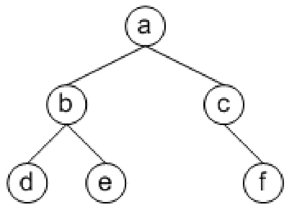
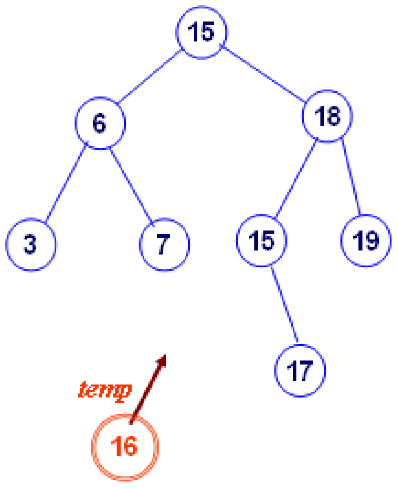
Tutorial 6

1. **Traverse the following binary tree: (a) in preorder; (b) in inorder; (c) in postorder. Show the contents of the traversal as the algorithm progresses.**



2. **(Understanding the execution of a recursive algorithm.) For the binary search tree below, trace step by step the execution of the algorithm *BSTinsert_recurs( )*.**

```
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
        if (root.right = = null)
            root.right = temp
        else
            BSTinsert_recurs (root.right,
temp);
    }
}
```



3. **The following algorithm seeks 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: T (i.e. a binary tree)
Output: the number of leaves in T

Algorithm LeafCounter (T);
    // Computes recursively the number of leaves in a binary tree
    if T == NULL then
        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 tree. 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* that computes the depth of a node *v* of the tree.**

   // 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);

6. **Write a linear-time algorithm that determines whether a binary tree is a binary search tree.**