HW 20 **Prof. Caldwell**
Due: 13 November 2007 **COSC 3015**

Here is the code for an implementation of sets as lists.

```
module Set where

import Lists

newtype (Eq a, Show a) => Set a = MkSet [a]

instance (Eq a, Show a ) => Show (Set a) where
   show m = "{" ++ contents ++ "}"
     where MkSet m' = m
           contents = reverse (drop 1 (reverse ( drop 1 (show (unique m')))))

instance (Eq a, Show a ) => Eq (Set a) where
    s == t = all ((flip elem) s') t' &&  all ((flip elem) t') s'
               where MkSet s' = s
                     MkSet t' = t

insert x (MkSet m) = MkSet (x:m)
delete x (MkSet m) = MkSet (remove_all x m)
union (MkSet m) (MkSet n) = MkSet (m ++ n)
intersection (MkSet m) (MkSet n) = MkSet (filter ((flip elem) n) m)
ismem x (MkSet m) = elem x m
```

Here is a module containing the supporting definitions for operations on lists.

```
module Lists where

unique [] = []
unique (h:t) = if (elem h t) then (unique t ) else h:(unique t)

remove_all x = filter (not . (==x))
```

Exercise 0.1. Write Haskell code to implement Sets as Binary search trees by modifying the Set module presented above. You'll have to reimplement `show`, `==`, `insert`, `delete`, `union`, `intersection`, and `ismem` in terms of the operations on binary search trees.