```
subalgorithm preorder(tree) is:
//pre: tree is a binary tree
    s: Stack //s is an auxiliary stack
    if tree.root ≠ NIL then
        push(s, tree.root)
    end-if
    while not isEmpty(s) execute
        currentNode ← pop(s)
        @visit currentNode
        if [currentNode].right ≠ NIL then
            push(s, [currentNode].right)
        end-if
        if [currentNode].left ≠ NIL then
            push(s, [currentNode].left)
        end-if
    end-while
end-subalgorithm
```

- Time complexity of the non-recursive traversal is $\Theta(n)$, and we also need $O(n)$ extra space (the stack)

```
subalgorithm inorder(tree) is:
//pre: tree is a BinaryTree
    s: Stack //s is an auxiliary stack
    currentNode ← tree.root
    while currentNode ≠ NIL execute
        push(s, currentNode)
        currentNode ← [currentNode].left
    end-while
    while not isEmpty(s) execute
        currentNode ← pop(s)
        @visit currentNode
        currentNode ← [currentNode].right
        while currentNode ≠ NIL execute
            push(s, currentNode)
            currentNode ← [currentNode].left
        end-while
    end-while
end-subalgorithm
```

- Time complexity $\Theta(n)$, extra space complexity $O(n)$

```
subalgorithm postorder(tree) is:
//pre: tree is a BinaryTree
    s: Stack //s is an auxiliary stack
    node ← tree.root
    while node ≠ NIL execute
        if [node].right ≠ NIL then
            push(s, [node].right)
        end-if
        push(s, node)
        node ← [node].left
    end-while
    while not isEmpty(s) execute
        node ← pop(s)
        if [node].right ≠ NIL and (not isEmpty(s)) and [node].right = top(s) th
            pop(s)
            push(s, node)
            node ← [node].right
//continued on the next slide
```

```
        else
            @visit node
            node ← NIL
        end-if
        while node ≠ NIL execute
            if [node].right ≠ NIL then
                push(s, [node].right)
            end-if
            push(s, node)
            node ← [node].left
        end-while
    end-while
end-subalgorithm
```

- Time complexity $\Theta(n)$, extra space complexity $O(n)$