Information Processing Letters 114 (2014) 152–157

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Efficient multiple-precision integer division algorithm

Debapriyay Mukhopadhyay [a], Subhas C. Nandy [b,*]

[a] IXIA Technologies Pvt. Ltd., Kolkata 700091, India
[b] Indian Statistical Institute, Kolkata 700108, India

ARTICLE INFO

Article history:
Received 16 February 2013
Received in revised form 3 October 2013
Accepted 19 October 2013
Available online 25 October 2013
Communicated by Jinhui Xu

Keywords:
Division algorithm
Normalization
Computational arithmetic
Cryptography

ABSTRACT

Design and implementation of division algorithm is one of the most complicated problems in multi-precision arithmetic. Huang et al. [1] proposed an efficient multi-precision integer division algorithm, and experimentally showed that it is about three times faster than the most popular algorithms proposed by Knuth [2] and Smith [3]. This paper reports a bug in the algorithm of Huang et al. [1], and suggests the necessary corrections. The theoretical correctness proof of the proposed algorithm is also given. The resulting algorithm remains as fast as that of [1].

1. Introduction

Arithmetic operations on large integers are often used in cryptographic algorithms. The usual arithmetic operations are performed in the machine using the built-in functions. Each machine has a base B in its number system, and can store unsigned integers of values $\{0, 1, 2, \ldots, B-1\}$ in built-in integer locations for that machine. The time complexity of a single usual arithmetic operation is assumed to be $O(1)$.

A large integer cannot be stored in machine dependent built-in size for integers, and arithmetic operations on such integer(s) cannot be performed using the built-in routines for those arithmetic operations. The operations on large integers are called multi-precision arithmetic operations. The multi-precision division is the hardest among all the four multi-precision arithmetic operations. Multi-precision division plays a crucial role in cryptographic research [4], and primality testing [5]. The commonly used multi-precision division algorithm is proposed by Knuth [2].

Normalization is one of the key steps of multi-precision division, and it is defined as the act of restoring the individual digits or words in the range $[0, B-1]$. Since each word or digit of the quotient is guessed in each step of the division, so it is difficult to skip normalization. The division algorithm proposed by Smith [3] reduces the intermediate normalization steps. Huang et al. [1] proposed an efficient algorithm for multi-precision integer division that reduces the number of normalizations to a single normalization step. The uniqueness of the algorithm is that, if it is applied for long integer division, then both the quotient and remainder simultaneously gets calculated at the end. There is no need for any correction step or any extra multiplication or subtraction for computing the remainder. It is experimentally shown that the algorithm in [1] is three times faster than the algorithm by Knuth [2].

We have identified a bug in the algorithm by Huang et al. [1], and propose the necessary corrections. We theoretically justify the correctness of our algorithm. The detailed experiment justifies that our corrected version of the algorithm runs with the same efficiency as suggested in [1].

The paper is organized as follows. Section 2 briefly describes the algorithm of Huang et al. [1]. In Section 3, we report the bug by providing an example. We describe the corrected algorithm in Section 4, and also provide the correctness proof. Finally, the concluding remarks appear in Section 5.

* Corresponding author.
E-mail addresses: debapriyaym@gmail.com (D. Mukhopadhyay), nandysc@isical.ac.in (S.C. Nandy).

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2013.10.005

2. Overview of the algorithm of [1]

Let B be the base of the multi-precision integers under consideration. For two multi-precision positive integers a and b $(a > b > 0)$, we need to find out multi-precision integers q and r such that $a = bq + r$. Let m and n denote the number of words required to store a and b respectively. Thus $a = a_1 + a_2.B + \cdots + a_m.B^{m-1}$ and $b = b_1 + b_2.B + \cdots + b_n.B^{n-1}$. Algorithm starts by copying a into a work array W of size $m+1$, whose each element can hold integers that require no more than 4 bytes, i.e., it ranges from $-2\,147\,483\,648$ to $2\,147\,483\,647$ in decimal number system. We use a work array W for the division, and use a variable MAX that contains the integer $2\,147\,483\,648$. A zero digit is put at $W[0]$; then the digits of a are stored in successive locations starting from the most significant digit a_m at $W[1]$, followed by a_{m-1} at $W[2]$ and so on. In this way, a_1 will be stored at $W[m]$. Similarly, the digits $b_1, b_2, \ldots, b_n$ are stored in $b[n-1], b[n-2], \ldots, b[0]$ of the array b. The work array W is updated during the iteration process and there are chances of overflow in some elements of W during the execution. So, normalization is required to restore the digits in the range $[0, B-1]$. At the end of the algorithm the least significant n elements of W gives the remainder r and the most significant $m-n+1$ elements correspond to the quotient q. Algorithm 1 gives an intuitive description of the algorithm of [1] assuming that the working register and the arithmetic circuit for division of the computer can handle the three digited numbers with base B. For detailed description of this algorithm, see [1].

Algorithm 1. Multi-precision_Division.

1. (* Form the denominator of division using first two consecutive elements of b *)
   Compute $D = b[0] * B + b[1]$
2. **for** $i = 0, 1, 2, \ldots, m-n$ **do** (* i indicates the iteration number. *)
3. (* Form the numerator using three consecutive elements $W[i]$, $W[i+1]$ and $W[i+2]$ *)
   Compute $N = W[i] * B^2 + W[i+1] * B + W[i+2]$
   (* Assume that N can store a number less than B^4. *)
4. Compute the quotient $Q = \lfloor \frac{N}{D} \rfloor$
5. (* Update the array W *)
   for $j = 1$ to n **do**
   $W[i+j] = W[i+j] - Q.b[j-1]$
   endfor
6. $W[i+1] = W[i] * B + W[i+1]$
7. $W[i] = Q$ (* Put Q in $W[i]$ *)
8. **endfor**
9. Normalize W
10. Report the quotient and remainder from the array W.

Now, we describe the normalization procedure as proposed in [1]. Here the objective is to consider each word of the quotient and remainder, and restore its value in the range $[0, B-1]$. The main idea of the normalization procedure centers around finding a multiplicative factor c such that when $c * B$ is added with the content of an element in the array W, it will lie in the range $[0, B-1]$. For a negative word, the algorithm finds a positive c, and for a word greater than B, the algorithm finds a negative c, to appropriately adjust the word. If a word $W[i]$ gets adjusted, then its immediate next higher word $W[i-1]$ needs to be adjusted so that the value remains unchanged. Next, $W[i-1]$ is considered for normalization. The normalization procedure described in [1] is stated below.

Algorithm 2. Normalize (X).

1. **for** $i = m, m-1, m-2, \ldots, 1$ **do** (* i indicates the iteration number *)
2. $c = 0$
3. **if** $(X[i] < 0)$ **then** $c = \lfloor (-X[i]-1)/B \rfloor + 1$
4. **else if** $(X[i] \geqslant B)$ **then** $c = -\lfloor X[i]/B \rfloor$
5. **end if**
6. $X[i] = X[i] + c * B$
7. $X[i-1] = X[i-1] - c$
8. **endfor**

We now describe two interesting properties of the normalization procedure of [1]. Let X be an array containing a multi-precision number obtained by the **Multi-precision_Division** algorithm prior to the normalization (i.e., the array W after Step 8). Thus, $X = (X[l-1] + X[l-2].B + \cdots + X[0].B^{l-1})$, where l is the number of words in X. We now define

$$Val_{l-i} = \sum_{j=1}^{i} X[l-j].B^{j-1}, \quad \text{for } i = 1, 2, \ldots, l. \tag{1}$$

Observe that, $Val_0 = X$. It needs to be mentioned that Val_i's, for $i = 0, 1, 2, \ldots, l-1$, need not be computed/stored in the algorithm. This only helps in characterizing which indices of the array X requires normalization.

Property 1. *For each $i = l-1, l-2, \ldots, 1$, if either $Val_i < 0$ or $Val_i \geqslant B^i$, then the normalization is required for $X[i]$. The normalization factor c is positive if $Val_i < 0$ and negative if $Val_i \geqslant B^i$.*

Property 2. *For any value of $i \in \{1, 2, \ldots, l-1\}$, if the normalization is not required for $X[i]$, then Val_i remains unaltered after the normalization. Since normalization algorithm described above doesn't normalize the location $X[0]$, $Val_0 (= X)$ remains same before and after the normalization.*

Let us demonstrate the algorithm with a small example in Table 1. Here $B = 10$, $a = 60\,541$, $b = 432$; thus $m = 5$ and $n = 3$. We show the iterations of the *for* loop of Step 2 of the algorithm **Multi-precision_Division** in Table 1 and also show the outcome of the normalization step of the algorithm of Huang et al. [1]. Note that, D remains equal to 43 throughout the execution.

After the normalization step, the higher order $m-n+1$ words form the quotient and the remaining n words form the remainder. Therefore, for the above example quotient $q = 140$ and remainder $r = 61$. But the above normalization procedure may lead to an incorrect result as described in the next section.

3. Description of the bug

In [1], the correctness of the algorithm (i.e., whether $a = b.q + r$ holds) is not established. We could identify a pathological instance to show that the algorithm of

Table 1
Demonstration of the algorithm in [1].

	$W[0]$	$W[1]$	$W[2]$	$W[3]$	$W[4]$	$W[5]$	Computed	
							N	Q
Initialization	0	6	0	5	4	1	60	1
Iteration 1	1	2	−3	3	4	1	173	4
Iteration 2	1	4	1	−9	−4	1	6	0
Iteration 3	1	4	0	1	−4	1	–	–
Normalization	1	4	0	0	6	1	–	–

Huang et al. [1] is not correct. We choose $B = 256$, $a = 94\ 6\ 142\ 2\ 78\ 236\ 223\ 88\ 169\ 92\ 10$,[1] and $b = 10\ 183\ 116\ 36\ 218\ 189$. Thus we have $m = 11$, $n = 6$. The contents of the work array at the end of each iteration is given in Table 2.

Thus, the algorithm of Huang et al. [1] produces $q = 8\ 198\ 24\ 187\ 1\ 186$, and $r = 255\ 205\ 220\ 211\ 214\ 251$. But the correct values are $q = 8\ 198\ 24\ 187\ 1\ 186$, and $r = 10\ 133\ 80\ 248\ 177\ 184$. These can also be validated by Knuth's [2] algorithm. Observe that, the relationship $a = b.q + r$ holds prior to the normalization step (Step IT-6 in the above example). The solution becomes faulty if the most significant word of the remainder requires normalization and it affects the least significant word of the quotient. Let c be the amount of change of the least significant word of the quotient during the normalization. In such a situation if the quotient is affected by an amount of c (i.e., $q' = q - c$), it is the correct quotient, but the remainder r (after normalization) is faulty. In order to satisfy the relationship among quotient, remainder, a and b, the remainder must satisfy $r' = r + b.c$. Now, we need to normalize r' so that all the words in it are within the range $[0, B-1]$.

4. Corrected algorithm

On the basis of the discussions in Section 3, the pseudocode of the revised normalization procedure is given in Algorithm 3. We will also justify the correctness of our algorithm.

Algorithm 3. Corrected Normalize (W).

```
1. Set adjust = false and X = W
2. for i = m, m−1, m−2, ..., 1 do (* i indicates the iteration number *)
3.     c = 0
4.     if (X[i] < 0) then c = ⌊(−X[i] − 1)/B⌋ + 1
5.     else if (X[i] ⩾ B) then c = −⌊X[i]/B⌋
6.     endif
7.     X[i] = X[i] + c * B
8.     X[i − 1] = X[i − 1] − c
9.     if (c ≠ 0) and (i = m − n + 1) then
10.        save_c = c
11.        adjust = true
12.    endif
13. endfor
14. if (adjust = true) then
15.    for i = m, m−1, ..., m−n+1 do
16.        W[i] = W[i] + save_c * b[i]
17.    endfor
18.    for i = m−n, m−n−1, ..., 0 do
19.        W[i] = X[i]
20.    endfor
21.    for i = m, m−1, m−2, ..., 1 do (* i indicates the iteration
           number *)
22.        c = 0
23.        if (W[i] < 0) then c = ⌊(−W[i] − 1)/B⌋ + 1
24.        else if (W[i] ⩾ B) then c = −⌊W[i]/B⌋
25.        endif
26.        W[i] = W[i] + c * B
27.        W[i − 1] = W[i − 1] − c
28.    endfor
29. else
30.    for i = 0, 1, 2, ..., m do
31.        W[i] = X[i]
32.    endfor
33. endif
```

4.1. Proof of correctness

Let $a = (a_1 + a_2.B + \cdots + a_m.B^{m-1})$ and $b = (b_1 + b_2.B + \cdots + b_n.B^{n-1}) \neq 0$ be the two arguments of the algorithm, where m and n ($m \geqslant n$) are the number of words in a and b respectively. We will prove the correctness of the algorithm for the case $m = 2.n - 1$. The proof for the other two cases $m > 2.n - 1$ and $n \leqslant m < 2.n - 1$ can be obtained similarly.

As mentioned earlier, the work array W is initialized as $W = (a_1 + a_2.B + \cdots + a_m.B^{m-1} + a_{m+1}.B^m)$, where $a_{m+1} = 0$. The algorithm consists of $m - n + 1$ iteration steps, and finally a normalization step as described in the earlier section. During the execution, the variables N and Q store the numerator and denominator of different iteration steps. We use $N^{(i)}$, $Q^{(i)}$ and $R^{(i)}$ to denote the value of the numerator, denominator and remainder respectively in the i-th iteration. By induction, we derive the form of $N^{(i)}$ and the work array W for each iteration step $i = 1, 2, \ldots, m - n + 1$ of Algorithm 1.

Iteration 1. $N^{(1)} = a_{m+1}.B^2 + a_m.B + a_{m-1} = a_m.B + a_{m-1} = Q^{(1)}.D + R^{(1)}$.

The work array W then gets updated as $W = Q^{(1)}.B^m + (a_m - Q^{(1)}.b_n).B^{m-1} + (a_{m-1} - Q^{(1)}.b_{n-1}).B^{m-2} + (a_{m-2} - Q^{(1)}.b_{n-2}).B^{m-3} + \cdots + (a_{m-n+1} - Q^{(1)}.b_1).B^{m-n} + a_{m-n}.B^{m-n-1} + \cdots + a_2.B + a_1$.

Iteration 2. $N^{(2)} = (a_m - Q^{(1)}.b_n).B^2 + (a_{m-1} - Q^{(1)}.b_{n-1}).B + (a_{m-2} - Q^{(1)}.b_{n-2}) = (a_m.B^2 + a_{m-1}.B + a_{m-2}) - Q^{(1)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) = Q^{(2)}.D + R^{(2)}$.

The work array W then gets updated as $W = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + \{(a_m - Q^{(1)}.b_n).B + (a_{m-1} - Q^{(1)}.b_{n-1} - Q^{(2)}.b_n)\}.B^{m-2} + (a_{m-2} - Q^{(1)}.b_{n-2} - Q^{(2)}.b_{n-1}).B^{m-3} + \cdots + (a_{m-n+1} - Q^{(1)}.b_1 - Q^{(2)}.b_2).B^{m-n} + (a_{m-n} -$

[1] For the ease of readability, we have separated the digits of a long integer using blanks.

Table 2
Demonstration of the bug in the algorithm of [1].

Iteration	$W[0]$	$W[1]$	$W[2]$	$W[3]$	$W[4]$	$W[5]$	$W[6]$	$W[7]$	$W[8]$	$W[9]$	$W[10]$	$W[11]$	N	Q
IT-0	0	94	6	142	2	78	236	223	88	169	92	10	24 070	8
IT-1	8	14	−1458	−786	−286	−1666	−1276	223	88	169	92	10	543 470	198
IT-2	8	198	146	−37 020	−23 254	−8794	−44 440	−37 199	88	169	92	10	67 882	24
IT-3	8	198	24	116	−27 646	−11 578	−45 304	−42 431	−4448	169	92	10	513 222	187
IT-4	8	198	24	187	180	−45 799	−66 996	−49 163	−45 214	−35 174	92	10	4940	1
IT-5	8	198	24	187	1	271	−67 179	−49 279	−45 250	−35 392	−97	10	513 153	187
IT-6	8	198	24	187	1	187	327	−83 500	−66 942	−42 124	−40 863	−35 333	–	–
IT-N	8	198	24	187	1	186	255	205	220	211	214	251	–	–

$Q^{(2)}.b_1).B^{m-n-1} + a_{m-n-1}.B^{m-n-2} + \cdots + a_2.B + a_1 = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + (R^{(1)} - Q^{(2)}.b_n).B^{m-2} + (a_{m-2} - Q^{(1)}.b_{n-2} - Q^{(2)}.b_{n-1}).B^{m-3} + \cdots + (a_{m-n+1} - Q^{(1)}.b_1 - Q^{(2)}.b_2).B^{m-n} + (a_{m-n} - Q^{(2)}.b_1).B^{m-n-1} + a_{m-n-1}.B^{m-n-2} + \cdots + a_2.B + a_1$.

Iteration 3. $N^{(3)} = (R^{(1)} - Q^{(2)}.b_n).B^2 + (a_{m-2} - Q^{(1)}.b_{n-2} - Q^{(2)}.b_{n-1}).B + (a_{m-3} - Q^{(1)}.b_{n-3} - Q^{(2)}.b_{n-2}) = (a_m.B^3 + a_{m-1}.B^2 + a_{m-2}.B + a_{m-3}) - Q^{(1)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - Q^{(2)}(b_n.B^2 + b_{n-1}.B + b_{n-2})$.

Let, $N^{(3)} = Q^{(3)}.D + R^{(3)}$. Thus, the work array W gets updated as

$W = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + Q^{(3)}.B^{m-2} + \{(R^{(1)} - Q^{(2)}.b_n).B + (a_{m-2} - Q^{(1)}.b_{n-2} - Q^{(2)}.b_{n-1} - Q^{(3)}.b_n)\}.B^{m-3} + (a_{m-3} - Q^{(1)}.b_{n-3} - Q^{(2)}.b_{n-2} - Q^{(3)}.b_{n-1}).B^{m-4} + \cdots + (a_{m-n} - Q^{(2)}.b_1 - Q^{(3)}.b_2).B^{m-n-1} + (a_{m-n-1} - Q^3.b_1).B^{m-n-2} + \cdots + a_2.B + a_1 = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + Q^{(3)}.B^{m-2} + (R^{(2)} - Q^{(3)}.b_n).B^{m-3} + (a_{m-3} - Q^{(1)}.b_{n-3} - Q^{(2)}.b_{n-2} - Q^{(3)}.b_{n-1}).B^{m-4} + \cdots + (a_{m-n} - Q^{(2)}.b_1 - Q^{(3)}.b_2).B^{m-n-1} + (a_{m-n-1} - Q^3.b_1).B^{m-n-2} + \cdots + a_2.B + a_1$.

In the $(m-n)$-th iteration, let

$N^{(m-n)} = (a_m.B^{m-n} + a_{m-1}.B^{m-n-1} + \cdots + a_n) - Q^{(m-n-1)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) - Q^{(m-n-2)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - \cdots - Q^{(2)}.(b_n.B^{m-n-1} + b_{n-1}.B^{m-n-2} + \cdots + b_3.B + b_2) - Q^{(1)}.(b_n.B^{m-n} + b_{n-1}.B^{m-n-1} + \cdots + b_2.B + b_1)$,

$W = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + Q^{(3)}.B^{m-2} + \cdots + Q^{(m-n)}.B^{n+1} + (R^{(m-n-1)} - Q^{(m-n)}.b_n).B^n + \{a_n - Q^{(m-n)}.b_{n-1} - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\}.B^{n-1} + \{a_{n-1} - Q^{(m-n)}.b_{n-2} - Q^{(m-n-1)}.b_{n-3} - \cdots - Q^{(2)}.b_1\}.B^{n-2} + \cdots + \{a_2 - Q^{(m-n)}.b_1\}.B + a_1$, and these satisfy $N^{(m-n)} = Q^{(m-n)}.D + R^{(m-n)}$.

We show the content of $N^{(m-n+1)}$ and the work array W in the $(m-n+1)$-th iteration.

Iteration $m-n+1$. Since $m = 2.n-1$, we have $m-n+1 = n$.

$N^{(m-n+1)} = (R^{(m-n-1)} - Q^{(m-n)}.b_n).B^2 + \{a_n - Q^{(m-n)}.b_{n-1} - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\}.B + \{a_{n-1} - Q^{(m-n)}.b_{n-2} - Q^{(m-n-1)}.b_{n-3} - \cdots - Q^{(2)}.b_1\} = R^{(m-n-1)}.B^2 - Q^{(m-n)}.\{b_n.B^2 + b_{n-1}.B + b_{n-2}\} - Q^{(m-n-1)}.\{b_{n-2}.B + b_{n-3}\} - Q^{(m-n-2)}.\{b_{n-3}.B + b_{n-4}\} - \cdots - Q^2.\{b_2.B + b_1\} - Q^1.b_1.B + a_n.B + a_{n-1} = \{N^{(m-n-1)} - Q^{(m-n-1)}.D\}.B^2 - Q^{(m-n)}.\{b_n.B^2 + b_{n-1}.B + b_{n-2}\} - Q^{(m-n-1)}.\{b_{n-2}.B + b_{n-3}\} - Q^{(m-n-2)}.\{b_{n-3}.B + b_{n-4}\} - \cdots - Q^2.\{b_2.B + b_1\} - Q^1.b_1.B + a_n.B + a_{n-1}$.

Assuming the similar expression of $N^{(m-n-1)}$, i.e., $N^{(m-n-1)} = (a_m.B^{m-n-1} + a_{m-1}.B^{m-n-2} + \cdots + a_{n+1}) - Q^{(m-n-2)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) - Q^{(m-n-3)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - \cdots - Q^{(2)}.(b_n.B^{m-n-2} + b_{n-1}.B^{m-n-3} + \cdots + b_4.B + b_3) - Q^{(1)}.(b_n.B^{m-n-1} + b_{n-1}.B^{m-n-2} + \cdots + b_3.B + b_2)$, we have

$N^{(m-n+1)} = (a_m.B^{m-n+1} + a_{m-1}.B^{m-n} + \cdots + a_n.B + a_{n-1}) - Q^{(m-n)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) -$

$Q^{(m-n-1)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - \cdots - Q^{(3)}.(b_n.B^{m-n-1} + b_{n-1}.B^{m-n-2} + \cdots + b_3.B + b_2) - Q^{(2)}.(b_n.B^{m-n} + b_{n-1}.B^{m-n-1} + \cdots + b_2.B + b_1) - Q^{(1)}.(b_n.B^{m-n+1} + b_{n-1}.B^{m-n} + \cdots + b_2.B^2 + b_1.B)$.

According to our convention, $N^{(m-n+1)} = Q^{(m-n+1)}.D + R^{(m-n+1)}$.

The work array W then gets updated as follows.

$W = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + Q^{(3)}.B^{m-2} + \cdots + Q^{(m-n+1)}.B^n + [\{R^{(m-n-1)} - Q^{(m-n)}.b_n\}.B + \{a_n - Q^{(m-n)}.b_{n-1} - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\} - Q^{(m-n+1)}.b_n].B^{n-1} + \{a_{n-1} - Q^{(m-n+1)}.b_{n-1} - Q^{(m-n)}.b_{n-2} - Q^{(m-n-1)}.b_{n-3} - \cdots - Q^{(2)}.b_1\}.B^{n-2} + \cdots + \{a_2 - Q^{(m-n+1)}.b_2 - Q^{(m-n)}.b_1\}.B + \{a_1 - Q^{(m-n+1)}.b_1\}$.

Now, consider the portion of the above expression in the square bracket

$\{R^{(m-n-1)} - Q^{(m-n)}.b_n\}.B + \{a_n - Q^{(m-n)}.b_{n-1} - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\} - Q^{(m-n+1)}.b_n = R^{(m-n-1)}.B - Q^{(m-n)}.\{b_n.B + b_{n-1}\} + \{a_n - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\} - Q^{(m-n+1)}.b_n = R^{(m-n-1)}.B + R^{(m-n)} - N^{(m-n)} + \{a_n - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\} - Q^{(m-n+1)}.b_n = R^{(m-n-1)}.B + R^{(m-n)} - \{(a_m.B^{m-n} + a_{m-1}.B^{m-n-1} + \cdots + a_n) - Q^{(m-n-1)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) - Q^{(m-n-2)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - \cdots - Q^{(2)}.(b_n.B^{m-n-1} + b_{n-1}.B^{m-n-2} + \cdots + b_3.B + b_2) - Q^{(1)}.(b_n.B^{m-n} + b_{n-1}.B^{m-n-1} + \cdots + b_2.B + b_1)\} + \{a_n - Q^{(m-n-1)}.b_{n-2} - \cdots - Q^{(1)}.b_1\} - Q^{(m-n+1)}.b_n = R^{(m-n-1)}.B + R^{(m-n)} - \{(a_m.B^{m-n} + a_{m-1}.B^{m-n-1} + \cdots + a_{n+1}.B) + Q^{(m-n-1)}.D.B + Q^{(m-n-2)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B) + \cdots + Q^{1}.(b_n.B^{(m-n)} + b_{n-1}.B^{(m-n-1)} + \cdots + b_2.B) = N^{(m-n-1)}.B + \{R^{(m-n)} - Q^{(m-n+1)}.b_n\} - N^{(m-n-1)}.B = R^{(m-n)} - Q^{(m-n+1)}.b_n$.

Substituting the portion of the square bracket in the expression of W in the $(m-n+1)$-th iteration by its simplified form, we have

$W = Q^{(1)}.B^m + Q^{(2)}.B^{m-1} + Q^{(3)}.B^{m-2} + \cdots + Q^{(m-n+1)}.B^n + (R^{(m-n)} - Q^{(m-n+1)}.b_n).B^{n-1} + \{a_{n-1} - Q^{(m-n+1)}.b_{n-1} - Q^{(m-n)}.b_{n-2} - \cdots - Q^{(2)}.b_1\}.B^{n-2} + \{a_{n-2} - Q^{(m-n+1)}.b_{n-2} - Q^{(m-n)}.b_{n-3} - \cdots - Q^{(3)}.b_1\}.B^{n-3} + \cdots + \{a_2 - Q^{(m-n+1)}.b_2 - Q^{(m-n)}.b_1\}.B + \{a_1 - Q^{(m-n+1)}.b_1\}$.

Since first $m-n+1 = n$ terms of work array W (from $W[0]$ to $W[n-1]$) contribute to the formation of quotient q and the remaining n terms take part in the formation of remainder r, so q and r before the normalization step will have the following form.

$q = Q^{(1)}.B^{m-n} + Q^{(2)}.B^{m-n-1} + Q^{(3)}.B^{m-n-2} + \cdots + Q^{(m-n+1)}$.

$r = (R^{(m-n)} - Q^{(m-n+1)}.b_n).B^{n-1} + \{a_{n-1} - Q^{(m-n+1)}.b_{n-1} - Q^{(m-n)}.b_{n-2} - \cdots - Q^{(2)}.b_1\}.B^{n-2} + \{a_{n-2} - Q^{(m-n+1)}.b_{n-2} - Q^{(m-n)}.b_{n-3} - \cdots - Q^{(3)}.b_1\}.B^{n-3} + \cdots + \{a_2 - Q^{(m-n+1)}.b_2 - Q^{(m-n)}.b_1\}.B + \{a_1 - Q^{(m-n+1)}.b_1\}$.

We now prove that, prior to the normalization the following quotient–remainder relationship holds.

Lemma 1. $q.b + r = a$.

Proof. $q.b = \{Q^{(1)}.B^{m-n} + Q^{(2)}.B^{m-n-1} + Q^{(3)}.B^{m-n-2} + \cdots + Q^{(m-n+1)}\}.b = \{Q^{(1)}.B^{m-n} + Q^{(2)}.B^{m-n-1} + Q^{(3)}.B^{m-n-2} + \cdots + Q^{(m-n)}.B\}.b + Q^{(m-n+1)}.(b_1 + b_2.B + \cdots + b_{n-2}.B^{n-3}) + Q^{(m-n+1)}.D.B^{n-2} = \{Q^{(1)}.B^{m-n} + Q^{(2)}.B^{m-n-1} + Q^{(3)}.B^{m-n-2} + \cdots + Q^{(m-n)}.B\}.b + Q^{(m-n+1)}.(b_1 + b_2.B + \cdots + b_{n-2}.B^{n-3}) + N^{(m-n+1)}.B^{n-2} - R^{(m-n+1)}.B^{n-2} = \{Q^{(m-n+1)}.(b_1 + b_2.B + \cdots + b_{n-2}.B^{n-3}) + Q^{(m-n)}.(b_1 + b_2.B + \cdots + b_{n-3}.B^{n-4}).B + \cdots + Q^{(3)}.b_1.B^{n-3}\} + \{a_m.B^{m-1} + a_{m-1}.B^{m-2} + \cdots + a_n.B^{n-1} + a_{n-1}.B^{n-2}\} - R^{(m-n+1)}.B^{n-2}$.

Therefore, $q.b + r = a + \{R^{(m-n)}.B^{n-1} + a_{n-1}.B^{n-2} - N^{(m-n+1)}.B^{n-2} - (Q^{(m-n)}.b_{n-2} + Q^{(m-n-1)}.b_{n-3} + \cdots + Q^{(2)}.b_1).B^{n-2} = a$ (by Lemma 2). □

Lemma 2. $R^{(m-n)}.B^{n-1} = \{N^{(m-n+1)} + Q^{(m-n)}.b_{n-2} + Q^{(m-n-1)}.b_{n-3} + \cdots + Q^{(2)}.b_1 - a_{n-1}\}.B^{n-2}$.

Proof. $\{N^{(m-n+1)} + Q^{(m-n)}.b_{n-2} + Q^{(m-n-1)}.b_{n-3} + \cdots + Q^{(2)}.b_1 - a_{n-1}\}.B^{n-2} = \{(a_m.B^{m-n+1} + a_{m-1}.B^{m-n} + \cdots + a_n.B + a_{n-1}) - Q^{(m-n)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) - Q^{(m-n-1)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - \cdots - Q^{(1)}.(b_n.B^{m-n+1} + b_{n-1}.B^{m-n} + \cdots + b_2.B^2 + b_1.B) + Q^{(m-n)}.b_{n-2} + Q^{(m-n-1)}.b_{n-3} + \cdots + Q^{(2)}.b_1 - a_{n-1}\}.B^{n-2} = \{(a_m.B^{m-n} + a_{m-1}.B^{m-n-1} + \cdots + a_n).B - Q^{(m-n)}.D.B - Q^{(m-n-1)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}).B - \cdots - Q^{(1)}.(b_n.B^{m-n} + b_{n-1}.B^{m-n-1} + \cdots + b_2.B + b_1).B\}.B^{n-2} = \{N^{(m-n)} - Q^{(m-n)}.D\}.B^{n-1} = R^{(m-n)}.B^{(n-1)}$ □

The following lemma helps to show that after the application of Algorithm 3 (corrected normalization procedure), the quotient–remainder relationship still holds.

Lemma 3. $Q^{(i)} \in [0, B-1]$, *for* $i = 1, 2, \ldots, m-n+1$.

Proof. Recall that $Q^{(i)}$ values are obtained by dividing $N^{(i)}$ with D, for $i = 1, 2, \ldots, m-n+1$. Let us first consider the case $i = 1$. We have, $N^{(1)} = a_m.B + a_{m-1}$ and $D = b_n.B + b_{n-1}$, where $a_m \in [1, B-1]$ and $b_n \in [1, B-1]$. After the division, we have $N^{(1)} = Q^{(1)}.D + R^{(1)}$. Thus, it is evident that $Q^{(1)} \in [0, B-1]$.

Similarly for $i = 2$, we have $N^{(2)} = (a_m.B^2 + a_{m-1}.B + a_{m-2}) - Q^{(1)}.(b_n.B^2 + b_{n-1}.B + b_{n-2}) = R^{(1)}.B + (a_{m-2} - Q^{(1)}.b_{n-2}) \leqslant (D-1).B + (a_{m-2} - Q^{(1)}.b_{n-2})$. Since $(a_{m-2} - Q^{(1)}.b_{n-2}) < B$, we have $N^{(2)} < D.B$. Hence, $Q^{(2)} = \frac{N^{(2)}}{D} \in [0, B-1]$.

Similarly for $i = 3$, $N^{(3)} = (a_m.B^3 + a_{m-1}.B^2 + a_{m-2}.B + a_{m-3}) - Q^{(1)}.(b_n.B^3 + b_{n-1}.B^2 + b_{n-2}.B + b_{n-3}) - Q^{(2)}(b_n.B^2 + b_{n-1}.B + b_{n-2}) = R^{(1)}.B + (a_{m-3} - Q^{(1)}.b_{n-3} - Q^{(2)}.b_{n-2}) \leqslant (D-1).B + (a_{m-3} - Q^{(1)}.b_{n-3} - Q^{(2)}.b_{n-2})$.

But since $(a_{m-3} - Q^{(1)}.b_{n-3} - Q^{(2)}.b_{n-2}) < B$, we have $N^{(3)} < D.B$ and this leads to the fact that $Q^{(3)} \in [0, B-1]$.

Following the same procedure, it can be shown that for each $i = 1, 2, \ldots, m-n+1$, we have $Q^{(i)} \in [0, B-1]$. □

Therefore, we have shown that the result $q.b + r = a$ holds before the normalization, and Lemma 3 states that the quotient does not require any normalization. Now,

Table 3
Performance analysis – our algorithm vs BigDigits implementation of [2].

Size of the dividend (m)	Size of the divisor (n)	Time required for Knuth's Algorithm (in sec)	Time required for corrected Huang's Algorithm (in sec)
10 001	5000	2	0.31
15 001	7500	4.4	0.72
20 001	10 000	7.9	1.2
30 001	15 000	17	3
40 001	20 000	28	5
50 001	25 000	42	10
60 001	30 000	71	15
80 001	40 000	110	29
100 001	50 000	200	46

we need to consider two distinct cases: (i) the most significant word of the remainder does not require normalization, and (ii) the most significant word of the remainder needs normalization.

In case (i) the normalization of the remainder does not affect the quotient. We now argue that after the normalization, the quotient–remainder relationship holds.

We have, $W = (W[m] + W[m-1].B + \cdots + W[m-n+1].B^{n-1} + W[m-n].B^n + \cdots + W[0].B^m)$, where $m+1$ is the number of words in W. Thus, $Val_{(m+1)-i} = \sum_{j=1}^{i} W[(m+1)-j].B^{j-1}$ (see Eq. (1)), for $i = 1, 2, \ldots, (m+1)$. It is now easy to see that $r = Val_{m-n+1}$. By Property 2, if the normalization is not required at the position $W[m-n+1]$ (the most significant word of the remainder), then $r = Val_{m-n+1}$ holds before and after the normalization. Thus, here the normalization step only ensures that each word of the remainder lies in the range $[0, B-1]$, and $q.b + r = a$ holds after the normalization.

Next, we consider case (ii), where the normalization is required at position $m-n+1$. Here, due to the normalization of the remainder, the quotient gets affected, and $r = Val_{m-n+1}$ holds before normalization but does not hold after normalization (see Property 2). In fact, here the value of q also gets changed and it becomes $q' = q - c$, where c is the normalization factor at position $m-n+1$. Now, the quotient–remainder relationship can be maintained by adjusting the remainder to $r' = r + b.c$. It is easily verifiable that $q'.b + r' = q.b + r = a$. With this change in the work array W, $r' = Val_{m-n+1}$ is neither less than 0, nor it is greater than equal to B^n. Thus, by Property 1, we do not need further normalization at the position $m-n+1$. Thus, we have the following theorem:

Theorem 1. *Algorithm 1 along with the normalization procedure stated in Algorithm 3 correctly computes quotient and remainder of a multi-precision integer division problem.*

5. Conclusion

The time complexity analysis of the corrected version of the algorithm remains the same as has been described in the original paper [1]. We have implemented our algorithm and compared its performance against the BigDigits [6] implementation of the Knuth's algorithm [2]. BigDigits library includes the classical multiple-precision arithmetic algorithms from Knuth [2] to carry out large natural number calculations as required in cryptographic algorithms. We have conducted our experiments in an Intel Dual Core 1.80 GHz machine with 1 GB of RAM, the results of which are shown in Table 3. Thus, the modification of Huang's algorithm [1] proposed in this paper is around 3.5 to 4 times faster compared to the Knuth's method, as was claimed in [1] for their version of the algorithm.

Acknowledgement

We thank the referee of this paper for the helpful comments. It improved the presentation of the paper.

References

[1] L. Huang, Y. Luo, H. Zhong, H. Shen, An efficient multiple-precision division algorithm, in: Proceedings of Int. Conf. on Parallel and Distributed Computing, Applications and Technologies, PDCAT-05, 2005, pp. 971–974.
[2] D.E. Knuth, The Art of Computer Programming: Semi-Numerical Algorithms, vol. 2, Addison-Wesley, 1981.
[3] D.M. Smith, A multi-precision division algorithm, Math. Comput. 65 (1996) 157–163.
[4] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures for public-key cryptography, Commun. ACM 21 (1978) 120–126.
[5] M.O. Rabin, Probabilistic algorithms for testing primality, J. Number Theory 12 (1980) 128–138.
[6] BigDigits multiple-precision arithmetic source code, http://www.di-mgt.com.au/bigdigits.html.