

Algorithms for the Multiplication Table Problem

Richard P. Brent
Australian National University and
CARMA, University of Newcastle

19 May 2021

Collaborators



Carl Pomerance



Jonathan Webster

Copyright © 2021, R. P. Brent

Abstract

Let $M(n)$ be the number of distinct entries in the multiplication table for integers $< n$. More precisely¹,

$$M(n) := \#\{i \times j \mid 0 \leq i, j < n\}.$$

The order of magnitude of $M(n)$ was established in a series of papers, starting with Erdős (1955) and ending with Ford (2008), but an asymptotic formula is still unknown. After describing some of the history of $M(n)$ we consider some algorithms for calculating/approximating $M(n)$ for large n . This naturally leads to consideration of algorithms, due to Bach (1985–88) and Kalai (2003), for generating random factored integers.

The talk describes joint work with Carl Pomerance (Dartmouth, New Hampshire) and Jonathan Webster (Butler, Indiana). See [arXiv:1908.04251](https://arxiv.org/abs/1908.04251) for details.

¹Often a slightly different definition is used.

Outline

- ▶ History
- ▶ Two algorithms for exact computation -
naive and incremental
- ▶ A (theoretically) faster algorithm for exact computation
- ▶ Two approximate (Monte Carlo) algorithms -
Bernoulli and product trials
- ▶ Avoiding factoring - algorithms of Bach and Kalai
- ▶ Counting divisors
- ▶ Numerical results

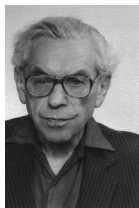
The multiplication table for $n = 8$

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 in gray borders of zeroes but not the row and column corresponding to multiplication by $n = 8$ (often the opposite convention is used). We could write rows in the reverse order. A 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



Paul Erdős



Yuri Linnik



I. M. Vinogradov



Gérald Tenenbaum



Kevin Ford

Quiz: which of these mathematicians fully understood $M(n)$?

None could give an asymptotic formula, though **Ford** gave the correct order of magnitude.

Notation

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

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

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

$f \ll g$ means that $f = O(g)$, and $f \gg g$ means that $g \ll f$.

$f \asymp g$ means that $f \ll g$ and $f \gg g$ (equivalently, $f = \Theta(g)$).

$\ln(x)$ means the natural logarithm.

$\lg(x) := \ln(x)/\ln(2)$, the logarithm to base 2.

$M(n) := \#\{i \times j \mid 0 \leq i, j < n\}$.

Note: often the slightly different definition

$\#\{i \times j \mid 0 < i, j \leq n\}$ is used.

This doesn't affect the asymptotics.

A multilingual history

There is an easy lower bound

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

Erdős (1955, in Hebrew) gave an upper bound $M(n) = o(n^2)$ as $n \rightarrow \infty$. After some encouragement by Linnik and Vinogradov, he proved (1960, in Russian) that

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

where $c = \int_1^{1/\ln 2} \frac{1}{\ln t} dt = 1 - \frac{1 + \ln \ln 2}{\ln 2} \approx 0.0861$.

In case you can't read Russian, there is a review in German.

Tenenbaum (1984, in French) clarified the “error term” $(\ln n)^{o(1)}$ but did not give its exact order of magnitude.

Erdős's first paper on $M(n)$

Here is an excerpt from Erdős's 1955 paper. The text runs right to left, but the mathematics runs left to right!

(3) נסמן ב $A(n)$ את מספר המספרים השלמים $1 \leq m \leq n^2$ הנתנים להכתב כמכפלת שני מספרים שלמים שאינם עולים על n . נוכיח ש

$$\lim_{n \rightarrow \infty} A(n)/n^2 = 0$$

כאמת, נוכיח שבשביל $0 < \alpha$ מסוים

$$A(n) = o(n^2 / (\log n)^\alpha) \quad (11)$$

נסמן ב $f(k)$ את מספר הגורמים הראשוניים של k , כשגורמים כפולים נחשבים לפי כפילותם. מסקנה ידועה של הרדי ורמנוין [4] טוענת שכמעט בשביל כל k , $f(k) = (1+o(1)) \log \log k$. כיתר דיוק, לכל ϵ קיים α כך שמספר המספרים השלמים $k \leq n$ שבשבילם $f(k) < (1-\epsilon) \log \log k$ או $f(k) > (1+\epsilon) \log \log k$ הוא $o(n / (\log n)^\alpha)$.

נחלק עתה את המספרים השלמים מצורת

$$a \cdot b, \quad 1 \leq a, b \leq n$$

לשתי מחלקות. למחלקה הראשונה שיכים המספרים השלמים שבשבילם קיימים כמות שני אי-השוויונות $f(a) > \frac{2}{3} \log \log n$ ו $f(b) > \frac{2}{3} \log \log n$. ברור כי $f(a \cdot b) > \frac{4}{3} \log \log n > (\frac{4}{3} - \epsilon) \log \log (n^2)$. מכאן לפי מסקנת הרדי ורמנוין, מספר המספרים השלמים במחלקה הראשונה הוא $o(n^2 / (\log n)^\alpha)$. בשביל המספרים השלמים במחלקה השנייה נוכל להניח כי $f(a) \leq \frac{2}{3} \log \log n$. אולם אז שוב לפי מסקנת הרדי ורמנוין, מספר הבחירות האפשריות בשביל a הוא $o(n / (\log n)^\alpha)$.

Recent history

Ford (2008, in English) got the exact order-of-magnitude

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

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

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

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

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

Ford did not give explicit values for c_1, c_2 , but they could probably be worked out from his paper.

Asymptotic behaviour

We still do not know if there exists

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

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, RB and H. T. Kung 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. On the basis of numerical evidence (next slide), we **conjectured** that $M(n) \gg n^2 / \ln \ln n$. Erdős wrote to me pointing out that he had **disproved our conjecture** (in his 1960 paper written in Russian). It was later noticed that Erdős’s 1960 proof was incorrect, but it can be fixed: see [MR603312 \(82c:10053\)](#).



RPB, HT Kung, and Geoffrey Brent
near Batemans Bay, 1975

Excerpt from Table II of Brent and Kung (1981)

$$n = 2^w, \quad M^*(n) = \frac{n^2}{0.71 + \lg \lg n}, \quad g(n) = \frac{n^2}{(\ln n)^c (\ln \ln n)^{3/2}}.$$

w	$M(n)$	$M(n)/M^*(n)$	$M(n)/g(n)$
12	3,902,357	0.999002	0.8606
13	15,202,050	0.999089	0.8922
14	59,410,557	0.999788	0.9220
15	232,483,840	0.999637	0.9490
16	911,689,012	0.999788	0.9745
17	3,581,049,040	1.000005	0.9986
28	13,023,772,682,665,849	0.997213	1.1915
30	204,505,763,483,830,093	0.996327	1.2176

On the basis of this table (excluding more recent gray entries), B&K conjectured that $M(n) \sim n^2 / \lg \lg n$. This contradicts the result of Erdős.

Moral

It is hard to tell the difference between $\ln \ln n$ and $(\ln n)^c$ numerically. For $2^6 < n < 2^{30}$, we have $(\ln n)^c \in (1.1, 1.3)$ and $\ln \ln n \in (1.4, 3.1)$.

We find numerically that

$$(\ln n)^c < \ln \ln n \quad \text{for} \quad 20 < n < 10^{5 \times 10^{18}},$$

even though

$$(\ln n)^c \gg \ln \ln n.$$

How large does n need to be in order to “see” the true asymptotic behaviour of $M(n)$?

The asymptotic region

Ford's result can be written as

$$M(n) \asymp n^2 / \Phi(n), \text{ where } \Phi(n) = (\ln n)^c (\ln \ln n)^{3/2}.$$

The factor $(\ln n)^c$ is asymptotically larger than the factor $(\ln \ln n)^{3/2}$. However, for small n , the second factor varies more rapidly than the first.

Write $x := \ln n$, $A = A(x) := x^c$, $B = B(x) := (\ln x)^{3/2}$, so $\Phi(n) = AB$.

Taking logarithmic derivatives, we have $\Phi'/\Phi = A'/A + B'/B$.

Now $|A'/A| < |B'/B|$ if $c/x < 3/(2x \ln x)$,

i.e. if $x < e^{3/2c} \approx 37,036,165$, or

$$n < e^{e^{3/2c}} \approx 2^{53,431,908}.$$

We need Monte Carlo methods to get numerical results for the region where the true asymptotic behaviour of $M(n)$ becomes evident (but first I'll mention some exact methods).

Exact computation of $M(n)$ – the naive algorithm

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

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

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.

Segmenting the sieve

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 $[a_0, a_1), [a_1, a_2), \dots$, 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 $[a_k, a_{k+1})$.

Exact computation - the incremental algorithm

The naive algorithm takes time $\asymp n^2$ to compute one value $M(n)$. If we want to tabulate $M(k)$ for $1 \leq k \leq n$, the time required is $\asymp n^3$. A more efficient approach might be to compute the differences $D(n) := M(n+1) - M(n)$.

Assuming we know $M(n)$, we need to consider the products $m \times n$, for $1 \leq m \leq n$. Let $\delta(n)$ denote the number of these products that have already occurred in the table. The number of elements that have *not* already appeared is $D(n) = n - \delta(n)$. Thus, it is sufficient to compute $\delta(n)$ in order to compute $D(n)$ and then $M(n+1)$.

Computing $\delta(n)$

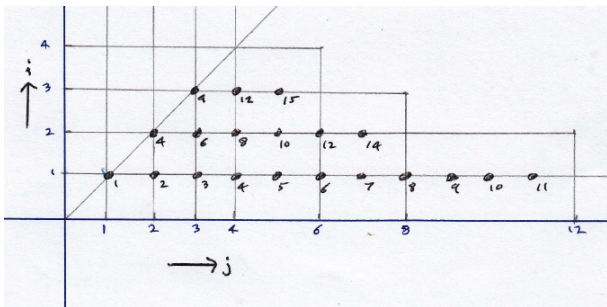
Assume we know the divisors of n (these can be computed by trial division). Suppose that $g|n$, and let $h = n/g$. By symmetry we can assume that $g \leq \sqrt{n} \leq h$.

Can we express $m \times n$ as a product that already occurred in the table? If $m = i \times j$ and $n = g \times h$, then $m \times n = ij \times gh = ih \times jg$. If $ih < n$ and $jg < n$, then the product $ih \times jg$ has already occurred. Observe that $ih < n$ iff $i < g$ and $jg < n$ iff $j < h$.

Thus, to compute $\delta(n)$, we need to count the unique products ij with $0 < i < g$ and $0 < j < n/g$, for each divisor $g \leq \sqrt{n}$ of n . This can be done by sieving in an array of size n (the naive algorithm required size n^2).

If this is implemented so that work is not duplicated where rectangles of size $g_1 \times n/g_1$ and $g_2 \times n/g_2$ overlap, then the work is bounded by the number of lattice points under the hyperbola $xy = n$. Thus, we can compute $\delta(n)$ in (worst case) time $O(n \ln n)$ and space $O(n)$.

Example: $n = 24$



If $n = 24$, the relevant divisors are $g \in \{2, 3, 4\}$.

If $g = 2$, we sieve over $(i, j) \in \{1\} \times \{1, 2, \dots, 11\}$.

If $g = 3$, we sieve over $(i, j) \in \{1^*, 2\} \times \{1^*, 2, 3, 4, 5, 6, 7\}$.

If $g = 4$, we sieve over $(i, j) \in \{1^*, 2^*, 3\} \times \{1^*, 2^*, 3, 4, 5\}$.

Starred entries give duplicates so may be omitted.

Conclude that $m \times n$ is already in the table iff

$m \in \{1, 2, \dots, 11, 12, 14, 15\}$, so $\delta(n) = 14$.

e.g. $12 \times n = 16 \times 18$, $14 \times n = 16 \times 21$, $15 \times n = 20 \times 18$.

Tabulating $M(1), \dots, M(n)$

Using the algorithm that we just described to compute the differences $D(n) = M(n+1) - M(n)$ we can compute *all* of $M(1), \dots, M(n)$ in time $O(n^2 \ln n)$ and space $O(n)$ (not counting space for the output).

This is much better than time $O(n^3)$ and space $O(n^2)$ using the naive algorithm!

Space requirements

In practice, reducing space often reduces time, because of the memory hierarchy built into modern computers.

The space requirement for the incremental algorithm can be reduced to $O(\sqrt{n})$ without changing the time bound, by splitting the sieve into $O(\sqrt{n})$ segments. It could be reduced to $O(n^{1/3}(\ln n)^{2/3})$, using ideas due to Harald Helfgott. What we have implemented is $O(\sqrt{n})$, which is fine for $n \leq 10^9$.

The distribution of divisors

Let $\tau(n)$ be the number of positive divisors of n . (τ stands for the German *Teiler*. Another notation $d(n)$.)

The mean value of $\tau(k)$ for $1 \leq k \leq n$ is $\sim \ln n$ [Dirichlet], but usually $\tau(n)$ is about $(\ln n)^{\ln 2}$.

More precisely, the *normal order* of $\ln \tau(n)$ is $\ln(2) \ln \ln n$, which is a way of saying that, for all positive ε , $\tau(n)$ is almost always in $[(\ln n)^{\ln(2)-\varepsilon}, (\ln n)^{\ln(2)+\varepsilon}]$.

Heuristically, this is because the number $\omega(n)$ of distinct *prime* divisors of n is asymptotically normal with mean and variance $\ln \ln n$ [Erdős-Kac theorem], but the number of (not necessarily prime) divisors is $2^{\omega(n)}$, assuming that n is square-free. Thus, the distribution of divisors has a long tail.

The situation is slightly more complicated if n is not square-free, but the conclusion remains the same.

OEIS A027417

The OEIS sequence A027417 is defined by $a_n = M(2^n)$.

Until recently only a_0, \dots, a_{25} were listed.

Using parallel implementations of the naive and incremental algorithms, we (RB and JW) have gone to a_{30} .

n	a_n	$4^n/a_n$
1	2	2.0000
2	7	2.2857
3	26	2.4615
4	90	2.8444
...
26	830751566970327	5.4211
27	3288580294256953	5.4779
28	13023772682665849	5.5328
29	51598848881797344	5.5860
30	204505763483830093	5.6376

The scaled sequence

Since $M(n) \asymp \frac{n^2}{(\ln n)^c (\ln \ln n)^{3/2}}$, replace n by $N = 2^n$ and define

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

n	b_n	f_n
10	4.0366	0.948
15	4.6186	0.821
20	5.0331	0.750
25	5.3624	0.704
30	5.6376	0.671

f_n appears to be monotonic decreasing. It is not clear from the table what $\lim_{n \rightarrow \infty} f_n$ is, or even if the limit exists. An estimate, using much larger values of n , is $\lim_{n \rightarrow \infty} f_n \approx 0.116$.

A subquadratic algorithm

Using the ideas of the incremental algorithm, and some optimisations that depend on whether or not n is B -smooth with $B = L(n)^{1/\sqrt{2}}$, where $L(n) := \exp(\sqrt{\ln n \ln \ln n})$, we can compute $M(n)$ in time $O(n^2/L(n)^{1/\sqrt{2}+o(1)})$.

For details, see our preprint [arXiv:1908.04251](https://arxiv.org/abs/1908.04251).

We have implemented the subquadratic algorithm for evaluation of $M(n)$. In practice, it is not competitive with the best quadratic algorithm for $n \leq 2^{30}$.

Although the time is theoretically subquadratic, the result is a little disappointing — we were hoping for something like Karatsuba's $O(n^{1.585})$, but we don't see how to obtain this. It is not clear whether we can evaluate $M(n)$ in time $O(n^\lambda)$ for any fixed $\lambda < 2$.

Monte Carlo computation

We can estimate $M(n)$ using two different Monte Carlo methods. Recall that

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

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 (but more on this later).

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 $\mathbb{E}(1/\nu) = M(n)/n^2 = p$, but we do not know $\mathbb{E}(1/\nu^2)$ 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 $n = s_0 \geq s_1 \geq \dots \geq s_\ell = 1$ 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 $1 \leq R \leq n$, then (after step 2)

$$\text{Prob}[r = R] = \frac{\mu_n}{R},$$

where

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

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

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

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

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

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. The idea of avoiding prime power tests was suggested independently by Herman Rubin. Bach's algorithm is more efficient than Kalai's – the expected number of primality tests is of order $\ln n$. 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 as well as more complicated.

We can generate integers in $[1, n]$ by calling Bach's algorithm appropriately, see [arXiv:1908.04251](https://arxiv.org/abs/1908.04251).

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 is 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 (again)

An integer

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

has

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

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

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.

Divisors in 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.

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 $a \leq d' \leq b$ 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 $\tau(x)^{1/2}$ over $x \leq n$. By a result of Ramanujan [Montgomery and Vaughan, (2.27)], this is $\asymp (\ln n)^\alpha$, where $\alpha = \sqrt{2} - 1 \approx 0.4142$.

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

Avoiding some primality testing

When n is very large, say greater than 2^{1000} , even Miller-Rabin probabilistic primality testing becomes very expensive. We can avoid primality tests on large numbers by making a plausible assumption about the density of primes in short intervals (but not so short that **Maier's theorem** applies). There is no time to discuss this in detail today. If you are interested, see the slides from my Hong Kong talk (6 February 2015), which are on my website.

Using the density assumption, we have estimated $M(n)$ with an accuracy of three or more significant digits for n up to $2^{500,000,000}$. This is in the region where the asymptotic behaviour of $M(n)$ might become evident (recall the argument we gave earlier).

Numerical results

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

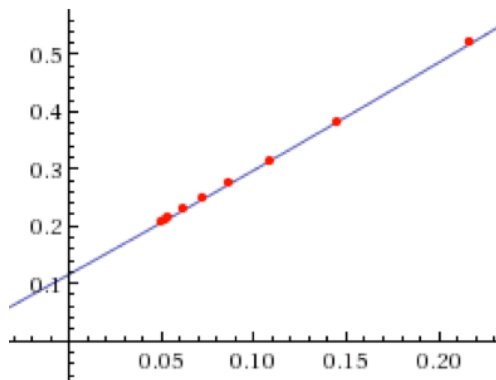
n	b_n	f_n
10	4.0366	0.948
10^2	7.6272	0.519
10^3	12.526	0.381
10^4	19.343	0.313
10^5	28.74	0.273
10^6	41.6	0.247
10^7	59.0	0.228
10^8	82.7	0.214
2×10^8	90.9	0.210
5×10^8	103.4	0.206

The value of $\lim_{n \rightarrow \infty} f_n$ is not obvious from this table!

Least squares quadratic fit

From the numerical evidence, it is plausible that f_n has an asymptotic expansion in negative powers of $\ln n$.

Fitting f_n by a quadratic in $x = (\ln n)^{-1}$ to the data for $n = 10^2, 10^3, \dots, 5 \times 10^8$ (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

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

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) asymptotic series

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

and the first two terms are of the same order of magnitude in the region where we can estimate $M(N) = M(2^n)$.

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.

[R. P. Brent, C. Pomerance, D. Purdum, J. Webster](#), Algorithms for the multiplication table, arXiv:1908.04251, 5 May 2021.

[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**, 13 (1960), 41–49 (Russian).
For a correction, see [MR603312 \(82c:10053\)](#).

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

[H. A. Helfgott](#), An improved sieve of Eratosthenes, *Math. Comp.* **89** (2020), 333–350.

[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.

[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).