Programming Assignment #1

GCD (Greatest Common Divisor) with Big Integers

---

1. Introduction

Cryptography is a key element in preserving confidentiality; it is used to scramble data such that it cannot be easily read by anyone except the intended receiver.

There are many protocols that use cryptography, SSH and SSL are just the two best known. PGP is a popular application that uses publick-key cryptography.

These protocols and the algorithms they use are largely based on the power of having large keys that are used to perform forward calcutions in a reasonable amount of time, but difficult to calulate the inverse (read very time consuming).

One way of doing this is by using prime numbers. It is very easy to multiply two very large numbers, say 1000 digits each, and get the 2000 digit result; multiplying two n-digit numbers requires $\mathbf{O}(n^2)$ operations. But it is extremely difficult to factor a large integer.

For very large number there is no known efficient integer factorization algorithm. In an effort that started in 2003 and was concluded in 2005, factoring a 200 digit number (RSA-200) took eighteen months and is estimated to have taken the equivalent of 55 years on a single 2.2 GHz Opteron CPU.

Not all numbers of a given length are equally hard to factor. The hardest instances of these problems are those where the factors are two randomly-chosen prime numbers of about the same size.

The supposed difficulty of factoring large numbers is at the heart of RSA.

In this first programming assignment, you will implement one of the key building blocks required to implement the RSA algorithm. GCD will come in handy when you try to determine if two numbers are relatively prime.

2. Details

Since you will have to compute large exponents as part of the RSA algorithm, the standard integer types (int - 16 or 32 bits, long int - 32 bits and and even long long int - 64 bits) of most computer languages will not suffice.

Here are your choices:

- **Java** has a built-in bigint class that behaves exactly like an int, except that it is not bound by a finite value
- **C++** you'll have to use a library such as NTL (Library for doing Number Theory) or GMP (the GNU Multiple Precision Arithmetic Library)
- In **Ruby** you don't have to worry about the size of numbers at all, Ruby automatically takes care of any conversion between numbers of fixed size (Fixnum - 16, 32, 64 bits) and Bignum (numbers with an arbitrary number of digits)

3. What You Have to Do

The assignment is broken down into four main parts. Each part will be graded separately, but each part will also be reused for other parts of the assignment. This will allow you to test each part conclusively before moving on.

3.1 GCD Algorithm (25%)

Develop your own algorithm or find an existing one for finding the GCD of two arbitrarily large integers.

Please write your algorithm in the space provided below:

Student name:
Type your content in this table row.

3.2 GCD Implementation (60%)

Now, implement your algorithm using your choice of language. Make sure that your program allows the use of arbitrarily large integers.

You may NOT use the gcd() library function, regardless of the language you're using for your implementation and the particular name of the function in the library. Instead, you have to develop your own function(s) for finding the GCD of two arbitrarily large integers.

Your program will be named **gcd** and runs from the command line. It takes two arguments, two integer numbers of arbitrary size, and prints out their GCD.

Example (the $ sign is the command line prompt):

```
$ gcd 34523452345234523542345 254345534435
$ 5
```

In the table below you'll find some test cases for testing your GCD function. Fill in the empty locations using your program. For those cases where you have to select your own numbers, make sure you select numbers that are at least 20 digits long.

First number	Second number	GCD
2345	72	1
1406700	164115	23445
1368	339	3
55534	434334	2
243532	0	243532
30315475	24440870	31415
37279462087332	366983722766	564958
3823485234523624	43882834845621	
233961810342958422635	3927915394316175446	
34823482334858234892934	438923489238932492	

Student name:		

3.3 Required Document (5%)

In addition to your source code and your output you are required to submit a memo to your instructor that is flawlessly written. No spelling errors. No grammatical errors, etc. If the document is deemed unprofessional, e.g. a significant number of grammatical or spelling errors, then it will be assigned a grade of zero.

If you're not sure about the format for a memo, then just search for "memo format" on Google.

Your memo should state clearly the status of the assignment. Is it done or not? Does it meet all of the requirements? If anything is missing then state clearly what is missing from your work.

Failure to give a status for the assignment will result in a grade of zero for the entire assignment. Providing a status that's false or significantly misleading will also result in a zero for the entire assignment.

Make sure your document is well-written, succinct, and easy to read. If you encountered problems with the assignment, then provide a detailed description of those problems and the solution(s) you found.

3.4 Assignment Submission (10%)

You are required to submit your work online, using the Blackboard. Late work will be accepted, however it is subject to late penalties as described in the syllabus.

Here are the requirements for your submission:

- Submit your electronic copy using the Blackboard (attach the assignment as a compressed archive file (.zip, .tgz, .tbz2, .rar)
- The name of the compressed archive should be: *fistName-lastName*-PA-*assignmentNumber*.*extension* (e.g. Jane-Doe-PA-1.zip)
- Include (i) your memo, (ii) a completed handout with your results and your name on it, (iii) the source code, (iv) a README file that explains how to build and on how to run each program, (v) compiled executable(s) and (vi) a shell script or batch file that will compile your programs when executed; a Makefile (see GNU Make for details) would be great, however any form of script or building tool will do
- Include your e-mail address in the Comment field when submitting the assignment through the Digital Drop Box
- If for any reason you are submitting the assignment more than once, indicate this in the Comment field by including the word COMPLEMENT