1. Use your favourite graph plotting program to verify that what is really important in asymptotic analysis of functions isn't the additive constant ($T(n) = g_1(n)+\mathbf{c_1}$) or the multiplicative constant ($T(n) = \mathbf{c_2}g_2(n)$) that appears in the function $T(n) = \cdots$ but rather the powers of n that appear in the function itself.

   Do this by comparing the behaviours of the pairs of functions.

   (a) $f(n) = n + 500$ and $g(n) = n^2$

   (b) $f(n) = 0.0001n^2$ and $g(n) = 150n$

   At first it looks like the additive constant in part 1 and the multiplicative constant in part 2 is having a major impact but as you consider larger and larger values of n, you will see that it is the power of n that holds most sway. The only difference a different constant makes is the point of transition.

   The following instructions will help you if you decide to use the *gnuplot* program that runs on Linux. By default gnuplot treats functions in terms of x, so we will use x when we mean n and we will use $x**2$ when we mean n^2. The numbers in brackets tell gnuplot what ranges of x to plot between.

```
gnuplot
plot [0:10] x+500, x**2
plot [0:100] x+500, x**2    // note different behaviour

plot [-5:10000] .0001*x**2, 25*x
plot [-5:1000000] .0001*x**2, 25*x
```

2. Use `gnuplot` to compare the behaviours of the functions $f(n) = n$, $g(n) = n \cdot \log n$ and $h(n) = n^2$. As above, when we think n `gnuplot` thinks x so the command that you will need to give is:

```
plot x, x*log(x), x**2
```

   Play around with different ranges of x to get a feel for the "bigger picture." Note that `gnuplot`'s idea of $\log(x)$ is to the base 10, whereas we would like base 2. Can you think of how to cope with this little annoyance?

3. Prove that

   (a) $\mathcal{T}(n) = T_1(n) + T_2(n) = max(O(f(n)) + O(g(n)))$, and

   (b) $\mathcal{T}(n) = T_1(n) * T_2(n) = O(f(n) * g(n))$

Hint: Write down the definition of $T_1(n) = O(f(n))$, using a constant of c_1 and $n > n_1$ and then do the same for T_2. Then come up new values of $c_{\mathcal{T}}$ and $n_{\mathcal{T}}$ that will match the definition of $\mathcal{T}(n) = O(\ldots)$.

4. Qs 2.1, 2.2, 2.4, 2.7 (note esp. (5) & (6))

5. In the lab we will implement the little chunks of code from Q 2.7 as separate programs and, by timing them, we will see if the measured running time matches the analytical running time