Vlad Boyden-Tudor, 917
VBT

11)

1) We will traverse the binary tree with Breadth-first search. We will push in the queue not just the nodes, but pairs of numbers - the node and the level. The first in the queue will be (root, 0). Whenever we push in the queue the children of the front of the queue its level will be the level of the front of the queue+1. Also, each time we pop something from the queue we increment the position of the level in a direct table (we use direct table since the height of binary trees is usually not huge).

At the end we iterate over the direct table, starting at both ends and stopping the iteration when the 2 iterators meet in the middle. The values of the left iterator must increase, the values of the right iterator must decrease, and the values of the 2 iterators must be equal.

Vlad Bogdan - Tudor, 917
VladB

2) Node:
info: TValue
left: ↑Node
right: ↑Node

BinaryTree:
root: ↑Node

3) function isDiamond(bt) is:
//pre: bt ∈ BinaryTree
//post: true
if bt.root = NIL then
isDiamond ← false // just an assumption, can be changed
end-if
maxLevel ← 0
init(q) // initialise the queue
init(dt, 1000) // initialise the direct table with 1000 positions
q.push(bt.root, 0)
while isEmpty(q) = false execute:
front ← q.pop()
if [front.first].left ≠ NIL then
q.push([front.first].left, front.second+1)
end-if

Vlad Bogdan-Traian, 917

```
        if [front.first].right ≠ NIL then
            q.push([front.first].right, front.second+1)
        end-if
        dt[front.second] ← dt[front.second]+1
        if front.second > maxLevel then
            maxLevel ← front.second
        end-if
    end-while
    prev ← -1
    i ← 0
    j ← maxLevel
    ok ← true
    while i < j execute
        if prev ≠ -1 then
            if dt[i] ≤ dt[prev] then
                ok ← false
                i ← j+1   // to stop the loop
            end-if
            if dt[i] ≠ dt[j] then
                ok ← false
                i ← j+1
            end-if
        end-if
```

Vlad Bogdan-Tudor, 917
VlBT

```
        prev ← i
        i ← i + 1
        j ← j - 1
    end-while
    is Diamond ← ok
end-function
```

4) The complexity comes from the traversal of the tree (BFS) and the traversal of the direct table. Even tough the second traversal has best/worst case, the first one doesn't and since the first one is $\Theta(n)$ and the second one is $\Theta(n)$ in worst-case, the complexity of the algorithm will be $\underline{\underline{\Theta(n)}}$ (without best/worst cases).