Algorithms & Data Structures
Assessed Exercise

Marcus Lancaster — Student №: 2339874L

Add Element Algorithm

```
SET current node TO root node
IF current node EQUALS null:
    SET root node TO added element
    TERMINATE
WHILE current node NOT EQUALS null:
    SET parent node TO current node
    IF added element EQUALS current node:
        INCREMENT CountedElement count
        TERMINATE
    ELSE IF added element GREATER THAN current element:
        SET current element TO current right node
        IF current element EQUALS null:
            SET parent right node TO added element
            TERMINATE
    ELSE:
        SET current element TO current left node:
        IF current element EQUALS null:
            SET parent left node TO added element
            TERMINATE
TERMINATE
```

Remove Element Algorithm

```
FUNCTION findReplacement(node):
    IF left node EQUALS null:
        RETURN right node
    ELSE IF right node EQUALS null:
        RETURN left node
    ELSE:
        SET search node TO right node
        WHILE search node left child NOT EQUALS null:
            SET search node TO search node left child
        SET return node TO search node
        SET search node TO findReplacement(search node)
        RETURN return node

SET parent node TO null
SET current node TO root node
WHILE current node NOT null:
    IF current node EQUALS remove node:
        IF current node's count GREATER THAN 1:
            DECREMENT current node's count
            TERMINATE
        ELSE:
```

```
            SET replacement TO findReplacement(current node);
            IF current node EQUALS root node:
                SET root node TO replacement
            ELSE IF current node EQUALS parent left node:
                SET parent left node TO  replacement
            ELSE:
                SET parent right node TO replacement
            TERMINATE
    ELSE:
        SET parent node TO current node
        IF current node GREATER THAN remove node:
            SET current node TO parent right node
        ELSE:
            SET current node TO parent left node
TERMINATE
```

Iterator Algorithm

```
CREATE NEW stack
SET current node TO root node
WHILE current node NOT null:
    ADD current node TO stack
    SET current node TO current left node
WHILE stack NOT empty:
    SET return node TO stack.pop()
    SET subtree node TO return right node
    WHILE subtree node NOT null:
        ADD subtree node TO stack
        SET subtree node TO subtree right node
    RETURN return node
TERMINATE
```