Investigation of factorisaton times for products of two primes

M J Brockway

October 1, 2017

A rough graph is obtained of times to factorise a product of two primes, as a function of the size in bits of the primes. The *logarithm* of the time appears ro depend lnearly on the number of bits, suggesting (as expected, in fact) that the time is exponential in the bit-size.

1 Set-up

1.1 `PrimeFact` class

This class contains public static method `primeFact(long b)` which times, using `System.nanotime()`, the factorisation of a java long (64-bit) prime number, and prints on stdout the factorisation and the time it took.

This class also has a main method and can be run on a command-line:

```
$ java PrimeFact n r
```

which will cause the prime factorisations of $n, n+1, n+2, ..., n+r$ to be printed out. For example,

```
$ java PrimeFact 99d7 5
Prime factorisation:
99d7 = 99d7^1. - 13144 us
99d8 = 2^3.3^2.223^1. - 1511 us
99d9 = 5^1.1ec5^1. - 2937 us
99da = 2^1.2f^1.1a3^1. - 1784 us
99db = 3^1.13^1.2b3^1. - 1439 us
```

Note the numbers are all in hexadecimal. $99d7_{hex} = 39383$; note that it is prime and the computation took 13144 microseconds. $99db_{hex} = 39387$ and its factorisation is $3^1 \times 13^1 \times 2b3^1$ in hex (ie $3^1 \times 19^1 \times 691^1$ in decimal).

1.2 PairsOfPrimes16-32.txt

is a file of pairs of primes, one pair per line, with sizes in bits ranging from 16 bits up to 32 bits. This procedural app discussed in the previous paragraph helped me build this file. This file provides the input data for the main part of the investigation.

1.3 FactProdLongs class

the the procdural program that drives the investigation.

```
$ java FactProdLongs PairsOfPrimes16-32.txt
```

This reads each pair of primes, mutltiplies them, and then (using `PrimeFact.primeFact()`, see 1.1) times the factorisation, outputting the results:

```
$ java FactProdLongs PairsOfPrimes16-32.txt
... ... ...
b6a5(16 bits) fa21(16 bits)  = b274ad45; b274ad45 = b6a5^1.fa21^1. - 2733 us
8047(16 bits) 883d(16 bits)  = 444448eb; 444448eb = 8047^1.883d^1. - 1715 us
e6ad(16 bits) eb07(16 bits)  = d3c71dbb; d3c71dbb = e6ad^1.eb07^1. - 2319 us
12af9(17 bits) 1c079(17 bits)  = 20bc10fb1; 20bc10fb1 = 12af9^1.1c079^1. - 6855 us
114c1(17 bits) 11e6f(17 bits)  = 135a79daf; 135a79daf = 114c1^1.11e6f^1. - 2971 us
14057(17 bits) 1c447(17 bits)  = 235f27421; 235f27421 = 14057^1.1c447^1. - 4473 us
... ... ...
```

(This run prodices 335 lines of results altogether.)

1.4 FactProdLongs_Log.txt

The program of section 1.3 was run and the results collected in `FactProdLongs_Log.txt`. Two ways of doing this are

```
$ java FactProdLongs PairsOfPrimes16-32.txt > FactProdLongs_Log.txt
$ java FactProdLongs PairsOfPrimes16-32.txt | tee FactProdLongs_Log.txt
```

The second method displays the output on the cosole and *also* saves it in `FactProdLongs_Log.txt`. The first method redirects the output to the log file but displays nothing on the console. It is also possible to *append* output to a file using the redirection symbol `>>` rather than `>`.

Have a glance through `FactProdLongs_Log.txt` - this is the raw data produced by the experiment.

2 Analysing the results

The first task is to collect the timings of all the 16-bit factorisations and compute their mean (average), then do the same for the 17-bit factorisations, the 18-bit factorisations, and so on up to 32 bit.

It can save tedious repeated copying and pasting if you have some practical skill with text editors with advanced editing capability, and/or spreadsheet programs. For instance, you can open `FactProdLongs_Log.txt` in Excel or Libre-Offic Calc and then delete columns so as to end up with just the timing data in a single column, and nothing else.

It is then straightforward to compute the mean timings for the 16-bit data, the 17-bit, etcetera.

`FactProdLongs_Anal.ods` is the spreadsheet I used for analysis. Take a look at rows 2 - 26, columns B to R. these are the timings from the log file: the 16-bit tiimings are in column B, the 17-bit timings are in column C, and so forth, up to 32-bit in column R. The columns B-R are headed in row 1 by the relevant number of bits.

Look at the formulae in cells B28-R28. These are the mean or average timings for the 16-bit data, the 17-bit data, ..., the 32-bit data. I am interesting in how these average timings depend on the number of bits shown in B1-R1.

I was expecting an *exponential* relationship, such as time $= O(2^n)$. Mathematically this corresponds to log(time) $= O(n)$. So I added in row 29 the logarithms of the numbers in row 28. Check the formulae in cells B29-R29.

Rough confirmation of my hypothesis is provided by the graph, which shows a straight-line relationship between the logarithmic values in row 29 with n, the number of bits in the input data, in row 1.

This is a crude analysis, admittedly - there are clever statistical techniques for providing a more rigorous analysis than this.