أسئلة على التري

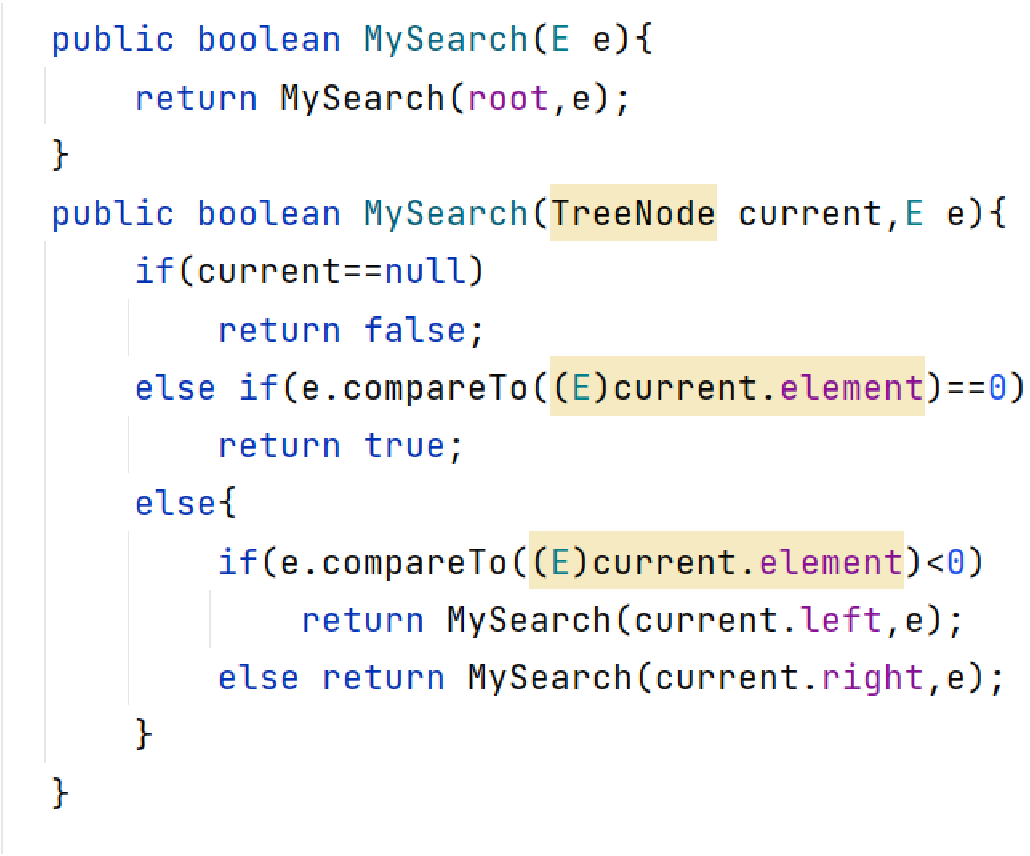
شعبة 3

محمد ابو صفط

1* ميثود يضيف عالتري:

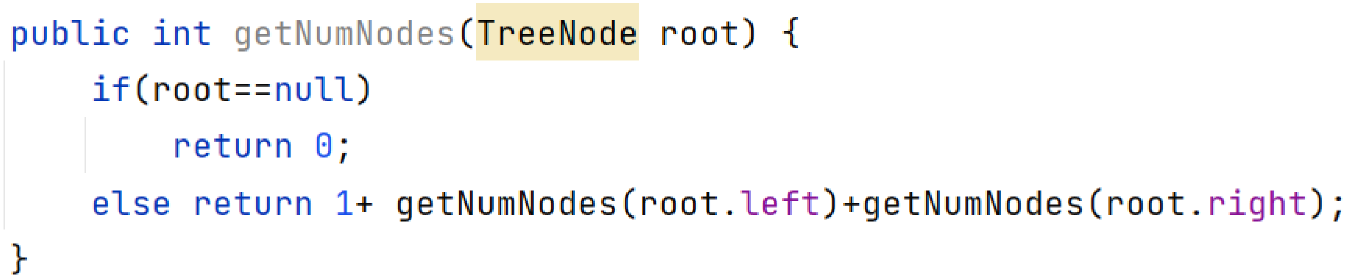
```java
public boolean addTree(E e){
    TreeNode current=root,parent=null;
    if(current==null){
        root = new TreeNode<>(e);
        return true;
    }
    else{
        while(current!=null){
            if(e.compareTo((E) current.element)<0){
                parent=current;
                current=current.left;
            }
            else if(e.compareTo((E) current.element)>0){
                parent=current;
                current=current.right;
            }
            else if(e.compareTo((E) current.element)==0){
                return false;
            }
        }
        if(e.compareTo((E) parent.element)<0)
            parent.left=new TreeNode(e);
        if(e.compareTo((E) parent.element)>0)
            parent.right=new TreeNode(e);
        return true;
    }
}
```

2* ميثود يعمل فحص اذا كانت القيمة موجودة ولا لا:



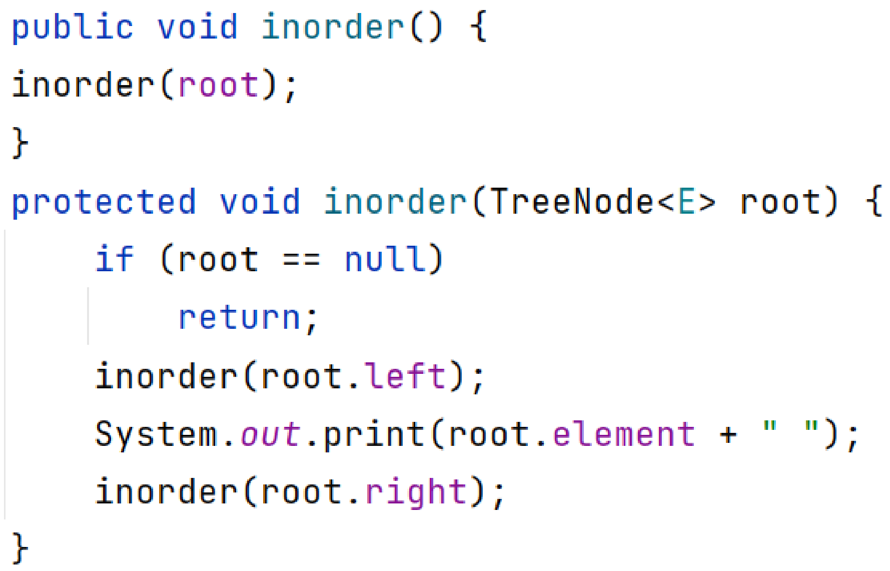
```
public boolean MySearch(E e){
    return MySearch(root,e);
}
public boolean MySearch(TreeNode current,E e){
    if(current==null)
        return false;
    else if(e.compareTo((E)current.element)==0)
        return true;
    else{
        if(e.compareTo((E)current.element)<0)
            return MySearch(current.left,e);
        else return MySearch(current.right,e);
    }
}
```

3* ميثود يرجعلي عدد النودز(السايز) بس بدون ما استخدم الاتريبيوت الجاهز

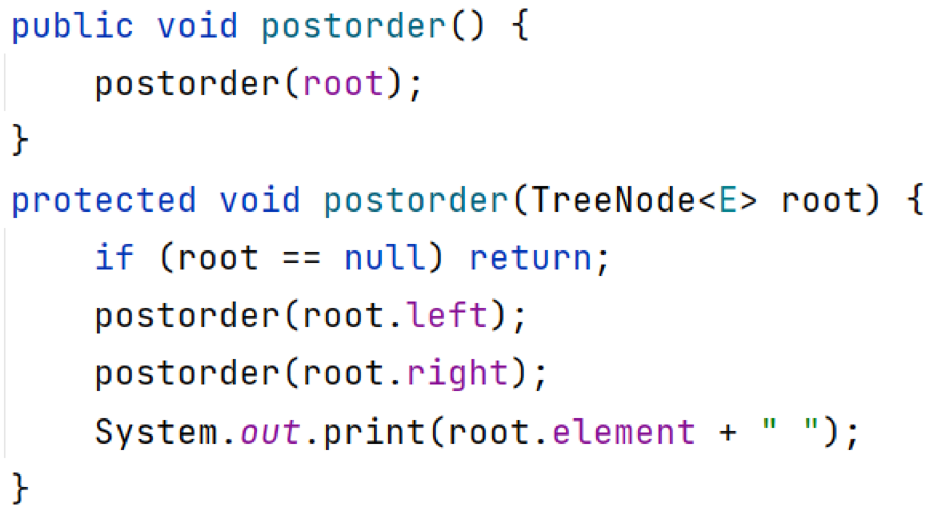


```java
public int getNumNodes(TreeNode root) {
    if(root==null)
        return 0;
    else return 1+ getNumNodes(root.left)+getNumNodes(root.right);
}
```

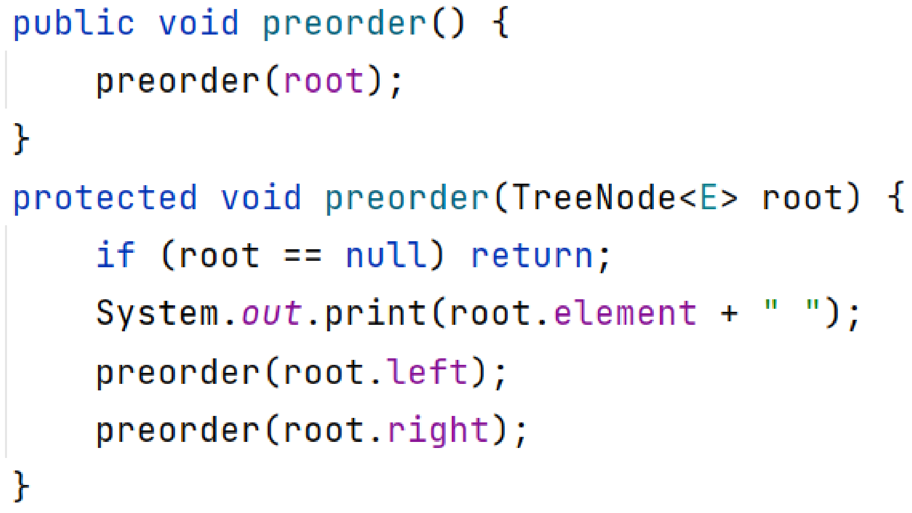
4* ميثودات الطباعة :



```java
public void inorder() {
inorder(root);
}
protected void inorder(TreeNode<E> root) {
    if (root == null)
        return;
    inorder(root.left);
    System.out.print(root.element + " ");
    inorder(root.right);
}
```

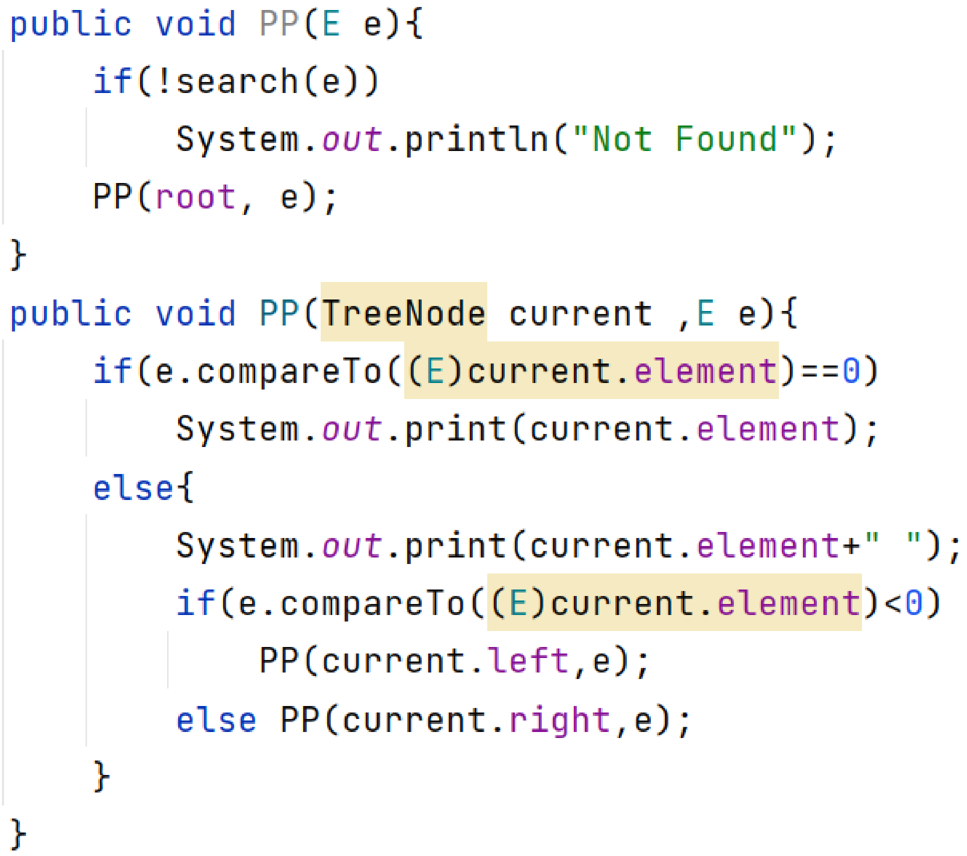


```java
public void postorder() {
    postorder(root);
}
protected void postorder(TreeNode<E> root) {
    if (root == null) return;
    postorder(root.left);
    postorder(root.right);
    System.out.print(root.element + " ");
}
```



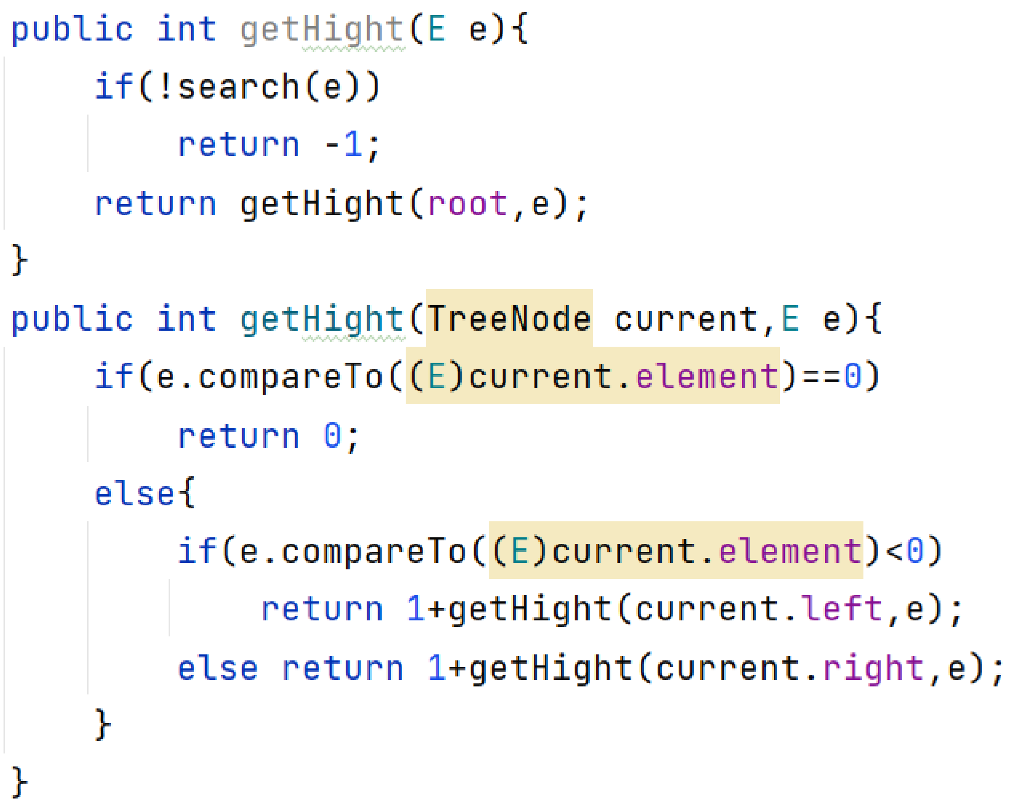
```java
public void preorder() {
    preorder(root);
}
protected void preorder(TreeNode<E> root) {
    if (root == null) return;
    System.out.print(root.element + " ");
    preorder(root.left);
    preorder(root.right);
}
```

5* ميثود يطبع الباث للعنصر :



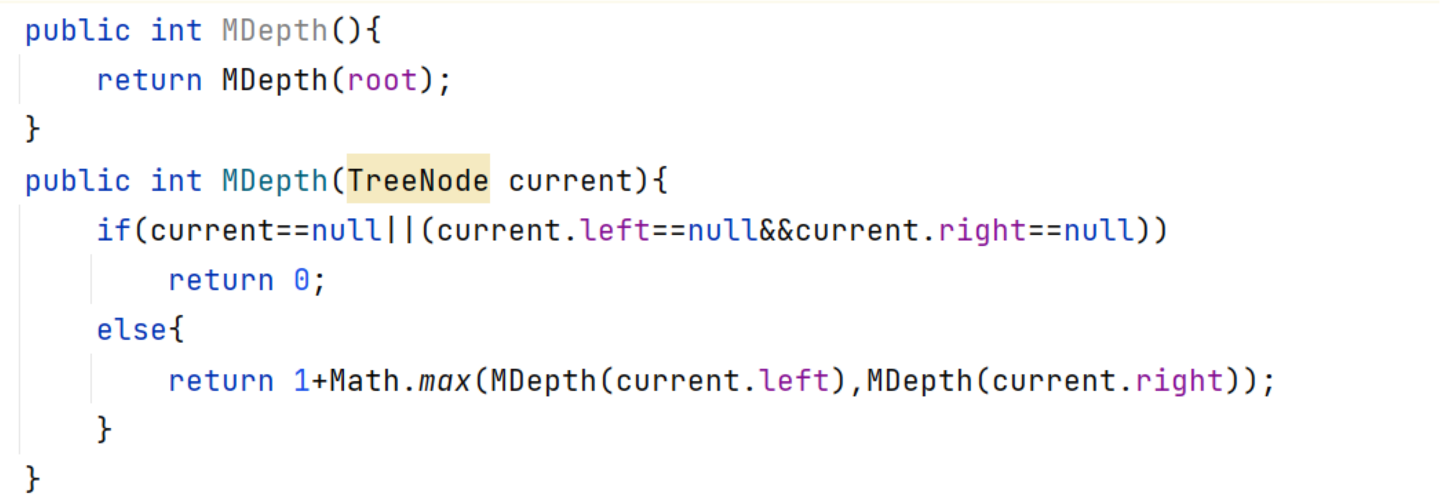
```
public void PP(E e){
    if(!search(e))
        System.out.println("Not Found");
    PP(root, e);
}
public void PP(TreeNode current ,E e){
    if(e.compareTo((E)current.element)==0)
        System.out.print(current.element);
    else{
        System.out.print(current.element+" ");
        if(e.compareTo((E)current.element)<0)
            PP(current.left,e);
        else PP(current.right,e);
    }
}
```

6*ميثود يعطيني اللفل للعنصر :



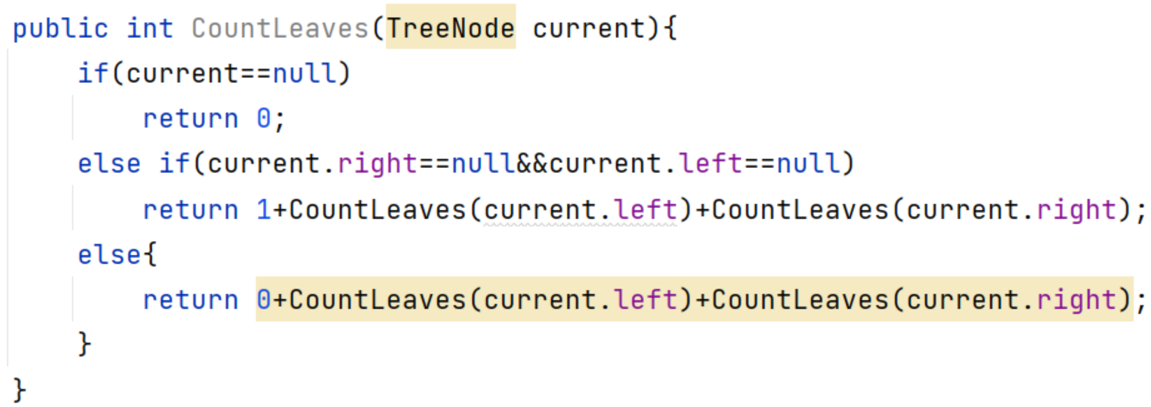
```
public int getHight(E e){
    if(!search(e))
        return -1;
    return getHight(root,e);
}
public int getHight(TreeNode current,E e){
    if(e.compareTo((E)current.element)==0)
        return 0;
    else{
        if(e.compareTo((E)current.element)<0)
            return 1+getHight(current.left,e);
        else return 1+getHight(current.right,e);
    }
}
```

7* ميثود يرجع اكبر لفل بالتري:



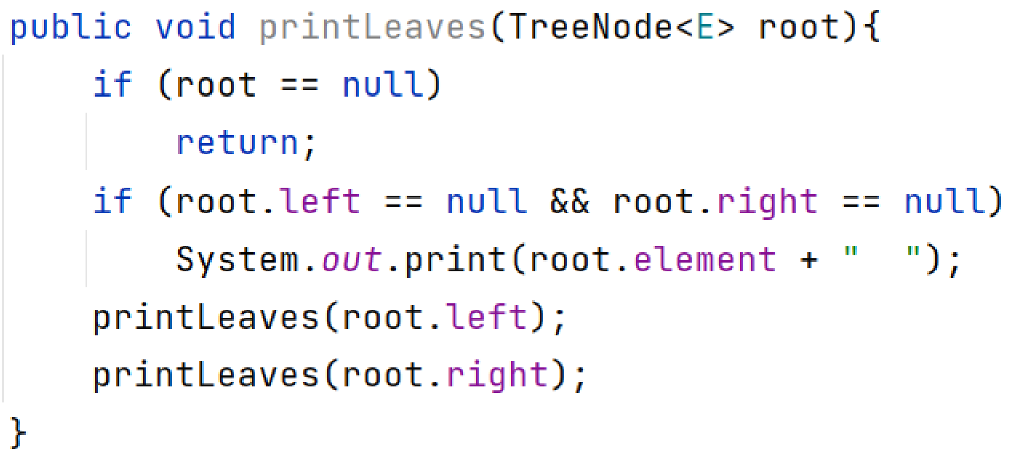
```java
public int MDepth(){
    return MDepth(root);
}
public int MDepth(TreeNode current){
    if(current==null||(current.left==null&&current.right==null))
        return 0;
    else{
        return 1+Math.max(MDepth(current.left),MDepth(current.right));
    }
}
```

8* ميثود يرجع عدد الليفز:



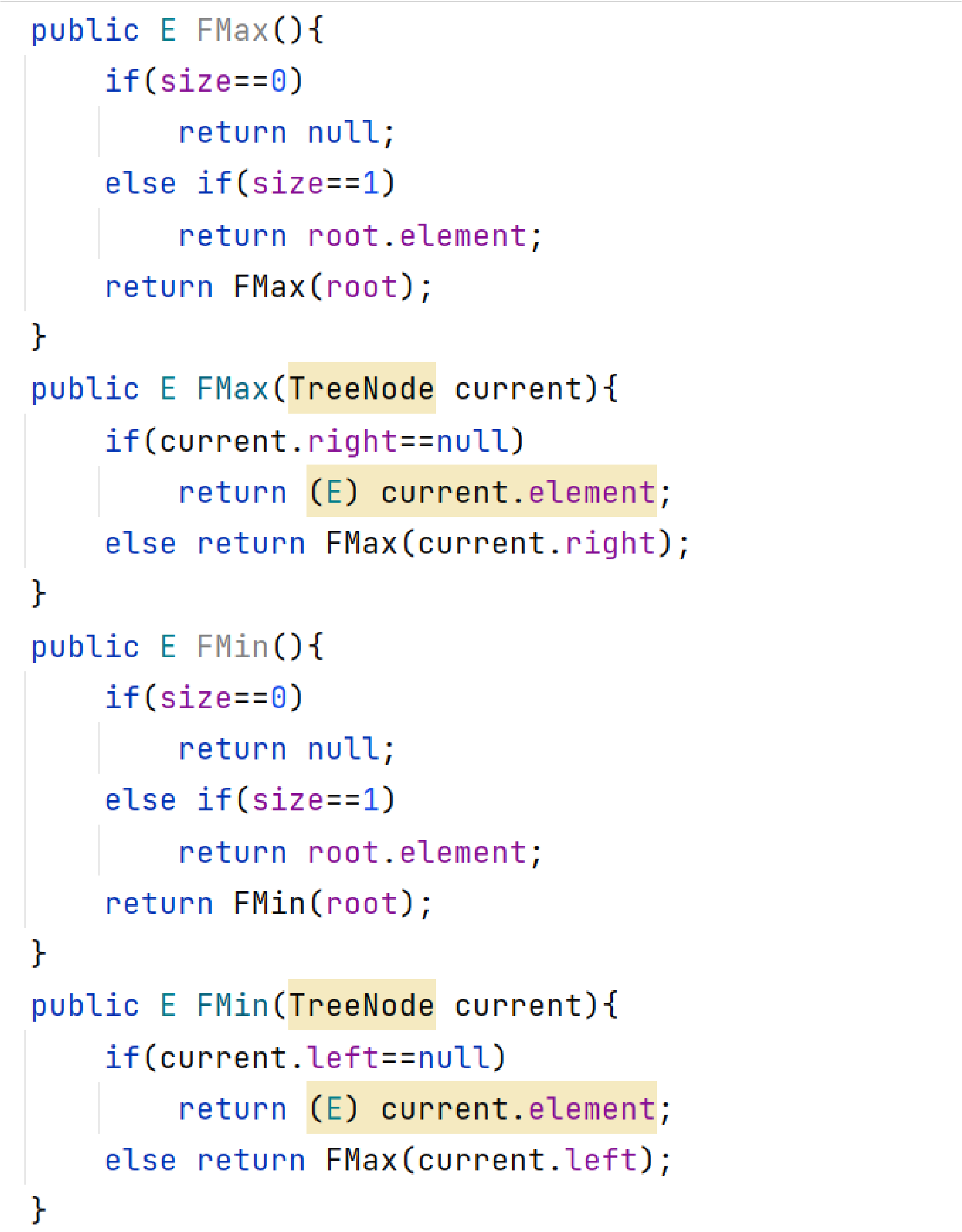
```java
public int CountLeaves(TreeNode current){
    if(current==null)
        return 0;
    else if(current.right==null&&current.left==null)
        return 1+CountLeaves(current.left)+CountLeaves(current.right);
    else{
        return 0+CountLeaves(current.left)+CountLeaves(current.right);
    }
}
```

9* ميثود يطبع الليفز:



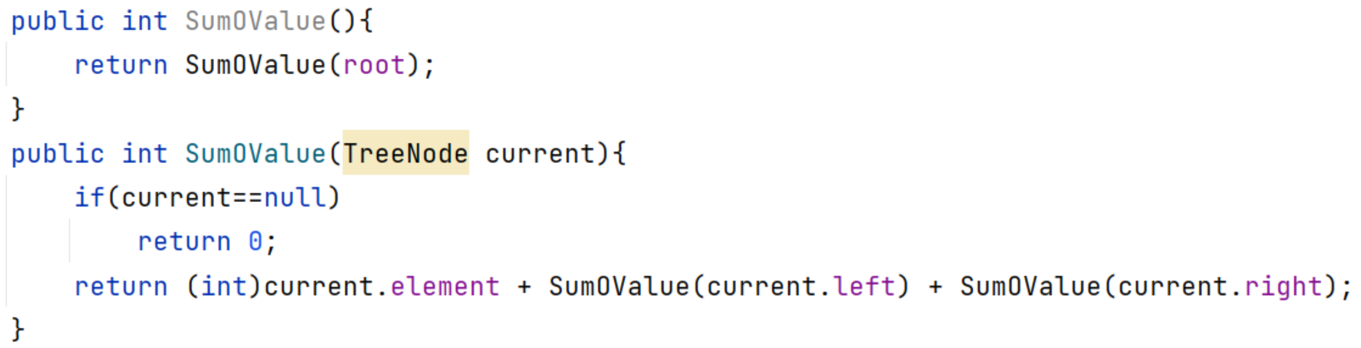
```java
public void printLeaves(TreeNode<E> root){
    if (root == null)
        return;
    if (root.left == null && root.right == null)
        System.out.print(root.element + "  ");
    printLeaves(root.left);
    printLeaves(root.right);
}
```

10* ميثود يرجعلي اكبر قيمة بالتري + ميثود يرجعلي اصغر قيمة :



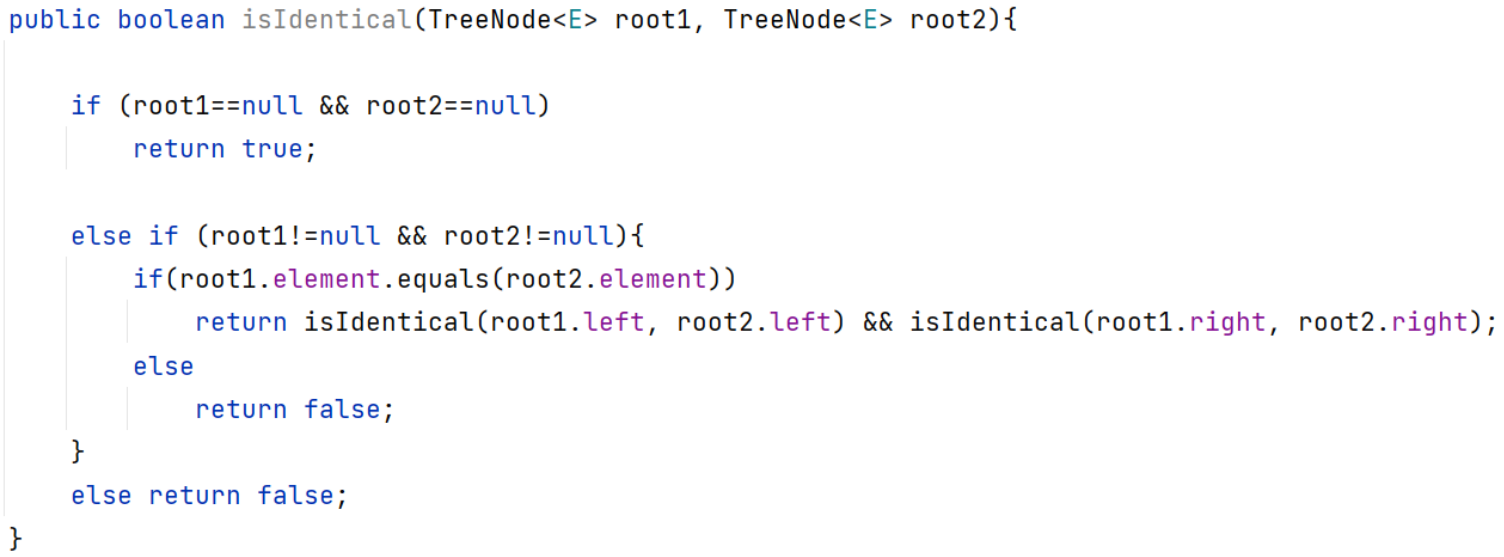
```
public E FMax(){
    if(size==0)
        return null;
    else if(size==1)
        return root.element;
    return FMax(root);
}
public E FMax(TreeNode current){
    if(current.right==null)
        return (E) current.element;
    else return FMax(current.right);
}
public E FMin(){
    if(size==0)
        return null;
    else if(size==1)
        return root.element;
    return FMin(root);
}
public E FMin(TreeNode current){
    if(current.left==null)
        return (E) current.element;
    else return FMax(current.left);
}
```

11* ميثود يرجعلي مجموع عناصر التري :



```java
public int SumOValue(){
    return SumOValue(root);
}
public int SumOValue(TreeNode current){
    if(current==null)
        return 0;
    return (int)current.element + SumOValue(current.left) + SumOValue(current.right);
}
```

12* ميثود يستقبل تو روتس ويشوف اذا التو تريز متماثلات او متساويات :

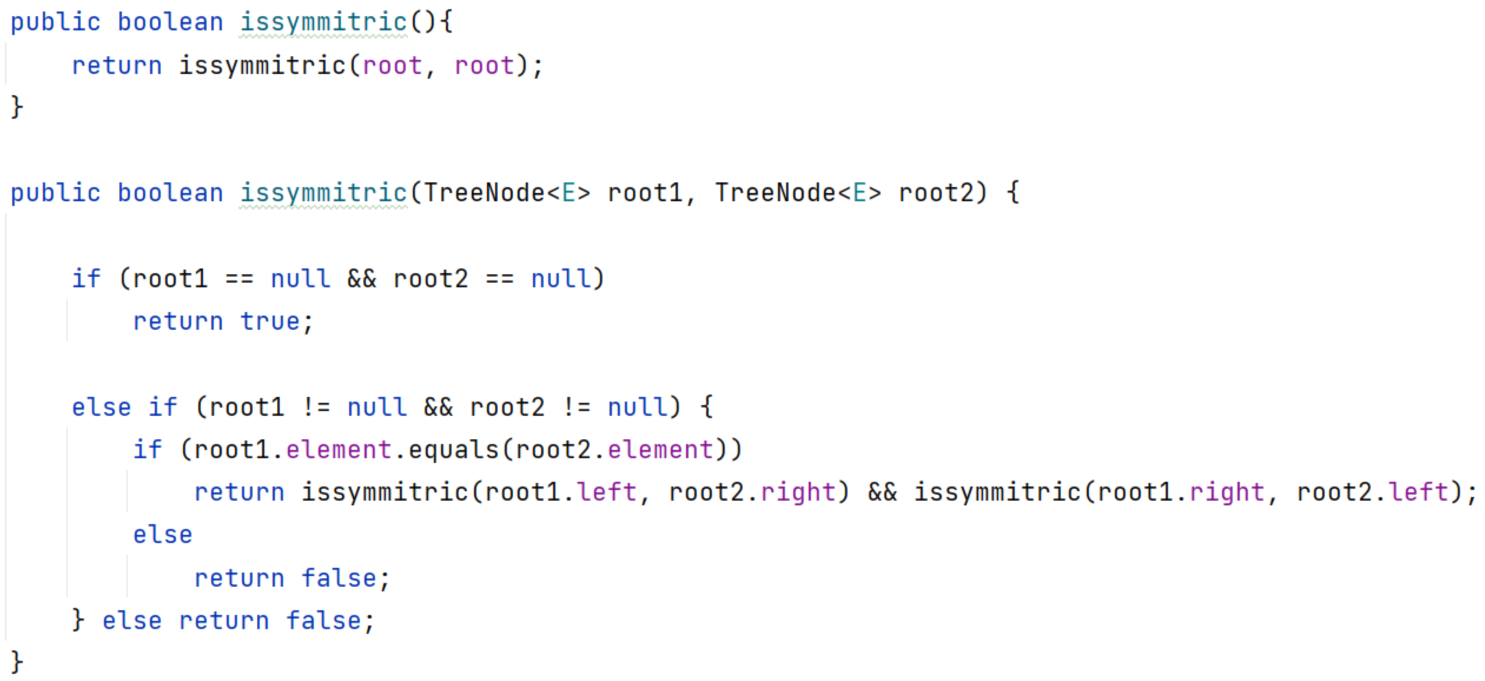


```java
public boolean isIdentical(TreeNode<E> root1, TreeNode<E> root2){

    if (root1==null && root2==null)
        return true;

    else if (root1!=null && root2!=null){
        if(root1.element.equals(root2.element))
            return isIdentical(root1.left, root2.left) && isIdentical(root1.right, root2.right);
        else
            return false;
    }
    else return false;
}
```

13* ميثود يفحص اذا التري سيميتريك :



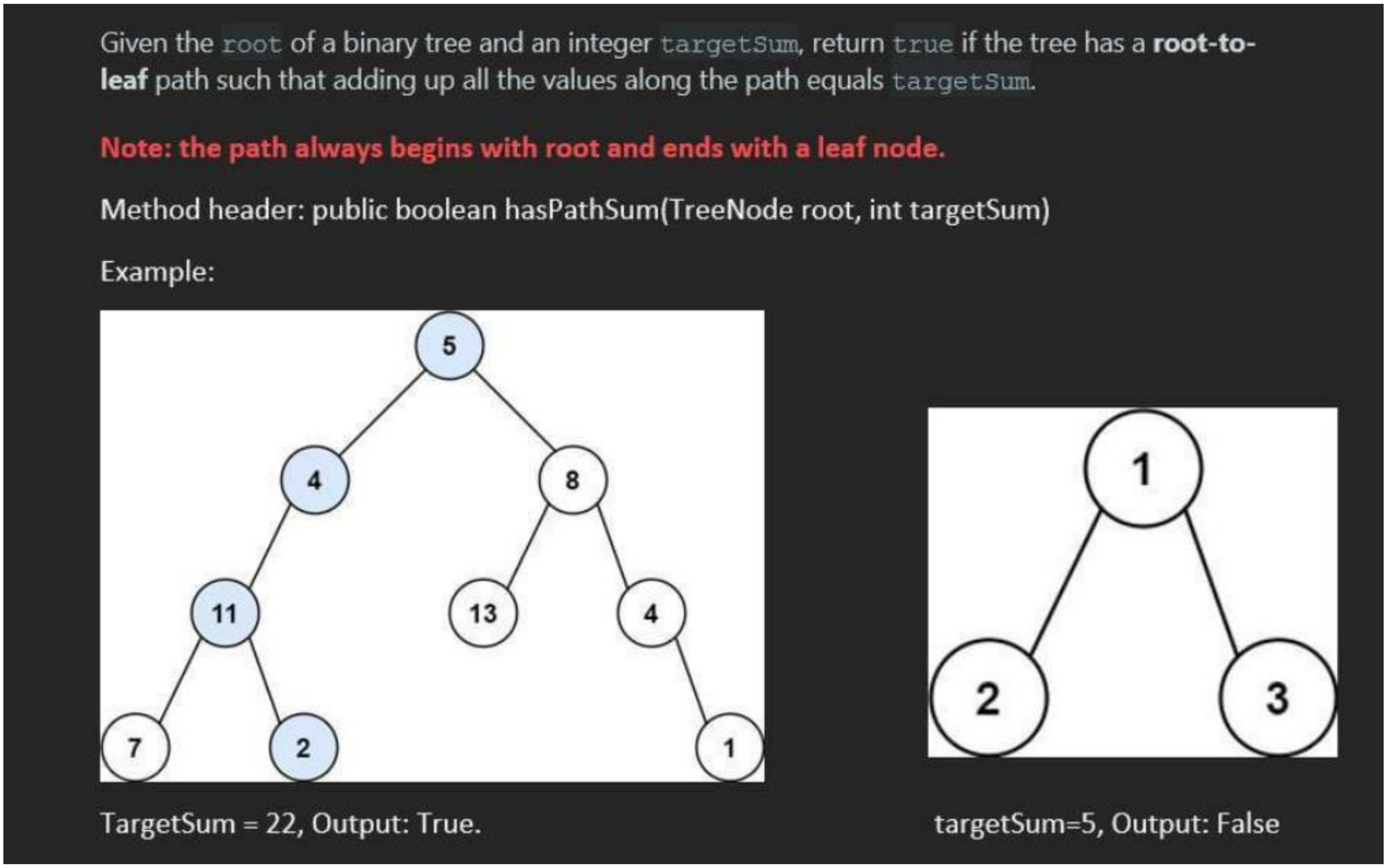
```
public boolean issymmitric(){
    return issymmitric(root, root);
}

public boolean issymmitric(TreeNode<E> root1, TreeNode<E> root2) {

    if (root1 == null && root2 == null)
        return true;

    else if (root1 != null && root2 != null) {
        if (root1.element.equals(root2.element))
            return issymmitric(root1.left, root2.right) && issymmitric(root1.right, root2.left);
        else
            return false;
    } else return false;
}
```

*14



Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

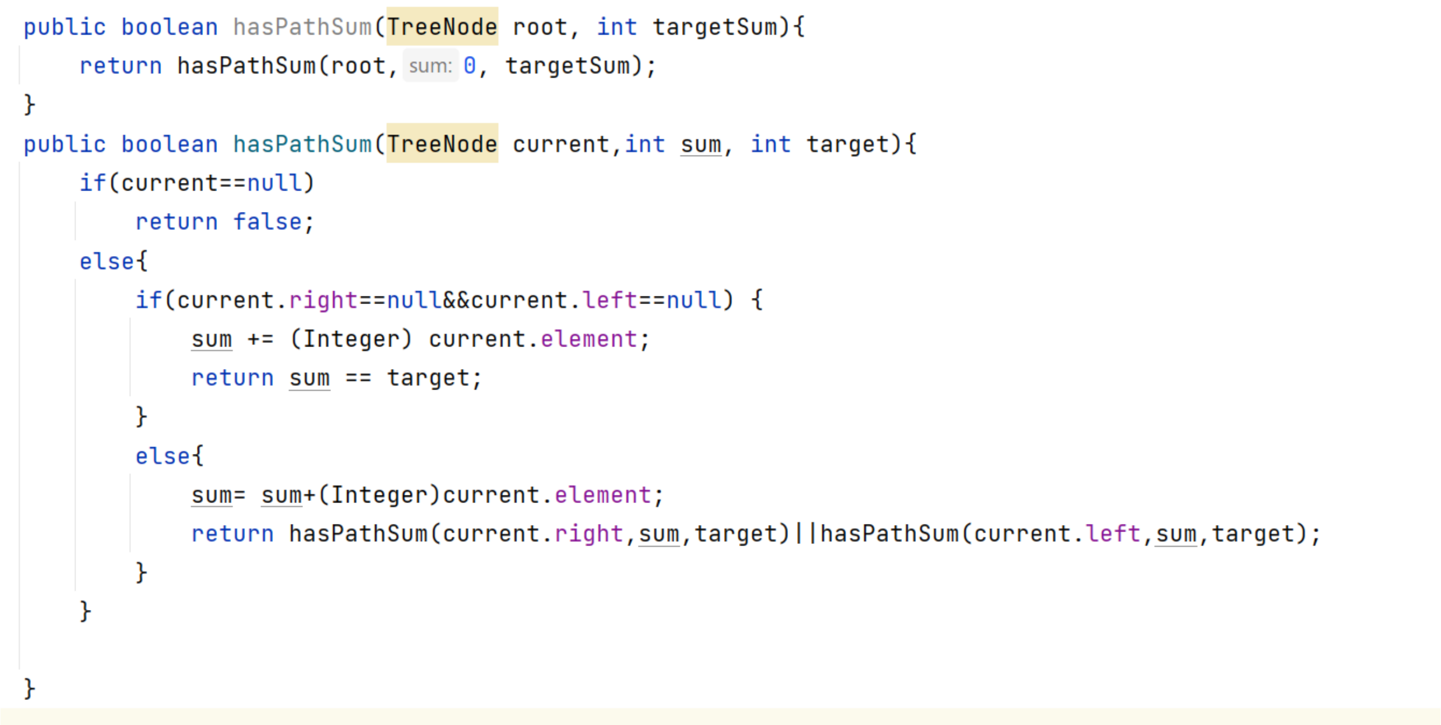
Note: the path always begins with root and ends with a leaf node.

Method header: public boolean hasPathSum(TreeNode root, int targetSum)

Example:

TargetSum = 22, Output: True.

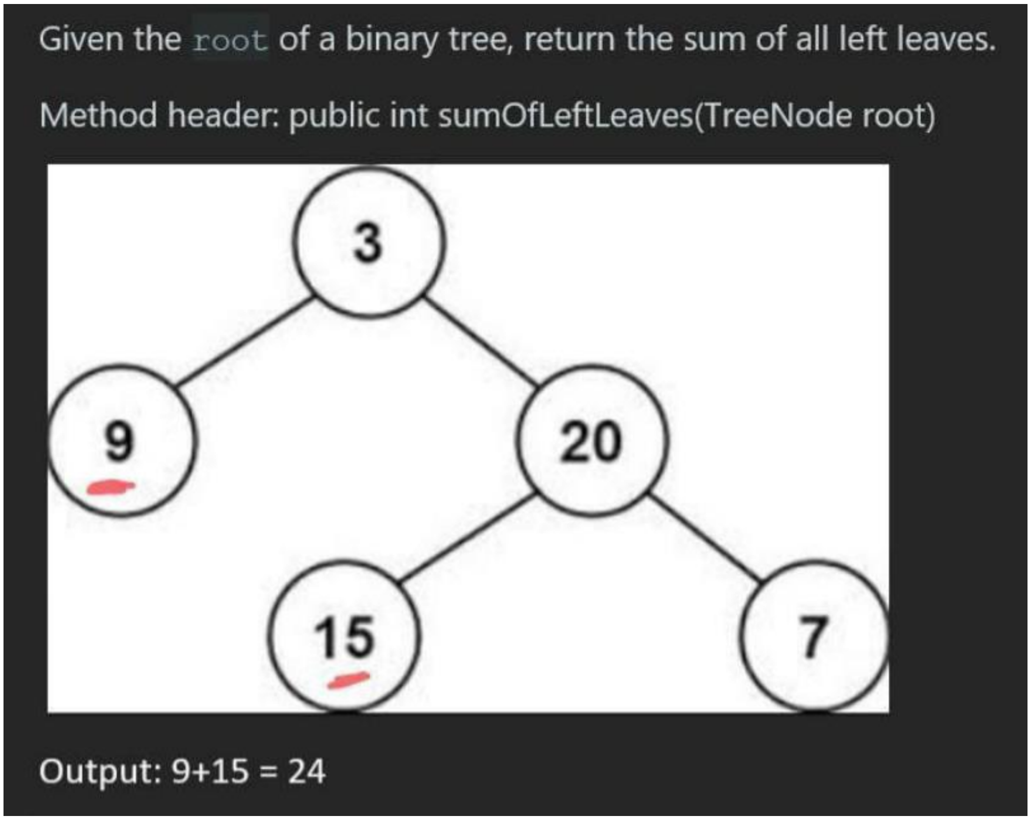
targetSum=5, Output: False



```
public boolean hasPathSum(TreeNode root, int targetSum){
    return hasPathSum(root, sum: 0, targetSum);
}
public boolean hasPathSum(TreeNode current,int sum, int target){
    if(current==null)
        return false;
    else{
        if(current.right==null&&current.left==null) {
            sum += (Integer) current.element;
            return sum == target;
        }
        else{
            sum= sum+(Integer)current.element;
            return hasPathSum(current.right,sum,target)||hasPathSum(current.left,sum,target);
        }
    }

}
```

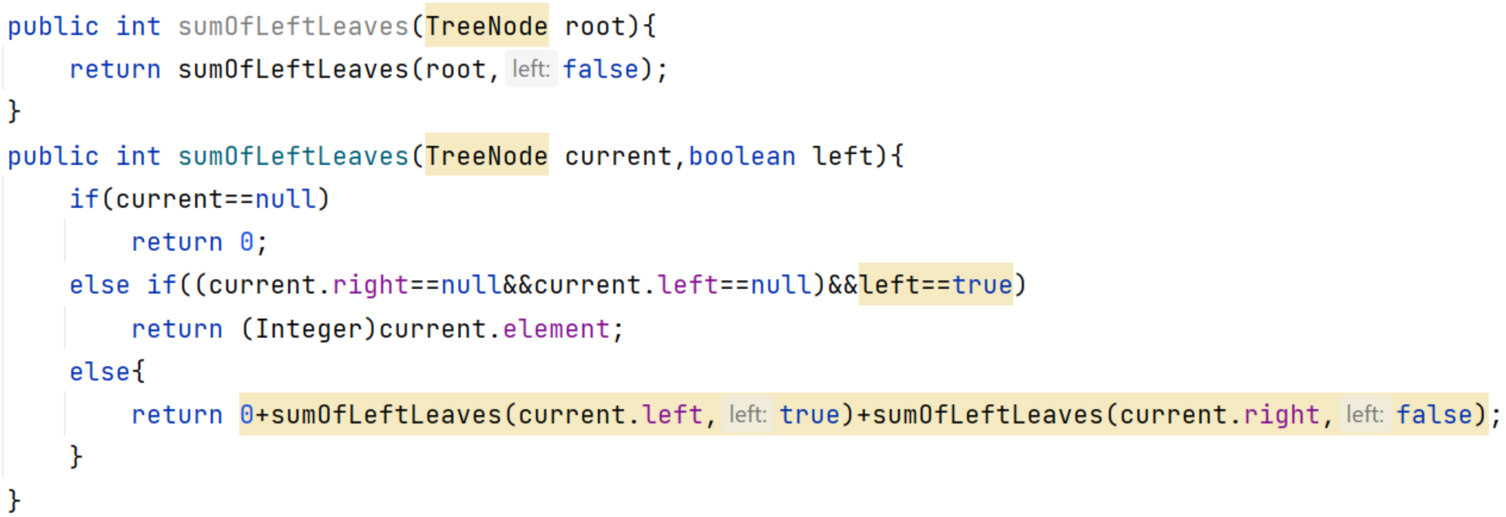
*15



Given the `root` of a binary tree, return the sum of all left leaves.

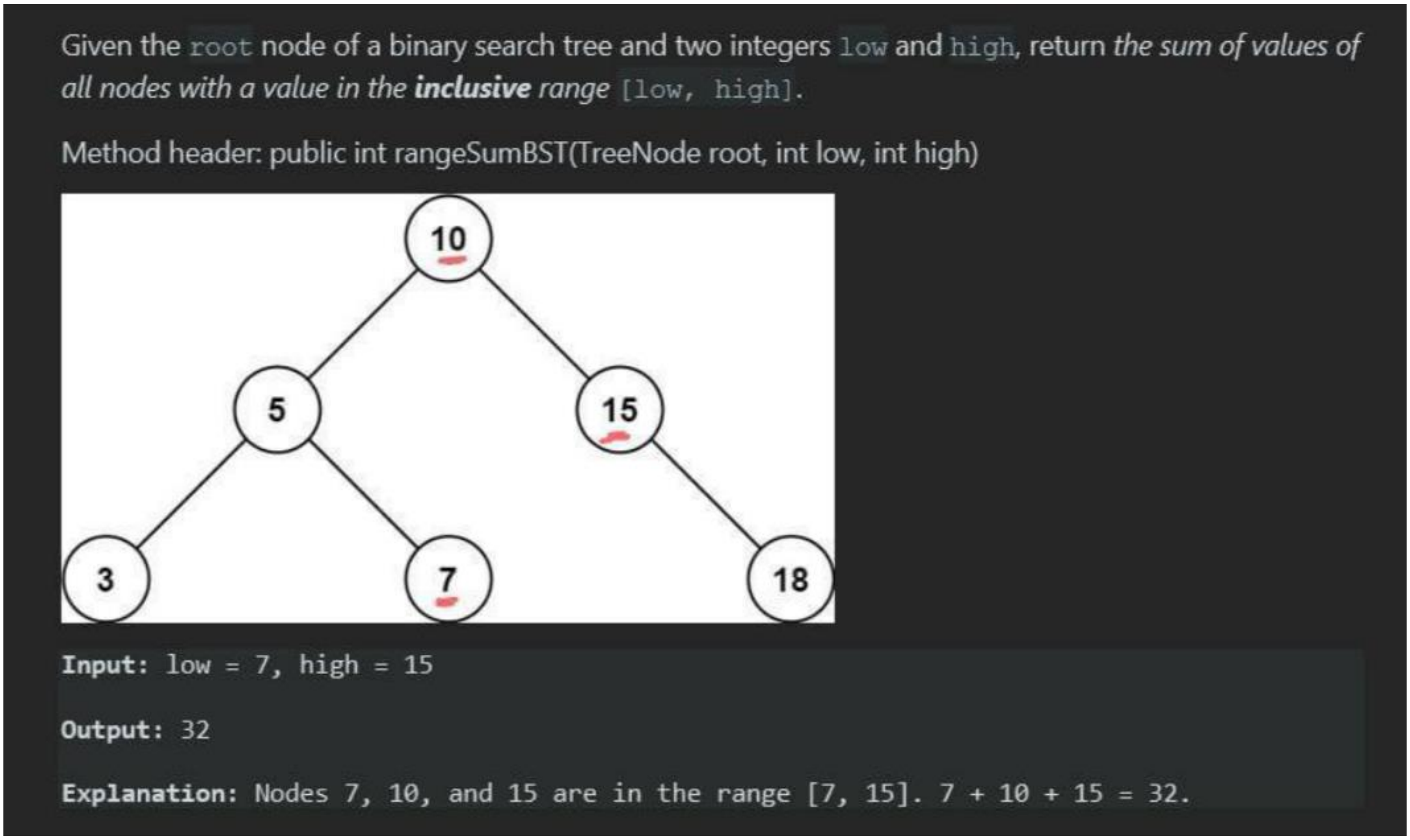
Method header: public int sumOfLeftLeaves(TreeNode root)

Output: 9+15 = 24



```
public int sumOfLeftLeaves(TreeNode root){
    return sumOfLeftLeaves(root, left: false);
}
public int sumOfLeftLeaves(TreeNode current,boolean left){
    if(current==null)
        return 0;
    else if((current.right==null&&current.left==null)&&left==true)
        return (Integer)current.element;
    else{
        return 0+sumOfLeftLeaves(current.left, left: true)+sumOfLeftLeaves(current.right, left: false);
    }
}
```

*16



Given the `root` node of a binary search tree and two integers `low` and `high`, return *the sum of values of all nodes with a value in the* ***inclusive*** *range* `[low, high]`.

Method header: public int rangeSumBST(TreeNode root, int low, int high)

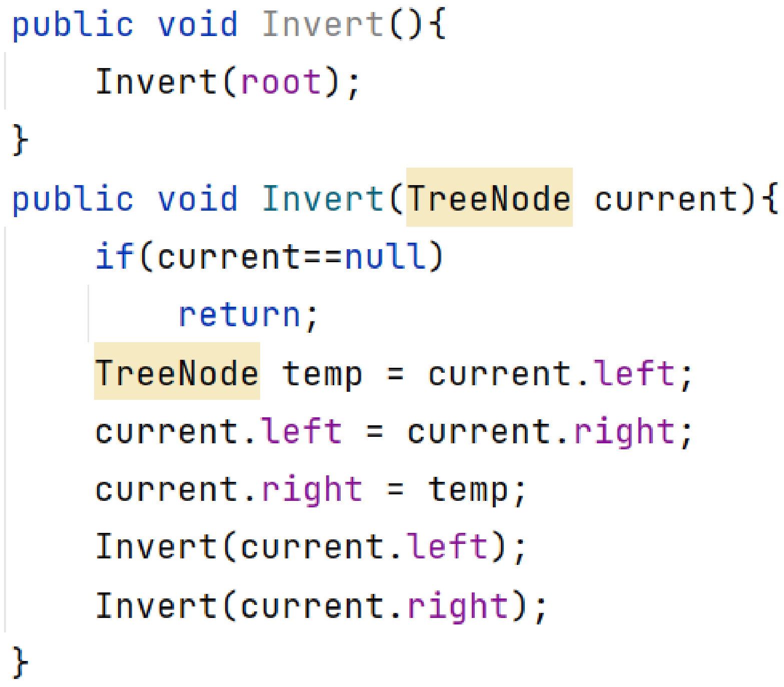
```
Input: low = 7, high = 15

Output: 32

Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32.
```

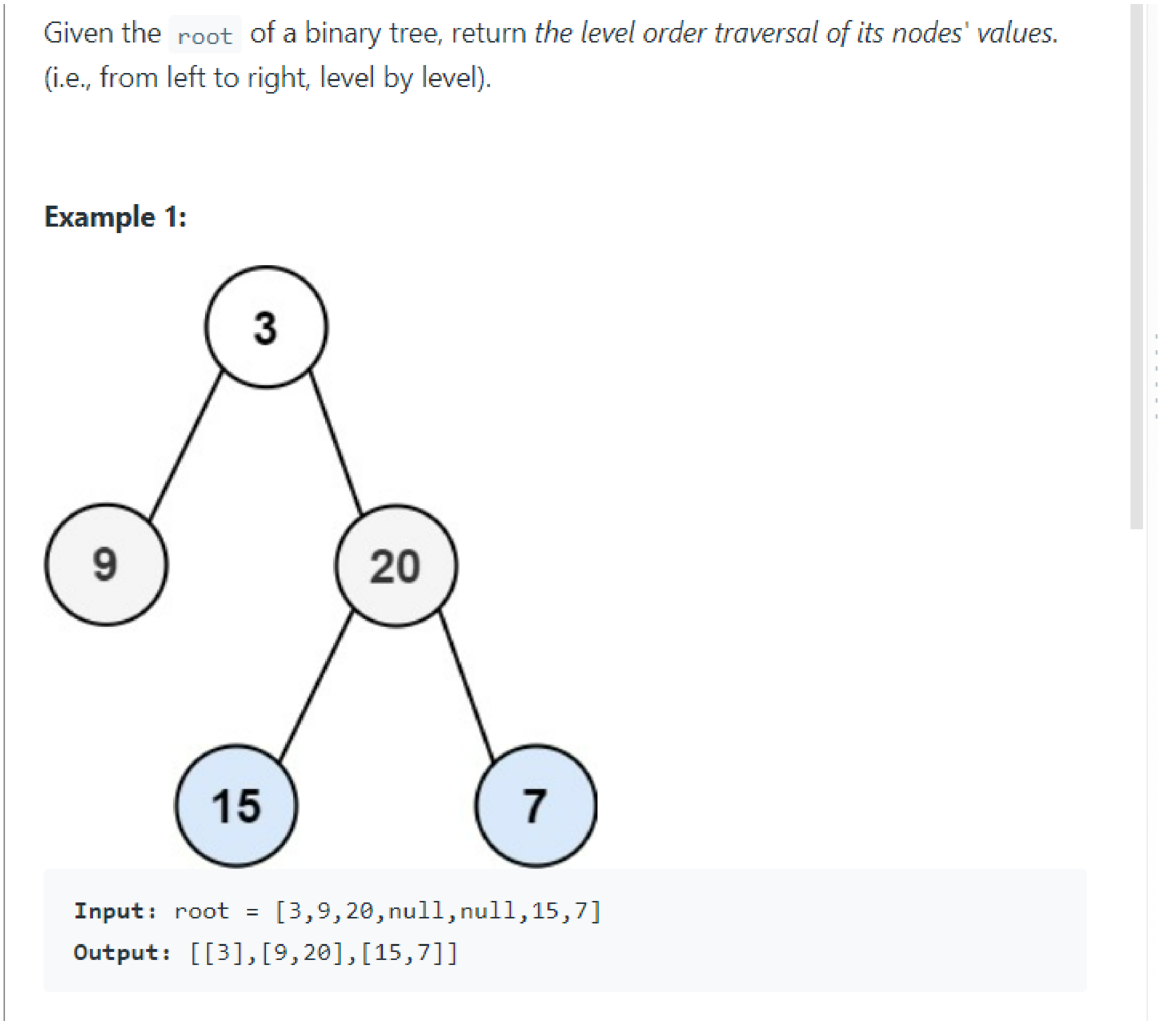
```
public int rangeSumBST(TreeNode root, int low, int high){
    if(root==null)
        return 0;
    else{
        if((int)root.element<=high&&(int)root.element>=low)
            return (int)root.element + rangeSumBST(root.left,low, high) + rangeSumBST(root.right,low, high);
        else return 0 + rangeSumBST(root.left,low, high) + rangeSumBST(root.right,low, high);
    }
}
```

17* ميثود يعمل انفيرت للتري:



```
public void Invert(){
    Invert(root);
}
public void Invert(TreeNode current){
    if(current==null)
        return;
    TreeNode temp = current.left;
    current.left = current.right;
    current.right = temp;
    Invert(current.left);
    Invert(current.right);
}
```

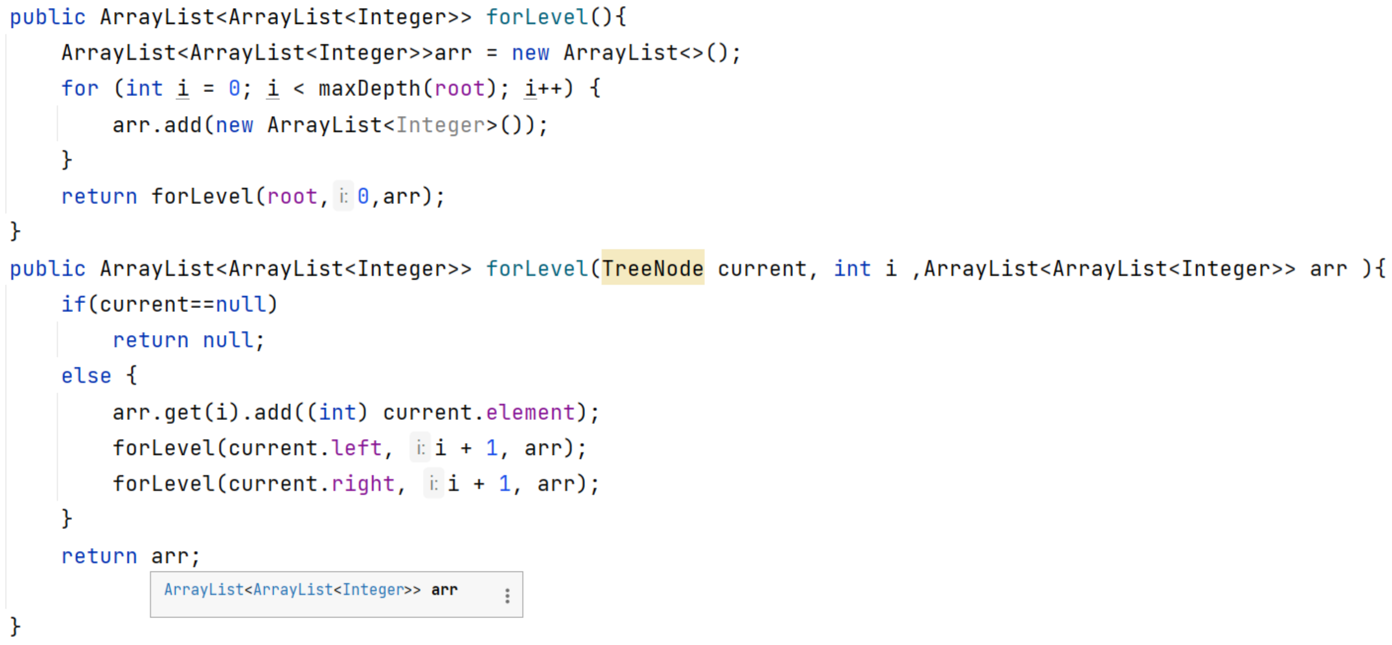
*18



Given the `root` of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Example 1:

```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```



```
public ArrayList<ArrayList<Integer>> forLevel(){
    ArrayList<ArrayList<Integer>>arr = new ArrayList<>();
    for (int i = 0; i < maxDepth(root); i++) {
        arr.add(new ArrayList<Integer>());
    }
    return forLevel(root, i: 0,arr);
}
public ArrayList<ArrayList<Integer>> forLevel(TreeNode current, int i ,ArrayList<ArrayList<Integer>> arr ){
    if(current==null)
        return null;
    else {
        arr.get(i).add((int) current.element);
        forLevel(current.left,  i: i + 1, arr);
        forLevel(current.right,  i: i + 1, arr);
    }
    return arr;
           ArrayList<ArrayList<Integer>> arr
}
```