1 The problem

This is a problem 72 from Project Euler.

2 Straight solution

```
import Control.Monad (guard)
import Math.Sieve.Factor
import qualified Math.Sieve.Phi as Phi
```

Let's start with naive approach:

```
max_d = 800
```

```
fraction_list_naive :: [(Int, Int)]
fraction_list_naive = do d ← [1 .. max_d]
  n ← [1 .. d − 1]
  guard (gcd n d ≡ 1)
  return (n, d)
```

It takes too long to compute even with $max\_d = 10000$.

3 Generalizing *fraction_list*

Let's generalize *fraction_list* by allowing to supply our own function which's going to return $\forall x : x : GCD(x, n) = 1, x < n$:

```
fraction_list :: (a → Int → [Int]) → a → [(Int, Int)]
fraction_list genr c = do d ← [1 .. max_d]
  n ← genr c d
  return (n, d)
```

What's the best approach to *genr* function? Naive one is simple:

```
genr_naive :: () → Int → [Int]
genr_naive () n = filter ((≡ 1) ∘ gcd n) [1 .. n − 1]
```

Another approach is to factorize n:

```
divides :: Int → Int → Bool
divides a b = a `mod` b ≡ 0
```

```
genr_factor :: FactorSieve → Int → [Int]
genr_factor si n = if length factors ≡ 0
  then [1 .. n − 1]
```

else $filter\ (\lambda d \rightarrow \neg\ \$\ any\ (d`divides`)\ factors)\ [1\,..\,n-1]$
where $factors = map\ fst\ \$\ factor\ si\ n$

$fr\_list = fraction\_list\ genr\_factor\ (sieve\ max\_d)$

The new appoarch becoming faster with large n's ($n = 800$ shows better results). But 11 seconds are way too long!

We can try to optimize *genr_factor* further by constructing a sieve, but I have a nicer idea.

4 Euler's Function

What we need is a count of pairs (n, d), where $y <= d_m ax, x < y, gcd(x, y) == 1$. We have a nice $\varphi(d)$ function, which is basically what we need but for some concrete d. Let's combine results of $\varphi(d)$.

Let's define P(d) as a set of all pairs (d, x), where x is a coprime to d, lower than d:

$$P(d) = (n, d) : n <= d, gcd(n, d) == 1$$

If we have d_1 and d_2 (d_1 != d_2), then $P(d_1) \cap P(d_2) = \varnothing$, because in every pair the greater number is second and it's different for both sets. That means that we can just add $\varphi(d_1)$ to $\varphi(d_2)$, or:

$$|P(1) \cup P(2) \cup ... \cup P(n)| = \varphi(1) + ... + \varphi(n)$$

Let's write it down is Haskell:

$coprimes\_total\ n = sum\ \$\ map\ (Phi.phi\ si)\ [2\,..\,n]$
where $si = Phi.sieve\ n$

That simple? I ought to be missing something! But the answer seems correct, cool