CA320 Computability & Complexity
Assignment 1

Joao Pereira
19354106
joao.pereira2@mail.dcu.ie

Declaration
In submitting this project, I declare that the project material, which I now submit, is my own work. Any assistance received by way of borrowing from the work of others has been cited and acknowledged within the work. I make this declaration in the knowledge that a breach of the rules pertaining to project submission may carry serious consequences.

Signed: Joao Maria Baeta Pereira

Code Analysis

To start the project off I first researched what a product number was. I found Wikipedia to contain the most useful and interesting information in regards to perfect numbers and then proceeded to use the formula they provided and implemented it into my code as a basis:

```
perfectFormula :: Int -> Int
perfectFormula p = 2^(p-1) * (2^p -1)
```

Still using the information Wikipedia provided, I commented all the perfect numbers between 1 to 1,000,000,000,000 and then summed them. This information would come in useful later on when testing to see if the right figures were being outputted.

The next step was to implement a function to search for all the perfect numbers necessary. First of all, I had to implement a Divisor function that would co-operate with the perfect numbers algorithm to fetch all the perfect numbers. I attempted to code my own divisor function. however, through several failed attempts, due to the program being too slow at outputting all the numbers, I found an algorithm (source -*https://solveforum.com/forums/threads/solved-finding-perfect-numbers-in-haskell.287739/#post-287751*) which then supported me in getting the output that I required. The function itself is also higher-order. Through the use of this source, outputting a list of all the perfect numbers was not an issue.

```
sumDivisors :: Integral a => a -> a
sumDivisors a =
   foldr (\n -> let (q,r) = a `quotRem` n in
      if r==0 then (+ (n+q)) else id) 1
      [2..(floor . sqrt . fromIntegral $ a)]
```

I proceeded to create a ‘isPerfect’ function to return True or False if the ‘sumDivisors’ function was outputting a perfect number.

A ‘valueReach’ function was developed to filter through the numbers using valid. The use of valid was to ensure that when I was trying to gather all the perfect numbers into a list, no number before 1 or after 1

```
valueReach :: Int -> [Int]
valueReach n = filter valid (fmap perfectFormula [1..n])
```

trillion would be calculated. Within my 'valueReach' function I implemented a fmap operation on the 'perFormula' to modify the calculation.

```
valid :: Int -> Bool
valid x
    | x < 1000000000000 && x > 1 = True
    | otherwise = False -- else it is fa
```

Lastly, the main function was developed to test the code and to check if the program summed to the value it needed to be calculated too.

Sources

En.wikipedia.org. 2021. *Perfect number - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Perfect_number> [Accessed 27 November 2021].

SolveForum. 2021. *[Solved] Finding Perfect Numbers in Haskell*. [online] Available at: <https://solveforum.com/forums/threads/solved-finding-perfect-numbers-in-haskell.287739/#post-287751> [Accessed 27 November 2021].

Code Review Stack Exchange. 2021. *Brute-force perfect-number algorithm*. [online] Available at: <https://codereview.stackexchange.com/questions/86689/brute-force-perfect-number-algorithm> [Accessed 27 November 2021].

Stack Overflow. 2021. *Haskell - lambda expression*. [online] Available at: <https://stackoverflow.com/questions/22220439/haskell-lambda-expression/22221169> [Accessed 27 November 2021].

Hackage.haskell.org. 2021. *Data.Functor*. [online] Available at: <https://hackage.haskell.org/package/base-4.16.0.0/docs/Data-Functor.html> [Accessed 28 November 2021].