DOI: https://doi.org/10.24297/jam.v16i0.8301

Spectacular Exponents: A semi modular Approach to Fast Exponentiation

Robert J. Valenza

Claremont McKenna College 500 E. Ninth Street Claremont, California, USA 91711

rvalenza@cmc.edu

Abstract

This paper introduces a computational scheme for calculating the exponential b^w where b and w are positive integers. This two-step method is based on elementary number theory that is used routinely in this and similar contexts, especially the Chinese remainder theorem (CRT), Lagrange's theorem, and a variation on Garner's algorithm for inverting the CRT isomorphism. We compare the performance of the new method to the standard fast algorithm and show that for a certain class of exponents it is significantly more efficient as measured by the number of required extended multiplications.

Keywords: Integer Exponentiation, Modular Exponentiation, Chinese Remainder Theorem, Garner's Algorithm, Generating Functions.

Introduction and Preliminary Estimates of Multiplicative Complexity

Throughout this analysis we are concerned with the multiplicative complexity of the exponential calculation, and accordingly we introduce two associated functions. The *multiplicative length* ML(a) of an integer a is a simple adjustment to the binary length of its absolute value:

$$\mathrm{ML}(a)=\begin{cases}0 & |a|\le 1\\ r+1 & \text{if } |a|=2^r \text{ for some } r\\ \lceil \log_2|a|\rceil & \text{otherwise.}\end{cases}$$

Provisionally, the *multiplicative complexity* MC(a,b) of the product of integers a and b is then defined as the product ML(a)·ML(b). This is admittedly a crude measure of the number of underlying elementary operations, but nonetheless useful for general analysis, especially in connection with extended integer arithmetic (i.e., class constructions that exceed the native integer data types in a given programming language on a given system). Less formally, we define the multiplicative complexity of any algorithmic calculation as the sum of the complexities of the component multiplications. Thus our subsequent analysis will not consider the implementation or particulars of the actual multiplications but simply use this notion of multiplicative complexity as a comparison device. Explicit additions (those not occurring in the context of a multiplication) are completely ignored. We are careful to use this convention fairly in the sense that we shall never artificially reduce the multiplicative complexity of a calculation by, for instance, introducing repeated additions. Along the same lines, we want to be clear at the outset that ultimately, we shall be directing our attention exclusively to multiplicative complexity accruing from multiplication of extended integers and neglecting the computational cost of native arithmetic. To the extent that these costs are not negligible, our development must be taken as somewhat theoretical, but we shall take care to point out where such native arithmetic enters significantly into the control flow of our algorithms in processing extended arithmetic data. (See [1] for a sophisticated survey of exponential methods and [2] for related material.)

Let us consider the calculation of b^w mod m where b and w are positive integers and m is a nonnegative integer. (Note that the case m = 0 is implicitly reduced to ordinary arithmetic, and hence we allow the expression mod 0 in subsequent pseudocode fragments.) A direct implementation of this calculation would, of course, proceed as follows:

```
function DirectExp(b,w,m)

   b = b mod m;

   x = 1;

   for (j = 1; j <= w; j++)

   {

      x = (x * b) mod m;

   }

return x;
```

As explained directly following the equation below, if *m* = 0, at step *j* we are multiplying a number of approximate multiplicative complexity $(j-1)\cdot(\mathrm{ML}(b) + 1 - 1/\ln(2))$ by another factor of *b*, so the multiplicative complexity of the entire calculation is approximated by

$$\sum_{j=1}^{w}(j-1)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})\mathrm{ML}(b)=\left(\frac{w(w-1)}{2}\right)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})\mathrm{ML}(b)\,. \tag{1}$$

In this and subsequent similar symbolic calculations, we often approximate ML by the base-two logarithm, and indeed in the preceding equation the middle factor is obtained by adjusting ML(b) by the average value of $\log_2(x)$ across the interval from 2^{n-1} to 2^n. The formula is naturally more accurate near the geometric mean of these limits than it is near the extremes. We shall see this approach to such approximations again in subsequent estimates but make no further comment upon it. (We shall, however, make an exact accounting for all specific numerical examples.)

If *m* > 0, except in degenerate cases, at every step but the first we are multiplying *b* mod *m* by a number that is, on average, of magnitude *m*/2. Therefore the multiplicative complexity in this case is approximately $(w-1)\cdot\mathrm{ML}(b \bmod m)\cdot(\mathrm{ML}(m)-1)$, an estimate that is more accurate as m approaches a power of 2 from below. (A better approximation for that last factor will occur subsequently in connection with the standard fast algorithm.) We can, of course, reduce the multiplicative complexity of the direct algorithm in the modular case by using a set of congruence class representative's mod *m* centered at 0 (hence allowing negative integers). With this modification, the final factor in the expression for multiplicative complexity is decremented by 1, and in half the cases the middle factor is likewise decreased.

Next, we consider the standard fast algorithm, which is based on the binary expansion of the exponent *w*; the point is to accumulate b^w from the successive squares $b^{2^{j-1}}$, where j runs from 1 to the bit length of w. This method generally occurs in the context of modular arithmetic (m > 0), but it makes sense for ordinary arithmetic, too. Here is the associated pseudocode:

```
function FastExp(b,w,m)

   s = b mod m;

   x = 1;

   while (w > 0)
```

```
    {
        If(w mod 2 != 0)
        {
            x = x * s mod m;
        }
        w = w/2;
        If(w > 0)
        {
            s = (s * s) mod m;
        }
    }
return x;
```

The multiplicative complexity of this algorithm again, of course, depends on whether m is 0. In both cases, we break the calculation into two parts: first, the calculation of the successive squares and, second, the product accumulation. Henceforth assume that $w > 1$, and let $J = J(w) = \lfloor \log_2 w \rfloor$. Note that the number of times that the loop in the algorithm executes is $J + 1$, while the number of times the square calculation executes is only J.

In the case of ordinary arithmetic, at the head of the loop on its j-th iteration, the multiplicative length of the square variable is estimated by $2^{j-1}(\mathrm{ML}(b) + 1 - 1/\ln(2))$. The multiplicative complexity of the square is accordingly $2^{2(j-1)}(\mathrm{ML}(b) + 1 - 1/\ln(2))^2$, and thus the approximate multiplicative complexity for this part of the calculation is given by

(2) $$\sum_{j=1}^{J} 2^{2(j-1)}(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 = \frac{4^J-1}{3}(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 \ .$$

For the second part of the estimate, consider first what happens when $w = 2^r - 1$, for some positive integer r, which is to say that the binary expansion of w consists of all ones. The value P_j of the accumulated product at the foot of the loop on its j-th iteration is clearly just b raised to the power $2^j - 1$, and accordingly,

$$\mathrm{ML}(P_j) = (2^j - 1)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)}) \, .$$

We can now use this to estimate the multiplicative complexity $\mathrm{MC}(P_j)$ of the calculation of P_j for each iteration of the loop. On the first iteration, one of the factors is one, so $\mathrm{MC}(P_1) = 0$. For $j > 1$, we need to multiply the multiplicative length of the square variable (as referenced at the head of the loop) by that of the accumulated product from the previous iteration. Accordingly,

$$\begin{aligned}\mathrm{MC}(P_j) &= 2^{j-1}(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})\cdot \mathrm{ML}(P_{j-1}) \\ &= 2^{j-1}(2^{j-1}-1)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2\end{aligned}$$

(with, in fact, no adjustment needed for $j = 1$) . Thus the total multiplicative complexity of the calculation of all the P_j (still apart from the calculation of the successive squares) is given by

$$\sum_{j=1}^{J+1} 2^{j-1}(2^{j-1}-1)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 = \left(\frac{4^{J+1}-1}{3}-2^{J+1}+1\right)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2$$

for the special case that all the binary digits of w are one.

In the general case of an unstructured sequence of binary digits, with the exception of the final term in the summation, the estimated value of $\mathrm{ML}(P_j)$ *and* the expected number of terms in the sum exhibited directly above are both reduced by half, and this reduces the estimate of the multiplicative complexity of the second part of the calculation by a factor of one quarter. The final term is extraordinary in that the most significant digit of w is by definition 1, not 0. Nonetheless, the expected value of $\mathrm{ML}(P_j)$ at the head of the last iteration is only half of the expected value in the special case just considered, and this gives us the following expression for the average complexity of the product accumulation:

(3)
$$\frac{1}{4}\cdot\sum_{j=1}^{J} 2^{j-1}(2^{j-1}-1)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 + \frac{1}{2}\cdot 2^J(2^J-1)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 =$$
$$\frac{1}{4}\cdot\left(\frac{4^J-1}{3}-2^J+1\right)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 + \frac{1}{2}\cdot(4^J-2^J)(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 .$$

We may now sum expressions (2) and (3) to get an estimate of the total complexity of the fast exponentiation algorithm for ordinary integer arithmetic ($m = 0$). Neglecting the "lower order" terms, for J not too small our approximation comes to

(4)
$$\left(\frac{1}{12}+\frac{1}{2}+\frac{1}{3}\right)4^J(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 = \frac{11}{12}4^J(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 .$$

To make a comparison with estimate (1) of the direct algorithm, we need to replace this expression in J with an appropriate approximation in the original variable w. This begins with a very special case of Euclidean division, namely that of w divided by 2^J. By construction, this defines two auxiliary variables r and κ, as follows:

$$\begin{aligned} w &= 2^J + r \quad (0 \le r < 2^J) \\ &= 2^J \cdot \kappa \quad (1 \le \kappa < 2) \end{aligned}$$

so that

$$\kappa = 1 + \frac{r}{2^J}$$

where r is subject to a uniform random distribution among the integers between 0 and $2^J - 1$. Thus the expected value of κ is approximated asymptotically by 3/2, and, of course, $2^J = w/\kappa$. The upshot is that the estimate (4) in terms of w rather than J amounts to

(5)
$$\frac{11}{27}w^2(\mathrm{ML}(b)+1-\frac{1}{\ln(2)})^2 .$$

Comparing this with expression (1) shows that in general the standard fast algorithm is also, by this measure, superior to the direct calculation even modulo 0. [Actual numerical experiments, which we shall not reproduce here, bear out this conclusion and the estimates (1) and (5). As noted earlier, estimate (1) is most accurate at the geometric mean of successive powers of two; estimate (5) exhibits similar behavior, but its accuracy also depends on the number and distribution of nonzero digits in the binary expansion of *w*.]

Finally, we look at the familiar modular case, $m > 0$, for the standard fast algorithm. To estimate the expected multiplicative complexity of the algorithm we must again look separately at the product accumulations and the squaring. In both cases, we need to know the expected multiplicative length of a non-negative integer smaller than the modulus, and we shall calculate an approximation to this in two steps.

First, assume that our modulus is of the form $m = 2^n - 1$. Then the expected value of the multiplicative length is clearly

$$E(\mathrm{ML}(0 \to 2^n - 1)) = \frac{1}{2^n}\left(\sum_{k=0}^{n-1}(k+1)2^k - 1\right)$$

where the adjustment term of –1 merely account for the fact that ML (1) = 0. This can be evaluated by introduction of the generating function (see [3])

$$A_n(x) = \sum_{k=0}^{n-1}(k+1)x^k$$

which is just the derivative of the geometric series in $\sum x^k$, for $k = 0,\ldots,n$. Consequently,

(6) $$A_n(x) = \left(\frac{x^{n+1}-1}{x-1}\right)' = \frac{(n+1)x^n - (x^{n+1}-1)}{(x-1)^2}$$

and, substituting $x = 2$, we have at once that

(7) $$E(\mathrm{ML}(0 \to 2^n - 1)) = \frac{1}{2^n}(A_n(2) - 1) = n - 1 \ .$$

This formalizes the intuitive argument that the expected value of the multiplicative length of a random integer mod 2^n should simply be $n - 1$.

Before completing the estimate for the multiplicative complexity of the standard fast algorithm modulo $m > 0$, it is convenient to introduce a second auxiliary function that arises in connection with the expectation of the square of the multiplicative length. Accordingly, let $B_n(x)$ be defined by

$$B_n(x) = \sum_{k=0}^{n-1}(k+1)^2 x^k \ .$$

The key relationship between $A_n(x)$ and $B_n(x)$ is

$$B_n(x) + 2A_n(x) + \frac{x^n - 1}{x - 1} = \frac{1}{x}\left\{B_n(x) - 1 + (n+1)^2 x^2\right\}$$

whence some routine but tedious algebra yields in particular that

(8)
$$E(\mathrm{ML}^2(0 \to 2^n - 1) = \frac{1}{2^n}(B_n(2) - 1)$$
$$= (n-1)^2 + 2 + \frac{1}{2^{n-2}} \quad .$$

We can now complete the general calculations for the expected values of both ML and ML^2 from 0 to $m = 2^n + r$, $0 \le r < 2^n$. Since the multiplicative length of integers from 2^n to m is $(n + 1)$, we have the precise expressions

$$E(\mathrm{ML}(0 \to 2^n + r) = \frac{2^n(n-1) + (r+1)(n+1)}{2^n + r + 1}$$

and

$$E(\mathrm{ML}^2(0 \to 2^n + r) = \frac{2^n[(n-1)^2 + 2 + \frac{1}{2^{n-2}}] + (r+1)(n+1)^2}{2^n + r + 1} \, .$$

Moreover, these expressions are, respectively, clearly bounded from above by $E(\mathrm{ML}(0 \to 2^{n+1} - 1))$ and $E(\mathrm{ML}^2(0 \to 2^{n+1} - 1))$. Thus the expected multiplicative complexity of the accumulation component is bounded from above by

$$\frac{n}{2} J(w)\mathrm{ML}(b)$$

and the expected multiplicative complexity of the squaring component is bounded by

$$(J(w) + 1)(n^2 + 2 + \frac{1}{2^{n-1}}) \, .$$

The consequence is that the expected multiplicative complexity of the fast algorithm modulo $m > 0$ is bounded from above by

(9)
$$\frac{n}{2} J(w)\mathrm{ML}(b) + (J(w) + 1)(n^2 + 2 + \frac{1}{2^{n-1}})$$

which involves only the products of logarithmic factors, and hence overwhelmingly outperforms direct exponentiation by this measure.

One final note that is paramount to the sequel is the familiar fact that in computing b^w mod m, for $m > 0$ and b relatively prime to m, we may reduce w modulo $\varphi(m)$, where φ is the Euler phi function, the number of congruence classes represented by numbers relatively prime to m. This number is easily computed from the prime factorization of m, and, for small m, essentially trivializes modular exponentiation regardless of the size of w.

A Semi modular Approach

For nonnegative integers m, we let $\mathbf{Z}/m\mathbf{Z}$ denote the ring of integers mod m (whence the natural abstract algebraic identification of $\mathbf{Z}/0\mathbf{Z}$ with $\mathbf{Z}$ is consistent with our previous convention). Given a family of relatively prime positive integers $m_1,\ldots,m_s$, the Chinese remainder theorem asserts that the following map is an isomorphism of rings with unity [4]:

$$\chi : \mathbf{Z} / \prod_{j=1}^{s} m_j \mathbf{Z} \to \mathbf{Z} / m_1 \mathbf{Z} \times \cdots \times \mathbf{Z} / m_s \mathbf{Z}$$
$$a \bmod \prod_{j=1}^{s} m_j \mapsto (a \bmod m_1, ..., a \bmod m_s)$$

Note that the elements of the codomain look like coordinate vectors for which the j-th coordinate of the image of a mod $\prod m_j$ is simply the projection of a into $\mathbf{Z}/m_j\mathbf{Z}$. For us, the most important particular elements of this assertion are the following:

1. The operations of addition and multiplication on the codomain are defined componentwise (similar to the operations of addition and scalar multiplication in linear algebra):

$$(a_j \bmod m_j) + (b_j \bmod m_j) = (a_j + b_j \bmod m_j)$$
$$(a_j \bmod m_j) \cdot (b_j \bmod m_j) = (a_j \cdot b_j \bmod m_j)$$

2. The map χ is both additive and multiplicative:

$$\chi(a+b) = \chi(a) + \chi(b)$$
$$\chi(a \cdot b) = \chi(a) \cdot \chi(b)$$

   Here, of course, the variables are understood as integers mod M, where $M = \prod m_j$. Moreover, $\chi(1) = (1,,1)$.

3. The map χ is bijective (hence, invertible) and the inverse map is likewise additive and multiplicative.

In the context of this paper, the clear temptation here is to compute $x = b^w$ as follows, using what we shall refer to as a *semi modular* approach:

- Choose $M = \prod m_j > x$.
- Compute $x_j = x^w$ mod m_j for all indices j.
- Compute $\chi^{-1}(x_1,...,x_s)$.

[Recall from above that in the second step, we have the possible reduction of w mod $\varphi(m_j)$.] Since χ is multiplicative and multiplication in the product ring is defined componentwise, we see that

$$\begin{aligned}\chi^{-1}(x_1,...,x_s) &= \chi^{-1}(b^w \bmod m_1,...,b^w \bmod m_s)\\ &= \chi^{-1}((b \bmod m_1,...,b \bmod m_s)^w)\\ &= \chi^{-1}(b \bmod m_1,...,b \bmod m_s)^w\\ &= \chi^{-1}(\chi(b))^w\\ &= \chi^{-1}(\chi(b^w))\\ &= \chi^{-1}(\chi(x))\\ &= x \bmod M\end{aligned}$$

and x mod M is just x because M is chosen to be larger than x, provided of course that χ^{-1} is constructed to return values between 0 and $m - 1$. Moreover, the middle step, where the exponentials occur, may be accomplished via fast modular exponentiation, hence the ostensible efficiency. However, what is obscured here is that the final step, the inversion of the CRT map χ, is hardly the most facile of computations. One might also note that unless the moduli and their associated partial products are reusable, and chosen and stored in advance, the computation of M will by itself have multiplicative complexity comparable to the naïve algorithm. We shall address both of these obstructions later.

Inversion of the Isomorphism of the Chinese Remainder Theorem

Let n be a positive integer and let a be any integer relatively prime to n. As a convenient notational device, we shall define

$$n^{-1}(a) = a^{-1} \bmod n$$

Thus in this context, n^{-1} is an operator that directs us to invert the indicated integer modulo n. Next, with M and its relatively prime factors m_j as above, let

$$M_{\hat{j}} = \prod_{k \neq j} m_k = m / m_j \ ,$$

so that the circumflex on the subscript j indicates the omission of m_j from the indicated product, and consequently $M_{\hat{j}}$ is relatively prime to m_j. By construction, it follows that

$$m_j^{-1}(M_{\hat{j}}) \cdot M_{\hat{j}} \bmod m_k \equiv \delta_{jk} \bmod m_k \ .$$

In other words, for any index j, the product $m_j^{-1}(M_{\hat{j}}) \cdot M_{\hat{j}}$ is congruent to 1 modulo m_j and congruent to 0 modulo every other modulus m_k. Since χ is a ring homomorphism, this implies at once that

$$\chi(\sum_{j=1}^{s} a_j m_j^{-1}(M_{\hat{j}}) \cdot M_{\hat{j}}) = (a_1, \ldots, a_s)\ .$$

Therefore

$$\chi^{-1}(a_1, \ldots, a_s) = \sum_{j=1}^{s} a_j \cdot m_j^{-1}(M_{\hat{j}}) M_{\hat{j}} \qquad (10)$$

and we have explicitly inverted χ. One sees at once, however, how expensive this is in terms of multiplicative complexity. Let us look at an estimate.

Consider each term in the preceding equation as the product of two factors, separated by the dot. We provisionally ignore the calculation of the factors themselves and assume that each of the s moduli m_j is approximately $\mathrm{M}^{1/s}$; some must be larger, some smaller. Again treating ML as if it were purely logarithmic, we have the approximations

$$\mathrm{ML}(m_{\hat{j}}) \approx \frac{1}{s}\mathrm{ML}(M)$$
$$\mathrm{ML}(M_{\hat{j}}) \approx (1 - \frac{1}{s})\mathrm{ML}(M)\ .$$

(For this part of the analysis, it is more convenient to deal with approximations rather than expectations.) Both a_j and $m_j^{-1}(M_j)$ may be regarded as uniformly distributed random nonnegative integers less than m_j, whence their bit lengths drop on average by 1. Accordingly,

$$\mathrm{ML}(a_j) \approx m_j^{-1}(M_j)) \approx (\frac{1}{s}\mathrm{ML}(M) - 1) \quad .$$

This gives at once an estimate for the multiplicative complexity of each of the s products occurring in the summation on the right-hand side of equation (4), which in turn gives the following total estimate for the multiplicative complexity of this naïve inversion of the CRT applied modulo M:

(11)
$$\begin{aligned}\mathrm{MC}(\chi_{\text{naive}}^{-1}) &\approx \sum_{j=1}^{s}(\frac{1}{s}\mathrm{ML}(M) - 1)(\mathrm{ML}(M) - 1)\\ &\approx \mathrm{ML}(M)^2 - s\mathrm{ML}(M) - \mathrm{ML}(M) + s \quad .\end{aligned}$$

The sign of the second term calls into question the ratio of $s/\mathrm{ML}(M)$ insofar as it decreases the coefficient of the $\mathrm{ML}(M)^2$-term. How large can this number be? To answer this, we introduce some notation:

$s(M)$	=	the number of distinct prime factors of M (equivalently, the maximum value for the number of relatively prime factors of M)
$R(M)$	=	$s(M)/\log_2(M)$
p_s	=	the s-th (positive) prime
Π_s	=	the product of the first s primes
R_s	=	$R(\Pi_s)$

The key result is that the values of $R(M)$ are governed by those of the R_s in the sense captured by the second part of the following proposition:

PROPOSITION. The function $R(M)$ has the following properties:

(i) The sequence R_s is strictly decreasing.

(ii) For every s, if $M > \Pi_s$, then $R(M) < R_s$.

(iii) $R(M) \to 0$ as $M \to \infty$.

PROOF. For part (i), we must show that

$$\frac{s+1}{\sum_{j=1}^{s+1}\log_2(p_j)} < \frac{s}{\sum_{j=1}^{s}\log_2(p_j)} \quad .$$

This, however, is equivalent to the assertion

$$\frac{1}{s} < \frac{\log_2(p_{s+1})}{\sum_{j=1}^{s}\log_2(p_j)}$$

which in turn is equivalent to the obvious inequality

$$\sum_{j=1}^{s}\log_2(p_j) < s\log_2(p_{s+1}) \quad .$$

This completes part (i).

Now let $M > \Pi_s$. If $s' = s(M) \le s$, the assertion is clear. So suppose that $s' > s$; that is, M has more than s prime factors. Let the prime decomposition of M be given by

$$M = \prod_{j=1}^{s'} q_j^{\alpha_j} \quad .$$

Then since the elements of the sequence $\{ p_j \}_{j=1,\ldots,s'}$ must be bounded by the corresponding elements of the sequence $\{ q_j \}_{j=1,\ldots,s'}$, it follows that

$$R(M) = \frac{s'}{\sum_{j=1}^{s'}\alpha_j \log_2 q_j} < \frac{s'}{\sum_{j=1}^{s'}\log_2 q_j} < \frac{s'}{\sum_{j=1}^{s'}\log_2 p_j} = R_{s'} < R_s \quad .$$

The last inequality is a consequence of part (i), and this completes part (ii).

For part (iii), it is enough to show that the sequence R_s goes to zero. Since there are infinitely many primes, for any positive integer N, we can choose s so large that at least half of the primes up to and including p_s exceed 2^N. We then have the following chain of inequalities:

$$R_s = \frac{s}{\sum_{j=1}^{s}\log_2 p_j} \le \frac{s}{\sum_{j=\lfloor s/2 \rfloor}^{s}\log_2 p_j} \le \frac{s}{\sum_{j=\lfloor s/2 \rfloor}^{s}\log_2 2^N} \le \frac{s}{\frac{s}{2}N} = \frac{2}{N} \, .$$

Hence R_s may be made arbitrarily small, and this completes the proof. ❑

NOTE. The rate of convergence to zero for $R(M)$ is glacially slow, as one might expect from the logarithm in the denominator; this is confirmed by the following short table of values for R_s (Table 1). The arithmetic was performed with 12-digit precision.

We now return to the approximation (11) and examine the consequences of this analysis of $R(M)$. Noting that $\mathrm{ML}(M) \ge s$, we have the following soft bound, which is, nonetheless, sufficient to our subsequent analysis:

$$\mathrm{MC}(\chi_{\mathrm{naive}}^{-1}) \ge (1 - R(M))\mathrm{ML}(M)^2 \tag{12}$$

Since $R(M)$ goes to zero as M goes to infinity, this bound on the multiplicative complexity of the naïve inversion of CRT is asymptotic to $\mathrm{ML}(M)^2$.

s	p_s	R_s
2^{20}	16,290,047	0.0446304
2^{21}	34,136,029	0.0425905
2^{22}	71,378,569	0.0407352
2^{23}	148,948,139	0.0390400
2^{24}	310,248,241	0.0374847
2^{25}	645,155,197	0.0360521

Table 1.

EXAMPLE. Noting that for $s = 28$, Π_s is approximately 10^{42}, and $R_s < 0.2$, it follows that for M larger than roughly 10^{42},

$$\mathrm{MC}(\chi_{\text{naive}}^{-1}) \geq \frac{4}{5}\mathrm{ML}(M)^2 .$$

The next step in our general analysis is to use the bound (12) to compare the multiplicative complexity of the standard fast algorithm for exponentiation (modulo 0), as given in expression (5), with that of the third and final step of the semi modular algorithm suggested above, namely, the inversion via formula (10) of the isomorphism χ.

The Multiplicative Complexity of the Semi modular Approach with Naïve Inversion of CRT

Again let $x = b^w$, and, since we are analyzing the efficiency of the calculation of x mod M via the Chinese remainder theorem for M greater than but near x, we may approximate M by b^w. Hence in this case the inequality (12) reduces to

(13) $$\mathrm{MC}(\chi_{\text{naive}}^{-1}) \geq (1-R(M))w^2(\mathrm{ML}(b)+1-\frac{1}{\ln 2})^2 \quad .$$

Notice that b^w is guaranteed to have relatively few distinct primes in its factorization, hence we do not want to replace $R(M)$ by $R(b^w)$ in this bound, but nonetheless the multiplicative complexity of the naïve inversion algorithm is asymptotic to $w^2(\mathrm{ML}(b) + 1 - 1/\ln 2)^2$. Recalling that estimate (5) for the standard fast algorithm was $(11/27)w^2(\mathrm{ML}(b) + 1 - 1/\ln 2)^2$, we see that for large exponentials—hence large M—whatever the efficiency of exponentiation modulo small moduli, the final step of the suggested algorithm is too costly unless we can find a better inversion method.

Garner's Algorithm

We can see in equation (10) that every term in the naïve algorithm for the inversion of CRT has a factor of the approximate order of $M^{(s-1)/s}$, where s is the number of relatively prime factors chosen for the factorization of M. This is improved by Garner's algorithm [5], which we shall express recursively. As above, we have the moduli m_j, for $j = 1, 2, \ldots, s$, and $(a_1,\ldots,a_s) \in \mathbf{Z}/m_1\mathbf{Z}\times\cdots\times\mathbf{Z}/m_s\mathbf{Z}$ is the element of the product ring to be inverted. First, we define two indexed sets of auxiliary parameters (redefining M_j as it was used previously):

(a) $M_j = \prod_{k=1}^{j} m_k \quad (j = 1,\dots,s-1)$

(b) $N_j = m_j^{-1}(M_{j-1}) = \prod_{k=1}^{j-1} m_j^{-1}(m_k) \quad (j = 1,\dots,s)$

Note that $N_1 = 1$, the empty product. The heart of the algorithm lies in two iterative calculations that are essentially intertwined. We thus introduce sequences U_j and V_j defined as follows. We begin with

(c) $U_1 = V_1 = a_1$

and proceed recursively with

(d) $U_j = N_j(a_j - V_{j-1}) \mod m_j \quad (j = 2,\dots,s)$

(e) $V_j = M_{j-1}U_j + V_{j-1} \quad (j = 2,\dots,s)$.

The point of the algorithm is that $V_j = \chi^{-1}(a_1,\dots,a_j)$, which is to say, via an implicitly polymorphic interpretation of χ, that V_j is congruent to a_k modulo m_k for all k from 1 to j. In particular, V_s inverts the full set of modular projections. This holds by definition for the case $j = 1$ as given, and for $j > 1$ we have

$$\begin{aligned} V_j &\equiv M_{j-1} \cdot m_j^{-1}(M_{j-1})(a_j - V_{j-1}) + V_{j-1} \mod m_j \\ &\equiv a_j \mod m_j \end{aligned}$$

as required.

To estimate the multiplicative complexity of Garner's algorithm (in this form), we note that assuming the moduli and associated products are computed in advance, only the calculation of the V_j involves large integers; hence we confine our attention to the last set of calculations. Again assume that each m_j is approximately $M^{1/s}$, so that

$$\mathrm{ML}(U_j) \approx \frac{1}{s}\mathrm{ML}(M) - 1$$

and

$$\mathrm{ML}(M_j) \approx \frac{j}{s}\mathrm{ML}(M) \quad .$$

The multiplicative complexity of calculating all of the V_j may thus be approximated by

$$\begin{aligned} \sum_{j=2}^{s} \frac{j}{s^2}(\mathrm{ML}(M)^2 - s\mathrm{ML}(M)) &\approx \sum_{j=2}^{s} \frac{j}{s^2}(\mathrm{ML}(M)^2 - R(M)\mathrm{ML}(M)^2) \\ &\approx \left\{\frac{1}{2} + \frac{1}{2s} - \frac{1}{s^2}\right\} \cdot (1 - R(M)) \cdot \mathrm{ML}(M)^2 \quad . \end{aligned}$$

Accordingly, for s not too small, we have

$$\mathrm{MC}(\chi_{\mathrm{Garner}}^{-1}) \approx \frac{1}{2}(1 - R(M))\mathrm{ML}(M)^2 \quad .$$

This, in turn, when applied to the case $x = b^w$, yields

$$\mathrm{MC}(\chi_{\mathrm{Garner}}^{-1}) \leq \frac{1}{2}(1 - R(M))w^2(\mathrm{ML}(b) + 1 - \frac{1}{\ln 2})^2 \quad .$$

Thus, recalling inequality (5) with its lead coefficient of 11/27, we have at least the potential for a significant improvement over the standard fast algorithm to the extent that (i) Garner's algorithm may be made more efficient, and (ii) certain exponents w admit moduli for which some part of the CRT inversion becomes computationally trivial.

Two Paths Forward

The remainder of this paper considers two variations on the abstract idea of semi modular exponentiation; the first is a general consideration for choosing a subset of the moduli independently of how the algorithm completes and invites an excursion into the Gaussian integers; the second is more specifically a variant on Garner's algorithm. We shall see that the two in tandem produce some worthwhile results in the right circumstances.

Phi Moduli

We recalled above that when b is relatively prime to m, b^w mod m need only be computed for w mod $\varphi(m)$, where φ is the Euler phi function. This follows from the identity $b^{\varphi(m)} \equiv 1$ mod m for such b. We shall now exploit this in connection with inverting the Chinese remainder theorem.

Consider the special case that $m = p^\alpha$ is a power of a positive prime with either $p > 2$ or $\alpha \leq 2$. Suppose moreover that $x^2 \equiv 1$ mod p^α. Then p^α divides the product $(x + 1)(x - 1)$, and indeed it must divide one of the factors. That tells us that even though $\mathbf{Z}/p^\alpha\mathbf{Z}$ is not necessarily a field, it is still the case that $x \equiv \pm 1$ mod p^α. It follows from this that if $\varphi(p^\alpha) = (p - 1)p^{\alpha - 1} | 2w$, then $b^{2w} \equiv 1$ mod p^α and so $x = b^w \equiv \pm 1$ mod p^α. The point is that if $\varphi(p^\alpha) | 2w$, the projection of $x = b^w$ into the residue ring $\mathbf{Z}/p^\alpha\mathbf{Z}$ is ± 1. Thus if we choose moduli $m_j = p_j^{\alpha j}$ for which $2w | \varphi(m_j)$, at the corresponding step in the naïve inversion of the CRT isomorphism, we need perform no multiplication. Similarly, if $b | m_j$, then $x \equiv 0$ mod p^α, and again the corresponding step in the naïve inversion is trivial. We shall illustrate all of this shortly, but first we show that it can be extended somewhat.

Next consider the extension of the integers $\mathbf{Z}$ to the so-called Gaussian integers $\mathbf{Z}[i] = \{a + bi : a, b \in \mathbf{Z}\}$. Thus the Gaussian integers are simply those complex numbers whose real and imaginary parts are ordinary integers. This extension loses the property of admitting a linear ordering compatible with ordinary arithmetic, but retains the key algebraic properties of $\mathbf{Z}$. We sketch these out minimally:

1. $\mathbf{Z}[i]$ is a Euclidean ring; that is, we can perform Euclidean division with quotients and remainders determined by the norm function, although not uniquely. The group of units in $\mathbf{Z}[i]$ (that is, the invertible Gaussian integers) expands to the set $\{\pm 1, \pm i\}$.

2. As a Euclidean ring, $\mathbf{Z}[i]$ is automatically a principal ideal domain. Thus prime (or irreducible elements) in $\mathbf{Z}[i]$ are exactly those that generate prime ideals. Moreover, we can speak of greatest common divisors and elements that are relatively prime. In particular, the quotient rings corresponding to arithmetic mod z for a Gaussian integer z satisfy the Chinese remainder theorem. The implied isomorphism and its inverse are both defined and computed as with ordinary integers, but using complex arithmetic.

3. The Euler phi function again makes sense for a Gaussian integer z and is defined as the size of the unit group of the corresponding quotient ring; equivalently, it is again the number of residue classes that are invertible mod z. The size and structure of the group of units for **Z**[i] differs from that of **Z**, and we note that the structure of the group of units for the Gaussian integers modulo p^α for powers of an integer prime p is given according to [6] by

$$(\mathbf{Z}[i]/p^\alpha\mathbf{Z}[i])^\times \cong C_{p^{\alpha-1}} \times C_{p^{\alpha-1}} \times C_{p^2-1}.$$

Here C_n denotes the cyclic group of order n, and the exponent for the group of units is thus $(p^2 - 1)\, p^{\alpha-1}$. This higher value might seem to work against us, insofar as good exponents would seem to be rarer in this setting, having to accommodate a larger factor, but since we are only concerned with exponentiating ordinary integers, this is in fact not the case, as we shall see in a moment.

Now the point of all this will be clear as soon as we recall a famous theorem by Gauss. In its simplest form—which is all we need—it states that an integer prime p factors (or *splits*) in **Z**[i] if and only if $p \equiv 1 \bmod 4$, in which case the factorization takes the form $p = (a + bi) \cdot (a - bi)$ for some ordinary integers a and b. (The resulting factors are now Gaussian primes.) For such p, we can exploit this in the choice of good moduli for the naïve inversion of the CRT as follows: Let $m = p^\alpha$ with p and α modestly restricted as above, but now assume that $x^4 \equiv 1 \bmod p^\alpha$ in **Z**. Then since $(a \pm bi) \mid p$, we have natural projections

$$\mathbf{Z}/p^\alpha\mathbf{Z} \to \mathbf{Z}[i]/(a \pm bi)^\alpha\mathbf{Z}[i]$$

and we may read $x^4 \equiv 1$ as a congruence in the Gaussian integers mod $(a \pm bi)^\alpha$. But over the latter ring, $x^4 - 1$ factor into $(x + 1)\,(x - 1)\,(x + i)\,(x - i)$ as an elementary matter of complex arithmetic. As above, this tells us that if $x^4 \equiv 1 \bmod p^\alpha$ in **Z**, then $x \equiv \pm 1, \pm i \bmod (a \pm bi)^\alpha$ in **Z**[i]. The upshot is that if p splits and $\varphi(p^\alpha)|4w$, the projection of $x = b^w$ into the residue rings $\mathbf{Z}/(a \pm bi)^\alpha$ is ±1 or ±i. Thus if we choose moduli $m_j = p_j^{\alpha j}$ for such primes subject to the further condition that $4w|\varphi(m_j)$, then the corresponding factors in the naïve inversion of the CRT on **Z**[i] require no multiplication whatsoever. [Note, by the way, that if p does not split and $\varphi(p^\alpha)|4w$, then since $p - 1$ has only a single factor of 2, then also $\varphi(p^\alpha)|2w$. Thus searching for good moduli over the Gaussian integers automatically yields good ordinary integer moduli.]

Numerical examples of what we shall call *phi moduli*—that is, moduli for which the projections of b^w constitute units in **Z** or **Z**[i]—are given in Table 2. (These rare but spectacular examples of exponents with extraordinarily large numbers of phi moduli are actually just curios because, as the exponent w increases, the part of the exponentiation that we get "for free" via these moduli evidently represents only a small part of the overall calculation.) For now, let us only note further that the functions ML and MC are easily extended to the Gaussian integers via ML($a + bi$) = ML(a) + ML(b). Accordingly, the value of MC is the sum of four terms as given by the distributive law.

w	$\log_{10}(m_1)$	Number of factors	First five moduli					Last five moduli				
98280	213.900	74	4	81	25	49	11	39313	65521	131041	196561	393121
97020	174.435	62	4	27	25	343	121	55441	77617	97021	129361	388081
96390	171.003	61	4	243	25	49	11	27541	38557	42841	128521	192781
95760	162.382	58	4	27	25	49	11	47881	54721	63841	127681	383041
90720	170.951	63	4	243	25	49	11	15121	20161	30241	45361	72577
90090	184.572	66	4	27	25	49	121	6037	51481	72073	120121	180181
86940	160.207	60	4	81	25	49	11	15121	16561	17389	24841	49681
85680	168.415	62	4	27	25	49	11	17137	20161	24481	34273	42841
83160	205.474	73	4	81	25	49	121	47521	55441	66529	110881	332641
81900	185.250	67	4	27	125	49	11	21841	54601	65521	81901	109201
75600	186.459	67	4	81	125	49	11	30241	33601	43201	100801	151201
71820	161.951	58	4	81	25	49	11	35911	47881	57457	71821	287281
69300	179.772	66	4	27	125	49	121	18481	19801	34651	55441	92401
65520	170.844	63	4	27	25	49	11	21841	26209	37441	65521	131041
64260	187.106	67	4	81	25	49	11	15121	17137	36721	42841	128521
56700	160.564	60	4	243	125	49	11	15121	28351	32401	45361	56701
49140	168.124	62	4	81	25	49	11	24571	28081	39313	65521	196561

Table 2. These are all the positive integers under 10^6 such that the product of the phi moduli (Gaussian case) is greater than 10^{160}. In the second column, m_1 denotes their product (which would be used as the first modulus for the semi modular exponentiation). The corresponding initial and final segment of the factors consolidate any complex conjugate pairs into a single product.

Sieving with Garner's Algorithm

Recall that the expensive part of Garner's algorithm is the extended-integer multiplication explicit in the final step. In the same way that the naïve inversion of the CRT is improved by having many of the projections b^w project onto ±1 or $\pm i$ in the product of the residue rings, Garner's algorithm is made more efficient by having the variable U_j reduced to something small. But this is a matter of a felicitous choice of the modulus m_j , and here we do seem to have some scope. Let us sketch out how this might be done, postponing certain details—and one enormous obstruction—for the moment.

(i) Find all the phi moduli, combine them by multiplication into an initial modulus m_1 with corresponding projection a_1 obtainable by nothing but (possibly complex) addition.

(ii) Using a list or array of primes from some chosen interval, sieve iteratively for the next apt modulus m_j by computing $U_j = U_j(m)$ via the successive choice of candidate moduli m from the list of primes, looking for small values of U_j. Keep in mind that this is a modular calculation that need not require extended arithmetic and that the list of moduli so obtained need not occur in increasing order: we are free at the completion of any iteration to go back to the beginning of the list, provided that we skip over primes that have already been chosen or have occurred in the factorization of m_1. (This actually enhances our efficiency.) Once m_j and U_j are chosen, compute V_j. This will require extended arithmetic, but the cost has been reduced by the choice of m_j.

(iii) The loop concludes when the composite modulus M_j exceeds the floating-point estimate of b^w. The current V_j is our result. (If we have chosen to use residue class representatives centered at 0, as suggested above, our final result will be $V_j + M_j$, should V_j be less than or equal to zero.)

Perhaps the most evident cost here is that the sieving for the U_j implicitly also entails the calculation of b^w mod m_k for several moduli. This is not a deal breaker since these calculations are modular and only incidentally involve extended integers. Indeed, we can reuse the projections b^w mod m_k to enhance the efficiency of even these short calculations. The not-so-evident cost—perhaps because this assumption has been buried for so long in our approach—is that *unless the moduli and their order of occurrence can be set in advance, the assumption that the products M_j and M itself can be precomputed collapses*. Since the calculation of M is comparable to the direct calculation of b^w, any advantage gained working via small moduli, with or without sieving with respect to Garner's algorithm, will be irretrievably lost.

Given that the italicized conditional just asserted is unavoidable, we turn now to a double-sieving variant that is successful with a plausible adjustment of the calculation of multiplicative complexity and the continued assumption that native arithmetic is essentially negligible in cost in comparison to the multiplication of extended integers. In one respect, to be addressed below, we are overworking this assumption to the point that what we present here must be regarded as a largely theoretical analysis of the degree to which extended multiplication may be controlled via sieving with native arithmetic.

The key is to refine the notion of multiplicative complexity by adjusting for the number of nonzero bits in the operands. Roughly speaking, an integer a (ordinary or extended) will typically have its bits equally distributed between ones and zeroes. Numbers with fewer nonzero bits require much less work to multiply—much more shifting and much less adding—so, if we can favor such *light-weight integers* (i.e., integers with relatively small Hamming distance from 0), the efficiency of all associated multiplications should improve correspondingly. Indeed, if $r(a)$ represents the ratio of the number of nonzero bits of a to its bit length, then an appropriate adjustment to the multiplicative complexity of the product ab is simply

$$\mathrm{MCW}(a,b) = 4r(a)r(b)\cdot\mathrm{MC}(a,b)$$

For the remainder of this paper, we shall be concerned with cases where the weight of only one of the factors is controlled, and so this formula reduces to $\mathrm{MCW}(a,b) = 2r(a)\cdot\mathrm{MC}(a,b)$ where, say, the first factor is controlled.

With this revision in mind, we would begin by sieving and storing in advance a large number of light-weight candidate (prime) moduli. The point here is that while the final sequence (or even set) of moduli cannot be chosen in advance if we are to sieve again on the U_j in step (ii) above, the accumulation of partial products required by Garner's algorithm can now be executed at a reduced cost. Nonetheless, one can anticipate two difficulties with this approach. First, while we can take our time sieving a set of light moduli, as the bit length of a random integer increases, the chances of finding one with a significant surfeit of zero bits correspondingly decreases. (The chances of getting only four tails in ten tosses of a fair coin is much larger than the chances of getting only forty tails in one hundred tosses of that same coin.) Second, to exploit the comparative speed of sieving for the U_j we are limited in the search for light moduli by the native precision of the system on which the algorithm is to run. The tests we report on below were executed on a standard-issue 64-bit PC and implemented in Mathematica.

A Few Trial Runs

Before discussing our modest tests, we should mention four auxiliary parameters implicit in the execution of this algorithm: the maximum bit-weight for the candidate moduli, the lower-bound cut-off for the U_j sieve, the starting point for the candidate moduli search and the candidate moduli search limit. All our reported runs were done with a limit of five nonzero bits for each candidate modulus (excluding the phi moduli). This provided a rich enough set of moduli to work with numbers of a few thousand digits. A higher limit would, of course, extend our range. The U_j-sieve cut-off was held to three; this means that if the sieve loop for m_j ever produced a result of bit weight three or lower, further searching was aborted. Since we were interested in the theoretical performance of our method, we set this parameter so low that we were essentially finding the minimum possible bit length for U_j at each step. Recognizing that we do not account for native arithmetic, this is cost-free in our set of experiments, but certainly it would have to be reckoned with if the approach were to be made practical. Finally, both the lower bound for the moduli search and the search limit became experimental parameters, and we found it most convenient to set the latter in terms of a power z of the exponential $x = b^w$. The search limit, which would seem enormous, is not directly a bound on the size of the moduli under consideration, but rather limits from below the product of the moduli that survive the bit-weight test. (The column header for this number in the subsequent tables is "search ext exp," the search extension exponent; the product of the candidate moduli is thus bounded from below by x^z, and z must be at least one for the CRT inversion to produce the correct answer.)

Table 3 suggests how this scheme responds to changes in the base, the exponent and other associated parameters for a few cases for which the target exponential is under 10^{1000}. The table header "light mods" designates the number of prime moduli of weight five or less in the implied range; the final set of moduli for the given calculation is thus a subset of those determined by sieving. The entry "last candidate" is the last candidate modulus found, and this is recorded to confirm that the moduli range is held under 2^{32} to avoid overflows in native 64-bit products. The "efficiency ratio" is simply the ratio of the multiplicative complexity (for extended multiplications) of our experimental scheme to that of the standard fast algorithm.

In these runs, about 4% of the total calculation is achieved via the phi moduli. Moreover, we see two trends that can be explained easily. First, a larger starting point for the moduli search tends to give better efficiency; this is because more arithmetic is packed into the native calculations with bigger numbers. Second, a longer search interval also tends to favor this scheme: the greater the number of candidate moduli, the more chance we have of finding light-weight U_j for Garner's algorithm. For this set of experiments, we needed a good number of good-sized candidates to begin to see any efficiency, and, even so, these results do not account for the (modular and native) overhead of sieving for the U_j.

Table 4 takes us into a range of results in excess of 10^{2000}, and the phi moduli play a correspondingly smaller role. Since the exponential results are already much larger, we need not take the search limit extension exponent so high to begin to see the effectiveness of double sieving: we find many light moduli for modest extension exponents, with more rapid convergence in overall efficiency.

Further Work

Faster algorithms for sieving on the bit weights of the candidate moduli are certainly tractable and might far exceed what we have used here, but then this is not at all the focus of these experiments. The difficulty remains in, say, for a fixed exponent w, finding some better solution to the problem of dynamically sieving on the U_j. Finer results from the experiments, which we have not presented in the last two tables, show that total multiplicative complexity is roughly evenly divided between required accumulation of the partial products of the moduli and the final step in Garner's algorithm. At present, we see no prospects for improving the latter calculation beyond what we have presented here, but possibly the candidate moduli might be chosen in a way that limits the bit weights of the partial products rather than the moduli themselves. Early attempts along these lines have not yielded any notable success, and we hope that fresher eyes than ours can spot the trick.

b	w	$\log_{10}(b^w)$	phi mod count	$\log_{10}(m_1)$	lower bound search	search ext exp	light mods	last candidate	efficiency ratio
41	252	406.422	24	24.262	1.00E+06	8	511	4,341,769	1.015
41	252	406.422	24	24.262	1.00E+06	12	749	8,527,873	1.022
41	252	406.422	24	24.262	1.00E+06	16	979	16,846,853	1.002
41	252	406.422	24	24.262	1.00E+06	20	1,199	33,571,849	0.974
41	252	406.422	24	24.262	1.00E+06	24	1,415	42,206,209	0.974
41	252	406.422	24	24.262	4.00E+06	8	476	11,272,193	1.031
41	252	406.422	24	24.262	4.00E+06	12	701	20,988,161	0.992
41	252	406.422	24	24.262	4.00E+06	16	917	34,603,033	0.964
41	252	406.422	24	24.262	4.00E+06	20	1,130	67,133,953	0.961
41	252	406.422	24	24.262	4.00E+06	24	1,337	85,983,241	0.927
41	252	406.422	24	24.262	1.60E+07	8	438	35,135,489	0.968
41	252	406.422	24	24.262	1.60E+07	12	649	67,248,161	0.978
41	252	406.422	24	24.262	1.60E+07	16	855	100,794,433	0.952
41	252	406.422	24	24.262	1.60E+07	20	1,056	135,331,969	0.961
41	252	406.422	24	24.262	1.60E+07	24	1,254	268,435,723	0.909
41	252	406.422	24	24.262	6.40E+07	8	408	134,252,609	0.965
41	252	406.422	24	24.262	6.40E+07	12	608	153,092,609	0.943
41	252	406.422	24	24.262	6.40E+07	16	802	268,961,801	0.954
41	252	406.422	24	24.262	6.40E+07	20	995	335,806,529	0.939
41	252	406.422	24	24.262	6.40E+07	24	1,182	537,673,729	0.917
41	504	812.843	30	32.381	1.00E+06	8	978	16,845,313	0.967
41	504	812.843	30	32.381	1.00E+06	12	1,414	42,205,217	0.973
41	504	812.843	30	32.381	1.00E+06	16	1,826	134,226,949	0.961
41	504	812.843	30	32.381	1.00E+06	20	2,222	268,566,817	0.975
41	504	812.843	30	32.381	1.00E+06	24	2,602	537,165,833	0.975
41	504	812.843	30	32.381	4.00E+06	8	916	34,603,013	0.962
41	504	812.843	30	32.381	4.00E+06	12	1,336	85,196,801	0.942
41	504	812.843	30	32.381	4.00E+06	16	1,735	201,719,809	0.942
41	504	812.843	30	32.381	4.00E+06	20	2,119	536,871,233	0.935
41	504	812.843	30	32.381	4.00E+06	24	2,491	805,310,977	0.942
41	504	812.843	30	32.381	1.60E+07	8	854	100,704,257	0.950
41	504	812.843	30	32.381	1.60E+07	12	1,253	268,435,649	0.950
41	504	812.843	30	32.381	1.60E+07	16	1,636	536,879,621	0.919
41	504	812.843	30	32.381	1.60E+07	20	2,007	872,448,001	0.919
41	504	812.843	30	32.381	1.60E+07	24	2,366	1,212,284,929	0.920
41	504	812.843	30	32.381	6.40E+07	8	802	268,961,801	0.953
41	504	812.843	30	32.381	6.40E+07	12	1,181	537,661,441	0.920
41	504	812.843	30	32.381	6.40E+07	16	1,548	1,073,881,093	0.917
41	504	812.843	30	32.381	6.40E+07	20	1,906	2,147,483,777	0.891
41	504	812.843	30	32.381	6.40E+07	24	2,254	2,233,466,881	0.891
47	504	842.737	30	32.381	1.00E+06	8	1,011	17,072,257	0.991
47	504	842.737	30	32.381	1.00E+06	12	1,460	67,108,913	0.951
47	504	842.737	30	32.381	1.00E+06	16	1,885	134,348,801	0.951
47	504	842.737	30	32.381	1.00E+06	20	2,292	270,533,633	0.944
47	504	842.737	30	32.381	4.00E+06	8	948	35,127,809	0.972
47	504	842.737	30	32.381	4.00E+06	12	1,380	102,760,961	0.938
47	504	842.737	30	32.381	4.00E+06	16	1,792	268,443,697	0.926
47	504	842.737	30	32.381	4.00E+06	20	2,188	536,973,313	0.926
47	504	842.737	30	32.381	1.60E+07	8	884	134,217,773	0.964
47	504	842.737	30	32.381	1.60E+07	12	1,295	268,456,961	0.941
47	504	842.737	30	32.381	1.60E+07	16	1,691	537,071,617	0.938
47	504	842.737	30	32.381	1.60E+07	20	2,073	1,073,774,657	0.914
47	504	842.737	30	32.381	6.40E+07	8	830	270,532,609	0.949
47	504	842.737	30	32.381	6.40E+07	12	1,222	540,016,769	0.923
47	504	842.737	30	32.381	6.40E+07	16	1,601	1,075,052,609	0.915
47	504	842.737	30	32.381	6.40E+07	20	1,970	2,147,549,219	0.885

Table 3. Three sets of exponentials with varied light-moduli search limits.

b	w	$\log_{10}(b^w)$	phi mod count	$\log_{10}(m_1)$	lower bound search	search ext exp	light mods	last candidate	efficiency ratio
101	1,008	2020.360	36	40.080	1.60E+07	1.0	272	33,556,673	1.125
101	1,008	2020.360	36	40.080	1.60E+07	1.5	407	34,603,081	1.033
101	1,008	2020.360	36	40.080	1.60E+07	2.0	540	50,332,673	0.993
101	1,008	2020.360	36	40.080	1.60E+07	2.5	669	67,502,081	0.976
101	1,008	2020.360	36	40.080	1.60E+07	3.0	798	79,691,809	0.961
101	1,008	2020.360	36	40.080	1.60E+07	3.5	924	134,225,929	0.956
101	1,008	2020.360	36	40.080	1.60E+07	4.0	1,048	135,267,329	0.946
101	1,008	2020.360	36	40.080	1.60E+07	4.5	1,172	167,772,161	0.944
101	1,008	2020.360	36	40.080	1.60E+07	5.0	1,293	268,455,953	0.937
101	1,008	2020.360	36	40.080	1.60E+07	5.5	1,413	272,630,021	0.930
101	1,008	2020.360	36	40.080	1.60E+07	6.0	1,532	310,378,753	0.927
171	1,008	2250.860	36	40.080	1.60E+07	1.0	303	33,571,873	1.134
171	1,008	2250.860	36	40.080	1.60E+07	1.5	452	35,653,637	1.019
171	1,008	2250.860	36	40.080	1.60E+07	2.0	599	67,117,097	0.980
171	1,008	2250.860	36	40.080	1.60E+07	2.5	743	71,303,171	0.956
171	1,008	2250.860	36	40.080	1.60E+07	3.0	885	134,217,779	0.969
171	1,008	2250.860	36	40.080	1.60E+07	3.5	1,023	134,520,833	0.949
171	1,008	2250.860	36	40.080	1.60E+07	4.0	1,161	151,388,161	0.951
171	1,008	2250.860	36	40.080	1.60E+07	4.5	1,296	268,460,033	0.927
171	1,008	2250.860	36	40.080	1.60E+07	5.0	1,430	274,726,913	0.928
171	1,008	2250.860	36	40.080	1.60E+07	5.5	1,563	339,804,161	0.919
171	1,008	2250.860	36	40.080	1.60E+07	6.0	1,692	537,133,057	0.927

Table 4. A pair of larger exponentials with a more refined granularity in the search limit extension exponent.

Conflicts of Interest

There are no conflicts of interest associated with this work.

Funding Statement

This work was funded entirely by the author’s home institution, Claremont McKenna College.

Acknowledgments

There are no further acknowledgements to be made.

References

1. Gordon, Daniel M. “A survey of fast exponentiation methods.” *Journal of Algorithms*, Vol. 27, No. 1 (April 1998), pp. 129–146.

2. Bernstein, Daniel J. “Detecting perfect powers in essentially linear time.” *Mathematics of Computation*, Vol. 67, No. 223, July 1988, pp. 1253–1283.

3. Tucker, Alan. *Applied Combinatorics* (Second Edition), John Wiley & Sons, New York, 1984.

4. Lang, Serge. *Algebra* (Revised Third Edition). Springer Graduate Texts in Mathematics 211, New York, 2002.

5. Knuth, Donald E. *The Art of Computer Programming*, Volume 2: *Semi numerical Algorithms* (Second Edition). Addison-Wesley, Reading, Massachusetts, 1981.

6. Cross, James T. “The Euler φ-function in the Gaussian integers.” The American Mathematical Monthly, Vol. 90, No. 8 (Oct. 1983), pp. 518–528.