Code (Wednesday Week 3)

Mersenne Prime Conjecture

```
import Test.QuickCheck
divides :: Integer -> Integer -> Bool
m `divides` n = (n `mod` m == 0)

isPrime :: Integer -> Bool
isPrime n
  | n<=1 = False
  | n==2 = True
  | otherwise = not $ any (`divides` n) [2 .. (n-1)]

propMersenne :: Integer -> Property
propMersenne n = (isPrime n ==> isPrime ((2 ^ n) - 1))
```

Ransom Note

```
module Ransom where
import Data.Char
import qualified Data.Map as M
import Data.List (delete)
import Test.QuickCheck


newtype Counts =
    Counts (M.Map Char Int) deriving (Eq, Show)

emptyCounts =
    Counts ( M.fromList (map (\x -> (x,0)) [minBound .. maxBound]))

instance Ord Counts where
    Counts m1 <= Counts m2 =
        and (M.elems (M.intersectionWith (<=) m1 m2))

increase :: Char -> Counts -> Counts
increase c (Counts m) = Counts (M.adjust (+1) c m)

countsFor :: String -> Counts
countsFor xs = foldr increase emptyCounts xs
```

```
rnote :: String -> String -> Bool
rnote message magazine =
     countsFor message <= countsFor magazine

count ::  Char -> String -> Int
count c s = length (filter (==c) s)

lookupCounts :: Char -> Counts -> Int
lookupCounts c (Counts m) = M.findWithDefault 0 c m

propCounts s c =
    lookupCounts c (countsFor s) == count c s

rnote' message magazine
  = all (\c -> count c message <= count c magazine) message

proprnote message magazine
  = rnote message magazine == rnote' message magazine


rnote'' [] magazine = True
rnote'' (x:xs) magazine
 = x `elem` magazine &&
  rnote'' xs (delete x magazine)

proprnote' message magazine
 = rnote message magazine == rnote'' message magazine

proplength :: String -> String -> Property
proplength message magazine
 = length magazine < length message ==>
    not (rnote message magazine)

proprev mes mag = rnote mes mag == rnote (reverse mes) mag
```