Roots in Endymion

Jos Grimm, Apics Team

June 10, 2005

We want to find the n-th root of an integer number x. We assume x positive, at least 2, and we want to find the integer part of $b = x^{1/n}$, the positive root. Consider

$$g_x(a) = a - \frac{a - (x/a^{n-1})}{n} = \frac{x + (n-1)a^n}{na^{n-1}} \ , \tag{1}$$

$$f_x(a) = a + \left\lfloor \frac{\lfloor x/a^{n-1} \rfloor - a}{n} \right\rfloor \ , \tag{2}$$

$$h_x(a) = a - \left\lfloor \frac{a - \lfloor x/a^{n-1} \rfloor}{n} \right\rfloor \ . \tag{3}$$

In most of the cases, we shall omit the index x. We shall define three algorithms, noted `isqrt`, `iroot` and `jroot`. In this document, all timings on dor on a 3.06Ghz Intel processor

1 The Newton method

One way to solve $F(a) = 0$ is to consider $a_{k+1} = a_k - F(a_k)/F'(a_k)$. If this converges to a fixed point a, we have $F(a) = 0$. If we apply this to $F(a) = a^n - x$, we get $a_{k+1} = g(a_k)$. We have

$$g(a) = \frac{a}{n} \left[n - 1 + \left(\frac{b}{a} \right)^n \right]. \tag{4}$$

This shows that $g(a) > 0$ if $a > 0$. Thus, we may assume $a_k > 0$ for each k. From (1), we have obviously $g(a) \geq a$ if and only if $a \leq b$, so that g has a unique fixed point. Let $c = b/a$. We have $g(a) \geq b$ if and only if $n - 1 + c^n \geq nc$. If $w(c) = n - 1 + c^n - nc$, it is obvious that w has a minimum at $c = 1$, and $w(1) = 0$. As a consequence the sequence a_k decreases for $k > 0$, and converges to b.

Assume that $a = b/(1 + \epsilon)$ and ϵ is small. Then

$$g(a) = \frac{b}{1+\epsilon}(1 + \epsilon + \frac{n-1}{2}\epsilon^2 + \ldots)$$

so that $g(a) = b/(1 + \alpha\epsilon^2 + \ldots)$, and the convergence is quadratic.

The idea is now the following: if we consider $a_{k+1} = f(a_k)$, or $a_{k+1} = h(a_k)$, each iteration doubles the number of exact digits. In fact, the number of exact digits is the initial number times $2 - \log\alpha$, the quantity $\log\alpha = \log(n/2)$ is ϵ is (10). If we start with one exact digit, and need N digits, the cost is $\log N$. For instance, if $N = 1024$, we need 10 iterations.

2 Square roots

In the case $n = 2$, we can rewrite f, g as

$$f(a) = \left\lfloor \frac{\lfloor x/a \rfloor + a}{2} \right\rfloor \qquad g(a) = \frac{1}{2}\left(a + \frac{x}{a}\right) .$$

The essential cost of f is the division of x by a, adding a and dividing by 2 is linear w.r.t. the size of x. Experimentally, if we chose $x = 2^{2045}$, if we start with $a_1 = 2^{1023}$, the number of correct digits is

a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
1	2	5	11	24	48	97	195	308

We have $a_{10} = a_{11}$, but this is not always true: assume $x = c^2 - 1$, where c is an integer. Then $f(c) = c - 1$ and $f(c-1) = c$, so that f need not have a fixed point. By the definition of the integer part, $f(a)$ is the only integer satisfying

$$\frac{1}{2}(a + \frac{x}{a} + \frac{1}{a} - 2) \leq f(a) \leq \frac{1}{2}(a + \frac{x}{a}) . \tag{5}$$

From this, we have immediately: if $a \leq f(a)$ then $a \leq b$; if $f(a) \leq a$ then $x + 2 \leq (a+1)^2$. This implies $x + 1 < (a+1)^2$, then $b < a + 1$. In particular, if a is a fixed point, i.e. $f(a) = a$, we have $a = \lfloor \sqrt{x} \rfloor$. The fixed point is unique, and is the desired result. The first term of (5) is $b - 1 + [(a-b)^2 + 1]/(2a)$, so that $f(a) > b - 1$. Consider the sequence $a_{k+1} = f(a_k)$. Since $a > b$ implies $a > f(a)$, we shall have $a_1 > a_2 > ... > a_{k+1}$ with $a_i > b$. As a consequence, there must exist an index k such that $a_{k+1} \leq b$. Since $b - 1 < a_{k+1}$, we have $a_{k+1} = \lfloor \sqrt{x} \rfloor$.

Let $A = a_{k+1}$. We consider here the smallest k such that $f(A) \geq A$. If $f(A) = A$, the algorithm converges. Otherwise, we have

$$b - 1 + \frac{1}{2b} \leq f(A) \leq b + \frac{1}{2(b-1)}$$

provided that $b - 1 < A < b$. This is because $a + x/a$ is decreasing for $a \leq b$. This implies $f(A) \leq A + 1$. If $f(A) = A + 1$, equation (5) says $(A+1)^2 \leq x + 1$. If x is not of the form $c^2 - 1$, we deduce $(A+1)^2 \leq x$, $A + 1 \leq b$, contradicting $A > b - 1$. Thus, $f(A) = A$, except in the exceptional case where f has no fixed point.

Consider now the case $f(f(a)) = a$. In the case $f(a) = a$, we have our result. Otherwise, let A be the smallest of a and $f(a)$. Since A is in the image of f, we have $A > b - 1$. Since $A < f(A)$, we have $A + 1 \leq (A + x/A)/2$, $(A+1)^2 \leq x + 1$. Since $A + 1 > b$, this implies $(A+1)^2 = x + 1$, and we are in the exceptional case.

The algorithm is the following: Consider l such that

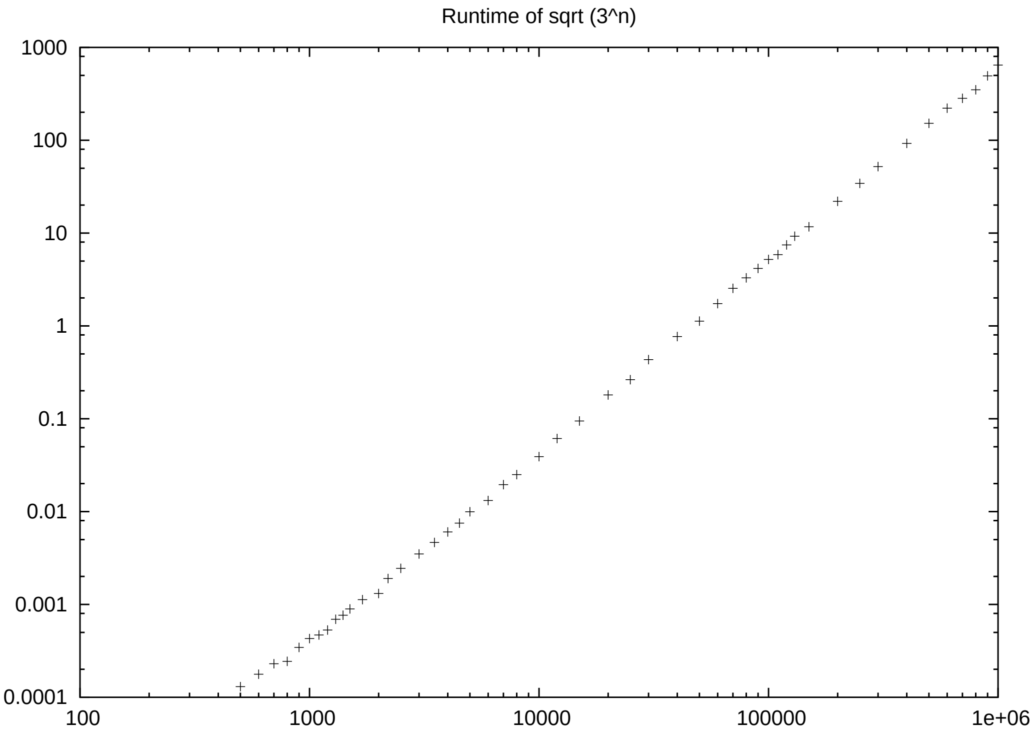
$$2^{l-1} \leq x \leq 2^l. \tag{6}$$

Let $s = \lfloor (l+1)/2 \rfloor$, and $a_0 = 2^s$. We consider $a_{k+1} = f(a_k)$ and find the first k such that $a_{k+1} = a_k$, or $a_{k+1} = a_{k-1}$, in this case, the smallest of a_k and a_{k+1} is returned.

3 General algorithm

Let l be as in (6). Write $l = sn + r$ by Euclidean division. Set

$$u_0 = \left\lfloor \frac{(n+1+r)2^s}{n} \right\rfloor. \tag{7}$$



Consider the sequence $u_{k+1} = h_x(u_k)$. Consider the first index k such that $k \geq 2$ and $u_k \geq u_{k-1}$. Write this quantity A_0. Consider this as a good guess of the result. Said otherwise, we find i such that $A_i = A + i$, and $A_i^n \leq x < A_{i+1}^n$, by trying in order A_0, A_1, A_2, etc, or downwards A_0, A_{-1}, A_{-2}, etc. This gives obviously the good result.

Example. Take $x = 10^8$ and $n = 3$. The initial value is 682, then comes 526, 471, and 464, that is a fixed point.

Example $x = 10^{10}$, $n = 3$. The values are 3413, 2562, 2216, 2156, and 2155. We also get a fixed point, but the result is 2154.

4 The J root algorithm

The alternate algorithm is the following. We consider $G_x(l)$ defined as follows. We defined u_0 as in (7). After that we write $u_{k+1} = f_x(u_k)$, and we consider the first k such that $u_{k+1} \geq u_k$. Then $G_x(l) = u_k$.

We consider $F(x, n, l, \epsilon)$ as follows. Let $l = sn + r$, Euclidean division of l by n. Let $y = \lfloor s/2 \rfloor - \epsilon$. If y is "small", then F is $G_x(l)$. Otherwise, let

$$u = F(\lfloor \frac{x}{2^{yn}} \rfloor, n, l - ny, \epsilon) \tag{8}$$

and

$$F(x, n, l, \epsilon) = \lfloor g_x(u2^y) \rfloor \, . \tag{9}$$

Finally, `jroot` is defined as follows. We define l and ϵ by

$$2^l \leq x < 2^{l+1} \qquad 2^{2\epsilon} \leq n < 2^{2\epsilon+2} \tag{10}$$

Let $A_0 = F(x, n, l, \epsilon)$. Consider this as a good guess of the result. Said otherwise, we find i such that $A_i = A + i$, and $A_i^n \le x < A_{i+1}^n$, by trying in order A_0, A_{-1}, A_{-2}, etc.

Example. Let $x = 10^{100000}$. Take $n = 3$. Let a be the result. The runtime for a^2 and a^3 is 0.62 and 1.87 seconds; the runtime for computing a via `iroot` is 36.27 seconds, 16 iterations are needed. In the case of `jroot`, the runtime is 4.48, with 1.47 for F.

Example. Let $x = 10^{1000000}$, $n = 30$. The number x is huge; it needs 97s to compute it, and 154s to print it. Computing the root with `iroot` costs 3840s. Each iteration costs 200s, 17 iterations are needed. In the other hand, the cost of `jroot` is 286.17, with 88 seconds for F, and 200s for checking that $a^n \le x$.

5 Complexity

Let us consider the cost of computing a^n. The method we use is the following

$$a^{2n} = (a^2)^n \qquad a^{2n+1} = a \times (a^2)^n.$$

For instance, for a^{16}, we compute a^2, a^4, a^8 and a^{16}. For a^{31} we compute a^2, a^3, a^4, a^7, a^{15}, a^{16} and a^{31}. Assume that $n = \sum b_k 2^k$ is the binary The number of products required is the number of the b_k plus the sum of the non-zero b_k (the last b_k is not counted here). This means that the number of multiplications required is of the order of $\log n$. However, assume that a is of size N, so that a^k is of size kN. Let's assume that the cost of a product of N_1 and N_2 bits costs $N_1 N_2$. Then the cost of a^{16} is $\sum k^2 N^2$, where k is 1, 2, 4 and 8. This is N^2 times the sum of powers of 4. If p the largest if the k^2, this is $N^2(4p - 1)/3$. In the case n is a power of two, we have $p = (n/2)^2$, the cost is near $N^2 n^2/3$. In the case of a^{31},

This is a sequence of real numbers, associated to the Newton Method. If we compare the two equations, then $f(a)$ is the unique integer satisfying

$$g(a) - \frac{a^{n-1} - 1}{na^{n-1}} \le h_x(a) \le g(a) + \frac{n-1}{n} \tag{11}$$

In particular, we get $|g(a) - f(a)| \le 1$. From the relation $g(a) \ge b$, we deduce $g(a) > 1$, hence $f(a) > 0$ (we exclude the case $x = 1$, where the solution is $b = 1$). As a consequence, the following algorithm gives the desidered result.

We start with u such that $2^{u-1} \le x \le 2^u$, write $u = qn + r$, define

$$a_0 = \left\lfloor \frac{2^q(n + r + 1)}{u} \right\rfloor$$

then iterate $a_{k+1} = f(a_k)$, and consider the first k such that $k \ge 2$ and $a_k \ge a_{k-1}$. Let $A_i = a_k + i$. We find by trial and error i such that $A_i^n \le x \le A_{i+1}^n$. Let N be the number of bits of b. This is essentially $\log(x)/n$. The numbers of iterations, the value of k, is essentially $\log N$. Let $M = \log x$. The cost of f is essentially $M^2 \log n$. This, we get a cost of $(i + \log N)M^2 \log n$. For instance, if $x = 10^{200}$, and $n = 13$, we have $i = 2$, and $\log N = 8$. In fact, $a_7 = a_8$, and this is too big, thus we have to compute $(a_8 - 1)^{13}$.

Question: what is the value of i in A_i. We have $f(a) \le a$ if and only if $b^n < a^n + a^{n+1}$. This is in particular true if $a \ge b$. We have $f(a) \ge a$ if and only if $b^n \ge a^n + (1 - n)a^{n-1}$. Note that this implies that f has a fixed point (but it is not unique for $n > 2$). In the case

$$a^n + (1 - n)a^{n-1} \ge (a - 1)^n$$

the condition $f(a) \ge a$ implies $b \ge a - 1$, hence $b - 1 \le a \le b + 1$. This condition is true if $a \ge n(n-1)/2$ (asymptotically). Thus, if a is big enough, the error with the result is rather small.