# *FIT1008 – Intro to Computer Science*
# *Tutorial 12*

Semester 1, 2017

## *Objectives of this tutorial*

- To understand Binary Trees
- To understand Binary Search Trees.

## *Exercise 1*

A binary expression tree is a binary tree used to represent algebraic expressions composed of unary and binary operators. The leaves of a binary expression tree are operands, such as constants or variable names, and the other nodes contain operations.

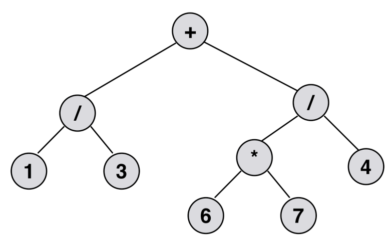



Figure 1: Expression three

The prefix notation of an algebraic expression results from traversing the corresponding expression tree in pre-order. The infix notation results from traversing the tree in in-order; and the postfix notation (or reverse polish notation) results from traversing the tree in post-order.
Give the unambiguous mathematical expression as well as the prefix, infix and postfix notation of the expression represented by the tree above.

## *Exercise 2*

Consider a `BinaryTree` class which defines a binary tree data type implemented using linked nodes, defined as follows:

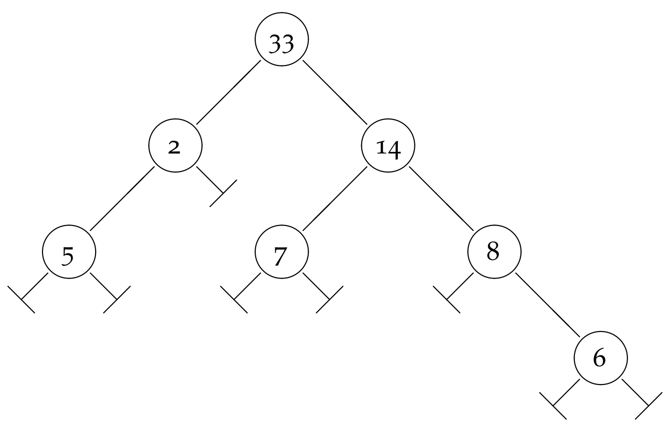
```
class TreeNode:
    def __init__(self, new_item=None, left=None, right=None):
            self.item = new_item
            self.left = left
            self.right = right

class BinaryTree:
```

```
    def __init__(self):
            self.root = None
```

Write down an attribute method for the class that returns the size of the Binary Tree.

## *Exercise 3*

Add to the class above the method `sum_leaves(self)` which returns 0 if the tree is empty and, otherwise, returns the result of adding the value of every leaf in the tree. For example, for `a_tree` of the form:




the result of `a_tree.sum_leaves()` would be $5 + 7 + 6 = 18$.

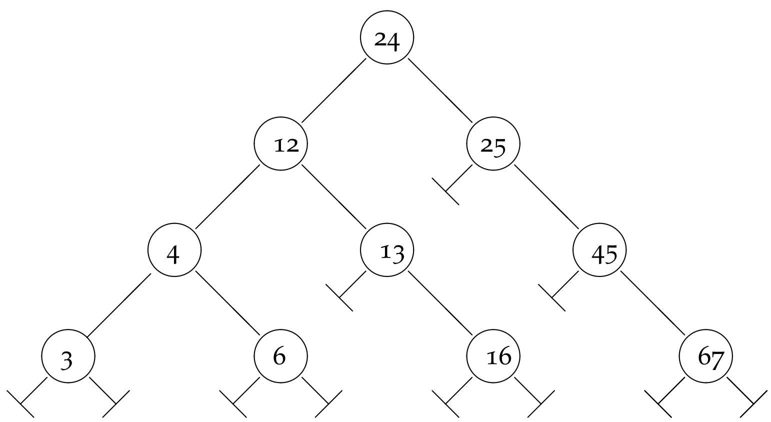
## *Exercise 4*

We want to extend the `BinarySearchTree` class defined in the lectures by adding a method `find_min()`. This method returns the minimum key in the tree, or `None` if the tree is empty. In doing so, it does not modify the tree. The following code shows two failed attempts at an implementation of such method:

```
def find_min_1(self):
    return self.find_min_aux_1(self.root)

def find_min_aux_1(self, current):
    if current is not None:
        return self.find_min_aux_1(current.left)
    else:
        return current

def find_min_2(self):
    if self.root is None:
```

```
        return self.root
    else:
        return self.find_min_aux_2(root)

def find_min_aux_2(self, current):
    if current.left is not None:
        return current
    else:
        return self.find_min_aux_2(current.left)
```

Consider a tree `the_tree` with integer keys with the form:




1. Show the value of result after calling `result = the_tree.find_min()` for each definition above.
2. Provide a correct definition for the above method.

## *Exercise 5*

Consider a binary search tree that uses strings as keys. Insert the following 14 names into such an empty binary search tree in the order they appear and draw the final tree:

**Jan, Guy, Jon, Ann, Jim, Eva, Amy, Tim, Ron, Kim, Tom, Roy, Kay, Dot**

For the resulting tree, write the names in the order given by a preorder, inorder, and postorder traversal, respectively.