CS4115 Week04 Lab Exercise

Lab Objective: We now have three programs under our belts. From Week02 we have `lin` and `quad` and from Week03 `matmult`. The claimed running times of these programs are $\mathcal{O}(n), \mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ (for a *square* matrix), respectively. By recording the running times over a range of values for n and observing the increasing **trends** we will see how theory matches reality.

Here's a quick summary of the tasks:

❶ Gather times for the running of the three programs

❷ Insert these into a spreadsheet

❸ Analyse the results and conclude their asymptotic running times

In Detail

As always, you should create a new lab (sub)directory for this week's work. It should be named `~/cs4115/labs/week04`. Since you will be creating files in this directory you should change your working directory with the command

```
cd ~/cs4115/labs/week04
```

Recall that the `lin` program requires a file called `blah.txt` in `/tmp`. You can do this by copying your favourite file to `/tmp` with the command

```
cp my-fave-file /tmp/blah.txt
```

❶ We described the `time` command in Week02. You should use this now to generate a set of timings for each of the three programs. As we have mentioned in class there is little point in taking readings for small values of `n` as there are too many other factors which influence the starting of a program, and for small values of `n` these have a bigger relative impact. The command to run, say, the `lin` program would be

```
../week02/lin 11
```

and the command to time this would be

```
time ../week02/lin 11
```

For small values a program can simply run too fast for the clock! Try running either of the two first programs (`lin` or `quad`) with two small but different values of `n` – maybe `n=10` and `n=20` – and then repeat this experiment a number of times. Note how the clock reports inconsistent times. This is because the program simply runs too fast for the clock to catch it.

Beat the clock

Giving a "value of n" is easy for the two programs from Week02. But what about making `matmult` operate on a "value of n"? What we mean by n in this case is the number of rows in the array. A square matrix will then have exactly n^2 entries. So how do we generate a decent-sized matrix of elements? Well, if you like, for $n = 200$, you can spend the time entering the necessary $200 \times 200 = 40\,000$ elements by hand and then another 40 000 for a second matrix so that we can time the multiplication of them. *Or,...*

In this week's lab directory, `~cs4115/labs/week04` there will be a perl script, `gen-matrix.pl` that will generate a random square matrix of a given size. You can control the size of the random numbers – although you shouldn't need to – and the size of the matrix, which you definitely will need to. **You will use this script to help generate input to your matrix multiplication program.**

You can generate a square matrix of random `float`s of size 200×200 with the command

```
perl gen-matrix.pl -n 200
```

But we need to save it in a file for use in the multiplication step so we should run

```
perl gen-matrix.pl -n 200 > mat.200
```

in order to save it to the file `mat.200`. If you run this command a second time it overwrites the file `mat.200` with the new random matrix.

But `matmult` expects *two* matrices for multiplication, one after the other and the perl script only generates *one*. Here's a handy UNIX shell trick to **append** to a file

```
perl gen-matrix.pl -n 200 >> mat.200
```

Note the `>>`, which is not related to the C++ operator of the same name. In this case, rather than overwriting anything that was in it previously, the file gets **added to**. In this way we can generate the necessary input for `matmult`.

gen-times.pl

Picking a starting value of `n` to begin a timing analysis is one issue with timing a program. On my machine I found that $n = 100$ gave useful values for the two small programs. A second issue is what increments of `n` to use in successive runs. For example, if your last run was `n=1000` would it make sense to run it next for `n=1010` or `n=1100`, even? Actually, no. With step sizes so small it would take forever to get into the really "interesting" values. Better, instead, to make the next choice of `n` be a *multiple* of the last one. So an acceptable sequence of test values might be `n=100,200,400,800,...`.

A `matmult` data point

To generate a `matmult` data point at $n = 400$

```
perl gen-matrix.pl -n 400 > mat.400
perl gen-matrix.pl -n 400 >> mat.400
time matmult 400 400 400 400 < mat.400 > out.400
```

Finding a sequence of values to run your program on is a trade-off between gathering lots of data points and the time it takes to generate those data points. You decide, but always keep in mind that the `n=100,000` data point is more valuable than the `n=1000` data point.

❷ As you generate your data you should write them into a spreadsheet. Keep the format simple: four columns labelled `n`, `lin`, `quad` and `mat` and then write in the value of `n` you ran them at and the running times for each. As you will soon observe it will be much easier to get large runs for one program than the others so, in effect, it will be much easier to fill out some columns than others. However, there is nothing to say that you have to use the *same* values of `n` in all cases: you could decide to run one program using a sequence of `n` that goes up in multiples of 10, while the others go up in smaller multiples. Just make sure that you leave a cell blank if you didn't record values for that program at that value of `n`.

All that said, it is always useful to have several *reference values* where the programs are run on the same value.

When you have generated all of your values save the spreadsheet as a `.csv` file.

LibreOffice spreadsheet

You should be able to find a spreadsheet program under **Applications**. The `LibreOffice` suite of programs will certainly have a spreadsheet and on my machine it is located in `/usr/bin/libreoffice`. This is most likely already on your `PATH` so you should be able to run it from the command line with `libreoffice`, or in some cases, `loffice`.

❸ Now that you have sampled the programs' running performance at a large no. of data points you should now look at your numbers to see what is happening. Did the faster program for small values remain so for larger values? Did the gap between the programs at the reference values widen or shrink? Can you quantify the asypmtotic behaviour of the programs in "Big-Oh" notation?