Student Name: _____ Roll No: _____ Section: _____

Experiment No. 11

Lab 11 – Binary Search Trees BST, insertion in BST, deletion in BST and applications using JAVA.

In this lab, we will learn about the constructing of Binary Search Tree, operations we can perform on BST and its applications.

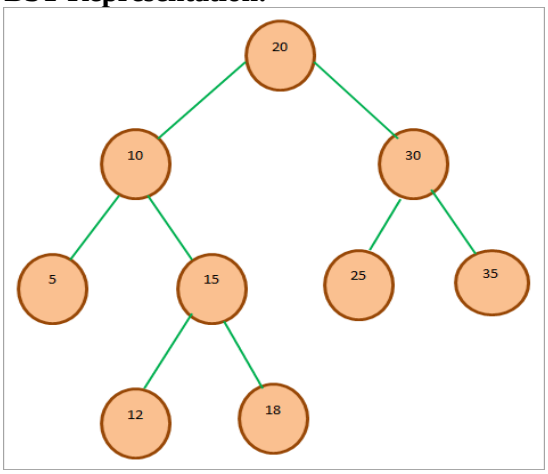
Binary Search Tree (BST)

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.
- There must be no duplicate nodes.

The above properties of Binary Search Tree provides an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search for a given key. A BST does not allow duplicate nodes.

The below diagram shows a BST Representation:



Above shown is a sample BST. We see that 20 is the root node of this tree. The left subtree has all the node values that are less than 20. The right subtree has all the nodes that are greater than 20. We can say that the above tree fulfills the BST properties.

The BST data structure is considered to be very efficient when compared to Arrays and Linked list when it comes to insertion/deletion and searching of items.

Student Name: _____ Roll No: _____ Section: _____

BST takes O (log n) time to search for an element. As elements are ordered, half the subtree is discarded at every step while searching for an element. This becomes possible because we can easily determine the rough location of the element to be searched.

Similarly, insertion and deletion operations are more efficient in BST. When we want to insert a new element, we roughly know in which subtree (left or right) we will insert the element.

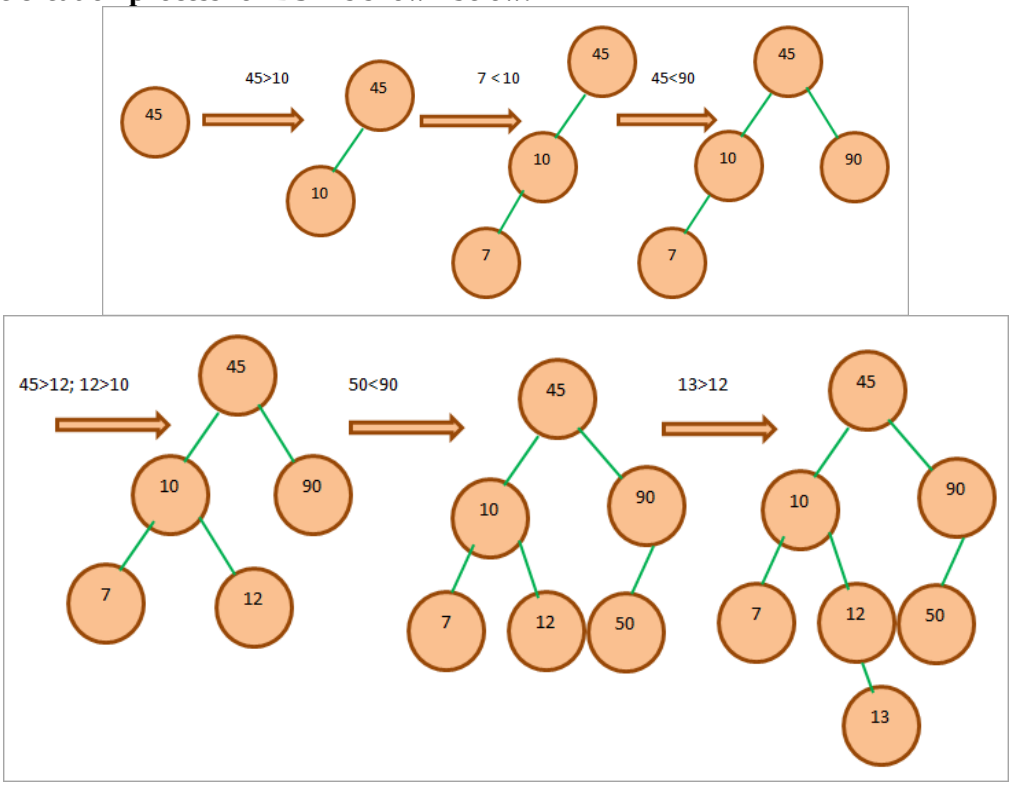
Creating A Binary Search Tree (BST)

Given an array of elements, we need to construct a BST.

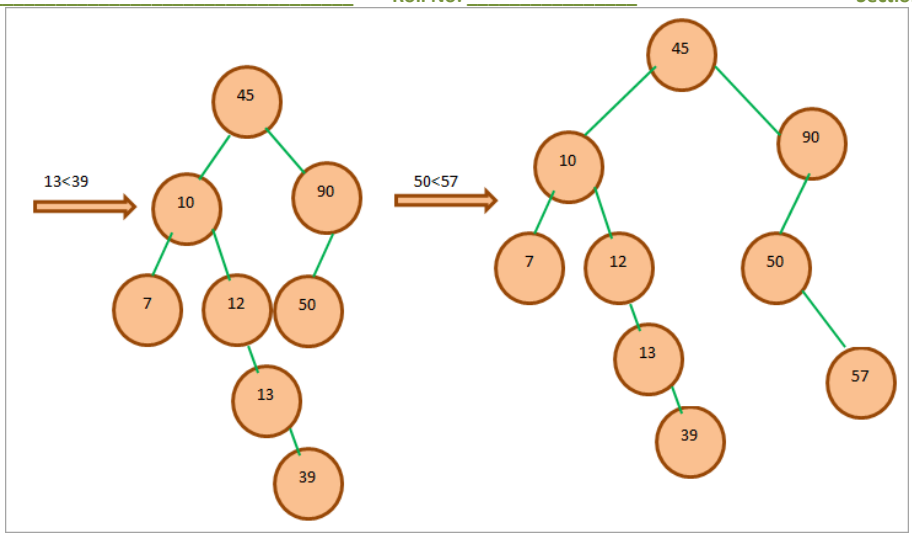
Given array: 45, 10, 7, 90, 12, 50, 13, 39, 57
Let's first consider the top element i.e. 45 as the root node. From here we will go on creating the BST by considering the properties already discussed.

To create a tree, we will compare each element in the array with the root. Then we will place the element at an appropriate position in the tree.

The entire creation process for BST is shown below.



Student Name: _____ Roll No: _____ Section: _____



Completed BST

Binary Search Tree Operations
BST supports various operations. The following table shows the methods supported by BST in Java. We will discuss each of these methods separately.

Method/operation	Description
Insert	Add an element to the BST by not violating the BST properties.
Delete	Remove a given node from the BST. The node can be the root node, non-leaf, or leaf node.
Search	Search the location of the given element in the BST. This operation checks if the tree contains the specified key.

Insert An Element In BST
An element is always inserted as a leaf node in BST.

Given below are the steps for inserting an element.

1. Start from the root.
2. Compare the element to be inserted with the root node. If it is less than root, then traverse the left subtree or traverse the right subtree.
3. Traverse the subtree till the end of the desired subtree. Insert the node in the appropriate subtree as a leaf node.

Student Name: _____ Roll No: _____ Section: _____

Consider the following BST and let us insert element 2 in the tree.

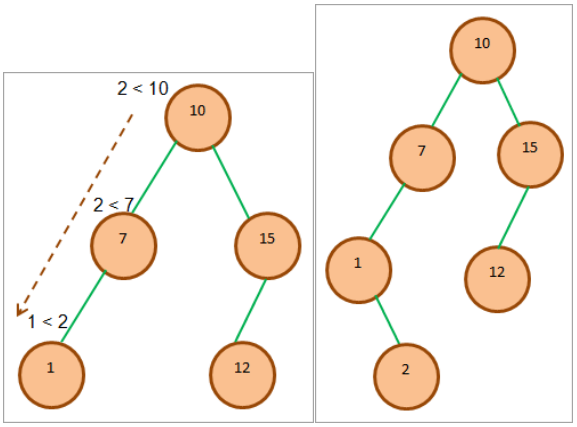


Figure 1 ***Figure 2***

The insert operation for BST is shown above. In fig (1), we show the path that we traverse to insert element 2 in the BST. We have also shown the conditions that are checked at each node. As a result of the recursive comparison, element 2 is inserted as the right child of 1 as shown in fig (2).

Search Operation In BST

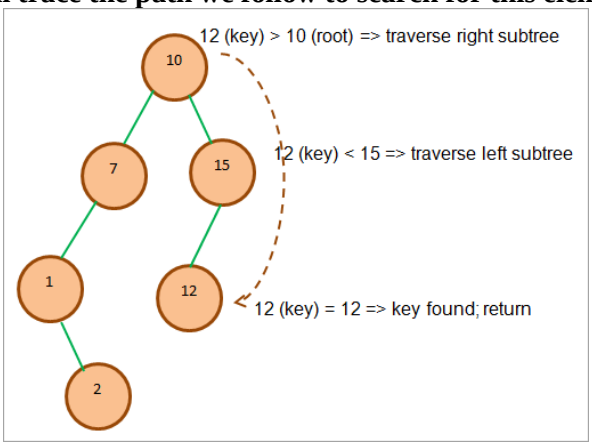
To search if an element is present in the BST, we again start from the root and then traverse the left or right subtree depending on whether the element to be searched is less than or greater than the root.

Enlisted below are the steps that we have to follow.

1. Compare the element to be searched with the root node.
2. If the key (element to be searched) = root, return root node.
3. Else if key < root, traverse the left subtree.
4. Else traverse right subtree.
5. Repetitively compare subtree elements until the key is found or the end of the tree is reached.

Let's illustrate the search operation with an example. Consider that we have to search the key = 12.

In the below figure, we will trace the path we follow to search for this element.



Student Name: _____ Roll No: _____ Section: _____

As shown in the above figure, we first compare the key with root. Since the key is greater, we traverse the right subtree. In the right subtree, we again compare the key with the first node in the right subtree.

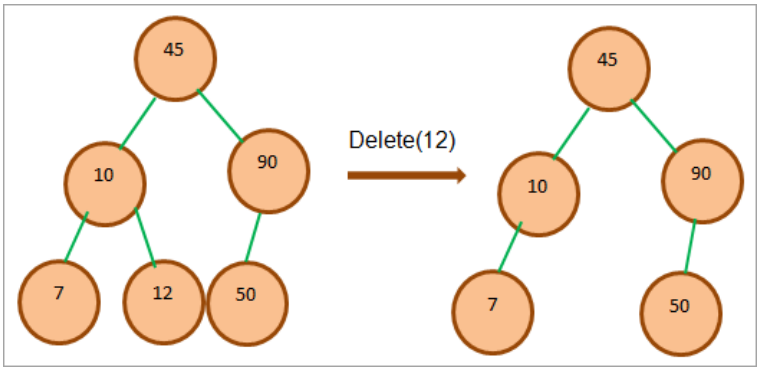
We find that the key is less than 15. So we move to the left subtree of node 15. The immediate left node of 15 is 12 that matches the key. At this point, we stop the search and return the result.

Remove Element from The BST

When we delete a node from the BST, then there are three possibilities as discussed below:

Node Is A Leaf Node

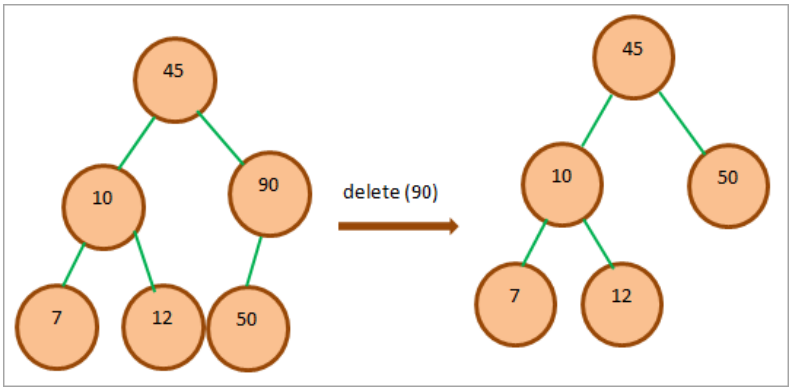
If a node to be deleted is a leaf node, then we can directly delete this node as it has no child nodes. This is shown in the below image.



As shown above, the node 12 is a leaf node and can be deleted straight away.

Node Has Only One Child

When we need to delete the node that has one child, then we copy the value of the child in the node and then delete the child.

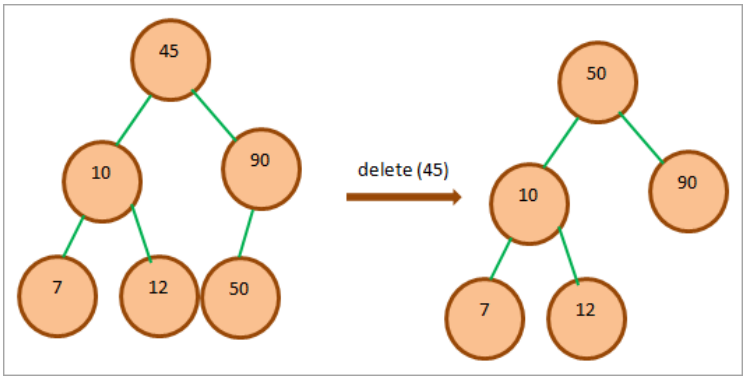


In the above diagram, we want to delete node 90 which has one child 50. So we swap the value 50 with 90 and then delete node 90 which is a child node now.

Student Name: _____ Roll No: _____ Section: _____

Node Has Two Children

When a node to be deleted has two children, then we replace the node with the inorder (left-root-right) successor of the node or simply said the minimum node in the right subtree if the right subtree of the node is not empty. We replace the node with this minimum node and delete the node.



In the above diagram, we want to delete node 45 which is the root node of BST. We find that the right subtree of this node is not empty. Then we traverse the right subtree and find that node 50 is the minimum node here. So we replace this value in place of 45 and then delete 45.

If we check the tree, we see that it fulfills the properties of a BST. Thus the node replacement was correct.

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

```
public class BinaryTree {
    static class Node {
        //instance variable of Node class
        public int data;
        public Node left;
        public Node right;

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

    // instance variable of binary tree class
    public Node root;

    // constructor for initialise the root to null BYDEFAULT
    public BinaryTree() {
        this.root = null;
    }

    // method to check the given tree is Binary search tree or not
```

Student Name: _____ Roll No: _____ Section: _____

```
    public boolean isBSTOrNot() {
        return isBSTOrNot(this.root, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }

    private boolean isBSTOrNot(Node root, int minValue, int maxValue) {
        // check for root is not null or not
        if (root == null) {
            return true;
        }
        // check for current node value with left node value and right node value
and recursively check for left sub tree and right sub tree
        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) {
        // Creating the object of BinaryTree class
        BinaryTree bt = new BinaryTree();
        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());

    }
}
```

Output:

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

```
public class BinarySearchTree {

    public class Node {
        //instance variable of Node class
        public int data;
```

Student Name: _____ Roll No: _____ Section: _____

```
        public Node left;
        public Node right;

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

    }
    // instance variable
    public Node root;

    // constructor for initialise the root to null BYDEFAULT
    public BinarySearchTree() {
        this.root = null;
    }

    // insert method to insert the new Date
    public void insert(int newData) {
        this.root = insert(root, newData);
    }

    public Node insert(Node root, int newData) {
        // Base Case: root is null or not
        if (root == null) {
            // Insert the new data, if root is null.
            root = new Node(newData);
            // return the current root to his sub tree
            return root;
        }
        // Here checking for root data is greater or equal to newData or not
        else if (root.data >= newData) {
            // if current root data is greater than the new data then now process
the left sub-tree
            root.left = insert(root.left, newData);
        } else {
            // if current root data is less than the new data then now process
the right sub-tree
            root.right = insert(root.right, newData);
        }
        return root;
    }

    //Traversal
    public void preorder() {
```

Student Name: _____ Roll No: _____ Section: _____

```
        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) {
        // Creating the object of BinarySearchTree class
        BinarySearchTree bst = new BinarySearchTree();
        // call the method insert
        bst.insert(2);
        bst.insert(4);
        bst.insert(1);
        bst.insert(3);
        bst.insert(5);
        bst.preorder();
    }

}
```

Output:

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

```
public class BinarySearchTree {
    public static class Node {
        //instance variable of Node class
        public int data;
        public Node left;
        public Node right;

        //constructor
        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
```

Student Name: _____ Roll No: _____ Section: _____

```
    }

    // instance variable
    public Node root;

    // constructor for initialise the root to null BYDEFAULT
    public BinarySearchTree() {
        this.root = null;
    }

    // insert method to insert the new Date
    public void insert(int newData) {
        this.root = insert(root, newData);
    }

    public Node insert(Node root, int newData) {
        // Base Case: root is null or not
        if (root == null) {
            // Insert the new data, if root is null.
            root = new Node(newData);
            // return the current root to his sub tree
            return root;
        }
        // Here checking for root data is greater or equal to newData or not
        else if (root.data >= newData) {
            // if current root data is greater than the new data then now process
the left sub-tree
            root.left = insert(root.left, newData);
        } else {
            // if current root data is less than the new data then now process
the right sub-tree
            root.right = insert(root.right, newData);
        }
        return root;
    }

    /*
     * Case 1:- No child
     * Case 2:- 1 child
     * case 3:- 2 child
     * */
    public void deleteANode(Node node) {
        deleteNode(this.root, node);
    }

    private Node deleteNode(Node root, Node node) {
        // check for node initially
```

Student Name: _____ Roll No: _____ Section: _____

```
        if (root == null) {
            return null;
        } else if (node.data < root.data) {
            // process the left sub tree
            root.left = deleteNode(root.left, node);
        } else if (node.data > root.data) {
            // process the right sub tree
            root.right = deleteNode(root.right, node);
        } else if(root.data==node.data){
            // case 3: 2 child
            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;
            }
            //case 2: one child
            // case i-> has only left child
            else if (root.left != null) {
                return root.left;
            }
            // case ii-> has only right child
            else if (root.right != null) {
                return root.right;
            }
            //case 1:- no child
            else {
                return null;
            }
        }
        return root;
    }

    // inorder successor of given node
    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){
```

Student Name: _____ Roll No: _____ Section: _____

```
        if(node!=null){
            System.out.print(node.data+" ");
            preorder(node.left);
            preorder(node.right);
        }
    }


    public static void main(String[] args) {
        // Creating the object of BinarySearchTree class
        BinarySearchTree bst = new BinarySearchTree();
        // call the method insert
        bst.insert(8);
        bst.insert(5);
        bst.insert(9);
        bst.insert(3);
        bst.insert(7);
        bst.preorder();
        bst.deleteANode(new Node(9));
        bst.preorder();
    }
}
```

Output:

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

```
public class BinarySearchTree {

    public class Node {
        //instance variable of Node class
        public int data;
        public Node left;
        public Node right;

        //constructor
        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
```

Student Name: _____ Roll No: _____ Section: _____

```
    // instance variable
    public Node root;

    // constructor for initialise the root to null BYDEFAULT
    public BinarySearchTree() {
        this.root = null;
    }

    // insert method to insert the new Date
    public void insert(int newData) {
        this.root = insert(root, newData);
    }

    public Node insert(Node root, int newData) {
        // Base Case: root is null or not
        if (root == null) {
            // Insert the new data, if root is null.
            root = new Node(newData);
            // return the current root to his sub tree
            return root;
        }
        // Here checking for root data is greater or equal to newData or not
        else if (root.data >= newData) {
            // if current root data is greater than the new data then now process
the left sub-tree
            root.left = insert(root.left, newData);
        } else {
            // if current root data is less than the new data then now process
the right sub-tree
            root.right = insert(root.right, newData);
        }
        return root;
    }

    // method for search the data , is data is present or not in the tree ?
    public boolean search(int data) {
        return search(this.root, data);
    }

    private 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);
        }
```

Student Name: _____ Roll No: _____ Section: _____

```
        return search(root.right, data);
    }

    //Traversal
    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) {
        // Creating the object of BinarySearchTree class
        BinarySearchTree bst = new BinarySearchTree();
        // call the method insert
        bst.insert(8);
        bst.insert(5);
        bst.insert(9);
        bst.insert(3);
        bst.insert(7);
        bst.preorder();
        System.out.println(bst.search(7));

    }
}
```

Output:

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

```
public class BinarySearchTree {
    public class Node {
        //instance variable of Node class
        public int data;
        public Node left;
        public Node right;
```

Student Name: _____ Roll No: _____ Section: _____

```
        //constructor
        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }

    // instance variable
    public Node root;

    // constructor for initialise the root to null BYDEFAULT
    public BinarySearchTree() {
        this.root = null;
    }

    // insert method to insert the new Date
    public void insert(int newData) {
        this.root = insert(root, newData);
    }

    public Node insert(Node root, int newData) {
        // Base Case: root is null or not
        if (root == null) {
            // Insert the new data, if root is null.
            root = new Node(newData);
            // return the current root to his sub tree
            return root;
        }
        // Here checking for root data is greater or equal to newData or not
        else if (root.data >= newData) {
            // if current root data is greater than the new data then now process
the left sub-tree
            root.left = insert(root.left, newData);
        } else {
            // if current root data is less than the new data then now process
the right sub-tree
            root.right = insert(root.right, newData);
        }
        return root;
    }
    // method to get maximum value in the binary search tree
    // we are assured the maximum value is present is in root data if root is
null otherwise
    // it is in right subtree of the binary search tree
```

Student Name: _____ Roll No: _____ Section: _____

```
    public int findMaximum() {
       if(root==null){
           return -1;
       }
       // processing the right sub tree
        Node current = this.root;
        while (current.right != null) {
            current = current.right;
        }
        return (current.data);
    }

    public static void main(String[] args) {
        // Creating the object of BinarySearchTree class
        BinarySearchTree bst = new BinarySearchTree();
        // call the method insert
        bst.insert(8);
        bst.insert(5);
        bst.insert(3);
        bst.insert(7);
        bst.insert(9);
        System.out.println(bst.findMaximum());
    }
}
```

Output:

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

```
public class BinarySearchTree {

    public class Node {
        //instance variable of Node class
        public int data;
        public Node left;
        public Node right;

        //constructor
        public Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
```

Student Name: _____ Roll No: _____ Section: _____

```
        }
    }

    // instance variable
    public Node root;

    // constructor for initialise the root to null BYDEFAULT
    public BinarySearchTree() {
        this.root = null;
    }

    // insert method to insert the new Date
    public void insert(int newData) {
        this.root = insert(root, newData);
    }

    public Node insert(Node root, int newData) {
        // Base Case: root is null or not
        if (root == null) {
            // Insert the new data, if root is null.
            root = new Node(newData);
            // return the current root to his sub tree
            return root;
        }
        // Here checking for root data is greater or equal to newData or not
        else if (root.data >= newData) {
            // if current root data is greater than the new data then now process
the left sub-tree
            root.left = insert(root.left, newData);
        } else {
            // if current root data is less than the new data then now process
the right sub-tree
            root.right = insert(root.right, newData);
        }
        return root;
    }
    // method to get minimum value in the binary search tree
    // we are assured the minimum value is present is in root data if root is
null otherwise
    // it is in left subtree of the binary search tree

    public int findMinimum() {
        if(root==null){
            return -1;
        }
        // processing the left sub tree
        Node current = this.root;
```

Student Name: _____ Roll No: _____ Section: _____

```
        while (current.left != null) {
            current = current.left;
        }
        return (current.data);
    }

    public static void main(String[] args) {
        // Creating the object of BinarySearchTree class
       BinarySearchTree bst = new BinarySearchTree();
        // call the method insert
        bst.insert(8);
        bst.insert(5);
        bst.insert(3);
        bst.insert(7);
        bst.insert(9);
        System.out.println(bst.findMinimum());
    }
}
```

Output:

Student Name: _____ Roll No: _____ Section: _____

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.
2. Write algorithm of Insert and delete specific elements from the BST.
3. 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]?**
4. Explain that for a given **data [11, 34, 8, 0, 81, 32, 4, 45, 37,3, 7, 100, 1, 90, 86].** you need to delete the min element and inserting the value of **[21, 63, 55, 70].** Explain step by step how you BST will work.
5. Write the complexity of the BST and its operations (insert, delete, search, traversing, min, max) and calculate time of execution on your machine.
6. Implement the Morse Code using BST. You will get the following graph if you successfully implement the code.