**DSA Lab 10**

**Name:** Hafsa Salman

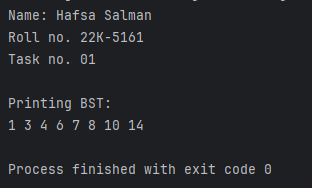
**Roll no.** 22K-5161

**Task no. 01**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 01  
  
public class Task\_01  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 01");  
 System.*out*.println();  
  
 BST bst = new BST();  
  
 bst.Insert(8);  
 bst.Insert(3);  
 bst.Insert(1);  
 bst.Insert(6);  
 bst.Insert(7);  
 bst.Insert(10);  
 bst.Insert(14);  
 bst.Insert(4);  
  
 System.*out*.println("Printing BST: ");  
 bst.Inorder();  
  
 System.*out*.println();  
 }  
}  
  
class BST  
{  
 class Node  
 {  
 int data;  
 Node left;  
 Node right;  
  
 public Node (int data)  
 {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 }  
 }  
  
 Node root;  
  
 public BST()  
 {  
 this.root = null;  
 }  
  
 public void Insert (int data)  
 {  
 root = InsertData (root, data);  
 }  
  
 public Node InsertData (Node root, int data)  
 {  
 if (root == null)  
 {  
 root = new Node(data);  
  
 return root;  
 }  
  
 if (data < root.data)  
 {  
 root.left = InsertData(root.left, data);  
 }  
  
 else if (data > root.data)  
 {  
 root.right = InsertData(root.right, data);  
 }  
  
 return root;  
 }  
  
 public void Inorder()  
 {  
 inorderVisit(root);  
 }  
  
 void inorderVisit (Node root)  
 {  
 if (root != null)  
 {  
 inorderVisit(root.left);  
 System.*out*.print(root.data + " ");  
 inorderVisit(root.right);  
 }  
 }  
}

Output:

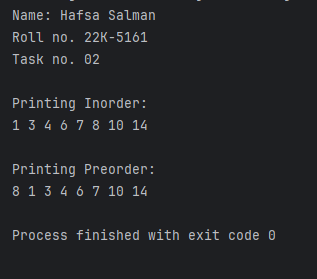


**Task no. 02**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 02  
  
public class Task\_02  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 02");  
 System.*out*.println();  
  
 BST\_02 bst = new BST\_02();  
  
 bst.Insert(8);  
 bst.Insert(3);  
 bst.Insert(1);  
 bst.Insert(6);  
 bst.Insert(7);  
 bst.Insert(10);  
 bst.Insert(14);  
 bst.Insert(4);  
  
 System.*out*.println("Printing Inorder: ");  
 bst.Inorder();  
  
 System.*out*.println();  
  
 System.*out*.println("\nPrinting Preorder: ");  
 bst.PreOrder();  
  
 System.*out*.println();  
 }  
}  
  
class BST\_02  
{  
 class Node  
 {  
 int data;  
 Node left;  
 Node right;  
  
 public Node (int data)  
 {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 }  
 }  
  
 Node root;  
  
 public BST\_02()  
 {  
 this.root = null;  
 }  
  
 public void Insert (int data)  
 {  
 root = InsertData (root, data);  
 }  
  
 public Node InsertData (Node root, int data)  
 {  
 if (root == null)  
 {  
 root = new Node(data);  
  
 return root;  
 }  
  
 if (data < root.data)  
 {  
 root.left = InsertData(root.left, data);  
 }  
  
 else if (data > root.data)  
 {  
 root.right = InsertData(root.right, data);  
 }  
  
 return root;  
 }  
  
 public void Inorder()  
 {  
 inorderVisit(root);  
 }  
  
 public void inorderVisit (Node root)  
 {  
 if (root != null)  
 {  
 inorderVisit(root.left);  
 System.*out*.print(root.data + " ");  
 inorderVisit(root.right);  
 }  
 }  
  
 public void PreOrder()  
 {  
 PreorderVisit(root);  
 }  
  
 public void PreorderVisit(Node root)  
 {  
 if (root != null)  
 {  
 System.*out*.print(root.data + " ");  
 inorderVisit(root.left);  
 inorderVisit(root.right);  
 }  
 }  
}

Output:

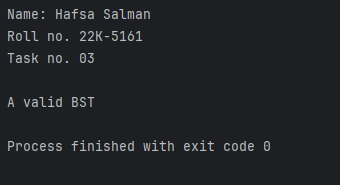


**Task no. 03**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 03  
  
public class Task\_03  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 03");  
 System.*out*.println();  
  
 BST\_03 bst = new BST\_03();  
  
 bst.Insert(8);  
 bst.Insert(3);  
 bst.Insert(1);  
 bst.Insert(6);  
 bst.Insert(7);  
 bst.Insert(10);  
 bst.Insert(14);  
 bst.Insert(4);  
  
 if (bst.isValidBST(bst.root,0,0))  
 {  
 System.*out*.println("A valid BST");  
 }  
  
 else  
 {  
 System.*out*.println("Not a valid BST");  
 }  
 }  
}  
  
class BST\_03  
{  
 class Node  
 {  
 int data;  
 Node left;  
 Node right;  
  
 public Node(int data)  
 {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 }  
 }  
  
 Node root;  
  
 public BST\_03()  
 {  
 this.root = null;  
 }  
  
 public void Insert(int data)  
 {  
 root = InsertNode(root, data);  
 }  
  
 public Node InsertNode (Node root, int data)  
 {  
 if (root == null)  
 {  
 root = new Node(data);  
  
 return root;  
 }  
  
 if (root.data > data)  
 {  
 root.left = InsertNode(root.left, data);  
 }  
  
 else if (root.data < data)  
 {  
 root.right = InsertNode(root.right, data);  
 }  
  
 return root;  
 }  
  
  
 public boolean isValidBST(Node root)  
 {  
 return isValidBST(root, 0, 0);  
 }  
  
 public boolean isValidBST(Node root, int max, int min)  
 {  
 if (root == null)  
 {  
 return true;  
 }  
  
 if (max != 0 && root.data >= max)  
 {  
 return false;  
 }  
  
 if (min != 0 && root.data <= min)  
 {  
 return false;  
 }  
  
 return isValidBST(root.left, root.data, min) && isValidBST(root.right, max, root.data);  
 }  
}

Output:

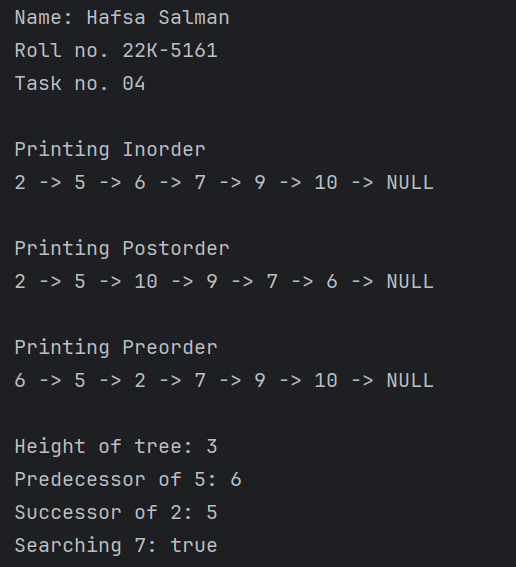


**Task no. 04**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 04  
  
public class Task\_04  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 04");  
 System.*out*.println();  
  
 BST\_04 bst = new BST\_04();  
  
 bst.Insert(6);  
 bst.Insert(5);  
 bst.Insert(7);  
 bst.Insert(2);  
 bst.Insert(9);  
 bst.Insert(10);  
  
 System.*out*.println("Printing Inorder");  
 bst.Inorder();  
  
 System.*out*.println();  
  
 System.*out*.println("\nPrinting Postorder");  
 bst.postorder();  
  
 System.*out*.println();  
  
 System.*out*.println("\nPrinting Preorder");  
 bst.Preorder();  
  
 System.*out*.println();  
  
 System.*out*.println("\nHeight of tree: " + bst.height(bst.root));  
 System.*out*.println("Predecessor of 5: " + bst.predecessor(5));  
 System.*out*.println("Successor of 2: " + bst.successor(2));  
 System.*out*.println("Searching 7: " + bst.search(7));  
  
 bst.delete(5);  
  
 System.*out*.println("\nDeleted 5");  
   
 System.*out*.println("\nPrinting Inorder: ");  
 bst.Inorder();  
 }  
}  
  
  
class BST\_04  
{  
 class Node  
 {  
 int data;  
 Node left;  
 Node right;  
  
 public Node (int data)  
 {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 }  
 }  
  
 public Node root;  
  
 public BST\_04()  
 {  
 this.root = null;  
 }  
 public void delete(int data)  
 {  
 root = Delete(root, data);  
 }  
  
 public Node Delete(Node root, int data)  
 {  
 if (root == null)  
 {  
 return root;  
 }  
  
 if (root.data > data)  
 {  
 root.left = Delete(root.left, data);  
 }  
  
 else if (root.data < data)  
 {  
 root.right = Delete(root.right, data);  
 }  
  
 else  
 {  
 if (root.left == null)  
 {  
 return root.right;  
 }  
  
 if (root.right == null)  
 {  
 return root.left;  
 }  
  
 root.data = findMin(root.right);  
 root.right = Delete(root.right, root.data);  
 }  
  
 return root;  
 }  
  
 public void Insert(int data)  
 {  
 root = insertNode(root, data);  
 }  
  
 public Node insertNode(Node root, int data)  
 {  
 if (root == null)  
 {  
 return new Node(data);  
 }  
  
 if (data < root.data)  
 {  
 root.left = insertNode(root.left, data);  
 }  
  
 else if (data > root.data)  
 {  
 root.right = insertNode(root.right, data);  
 }  
  
 return root;  
 }  
  
 public void Inorder()  
 {  
 inorderRecursive(root);  
 System.*out*.print("NULL");  
 }  
  
 public void Preorder()  
 {  
 preorderRecursive(root);  
 System.*out*.print("NULL");  
 }  
  
 public void postorder()  
 {  
 postorderRecursive(root);  
 System.*out*.print("NULL");  
 }  
  
 private void inorderRecursive (Node root)  
 {  
 if (root != null)  
 {  
 inorderRecursive(root.left);  
 System.*out*.print(root.data + " -> ");  
 inorderRecursive(root.right);  
 }  
 }  
  
 private void preorderRecursive (Node node) {  
  
 if (node != null)  
 {  
 System.*out*.print(node.data + " -> ");  
 preorderRecursive(node.left);  
 preorderRecursive(node.right);  
 }  
 }  
  
 private void postorderRecursive(Node node)  
 {  
 if (node != null)  
 {  
 postorderRecursive(node.left);  
 postorderRecursive(node.right);  
 System.*out*.print(node.data + " -> ");  
 }  
 }  
  
 private Node searchRecursive (Node root, int data)  
 {  
 if (root == null || root.data == data)  
 {  
 return root;  
 }  
  
 return data < root.data ? searchRecursive(root.left, data) : searchRecursive(root.right, data);  
 }  
  
 public boolean search (int data)  
 {  
 return searchRecursive(root, data) != null;  
 }  
  
 public int height(Node node)  
 {  
 if (node == null)  
 {  
 return -1;  
 }  
  
 return 1 + Math.*max*(height(node.left), height(node.right));  
 }  
  
 public int predecessor (int data)  
 {  
 Node current = searchRecursive(root, data);  
 if (current == null)  
 {  
 return -1;  
 }  
  
 if (current.left != null)  
 {  
 return findMax(current.left);  
 }  
  
 Node predecessor = null;  
 Node ancestor = root;  
  
 while (ancestor != current)  
 {  
 if (data > ancestor.data)  
 {  
 predecessor = ancestor;  
 ancestor = ancestor.right;  
 }  
  
 else  
 {  
 ancestor = ancestor.left;  
 }  
 }  
  
 if (predecessor != null)  
 {  
 return predecessor.data;  
 }  
  
 else  
 {  
 return -1;  
 }  
 }  
  
  
 public int successor (int data)  
 {  
 Node current = searchRecursive(root, data);  
  
 if (current == null)  
 {  
 return -1;  
 }  
  
 if (current.right != null)  
 {  
 return findMin(current.right);  
 }  
  
 Node successor = null;  
 Node ancestor = root;  
  
 while (ancestor != current)  
 {  
 if (data < ancestor.data)  
 {  
 successor = ancestor;  
 ancestor = ancestor.left;  
 }  
  
 else  
 {  
 ancestor = ancestor.right;  
 }  
 }  
  
 if (successor != null)  
 {  
 return successor.data;  
 }  
  
 else  
 {  
 return -1;  
 }  
 }  
  
 int findMax (Node node)  
 {  
 int max;  
  
 max = root.data;  
 while (node.right != null)  
 {  
 node = node.right;  
 }  
  
 return max;  
 }  
  
 int findMin (Node node)  
 {  
 int min;  
  
 min = root.data;  
  
 while (node.left != null)  
 {  
 node = node.left;  
 }  
  
 return min;  
 }  
}

Output:



A computer screen shot of a code

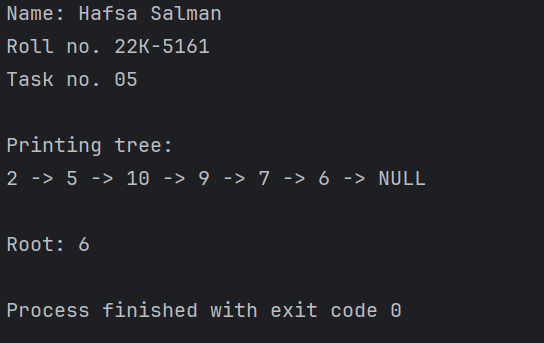
Description automatically generated

**Task no. 05**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 05  
  
public class Task\_05  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 05");  
 System.*out*.println();  
  
 BST\_05 bst = new BST\_05();  
  
 int[] preorder = {6, 5, 2, 7, 9, 10};  
  
 System.*out*.println("Printing tree: ");  
 bst.Preorder(preorder);  
 bst.Postorder();  
  
 System.*out*.println("\nRoot: " + bst.getRoot());  
 }  
}  
  
class BST\_05  
{  
 class Node  
 {  
 int data;  
 Node left;  
 Node right;  
  
 public Node (int data)  
 {  
 this.data = data;  
 this.left = null;  
 this.right = null;  
 }  
  
 @Override  
 public String toString()  
 {  
 return Integer.*toString*(data);  
 }  
 }  
  
 Node root;  
  
 public BST\_05()  
 {  
 this.root = null;  
 }  
  
 public void Insert(int data)  
 {  
 root = insertNode(root, data);  
 }  
  
 private Node insertNode (Node root, int data)  
 {  
 if (root == null)  
 {  
 root = new Node(data);  
  
 return root;  
 }  
  
 if (data < root.data)  
 {  
 root.left = insertNode(root.left, data);  
 }  
  
 else if (data > root.data)  
 {  
 root.right = insertNode(root.right, data);  
 }  
  
 return root;  
 }  
  
 public void Preorder(int[] preorder)  
 {  
 for (int i = 0; i < preorder.length; i++)  
 {  
 int value;  
  
 value = preorder[i];  
  
 Insert(value);  
 }  
  
 }  
  
 private void postorderRecursive (Node node)  
 {  
 if (node != null)  
 {  
 postorderRecursive(node.left);  
 postorderRecursive(node.right);  
 System.*out*.print(node.data + " -> ");  
 }  
 }  
  
 public void Postorder()  
 {  
 postorderRecursive(root);  
  
 System.*out*.println("NULL");  
 }  
  
 public Node getRoot()  
 {  
 return root;  
 }  
}

Output:

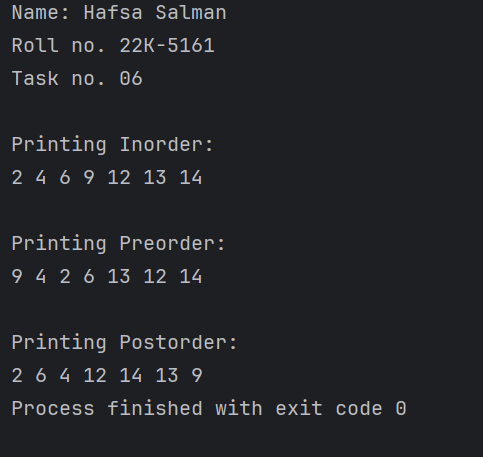


**Task no. 06**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 06  
  
public class Task\_06  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 06");  
 System.*out*.println();  
  
 AVL tree = new AVL();  
  
 tree.root = tree.insert(tree.root, 6);  
 tree.root = tree.insert(tree.root, 7);  
 tree.root = tree.insert(tree.root, 9);  
  
 tree.root = tree.deleteNode(tree.root, 7);  
  
 tree.root = tree.insert(tree.root, 12);  
 tree.root = tree.insert(tree.root, 13);  
 tree.root = tree.insert(tree.root, 14);  
 tree.root = tree.insert(tree.root, 4);  
 tree.root = tree.insert(tree.root, 2);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 8);  
  
 tree.root = tree.deleteNode(tree.root, 8);  
 tree.root = tree.deleteNode(tree.root, 5);  
  
 System.*out*.println("Printing Inorder: ");  
 tree.inOrder(tree.root);  
  
 System.*out*.println("\n\nPrinting Preorder: ");  
 tree.preOrder(tree.root);  
  
 System.*out*.println("\n\nPrinting Postorder: ");  
 tree.postOrder(tree.root);  
 }  
}  
  
class AVL  
{  
 class Node  
 {  
 int data;  
 int height;  
 Node left;  
 Node right;  
  
 public Node (int data)  
 {  
 this.data = data;  
 this.height = 1;  
 this.left = null;  
 this.right = null;  
 }  
 }  
  
 Node root;  
  
 public AVL ()  
 {  
 this.root = null;  
 }  
  
 int height (Node N)  
 {  
 if (N == null)  
 {  
 return 0;  
 }  
  
 return N.height;  
 }  
  
 int max(int a, int b)  
 {  
 if (a > b)  
 {  
 return a;  
 }  
  
 else  
 {  
 return b;  
 }  
 }  
  
 Node rightRotate (Node y)  
 {  
 Node x = y.left;  
 Node T2 = x.right;  
  
 x.right = y;  
 y.left = T2;  
  
 y.height = max(height(y.left), height(y.right)) + 1;  
 x.height = max(height(x.left), height(x.right)) + 1;  
  
 return x;  
 }  
  
 Node leftRotate (Node x)  
 {  
 Node y = x.right;  
 Node T2 = y.left;  
  
 y.left = x;  
 x.right = T2;  
  
 x.height = max(height(x.left), height(x.right)) + 1;  
 y.height = max(height(y.left), height(y.right)) + 1;  
  
 return y;  
 }  
  
 int getBalance (Node N)  
 {  
 if (N == null)  
 {  
 return 0;  
 }  
  
 return height(N.left) - height(N.right);  
 }  
  
 Node insert(Node node, int key)  
 {  
 if (node == null)  
 {  
 return (new Node(key));  
 }  
  
 if (key < node.data)  
 {  
 node.left = insert(node.left, key);  
 }  
  
 else if (key > node.data)  
 {  
 node.right = insert(node.right, key);  
 }  
  
 else  
 {  
 return node;  
 }  
  
 node.height = 1 + max(height(node.left), height(node.right));  
  
 int balance;  
  
 balance = getBalance(node);  
  
 if (balance > 1 && key < node.left.data)  
 {  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && key > node.right.data)  
 {  
 return leftRotate(node);  
 }  
  
 if (balance > 1 && key > node.left.data)  
 {  
 node.left = leftRotate(node.left);  
  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && key < node.right.data)  
 {  
 node.right = rightRotate(node.right);  
  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
 Node minValueNode(Node node)  
 {  
 Node current = node;  
  
 while (current.left != null)  
 {  
 current = current.left;  
 }  
  
 return current;  
 }  
  
 Node deleteNode(Node root, int key)  
 {  
 if (root == null)  
 {  
 return root;  
 }  
  
 if (key < root.data)  
 {  
 root.left = deleteNode(root.left, key);  
 }  
  
 else if (key > root.data)  
 {  
 root.right = deleteNode(root.right, key);  
 }  
  
 else  
 {  
 if ((root.left == null) || (root.right == null))  
 {  
 Node temp = null;  
  
 if (temp == root.left)  
 {  
 temp = root.right;  
 }  
  
 else  
 {  
 temp = root.left;  
 }  
  
 if (temp == null)  
 {  
 temp = root;  
 root = null;  
 }  
  
 else  
 {  
 root = temp;  
 }  
  
 }  
  
 else  
 {  
 Node temp = minValueNode(root.right);  
  
 root.data = temp.data;  
 root.right = deleteNode(root.right, temp.data);  
 }  
 }  
  
 if (root == null)  
 {  
 return root;  
 }  
  
 root.height = max(height(root.left), height(root.right)) + 1;  
  
 int balance;  
  
 balance = getBalance(root);  
  
 if (balance > 1 && getBalance(root.left) >= 0)  
 {  
 return rightRotate(root);  
 }  
  
 if (balance > 1 && getBalance(root.left) < 0)  
 {  
 root.left = leftRotate(root.left);  
  
 return rightRotate(root);  
 }  
  
 if (balance < -1 && getBalance(root.right) <= 0)  
 {  
 return leftRotate(root);  
 }  
  
 if (balance < -1 && getBalance(root.right) > 0)  
 {  
 root.right = rightRotate(root.right);  
  
 return leftRotate(root);  
 }  
  
 return root;  
 }  
  
 void preOrder(Node node)  
 {  
 if (node != null)  
 {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 void inOrder(Node node)  
 {  
 if (node != null)  
 {  
 inOrder(node.left);  
 System.*out*.print(node.data + " ");  
 inOrder(node.right);  
 }  
 }  
  
 void postOrder(Node node)  
 {  
 if (node != null)  
 {  
 postOrder(node.left);  
 postOrder(node.right);  
 System.*out*.print(node.data + " ");  
 }  
 }  
}

Output:

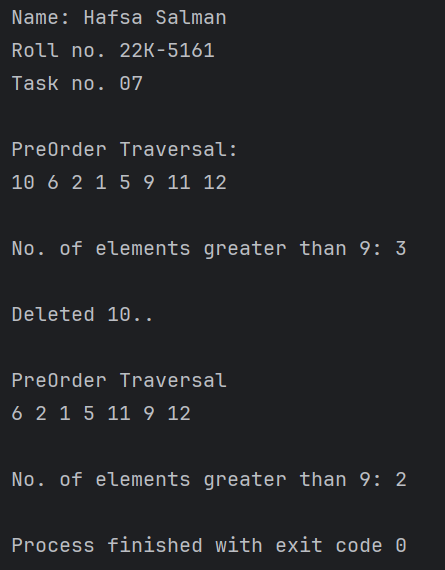


**Task no. 07**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 07  
  
public class Task\_07  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 07");  
 System.*out*.println();  
  
 AVL\_02 tree = new AVL\_02();  
  
 tree.root = tree.insertion(tree.root, 11);  
 tree.root = tree.insertion(tree.root, 5);  
 tree.root = tree.insertion(tree.root, 12);  
 tree.root = tree.insertion(tree.root, 10);  
 tree.root = tree.insertion(tree.root, 6);  
 tree.root = tree.insertion(tree.root, 9);  
 tree.root = tree.insertion(tree.root, 1);  
 tree.root = tree.insertion(tree.root, 2);  
  
 System.*out*.println("PreOrder Traversal: ");  
 tree.Preorder(tree.root);  
  
 System.*out*.println();  
  
 int countB;  
  
 countB = tree.greaterNode(tree.root, 9);  
 System.*out*.println("\nNo. of elements greater than 9: " + countB);  
  
 tree.root = tree.delete(tree.root, 10);  
  
 System.*out*.println("\nDeleted 10..");  
  
 System.*out*.println("\nPreOrder Traversal");  
 tree.Preorder(tree.root);  
  
 System.*out*.println();  
  
 int countA;  
  
 countA = tree.greaterNode(tree.root, 9);  
   
 System.*out*.println("\nNo. of elements greater than 9: " + countA);  
 }  
}  
  
class AVL\_02  
{  
 class Node  
 {  
 int data, height;  
 Node left;  
 Node right;  
  
 public Node(int data)  
 {  
 this.data = data;  
 this.height = 1;  
 this.right = null;  
 this.left = null;  
 }  
 }  
  
 Node root;  
  
 public AVL\_02 ()  
 {  
 this.root = null;  
 }  
  
 public int getHeight (Node node)  
 {  
 if (node == null)  
 {  
 return 0;  
 }  
  
 return node.height;  
 }  
  
 public int getBalance (Node node)  
 {  
 if (node == null)  
 {  
 return 0;  
 }  
  
 return getHeight(node.left) - getHeight(node.right);  
 }  
  
 public Node rightRotate (Node y)  
 {  
 Node x = y.left;  
 Node T2 = x.right;  
  
 x.right = y;  
 y.left = T2;  
  
 y.height = Math.*max*(getHeight(y.left), getHeight(y.right)) + 1;  
 x.height = Math.*max*(getHeight(x.left), getHeight(x.right)) + 1;  
  
 return x;  
 }  
  
 public Node leftRotate (Node x)  
 {  
 Node y = x.right;  
 Node T2 = y.left;  
  
 y.left = x;  
 x.right = T2;  
  
 x.height = Math.*max*(getHeight(x.left), getHeight(x.right)) + 1;  
 y.height = Math.*max*(getHeight(y.left), getHeight(y.right)) + 1;  
  
 return y;  
 }  
  
 public Node insertion(Node node, int val)  
 {  
 if (node == null)  
 {  
 return new Node(val);  
 }  
  
 if (val < node.data)  
 {  
 node.left = insertion(node.left, val);  
 }  
  
 else if (val > node.data)  
 {  
 node.right = insertion(node.right, val);  
 }  
  
 else  
 {  
 return node;  
 }  
  
 node.height = Math.*max*(getHeight(node.left), getHeight(node.right)) + 1;  
  
 int balance;  
  
 balance = getBalance(node);  
  
 if (balance > 1 && val < node.left.data)  
 {  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && val > node.right.data)  
 {  
 return leftRotate(node);  
 }  
  
 if (balance > 1 && val > node.left.data)  
 {  
 node.left = leftRotate(node.left);  
  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && val < node.right.data)  
 {  
 node.right = rightRotate(node.right);  
  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
 public Node minValueNode (Node node)  
 {  
 Node current = node;  
  
 while (current.left != null)  
 {  
 current = current.left;  
 }  
  
 return current;  
 }  
  
 public Node delete(Node node, int val)  
 {  
 if (node == null)  
 {  
 return node;  
 }  
  
 if (val < node.data) {  
 node.left = delete(node.left, val);  
 }  
  
 else if (val > node.data)  
 {  
 node.right = delete(node.right, val);  
 }  
  
 else  
 {  
 if (node.left == null || node.right == null)  
 {  
 Node temp = null;  
  
 if (temp == node.left)  
 {  
 temp = node.right;  
 }  
  
 else  
 {  
 temp = node.left;  
 }  
  
 if (temp == null)  
 {  
 temp = node;  
 node = null;  
 }  
  
 else  
 {  
 node = temp;  
 }  
 }  
  
 else  
 {  
 Node temp = minValueNode(node.right);  
  
 node.data = temp.data;  
 node.right = delete(node.right, temp.data);  
 }  
 }  
  
 if (node == null)  
 {  
 return node;  
 }  
  
 node.height = Math.*max*(getHeight(node.left), getHeight(node.right)) + 1;  
  
 int balance;  
  
 balance = getBalance(node);  
  
 if (balance > 1 && getBalance(node.left) >= 0)  
 {  
 return rightRotate(node);  
 }  
  
 if (balance > 1 && getBalance(node.left) < 0)  
 {  
 node.left = leftRotate(node.left);  
  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && getBalance(node.right) <= 0)  
 {  
 return leftRotate(node);  
 }  
  
 if (balance < -1 && getBalance(node.right) > 0)  
 {  
 node.right = rightRotate(node.right);  
  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
 public void Preorder (Node node)  
 {  
 if (node == null)  
 {  
 return;  
 }  
  
 System.*out*.print(node.data + " ");  
 Preorder(node.left);  
 Preorder(node.right);  
 }  
  
 public int greaterNode (Node node, int val)  
 {  
 if (node == null)  
 {  
 return 0;  
 }  
 int count;  
  
 count = greaterNode(node.left, val) + greaterNode(node.right, val);  
  
 if (node.data > val)  
 {  
 count++;  
 }  
  
 return count;  
 }  
}

Output:



**Task no. 08**

Code:

//Hafsa Salman  
//22K-5161  
//Task no. 08  
  
import java.util.\*;  
  
public class Task\_08  
{  
 public static void main(String[] args)  
 {  
 System.*out*.println("Name: Hafsa Salman");  
 System.*out*.println("Roll no. 22K-5161");  
 System.*out*.println("Task no. 08");  
 System.*out*.println();  
  
 AVL\_03 tree = new AVL\_03();  
  
 tree.root = tree.insert(tree.root, 15);  
 tree.root = tree.insert(tree.root, 10);  
 tree.root = tree.insert(tree.root, 5);  
 tree.root = tree.insert(tree.root, 10);  
 tree.root = tree.insert(tree.root, 15);  
  
 System.*out*.println();  
  
 System.*out*.println("Preorder traversal");  
 tree.Preorder(tree.root);  
  
 System.*out*.println("\n\nPalindrome: " + tree.isPalindrome());  
 }  
}  
  
class AVL\_03  
{  
 class Node  
 {  
 int data, height;  
 Node left;  
 Node right;  
  
 public Node (int data)  
 {  
 this.data = data;  
 this.height = 1;  
 this.left = null;  
 this.right = null;  
 }  
 }  
  
 Node root;  
  
 public AVL\_03 ()  
 {  
 this.root = null;  
 }  
  
 int height (Node N)  
 {  
 if (N == null)  
 {  
 return 0;  
 }  
  
 return N.height;  
 }  
  
 int Max(int a, int b)  
 {  
 if (a > b)  
 {  
 return a;  
 }  
  
 else  
 {  
 return b;  
 }  
 }  
  
 public Node rightRotate(Node y)  
 {  
 Node x = y.left;  
 Node T2 = x.right;  
  
 x.right = y;  
 y.left = T2;  
  
 y.height = Max(height(y.left), height(y.right)) + 1;  
 x.height = Max(height(x.left), height(x.right)) + 1;  
  
 return x;  
 }  
  
 public Node leftRotate(Node x)  
 {  
 Node y = x.right;  
 Node T2 = y.left;  
  
 y.left = x;  
 x.right = T2;  
  
 x.height = Max(height(x.left), height(x.right)) + 1;  
 y.height = Max(height(y.left), height(y.right)) + 1;  
  
 return y;  
 }  
  
 public int getBalance (Node N)  
 {  
 if (N == null)  
 {  
 return 0;  
 }  
  
 return height(N.left) - height(N.right);  
 }  
  
 public Node insert (Node node, int key)  
 {  
 if (node == null)  
 {  
 return (new Node(key));  
 }  
  
 if (key < node.data)  
 {  
 node.left = insert(node.left, key);  
 }  
  
 else if (key > node.data)  
 {  
 node.right = insert(node.right, key);  
 }  
  
 else  
 {  
 return node;  
 }  
  
 node.height = 1 + Max(height(node.left), height(node.right));  
  
 int balance;  
  
 balance = getBalance(node);  
  
 if (balance > 1 && key < node.left.data)  
 {  
 System.*out*.println("LL Rotation");  
  
 return rightRotate(node);  
 }  
  
 if (balance < -1 && key > node.right.data)  
 {  
 System.*out*.println("RR Rotation");  
  
 return leftRotate(node);  
 }  
  
 if (balance > 1 && key > node.left.data)  
 {  
 System.*out*.println("LR Rotation (Skipping rotation, performing sum and print)");  
 printInOrder(root);  
 System.*out*.println("Sum: " + sum(root));  
  
 return node;  
 }  
  
 if (balance < -1 && key < node.right.data)  
 {  
 System.*out*.println("RL Rotation (Skipping rotation, performing sum and print)");  
 printInOrder(root);  
 System.*out*.println("Sum: " + sum(root));  
  
 return node;  
 }  
  
 System.*out*.println("No Rotation");  
  
 return node;  
 }  
  
 void Preorder(Node node)  
 {  
 if (node != null)  
 {  
 System.*out*.print(node.data + " ");  
 Preorder(node.left);  
 Preorder(node.right);  
 }  
 }  
  
 void printInOrder(Node node)  
 {  
 if (node != null)  
 {  
 printInOrder(node.left);  
 System.*out*.print(node.data + " ");  
 printInOrder(node.right);  
 }  
 }  
  
 int sum(Node node)  
 {  
 if (node == null)  
 {  
 return 0;  
 }  
  
 return node.data + sum(node.left) + sum(node.right);  
 }  
  
 boolean isPalindrome()  
 {  
 List<Integer> inOrder = new ArrayList<>();  
 inorderToList(root, inOrder);  
  
 return checkPalindrome(inOrder);  
 }  
  
 void inorderToList(Node node, List<Integer> list)  
 {  
 if (node != null)  
 {  
 inorderToList(node.left, list);  
 list.add(node.data);  
 inorderToList(node.right, list);  
 }  
 }  
  
 boolean checkPalindrome(List<Integer> list) {  
  
 int i, j;  
  
 i = 0;  
 j = list.size() - 1;  
  
 while (i < j)  
 {  
 if (!list.get(i).equals(list.get(j)))  
 {  
 return false;  
 }  
  
 i++;  
 j--;  
 }  
  
 return true;  
 }  
}

Output:

