CS4115 Week03 Lab Exercise

Lab Objective: We will use this first lab to set up a directory structure to use for the remainder of the semester, to look at the `gnuplot` program, which will be very handy for visually comparing functions that we encounter throughout the semester, and to compare running times of pieces of code. Here's a quick summary of the tasks:

❶ Create a series of hierarchical directories to manage our work throughout the semester

❷ Play around with the `gnuplot` program following the instructions given in this week's tute sheet

❸ Implement the functions given in Q 2.7(1)-(6) as separate programs

❹ Look at their running-time with a view to estimating their asymptotic behaviour

In Detail

❶ Note that there are a few alternatives given below for this first step so please read the entire instructions before acting. Over the semester we will have labs, programming assignments, etc. and it is a good idea to keep these in an organised way. My suggestion is that you create a subdirectory of your home directory called `cs4115` that will be the top-level point for all module-related material. The command to create a directory in Linux is `mkdir`, so you could do

```
mkdir ~/cs4115
```

which makes a subdirectory located in your home directory. Next you should make a subdirectory called `labs` in this for each week's work. This can be done with

```
mkdir ~/cs4115/labs
```

Finally, for this week's lab, Week03, you should create its own subdirectory with

```
mkdir ~/cs4115/labs/week03
```

An alternative to making each level of the hierarchy at a time is to tell mkdir to make the "parent" subdirectory if it doesn't exist. So the following command can take the place of all of the previous ones.

```
mkdir -p ~/cs4115/labs/week03
```

The `-p` is for "make parents if they don't already exist." Note that it is very similar to the command before it, but you had to do a lot of extra work in order to achieve that.

Yet another alternative is to bring up a file manager and use the "Create ..." menu option there.

❷ Use `gnuplot` to plot the functions given in this week's tute (click here for the text).

❸ Using the code provided in the class directory, `~cs4115/labs/week03`, as a template write 6 little programs, one for each of the 6 parts of Q. 2.7, and time their execution.

You can start the process by copying the supplied code into the 6 files with:

```
cp ~cs4115/labs/week03/q.cc q-2-7-1.cc
cp ~cs4115/labs/week03/q.cc q-2-7-2.cc
          ⋮
```

Now type in the code given in part (1) into `q-2-7-1.cc`. Compile and link this with

```
g++ -o q271 q-2-7-1.cc
```

and you should have an executable called `q271` that you can run as follows:

```
q271 10000
```

This will pass the value of 1000 to n in the program.

Repeat this process to create 6 executables called `q271`, ..., `q276`.

❹ The next task is to time the running of the programs and record your results. You can measure the elapsed time of a program with the command `time`. So if I want to run program `q274` on an input of size `n=10000`, at the terminal prompt I would type:

```
time q274 10000
```

Three values are returned, each a "minute and second" amount. Of the numbers returned it is the "real" one you are after.

Run the program giving approx. 6 different values of `n`; make sure to make the gap between values of `n` to be wide. I would suggest something like 10, 100, 1000, 10000, etc. although for some of the programs using mega-values may take long running time. Cut your cloth accordingly. In order to smooth out any inconsistencies between runs I would suggest running each program 5 times at each value of `n` you consider.

You should save your results in a spreadsheet. You can fire up the OpenOffice spreadsheet either by clicking on "Applications" and then "OpenOffice calc" in the top-left corner, or just typing `oocalc` from the prompt. Set up your spreadsheet so that there is a "page" (worksheet) for each program and so that you compute the average run-time for each value of `n` that you considered in each worksheet.

The next thing we will consider – next time – is how to derive a big-oh expression for the average run-times we recorded and compare them to the run-times predicted by our pencil-and-paper analysis.