mpi
INFORMATIK

Fast Recursive Division

Christoph Burnikel Joachim Ziegler

MPI–I–98–1–022 October 1998

FORSCHUNGSBERICHT RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Im Stadtwald 66123 Saarbrücken Germany

Author's Address

Max-Planck-Institut für Informatik Im Stadtwald D-66123 Saarbrücken
email: burnikel,ziegler@mpi-sb.mpg.de

Acknowledgements

This work was supported in part by the ESPRIT Basic Research Actions Program of the EC under contract No. 7141(ALCOM II) and the BMFT(Förderungskennzeichen ITS 9103).

Abstract

We present a new recursive method for division with remainder of integers. Its running time is $2K(n)+O(n\log n)$ for division of a $2n$-digit number by an n-digit number where $K(n)$ is the Karatsuba multiplication time. It pays in practice for numbers with 860 bits or more. Then we show how we can lower this bound to $3/2K(n)+O(n\log n)$ if we are not interested in the remainder. As an application of division with remainder we show how to speedup modular multiplication. We also give practical results of an implementation that allow us to say that we have the fastest integer division on a SPARC architecture compared to all other integer packages we know of.

Keywords

Multiprecision Arithmetic, Integer Arithmetic, Karatsuba Method, School Methods, Fast Division, Division Without Remainder, Modular Reduction

1 Introduction

Multiprecision arithmetic is the art of computing with numbers that are larger than one machine word. It is an important subject in many domains of computer science, e.g. cryptography and computer algebra. Addition, subtraction, multiplication, and division of arbitrarily large numbers lie at the very heart of all computations involved in the algorithms used in these domains. For many practical applications it suffices to implement the classical school algorithms for basic arithmetic. However, if the numbers become sufficiently large, say about 200 decimal digits, it pays well to use more sophisticated algorithms, like for example Karatsuba's famous method for multiplication [10], which lowers the asymptotic running time of $\Theta(n^2)$ for multiplying two n-digit numbers by ordinary school multiplication to $O(n^{\log 3})$. We show in this paper how we can lower the $\Theta(n^2)$ bound for school division of a $2n$-digit number by an n-digit number to twice the time it takes to multiply two n-digit numbers with Karatsuba's method. Then we expand this method to a fast algorithm for dividing arbitrary integers. Our algorithm performs very well in practice and yields a speedup of more than 20% with numbers in the typical cryptographic range of 1024–2048 bits. The LEDA library [7] will soon ship with this algorithm implemented into its `integer` class for arbitrarily large integers. Then we show how we can lower this bound even further to $3/2K(n) + O(n \log n)$ if we are not interested in the remainder. We argue why one should use our method as the division method of choice in the cryptographic range. As an application of division with remainder we show how to speedup modular multiplication. This is the basis step of modular exponentiation, a method widely used in public key cryptosystems. We also give practical results of an implementation. By comparing it to other public domain packages we can say that we have the fastest integer division on a SPARC architecture compared to all other integer packages we know of.

In general, integers of arbitrary length are represented internally by combining several integers into one large vector, just as (in C notation) a `long` may consist of two consecutive `shorts` on the underlying machine. All algorithms on large integers can then be performed on these vectors. So let us introduce some important terms for this situation. A *digit* is the largest integer unit that can be handled directly by our machine's hardware. So, in C notation, a digit is an `unsigned long int`. At the time of this writing, it is best to have a 32-bit integer in mind when speaking of a digit. Now with $\beta - 1$ defined as the maximal value of a digit, β is the *base* of our number system. This means that the n-digit vector $[a_{n-1}, a_1, \ldots, a_0]_\beta$ represents the n-digit number $A = a_{n-1}\beta^{n-1} + \ldots + a_1\beta + a_0$ with respect to the base β. We will omit β in this notation if the base results from the context.

β is machine dependent and has grown successively in the last 30 years from 2^8 over 2^{16} to 2^{32}, and will be 2^{64} on all fast machines in the near future. With the assumption that β will always be a power of two, $\gamma := \sqrt{\beta}$ is a number with half as many bits as β and will be called the *halfbase*. By a *halfdigit* we mean a digit whose most significant half is zero, or in other words, a digit that is less than γ. A number A is said to *normalized* if its most significant digit a_{n-1} is greater or equal than $\beta/2$, i.e., if it has a leading 1 in its binary representation. If $X = [x_{n-1}, \ldots, x_0]$ and $Y = [y_{m-1}, \ldots, y_0]$ we denote with $[X, Y]$ the *composition of* X *and* Y, i.e., the $n + m$-digit number $[x_{n-1}, \ldots, x_0, y_{m-1}, \ldots, y_0]$. Let $K(n) := c_K \cdot n^{\log 3}$ denote the *Karatsuba multiplication time* for some convenient constant c_K.

2 Division with remainder

In this section we deal with the problem to find for non-negative integers A and B a representation $A = Q \cdot B + R$ with the (integral) quotient $Q = \lfloor A/B \rfloor$ and remainder R with $0 \leq R < B$.

2.1 Dividing two digits by one digit

The well known school method for dividing integer numbers uses as basic step the division of a two digit number by a one digit number to make a guess about the next digit of the quotient [5]. In elementary school we all have learned how to carry out this computation: we just invert the one-by-one multiplication table which we know by heart. In multiprecision arithmetic we are faced with the same problem, except that now our digits may be much larger so that a pre-built table were extremely large. Now, for example, how do we divide a 64-bit integer by a 32-bit integer on a 32-bit machine?

Suppose our machine has no built-in floating point arithmetic so that we cannot use tricks like computing a floating point approximation of the quotient and doing necessary adjustings to compute the desired result. Suppose further that the machine has no built-in integer division instruction and/or that we want to write portable C/C++ code so that we cannot use machine instructions (like `UDIV` on a SPARC Ultra) that are explicitly designed for this task. Then we need a function `DivTwoDigitsByOne` based on C/C++'s internal `unsigned int` arithmetic. You may wonder why we begin with such a basic problem but as you will soon see its solution will expand to a fast recursive algorithm for dividing arbitrary integers. The main idea is to reduce the division of two digits by one digit (which cannot be carried out by the machine or the high level language) to several divisions of a digit by a halfdigit, simulating school division.

For this we partition the dividend A into four halfdigits $A = [a_1, a_2, a_3, a_4]$ and the divisor B into two halfdigits $B = [b_1, b_2]$ (see Figure 1). Note that our machine is able to divide two halfdigits by one halfdigit internally. Then we do ordinary school division with these parts. This means that we use the parts $[a_1, a_2, a_3]$ and $[b_1, b_2]$ to determine the first digit q_1 of the quotient (note that q_1 does not depend on a_4, see Lemma 1). This is done by dividing $[a_1, a_2]$ by b_1, getting a candidate $\widehat{q_1}$ for q_1 which is at most 2 too large (see Lemma 2) and multiplying $\widehat{q_1}$ back with b_1 and b_2, subtracting it from $[a_1, a_2, a_3]$ and testing whether we have now a nonnegative remainder $R = [r_1, r_2]$. If not so, we decrement $\widehat{q_1}$ by one, increment R by B and test again. This test may fail now only once more, so that eventually we have to adjust $\widehat{q_1}$ and R a second time. Then we have found $q_1 = \widehat{q_1}$. Let us combine this process of finding the first digit of the quotient consisting of a division followed by a backmultiplication, some comparisons and adjustings into the subroutine `DivThreeHalvesByTwo`. It combines exactly the steps we make in school division to compute the first digit of the quotient.

Then like in school division we continue with the intermediate remainder $R = [r_1, r_2]$ and a_4 to compute the second digit q_2 of the quotient and the overall remainder $S = [s_1, s_2]$ by applying `DivThreeHalvesByTwo` a second time.

`DivTwoDigitsByOne` has two preconditions. The quotient Q has to fit into one digit. B has to be normalized, i.e., it must be greater or equal than $\beta/2$ (this amounts to say that it has a leading 1 in its binary representation; if this is not the case we can shift B and A to the left). This guarantees that $\widehat{q_1}$ is at most 2 too large (see Lemma 2).

Under these preconditions the pseudo-code looks like this. Capital letters stand for digits, small letters for halfdigits.

```
function DivTwoDigitsByOne(digit AH, digit AL, digit B)
{
(1)   Let [a1,a2]=AH, [a3,a4]=AL, and [b1,b2]=B;
(2)   [q1,R]=DivThreeHalvesByTwo(a1,a2,a3,b1,b2);
(3)   Let [r1,r2]=R;
(4)   [q2,S]=DivThreeHalvesByTwo(r1,r2,a4,b1,b2);
(5)   Return the quotient Q=[q1,q2] and the remainder S;
}
```

All computations are carried out in

```
function DivThreeHalvesByTwo(digit a1,a2,a3,b1,b2)
{
(6)   Divide q = [a1,a2]/b1;
(7)   Let c=[a1,a2]-q*b1;
(8)   D = q*b2;
(9)   Let R:=[r1,r2]=[c,a3]-D
(10)   if (R<0)  {   // q is too large by at least one
(11)      q--;
(12)      R+=B;
(13)      if (R<0) { // q is still too large
(14)         q--;
(15)        R+=B;    // now R is correct
(16)      }
(17)   }
(18)  Return the quotient q and the remainder R;
}
```

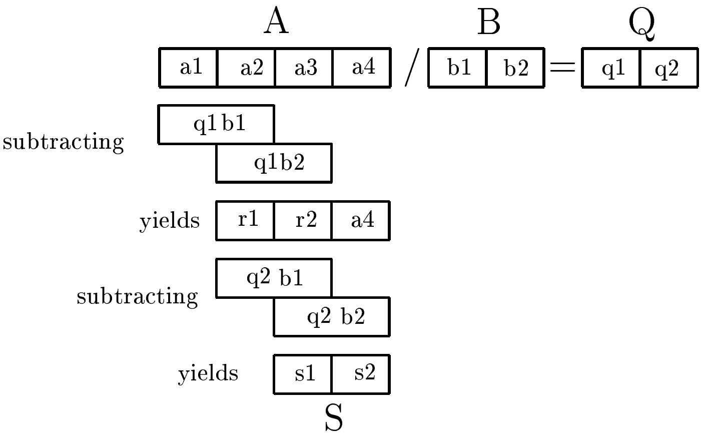

Figure 1: DivTwoDigitsByOne is a school division on halfdigits

The test in line (10) is actually the test whether $[c, a_3]$ is less than D because digits are never negative. If we look closely at this function, we see that all computations can be carried out with single digits unless q in line (6) does not fit into a halfdigit, which is the case if and only if $a_1 = b_1$, a case that may well occur (q will then be $\beta/2$ or $\beta/2+1$). In this case we need a special function `DivThreeHalvesByTwoSpecial` whose structure is so similar to that of `DivThreeHalvesByTwo` that we do not give the code here.

We will prove the correctness of our code in more generality in the next section. `DivTwoDigitsByOne` is just a toy for introducing the ideas of our recursive division. Moreover, it gives a fine, portable implementation of the basic case of division that has an acceptable performance.

2.2 Recursive division

Now we extend the method for dividing two digits by one to a recursive algorithm for dividing a $2n$-digit number by an n-digit number with a n-digit remainder. We will call this algorithm $D_{2n/1n}$. Under the assumption that n is even and large enough to benefit from a recursion, we again split A into four parts $[A_1, A_2, A_3, A_4]$ of length $n/2$ each, and B into two parts $[B_1, B_2]$. First we compute the upper part Q_1 of the quotient $\lfloor A/B \rfloor$ using the parts A_1, A_2, A_3 of A and the upper part B_1 of B by an appropriate algorithm called $D_{3n/2n}$ that recursively calls $D_{2n/1n}$ and is basically an expansion of `DivThreeHalvesByTwo`.. Using the remainder corresponding to Q_1 and the lowest part A_4 of A we then get similarly the lower part Q_2 of the quotient by algorithm $D_{3n/2n}$ again.

Algorithm 1. *($D_{2n/1n}$)*
Let A and B be nonnegative integers, and let $A < \beta^n B$ and $\beta^n/2 \le B < \beta^n$. Algorithm $D_{2n/1n}$ computes the quotient $Q = \lfloor A/B \rfloor$ with remainder $R = A - Q \cdot B$.

1. *If n is odd or smaller than some convenient constant, compute Q and R by school division and return.*

2. *Split A into four parts $A = [A_1, \dots, A_4]$ with $A_i < \beta^{n/2}$. Split B into two parts $B = [B_1, B_2]$ with $B_i < \beta^{n/2}$.*

3. *Compute the high part Q_1 of $\lfloor A/B \rfloor$ as $Q_1 = \lfloor [A_1, A_2, A_3]/[B_1, B_2] \rfloor$ with remainder $R_1 = [R_{1,1}, R_{1,2}]$, using algorithm $D_{3n/2n}$.*

4. *Compute the low part Q_2 of $\lfloor A/B \rfloor$ as $Q_2 = \lfloor [R_{1,1}, R_{1,2}, A_4]/[B_1, B_2] \rfloor$ with remainder R, using algorithm $D_{3n/2n}$.*

5. *Return $Q = [Q_1, Q_2]$ and R.*

Now let us argue why algorithm $D_{2n/1n}$ is correct (under the assumption that $D_{3n/2n}$ is correct).

Lemma 1. *Let for $k \in \mathbb{N}$ and $B > 0$, $A_1 \ge 0$*

$$A = A_1 \beta^k + A_r \quad , \quad 0 \le A_r < \beta^k.$$

Then for $Q = \lfloor A/B \rfloor$ and $Q_1 = \lfloor A_1/B \rfloor$ we have

$$Q = Q_1 \beta^k + Q_r \quad , \quad 0 \le Q_r < \beta^k.$$

Moreover with $R_1 := A_1 - Q_1 B$, we have

$$Q_r = \lfloor (R_1 \beta^k + A_r)/B \rfloor$$

and

$$A - BQ \;=\; R_1\beta^k + A_r - Q_r B.$$

Informally this says that if B is any number greater than 0 and A any number, then by dividing all but the last k digits of A by B we already get all but the last k digits of the quotient. (We use the lemma in Algorithm $D_{2n/1n}$ with $k = n/2$ and $A_r = A_4$, $A_1 = [A_1, A_2, A_3]$ and $B_r = B_2$.)

In particular, Lemma 1 shows that in algorithm $D_{2n/1n}$ the value Q_1 does not depend on A_4 and hence our way of calling $D_{3n/2n}$ is correct. Moreover, the lemma says that it is correct to go on with the remainder R_1 with which we are left after determining Q_1 in order to compute Q_2 by applying $D_{3n/2n}$ a second time to $[R_1, A_4]$ and B and that the overall remainder will be the remainder we get out of the second division. This proves the correctness of algorithm $D_{2n/1n}$.

Proof of Lemma 1: Since $R_1 < B$ and $A_r < \beta^k$ we also have $R_1\beta^k + A_r < \beta^k B$. Now

$$\begin{aligned} A - (1+Q_1)\beta^k B &= (A_1 - Q_1 B)\beta^k + A_r - B\beta^k \\ &= R_1\beta^k + A_r - B\beta^k \\ &< B\beta^k - B\beta^k \\ &= 0 \end{aligned}$$

implies that $Q < (1+Q_1)\beta^k$. This shows that $Q_r < \beta^k$. Moreover, for $Q'_r = \lfloor (R_1\beta^k + A_r)/B \rfloor$ and $Q' = Q_1\beta^k + Q'_r$ we have

$$\begin{aligned} A - BQ' &= A_1\beta^k + A_r - BQ_1\beta^k - BQ'_r \\ &= (A_1 - BQ_1)\beta^k + A_r - BQ'_r \\ &= R_1\beta^k + A_r - BQ'_r, \end{aligned} \tag{1}$$

which is a number from $[0, B-1]$. This shows that $Q = Q'$ and hence $Q_r = Q'_r$. Equation 1 completes the proof of our lemma. □

Now we come to algorithm $D_{3n/2n}$. Our goal is to divide a $3n$-digit number A by a $2n$-digit number B computing an n-digit remainder R. The idea is to truncate A to $2n$ digits and to truncate B to n digits and to compute the quotient $\widehat{Q}$ of the truncated A and B using algorithm $D_{2n/1n}$. As we will soon see, $\widehat{Q}$ is a good approximation of the exact quotient Q and not smaller than Q. Computing the remainder $\widehat{R} = A - B\widehat{Q}$ requires a multiplication on n-digit numbers, for which we will use Karatsuba's method. This use of Karatsuba multiplication is exactly the reason why we can expect a speed-up of $D_{2n/1n}$ over school division. Karatsuba's method is particularly convenient because it also divides its input into two halves.

Algorithm 2. *($D_{3n/2n}$)*
Let A and B be nonnegative integers, and let $A < \beta^n B$ and $\beta^{2n}/2 \le B < \beta^{2n}$. Algorithm $D_{3n/2n}$ computes the quotient $Q = \lfloor A/B \rfloor$ with remainder $R = A - Q \cdot B$.

1. *Split A into three parts $A = [A_1, A_2, A_3]$ with $A_i < \beta^n$.*

2. *Split B into two parts $B = [B_1, B_2]$ with $B_i < \beta^n$.*

3. *Distinguish the cases $A_1 < B_1$ or $A_1 \ge B_1$.*

(a) *If* $A_1 < B_1$, *compute* $\widehat{Q} = \lfloor [A_1, A_2]/B_1 \rfloor$ *with remainder* R_1 *using algorithm* $D_{2n/1n}$.

(b) *If* $A_1 \geq B_1$, *set* $\widehat{Q} = \beta^n - 1$ *and set* $R_1 = [A_1, A_2] - [B_1, 0] + [0, B_1] \quad (= [A_1, A_2] - \widehat{Q}B_1)$.

4. *Compute* $D = \widehat{Q} \cdot B_2$ *using Karatsuba multiplication.*

5. *Compute* $\widehat{R} = R_1\beta^n + A_4 - D$.

6. *As long as* $\widehat{R} < 0$, *repeat*

   (a) $\widehat{R} = \widehat{R} + B$

   (b) $\widehat{Q} = \widehat{Q} - 1$.

7. *Return* $R = \widehat{R}$, $Q = \widehat{Q}$.

Now why is this algorithm correct? To answer this we use the following lemma.

Lemma 2. *Let for* $k, l, A_1, A_r, B_1, B_r \in \mathbb{N}$

$$\begin{array}{rclcccc} A & = & A_1\beta^k + A_r & , & 0 \leq A < \beta^l B & , & 0 \leq A_r < \beta^k \\ B & = & B_1\beta^k + B_r & , & \beta^l/2 \leq B_1 < \beta^l & , & 0 \leq B_r < \beta^k. \end{array}$$

Note that our assumptions guarantee that $A/B < \beta^l$ *and* B *is normalized. Then for* $Q = \lfloor A/B \rfloor$ *and* $\widehat{Q} = \lfloor A_1/B_1 \rfloor$ *we have*

$$Q \leq \widehat{Q} \leq Q + 2.$$

Moreover, if $Q < \beta^l/2$ *we have the sharper estimate* $Q \leq \widehat{Q} \leq Q + 1$.

Informally this says that if A is any number, B is any normalized number with $l+k$ digits and the quotient A/B fits into l digits we can make a good guess about this quotient if we just divide all but the last k digits of A by the first l digits of B.

We apply the lemma with $k = l = n$. The lemma's bound for $\widehat{Q}$ applies immediately for the more likely case of $A_1 < B_1$ in algorithm $D_{3n/2n}$. In the special case of $A_1 \geq B_1$, our choice of $\widehat{Q} = \beta^n - 1$ is even smaller than the choice of $\widehat{Q}$ in the lemma, but on the other hand $\beta^n - 1$ is certainly no less than the exact value of Q. Hence the lemma's bound applies in the special case too. Note that we always maintain the invariant in our algorithm that $\widehat{R} = A - B\widehat{Q}$. We conclude that the body of the loop in step 6 of algorithm $D_{3n/2n}$ is processed at most twice. Since the initial value of $\widehat{Q}$ is no less than the exact Q, the initial value of $\widehat{R}$ is no bigger than B, which shows that at the end of step 6 we must have $\widehat{R} = R$ and consequently $\widehat{Q} = Q$. This shows the correctness of algorithm $D_{3n/2n}$.

Proof of Lemma 2: Dividing A_1 by B_1 yields a representation $A_1 = B_1\widehat{Q} + R_1$. Now let us consider $R(\widehat{Q}) = A - B\widehat{Q}$:

$$\begin{aligned} R(\widehat{Q}) = A - B\widehat{Q} &= A_1\beta^k + A_r - (B_1\beta^k + B_r)\widehat{Q} \\ &= (A_1 - B_1\widehat{Q})\beta^k + A_r - B_r\widehat{Q} \\ &= R_1\beta^k + A_r - B_r\widehat{Q}. \end{aligned} \qquad (2)$$

Since $R_1 < B_1$ and $A_r < \beta^k$ we also have $R_1\beta^k + A_r < R_1\beta^k + \beta^k = (R_1+1)\beta^k \leq B_1\beta^k$. Hence

$$\begin{array}{rcll} R(\widehat{Q}) & \leq & R_1\beta^k + A_r & \text{by equation 2 since } B_r\widehat{Q} \geq 0 \\ & < & B_1\beta^k & \text{see above} \\ & \leq & B, & \end{array}$$

which implies $Q \leq \widehat{Q}$. For the lower bound of $R(\widehat{Q})$ we need the following fact.

Claim 1. $\widehat{Q} \leq \beta^l + 1$.

Proof: The estimate

$$\begin{array}{rcll} A_1\beta^k & = & A - A_r & \\ & < & B\beta^l - A_r & \text{by our assumption on } A \\ & = & (B_1\beta^k + B_r)\beta^l - A_r & \\ & < & (B_1+1)\beta^l\beta^k & \text{because } B_r < \beta^k \text{ and } A_r \geq 0 \end{array}$$

shows by canceling out β^k that $A_1 - B_1\beta^l < \beta^l$. By this we get

$$A_1 - (\beta^l + 2)B_1 = (A_1 - \beta^l B_1) - 2B_1 < \beta^l - 2B_1 \leq 0$$

by our assumption on B_1, which shows that in fact $\widehat{Q} < \beta^l + 2$. Now we proceed with the proof of our lemma. Using Claim 1 we find

$$\begin{array}{rcll} A - B(\widehat{Q} - 2) & = & 2B + R(\widehat{Q}) & \\ & = & 2B + (R_1\beta^k + A_r) - B_r\widehat{Q} & \text{by equation 2} \\ & \geq & 2(B_1\beta^k + B_r) - B_r\widehat{Q} & \text{since } R_1\beta^k + A_r \geq 0 \\ & = & 2B_1\beta^k - B_r(\widehat{Q} - 2) & \\ & > & \beta^{k+l} - \beta^k\beta^l & \text{by claim 1 and our assumption on } B_1 \\ & = & 0. & \end{array}$$

This implies $\widehat{Q} - 2 \leq Q$. Similarly, for $Q < \beta^l/2$ we have

$$\begin{array}{rcll} A - B(\widehat{Q} - 1) & \geq & (B_1\beta^k + B_r) - B_r\widehat{Q} & \\ & = & B_1\beta^k - B_r(\widehat{Q} - 1) & \\ & \geq & \beta^l/2 \cdot \beta^k - B_r(Q+1) & , \quad \text{since } \widehat{Q} \leq Q + 2 \\ & > & \beta^{k+l}/2 - \beta^{k+l}/2 & , \quad \text{since } Q + 1 \leq \beta^l/2 \\ & = & 0. & \end{array}$$

This implies $\widehat{Q} - 1 \leq Q$, completing the proof of our lemma. □

We implement algorithm $D_{2n/1n}$ by the procedure `RecursiveDivision`. This procedure has the same structure as `DivTwoDigitsByOne` except that it works on large numbers instead of single digits. So it uses a subroutine `DivThreeLongHalvesByTwo` implementing algorithm $D_{3n/2n}$ to compute the upper and lower parts Q_1, Q_2 of the quotient which consist now of $n/2$ digits each. If n is odd or less than some constant `DIV_LIMIT` we are in the base case of the recursion in which we switch back to ordinary school division (although we could also run down the whole recursion tree until we are left with 2 digits to be divided by 1 digit, a fine task for `DivTwoDigitsByOne`).

We have two preconditions for this recursion to work. B has to be normalized and the quotient Q must fit into n digits. Then the pseudo-code looks like this.

```
function RecursiveDivision(digit_array A,digit_array B, integer n)
{
(19)  if (n is odd or n <= DIV_LIMIT)
(20)    SchoolDivision(A,B,Q,S); // base of the recursion
(21)  else {
(22)    Let [A1,A2,A3,A4]=A and [B1,B2]=B;
(23)    [Q1,R]=DivThreeLongHalvesByTwo(A1,A2,A3,B1,B2,n/2);
(24)    Let [R1,R2]=R;
(25)    [Q2,S]=DivThreeLongHalvesByTwo(R1,R2,A4,B1,B2,n/2);
(26)    }
(27)  Return the quotient Q=[Q1,Q2] and the remainder S;
}
```

Procedure `DivThreeLongHalvesByTwo` has a similar structure as `DivThreeHalvesByTwo`, except that lines (6) and (7) now become

```
(6') [Q,C]=RecursiveDivision([A1,A2],B1) ;
```

Moreover, we have to handle the case of $A_1 = B_1$ separately.

Let us now analyze the running time of algorithm $D_{2n/1n}$ and procedure `RecursiveDivision`, respectively. Let $M(n)$ denote the time it takes to multiply two n-digit numbers and $D(n)$ the time it takes to divide a $2n$-digit number by an n-digit number. The splitting operations in lines (22) and (24) are done in $O(n)$ (or even in time $O(1)$ if we use clever pointer arithmetic and temporary space management). `DivThreeLongHalvesByTwo` makes a recursive call in line (6') which takes $D(n/2)$. The backmultiplication $Q \cdot B_2$ in line (8) takes $M(n/2)$. The remaining additions and subtractions take $O(n)$. Let c_r be a convenient constant so that the overall time for all splitting operations, additions, and subtractions is less than $c_r n$. `DivThreeLongHalvesByTwo` is called twice, so we have the recursion

$$\begin{aligned}
D(n) &\le 2D(n/2) + 2M(n/2) + c_r n \\
&\le 2[2D(n/4) + 2M(n/4) + c_r n/2)] + 2M(n/2) + c_r n \\
&= 4D(n/4) + 4M(n/4) + 2M(n/2) + 2c_r n/2 + c_r n \\
&\le \cdots \\
&\le 2^{\log n} D(1) + \sum_{i=1}^{\log n} 2^i M(n/2^i) + \sum_{i=1}^{\log n} c_r n \\
&= O(n) + \sum_{i=1}^{\log n} 2^i M(n/2^i) + O(n \log n)
\end{aligned}$$

At this point we see how the running time of our method depends on the multiplication method we use. If we use ordinary school method, we have $M(n) = n^2$ and $D(n)$ will be $n^2 + O(n \log n)$, which is even a little bit worse than school division.

If we use Karatsuba's method with $M(n) = K(n) = c_K \cdot n^{\log 3}$, the above sum has the upper bound $2K(n) + O(n \log n)$ because in this case $K(n/2^i) = K(n)/3^i$. The use of even faster multiplication methods is discussed in section 4.

We summarize our result in the following

Theorem 1. *A $2n$-digit number can be divided by an n-digit number (including the computation of the remainder) with $2K(n) + O(n \log n)$ steps if the quotient fits into n digits and the divisor is normalized.*

Note that, as in school division, the only basic divisions of two digits by one that are really carried out are divisions by the most significant digit of B. Further note that the constant 2 is just an upper bound for the sum $\sum_{i=1}^{\log n}(2/3)^i$. We shall see in the experiments that the difference is noticeable.

The idea behind our recursive division $D_{2n/1n}$ is not completely new. Jebelean [4] proposes an algorithm that also splits the division into computing first the upper part and then the lower part of the quotient, by two recursions. However it is based on a probabilistic technique that yields correct results only with high probability. Jebelean claims that his algorithm has average running time $2K(n)$ like ours. However, in the worst case, Jebelean's algorithm resorts to school division which makes the running time quadratic.

2.3 Division of arbitrary integers

Now that we have seen how we can divide a $2n$-digit number by an n-digit number we still have to solve the problem how to divide an arbitrary r-digit number A by an arbitrary s-digit number B.

The main idea is to split up A into parts which are as long as B and to view these parts as (very large) digits. Then we can conceptually apply an ordinary school division to these large digits in which the base task of dividing two digits by one is carried out by `RecursiveDivision`. The divisor for `RecursiveDivision` is always B and the dividend is always the composition of the current remainder and the next digit of A. Figure 2 illustrates the method.

First we conceptually divide B into m so called "division blocks" of `DIV_LIMIT` digits each. We compute m as the minimal 2^k that is greater or equal than the number of division blocks of B. In other words, k will be the depth of the recursion applied to B in `RecursiveDivision`. We have to extend B to a number that has m·`DIV_LIMIT` digits. Note that the topmost recursive call will work no matter how big `DIV_LIMIT` actually is, i.e., we can try to minimize the number of digits in each division block (at least conceptually) as long as the overall number of digits of the m division blocks is greater or equal than the number of digits of B. This trick will minimize the number of 0s we waste when extending B and leave us with smaller numbers in the base case of the recursion; our tests have indicated that we gain a speedup of 25% in running time. So we compute $n = m \cdot j$ with j minimal such that n is greater or equal than s. Then we extend B from the right to n digits. B has to be shifted bitwise again for normalization until its most significant digit fulfills $b_{n-1} \geq \beta/2$. Note that A has also to be extended and shifted by the same amount.

Then we conceptually divide A into t division blocks of length n each. The division we will carry out will then conceptually be a school division of a t-block number by a 1-block number (here the term "block" stands for a very large block of digits, namely for n digits). This is a particularly simple type of school division since the divisor consists of only one block, which now plays the role of a single digit. No backmultiplication is needed, so school division becomes a linear time algorithm in the number of blocks.

Algorithm 3. *($D_{r/s}$)*
Let A and B be nonnegative integers such that $A < \beta^r$ and $B < \beta^s$. Algorithm $D_{r/s}$ computes

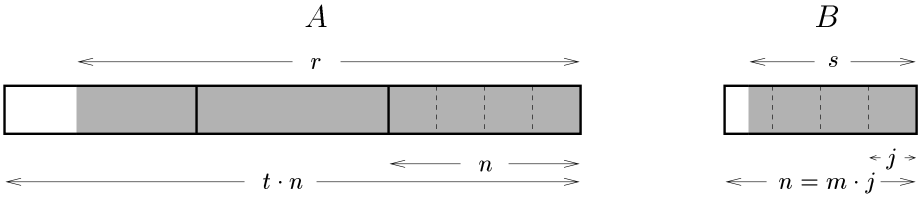

Figure 2: FastDivision is a t-by-1 division on large digit blocks. B is extended from s to n digits. Then A is extended to a multiple of n digits.

the quotient $Q = \lfloor A/B \rfloor$ with remainder $R = A - Q \cdot B$.

1. *Set $m = min\{2^k | 2^k \cdot \texttt{DIV\_LIMIT} > s\}$.*
2. *Set $j = \lceil s/m \rceil$ and $n = j \cdot m$.*
3. *Set $\sigma = max\{\tau | 2^\tau \cdot B < \beta^n\}$.*
4. *(a) Set $B = B \cdot 2^\sigma$ to normalize B.*
   (b) Set $A = A \cdot 2^\sigma$ to shift A by the same amount as B.
5. *Set $t = min\{l \geq 2 | A < \beta^{l \cdot n}/2\}$.*
6. *Split A conceptually into t blocks $[A_{t-1}, \ldots, A_0]_{\beta^n} = A$. Note that the leftmost bit of A_{t-1} is 0 by our choice of t.*
7. *Set $Z_{t-2} = [A_{t-1}, A_{t-2}]$.*
8. *For i from $t-2$ downto 0 do*
   (a) Using algorithm $D_{2n/1n}$ compute Q_i, R_i such that $Z_i = BQ_i + R_i$.
   (b) If $i > 0$, set $Z_{i-1} = [R_i, A_{i-1}]$.
9. *Return $Q = [Q_{t-2}, \ldots, Q_0]$ and $R = R_0 \cdot 2^{-\sigma}$.*

We prove now that algorithm $D_{r/s}$ is correct. The number Z_{t-2} with which we start the computation is bounded by $\beta^{2n}/2$ and hence that precondition of algorithm $D_{2n/1n}$ is fulfilled that ensures that the quotient is less than β^n. In the loop over the index i, the same precondition is always fulfilled because the remainders that we propagate are all smaller than B. Repeated application of Lemma 1 show that our way of successively computing the parts $Q_{t-2}, \ldots, Q_0$ of Q and propagating the remainders is correct and that in the end we will get the overall division remainder (left-shifted by 2^σ). This shows that algorithm $D_{2n/1n}$ computes the correct values of Q and R.

Let us analyze the running time. How many iterations do we have?

Claim 2. *We have $t \leq max\{2, \lceil (r+1)/s \rceil\}$ in algorithm $D_{r/s}$.*

Proof: We may assume that the highest digit b_{s-1} of B is nonzero. Then our choice of t ensures $t \leq \lceil (r+n-s+1)/n \rceil$ because, after shifting, $A < \beta^{r+n-s+1}/2$. If $r \leq s+1$, we clearly have $t = 2$. Otherwise, we consider the estimate $(u + x)/(v + x) \leq u/v$ which is true for $0 < v \leq u, x \geq 0$. Using this with $u = r + 1$, $v = s$, $x = n - s$ we get

$$\frac{r + n - s + 1}{n} = \frac{(r + 1) + (n - s)}{s + (n - s)} \leq \frac{r + 1}{s},$$

showing our claim by rounding up to the next integer. □

Algorithm $D_{2n/1n}$ is executed $t-1$ times and, because $n \leq 2 \cdot s$, every execution takes time $D(n) \leq D(2s) \leq K(2s) + O(2s \log 2s)$. With $K(s) = c_K \cdot s^{\log 3}$ we conclude that the overall running time is $O(r/s \cdot (K(s) + O(s \log s))) = O(rs^{\log(3)-1} + r \log s)$ which is a remarkable asymptotic improvement over the $\Theta(rs)$ time bound for ordinary school division. Let us summarize this result in the following

Theorem 2. *An s-digit number can be divided by an r-digit number (including the computation of the remainder) with $O(rs^{\log(3)-1} + r \log s)$ steps.*

3 Division without remainder

In this section we discuss divisions without remainder. The problem here is to compute only the quotient $Q = \lfloor A/B \rfloor$ of A and B but *not* the remainder $R = A - QB$. Of course, our hope is that Q alone can be computed faster than Q and R together. It will turn out that we can improve the algorithm for recursive division from the last section such that it computes quotients without remainder in average time $3/2K(n) + O(n \log n)$. However, in the worst case, namely if the remainder is small, the running time may become $5/2K(n) + O(n \log n)$. One important application of division without remainder is the computation of roots using Newton iteration. For the exact computation of the signs of arithmetic expressions involving roots, these roots may have to be computed up to very high precision, see [2]. Hence a subquadratic method for division is a must for this application.

3.1 Procedure D^-

Our new procedure for computing divisions without remainder has three phases. The first phase, called D^-, is very similar to the procedure $D_{2n/1n}$ which we henceforth denote by D^+ for simplicity. The basic idea is to omit the computation of the overall remainder in the second recursion. Look at Figure 3 for reference.

Algorithm 4. *(D^-)* **Procedure D^-:**
Let A and B be the divisor and the dividend, and let $A < \beta^n B$ and $\beta^n/2 \leq B < \beta^n$. Procedure D^- computes an approximation $\widehat{Q}$ of the quotient $\lfloor A/B \rfloor$.

1. *If n is odd or too small, compute $\widehat{Q} = \lfloor A/B \rfloor$ by school division.*
2. *Split A into four parts $A = [A_1, \ldots, A_4]$ with $A_i < \beta^{n/2}$. Split B into two parts $B = [B_1, B_2]$ with $B_i < \beta^{n/2}$.*
3. *Compute the high part Q_1 of $\lfloor A/B \rfloor$ as $Q_1 = \lfloor [A_1, A_2, A_3]/[B_1, B_2] \rfloor$ with remainder $R_1 = [R_{1,1}, R_{1,2}]$, using algorithm $D_{3n/2n}$.*
4. *Compute an approximation $\widehat{Q}_2$ of the low part Q_2 of $\lfloor A/B \rfloor$: Recursively set $\widehat{Q}_2$ to the result of algorithm D^- for the division $\lfloor R_1/B_1 \rfloor$.*

5. Concatenate Q_1 and $\widehat{Q}_2$ to the approximate quotient $\widehat{Q}$.

Steps 1, 2, 3 and 5 are identical for D^- and D^+. Note that in step 3 of D^- we need the full temporary remainder R_1 to proceed with the computation of $\widehat{Q}_2$. Here we call D^+. Later, in step 4 of D^-, we do *not* compute the remainder. Instead, we recursively call D^- to find an approximation $\widehat{Q}_2$ of the lower division part Q_2. Compare this to procedure $D^+ = D_{2n/1n}$ where in the second call of $D_{3n/2n}$ we do the same recursive division of R_1 by B_1, but using D^+, giving a candidate $\widehat{Q}_2$ for Q_2 that might be too large by at most 2. Another multiplication of $\widehat{Q}_2$ by B_2 is needed in step 4 of algorithm $D_{3n/2n}$ to find the corrected quotient Q_2 and the division remainder. The basic difference between D^+ and D^- is that this last multiplication step is omitted in D^- and hence $\widehat{Q}_2$ remains uncorrected. Of course, the result $\widehat{Q} = [Q_1, \widehat{Q}_2]$ is in general not equal to the exact result Q. As the following lemma shows, the error of $\widehat{Q}$ is not too large.

Lemma 3. $Q \leq \widehat{Q} \leq Q + 2\log(n)$.

Proof: Our argumentation is based on the observation that procedure D^- is called exactly once for each depth of the recursion. Let $\widehat{Q}_i$ be the approximate quotient computed by D^- at recursion depth i. Let Q_i be the exact quotient that $\widehat{Q}_i$ approximates. If we are in the bottom of the recursion, i.e. $i = \log n$, we have $\widehat{Q}_i = Q_i$ since we use here school division. Inductively, suppose that for an arbitrary depth i the computed quotient $\widehat{Q}_i$ is no less than the exact Q_i and no more than $2(\log(n) - i)$ too large. At recursion depth $i-1$, $\widehat{Q}_{i-1}$ is computed as the concatenation $\widehat{Q}_{i-1} = [Q_{1,i}, \widehat{Q}_i]$ of its exact upper half $Q_{1,i}$ and its inexact lower half $\widehat{Q}_i$. From the analysis of procedure D^+ we already know that $[Q_{1,i}, Q_i]$ is larger than Q_{i-1} by at most 2, hence by induction hypothesis $\widehat{Q}_{i-1}$ is no more than $2 + 2(\log(n) - i) = 2(\log(n) - (i-1))$ too large. Figure 3 illustrates the situation. This proves the lemma. □

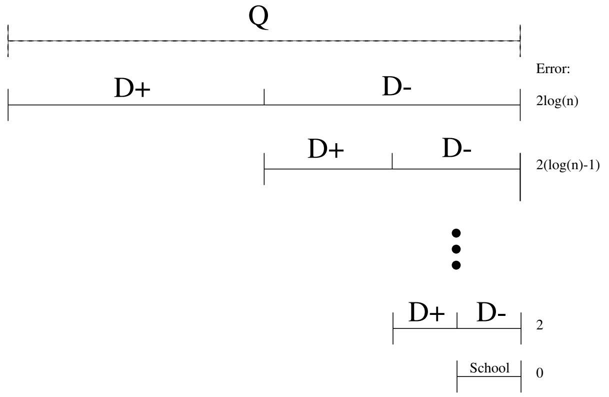

Figure 3: The error of $\widehat{Q}$ as computed by D^-

Lemma 4. *For $n = 2^k$, the running time of D^- is at most $3/2K(n) + O(n \log n)$.*

Proof: We denote the running time of algorithm D^- for an instance of size n by $D^-(n)$. Similarly, we denote the running time of algorithm D^+ by $D^+(n)$. Remember that $D^+(n) = 2K(n)$. To compute $D^-(n)$, we call once $D^+(n/2)$, do one Karatsuba multiplication on integers with $n/2$ digits, and do one recursive call to $D^-(n/2)$. All other operations have cost $O(n)$. Hence we have the recursion

$$\begin{aligned} D^-(n) &= D^+(n/2) + K(n/2) + D^-(n/2) + O(n) \\ &= 3K(n/2) + O(n \log n) + D^-(n/2) + O(n) \\ &= K(n) + D^-(n/2) + O(n \log n) \\ &\leq K(n) + D^-(n/2) + c \cdot n \log n, \end{aligned}$$

where in the last equation we used $K(n/2) = 1/3K(n)$. Solving the recursion we get

$$\begin{aligned} D^-(n) &\leq \sum_{i=0}^{k} K(n/2^i) + D^-(1) + \sum_{i=0}^{k} c \cdot (n/2^i) \log(n/2^i) \\ &\leq \sum_{i=0}^{k} K(n/2^i) + O(1) + 2c \cdot n \log n \\ &= \sum_{i=0}^{k} K(n)/3^i + O(n \log n) \\ &\leq K(n) \sum_{i=0}^{\infty} 1/3^i + O(n \log n) \\ &= 3/2K(n) + O(n \log n). \quad \square \end{aligned}$$

3.2 Division without remainder: General outline

Because in general D^- does not give us the exact division result, further correction steps are required to adjust $\widehat{Q}$ until we are left with the correct Q. The following lemma shows how we can do this. Let $\widetilde{Q}$ now be any approximate quotient of A and B.

1. $R(\widetilde{Q}, A, B) := A - B\widetilde{Q}$ is the remainder that corresponds to an approximate quotient $\widetilde{Q}$ of A and B. In the special case of $\widetilde{Q} = \lfloor A/B \rfloor$, it is the ordinary division remainder. If the division operands A and B are obvious from the context, we write $R(\widetilde{Q})$ for short. Moreover, we write R instead of $R(\lfloor A/B \rfloor)$.

2. $\delta(\widetilde{Q}, A, B) := \lfloor A/B \rfloor - \widetilde{Q}$. Again, if A and B are obvious from the context we write $\delta(\widetilde{Q})$ for short.

The following lemma states some useful relations between these quantities.

Lemma 5. *For any integral nonnegative A, B, $\widetilde{Q}$ we have*

1. $\delta(\widetilde{Q}, A, B) = \lfloor R(\widetilde{Q}, A, B)/B \rfloor$
2. $R(\lfloor A/B \rfloor, A, B) = R(\delta(\widetilde{Q}, A, B), R(\widetilde{Q}, A, B), B)$
3. $|R(\widetilde{Q}, A, B)| < (1 + |\delta(\widetilde{Q}, A, B)|)B$.

Proof: The quantity $q = \delta(\widetilde{Q}, A, B)$ satisfies

$$\begin{aligned} R(q, R(\widetilde{Q}, A, B), B) &= R(\widetilde{Q}, A, B) - \delta(\widetilde{Q}, A, B)B \\ &= (A - B\widetilde{Q}) - (\lfloor A/B \rfloor - \widetilde{Q})B \\ &= A - \lfloor A/B \rfloor B \qquad (3) \end{aligned}$$

which is equal to $R(\lfloor A/B \rfloor, A, B)$ and hence is contained in the interval $[0, \ldots, B-1]$. Because $\lfloor R(\widetilde{Q}, A, B)/B \rfloor$ is the only integer with this property, the first two parts of our lemma follow. The third part follows by using Equation 3 again:

$$\begin{aligned} |R(\widetilde{Q}, A, B)| &\leq |R(\widetilde{Q}, A, B) - \delta(\widetilde{Q}, A, B)B| + |\delta(\widetilde{Q}, A, B)|B \\ &= |A - \lfloor A/B \rfloor B| + |\delta(\widetilde{Q}, A, B)|B \\ &< (1 + |\delta(\widetilde{Q}, A, B)|)B. \quad \Box \end{aligned}$$

Lemma 5 can be applied as follows. Suppose that we want to compute the quotient $Q = \lfloor A/B \rfloor$ but we only know an approximation $\widehat{Q}$ of Q and its remainder $R(\widehat{Q}, A, B)$, which is bounded by $(1 + |\delta(\widehat{Q}, A, B)|)B$ according to lemma. If $\delta(\widehat{Q}, A, B)$ is not too large, say if it is bounded by a constant, then we can compute the quotient $\delta(\widehat{Q}, A, B) = \lfloor R(\widehat{Q}, A, B)/B \rfloor$ by ordinary school division. By the second part of the lemma, the remainder of this division is the wanted $R(\lfloor A/B \rfloor, A, B)$. If B has at most n digits, the school division takes only time $O(n)$ because all we have to do is to compute a (bounded) number of bits and each iteration takes time $O(n)$.

With this lemma at hands our algorithm for division without remainder has the following outline.

Algorithm 5. Division without remainder (Outline): *Let A and B be as in Algorithm 4. Proceed in three phases.*

1. *Call D^- to compute an approximation $\widehat{Q}$ of the exact quotient $Q = \lfloor A/B \rfloor$.*

2. *Try to compute $\delta(\widehat{Q}) = Q - \widehat{Q}$ in linear time. This phase may fail with a small probability.*

3. *If phase 2 fails, compute the remainder $R(\widehat{Q}) = A - B \cdot \widehat{Q}$, using Karatsuba multiplication. Then compute $\delta(\widehat{Q}) = \lfloor R(\widehat{Q})/B \rfloor$ using school division.*

4. *Set $Q = \widehat{Q} + \delta(\widehat{Q})$.*

We describe the simpler phase 3 first. This is the bad case but as we will see it will occur only with a very small probability.

Lemma 6. *Phase 3 has running time $K(n) + O(n \log \log n)$.*

Proof: The Karatsuba multiplication in phase 3 takes time $K(n)$. By Lemma 5 we know that in fact $\delta(\widehat{Q}) = Q - \widehat{Q} = \lfloor R(\widehat{Q})/B \rfloor$ and by Lemma 3 we know that this quotient is bounded by $2\log(n)$. Hence $\delta(\widehat{Q})$ fits into one digit for all practical values of β and of n. Under the assumption of $2 \log n < \beta$, we can compute $\lfloor R(\widehat{Q})/B \rfloor$ in linear time by school division. In general, $\delta(\widehat{Q})$ has size $O(\log \log n)$ and the division takes time $O(n \log \log n)$. $\Box$

3.3 Probabilistic quotient correction: The algorithm

In this section we explain the algorithm that implements phase 2 of Algorithm 5. Henceforth we make two (realistic) assumptions on out machine and the numbers with which we compute.

Assumption 1. *1. The digit length β is at least 2^{32}.*

2. The number of digits is bounded by $n < \beta/2$.

Assumption 1.2 is justified because already for $n > 2^{20}$ it is better to use Newton's method in combination with a FFT type multiplication, see section 4. Assumption 1.1 implies that the following procedure does not work for 8 bit or 16 bit systems. Straightforward modifications are required.

Phase 2 basically does the same job as phase 3, but it avoids the expensive back–multiplication $B \cdot \widehat{Q}$. Instead, here we only approximate this product. For technical reasons that will become clear later, we choose $Q' := max(0, \widehat{Q} - 2\log n)$ for the backmultiplication and approximate the product $P = B \cdot Q'$. This gives us upper and lower bounds for the remainder $R(Q') = A - Q' \cdot B$, which in turn give us upper and lower bounds $\delta_l \leq \delta(Q') \leq \delta_h$ for the desired quantity $\delta(Q') = \lfloor A/B \rfloor - Q'$. Our approximations will be good enough to guarantee $\delta_h - \delta_l \leq 1$. Phase 2 is successful if and only if $\delta_h = \delta_l$. As we will prove, the event $\delta_h \neq \delta_l$ that causes phase 2 to fail is very unlikely. We start now with the details of phase 2.

With respect to the base β, the numbers A, B and Q' read as

$$A = \sum_{i=0}^{2n-1} a_i \beta^i, \quad B = \sum_{i=0}^{n-1} b_i \beta^i, \quad Q' = \sum_{i=0}^{n-1} q'_i \beta^i$$

In the school multiplication of B and Q', there are n^2 products of digits $b_i q'_j$. We group these products by the sum of the exponents $i + j$. For $0 \leq k < 2n - 1$ we set

$$w_k = \sum_{i+j=k} b_i q'_j \beta^k.$$

It is easy to see that $P = Q' \cdot B$ is the sum of the w_k, i.e., $P = \sum_{k=0}^{2n-2} w_k$. The next lemma shows that the parts $w_0, \ldots, w_{n-4}$ only contribute a small amount to P, hence we will not compute them. Recall that we want to perform phase 2 in linear time.

Lemma 7. *For any l with $0 \leq l < 2l$,*

$$\sum_{k=0}^{l-1} w_k < 2n \cdot \beta^{l+1}.$$

Proof:

$$\sum_{k=0}^{l-1} w_k \leq \sum_{k=0}^{l-1} (k+1)\beta^{k+2} \leq l \cdot \beta^2 \cdot \sum_{k=0}^{l-1} \beta^k = l \cdot \frac{\beta^2}{\beta - 1} \cdot (\beta^l - 1) < l \cdot 2\beta \cdot \beta^l. \quad \square$$

We now apply the previous lemma with $l = n - 3$, which gives the bound

$$\sum_{k=0}^{n-4} w_k < 2(n-3)\beta^{n-2} < \beta^{n-1} \tag{4}$$

because of assumption 1.2. If we omit in P the terms $w_0, \ldots, w_{n-4}$, we get an approximation $P' := \sum_{k=n-3}^{2n-2} w_k$ of P. Remember that we chose Q' as the maximum of 0 and $\widehat{Q} - 2\log n$. We show now that $P' \leq A$. For $Q' = 0$ we have $P = P' = 0$ and so our statement is clear. Otherwise we have by Lemma 3

$$0 \leq P' \leq P = (\widehat{Q} - 2\log n)B \leq QB \leq A.$$

This implies that the approximate remainder $R'(Q') := A - P'$ is nonnegative. Equation 4 gives us an inclusion for $R(Q')$ in terms of $R'(Q')$:

$$R'(Q') \leq R(Q') \leq R'(Q') + 2(n-3)\beta^{n-2} < R'(Q') + \beta^{n-1}. \tag{5}$$

To simplify the computations, we only use the highest two digits $r' = [r'_n, r'_{n-1}]$ of $R'(Q')$ in the following. The situation is illustrated by Figure 4. Truncating the last $n-1$ digits of

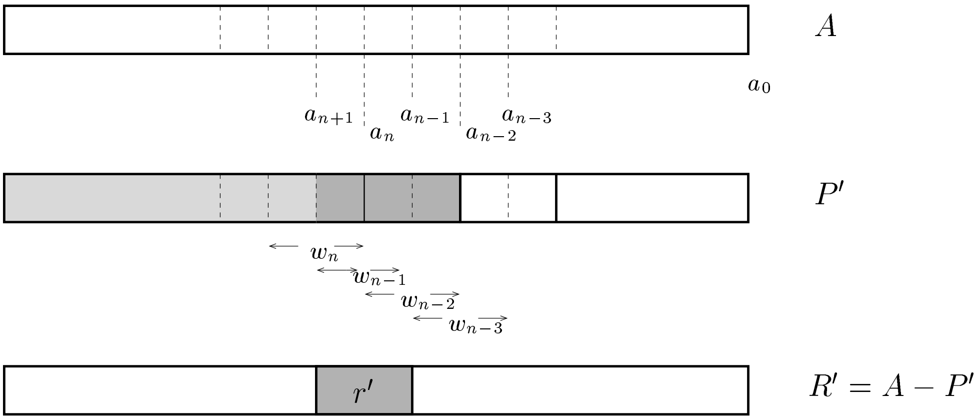

Figure 4: We compute only the dark shaded region of P' because the light shaded region is equal to the corresponding region of A and the unshaded region is too small anyway. From the difference $R' = A - P'$ we truncate the last $n-1$ digits; the remaining two digits form r'.

$R'(Q')$ causes an error of at most β^{n-1}, hence

$$r'\beta^{n-1} \leq R'(Q') \leq (r'+1)\beta^{n-1}.$$

Together with Equation 5 this implies another inclusion for $R(Q')$ that involves only r':

$$r'\beta^{n-1} \leq R(Q') \leq (r'+2)\beta^{n-1}. \tag{6}$$

There is a similar inclusion for the divisor B if we use only the highest digit b_{n-1} of B:

$$b_{n-1}\beta^{n-1} \leq B \leq (b_{n-1}+1)\beta^{n-1}.$$

Putting the last two inclusions together we obtain

$$\frac{r'}{b_{n-1}+1} \leq \frac{R(Q')}{B} \leq \frac{r'+2}{b_{n-1}}.$$

Remember that we want to compute the quantity $\delta(Q') = Q - Q'$, which is $\lfloor R(Q')/B \rfloor$ by Lemma 5. Rounding down to the next integer in the last estimate we finally get

$$\delta_l := \lfloor \frac{r'}{b_{n-1}+1} \rfloor \leq \delta(Q') \leq \lfloor \frac{r'+2}{b_{n-1}} \rfloor =: \delta_h. \tag{7}$$

If the lower bound δ_l is equal to the upper bound δ_h, then $\delta_l = \delta_h = \delta(Q')$ and we are done. Otherwise we declare phase 2 as failed.

Lemma 8. *The quantities δ_h and δ_l can be computed in linear time from Q'.*

Proof: We need to compute r'. Because of our assumptions $\beta \geq 2^{32}$ and $n < \beta/2$, we have

$$\delta(Q') \leq 2\log(n) \leq 64. \tag{8}$$

By Lemma 5, $R'(Q')$ is bounded by

$$R'(Q') \leq R(Q') < |1 + \delta(Q')|B \leq 65 \cdot \beta^n < \beta^{n+1}.$$

Hence, setting $M = \beta^{n+1}$ we have $R'(Q') = R'(Q')$ *mod* M and therefore

$$R'(Q') \;=\; \left(\sum_{i=0}^{n} a_i\beta^i - \sum_{i=n-3}^{n} w_k\right) \quad mod\ M.$$

This means that the parts $w_{n+1}, w_{n+3}, \ldots, w_{2n-2}$ need not be computed because they vanish in the reduction modulo M. Therefore only the 4 quantities $w_{n-3}, \ldots, w_n$ remain to be computed. Every term w_k takes $k+1$ digit multiplications. All other operations are done in constant time. This proves our lemma. □

We summarize the discussion in the following algorithm.

Algorithm 6. *(Probabilistic quotient correction) Let A, B be as in algorithm D^- and let $\widehat{Q}$ be the approximate quotient of A and B as computed by algorithm D^-. Either the algorithm returns $Q = \lfloor A/B \rfloor$ or it declares the computation as failed.*

1. *Set $Q' = max\{0, \widehat{Q} - 2\log n\}$.*
2. *Set $R'(Q') = \sum_{i=0}^{n} a_i - \sum_{i=n-3}^{n} w_k$.*
3. *Set $r' = \lfloor R'(Q') \cdot \beta^{-(n-1)} \rfloor$ and $b_{n-1} = \lfloor B \cdot \beta^{-(n-1)} \rfloor$.*
4. *Set $\delta_h = \lfloor (r'+2)/b_{n-1} \rfloor$ and $\delta_l = \lfloor r'/(b_{n-1}+1) \rfloor$.*
5. *If $\delta_h = \delta_l$ return $Q = Q' + \delta_h$, otherwise declare the computation as failed.*

Our algorithmic description of phase 2 is now complete. What remains is to give a good estimate for the probability of failure to show the power of the method.

3.4 Probabilistic quotient correction: The analysis

Lemma 9. *We suppose that for every fixed divisor B every possible dividend A is equally likely. Then the probability for the case $\delta_l < \delta_h$ is bounded by*

$$prob(\delta_l < \delta_h) \leq 2^{-23}.$$

Proof: We split the proof of our lemma into several claims. The key quantity in our proof is r_{n-1}, the highest digit of the exact remainder R of the division $\lfloor A/B \rfloor$. Since $R < B$ we have $0 \leq r_{n-1} \leq b_{n-1}$.

Claim 3. *Every value $0 \leq r_{n-1} < b_{n-1}$ is equally likely for r_{n-1}. Only the value $r_{n-1} = b_{n-1}$ is less likely than the others. In particular, each value $r_{n-1} \in [0, \ldots, b_{n-1}-1]$ has a probability p with $p < 1/b_{n-1}$.*

Proof of claim: Recall that we assumed $0 < B < \beta^n$ and $0 \leq A < \beta^n B$. For every possible remainder R, $0 \leq R < B$ there are exactly β^n values of A that generate this R, one for each quotient Q, $0 \leq Q < \beta^n$. Hence every remainder R is equally likely. Furthermore, to every $0 \leq r_{n-1} < b_{n-1}$ correspond exactly β^{n-1} possible values of R having first digit r_{n-1}, namely all $R = r_{n-1}\beta^{n-1} + z$ with $0 \leq z < \beta^{n-1}$. Only for $r_{n-1} = b_{n-1}$ there are less remainders that generate the first digit b_{n-1}. These are of the form $R = b_{n-1}\beta^{n-1} + z$ with $R + z < B$, which is equivalent to

$$R - r_{n-1}\beta^{n-1} = z < \sum_{i=0}^{n-2} b_i\beta^i < \beta^{n-1}.$$

Hence in this case there are less than β^{n-1} possible values for z and, consequently, for R. Moreover, since the values $r_{n-1} < b_{n-1}$ are all equally likely we have

$$1 > \sum_{x=0}^{b_{n-1}-1} p(r_{n-1} = x) = p \cdot b_{n-1}.$$

which implies the desired bound on p. This proves our claim. □

Claim 4. $\delta_h - \delta_l \in \{0, 1\}$.

Proof of claim: We set

$$\begin{array}{rclcl} r^+ & := & (r' + 2) - \delta_l b_{n-1} & = & Q(\delta_l, r' + 2, b_{n-1}) \\ r^- & := & r' - \delta_l(b_{n-1} + 1) & = & Q(\delta_l, r', b_{n-1} + 1). \end{array}$$

Clearly, we have $r^+ - r^- = 2 + \delta_l \leq 2 + \delta(Q')$. Furthermore, since $r^+ = Q(\delta_l, r' + 2, b_{n-1})$ it follows by lemma 5 that

$$\lfloor r^+/b_{n-1} \rfloor = \delta_h - \delta_l. \tag{9}$$

Since $\delta_l = \lfloor r'/(b_{n-1} + 1) \rfloor$ we have $r^- = Q(\delta_l, r', b_{n-1} + 1) \in [0, \ldots, b_{n+1}]$. Together with Equation 8 and the assumption $b_{n-1} \geq \beta/2 \geq 2^{31}$, this gives

$$r^+ \leq r^- + (2 + \delta(Q')) \leq b_{n-1} + (2 + \delta(Q')) \leq b_{n-1} + 66 < 2b_{n-1}.$$

In view of Equation 9, this implies that $\delta_h - \delta_l < 2$, proving our claim. □

Claim 5. *If $\delta(Q') = \delta_l < \delta_h$, then $r^- \leq r_{n-1} \leq r^+$.*

Proof of claim: The two estimates

$$\begin{array}{rcll} r^-\beta^{n-1} & = & (r' - \delta_l(b_{n-1} + 1))\beta^{n-1} & \text{by definition of } r^- \\ & \leq & R(Q') - \delta_l(b_{n-1} + 1)\beta^{n-1} & \text{by left part of equation 6} \\ & \leq & R(Q') - \delta_l B & \end{array}$$

and

$$\begin{array}{rcll} r^+\beta^{n-1} & = & (r' + 2 - \delta_l b_{n-1})\beta^{n-1} & \text{by definition of } r^+ \\ & \geq & R(Q') - \delta_l b_{n-1}\beta^{n-1} & \text{by right part of equation 6} \\ & \geq & R(Q') - \delta_l B & \end{array}$$

give us an upper bound and a lower bound on $R(Q') - \delta_l B$. Writing the estimates together into one, we get

$$r^-\beta^{n-1} \leq R(Q') - \delta_l B \leq r^+\beta^{n-1}. \tag{10}$$

Because $\delta_l = \delta(Q')$, we have $R = R(Q') - \delta(Q')B = R(Q') - \delta_l B$ and so Equation 10 reads $r^- \beta^{n-1} \leq R \leq r^+ \beta^{n-1}$. Our claim follows by dividing this equation by β^{n-1}, because $\lfloor R/\beta^{n-1} \rfloor = r_{n-1}$. □

Claim 6. *If $\delta_l < \delta(Q') = \delta_h$, $r^- - 1 \leq r_{n-1} + b_{n-1} \leq r^+$.*

Proof of claim: Because $\delta_l = \delta(Q') - 1$, we have

$$R(Q') - \delta_l B = R(Q') - \delta(Q')B + B = R(Q) + B.$$

Equation 10 reads now $r^- \beta^{n-1} \leq R + B \leq r^+ \beta^{n-1}$. Again we divide this equation by β^{n-1}. Because

$$\lfloor (R + B)/\beta^{n-1} \rfloor \geq \lfloor R/\beta^{n-1} \rfloor + \lfloor B/\beta^{n-1} \rfloor = r_{n-1} + b_{n-1},$$

the upper bound of our claim follows. Similarly, the lower bound follows because

$$\lfloor (R + B)/\beta^{n-1} \rfloor \leq \lfloor R/\beta^{n-1} \rfloor + \lfloor B/\beta^{n-1} \rfloor + 1 = r_{n-1} + b_{n-1} + 1. \quad \square$$

Claim 7. *$\delta_l < \delta_h$ is only possible if either*

1. *$\delta(Q') = \delta_h$ and $0 \leq r_{n-1} \leq 2 + \delta(Q')$*
2. *$\delta(Q') = \delta_l$ and $b_{n-1} - (2 + \delta(Q')) \leq r_{n-1} \leq b_{n-1}$.*

Proof of claim: First note that because $\delta_l < \delta_h$ we have $r^+ \geq b_{n-1}$. This follows by the representation of r^+ as $r^+ = Q(\delta_l, r' + 2, b_{n-1})$. If $\delta(Q') = \delta_l$, it follows by Claim 5 that

$$r_{n-1} \geq r^- \geq r^+ - (2 + \delta(Q')) \geq b_{n-1} - (2 + \delta(Q'))$$

as desired. Next, if $\delta(Q') = \delta_h = \delta_l + 1$ it follows by Claim 6 that

$$r_{n-1} + b_{n-1} \leq r^+ \leq r^- + (2 + \delta(Q')) \leq b_{n-1} + (2 + \delta(Q')).$$

From this we get $r_{n-1} \leq 2 + \delta(Q')$. □

Claim 8. *Claims 7 and 3 prove Lemma 9.*

Proof of claim and lemma: Of all $1 + b_{n-1}$ possible values of r_{n-1}, only $2\delta(Q') + 6 \leq 2 \cdot 66 + 6 < 2^8$ values cause phase 2 to fail by Claim 7. By Claim 3, the probability for each value is at most $1/b_{n-1}$. Hence we find that

$$prob(\delta_l < \delta_h) \leq 2^8/b_{n-1} \leq 2^8/2^{31} = 2^{-23}.$$

This concludes the proof of Lemma 9. □

4 Other division methods

In this section we look at other subquadratic methods for division that might be applicable for large numbers in practice. Neglecting the Fast Fourier Transformation because of its huge constant overhead we consider Karatsuba's method as the fastest multiplication in practice for numbers with reasonable length. With this assumption we develop another fast algorithm for division by combining three well known methods (Barrett's method and inverses computed by Newtonian iteration combined with Karatsuba multiplication) and argue that recursive division is still superior even if we assume that an actual implementation of the combined methods uses all known tricks and shortcuts. So to speak, we take the best ingredients known so far to make a mix of algorithms for division.

4.1 Barrett's method

Barrett's method is known in applied cryptography as an algorithm for modular reduction. Here the goal is to reduce a number A modulo another number M, which means that only the remainder of the quotient $\lfloor A/M \rfloor$ is important but not the quotient itself. As it turns out, modular reduction is no easier than division with remainder. However, Barrett's method profits much from the fact that in some applications many modular reductions are done modulo the same number, in which case it is advisable to precompute the (shifted) inverse of M up to a certain precision. Let us sketch Barrett's algorithm as found in [3] in our notation.

Algorithm 7. *(Barrett division) Let A and M be nonnegative integers such that $\beta^{n-1} \leq M < \beta^n$ and $A < \beta^{2n}$. The algorithm returns the quotient $Q = \lfloor A/M \rfloor$ with remainder R.*

1. *Precompute the shifted inverse of M as $\mu = \lfloor \beta^{2n}/M \rfloor < \beta^{n+1}$.*
2. *Set $A_1 = \lfloor A \cdot \beta^{-(n-1)} \rfloor$.*
3. *Set $\widehat{Q} = \lfloor A_1 \cdot \mu \cdot \beta^{-(n+1)} \rfloor$.*
4. *Set $R(\widehat{Q}) = A - M \cdot \widehat{Q}$.*
5. *Compute $\delta(\widehat{Q}) = \lfloor R(\widehat{Q})/M \rfloor$ by school division and the overall remainder $R = R(\widehat{Q}) - M\delta(\widehat{Q})$.*
6. *Return R and $Q = \widehat{Q} + \delta(\widehat{Q})$.*

This formulation of Barrett's algorithm requires a few explanations. First, the precomputation of μ is not strictly a part of the algorithm; we will discuss this step in the next section. Next, if we omitted the downwards-rounding in steps 1–3 and computed every operation exactly, the resulting quotient would be exact. We omit the error analysis here; details can be found in [1]. It suffices to know that only the least significant digit of $\widehat{Q}$ is not exact, so we find that $\delta(\widehat{Q})$ is computable in linear time by school division, see Lemma 5.

The running time of Barrett's algorithm depends on the chosen multiplication method. There are only two multiplications that are not mere digit–shifts, namely the computation of $A_1 \cdot \mu$ in step 3 and the computation of $M \cdot \widehat{Q}$ in step 4. All operations beside the precomputation of μ can be done in linear time. We conclude that Barrett's algorithm has running time $2M(n)$. Used with Karatsuba multiplication, Barrett's algorithm has the same running time as $D_{2n/1n}$. In other words, Barrett's algorithm is asymptotically no faster than our recursive division *even if the precomputation of the divisor's inverse is neglected.* This statement is true even if special versions of Karatsuba multiplication are used that are fine-tuned for the particular situation inside Barrett's algorithm. The key for the speed-up of the multiplication in step 3 is that here we only need the $n+1$ most significant digits of the result. Similarly, in step 4 we only need (roughly) the $n+1$ least significant digits of the result $M \cdot \widehat{Q}$, because the higher $n+1$ digits are known to be equal to those of A by the error analysis. Unfortunately, the advantage we get from using these special multiplication routines over normal Karatsuba tends to zero as n increases; for details see [6].

It should be remarked that in cryptography Barrett's algorithm is currently used with special versions of school multiplication. Then the number of digit multiplications for *both* products in Barrett's algorithm together is $n^2 + O(n)$, the same number of steps that are needed for *one* ordinary school multiplication of n-digit numbers. This classical implementation of Barrett's algorithm is about as fast as school division, but it avoids division operations altogether [8].

4.2 Inverses by Newtonian iteration

Given a real number $v \in (1/2, 1]$ and an appropriate start value for x, the iteration

$$x \leftarrow 2 \cdot x - v \cdot x^2$$

converges to the inverse $1/v$. Knuth [5] describes an implementation of this iteration that can be easily adapted to compute the integral shifted inverse $\lfloor \beta^k / v \rfloor$ for any value of k. Note however that the inverse here is not computed with exact rounding to the next smaller integer; the algorithm only guarantees a precision of k digits for its result. Knuth's implementation is made efficient by starting with a small precision of x and doubling the precision after each iteration. It can be shown that the time to find n digits of $1/v$ by this method using Karatsuba multiplication is $2K(n)$. This is still not the best one can do; half of the multiplications actually compute squares, which is considerably simpler than performing ordinary multiplications. Seeking for the best performance one can implement the squaring by a straightforward variant of Karatsuba multiplication, whose running time we denote by $K_{sq}(n)$. One should expect $1/2K(n) < K_{sq}(n) < K(n)$ in practice. There are more tricks one could use to speed up the computation, too difficult for us to describe in this paper. However, the running time of the *last step* of the above iteration is certainly a lower bound for the running time of any implementation that uses this iteration. The cost for the last iteration step is $K(n) + K_{sq}(n/2)$, since the old iteration value x that we square has only $n/2$ digits. Together with our analysis of Barrett's algorithm from the last section we consider $3K(n) + K_{sq}(n/2)$ as a lower bound for the running time of Barrett division, including the time for computing the inverse by Newtonian iteration. This shows that the recursive division algorithm $D_{2n/1n}$ is clearly superior. Only for applications where the cost for the inversion is negligible, both methods of division are about equally fast.

4.3 The fastest known method

If we use the fastest known multiplication method by Schönhage and Strassen, which has an asymptotic running time of $O(n \log n \log \log n)$, the sum for the running time of Algorithm $D_{2n/1n}$ evaluates to $O(n \log^2 n \log \log n)$. This is *not* the fastest known division time (which is achieved by combining Schönhage's method with Newtonian iteration, yielding a running time asymptotically equal to the multiplication time, see [5]). Unfortunately this result only pays for huge numbers because the constants involved in this methods are very large.

5 Concrete running times

Our testing environment was a SPARC Ultra-1. On this machine we have $\beta = 2^{32}$, so a digit length of y corresponds to a decimal number with $y \cdot 32 / \log_2 10 \approx 9.63 \cdot y$ digits. We have used and modified LEDAs `integer` class for our experiments. During the implementation phase we substantially speeded up this integer library. At the time of this writing we know of no other integer package faster than ours on a SPARC architecture. The code including complete documentation is found in [9]. Basically we have two versions of our library. One version uses only portable C/C++ code whereas the other version has a few additional hand-tuned assembler routines for the innermost arithmetic loops .

5.1 Portable C/C++ code

Figures 5, 6, 7, 8, and 9 show experimental running times measured with portable C/C++ code. The tests consisted of dividing a $2n$-digit number by an n-digit number 1000 times. From Figure 5 we see that the break-even point is about 26 digits, i.e., `FastDivision` pays for numbers with more than 250 decimal digits or 832 bits. Figure 6 shows the speedup we gain as the numbers get larger and larger. Figure 7 shows the running times for multiplying two n-digit numbers with Karatsuba's method compared to `FastDivision`. Figure 9 shows the quotient of the running times of Figure 7. It confirms the theoretical running time of $2K(n)+O(n \log n)$ for `FastDivision`. Figure 8 shows how the running time of `FastDivision` varies with different lengths of the two input numbers.

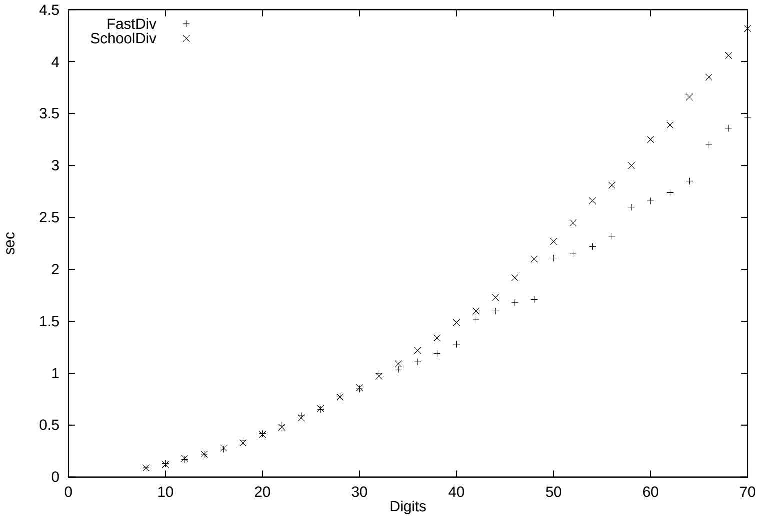

Figure 5: Comparing FastDivision to School Division in a small range

5.2 Hand-coded assembler routines

Using handwritten assembler code for the innermost loops lowers the running times significantly, but also moves up the break-even point of `FastDivision` with respect to school division to about 64 digits. In this context it is more useful to compare `FastDivision` to other integer packages. These other packages were CLN [12]), libI [11], and LIP [13] available to us in the LiDIA library [15]. Figure 10 shows that for numbers less than 100 machine digits there is no significant difference in the running times compared to CLN (which is the fastest public domain library for integer arithmetic beside ours at the time of this writing). These figures show the average running time per division. However, Figure 11 shows that we are much faster as the digits become larger. We expect that we can lower our running times even more if we have more time to fine-tune our assembler code.

5.3 Fast modular multiplication

The effectiveness of public key cryptosystems depends heavily on the ability to quickly evaluate expressions of the form x^e *mod* m, where x is the *message* to be encrypted or decrypted

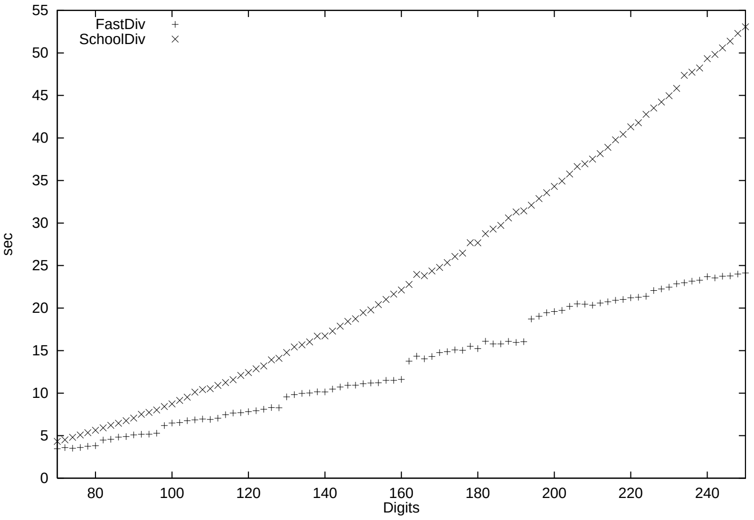

Figure 6: Comparing FastDivision to School Division in a big range

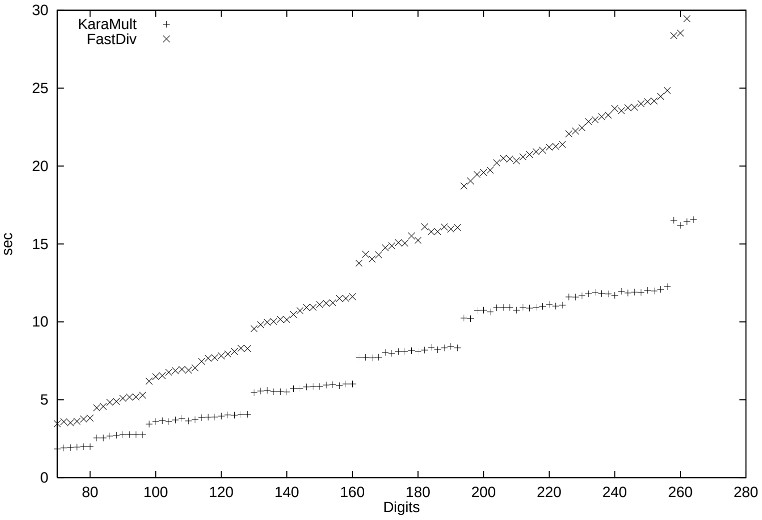

Figure 7: Comparing FastDivison with Karatsuba multiplication

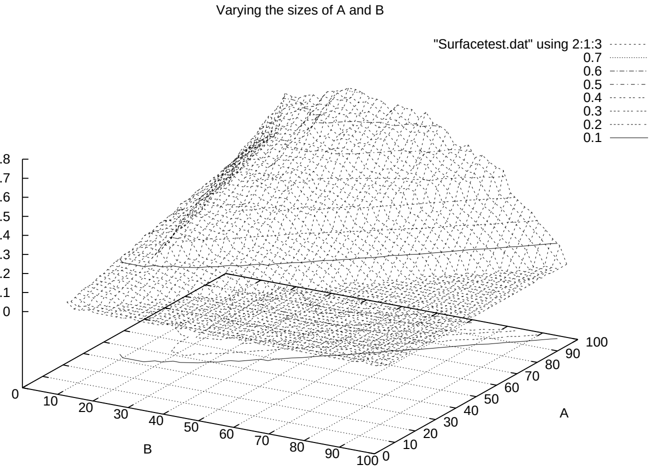

Figure 8: Varying the input sizes of A and B

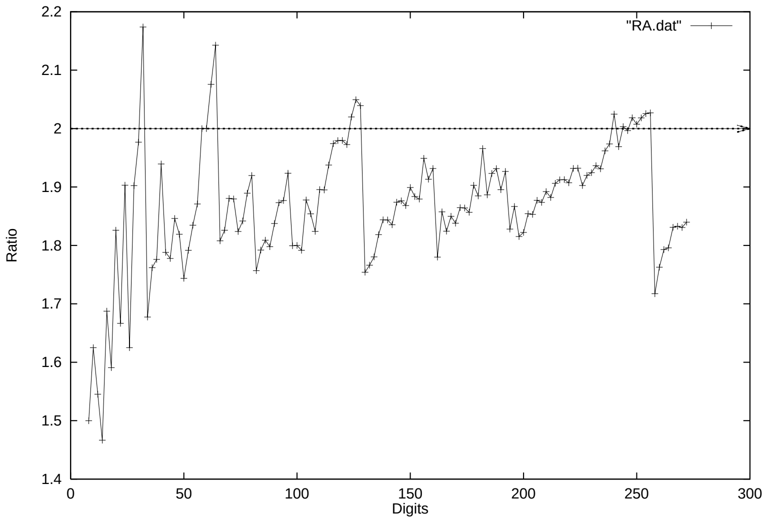

Figure 9: The ratio between FastDiv and Karatsuba multiplication is less than 2

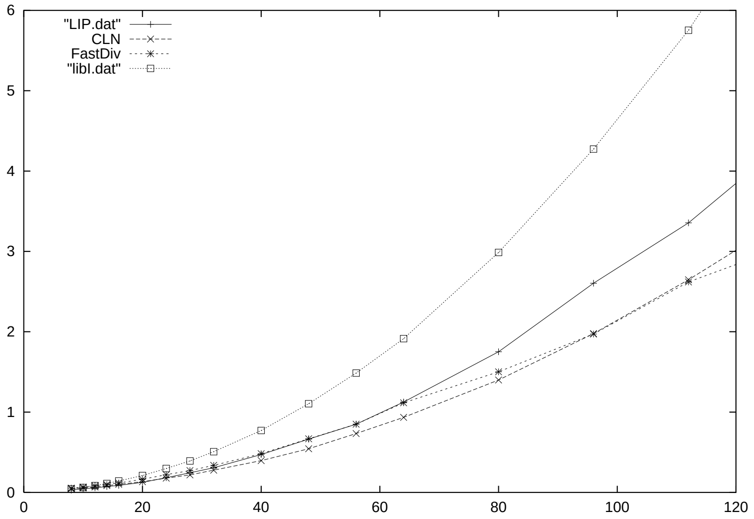

Figure 10: Comparing FastDivision to other packages in a small range

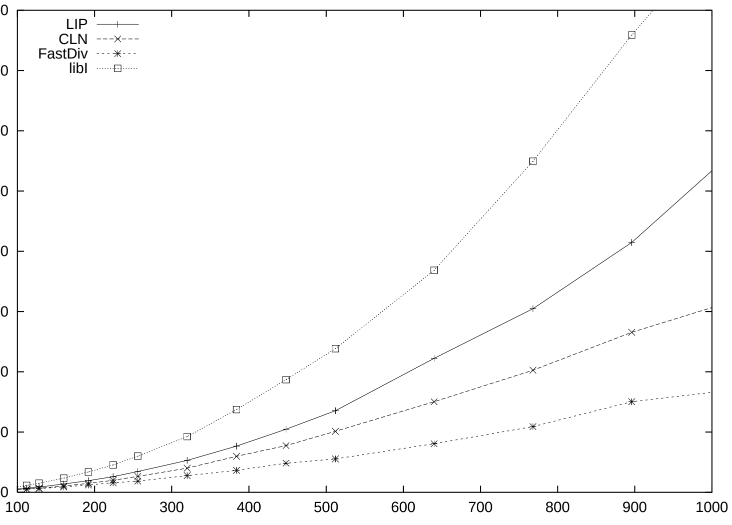

Figure 11: Comparing FastDivision to other packages in a big range

and the pair (e, m) is the *key*. The basis of these *modular exponentiation* algorithms is the computation of the expression $x \cdot y$ *mod* m, where x, y, and m are n-digit numbers.

There are three methods in use for this *modular multiplication*, ordinary school method (which multiplies x and y as integers and performs an integer division afterwards), Montgomery multiplication, and Barrett reduction (see [8] and section 4). They all need $O(n^2)$ operations on single digits and perform equally well in concrete implementations [8]. The exact numbers of steps of the school method is $2n^2 + O(n)$. It is easy to write a function `FastModMult` that performs this task asymptotically faster by computing $x \cdot y$ with Karatsuba multiplication and reducing modulo m by `FastDivision` afterwards.

Theorem 3. *The expression $x \cdot y$ mod m where x, y, and m are n-digit numbers can be computed in $3K(n) + O(n \log n)$ steps.*

Figure 12 shows experimental running times measured with portable C/C++ code. The break-even point is about 30 digits or 960 bits, so we are well in the cryptographic range. This can be used to substantially speedup public key cryptosystems in the near future when secure cryptosystems are assumed to have 1024 bits as their minimal key length.

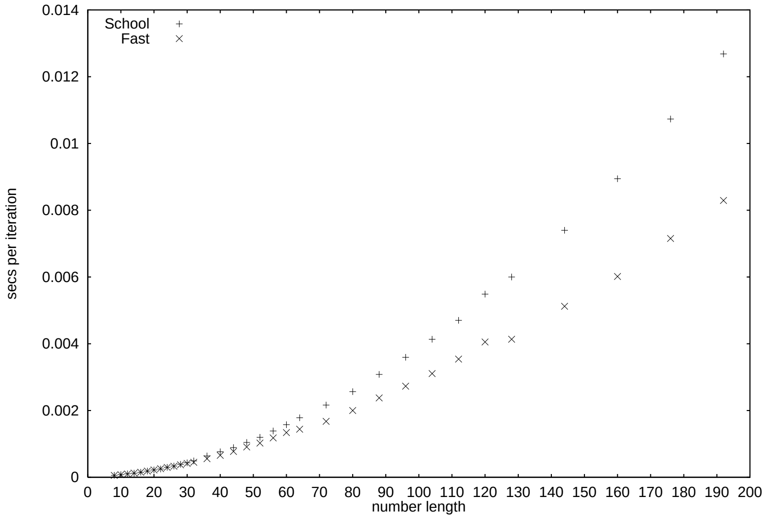

Figure 12: Comparing school modular multiplication with fast modular multiplication

6 Conclusion

We have presented a new recursive division algorithm that takes the time of about two Karatsuba multiplications. It is clearly superior in practice to all other known methods for numbers with reasonable length. The algorithm has already been implemented and performs very well in practice. It can be used to significantly speed up the time for modular exponentiation, which is the most important operation in current crypto-systems. We have shown further that it is possible to save the time of half a Karatsuba multiplication on the average if the computation of the division remainder is not required.

References

[1] P.D. Barrett. Implementing the Rivest, Shamir, and Adleman public key encryption algorithm on a standard digital signal processor. In *Advances in Cryptology, Proc. Crypto 86, LNCS 263*, pages 311–323.

[2] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving square roots. *Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[3] A.J. Menezes, P.C. v. Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[4] T. Jebelean. Practical integer division with Karatsuba complexity. In *Proc. Symp. on Symbolic and Algebraic Computation, ISSAC*, pages 339–341, 1997.

[5] D. Knuth: The Art Of Computer Programming, Vol. 2, 3rd Edition, Section 4.3

[6] W. Krandick and J.R. Johnson. Efficient multiprecision floating point multiplication with exact rounding. Technical Report 93-76, RISC-Linz, Johannes Kepler University, 1993.

[7] K. Mehlhorn, S. Näher: LEDA, A Platform for Combinatorial and Geometric Computing, Cambridge University Press, 1998

[8] A. Bosselaers, R. Govaerts, J. Vandewalle: Comparison of three modular reduction functions, Advances in Cryptology–Crypto 96 (LNCS 1109), 298–3123, 1996

[9] C. Burnikel, S. Näher, C. Uhrig, J. Ziegler: The design and implementation of an arbitrary precision integer library, Technical Report, to appear. Draft version available from `http://www.mpi-sb.mpg.de/~ziegler/bigint.ps`

[10] A. Karatsuba, Yu. Ofman: Multiplication of multidigit numbers on automata, Soviet Physics – Doklady, 7 (1963), 595-596

[11] R. Dentzer: libI: Eine lange ganzzahlige Arithmetik, Universität Heidelberg 1991

[12] B. Haible: CLN, a class library for numbers. Available via anonymous FTP from `ftp://ftp.santafe.edu/pub/gnu/cln.tar.gz` (1996)

[13] A. Lenstra: lip: long integer package, Bellcore 1989

[14] Th. Papanikolaou: Entwurf und Entwicklung einer objektorientierten Bibliothek für algorithmische Zahlentheorie, PhD Thesis, Universität des Saarlandes, 1997

[15] I. Biehl, J. Buchmann, Th. Papanikolaou, LiDIA: a library for computational number theory, Universität des Saarlandes, SFB 124 report, 1995

INFORMATIK

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Birgit Hofmann
Im Stadtwald
D-66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid E-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-1-021	S. Albers, G. Schmidt	Scheduling with Unexpected Machine Breakdowns
MPI-I-98-1-020	C. Rüb	On Wallace's Method for the Generation of Normal Variates
MPI-I-98-1-019		2nd Workshop on Algorithm Engineering WAE '98 - Proceedings
MPI-I-98-1-018	D. Dubhashi, D. Ranjan	On Positive Influence and Negative Dependence
MPI-I-98-1-017	A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, E. Ramos	Randomized External-Memory Algorithms for Some Geometric Problems
MPI-I-98-1-016	P. Krysta, K. Loryś	New Approximation Algorithms for the Achromatic Number
MPI-I-98-1-015	M.R. Henzinger, S. Leonardi	Scheduling Multicasts on Unit-Capacity Trees and Meshes
MPI-I-98-1-014	U. Meyer, J.F. Sibeyn	Time-Independent Gossiping on Full-Port Tori
MPI-I-98-1-013	G.W. Klau, P. Mutzel	Quasi-Orthogonal Drawing of Planar Graphs
MPI-I-98-1-012	S. Mahajan, E.A. Ramos, K.V. Subrahmanyam	Solving some discrepancy problems in NC*
MPI-I-98-1-011	G.N. Frederickson, R. Solis-Oba	Robustness analysis in combinatorial optimization
MPI-I-98-1-010	R. Solis-Oba	2-Approximation algorithm for finding a spanning tree with maximum number of leaves

MPI-I-98-1-009	D. Frigioni, A. Marchetti-Spaccamela, U. Nanni	Fully dynamic shortest paths and negative cycle detection on diagraphs with Arbitrary Arc Weights
MPI-I-98-1-008	M. Jünger, S. Leipert, P. Mutzel	A Note on Computing a Maximal Planar Subgraph using PQ-Trees
MPI-I-98-1-007	A. Fabri, G. Giezeman, L. Kettner, S. Schirra, S. Sch'onherr	On the Design of CGAL, the Computational Geometry Algorithms Library
MPI-I-98-1-006	K. Jansen	A new characterization for parity graphs and a coloring problem with costs
MPI-I-98-1-005	K. Jansen	The mutual exclusion scheduling problem for permutation and comparability graphs
MPI-I-98-1-004	S. Schirra	Robustness and Precision Issues in Geometric Computation
MPI-I-98-1-003	S. Schirra	Parameterized Implementations of Classical Planar Convex Hull Algorithms and Extreme Point Compuations
MPI-I-98-1-002	G.S. Brodal, M.C. Pinotti	Comparator Networks for Binary Heap Construction
MPI-I-98-1-001	T. Hagerup	Simpler and Faster Static AC^0 Dictionaries
MPI-I-97-2-012	L. Bachmair, H. Ganzinger, A. Voronkov	Elimination of Equality via Transformation with Ordering Constraints
MPI-I-97-2-011	L. Bachmair, H. Ganzinger	Strict Basic Superposition and Chaining
MPI-I-97-2-010	S. Vorobyov, A. Voronkov	Complexity of Nonrecursive Logic Programs with Complex Values
MPI-I-97-2-009	A. Bockmayr, F. Eisenbrand	On the Chvátal Rank of Polytopes in the 0/1 Cube
MPI-I-97-2-008	A. Bockmayr, T. Kasper	A Unifying Framework for Integer and Finite Domain Constraint Programming
MPI-I-97-2-007	P. Blackburn, M. Tzakova	Two Hybrid Logics
MPI-I-97-2-006	S. Vorobyov	Third-order matching in $\lambda \rightarrow$-Curry is undecidable
MPI-I-97-2-005	L. Bachmair, H. Ganzinger	A Theory of Resolution
MPI-I-97-2-004	W. Charatonik, A. Podelski	Solving set constraints for greatest models
MPI-I-97-2-003	U. Hustadt, R.A. Schmidt	On evaluating decision procedures for modal logic
MPI-I-97-2-002	R.A. Schmidt	Resolution is a decision procedure for many propositional modal logics
MPI-I-97-2-001	D.A. Basin, S. Matthews, L. Viganò	Labelled modal logics: quantifiers
MPI-I-97-1-028	M. Lermen, K. Reinert	The Practical Use of the $\mathcal{A}^*$ Algorithm for Exact Multiple Sequence Alignment
MPI-I-97-1-027	N. Garg, G. Konjevod, R. Ravi	A polylogarithmic approximation algorithm for group Steiner tree problem
MPI-I-97-1-026	A. Fiat, S. Leonardi	On-line Network Routing - A Survey
MPI-I-97-1-025	N. Garg, J. Könemann	Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems
MPI-I-97-1-024	S. Albers, N. Garg, S. Leonardi	Minimizing Stall Time in Single and Parallel Disk Systems
MPI-I-97-1-023	S.A. Leonardi, A.P. Marchetti-Spaccamela	Randomized on-line call control revisited
MPI-I-97-1-022	E. Althaus, K. Mehlhorn	Maximum Network Flow with Floating Point Arithmetic
MPI-I-97-1-021	J.F. Sibeyn	From Parallel to External List Ranking
MPI-I-97-1-020	G.S. Brodal	Finger Search Trees with Constant Insertion Time
MPI-I-97-1-019	D. Alberts, C. Gutwenger, P. Mutzel, S. Näher	AGD-Library: A Library of Algorithms for Graph Drawing
MPI-I-97-1-018	R. Fleischer	On the Bahncard Problem