Binary Search Tree

Insertion

```
Function Insert(root, value):
    if root is NULL:
        return create a new node with value

    if value < root.data:
        root.left = Insert(root.left, value)

    else if value > root.data:
        root.right = Insert(root.right, value)

    return root
```

Search

```
Function Search(root, target):
    if root is NULL:
        return False   // target not found

    if root.data == target:
        return True     // target found

    else if target < root.data:
        return Search(root.left, target)

    else:
        return Search(root.right, target)
```

Deletion

```
Function DeleteNode(root, key):
    if root is NULL:
        return NULL

    if key < root.data:
        root.left = DeleteNode(root.left, key)

    else if key > root.data:
        root.right = DeleteNode(root.right, key)

    else:
        if root.left == NULL and root.right == NULL:
            delete root
            return NULL

        else if root.left == NULL:
            temp = root.right
            delete root
            return temp

        else if root.right == NULL:
            temp = root.left
            delete root
            return temp

        else:
            temp = FindMin(root.right)
            root.data = temp.data
            root.right = DeleteNode(root.right, temp.data)

    return root


Function FindMin(node):
    while node.left ≠ NULL:
        node = node.left
    return node
```

Minimum in BST

```
Function FindMin(node):
    current = node
    while current.left ≠ NULL:
        current = current.left
    return current
```

Maximum in BST

```
Function FindMax(node):
    current = node
    while current.right ≠ NULL:
        current = current.right
    return current
```

Tree Traversals

Inorder Traversal (Left → Root → Right)

```
Function Inorder(node):
    if node ≠ NULL:
        Inorder(node.left)
        visit(node)
        Inorder(node.right)
```

Preorder Traversal (Root → Left → Right)

```
Function Preorder(node):
    if node ≠ NULL:
        visit(node)
        Preorder(node.left)
        Preorder(node.right)
```

Postorder Traversal (Left → Right → Root)

```
Function Postorder(node):
    if node ≠ NULL:
        Postorder(node.left)
        Postorder(node.right)
        visit(node)
```

Successor

```
Function findMin(root):
    while root ≠ NULL and root.left ≠ NULL:
        root = root.left
    return root
```

```
Function findSuccessor(root, target):
    if target.right ≠ NULL:
        return findMin(target.right)

    successor = NULL
    while root ≠ NULL:
        if target.key < root.key:
            successor = root
            root = root.left
        else if target.key > root.key:
            root = root.right
        else:
            break

    return successor
```

Predecessor

```
Function findMax(root):
    while root ≠ NULL and root.right ≠ NULL:
        root = root.right
    return root
```

```
Function findPredecessor(root, target):
    if target.left ≠ NULL:
        return findMax(target.left)

    predecessor = NULL
    while root ≠ NULL:
        if target.key > root.key:
            predecessor = root
            root = root.right
        else if target.key < root.key:
            root = root.left
        else:
            break

    return predecessor
```