# CPSC 221 2014W1, PP#2
# Modifying a Modular Timing Module

Alan Hu and the CPSC 221 Course Staff

October 6, 2014

This is a hybrid of a programming and theoretical assignment. Please submit it to the handin target `pp2`. Edit the `README.txt` file for this project linked from the course webpage to insert your answers. You'll also need to submit one file `plot.pdf` that plots the runtime of a function versus `x`. (See the last question below.)

To complete the project, download the file `modexp.cc` from the course webpage and answer the questions below about it.

`modexp.cc` is a small C++ program that contains three different implementations to raise a number $b$ to the $x^{\text{th}}$ power, modulo another number $m$.

In real life, this is an important computation used for encryption and decryption. (Google "RSA encryption" if you're curious.) To keep the code simple, I am using only 64-bit integers (32-bits if you run on an older machine), but in real applications, the numbers are thousands of bits long, so the efficiency matters a lot.

1. Find a good big-$O$ upper bound on the runtime for the function `mod_exp1`. Write your result in terms of the exponent $x$ (treating $b$ and $m$ as constants). You need not prove your result, but you should show your work if you'd like to receive partial credit, in case you make any mistakes.

2. Find a good big-$O$ upper bound on the runtime for the function `mod_exp2`. Write your result in terms of the exponent $x$. You need not prove your result, but show your work, as above.

3. People sometimes think they can optimize a program by using fewer lines of code. The function `mod_exp2o` is an example of such a misguided attempt. Find a good big-$O$ upper bound on the runtime for this function. As above: Write your result in terms of the exponent $x$. You do not need to prove your result, etc.

4. Suppose you keep $x$ as a constant and instead compute a big-$O$ upper bound on the runtime as a function of the modulus $m$. What is your big-$O$ bound for `mod_exp1`? (As above: you need not prove your result, and you may not find that you have much work. **Briefly** justify your answer instead!)

5. Now we're going to look at how the scaling behaviour you've predicted relates to the practical behaviour of the code.

   The code you downloaded is set to use `mod_exp1`. Make sure that is still the case. Compile the program using the default settings (in particular, do not enable any fancy optimization). Then, run the program for various values of $b$, $x$, and $m$, until you find values for which the program takes about 10–20 seconds to run. For example, on a high-end desktop from 2011 (Intel Core i7), $b = 2$, $x = 20$, and $m = 1000$ takes about 12 seconds (leaving the delay loop in the code unmodified). Write down the values you found, how long it took to run, and what kind of computer you are using.

6. Based on your asymptotic analysis of `mod_exp1`, roughly what value of $x$ would make the program take twice as long?[1] Run the program with the value of $x$ you estimate. What was the run time? Was your estimate a good one?

7. Now, comment out the call to `mod_exp1`, uncomment the call to `mod_exp2`, and recompile the program. Again, find values of $b$, $x$, and $m$ that will make the program take around 10–20 seconds to run. For example, on the computer mentioned above, $b = 2$, $x = 400$, and $m = 1000$ takes about 12 seconds. Write down the values you found, and how long it took to run.

[1]For example, if your analysis said the runtime was $O(x^2)$, then multiplying $x$ by $\sqrt{2}$ should make the program run twice as long.

8. Based on your asymptotic analysis of `mod_exp2`, roughly what value of $x$ would make the program take twice as long? Run the program with the value of $x$ you estimate. What was the run time? Was your estimate a good one?

9. Now, comment out the call to `mod_exp2`, uncomment the call to `mod_exp2o`, and recompile the program. Again, find values of $b$, $x$, and $m$ that will make the program take around 10–20 seconds to run. Write down the values you found, and how long it took to run.

10. Based on your asymptotic analysis of `mod_exp2o`, what value of $x$ will make the program take twice as long? Run the program with the value of $x$ you estimate. What was the run time? Was your estimate a good one?

11. Create a plot of runtime versus $x$ for your choice of `mod_exp1`, `mod_exp2`, or `mod_exp2o`. You should have enough points plotted to either make the curve's expected shape clear or illustrate that it does not have the expected shape (say, at least 10). Be sure to tell us which function you plotted and label the axes so we can tell what values you found!

    Note 1: We think `mod_exp2` is probably the most interesting one to plot, but you're free to pick one of the others.

    Note 2: Use any tool you like to do your plotting, but you *must* submit the plot as a PDF file named `plot.pdf`. Tools that might work for you include: matlab, Excel, LibreOffice, C++, python, gnuplot, ploticus, etc. (gnuplot may be the easiest way to go if you want to do this from our servers.)

**IMPORTANT:** On a Unix, Linux, or MacOS machine (or a Windows machine running Cygwin), you can time a command by typing the word `time` before the command you want to time. You can run `man time` on the command prompt (or search for help on the Unix command "time" on the web) for help on this command. Use the "user time" returned by the command, not the wall clock time (which includes the time it takes for you to type in your input, time spent doing things besides your program, etc.).