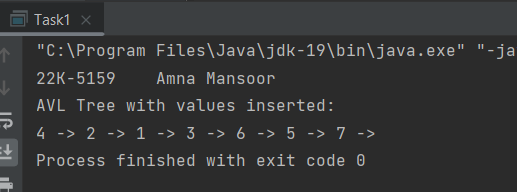
**22K-5159** **Amna Mansoor** BSE-3B DS LAB #9

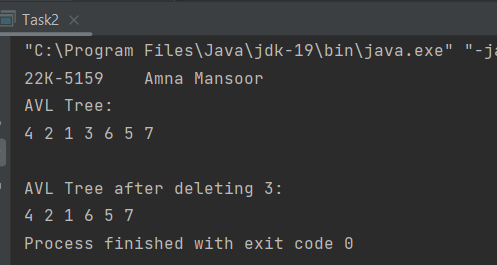
***Task 1:***

class Node {  
 int data, height;  
 Node left, right;  
  
 Node(int d) {  
 data = d;  
 height = 1;  
 }  
}  
  
class Task1 {  
 Node root;  
  
 int height(Node N) {  
 if (N == null) {  
 return 0;  
 }  
 return N.height;  
 }  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
 Node rightRotate(Node y) {  
// if (y == null || y.left == null) return y; // Check for null nodes  
  
 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) {  
// if (x == null || x.right == null) return 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 Balance(Node N) {  
 if (N == null) return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node Insert(Node node, int data) {  
 if (node == null) {  
 return new Node(data);  
 }  
 if (data < node.data) {  
 node.left = Insert(node.left, data);  
 }  
 else if (data > node.data) {  
 node.right = Insert(node.right, data);  
 }  
 else{  
 return node;  
 }  
  
 node.height = 1 + max(height(node.left), height(node.right));  
 int balance = Balance(node);  
  
 // Left Left  
 if (balance > 1 && data < node.left.data) {  
 return rightRotate(node);  
  
 }  
 // Right Right  
 if (balance < -1 && data > node.right.data){  
 return leftRotate(node);  
 }  
 // Left Right  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 // Right Left  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
 return node;  
 }  
  
 void preOrder(Node node) {  
 if (node != null) {  
 System.*out*.print(node.data + " -> ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 Task1 avl = new Task1();  
 avl.root = avl.Insert(avl.root, 1);  
 avl.root = avl.Insert(avl.root, 2);  
 avl.root = avl.Insert(avl.root, 3);  
 avl.root = avl.Insert(avl.root, 4);  
 avl.root = avl.Insert(avl.root, 5);  
 avl.root = avl.Insert(avl.root, 6);  
 avl.root = avl.Insert(avl.root, 7);  
  
 System.*out*.println("AVL Tree with values inserted: ");  
 avl.preOrder(avl.root);  
 }  
}



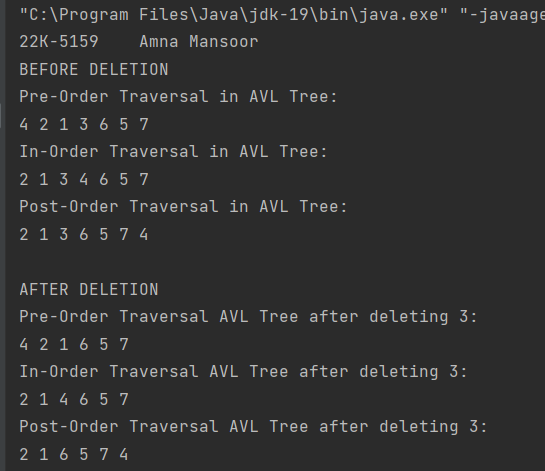
***Task 2:***

class Node\_2 {  
 int data;  
 int height;  
 Node\_2 left, right;  
  
 Node\_2(int d) {  
 data = d;  
 height = 1;  
 }  
}  
  
class Task2 {  
 Node\_2 root;  
  
 int height(Node\_2 N) {  
 if (N == null) {  
 return 0;  
 }  
 return N.height;  
 }  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
 Node\_2 rightRotate(Node\_2 y) {  
// if (y == null || y.left == null) return y; // Check for null nodes  
  
 Node\_2 x = y.left;  
 Node\_2 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\_2 leftRotate(Node\_2 x) {  
// if (x == null || x.right == null) return x;  
  
 Node\_2 y = x.right;  
 Node\_2 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 Balance(Node\_2 N) {  
 if (N == null) return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node\_2 Insert(Node\_2 node, int data) {  
 if (node == null) {  
 return new Node\_2(data);  
 }  
 if (data < node.data) {  
 node.left = Insert(node.left, data);  
 }  
 else if (data > node.data) {  
 node.right = Insert(node.right, data);  
 }  
 else{  
 return node;  
 }  
  
 node.height = 1 + max(height(node.left), height(node.right));  
 int balance = Balance(node);  
  
 // Left Left  
 if (balance > 1 && data < node.left.data) {  
 return rightRotate(node);  
  
 }  
 // Right Right  
 if (balance < -1 && data > node.right.data){  
 return leftRotate(node);  
 }  
 // Left Right  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 // Right Left  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
 return node;  
 }  
  
 void preOrder(Node\_2 node) {  
 if (node != null) {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
 Node\_2 minValue(Node\_2 node){  
 Node\_2 current=node;  
 while (current.left!=null){  
 current=current.left;  
 }  
 return current;  
 }  
  
 Node\_2 Delete(Node\_2 root, int data){  
 if(root==null){  
 return root;  
 }  
 if(data<root.data){  
 root.left=Delete(root.left,data);  
 } else if (data> root.data) {  
 root.right=Delete(root.right,data);  
 }else{  
 if((root.left==null)||(root.right==null)){  
 Node\_2 temp=null;  
 if(root.left!=null){  
 temp=root.left;  
 }  
 else{  
 temp=root.right;  
 }  
 if(temp==null){  
 temp=root;  
 root=null;  
 }  
 else{  
 root=temp;  
 }  
 }  
 else{  
 Node\_2 temp=minValue(root.right);  
 root.data=temp.data;  
 root.right=Delete(root.right,temp.data);  
 }  
 }  
 if(root==null){  
 return root;  
 }  
  
 root.height = 1 + max(height(root.left), height(root.right));  
 int balance = Balance(root);  
  
 // Left Left  
 if (balance > 1 && data < root.left.data) {  
 return rightRotate(root);  
  
 }  
 // Right Right  
 if (balance < -1 && data > root.right.data){  
 return leftRotate(root);  
 }  
 // Left Right  
 if (balance > 1 && data > root.left.data) {  
 root.left = leftRotate(root.left);  
 return rightRotate(root);  
 }  
 // Right Left  
 if (balance < -1 && data < root.right.data) {  
 root.right = rightRotate(root.right);  
 return leftRotate(root);  
 }  
 return root;  
  
 }  
  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 Task2 avl = new Task2();  
 avl.root = avl.Insert(avl.root, 1);  
 avl.root = avl.Insert(avl.root, 2);  
 avl.root = avl.Insert(avl.root, 3);  
 avl.root = avl.Insert(avl.root, 4);  
 avl.root = avl.Insert(avl.root, 5);  
 avl.root = avl.Insert(avl.root, 6);  
 avl.root = avl.Insert(avl.root, 7);  
 System.*out*.println("AVL Tree:");  
 avl.preOrder(avl.root);  
  
 avl.root=avl.Delete(avl.root,3);  
 System.*out*.println("\n\nAVL Tree after deleting 3: ");  
 avl.preOrder(avl.root);  
 }  
}



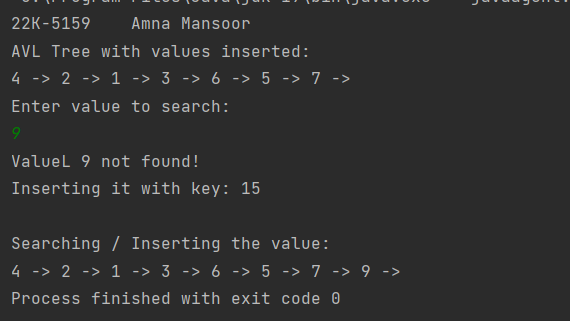
***Task 3:***

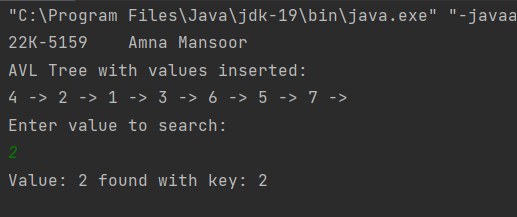
class Node\_3 {  
 int data;  
 int height;  
 Node\_3 left, right;  
  
 Node\_3(int d) {  
 data = d;  
 height = 1;  
 }  
}  
  
class Task3 {  
 Node\_3 root;  
  
 int height(Node\_3 N) {  
 if (N == null) {  
 return 0;  
 }  
 return N.height;  
 }  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
 Node\_3 rightRotate(Node\_3 y) {  
// if (y == null || y.left == null) return y; // Check for null nodes  
  
 Node\_3 x = y.left;  
 Node\_3 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\_3 leftRotate(Node\_3 x) {  
// if (x == null || x.right == null) return x;  
  
 Node\_3 y = x.right;  
 Node\_3 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 Balance(Node\_3 N) {  
 if (N == null) return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node\_3 Insert(Node\_3 node, int data) {  
 if (node == null) {  
 return new Node\_3(data);  
 }  
 if (data < node.data) {  
 node.left = Insert(node.left, data);  
 }  
 else if (data > node.data) {  
 node.right = Insert(node.right, data);  
 }  
 else{  
 return node;  
 }  
  
 node.height = 1 + max(height(node.left), height(node.right));  
 int balance = Balance(node);  
  
 // Left Left  
 if (balance > 1 && data < node.left.data) {  
 return rightRotate(node);  
  
 }  
 // Right Right  
 if (balance < -1 && data > node.right.data){  
 return leftRotate(node);  
 }  
 // Left Right  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 // Right Left  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
 return node;  
 }  
  
 Node\_3 minValue(Node\_3 node){  
 Node\_3 current=node;  
 while (current.left!=null){  
 current=current.left;  
 }  
 return current;  
 }  
  
 Node\_3 Delete(Node\_3 root, int data){  
 if(root==null){  
 return root;  
 }  
 if(data<root.data){  
 root.left=Delete(root.left,data);  
 } else if (data> root.data) {  
 root.right=Delete(root.right,data);  
 }else{  
 if((root.left==null)||(root.right==null)){  
 Node\_3 temp=null;  
 if(root.left!=null){  
 temp=root.left;  
 }  
 else{  
 temp=root.right;  
 }  
 if(temp==null){  
 temp=root;  
 root=null;  
 }  
 else{  
 root=temp;  
 }  
 }  
 else{  
 Node\_3 temp=minValue(root.right);  
 root.data=temp.data;  
 root.right=Delete(root.right,temp.data);  
 }  
 }  
 if(root==null){  
 return root;  
 }  
  
 root.height = 1 + max(height(root.left), height(root.right));  
 int balance = Balance(root);  
  
 // Left Left  
 if (balance > 1 && data < root.left.data) {  
 return rightRotate(root);  
  
 }  
 // Right Right  
 if (balance < -1 && data > root.right.data){  
 return leftRotate(root);  
 }  
 // Left Right  
 if (balance > 1 && data > root.left.data) {  
 root.left = leftRotate(root.left);  
 return rightRotate(root);  
 }  
 // Right Left  
 if (balance < -1 && data < root.right.data) {  
 root.right = rightRotate(root.right);  
 return leftRotate(root);  
 }  
 return root;  
  
 }  
 void preOrder(Node\_3 node) {  
 if (node != null) {  
 System.*out*.print(node.data + " ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
 void InOrder(Node\_3 node) {  
 if (node != null) {  
 preOrder(node.left);  
 System.*out*.print(node.data + " ");  
 preOrder(node.right);  
 }  
 }  
 void PostOrder(Node\_3 node) {  
 if (node != null) {  
 preOrder(node.left);  
 preOrder(node.right);  
 System.*out*.print(node.data + " ");  
 }  
 }  
  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 Task3 avl = new Task3();  
 avl.root = avl.Insert(avl.root, 1);  
 avl.root = avl.Insert(avl.root, 2);  
 avl.root = avl.Insert(avl.root, 3);  
 avl.root = avl.Insert(avl.root, 4);  
 avl.root = avl.Insert(avl.root, 5);  
 avl.root = avl.Insert(avl.root, 6);  
 avl.root = avl.Insert(avl.root, 7);  
 System.*out*.println("BEFORE DELETION");  
 System.*out*.println("Pre-Order Traversal in AVL Tree:");  
 avl.preOrder(avl.root);  
 System.*out*.println("\nIn-Order Traversal in AVL Tree:");  
 avl.InOrder(avl.root);  
 System.*out*.println("\nPost-Order Traversal in AVL Tree:");  
 avl.PostOrder(avl.root);  
  
 System.*out*.println("\n\nAFTER DELETION");  
 avl.root=avl.Delete(avl.root,3);  
 System.*out*.println("Pre-Order Traversal AVL Tree after deleting 3: ");  
 avl.preOrder(avl.root);  
 System.*out*.println("\nIn-Order Traversal AVL Tree after deleting 3: ");  
 avl.InOrder(avl.root);  
 System.*out*.println("\nPost-Order Traversal AVL Tree after deleting 3: ");  
 avl.PostOrder(avl.root);  
 }  
}



***Task 4:***

import java.util.Scanner;  
  
class Node\_4 {  
 int data, height;  
 Node\_4 left, right;  
  
 Node\_4(int d) {  
 data = d;  
 height = 1;  
 }  
}  
  
class Task4 {  
 Node\_4 root;  
  
 int height(Node\_4 N) {  
 if (N == null) {  
 return 0;  
 }  
 return N.height;  
 }  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
 Node\_4 rightRotate(Node\_4 y) {  
// if (y == null || y.left == null) return y; // Check for null nodes  
  
 Node\_4 x = y.left;  
 Node\_4 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\_4 leftRotate(Node\_4 x) {  
// if (x == null || x.right == null) return x;  
  
 Node\_4 y = x.right;  
 Node\_4 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 Balance(Node\_4 N) {  
 if (N == null) return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node\_4 Insert(Node\_4 node, int data) {  
 if (node == null) {  
 return new Node\_4(data);  
 }  
 if (data < node.data) {  
 node.left = Insert(node.left, data);  
 }  
 else if (data > node.data) {  
 node.right = Insert(node.right, data);  
 }  
 else{  
 return node;  
 }  
  
 node.height = 1 + max(height(node.left), height(node.right));  
 int balance = Balance(node);  
  
 // Left Left  
 if (balance > 1 && data < node.left.data) {  
 return rightRotate(node);  
  
 }  
 // Right Right  
 if (balance < -1 && data > node.right.data){  
 return leftRotate(node);  
 }  
 // Left Right  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 // Right Left  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
 return node;  
 }  
  
 void preOrder(Node\_4 node) {  
 if (node != null) {  
 System.*out*.print(node.data + " -> ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
 void Search\_Insert(int data){  
 root=Search\_Insert(root,data,1);  
 }  
 private Node\_4 Search\_Insert(Node\_4 node, int data, int key){  
 if (node==null){  
 System.*out*.println("ValueL "+data+" not found!");  
 System.*out*.println("Inserting it with key: "+key);  
 return new Node\_4(data);  
 }  
 if(data<node.data){  
 node.left=Search\_Insert(node.left,data,2\*key);  
 } else if (data>node.data) {  
 node.right=Search\_Insert(node.right,data,2\*key+1);  
 }else {  
 System.*out*.println("Value: "+data+" found with key: "+key);  
 return node;  
 }  
 node.height=1+max(height(node.left),height(node.right));  
 int balance=Balance(node);  
 if (balance > 1 && data < node.left.data) {  
 return rightRotate(node);  
 }  
 if (balance < -1 && data > node.right.data) {  
 return leftRotate(node);  
 }  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
  
 return node;  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 Task4 avl = new Task4();  
 avl.root = avl.Insert(avl.root, 1);  
 avl.root = avl.Insert(avl.root, 2);  
 avl.root = avl.Insert(avl.root, 3);  
 avl.root = avl.Insert(avl.root, 4);  
 avl.root = avl.Insert(avl.root, 5);  
 avl.root = avl.Insert(avl.root, 6);  
 avl.root = avl.Insert(avl.root, 7);  
  
 System.*out*.println("AVL Tree with values inserted: ");  
 avl.preOrder(avl.root);  
  
 Scanner sc=new Scanner(System.*in*);  
 System.*out*.println("\nEnter value to search:");  
 int value=sc.nextInt();  
 avl.Search\_Insert(value);  
  
 System.*out*.println("\nSearching / Inserting the value: ");  
 avl.preOrder(avl.root);  
  
 }  
}





***Task 5:***

class Node\_5 {  
 int data, height;  
 Node\_5 left, right;  
  
 Node\_5(int d) {  
 data = d;  
 height = 1;  
 }  
}  
  
class Task5 {  
 Node\_5 root;  
  
 int height(Node\_5 N) {  
 if (N == null) {  
 return 0;  
 }  
 return N.height;  
 }  
  
 int max(int a, int b) {  
 return (a > b) ? a : b;  
 }  
  
 Node\_5 rightRotate(Node\_5 y) {  
// if (y == null || y.left == null) return y; // Check for null nodes  
  
 Node\_5 x = y.left;  
 Node\_5 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\_5 leftRotate(Node\_5 x) {  
// if (x == null || x.right == null) return x;  
  
 Node\_5 y = x.right;  
 Node\_5 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 Balance(Node\_5 N) {  
 if (N == null) return 0;  
 return height(N.left) - height(N.right);  
 }  
  
 Node\_5 Insert(Node\_5 node, int data) {  
 if (node == null) {  
 return new Node\_5(data);  
 }  
 if (data < node.data) {  
 node.left = Insert(node.left, data);  
 }  
 else if (data > node.data) {  
 node.right = Insert(node.right, data);  
 }  
 else{  
 return node;  
 }  
  
 node.height = 1 + max(height(node.left), height(node.right));  
 int balance = Balance(node);  
  
 // Left Left  
 if (balance > 1 && data < node.left.data) {  
 return rightRotate(node);  
  
 }  
 // Right Right  
 if (balance < -1 && data > node.right.data){  
 return leftRotate(node);  
 }  
 // Left Right  
 if (balance > 1 && data > node.left.data) {  
 node.left = leftRotate(node.left);  
 return rightRotate(node);  
 }  
 // Right Left  
 if (balance < -1 && data < node.right.data) {  
 node.right = rightRotate(node.right);  
 return leftRotate(node);  
 }  
 return node;  
 }  
  
 void preOrder(Node\_5 node) {  
 if (node != null) {  
 System.*out*.print(node.data + " -> ");  
 preOrder(node.left);  
 preOrder(node.right);  
 }  
 }  
  
 void kth\_Smallest\_Largest(Node\_5 node, int k, int[] count){  
 if(node==null||count[0]>=k){  
 return;  
 }  
 kth\_Smallest\_Largest(node.left,k,count);  
 count[0]++;  
 if(count[0]==k){  
 System.*out*.println();  
 System.*out*.println(k+"th Smallest: "+node.data);  
 }  
 kth\_Smallest\_Largest(node.right,k,count);  
 }  
 void SideHeights(Node\_5 node, int[] leftHeight, int[] rightHeight, int currentHeight){  
 if(node==null){  
 return;  
 }  
  
 SideHeights(node.left, leftHeight, rightHeight, currentHeight+1);  
 leftHeight[0]=Math.*max*(leftHeight[0],currentHeight);  
  
 SideHeights(node.right, leftHeight, rightHeight, currentHeight+1);  
 rightHeight[0]=Math.*max*(rightHeight[0],currentHeight);  
  
  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor");  
 Task5 avl = new Task5();  
 avl.root = avl.Insert(avl.root, 1);  
 avl.root = avl.Insert(avl.root, 2);  
 avl.root = avl.Insert(avl.root, 3);  
 avl.root = avl.Insert(avl.root, 4);  
 avl.root = avl.Insert(avl.root, 5);  
 avl.root = avl.Insert(avl.root, 6);  
 avl.root = avl.Insert(avl.root, 7);  
  
 System.*out*.println("AVL Tree with values inserted: ");  
 avl.preOrder(avl.root);  
  
 int k=3;  
 int[] count={0};  
 avl.kth\_Smallest\_Largest(avl.root,k,count);  
 int [] leftHeight={0};  
 int [] rightHeight={0};  
 avl.SideHeights(avl.root, leftHeight, rightHeight, 1);  
 System.*out*.println("Left side height: "+leftHeight[0]);  
 System.*out*.println("Left right height: "+rightHeight[0]);  
 }  
}

