The multiplication table, and random factored integers

Richard P. Brent
ANU

27 September 2012

joint work with
Carl Pomerance

Presented at the 56th annual meeting of the Australian Mathematical Society, Ballarat

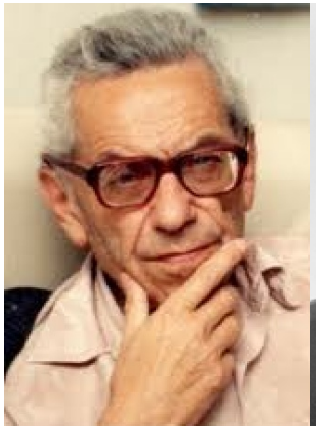
Abstract

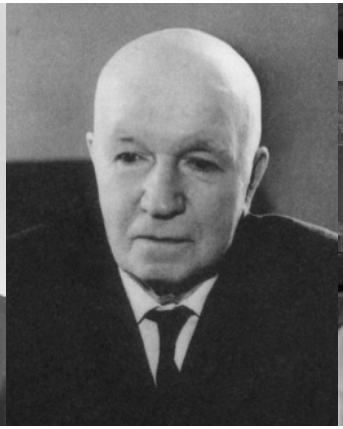
Let $M(n)$ be the number of distinct entries in the multiplication table for integers $< n$. The order of magnitude of $M(n)$ was established in a series of papers, starting with Erdős (1950) and ending with Ford (2008), but an asymptotic formula is still unknown. After describing some of the history 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.

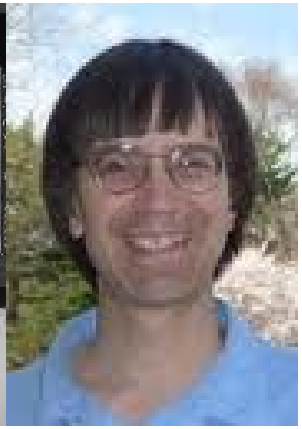
Outline

- History
- Exact computation - serial and parallel, memory problems
- Approximate computation - Bernoulli and non-Bernoulli
- Avoiding factoring - algorithms of Bach and Kalai
- Counting divisors
- Avoiding primality testing
- Numerical results

Quiz: which of these mathematicians knew their multiplication tables?







Erdős, Linnik, Vinogradov, Tenenbaum, Ford

Answer: none of them!

At least, none had a good asymptotic formula for the size of the table, though Ford at least got the right order of magnitude.

History

There is an easy lower bound $M(n) \gg n^2/\log n$, by considering $\{kp : p \text{ prime }, 1 \le k < p < n\}$.

Erdős (1955) proved that $M(n) = o(n^2)$ as $n \to \infty$ and indicated that $n^2/(\log n)^c$ for some positive c was likely.

After prodding by Linnik and Vinogradov, Erdős (1960) proved that

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

$$c = 1 - \frac{1 + \log\log 2}{\log 2} \approx 0.0861\,.$$

Tenenbaum (1984) partially clarified the $(\log n)^{o(1)}$ factor.

Recent history

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

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

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

Here $f(n) \asymp g(n)$ means that

$$f(n) \ll g(n) \ll f(n),$$

sometimes written as

$$f(n) = \Theta(g(n)).$$

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

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

for all sufficiently large n.

Asymptotic behaviour still unknown

We still do not know if there exists

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

or have any good estimate of the value of K (if it exists). Ford's result only shows that the lim inf and lim sup are positive and finite.

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 order n^2 bits, which is initially cleared. Then, using two loops, set $A[ij] \leftarrow 1$ for $0 \le i \le j < n$. Finally, count the number of bits in the array that are 1. 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 order 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 we define

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

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

n	b_n	b_n/n^c	$\frac{b_n}{n^c \log n}$	$f_n = \frac{b_n}{n^c log^{3/2} n}$
5	3.0118	2.622	1.629	1.284
10	4.0366	3.311	1.438	0.948
15	4.6186	3.658	1.351	0.821
20	5.0331	3.889	1.298	0.750
25	5.3624	4.065	1.263	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

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

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 the divisors of x, which probably requires the prime factorisation of x.

Monte Carlo computation - alternative method

There is another Monte Carlo algorithm, using what (for lack of a better name) we call non-Bernoulli 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 estimate $M(n)/n^2$ correctly.

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

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 non-Bernoulli 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 non-Bernoulli 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 non-Bernoulli trials), and then find (some of) their divisors.

For large n, the most expensive step is factoring, which is easier for non-Bernoulli 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 and refer you to Bach's paper for a description of his algorithm. There is also a description in the book *Algorithmic Number Theory* by Bach and Shallit.

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 with probability r/n, otherwise restart at step 1.

Why is Kalai's algorithm correct?

Kalai shows that

$$\text{Prob}\left[r = R := \prod_{p \le n} p^{\alpha_p}\right] = \frac{\mu_n}{R},$$

where

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

Step 3 accepts $r \le n$ with probability r/n, so the probability of outputting such an 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 $[1, n]$.

Running time

The expected number of primality tests is H_n/μ_n, where

$$H_n = 1 + 1/2 + \cdots + 1/n \sim \log n.$$

By a theorem of Mertens,

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

so the expected number of primality tests is

$$\sim e^\gamma \log^2 n.$$

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 the version that we implemented.)

Bach's algorithm is more efficient than Kalai's – the expected number of primality tests is $O(\log n)$. 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. We 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 (e.g. the deterministic test of Agrawal, Kayal and Saxena, or the elliptic curve test 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. It turns out that we only need the divisors in a certain interval. We'll consider the algorithms using Bernoulli and non-Bernoulli trials separately.

Bernoulli and non-Bernoulli 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.

Non-Bernoulli trials

For non-Bernoulli 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 non-Bernoulli trials

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

$$\sim \log n^2 = 2 \log n$$

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

However, for non-Bernoulli trials, our numbers $z = xy$ have on average $\gtrsim \log^2 n$ divisors, because x and y each have $\sim \log n$ divisors on average.

Thus the divisor computation for non-Bernoulli 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, we can 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 log log factors) proportional to the mean value of $d(x)^{1/2}$ over $x \le n$. By a result of Ramanujan [Montgomery and Vaughan, eqn. (2.27)], this is $\asymp (\log n)^{\alpha}$, where $\alpha = \sqrt{2} - 1 \approx 0.4142$.

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

Avoiding primality testing

The times for counting divisors would 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) $\log x$ instead.

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

If we assume that the density of primes near x is $1/\log x$, then $\log p/\log s_0$ 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 (*cf* Maier's theorem). However, it 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 also agrees with a theorem of Vershik, which says that if k is fixed and $n \to \infty$ then $\log p_k / \log p_{k+1}$ 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 $\ell := \log N$.

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

1. $S \leftarrow \ell$
2. $S \leftarrow S \times \text{uniform}(0, 1)$
3. if $S \leq \text{crossover}$ then
   $S \leftarrow \text{crossover}$
   start usual Kalai in $[1, S]$
   else
   treat S like $\log(p)$ in usual Kalai
   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 non-Bernoulli trials, we can estimate $M(n)$ for $n \leq 2^{1000}$ before the algorithm becomes impractically slow.

Using the approximate Bach algorithm with Bernoulli trials, we can extend the domain to $n \leq 2^{5\times 10^8}$ and still get reasonable accuracy (say 3 significant decimal digits).
The speed is about 1 trial per second on a 2GHz machine.

Numerical results

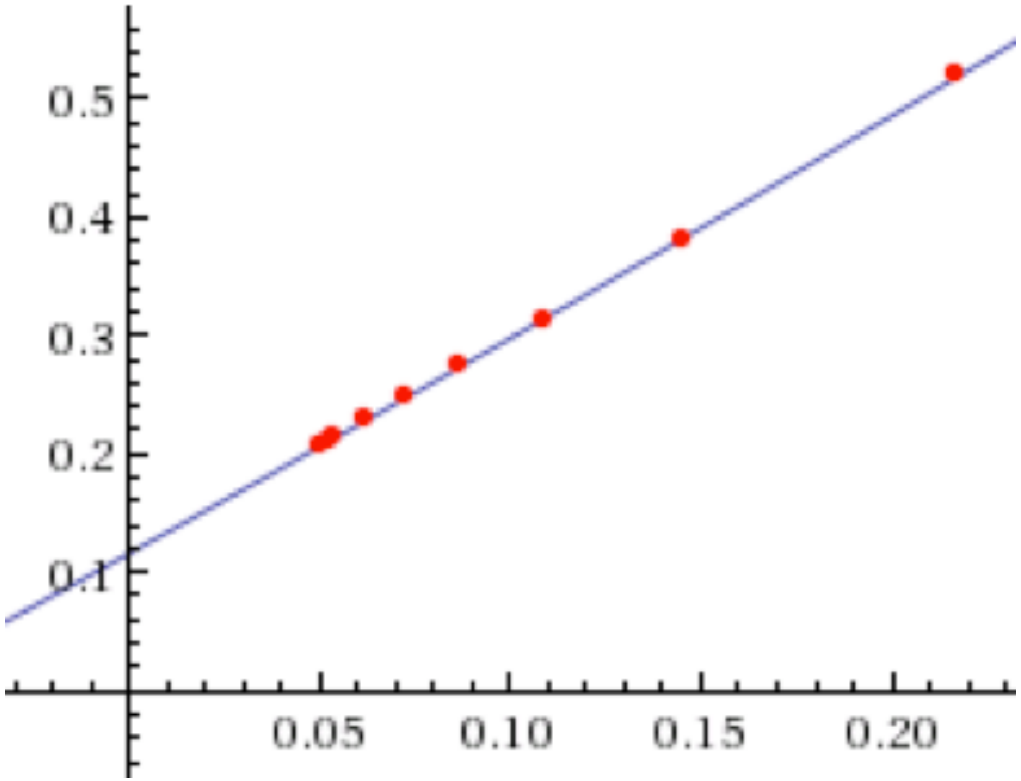
$$b_n := \frac{4^n}{M(2^n)}, \quad f_n := \frac{b_n}{n^c (\log n)^{3/2}} \asymp 1.$$

n	b_n	b_n/n^c	$\frac{b_n}{n^c \log n}$	$f_n = \frac{b_n}{n^c \log^{3/2} n}$
10	4.0366	3.311	1.438	0.948
10^2	7.6272	5.131	1.114	0.519
10^3	12.526	6.912	1.001	0.381
10^4	19.343	8.755	0.951	0.313
10^5	28.74	10.67	0.927	0.273
10^6	41.6	12.67	0.917	0.247
10^7	59.0	14.74	0.914	0.228
10^8	82.7	16.94	0.920	0.214
2×10^8	90.9	17.54	0.918	0.210
5×10^8	103.4	18.44	0.921	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 $x = (\log n)^{-1}$ 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

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

which suggests

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

with

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

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

$$f_n \sim c_0 + c_1/\log n + c_2/\log^2 n + \cdots,$$

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.

I. Damgård, P. Landrock and C. Pomerance, Average case error estimates for the strong probable prime test, *Math. Comp.* **61** (1993), 177–194.

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.

References continued

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).