Coursework 6 (Scala)

This coursework is about Scala and is worth 10%. The first and second part is due on 16 November at 11:00, and the second part on 23 November at 11:00. You are asked to implement three programs about list processing and recursion.

Disclaimer

It should be understood that the work you submit represents your own effort. You have not copied from anyone else. An exception is the Scala code I showed during the lectures or uploaded to KEATS, which you can freely use.

Part 1 (3 Marks)

This part is about recursion. You are asked to implement a Scala program that tests examples of the $3n + 1$-*conjecture*. This conjecture can be described as follows: Start with any positive number n greater than 0. If n is even, divide it by 2 to obtain $n/2$. If n is odd, multiply it by 3 and add 1 to obtain $3n + 1$. Repeat this process and you will always end up with 1. For example if you start with 6, respectively 9, you obtain the series

$$\begin{array}{ll} 6, 3, 10, 5, 16, 8, 4, 2, 1 & (=9 \text{ steps}) \\ 9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1 & (=20 \text{ steps}) \end{array}$$

As you can see, the numbers go up and down like a roller-coaster, but curiously they seem to always terminate in 1. While it is relatively easy to test this conjecture with particular numbers, it is an interesting open problem to *prove* that the conjecture is true for all numbers (> 0). Paul Erdös, a famous mathematician you might have hard about, said about this conjecture: "Mathematics may not be ready for such problems." and also offered $500 cash prize for its solution. Jeffrey Lagarias, another mathematician, claimed that based only on known information about this problem, "this is an extraordinarily difficult problem, completely out of reach of present day mathematics." There is also a xkcd cartoon about this conjecture.

Tasks: (1) You are asked to implement a recursive function that calculates the number of steps needed number until a series ends with 1. In case of starting with 6, it takes 9 steps and in case of starting with 9, it takes 20 (see above). In order to try out this function with large numbers, you should use `Long` as argument type instead of `Int`. You can assume this function will be called with numbers between 1 and 10 million.

(2) Then write a second function that takes an upper bound as argument and calculates the steps for all numbers in the range from 1 upto this bound, and returns the maximum of steps needed by a number in that range. This function should produce for ranges

- $1 - 10$ where 9 takes 20 steps
- $1 - 100$ where 97 takes 119 steps,
- $1 - 1,000$ where 871 takes 179 steps,
- $1 - 10,000$ where $6,171$ takes 262 steps,
- $1 - 100,000$ where $77,031$ takes 351 steps,
- $1 - 1$ million where $837,799$ takes 525 steps, and
- $1 - 10$ million where $8,400,511$ takes 686 steps

Part 2 (4 Marks)

This part is about list processing—it's a variant of Buy-low-sell-high in Scala. (1) Given a list of prices for a commodity, for example

```
List(28.0, 18.0, 20.0, 26.0, 24.0)
```

you need to write a function that returns a pair for when to buy and when to sell this commodity. In the example above it should return `(1, 3)` because at index 1 the price is lowest and then at index 3 the price is highest. Note the prices are given as lists of `Floats`.

(2) Write a function that requests a comma-separated value list from a Yahoo websevice that provides historical data for stock indices. For example if you query the URL

```
http://ichart.yahoo.com/table.csv?s=GOOG
```

where `GOOG` stands for Google's stock market symbol then you will receive a comma-separated value list of the daily stock prices since Google was floated. You can also try this with other strock market symbols, for instance AAPL, MSFT, IBM, FB, YHOO, AMZN, BIDU and so on. This function should return a List of strings, where each string is one line in this comma-separated value list (representing one days data). Note that Yahoo generates its answer such that the newest data is at the front of this list, and the oldest data is at the end.

(3) As you can see, the financial data from Yahoo is organised in 7 columns, for example

```
Date,Open,High,Low,Close,Volume,Adj Close
2016-11-04,750.659973,770.359985,750.560974,762.02002,2126900,762.02002
2016-11-03,767.25,769.950012,759.030029,762.130005,1914000,762.130005
2016-11-02,778.200012,781.650024,763.450012,768.700012,1872400,768.700012
2016-11-01,782.890015,789.48999,775.539978,783.609985,2404500,783.609985
....
```

Write a function that ignores the first line (the header) and then extracts from each line the date (first column) and the Adjusted Close price (last column). The Adjusted Close price should be converted into a float. So the result of this function is a list of pairs where the first components are strings (the dates) and the second are floats (the adjusted close prices).

(4) Write a function that takes a stock market symbol as argument (you can assume it is a valid one, like GOOG, AAPL, MSFT, IBM, FB, YHOO, AMZN, BIDU). The function calculates the dates when you should have bought Google shares (lowest price) and when you should have sold them (highest price).

Test Data: In case of Google, the financial data records 3077 entries starting from 2004-08-19 until 2016-11-04 (which is the last entry on the day when I prepared the course work...on 6 November; remember stock markets are typically closed on weekends and no financial data is produced then; also I did not count the header line). The lowest for Goole was on 2004-09-03 with $49.95513 per share and the highest on 2016-10-24 with $813.109985 per share.

Part 3 (3 Marks)