1. (a) Define a Lambda function that takes two arguments and returns the first one (discarding the second).

[5 marks]

(b) Explain the following concepts, making use of an example in Haskell to aid your explanation:
   - Referential Transparency
   - Currying
   - Higher Order Functions

[10 marks]

(c) The bubble sort is defined as follows:
   Sort by comparing each adjacent pair of items in a list in turn, swapping the items if necessary, and repeating the pass through the list until no swaps are done.\(^1\)
   Implement the bubble sort in Haskell.

[20 marks]

(d) If natural numbers are defined as follows:

    data Nat = Zero | Succ Nat

Define a function

\(^1\)http://www.itl.nist.gov/div897/sqg/dads/HTML/bubblesort.html
first:: n -> [Nat]

that populates a list with the first n naturals.

(e) Make Nat an element of Eq a (hint: (==) is a function that takes an a, then another a and returns a Bool).
2. (a) (i) Define the type `Time` in Haskell that will store time in hours, minutes and seconds.

(ii) Make `Time` an element of the `Show a` class (hint: `show` takes `a` and returns a `String`). An example output is ‘14:47:23’

(iii) Make `Time` an element of the `Eq a` class (hint `==` takes `a`, another `a` and returns a `Bool`)

(iv) Implement

\[ (+) :: \text{Time} \to \text{Time} \to \text{Time} \]

(b) With reference to your assignment, define the data type `MyBoard` that represents a Nonogram board.

(c) Now define the following:

\[
\begin{align*}
\text{width} & :: \text{MyBoard} \to \text{Int} \\
\text{isOn} & :: \text{MyBoard} \to (\text{XPos}, \text{YPos}) \to \text{Bool} \\
\text{rowDetails} & :: \text{MyBoard} \to [[[\text{Int}]]] \\
\text{colDetails} & :: \text{MyBoard} \to [[[\text{Int}]]]
\end{align*}
\]

(d) Given a range of integers by its lower and upper limit, construct a list of all prime numbers in that range. For example `primesR 10 20` should give \([11, 13, 17, 19]\).
3. The game of paper, rock and scissors should be familiar to all. Two players simultaneously call 'paper', 'rock' or 'scissors' or show it with their hand and according to some rules a winner is declared. With reference to the code snippet below:

```haskell
-- moves in paper, rock, scissors game
data Move =
    Paper -- beats rock
  | Rock -- beats scissors
  | Scissors -- beats paper
  deriving (Show)

-- possible winners
data Winner =
    Player1 | Player2 | Draw
  deriving (Eq)

type Game = (Move,Move)
type Games = [Game]

-- given two moves find out who wins
winner :: Move -> Move -> Winner

-- calculate the score for a number of games
-- so if player one wins one games score is (1,0)
score :: Games -> (Int,Int)

-- determine the overall winner
overallWinner :: Games -> Winner

-- generates a bunch of games
-- seemingly random
-- you give it first game to generate
-- then the offset is given as an argument
-- third argument is number of games to generate
generate :: Game -> (Int,Int) -> Int -> Games

(a) Make Winner an instance of show with the following outputs

- Player1 = “Player One”
- Player2 = “Player Two”
- Draw = “No Winner”

[10 marks]
(b) Write the definitions of the functions in 3(a).

`generate` takes an initial game and then takes two integers. These two integers define the offset of each move. So an offset of 1 starting from Paper gives Paper, Rock, Scissors, ... an offset of 2 gives Paper, Scissors, Rock, ... and an offset of 3 gives Paper, Paper, Paper, ...! The third argument is the number of games to generate. For example

```
  generate (Paper, Rock) (1, 2) 2
gives
  [(Rock, Paper), (Scissors, Scissors)]
```

[40 marks]