FIT1008 – Intro to Computer Science
Solutions to Tutorial 12

Semester 1, 2018

Exercise 1

The expression is $((1/3) + ((6 * 7)/4))$.

- **Prefix:** $+ \quad / \quad 1 \quad 3 \quad / \quad * \quad 6 \quad 7 \quad 4$
- **Infix:** $1 \quad / \quad 3 \quad + \quad 6 \quad * \quad 7 \quad / \quad 4$
- **Postfix:** $1 \quad 3 \quad / \quad 6 \quad 7 \quad * \quad 4 \quad / \quad +$

Exercise 2

```
def height(self):
    return self._aux_len(self.root)

def height_aux(self, current):
    if current is None:
        return -1
    else:
        return 1 + max(self.height_aux(current.left)
                  , self.height_aux(current.right))
```

Exercise 3

```
def sum_leaves(self):
        return sum_leaves_aux(root);

def sum_leaves_aux(current):
    if current is None:
        return 0
    elif current.left is None and current.right is None:
        return current.item
    else:
        return sum_leaves_aux(current.left)
                    + sum_leaves_aux(current.right)
```

Exercise 4

The first definition goes all the way down the leftmost branch and then returns `null`. The problem is that the condition for the if-then-else is `current is not None` rather than `current.left is not None`. Additionally, it returns `current`, rather than the key of the node.

The second definition stops right at the root node (since its left child is not empty) and thus returns a reference to the root node. The problem is that the THEN and ELSE branches of the if-then-else are swapped (or the condition negated). It also suffers from the same problem as the one before: it returns `current` rather than `current.key`

A correct definition is:

```
def find_min(self):
    if self.root is None:
        return None
    else:
        return self.find_min_aux(self.root)

def find_min_aux(self, current):
    if current.left is None:
        return current.key
    else:
        return find_min_aux(current.left)
```

This method is linear (only one recursive call) and direct (calls itself). It is also tail recursive and can therefore be converted to a simple iteration without the need of a stack.
The value returned by this method for the tree given in the figure would be 3.

Exercise 5

```
    def tree_range(self, a, b):
        ans = []
        self._range(a, b, self.root, ans)
        return ans


    def _range(self, a, b, current, a_list):
        if current is None:
            return
        if a < current.key:
            self._range(a, b, current.left, a_list)
        if a <= current.key <= b:
            a_list.append(current.key)
        if current.key < b:
            self._range(a, b, current.right, a_list)
```

Exercise 6

```
    def k_largest(self, k):
        a_list = []
        self._k_largest(self.root, k, a_list)
        return a_list[-1]

    def _k_largest(self, current, k, a_list):
        if current is not None:
            ans = self._k_largest(current.right, k, a_list)
            if ans is not None:
                return ans
            a_list.append(current.key)
            if len(a_list) == k:
                return a_list
            ans = self._k_largest(current.left, k, a_list)
            if ans is not None:
                return ans
```