

# Re: JSH: SF Algorithm

---

*Source:* <http://sci.tech-archive.net/Archive/sci.math/2007-09/msg02158.html>

---

- *From:* rossum <[rossum48@xxxxxxxxxxxxx](mailto:rossum48@xxxxxxxxxxxxx)>
  - *Date:* Tue, 11 Sep 2007 16:07:57 +0100
- 

On Tue, 11 Sep 2007 03:02:25 -0000, JSH <[jstevh@xxxxxxxxxx](mailto:jstevh@xxxxxxxxxx)> wrote:

Oh well, enough bravado on my part as I'm not certain this will work and waiting for Wednesday just seems silly. The Bulletin of the AMS did reject. Why would they change their minds between now and then?

The expert opinion is noted. Here is what my research says, which presumably then will not work, but I do not know why it would not.

Given a target composite T, from theory using  $x^2 = y^2 \pmod T$  and  $k = 2x \pmod T$ , it can be proven that

$$(x+k)^2 = y^2 + 2k^2 \pmod T$$

must be true for any solution of a difference of squares.

Explicitly to solve you need solutions for

$$(x+k)^2 = y^2 + 2k^2 + nT.$$

The algorithm picks x directly, choosing  $x = \text{floor}(\text{sqrt}(T))$ , so  $k = 2x$ , and then ranges for the n's from

$$n_{\text{max}} = \text{floor}(((x+k)^2 - 2k^2)/T)$$

and

$$n_{\text{min}} = \text{floor}((4(x+k-1) - 2k^2)/T)$$

which with my program has meant roughly 32 surrogates to factor.

By the theory, if you can fully factor all 32 surrogates for any target T, then you will non-trivially factor T.

If you cannot factor all 32 with the given x, you can increment it by 1 and try again, indefinitely.

Note that you can also use  $x = \text{floor}(\text{sqrt}(2T))$  to have about 64

## Re: JSH: SF Algorithm

surrogates and much greater odds but I'm not clear how that works exactly and besides if you can factor 32 with the first one then you have the target in hand.

It is so weirdly simple and I think the theory is correct, but I guess I could be wrong.

I have tried to implement with my own programs but as I pointed out in a previous post, I use recursion and with big numbers fewer and fewer of the surrogates get factored, so it craps out.

I am not confident that I can work that problem out so what I said earlier was bravado on my part.

James Harris

As I have done previously, I tested James' latest version of his surrogate method on 500 random composite odd numbers that are multiples of two different primes, each in the range 500 to 1000. The results are compared to Fermat's method, trial factorisation (both forward and reverse) and random picking.

Fermat average = 7.01 probes.  
JSH average = 892.29 probes.  
Probe ratio = 1 : 127.361  
Trial average = 120.62 probes.  
Reverse average = 11.72 probes.  
Random average = 754.83 probes.  
500 trials, 0 misfactors found.

Average k's tried per factorisation: 4.276  
Average n's tried per factorisation: 26.344  
Average n's tried per k: 7.736

As seems usual with James' methods it will factorise the target number, but more slowly than existing methods.

As a side point, I made two errors in the initial version of my code: I swapped the expressions for nMin and nMax, and I miscoded the expression for nMax (which I was assigning to nMin). In effect I had:

```
nMin = ((x+k) - (2*k*k))/T; // (x+k) should be (x+k)*(x+k)
nMax = (4*(x+k+1) - (2*k*k))/T;
```

With these two errors in place my results were:

Fermat average = 7.09 probes.  
JSH average = 300.20 probes.

Re: JSH: SF Algorithm

Re: JSH: SF Algorithm

Probe ratio = 1 : 42.318  
Trial average = 120.78 probes.  
Reverse average = 11.63 probes.  
Random average = 710.27 probes.  
500 trials, 0 misfactors found.

Average k's tried per factorisation: 7.742  
Average n's tried per factorisation: 8.110  
Average n's tried per k: 1.055

This version runs faster than both random picking and James' current version. It uses more k's than James' version, but within each k it needs fewer n's. On balance this version needs fewer probes to find a factor.

It might be interesting to analyse why my mistakes had such a beneficial effect – they might point out a way to improve the method further.

rosum

.