NANYANG TECHNOLOGICAL UNIVERSITY
School of Electrical & Electronic Engineering

EE2008/IM1001 Data Structures and Algorithms

Tutorial No. 6 (Sem 2, AY2021-2022)

1. Traverse the binary tree shown in Figure 1: (a) in preorder; (b) in inorder; (c) in postorder. Show the content of the traversal for each algorithm.

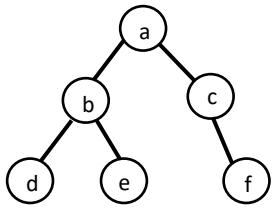


Figure 1

2. **(Understanding the execution of a recursive algorithm.)** For the binary search tree shown in Figure 2, trace the execution of the algorithm `BSTinsert_recurs(.)` step-by-step.

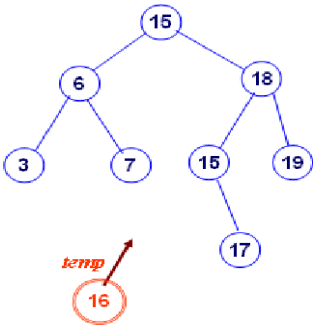
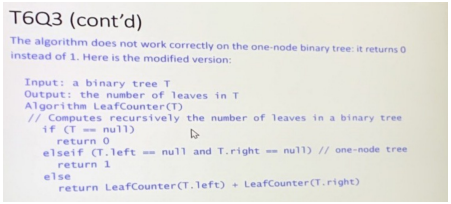


Figure 2

```
BSTinsert_recurs(root, temp) {
  if (temp.data ≤ root.data) {
    if (root.left == null)
      root.left = temp
    else
      BSTinsert_recurs(root.left,temp)
  }
  else { // goes to right subtree
    if (root.right == null)
      root.right = temp
    else
      BSTinsert_recurs (root.right, temp)
  }
}
```


T6Q3 (cont'd)

The algorithm does not work correctly on the one-node binary tree: it returns 0 instead of 1. Here is the modified version:

```
Input: a binary tree T
Output: the number of leaves in T
Algorithm LeafCounter(T)
 // Computes recursively the number of leaves in a binary tree
  if (T == null)
    return 0
  elseif (T.left == null and T.right == null) // one-node tree
    return 1
  else
    return LeafCounter(T.left) + LeafCounter(T.right)
```

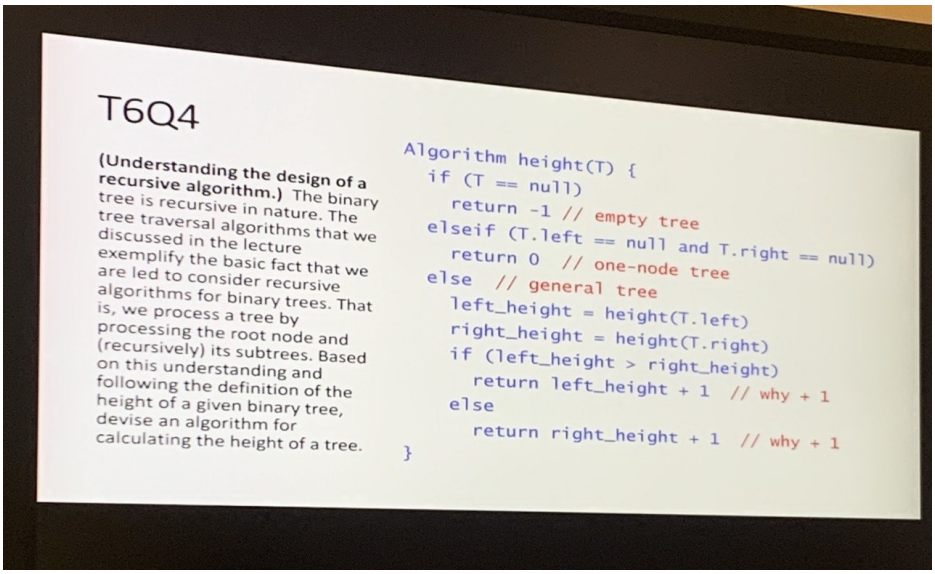
3. The following algorithm is used to compute the number of leaves in a binary tree. Is this algorithm correct? If it is, prove it; if it is not, make an appropriate correction.

```
Input: a binary tree T
Output: the number of leaves in T
Algorithm LeafCounter(T)
  // Computes recursively the number of leaves in a binary tree
  if (T == null)
    return 0
  else
    return LeafCounter(T.left) + LeafCounter(T.right)
```

4. **(Understanding the design of a recursive algorithm.)** The binary tree is recursive in nature. The tree traversal algorithms that we discussed in the lecture exemplify the basic fact that we are led to consider recursive algorithms for binary trees. That is, we process a tree by processing the root node and (recursively) its subtrees. Based on this understanding and following the definition of the height of a given binary tree, devise an algorithm for calculating the height of a tree.

5. Assume that the following algorithms are available for binary trees. Write an algorithm `depth(T, v)` that computes the depth of a node `v` of the tree T.

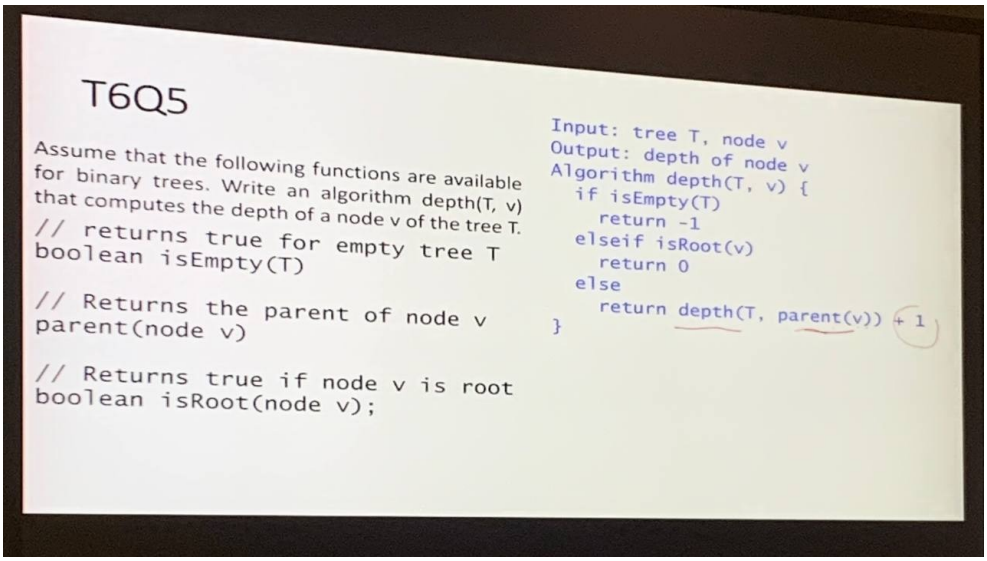
```
// Returns whether the tree T is empty
boolean isEmpty(T)

// Returns the parent of a given node
parent(node v)

// Returns whether a given node is the root of the tree
boolean isRoot(node v);
```



T6Q4

(Understanding the design of a recursive algorithm.) The binary tree is recursive in nature. The tree traversal algorithms that we discussed in the lecture exemplify the basic fact that we are led to consider recursive algorithms for binary trees. That is, we process a tree by processing the root node and (recursively) its subtrees. Based on this understanding and following the definition of the height of a given binary tree, devise an algorithm for calculating the height of a tree.

```
Algorithm height(T) {
  if (T == null)
    return -1 // empty tree
  elseif (T.left == null and T.right == null)
    return 0  // one-node tree
  else  // general tree
    left_height = height(T.left)
    right_height = height(T.right)
    if (left_height > right_height)
      return left_height + 1  // why + 1
    else
      return right_height + 1  // why + 1
}
```



T6Q5

Assume that the following functions are available for binary trees. Write an algorithm depth(T, v) that computes the depth of a node v of the tree T.

```
// returns true for empty tree T
boolean isEmpty(T)

// Returns the parent of node v
parent(node v)

// Returns true if node v is root
boolean isRoot(node v);
```

```
Input: tree T, node v
Output: depth of node v
Algorithm depth(T, v) {
  if isEmpty(T)
    return -1
  elseif isRoot(v)
    return 0
  else
    return depth(T, parent(v)) + 1
}
```