Some mysteries of multiplication, and how to generate random factored integers

Richard P. Brent
Australian National University

6 February 2015

joint work with
Carl Pomerance

Presented at Hong Kong Baptist University

Abstract

Let $M(n)$ be the number of distinct entries in the multiplication table for integers $< n$. The order of magnitude of $M(n)$ was established in a series of papers, starting with Erdős (1950) and ending with Ford (2008), but an asymptotic formula is still unknown. After describing some of the history of $M(n)$ I will consider some algorithms for estimating $M(n)$ accurately for large n. This naturally leads to consideration of algorithms, due to Bach (1985–88) and Kalai (2003), for generating random factored integers.

Outline

- History
- Exact computation - serial and parallel
- Approximate computation - Bernoulli and product trials
- Avoiding factoring - algorithms of Bach and Kalai
- Counting divisors
- Avoiding primality testing
- Numerical results

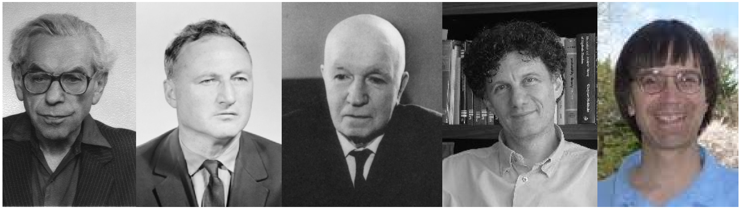
The multiplication table for $n = 8$

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7
0	2	4	6	8	10	12	14
0	3	6	9	12	15	18	21
0	4	8	12	16	20	24	28
0	5	10	15	20	25	30	35
0	6	12	18	24	30	36	42
0	7	14	21	28	35	42	49

We've included the border of zeroes but not the row and column corresponding to multiplication by $n = 8$. The set of distinct entries is shown in blue.

The function $M(n)$ is the number of distinct entries in the $n \times n$ multiplication table: $M(8) = 26$ in the example shown. We say that $M(n)$ is the *size* of the table.

The cast



Erdős Linnik Vinogradov Tenenbaum Ford

Quiz: which of these knew their multiplication tables?

Answer: none of them!

At least, none could give an asymptotic formula for the size, though Ford at least knew the correct order of magnitude of the size (number of distinct entries).

Notation

$f \sim g$ if $\lim_{x\to\infty} f(x)/g(x) = 1$.

$f = o(g)$ if $\lim_{x\to\infty} f(x)/g(x) = 0$ (Landau).

$f = O(g)$ if, for some constant K and all sufficiently large x, $|f(x)| \le K|g(x)|$ (Bachmann/Landau).

$f \ll g$ means the same as $f = O(g)$ (Vinogradov).

$f \gg g$ means the same as $g \ll f$ (Vinogradov) (sometimes written as $f = \Omega(g)$, but we won't use this).

$f \asymp g$ means that $f \ll g$ and $f \gg g$. (sometimes written as $f = \Theta(g)$, but we won't use this).

$\lg(x)$ means $\log_2(x)$, $\ln(x)$ means the natural logarithm.

$\log(x)$ means a logarithm to any fixed base.

History

There is an easy lower bound

$$M(n) \geq \sum_{\text{prime } p < n} p \gg \frac{n^2}{\ln n}\,.$$

Erdős (1955) gave an upper bound $M(n) = o(n^2)$ as $n \to \infty$. After some prodding by Linnik and Vinogradov, he proved [Erdős, 1960] that

$$M(n) = \frac{n^2}{(\ln n)^{c+o(1)}} \text{ as } n \to \infty,$$

where $$c = 1 - \frac{1 + \ln\ln 2}{\ln 2} \approx 0.0861$$

is a mysterious positive constant.

Tenenbaum (1984) partially clarified the "error term" $(\ln n)^{o(1)}$ but did not give its exact order of magnitude.

Recent history

Ford (2008) got the exact order-of-magnitude

$$M(n) \asymp \frac{n^2}{(\ln n)^c (\ln \ln n)^{3/2}}\,, \tag{1}$$

where $c \approx 0.0861$ is as in Erdős's result.

In other words, there exist positive constants c_1, c_2 such that

$$c_1 g(n) \le M(n) \le c_2 g(n)$$

for all sufficiently large n, where $g(n)$ is the RHS of (1).

Ford did not give explicit values for c_1, c_2.

Asymptotic behaviour still unknown

We still do not know if there exists

$$K = \lim_{n\to\infty} \frac{M(n)(\ln n)^c(\ln\ln n)^{3/2}}{n^2}\,,$$

or have any good estimate of the value of K (if it exists).

Ford's result only shows that the $\liminf$ and $\limsup$ are positive and finite.

Area-time complexity of multiplication

In 1981, H. T. Kung and I considered how much area A and time T are needed to perform n-bit binary multiplication in a model of computation that was meant to be realistic for VLSI circuits. We proved an "area-time" lower bound

$$AT \gg n^{3/2},$$

or more generally, for all $\alpha \in [0, 1]$,

$$AT^{2\alpha} \gg n^{1+\alpha}.$$

To prove this we needed a lower bound on $M(n)$. The easy bound $M(n) \gg n^2 / \ln n$ was sufficient. However, we could have improved the constants in our result if $M(n) \gg n^2 / \ln \ln n$, and we made this conjecture based on numerical evidence (next slide). Erdős wrote to me pointing out that he had disproved our conjecture (in a paper written in Russian).

Excerpt from Table II of Brent and Kung (1981)

$$n = 2^w, \quad M^*(n) = \frac{n^2}{0.71 + \lg\lg n}, \quad M^{**}(n) = \frac{n^2}{(\ln n)^c}.$$

w	$M(n)$	$M(n)/M^*(n)$	$M(n)/M^{**}(n)$
10	259,768	0.998846	0.2927
11	1,004,348	0.998392	0.2852
12	3,902,357	0.999002	0.2791
13	15,202,050	0.999089	0.2737
14	59,410,557	0.999788	0.2691
15	232,483,840	0.999637	0.2649
16	911,689,012	0.999788	0.2611
17	3,581,049,040	1.000005	0.2577

On the basis of the first 3 columns of this table, B&K conjectured that $M(n) \sim n^2/\lg\lg n$. This is wrong as it contradicts the result of Erdős. Moral: it is hard to tell the difference between $\lg\lg n$ and $(\ln n)^{c+o(1)}$ numerically.

Exact computation of $M(n)$ – sieving

It is easy to write a program to compute $M(n)$ for small values of n. We need an array A of size n^2 bits, indexed 0 to $n^2 - 1$, which is initially cleared. Then, using two nested loops, set

$$A[i \times j] \leftarrow 1 \ \text{ for } \ 0 \le i \le j < n.$$

Finally, count the number of bits in the array that are 1 (or sum the elements of the array). The time and space requirements are both of order n^2.

The inner loop of the above program is essentially the same as the inner loop of a program that is sieving for primes. Thus, the same tricks can be used to speed it up. For example, multiplications can be avoided as the inner loop sets bits indexed by an arithmetic progression.

Exact computation – partitioning

If the memory requirement of n^2 bits is too large, the problem can be split into pieces. For given $[a_k, a_{k+1}) \subseteq [0, n^2)$, we can count the products $ij \in [a_k, a_{k+1})$. Covering $[0, n^2)$ by a set of disjoint intervals $[a_0, a_1), [a_1, a_2), \ldots$, we can split the problem into as many pieces as desired.

There is a certain startup cost associated with each interval, so the method becomes inefficient if the interval lengths are smaller than n.

A parallel program can easily be implemented if each parallel process handles a separate interval $[a_k, a_{k+1})$.

The sequence OEIS A027417

The *Online Encyclopedia of Integer Sequences* (OEIS) sequence A027417 is defined by $a_n = M(2^n)$. Until recently only $a_0, \ldots, a_{12}$ were listed, although $a_1, \ldots, a_{17}$ had been published by Brent and Kung (1981).

Using a parallel program as outlined on the previous slide, we recently extended the computation to a_{25}.

n	a_n
1	2
2	7
3	26
4	90
...	...
23	13433148229639
24	53092686926155
25	209962593513292

The scaled sequence

In view of Ford's result

$$M(n) \asymp \frac{n^2}{(\ln n)^c (\ln\ln n)^{3/2}},$$

we define

$$b_n := 4^n / M(2^n) \;\asymp\; n^c (\ln n)^{3/2},$$

and

$$f_n := \frac{b_n}{n^c (\ln n)^{3/2}} \;\asymp\; 1.$$

n	b_n	b_n/n^c	$f_n = \frac{b_n}{n^c(\ln n)^{3/2}}$
5	3.0118	2.622	1.284
10	4.0366	3.311	0.948
15	4.6186	3.658	0.821
20	5.0331	3.889	0.750
25	5.3624	4.065	0.704

It's not clear what $\lim_{n\to\infty} f_n$ is (if it exists).

Monte Carlo computation

We can estimate $M(n)$ using a Monte Carlo method. Recall that

$$M(n) \;=\; \#S_n, \;\; S_n = \{ij : 0 \le i < n,\; 0 \le j < n\}.$$

Bernoulli trials

We can generate a random integer $x \in [0, n^2)$, and count a *success* if $x \in S_n$. Repeat several times and estimate

$$\frac{M(n)}{n^2} \approx \frac{\#\text{successes}}{\#\text{trials}}\,.$$

To check if $x \in S_n$ we need to find some of the divisors of x, which probably requires the prime factorisation of x (there is no obvious algorithm that is much more efficient than finding the prime factors of x).

Monte Carlo computation - alternative method

There is another Monte Carlo algorithm, using what we call *product trials*.

Generate random integers $x, y \in [0, n)$. Count the number $\nu = \nu(xy)$ of ways that we can write $xy = ij$ with $i < n$, $j < n$. Repeat several times, and estimate

$$\frac{M(n)}{n^2} \approx \frac{\sum 1/\nu}{\#\text{trials}} .$$

This works because $z \in S_n$ is sampled at each trial with probability $\nu(z)/n^2$, so the weight $1/\nu(z)$ is necessary to give an unbiased estimate of $M(n)/n^2$.

To compute $\nu(xy)$ we need to find the divisors of xy.

Note that $x, y < n$, whereas for Bernoulli trials $x < n^2$, so the integers considered in product trials are generally smaller than those considered in Bernoulli trials.

Comparison

For Bernoulli trials, $p = M(n)/n^2$ is the probability of a success, and the distribution after T trials has mean pT, variance $p(1-p)T \approx pT$.

For product trials, we know $E(1/\nu) = M(n)/n^2 = p$, but we do not know $E(1/\nu^2)$ theoretically. We can estimate it from the sample variance.

It turns out that, for a given number T of trials, the product method has smaller expected error (by a factor of 2 to 3 in typical cases).

This is not the whole story, because we also need to factor x (for Bernoulli trials) or xy (for product trials), and then find (some of) their divisors.

For large n, the most expensive step is factoring, which is easier for product trials because the numbers involved are smaller.

Avoiding factoring large integers

We can avoid the factoring steps by generating random integers together with their factorisations, using algorithms due to Bach (1988) or Kalai (2003).

Bach's algorithm is more efficient than Kalai's, but also much more complicated, so I will describe Kalai's algorithm and refer you to Bach's paper, or the book *Algorithmic Number Theory* (by Bach and Shallit), for a description of Bach's algorithm.

Kalai's algorithm

Input: Positive integer n.

Output: A random integer r, uniformly distributed in $[1, n]$, and its prime power factorisation.

1. Generate a sequence $n = s_0 \geq s_1 \geq \cdots \geq s_\ell = 1$ by choosing s_{i+1} uniformly in $[1, s_i]$ until reaching 1.
2. Let r be the product of all the prime $s_i,\ i > 0$.
3. If $r \leq n$, output r and its prime factorisation with probability r/n, otherwise restart at step 1.

Kalai's algorithm clearly outputs an integer $r \in [1, n]$ and its prime factorisation, but why is r uniformly distributed in $[1, n]$? The answer is the *acceptance-rejection* step 3.

Correctness of Kalai's algorithm

Kalai shows that, if

$$R = \prod_{p \text{ prime},\ p \le n} p^{\alpha_p},$$

then (after step 2)

$$\mathbb{P}\mathrm{rob}[r = R] = \frac{\mu_n}{R},$$

where

$$\mu_n = \prod_{p \text{ prime},\ p \le n} (1 - 1/p).$$

If $1 \le r \le n$, then step 3 accepts r with probability r/n, so the probability of outputting r at step 3 is proportional to

$$\frac{\mu_n}{r}\frac{r}{n} = \frac{\mu_n}{n},$$

which is independent of r. Thus, the output is uniformly distributed in $[1, n]$.

The expected running time

The running time is dominated by the time for primality tests.
The expected number of primality tests is H_n/μ_n, where

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = \ln n + \gamma + O\left(\frac{1}{n}\right) \cdot$$

By a theorem of Mertens (1874),

$$\frac{1}{\mu_n} \sim e^{\gamma} \ln n,$$

so the expected number of primality tests is

$$\sim e^{\gamma}(\ln n)^2.$$

Bach's algorithm

Bach's algorithm requires prime power tests which are (slightly) more expensive than primality tests. However, it is possible to modify the algorithm so that only primality tests are required. (This is what we implemented.)

Bach's algorithm is more efficient than Kalai's – the expected number of primality tests is of order $\log n$. The reason is that Bach's algorithm generates factored integers uniform in $(n/2, n]$ rather than $[1, n]$, which makes the acceptance/rejection process more efficient.

We can generate integers in $[1, n]$ by calling Bach's algorithm appropriately. First choose an interval $(m/2, m] \subseteq [1, n]$ with the correct probability $\lceil m/2 \rceil / n$, then call Bach's algorithm.

Primality testing

For large n, the main cost of both Bach's algorithm and Kalai's algorithm is the primality tests.

Since we are using Monte Carlo algorithms, it seems reasonable to use the Miller-Rabin probabilistic primality test, which has a nonzero (but tiny) probability of error, rather than a much slower "guaranteed" test such as the polynomial-time deterministic test of Agrawal, Kayal and Saxena (AKS), or the randomised but error-free ("Las Vegas") elliptic curve test (ECPP) of Atkin and Morain.

The Miller-Rabin test makes it feasible to use Bach's or Kalai's algorithm for n up to say 2^{1000}.

Divisors

An integer

$$x = \prod p_i^{\alpha_i}$$

has

$$d(x) = \prod (\alpha_i + 1)$$

distinct divisors, each of the form $\prod p_i^{\beta_i}$ for $0 \le \beta_i \le \alpha_i$.

We do not need all the divisors of the the random integers x, y that occur in our Monte Carlo computation. We only need the divisors in a certain interval.

We'll consider the algorithms using Bernoulli and product trials separately.

Bernoulli and product trials

Bernoulli trials

For Bernoulli trials, we merely need to know if a given $x < n^2$ has a divisor $d < n$ such that $x/d < n$, i.e. $x/n < d < n$. Thus, given n and $x \in [1, n)$, it is enough to compute the divisors of x in the interval $(x/n, n)$ until we find one, or show that there are none.

Product trials

For product trials we generate random (factored) $x, y < n$ and need (some of) the divisors of xy. We can easily compute the prime-power factorisation of $z := xy$ from the factorisations of x and y. We then need to count the divisors of z in the interval $(z/n, n)$.

Cost of Bernoulli and product trials

An integer $x \in [1, n^2)$ has on average

$$\sim \ln n^2 = 2 \ln n$$

divisors [Dirichlet]. This is relevant for Bernoulli trials.

However, for product trials, our numbers $z = xy$ have on average $\gg (\ln n)^2$ divisors, because x and y have on average $\sim \ln n$ divisors [Dirichlet].

Thus, the divisor computation for product trials is more expensive than that for Bernoulli trials.

Counting divisors in an interval

We can count the divisors of x in a given interval $[a, b]$ faster than actually computing all the divisors in this interval, by using a “divide and conquer” approach.

Here is an outline. Write $x = uv$ where $(u, v) = 1$ and u, v have about equal numbers of divisors. Find all the divisors of v and sort them. Then, for each divisor d of u, compute bounds $a \le d' \le b$ for relevant divisors d' of v, and search for a and b in the sorted list, using a binary search.

The expected running time is roughly (ignoring $\ln \ln n$ factors) proportional to the mean value of $d(x)^{1/2}$ over $x \le n$. By a result of Ramanujan [see Montgomery and Vaughan], this is $\asymp (\ln n)^{\alpha}$, where $\alpha = \sqrt{2} - 1 \approx 0.4142$.

Thus, for Bernoulli trials the cost is $O(\log^{\alpha} n)$
and for product trials the cost is $O(\log^{2\alpha} n)$.

Avoiding primality testing

The times for counting divisors appear to be dominated by the time for Miller-Rabin primality testing – but we shall see that most of the primality tests can be avoided, without significant loss of accuracy.

Consider implementing Kalai's algorithm for very large inputs $N = 2^n - 1$. To avoid storing very large integers $x \in [1, N]$ we might store (an approximation to) $\ln x$ instead.

Recall that Kalai's algorithm starts by generating a sequence $N = s_0 \geq s_1 \geq \cdots$ until finding a prime p (or 1).
What is the distribution of $\ln p$?

If we assume that the density of primes near x is $1/\ln x$, then $\ln p/\ln s_0$ is uniformly distributed (up to a discretisation error of order $1/x$).

Is the density assumption reasonable?

We know that the density assumption is false in sufficiently short intervals (Maier's theorem). However, the assumption seems reasonable in our application, provided x is sufficiently large, say $x > 1/\varepsilon^2$ if the expected error of the Monte Carlo algorithm is ε.

The assumption is consistent with a theorem of Vershik: if k is fixed and $n \to \infty$, then $\ln p_k / \ln p_{k+1}$ is asymptotically uniformly distributed, where the prime factors of n are $p_1 \le p_2 \le \cdots$.

Approximate Kalai algorithm

Input: Large positive integer N, represented by $\ell := \ln N$.

Output: A random integer r, uniformly distributed in $[1, N]$, and its prime power factorisation, also represented logarithmically.

```
1. $S \leftarrow \ell$
2. $S \leftarrow S \times \mathrm{uniform}(0, 1)$
3. if $S \leq \mathrm{crossover}$ then
         $S \leftarrow \mathrm{crossover}$
         start usual Kalai algorithm in $[1, S]$
   else
         treat $S$ like $\ln(p)$ in usual Kalai algorithm
         go to step 2.
```

Now we only need to do real primality tests for numbers that are smaller than the crossover, taking time $O(1)$.

There is an analogous “approximate Bach” algorithm.

How far can we estimate $M(n)$?

Using the "exact" Kalai or Bach algorithms with Miller-Rabin probabilistic primality tests and Bernoulli or product trials, we can estimate $M(n)$ for $n \le 2^{1000}$ before the algorithm becomes impractically slow.

Using the approximate Bach algorithm with Bernoulli trials, we can extend the domain to $n \le 2^{5\times 10^8}$ and still get reasonable accuracy (say three significant decimal digits).
The speed is about one trial per second on a 2GHz machine.

Numerical results

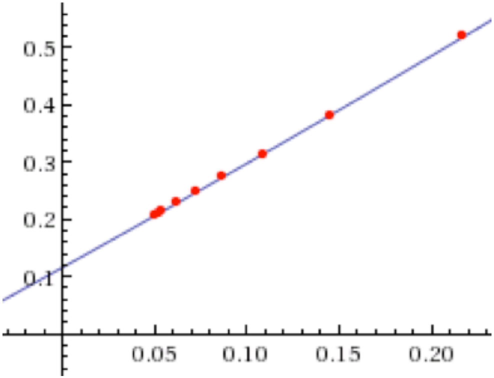
$$b_n := \frac{4^n}{M(2^n)}, \quad f_n := \frac{b_n}{n^c(\ln n)^{3/2}} \asymp 1.$$

n	b_n	b_n/n^c	$f_n = \frac{b_n}{n^c \ln^{3/2} n}$
10	4.0366	3.311	0.948
10^2	7.6272	5.131	0.519
10^3	12.526	6.912	0.381
10^4	19.343	8.755	0.313
10^5	28.74	10.67	0.273
10^6	41.6	12.67	0.247
10^7	59.0	14.74	0.228
10^8	82.7	16.94	0.214
2×10^8	90.9	17.54	0.210
5×10^8	103.4	18.44	0.206

The value of $\lim_{n\to\infty} f_n$ is not immediately clear from this table!

Least squares quadratic fit

Fitting f_n by a quadratic in $x = (\ln n)^{-1}$ to the data for $n = 10^2, 10^3, \ldots, 5 \times 10^8$ (as in the previous table) gives $f_n \approx 0.1157 + 1.7894x + 0.2993x^2$.



Conclusion

On the basis of the numerical results, a plausible conjecture is

$$b_n = 4^n / M(2^n) \sim c_0 n^c (\ln n)^{3/2}, \quad c_0 \approx 0.1157,$$

which suggests

$$M(N) \sim K \frac{N^2}{(\ln N)^c (\ln \ln N)^{3/2}}$$

with

$$K = \frac{(\ln 2)^c}{c_0} \approx 8.4.$$

This estimate of K might be inaccurate, since we have only taken three terms in a plausible (but not proved to exist!) asymptotic series

$$f_n \sim c_0 + c_1 / \ln n + c_2 / (\ln n)^2 + \cdots ,$$

and the first two terms are of the same order of magnitude in the region where we can estimate $M(N)$.

References

E. Bach, How to generate factored random numbers, *SIAM J. on Computing* **17** (1988), 179–193.

E. Bach and J. Shallit, *Algorithmic Number Theory*, Vol. 1, MIT Press, 1996.

R. P. Brent and H. T. Kung, The area-time complexity of binary multiplication, *J. ACM* **28** (1981), 521–534 & **29** (1982), 904.

P. Erdős, Some remarks on number theory, *Riveon Lematematika* **9** (1955), 45–48 (Hebrew).

P. Erdős, An asymptotic inequality in the theory of numbers, *Vestnik Leningrad Univ.* **15** (1960), 41–49 (Russian).

K. Ford, The distribution of integers with a divisor in a given interval, *Annals of Math.* **168** (2008), 367–433.

A. Kalai, Generating random factored numbers, easily, *J. Cryptology* **16** (2003), 287–289.

H. Maier, Primes in short intervals, *Michigan Math. J.* **32** (1985), 221–225.

H. L. Montgomery and R. C. Vaughan, *Multiplicative Number Theory I. Classical Theory*, Cambridge Univ. Press, 2007. See eqn. (2.27).

S. Ramanujan, Some formulæ in the analytic theory of numbers, *Messenger of Math.* **45** (1916), 81–84.

G. Tenenbaum, Sur la probabilité qu'un entier possède un diviseur dans un intervalle donné, *Compositio Math.* **51**(1984), 243–263 (French).

A. M. Vershik, The asymptotic distribution of factorizations of natural numbers into prime divisors, *Dokl. Akad. Nauk SSSR* **289** (1986), 269–272 (Russian).