Question 1

1)

```
pAverage :: (Fractional a) => [a] -> a
pAverage ls = sum ls / fromRational (toRational (length ls))
```

2)

```
hAverage :: (Fractional a) => a -> a -> [a] -> a
hAverage _   avg []      = avg
hAverage len avg (x:xs) =
  hAverage (len + 1) (((avg * len) + x) / (len + 1)) xs
```

3)

```
average :: (Fractional a) => [a] -> a
average = hAverage 0 0
```

4)

assume head', addLen and avg given below,

```
head' [] = 0

head' (x:xs)
= x

addLen len []
= len

addLen len (x:xs)
= len + 1

avg
= (sum / len)
```

iteration invariant,

```
h_average len avg xs
= ((avg * len) + head' xs) / addLen len xs
```

5)

Proof h_average is satisfied by Iteration Invariant,

```
LHS(11) : h_average len avg []
          = ((avg * len) + head' []) / (addLen len [])           [ apply assump. avg    ]
          = (((sum / len) * len) + head' []) / (addLen len [])   [ apply assump. addLen ]
          = (((sum / len) * len) + head' []) / len               [ apply assump. head'  ]
          = (((sum / len) * len) + 0) / len                      [ simplify             ]
          = (sum + 0) / len                                      [ simplify             ]
          = sum / len                                            [ unapply assump. avg  ]
          = avg : RHS(11)

LHS(12) : h_average len avg (x:xs)
          = ((avg * len) + head' (x:xs)) / (addLen len (x:xs))
                                                        [ apply assump. avg    ]
          = (((sum / len) * len) + head' (x:xs)) / (addLen len (x:xs))
                                                        [ apply assump. addLen ]
          = (((sum / len) * len) + head' (x:xs)) / (len + 1)
                                                        [ apply assump. head'  ]
          = (((sum / len) * len) + x) / (len + 1)
                                                        [ unapply assump. avg  ]
          = ((avg * len) + x) / (len + 1)
                                                        [ as desired           ]
          = h_average (len + 1) (((avg * len) + x) / (len + 1)) xs : RHS(13)
```

6)

QuickCheck

a)

```haskell
prop1 :: [Float] -> Bool
prop1 xs =
  let l = length xs
  in  (average [left, right] == ((left + right) / 2))
      where
        left = average (take l xs)
        right = average (drop (l-1) xs)
        l = length xs
{-
*Q1 Test.QuickCheck> quickCheck $ prop1
+++ OK, passed 100 tests.
-}
```

b)

```
prop2 :: [Float] -> Property
prop2 xs =
  l > 0 ==> avgXS == ((avgXS' * l') + last xs) / convert l
  where
    l    = length xs
    l'   =  convert (l - 1)
    xs' = take (l-1) xs
    avgXS  = average xs
    avgXS' = average xs'
    convert = fromRational . toRational
{-
*Q1 Test.QuickCheck> quickCheck $ prop2
+++ OK, passed 100 tests; 23 discarded.
*Q1 Test.QuickCheck>
-}
```