THE INTEGER SQUARE ROOT OF N VIA A BINARY SEARCH

by

T. F. Higginbotham, Ph.D.

Southeastern Louisiana University

ABSTRACT

An algorithm is presented which may be used to find the integer square root of N. The method is intended for use on a binary computer, where only addition, subtraction, multiplication, or division by 2 is required. The problem arose when the author was working on factoring large numbers, where the machine, the Honeywell DPS 8, had double precision integer addition and subtraction, and the simulation of multiplication was easy. The actual factoring of the large number was to be Fermat's Method, requiring only addition and subtraction, but the integer square root is required in order to test for termination. The algorithm is implemented in FORTRAN for ease of reading.

Students enjoy the unconventional approach to solving this problem. It isn't long before some of them think of other unusual solutions.

INTRODUCTION

Some students enjoy novel applications of a particular method to a common problem. In this case, we will find the integer square root of a number N without using anything but addition, subtraction, and multiplication, and division by 2, using a binary search.
The integer square root of a number is taken to be the integer part of the square root. For example, the integer square root of 5 is 2, of 10, 3, and of 105, 10.

This problem arose in connection the use of Fermat's Method of Factoring [1], in which only addition and subtraction are required. The machine on which the work was being carried out was a Honeywell DPS 8, a 36-bit machine with an accumulator - quotient register architecture, with double precision integer addition and subtraction, but not double precision integer multiplication or division. Double precision multiplication is easy to simulate on this system, but double precision division is somewhat more difficult.

To restate the problem; find the square root of a number N without the use of division, other than by 2, which may be accomplished by shifting right one bit.

ALGORITHM DEVELOPMENT.

The binary search [2] is usually thought of as an efficient method of locating a record based on the use of record numbers and a key.

But suppose the object of interest is the square root of N? The only difference then is the termination of the algorithm, in this case locating the integer root rather than a particular record.

ALGORITHM I.

This algorithm can be used to find the integer square root of N, using only integer addition, subtraction, and division by 2.

```
1.  Set the lower limit to 1
2.  Set the upper limit to N.
3.  Repeatedly
    (a)  calculate the middle value as
         (upper limit  +  lower limit) / 2
    (b)  if (lower limit) squared > N,
         then adjust the lower limit to one greater than
         the middle,
         else adjust the upper limit to one less than the
         middle.
4.  Set the square root accordingly.
```

This algorithm is easy to follow, and is easy to implement on a computer, either in assembly language or a higher level language. But there is a problem -- do you see it? Note that when N/2 is squared, it is a very large number as compared to the actual square root of N. In fact, with only moderate bad luck, the result will cause an overflow. For example, on a 36-bit machine where the zeroth bit is taken to be the sign bit (the usual case), division by 2 of a 24-bit number will result in a 23-bit number (shift one bit to the right), which, when squared, can result in a 46-bit number, which will cause an overflow!. The implication is that the largest number which can be squared on a 36-bit machine is only 17 bits, which is 131071 -- any reasonable factor table is larger than this!

So using N itself in isn't a good approach. We might approach this problem with the aid of logarithms, using base 2 because we do not have a logarithm table -- if we had such a table, the problem would be straight - forward. Calculating the integer part of the logarithm base 2 isn't too hard -- all that is needed is powers of 2. And when we get this logarithm, we can find a reasonable upper bound just by division by 2. So, our second algorithm could be the following;

ALGORITHM II

The only difference between this algorithm and Algorithm I is the method of finding the limits for the binary search. Let N be the number for which the integer root is desired, KP2 be powers of 2, KP2N be the approximate $\log_2$ of N, KL be the lower bound, KU be the upper bound, K be the estimated root, K1 be the integer square root, and K2 be the integer square root plus 1.

```
1.  Input N
2.  Output N
3.  set KP2     =   1
        KP2N    =   0
4.  Repeat, while KP2 is less than or equal to N
       set KP2  =   2 * KP2
       set KP2N =   KP2N   +   1
5.  set KL   =   $2^{((KP2N - 1) / 2)}$
    set KU   =   $2^{((KP2N + 1) / 2)}$
6.  Repeat, while KL is less than KU
       K    =    (KU + KL) / 2
       If $K^2$ is greater than N, set KU  =  K
       If $K^2$ is less than N,    set KL  =  K  +  1
       If $K^2$ is equal to N,     set KL  =  LU
7.  If $K^2$ is less than N,  then
        set K1    =    K
        set K2    =    K  +  1
8.  If $K^2$ is greater than N,  then
        set K1    =    K  -  1
        set K2    =    K
9.  If $K^2$ is equal to N,  then
        set K1    =    K
        set K2    =    K
10.  Output, K1, $K1^2$, N, K2, $K2^2$
```

Algorithm II is easy to implement, as can be seen from Figure I.

The worst possible case for a binary search is $\log_2 N$, and the average case is approximated as $\log_2(N + 1) - 1$,
$N > 50$.

So, to examine the search length required to find the integer square, we must look at the difference between

$$KL = 2^{((K2PN - 1) / 2)} <= N, \text{ and}$$
$$KU = 2^{((K2PN + 1) / 2)} >= N.$$

That difference is maximum when K2PN is odd; it being $2^{((K2PN - 1) / 2)}$. For example, if N were 120, K2PN would be 7, $KL = 2^{((7 - 1) / 2)} = 3$, $KU = 2^{((7 + 1) / 2)} = 4$, and the difference between KU and KL, $2^{((7 - 1) / 2)} = 8$.

Assuming the search length to be $2^{((K2PN - 1) / 2)}$, we have the maximum search to be

$$\log_2(2^{((K2PN - 1) / 2)}),$$

which is

$$(K2PN - 1) / 2.$$

Figure I

```
C
C     INTEGER SQUARE ROOT VIA A BINARY SEARCH
C
   10 CONTINUE
      INPUT N
      OUTPUT N
C
      KP2    =   1
      KP2N   =   0
C
      REPEAT  05, WHILE (KP2 .LE. N)
           KP2  =   2 * KP2
           KP2N =   KP2N   +   1
   05 CONTINUE
C
      KL    =   2 ** ((KP2N - 1) / 2)
      KU    =   2 ** ((KP2N + 1) / 2)
C
      REPEAT 20, WHILE (KL .LT. KU)
      OUTPUT, KP2N, KL, KU, K, ' ---- '
           K    =   (KU + KL) / 2
           IF (K ** 2 .GT. N)   KU  =  K
           IF (K ** 2 .LT. N)   KL  =  K  +  1
           IF (K ** 2 .EQ. N)   KL  =  KU
           OUTPUT, KL, KU, K, (K - 1) ** 2, N, K ** 2
           OUTPUT, '  '
   20 CONTINUE
C
      IF (K ** 2 .LT. N) THEN
          K1    =   K
          K2    =   K  +  1
      END IF
C
      IF (K ** 2 .GT. N) THEN
          K1    =   K  -  1
          K2    =   K
      END IF
C
      IF (K ** 2 .EQ. N) THEN
          K1    =   K
          K2    =   K
      END IF
C
      OUTPUT, K1, K1 ** 2, N, K2, K2 ** 2
      OUTPUT, ' '
C
      GO TO 10
      STOP
      END
```

The average search length would be

$$\log_2(2^{((K2PN - 1) / 2)} + 1) - 1$$

which is

$$(K2PN - 1) / 2 + 1) - 1 = (K2PN - 1) / 2.$$

Then the average and maximum search length are the same, which is a little surprising. In any event, the search for the integer square root of the largest number (17179869183) on a 36-bit machine, of which 34 bits (no overflows allowed), are used, requires only fifteen iterations. A trace of this may be found in Figure II.

Figure II

Iteration	Lower Limit	Upper Limit	Difference
1	65536	131072	65536
2	98305	131072	32767
3	114689	131072	16383
4	126977	131072	4095
5	129025	131072	2047
6	130049	131072	1023
7	130561	131072	511
8	130817	131072	255
9	130945	131072	127
10	131009	131072	63
11	131041	131072	31
12	131057	131072	15
13	131065	131072	7
14	131069	131072	3
15	131071	131072	1

$131071^2 = 17179607041$
$N = 17179869183$
$131072^2 = 17179869184$
$\sqrt{N} = 131071.999996$
$\mathrm{Trunc}(\sqrt{N}) = 131071$

DISCUSSION AND CONCLUSION

The method presented above can be used to find the integer square root of a number, using only addition, subtraction, and multiplication. Execution time isn't particularly important, although the actual execution time required was gratifying -- it is only carried out once per run, and will require only a fraction of a second compared to the time required for factoring the number.

Certainly, it would not take much effort to extend this method to cube, and higher roots.

**************************Integer Square Root References Continued On Page 49*********************

APPENDIX B: SAMPLE TOPICS FOR DEBATE*#

1. That there should be certification of computer professionals.
2. That computing should be taught in schools.
3. That playing computer games is purely a recreational activity.
4. That information technology is responsible for unemployment.
5. That information technology deskills the work force.
6. That the Australian legal system has kept up with advances in information technology.
7. That software prices are too high in Australia.
8. That the Australian government supports the hardware/software industry.
9. That information technology has improved our quality of life.
10. That the provision of telephone caller identification will be beneficial.
11. That computers provide an improved quality of life for the disabled.
12. That the Australia card has been introduced by the back door via the tax file number.
13. That information technology has created employment opportunities.

* Topical in Australia in 1992.
# The students were divided into three tutorial groups of about fourteen members, so each of the topics could be used more than once.

************************Integer Square Root References Continued From Page 45********************

I have found my students enjoy unconventional approaches to conventional problems. Neither they or I have ever seen a square root extracted using a binary search. And shouldn't education, or at least part of it, be fun?

References

1. Hans Riesel, *Prine Numbers and Computer Methods for Factorization*, Birkhauser, Boston, 1985, pg. 153-156.

2. R. G. Dromey, *How to Solve it by Computer*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982, pp. 227 - 237.