```java
public void printPreOrder(BTNode<T> n) {
    if(n != null) {
        System.out.println(n.data);
        printPreOrder(n.left);
        printPreOrder(n.right);
    }
}

public void printInOrder(BTNode<T> n) {
    if(n != null) {
        printInOrder(n.left);
        System.out.println(n.data);
        printInOrder(n.right);
    }
}

public void printPostOrder(BTNode<T> n) {
    if(n != null) {
        printPostOrder(n.left);
        printPostOrder(n.right);
        System.out.println(n.data);
    }
}

public static int countLeaf(BSTNode<T> n) {
    if(n == null)
        return 0;
    else if(n.right == null && n.left == null)
        return 1;
    else
        return countLeaf(n.left) + countLeaf(n.right);
}

public static int treeHeight(BSTNode<T> n) {
    if(n == null) {
        return 0;
    }
    else {
        int l = 1 + treeHeight(n.left);
        int r = 1 + treeHeight(n.right);
        if(l > r)
            return l;
        else
            return r;
        // OR: return l > r ? l : r;
    }
}

public static int maxKey(BSTNode<T> n) {
    if(n.right == null)
        return n.key;
    else
        return maxKey(n.right);
}

public static T maxData(BTNode<T> n) {
    if(n.left == null && n.right == null) {
        return n.data;
    }
    else {
        T l = maxData(n.left);
        T r = maxData(n.right);
        if(l.compareTo(n.data) > 0 && l.compareTo(n.data) > 0)
            return l;
        else if(r.compareTo(n.data) > 0 && r.compareTo(l) > 0)
            return r;
        else
            return n.data;
    }
}

public static int countOneChild(BTNode<T> n) {
    if(n == 0)
```

```
        return 0;
    else if(n.left != null && n.right == null)
        return 1 + countOneChild(n.left);
    else if(n.left == null && n.right != null)
        return 1 + countOneChild(n.right);
    else
        return countOneChild(n.left) + countOneChild(n.right);
}

public static boolean isAVL(BSTNode<T> n) {
    if(n == null) {
        return true;
    }
    else {
        int l = 1 + treeHeight(n.left);
        int r = 1 + treeHeight(n.right);
        int balance = r - l;
        if(balance >= -1 && balance <= 1)
            return isAVL(n.left) && isAVL(n.right);
        else
            return false;
    }
}

public int countEven(int[] array, int n) {
    if(n == 0)
        return 0;
    else if(array[n - 1] % 2 == 0)
        return 1 + countEven(array, n - 1)
    else
        return 0 + countEven(array, n - 1)
}

public int sum(int[] array, int n) {
    if(n == 1)
        return a[0];
    else
        return a[n - 1] + sum(array, n - 1);
}

public int multiply(int[] array, int n) {
    if(n == 1)
        return a[0];
    else
        return a[n - 1] * multiply(array, n - 1);
}

public boolean allGreaterThanX(int[] array, int n, int x) {
    if(n == 0)
        return true;
    else
        return (a[n - 1] > x) && allGreaterThanX(array, n - 1, x);
}

public boolean allPositive(int[] array, int n) {
    if(n == 0)
        return true;
    else
        return a[n - 1] >= 0 && allPositive(array, n - 1)
}
```