        for(int i = 0; i < n; i++){
            int x = sc.nextInt();
            bst.insert(x);
        }

        System.out.println();
        System.out.println();
        System.out.println("In-
Order Traversal of the Binary Search Tree Formed is: ");
        bst.inOrderTraversal();

        System.out.println();
        for(int i = 0; i < 2; i++){
            System.out.print("Enter an element to search in Binary Search Tree
: ");
            int x = sc.nextInt();

            if(bst.search(x) != null){
                System.out.println(x + " is present in binary search tree.");
            } else {
                System.out.println(x + " is not present in binary search tree.
");
            }
            System.out.println();
        }

        System.out.print("Add n more elements to insert in binary search tree,
 enter n: ");
        n = sc.nextInt();

        System.out.println("Enter the elements:");
        for(int i = 0; i < n; i++){
            int x = sc.nextInt();
            bst.insert(x);
        }

        System.out.println();
        System.out.println("In-
Order Traversal of the Binary Search Tree Formed is: ");
        bst.inOrderTraversal();
        System.out.println();

        System.out.println("Enter a value to delete from the created binary se
arch tree: ");
        int x = sc.nextInt();

        System.out.println();
        bst.delete(x);
        System.out.println("In-
Order Traversal of the Binary Search Tree Formed is: ");
        bst.inOrderTraversal();

        System.out.println();
        System.out.println("----------------------------------------------------
------------------------------------------------");
        System.out.println();

        System.out.println("Creating a binary tree in level-order fashion.");

        BinaryTree tree = new BinaryTree();

        System.out.println();
        System.out.print("Enter the number of elements to insert in the binary
 tree: ");
        n = sc.nextInt();
        System.out.println();
        int []arr = new int[n];
        System.out.println("Enter the elements to be inserted in binary tree i
n level-order fashion");
```

```
        for(int i = 0; i < n; i++){
            arr[i] = sc.nextInt();
        }

        tree.root = tree.constructLevelOrder(arr, tree.root, 0);
        System.out.println("In-Order Traversal of the created binary tree:");
        tree.inOrderTraversal();
        System.out.println();
        System.out.println("Checking if the binary tree created is a binary se
arch tree or not.");
        if(bst.isBST(tree.root)){
            System.out.println("Yes, the created binary tree is a binary searc
h tree.");
        } else {
            System.out.println("No, the created tree is not a binary search tr
ee.");
        }

        System.out.println("---------------------------------------------------
------------------------------------------------");
        System.out.println();

        System.out.println("Creating another binary tree in level-
order fashion.");

        BinaryTree tree2 = new BinaryTree();

        System.out.println();
        System.out.print("Enter the number of elements to insert in the binary
 tree: ");
        n = sc.nextInt();
        System.out.println();
        int []brr = new int[n];
        System.out.println("Enter the elements to be inserted in binary tree i
n level-order fashion:");
        for(int i = 0; i < n; i++){
            brr[i] = sc.nextInt();
        }

        tree2.root = tree2.constructLevelOrder(brr, tree2.root, 0);
        System.out.println("In-Order Traversal of the created binary tree:");
        tree2.inOrderTraversal();
        System.out.println();
        System.out.println("Checking if the binary tree created is a binary se
arch tree or not.");
        if(bst.isBST(tree2.root)){
            System.out.println("Yes, the created binary tree is a binary searc
h tree.");
        } else {
            System.out.println("No, the created tree is not a binary search tr
ee.");
        }
    }
}
```

OUTPUT



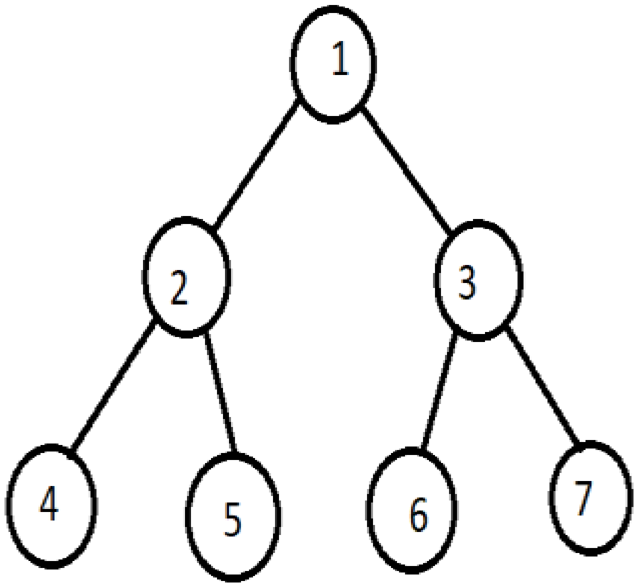
```
Enter the number of elements you want to insert in Binary Search Tree: 7

Enter the elements one by one(seperate them by spaces or new line)
4 2 6 9 1 5 7


In-Order Traversal of the Binary Search Tree Formed is:
1 2 4 5 6 7 9
Enter an element to search in Binary Search Tree: 6
6 is present in binary search tree.

Enter an element to search in Binary Search Tree: 3
3 is not present in binary search tree.

Add n more elements to insert in binary search tree, enter n: 5
Enter the elements:
10 3 12 8 11

In-Order Traversal of the Binary Search Tree Formed is:
1 2 3 4 5 6 7 8 9 10 11 12
Enter a value to delete from the created binary search tree:
7

In-Order Traversal of the Binary Search Tree Formed is:
1 2 3 4 5 6 8 9 10 11 12
```



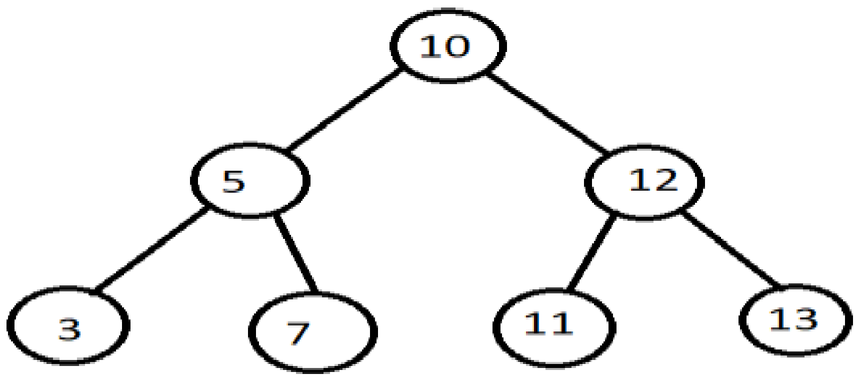
```
--------------------------------------------------------------------------------------------------

Creating a binary tree in level-order fashion.

Enter the number of elements to insert in the binary tree: 7

Enter the elements to be inserted in binary tree in level-order fashion
1 2 3 4 5 6 7
In-Order Traversal of the created binary tree:
4 2 5 1 6 3 7
Checking if the binary tree created is a binary search tree or not.
No, the created tree is not a binary search tree.
--------------------------------------------------------------------------------------------------

Creating another binary tree in level-order fashion.

Enter the number of elements to insert in the binary tree: 7

Enter the elements to be inserted in binary tree in level-order fashion:
10 5 12 3 7 11 13
In-Order Traversal of the created binary tree:
3 5 7 10 11 12 13
Checking if the binary tree created is a binary search tree or not.
Yes, the created binary tree is a binary search tree.


...Program finished with exit code 0
```

The two trees that are checked to be BST or not in the example are:

tree



tree2



2. Create a table showing the time complexity of all the methods.

S.No.	***Method***	***Time Complexity***
1.	Insert a new Node in BST	O(h)
2.	Delete a Node from BST	O(h)
3.	Search an element in BST	O(h)
4.	Check if input tree is BST or not	O(n^2)
5.	In-Order Traversal of BST	O(n)

where h is the height of the tree and n is number of elements in the tree.

3. Write your observation about BST. Also comment in which case BST and Linked List will behave the same?

A binary search tree, also called a ordered or sorted binary tree, is a type of data structure for storing data such as numbers in an organized way.

- In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
- Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.

Advantages of using Binary Search Trees (BST):

- Searching become very efficient in a binary search tree since, we get a hint at each step, about which sub-tree contains the desired element.
- The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step. Searching for an element in a binary search tree takes o(log2n) time. In worst case, the time it takes to search an element is 0(n).

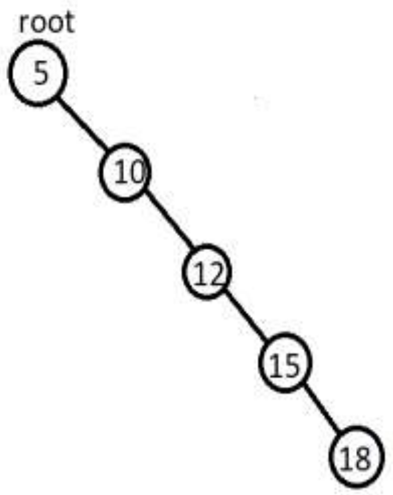
Observations:

- The in-order traversal of a BST is always sorted and the vice-verse i.e. a tree whose in-order traversal is sorted is a Binary Search Tree (BST).
- Unlike Heaps, the root node of the BST is fixed, that is the very first inserted node will be the root till the end if only insertions are done (no deletion took place). Whereas in case of heaps, the root node changes according to its type (maxheap or minheap).

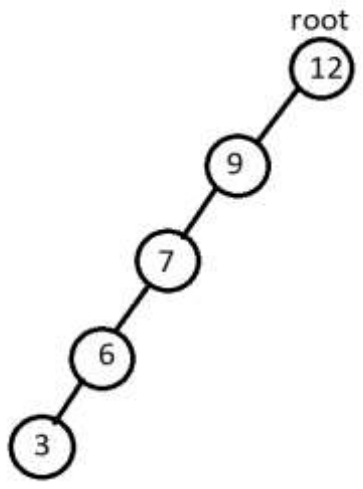
In which case BST and LinkedList will behave the same?

Since, the root node of a BST does not change and it has lesser elements in the left subtree and greater in the right, if the insertions made in the BST are in either ascending order or in descending order, then BST will behave like a singly linked list.

Let's look at an example. Suppose, I want the elements {5, 10, 12, 15, 18} in the BST. Now, let us look how will the BST be if the elements are inserted in the same order as mentioned.



This is how the BST will look like. It is skewed i.e. every node has either none or one child. If we take node as the head, then it can behave as a singly linked list. Similarly, if we insert the elements {12, 9, 7, 6, 3} in the same order as mentioned, the BST will be:



Again, we can see that the BST can behave like a linked list if its root node is considered as head.

If we try to compare a BST with a doubly linked list as both of them have pointers to two nodes, it will still not be same. This is because a double linked list allows us to access its elements in any direction but the binary search tree doesn't allow this. In BST, we can only start accessing elements starting from the root node and moving to the leaves.

Since a linked list contains only the pointer to the next node, only a skewed binary search tree behaves similarly like a sorted singly linked list. Hence, a left skewed BST, as in the latter case, it can behave as a linked list sorted in descending order and a right skewed BST, the former one, behaves as an linked list sorted in ascending order.