# EECS 203: Discrete Mathematics
# Group Homework 4 (Spring 2016)

Answer Key

Due: June 2, 2016

1. (20 points) Posted on-line with this assignment is a bit of C++ code which computes the gcd of two numbers by finding the prime factorization of those two numbers. In this assignment you will be timing that code to see how its runtime varies with the size of the two numbers that you are computing the gcd of. But first, youll answer a few questions about the code.

   a) Describe what the primeFactors function does. In particular, be sure to explain the two arguments and the return value. [1]

   b) There is an if/else structure in the gcd function. What is it doing? Your answers should make it clear why we are sometimes incrementing i, sometimes j and sometimes both. [1]

   c) What does the randomX function do? [1]

   d) Explain what is going on with tick and toc and the clock function. Reading http://stackoverflow.com/quest time-of-c-program might be useful. [1]

   e) Compile the C++ code. It should work just about anywhere, but does need a 64-bit machine and a 64-bit compiler (any modern Linux machine should work with g++ but you should be able to use whatever compiler/system youd like). Graph the runtime of the average case gcd call vs the number of decimal digits in the numbers passed to gcd. In order to get useful number and still finish in a reasonable amount of time, you probably will have to vary the number of iterations of the loop. Take at least 10 data points. [3]

   f) By looking at your graph from part e, what would you say the relationship is between the number of digits in the input and the average run time of this program? That is, what order is this algorithm in terms of the number of digits (big Theta). Justify your answer. [3]

   g) Change the gcd algorithm used here to use the Euclidian Algorithm. Note: this should be really (really) short. Feel free to look for code on line, just be sure you understand it and cite it if you use it. Redo parts e and f with this new algorithm. Make sure you keep your runtimes reasonable (at least 5 seconds) by varying the number of times gcd gets called. If you find the code for this hard to write, just ask for help. [6]

   h) Discuss how the two different algorithms differ in terms of runtime. You should be using the language of the growth of functions. [4]

**Solution:**

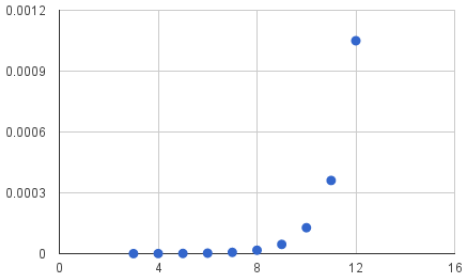
a) The first argument is the number to be factored into primes. Because we cannot simply return a list in c++, we pass a list to be populated as the second argument. primeFactors finds all of the prime factors (with repetition) of n and puts them in list in increasing order. Finally it returns the number of elements in the list because otherwise there would be no way for the calling function to know how many it has.

b) Because these lists are sorted, if the lists share a minimum prime factor, they will both be at the beginning, while if one is less than the other, we can move along in just that list until it catches up. Whenever we find a match, we multiply it into a running product and move on to the next term in both lists so we don't count the same copy of that factor multiple times.

c) randomX generates two 32-bit random numbers and concatenates them. It then takes this mod the number of possible return values there are and shifts the resulting range to the right location.

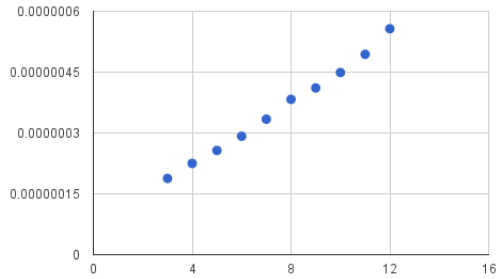
d) Each call to clock() returns the current clock cycle of the processor. By noting the cycle number when we start and when we end, we can subtract the two to get how many cycles it took to run our function calls.

e)




f) This is an exponential curve, which can be checked by comparing the ratio of consecutive digit numbers. Whenever we add a digit, it multiplies the average runtime by a number between 2 and 3.

g)
```
long gcd2(long a, long b) {
    if(a*b==0) return a+b;
    return gcd2(b,a%b);
}
```




This is clearly a simple linear order. We can confirm by noting that we add a value between 3 and 5 $\times 10^{-}8$ every time we add a digit.

h) We know that $n \in O(2^n)$, so $eulidean \in O(factor_gcd)$

2. (5 points) Problem 26 on page 305. Finding the inverse might be the hard part here. Consider the Extended Euclidian Algorithm. Lots of sources on line. Show how you got the inverse.

**Solution:** We first find the decryption exponent $d$; to do so, we find the inverse of 17 modulo $52 \cdot 60 = 3120$ using the extended Euclidean algorithm. First, we have that

$$\begin{aligned} 3120 &= 183 \cdot 17 + 9 \\ 17 &= 1 \cdot 9 + 8 \\ 9 &= 1 \cdot 8 + 1 \\ 8 &= 8 \cdot 1. \end{aligned}$$

This shows us that $\gcd(17, 3120) = 1$. Now we find the inverse:

$$\begin{aligned} 1 &= 9 - 1 \cdot 8 \\ 1 &= 9 - 1(17 - 9) \\ 1 &= 2 \cdot 9 - 1 \cdot 17 \\ 1 &= 2(3120 - 183 \cdot 17) - 1 \cdot 17 \\ 1 &= 2 \cdot 3120 - 367 \cdot 17. \end{aligned}$$

This tells us that $-367$ is the inverse of 17 modulo 3120. But note that modulo 3120, $-367 \equiv 2753$. Therefore, we have that $d = 2753$.

Now, to decrypt a block $C$, we have to calculate

$$C^{2753} \bmod 3233.$$

We go block by block. We have that

$$3185^{2753} \bmod 3233 = 1816$$

$$2038^{2753} \bmod 3233 = 2008$$

$$2460^{2753} \bmod 3233 = 1717$$

$$2550^{2753} \bmod 3233 = 0411$$

From this, we get that our decrypted message, corresponding to 18 16 20 08 17 17 04 11 is $SQUIRREL$.

3. (5 points) Use induction to prove the following (you must use induction, any other proof technique will get zero points).

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$

**Solution:**
**Theorem:** For all $n \in \mathbb{Z}^+$,

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}.$$

**Proof:** Let $P(n)$ be the proposition that

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}.$$

We use induction to prove $P(n)$ for all $n \in \mathbb{Z}^+$.

**Base Case:** When $n = 1$, we have that

$$\sum_{i=1}^{1} \frac{1}{i(i+1)} = \frac{1}{1(1+1)} = \frac{1}{2}$$

and

$$\frac{1}{1+1} = \frac{1}{2}$$

and therefore $P(1)$ is true.

**Induction Step:** Assume that $P(k)$ is true for arbitrary $k \in \mathbb{Z}^+$. We will show that

$P(k+1)$ is true. By assumption,

$$\sum_{i=1}^{k} \frac{1}{i(i+1)} = \frac{k}{k+1}$$

$$\Rightarrow \quad \sum_{i=1}^{k} \frac{1}{i(i+1)} + \frac{1}{(k+1)((k+1)+1)} = \frac{k}{k+1} + \frac{1}{(k+1)((k+1)+1)}$$

$$\Rightarrow \quad \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \frac{k((k+1)+1)}{(k+1)((k+1)+1)} + \frac{1}{(k+1)((k+1)+1)}$$

$$\Rightarrow \quad \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \frac{k((k+1)+1)+1}{(k+1)((k+1)+1)}$$

$$\Rightarrow \quad \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \frac{k^2+2k+1}{(k+1)(k+2)}$$

$$\Rightarrow \quad \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \frac{(k+1)^2}{(k+1)(k+2)}$$

$$\Rightarrow \quad \sum_{i=1}^{k+1} \frac{1}{i(i+1)} = \frac{k+1}{k+2}$$

which is exactly $P(k+1)$!