```
sum :: [Int] -> Int

sum []     = 0
sum (x:xs) = x + sum xs
```

```
elem :: Eq a => a -> [a] -> Bool
elem :: Int -> [Int] -> Bool

elem y []     = False
elem y (x:xs) = x == y || elem y xs
```

```
reverse :: [a] -> [a]
reverse []     = []
reverse (x:xs) = reverse xs ++ [x]
```

1

sum [1,2,3,4] = 1 + sum [2,3,4]

= 1 + 2 + sum [3,4]

= 1 + 2 + 3 + sum [4]

=

= a + sum xs

WANT / INVARIANT

sum' acc xs = acc + sum xs

```
sum' acc []     = acc
sum' acc (x:xs) = sum' (acc+x) xs

sum xs = sum' 0 xs
```

```
reverse [1,2,3] = reverse [2,3] ++ [1]
                = reverse [3] ++ [2] ++ [1]
                = reverse [] ++ [3] ++ [2] ++ [1]
```

INVARIANT

```
rev acc xs = reverse xs ++ acc
```

```
rev acc []     = acc
rev acc (x:xs) = rev (x:acc) xs

reverse xs = rev [] xs
```

```
fizzBuzz (m,n) []     = ([],[],[])

fizzBuzz (m,n) (x:xs) =
  case (x `mod` m == 0
       ,x `mod` n == 0
       ) of
    (True  ,_)     -> (x:ms,ns,os)
    (False,True)   -> (ms,x:ns,os)
    (False,False)  -> (ms,ns,x:os)
  where
    (ms,ns,os) = fizzBuzz (m,n) xs
```