CSC1048 Computability and Complexity

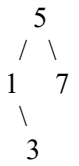
Functional Programming Lab 5: Abstract Data Types and Higher-order Functions

Aim

The aim of this week's exercise is to help you to learn to use algebraic data types and higher-order functions.

1. Monkey Puzzle Sort

A binary search tree has a value at each node. The left subtree of each node holds only values less than at the node, and the right subtree holds only values greater than or equal to that at the node. For example, the following is a binary search tree:



This can be represented in Haskell by creating a data type for a binary tree.

```
data BinTree t = Empty | Root t (BinTree t) (BinTree t)
                 deriving (Eq, Ord, Show)
```

The above binary tree would then be repersented as:

```
myTree = Root 5 (Root  1 (Empty) (Root 3 Empty Empty))
                (Root 7 Empty Empty)
```

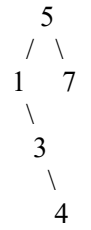
Since the leaf node pattern of `Root x Empty Empty` will occur often we can define this as

```
leaf x = Root x Enpty Empty
```

and hence the above binary tree could be defined as

```
myTree1 = Root 5 (Root  1 (Empty) (Leaf 3)
                 (Leaf 7)
```

(a) Write a recursive function addnode :: Ord a => a -> BinTree a -> BinTree a which, when given an integer and a binary search tree, will add the integer at the correct position in the tree. In order to insert the integer at the correct position, if the integer is less than the value at the current node, then it is placed in the left subtree, otherwise it is placed in the right subtree. The integer is placed at the first unoccupied node (a leaf). For example, if the integer 4 were added to the above tree, the following tree would result:



(b) Write a recursive function maketree :: Ord a => [a] -> BinTree a which, when given a list of integers, will create a binary search tree by inserting the head of the list into the correct position in the tree created from the tail of the list. For example, applying maketree to the list [4,3,1,7,5] would create the tree given above. This function should make use of the addnode function.

(c) The values in a tree can be converted into a list by traversing the tree in a specified order. For example, an inorder traversal traverses the left subtree first, then places the root in the result, and then traverses the right subtree. Write a recursive function inorder :: BinTree a -> [a] which, when given a tree, will return the list giving the result of an inorder traversal of the tree. For example, applying inorder to the tree given above would give the list [1,3,4,5,7].

(d) Monkey puzzle sort works by creating a binary search tree from a list, and then traversing the list in inorder. Write a function mpsort :: Ord a => [a] -> [a] which, when given a list of integers, will return the list in ascending numerical order using monkey puzzle sort. For example:

```
mpsort [4,3,1,7,5] = [1,3,4,5,7]
```

This function should make use of the maketree and inorder functions.

7. Higher Order Sort

Write a higher order sort function hosort :: (a -> a -> Bool) -> [a] -> [a] which takes two arguments: a relative ordering relation and the list to be sorted. The list should be sorted according to the given ordering relation (all consecutive elements in the list satisfy the relation). For example:

```
hosort (>) [4,3,1,7,5] = [7,5,4,3,1]
```