

Re: Integer test for perfect square

Source: <http://sci.tech-archive.net/Archive/sci.math/2009-03/msg01845.html>

- *From:* bert <bert.hutchings@xxxxxxxxxxxxxxxx>
 - *Date:* Fri, 13 Mar 2009 04:58:20 -0700 (PDT)
-

On 12 Mar, 17:41, riderofgiraffes <mathforum.org...@xxxxxxxxxxxxxxxx> wrote:

Is there an integer test for a perfect square?

Henri Cohen's ... algorithm 1.7.3. ... check if the number is a square modulo some small numbers (11, 63, 64, 65 is suggested), then square the integer square root.

I guess Cohen outlines a way to find the integer square root without using floating point? Some form of Newton's method?

Unlikely. There is an algorithm, resembling long division, which rapidly finds the integer floor R of the square root of any integer N , and the remainder S , i.e. $N = R^2 + S$ where $0 \leq S \leq 2R$. You can find a description of it at:

<http://www.mathpath.org/Algor/squareroot/algor.square.root.why.htm>

and on various other web sites.

The algorithm was taught in schools up until perhaps 1960, so only quite elderly people are familiar enough with it to just do it on automatically on paper. But it could be expected to be the method of choice in a multiple-precision integer package.

Re: Integer test for perfect square

I have implemented it several times, in different languages. Newton's method would be much slower.

Are you sure? Have you tested that? Newton's method doubles the number of digits on each iteration, but the "long-division" method that I know so well only gives one extra digit per phase. My speed tests show integer variants of Newton's Method to be far faster for large numbers.

In short, my results disagree with your assertion, and I would be interested to see how your evidence compares with my tests.

Well, that's probably apples and oranges. If you already have a long division routine in a (very fast) black box, then for sure, Newton's method will win. But the cases that I implemented were where I had already had to build up the division routine with shifts, compares, and subtracts, so it was generating the quotient bits (or digits, or superdigits) one at a time; and my square root routine was also generating its result bits (or digits, or superdigits) one at a time, with roughly the same type and amount of underlying work for each, so it was finished in about the time of ONE division operation, whereas Newton's method would have taken several of them.

--

.