```
-- Pedro Henrique Tôrres Santos (phts)

-- Questão 2
import Data.List (sort)

data CInt = Conjunto [Int] deriving (Show)

getInts :: CInt -> [Int]
getInts (Conjunto a) = a

-- Questão 2 Letra A
makeSet :: [Int] -> CInt
makeSet [] = Conjunto []
makeSet (x:xs) = Conjunto . sort $ if x `notElem` xs then x:ints else ints
   where ints = getInts $ makeSet xs

-- Questão 2 Letra B
union :: CInt -> CInt -> CInt
union a b = makeSet $ getInts a ++ getInts b

-- Questão 2 Letra C
mapSet :: (Int -> Int) -> CInt -> CInt
mapSet f a = makeSet $ map f (getInts a)

-- Questão 3
type Texto = String
type Id = String
type DataHoraPub = Int

data Post = Post (Id,DataHoraPub) Texto deriving (Show,Eq)
data Thread = Nil | T Post (Thread)

getId :: Post -> Id
getId (Post (id,_) _) = id

getDataHoraPub :: Post -> DataHoraPub
getDataHoraPub (Post (_,dataHoraPub) _) = dataHoraPub

getTexto :: Post -> Texto
getTexto (Post _ texto) = texto

isNil :: Thread -> Bool
isNil (Nil) = True
isNil _ = False

-- Questão 3 Letra A
instance Show Thread where
   show (Nil) = []
   show (T p (t)) = "(" ++ getId p ++ " " ++ show (getDataHoraPub p) ++ " " ++ getTexto p ++ ")" +
+ show t

-- Questão 3 Letra B
```

```
inserirPost :: Post -> Thread -> Thread
inserirPost p (Nil) = T p (Nil)
inserirPost p (T p' (t)) = T p' (inserirPost p t)

-- Questão 3 Letra C
threadToList :: Thread -> [Post]
threadToList (Nil) = []
threadToList (T p (t)) = p:(threadToList t)

-- Questão 3 Letra D
listToThread :: [Post] -> Thread
listToThread [] = Nil
listToThread (x:xs) = T x (listToThread xs)

-- Questão 3 Letra E
removerPost :: (Id,DataHoraPub) -> Thread -> Thread
removerPost (id,dhp) t = listToThread $ filter (\e -> getId e /= id || getDataHoraPub e /= dhp) $
threadToList t

-- Questão 1
s1 = (+ 2)
s2 = (> 3)
```