CA320 Computability & Complexity

Assignment 2: Perfect Numbers and Higher-order functions in Haskell

Student Name: David Weir

Student Number: 19433086

Declaration on Plagiarism

Assignment Submission Form

This form must be filled in and completed by the student(s) submitting an assignment

Name(s): David Weir

Programme: Computability & Complexity

Module Code: CA320

Assignment Title: Perfect Numbers and Higher-order functions in Haskell

Submission Date: 27/11/2021

Module Coordinator: Dr. David Sinclair

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at http://www.dcu.ie/info/regulations/plagiarism.shtml , https://www4.dcu.ie/students/az/plagiarism and/or recommended in the assignment guidelines.

Name(s): David Weir　　　　　　Date: 27/11/2021

Assignment Task

Write a Haskell program that sums all the perfect numbers between 1 and 1,000,000,000,000. Use higher-order functions as much as possible. A perfect number is a positive integer that is equal to the sum of its factors, excluding itself. For example, 6 is a perfect number since its factors are 1,2,3 and 6, and the sum of its factors excluding itself is 1+2+3 = 6.

The Design

The design of my application is based on the interlinked topics of Mersenne primes and perfect numbers. A Mersenne Prime is a prime number that can be expressed as "2^p -1", where the resulting number is prime. For example, 2^2 = 4 – 1 = 3, which is prime so therefore 3 is a Mersenne prime however, (2^11) – 1 = 2047 which is not prime and therefore 11 is not a Mersenne prime.

There is a one-to-one relationship between Mersenne primes and perfect numbers, proven by the Euclid-Euler theorem. Under this theorem even numbers are perfect numbers if they can be expressed as "2^(p-1) x ((2^p) -1)". Here 2^p -1 represents a Mersenne prime as discussed above. However, the salient point is that each Mersenne prime correlates to some perfect number through this equation.

The Program

The main challenge of this application is to create an efficient primality checker to determine if any given number is a prime. When working with such large numbers this becomes difficult to do efficiently.

To begin, I created a simple "isprime" function to do just that. The functionality of this function is split into 2: isprime and factors.

The "factors" function, seen below, takes in an Int type and returns and list of Ints, this list contains the factors of the number passed through the function. For example, if 31 was passed in then factors with return [1], as the only factors of 31 are 1 and itself.

The function uses list comprehension in Haskell to check if any number less than the floored square root of given Int divides into the Int evenly. This provides a list of factors for the given Int type number, which will only contain [1], if it is prime.

```
factors :: Int -> [Int]
factors num = [x | x <- [1..(floor (sqrt (fromIntegral num)))], num `rem` x == 0]
```

The counterpart to "factors" is the function "isprime", that takes a number (Int) and returns a Boolean value. This value is True if the number is prime and False otherwise. Isprime works by checking if the length of the list of factors is 1. If it is equal to 1 isprime returns True as the number is a prime.

```
isprime :: Int -> Bool
isprime num = length (factors num) == 1
```

However, this is still quite inefficient for larger numbers, so I used the Lucas-Lehmer test. This function takes an integer and returns a Boolean value. Under this test, we have a prime number "p" and a Mersenne prime M_p and we create a sequence beginning with $s_0 = 4$ and continuing for s>0 using $s_k = (s_{k-1}^2 - 2) \bmod M_p$. In this test M_p is only prime if $s_{p-2} = 0$ for any prime > 2.

In Haskell, this means we set 2 to be True always as it is known to be prime. For any number greater than 2, I first check that the number itself is prime, then we use $s_{p-2} = 0$ to check if M_p is a Mersenne prime. To do this, I set Mersenne or "mers" to be equal to its formula $M_n = 2^n - 1$. I then set s0 to equal 4 to begin the sequence. After this, every other number in the sequence is calculated using $s_k = (s_{k-1}^2 - 2) \bmod M_p$.

```
ll :: Int -> Bool
ll 2 = True
ll num = isprime num && s (num-2) == 0
    where
        mers = (2^num) - 1
        s 0 = 4
        s num =
            let n = s (num-1)
                in ((n^2)-2) `rem` mers
```

Finally, I create the sumperfect function that outputs the final sum of all perfect numbers under 1 trillion. To do this, I create a list using list comprehension. This list comprehension is made by using Haskell's lazy implementation of lists to iterate through an infinite list. This list is then filtered using the Lucas-Lehmer function so we are only dealing with Mersenne primes. Using these Mersenne primes I calculate their corresponding perfect number using the 2^(x - 1)*((2^x)-1 equation. This process is looped through until we reach a perfect number that is <= 1 trillion, this set stop point is made using the takeWhile function. This all results in a list: [6,28,496,8128,33550336,8589869056,137438691328], that I call the sum function on to return the sum of the perfect numbers which is 146,062,119,378.

Running the Program

My application is used by loading sumperfectnums.hs and calling the sumperfect function which will output the final answer, as seen below. The heavy lifting is done in the background so it can be called simply using this one command.

```
GHCi, version 9.2.1: https://www.haskell.org/ghc/  :? for help
ghci> :set +s
ghci> :l sumperfectnums.hs
[1 of 1] Compiling Main             ( sumperfectnums.hs, interpreted )
Ok, one module loaded.
(0.06 secs,)
ghci> sumperfect
146062119378
(0.01 secs, 325,912 bytes)
ghci>
```

The running process is shown above. As well as this, we can see that the program runs efficiently and completes in 0.01 seconds.