1.

```
public void reverseQueue(Queue<E> currentQueue){
        if (!currentQueue.isEmpty()){
                item tmp = currentQueue.dequeue();
                currentQueue.reverseQueue();
                currentQueue.enqueue(tmp);
        }
}
```

2.

Post order: c h d e g f b a

3.

```
public boolean isBST(node, root){
        return isBSTExtra(root, Integer.MIN_VALUE, Integer_MAX_VALUE){
}
private boolean isBSTExtra(node root, int min, int max){
        if (root == null){
                return true;
        }
        if(root.getData() > min && root.getData() < max){
                        return (isBSTExtra(root.leftChild, min, Math.min(root.getData(),
max)&&                                  isBSTExtra(root.rightChild, Math.max(root.getData(),
min), max);
        }
        else{
                return false;
        }
}
```

Complexity O(n)

4.

If the root stands by itself or does not exist, then the algorithm gives 0, else, it will give the number of nodes that are not leaves.

EC.

Given post-order and in-order traversal:

We travel from the end of the post-order: The first member will be root, the divide the in-order list into two sub-trees connected the root.

The second member of the post-order will be root of the left sub-tree, use that information to divide the in-order lists into smaller subtree, delete all the known elements along the way in the post-order.

Recursively, we will be able to construct the binary tree.

From pre-order and post-order, we may not be able to reconstruct the tree since we

cannot tell which one is left child and which one is right child.
Example: root A connected to left child B gives preorder AB and post-order BA same as a binary tree with root A connected to right child B.