BSTNode:
- info: TElem
- left: ↑ BSTNode
- right: ↑ BSTNode

BinarySearchTree:
- root: ↑ BSTNode

```
function search_rec (node, elem) is:
//pre: node is a BSTNode and elem is the TElem we are searching for
    if node = NIL then
        search_rec ← false
    else
        if [node].info = elem then
            search_rec ← true
        else if [node].info < elem then
            search_rec ← search_rec([node].right, elem)
        else
            search_rec ← search_rec([node].left, elem)
    end-if
end-function
```

- Complexity of the search algorithm: $O(h)$ (which is $O(n)$)

```
function search (tree, elem) is:
//pre: tree is a BinarySearchTree and elem is the TElem we are searching for
    currentNode ← tree.root
    found ← false
    while currentNode ≠ NIL and not found execute
        if [currentNode].info = elem then
            found ← true
        else if [currentNode].info < elem then
            currentNode ← [currentNode].right
        else
            currentNode ← [currentNode].left
        end-if
    end-while
    search ← found
end-function
```

```
function initNode(e) is:
//pre: e is a TComp
//post: initNode ← a node with e as information
    allocate(node)
    [node].info ← e
    [node].left ← NIL
    [node].right ← NIL
    initNode ← node
end-function
```

```
function insert_rec(node, e) is:
//pre: node is a BSTNode, e is TComp
//post: a node containing e was added in the tree starting from node
    if node = NIL then
        node ← initNode(e)
    else if [node].info ≤ e then
        [node].left ← insert_rec([node].left, e)
    else
        [node].right ← insert_rec([node].right, e)
    end-if
    insert_rec ← node
end-function
```

- Complexity: O(n)

```
function minimum(tree) is:
//pre: tree is a BinarySearchTree
//post: minimum ← the minimum value from the tree
    currentNode ← tree.root
    if currentNode = NIL then
        @empty tree, no minimum
    else
        while [currentNode].left ≠ NIL execute
            currentNode ← [currentNode].left
        end-while
        minimum ← [currentNode].info
    end-if
end-function
```

```
function parent(tree, node) is:
//pre: tree is a BinarySearchTree, node is a pointer to a BSTNode, node ≠ NIL
//post: returns the parent of node, or NIL if node is the root
    c ← tree.root
    if c = node then //node is the root
        parent ← NIL
    else
        while c ≠ NIL and [c].left ≠ node and [c].right ≠ node execute
            if [c].info ≥ [node].info then
                c ← [c].left
            else
                c ← [c].right
            end-if
        end-while
        parent ← c
    end-if
end-function
```

- Complexity: $O(n)$

```
function successor(tree, node) is:
//pre: tree is a BinarySearchTree, node is a pointer to a BSTNode, node ≠ NIL
//post: returns the node with the next value after the value from node
//or NIL if node is the maximum
    if [node].right ≠ NIL then
        c ← [node].right
        while [c].left ≠ NIL execute
            c ← [c].left
        end-while
        successor ← c
    else
        p ← parent(tree, c)
        while p ≠ NIL and [p].left ≠ c execute
            c ← p
            p ← parent(tree, p)
        end-while
        successor ← p
    end-if
end-function
```