Algorithms to Generate Factored Smooth Integers

Eric Bach and Jon Sorenson

University of Wisconsin-Madison, USA, bach@cs.wisc.edu (retired)
Butler University, USA, sorenson@butler.edu (not retired)

July 8, 2026

17th Algorithmic Number Theory Symposium
Rijksuniversiteit Groningen, Netherlands

Thanks to the Holcomb Awards Committee.

Outline and Definitions

Outline:

- Buchstab's Identity
- Algorithm to enumerate factored smooth numbers
- Algorithms to generate a random factored smooth number

Definitions:

- $P(n)$ is the largest prime divisor of n; $P(1) = 1$.
- n is *y-smooth* if $P(n) \leq y$.
- $\Psi(x, y)$ counts y-smooth integers $\leq x$.

Buchstab's Identity

$$\Psi(x,y) = 1 + \sum_{p \le y} \Psi(x/p, p) \tag{1}$$

$\Psi(1,1)$ $+\Psi(x/2,2)$ $+\Psi(x/3,3)$ $+\Psi(x/5,5)$ $\cdots + \Psi(x/y,y)$

This decomposes $\Psi(x,y)$ by its largest prime divisor.
Write $n = mp$ with $P(n) = p$, so m is p-smooth and $m \le x/p$.
Then

- $\Psi(x/p,p) = \Psi(x,p) - \Psi(x,p-1)$ for odd primes p.

If we choose n uniformly from $1 \ldots x$ we have:

- $P(n) = p$ with probability $\Psi(x/p,p)/\Psi(x,y)$.
- $P(n) \le p$ with probability $\Psi(x,p)/\Psi(x,y)$.
- $n = 1$ with probability $1/\Psi(x,y)$.

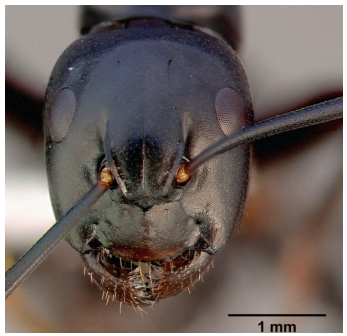
Example: $\Psi(12,3)$

$$
\begin{aligned}
\Psi(12,3) &= 1 + \Psi(12/2,2) + \Psi(12/3,3) \\
&= 1 + \Psi_2(6,2) + \Psi_3(4,3). \quad \{1\} \\
\Psi_2(6,2) &= 1 + \Psi(6/2,2) \\
&= 1_2 + \Psi_{2,2}(3,2) \\
&= 1_2 + (1_{2,2} + \Psi_{2,2,2}(3/2,2)) \quad = 3. \\
&\quad \{2,4,8\} \\
\Psi_3(4,3) &= 1 + \Psi(4/2,2) + \Psi(4/3,3) \\
&= 1_3 + \Psi_{3,2}(2,2) + \Psi_{3,3}(1,3) \\
&= 1_3 + (1_{3,2} + \Psi_{3,2,2}(1,2)) + \Psi_{3,3}(1,3) \quad = 4. \\
&\quad \{3,6,12,9\} \\
\Psi(12,3) &= 1 + 3 + 4 = 8. \quad \{1,\ 2,4,8,\ 3,6,12,9\}
\end{aligned}
$$

Lexicographic Order

$\{1, 2, 4, 8, 3, 6, 12, 9\}$:

$$\begin{aligned} 1 &= 1 \\ 2 &= 2 \\ 2 \cdot 2 &= 4 \\ 2 \cdot 2 \cdot 2 &= 8 \\ 3 &= 3 \\ 2 \cdot 3 &= 6 \\ 2 \cdot 2 \cdot 3 &= 12 \\ 3 \cdot 3 &= 9 \end{aligned}$$



Camponotus laevigatus

Enumeration Algorithm (recursive)

Inputs: integers $x \geq y \geq 1$.

1. Find the primes $\leq y$ if you don't have them already.
2. Output 1. $(n = 1)$
3. For each prime $p \leq y$:
   - Recursively construct a list of all p-smooth integers $\leq x/p$,
   - Multiply each number on that list by p,
   - Then output the resulting numbers. $(n = mp)$

Smooth numbers are listed in lexicographic order of their prime divisors.

Lexicographic Enumeration

Example starting at position 100 in the enumeration of 7-smooth integers ≤ 1000:

$$\begin{aligned}
3 \cdot 3 \cdot 7 &= 63 \\
2 \cdot 3 \cdot 3 \cdot 7 &= 126 \\
2 \cdot 2 \cdot 3 \cdot 3 \cdot 7 &= 252 \\
2 \cdot 2 \cdot 2 \cdot 3 \cdot 3 \cdot 7 &= 504 \\
3 \cdot 3 \cdot 3 \cdot 7 &= 189 \\
2 \cdot 3 \cdot 3 \cdot 3 \cdot 7 &= 378 \\
2 \cdot 2 \cdot 3 \cdot 3 \cdot 3 \cdot 7 &= 756 \\
3 \cdot 3 \cdot 3 \cdot 3 \cdot 7 &= 567 \\
5 \cdot 7 &= 35 \\
2 \cdot 5 \cdot 7 &= 70 \\
2 \cdot 2 \cdot 5 \cdot 7 &= 140
\end{aligned}$$

Generation Algorithm - Setup

Inputs:

- Integers x,y, with $x \geq y > 0$, and
- A real number $r \in [0, 1)$ (chosen uniformly at random).

Outputs:

- Integer n, with $0 < n \leq x$,
- A list of primes $p_1, p_2, \ldots$ such that $n = p_1 p_2 \cdots$, with $p_i \leq y$ for all i, and
- n is at position $r\Psi(x, y)$ in the enumeration of y-smooth integers $\leq x$, lexicographically ordered by prime divisors.

  If we estimate Ψ, then n is at position $r\Psi(x, y)(1 + o(1))$.

Generation Algorithm - Steps

Inputs: x, y, r.

1. If $r < 1/\Psi(x, y)$, output 1 and halt.
2. Find real t such that $\Psi(x, t) - r\Psi(x, y)$ is near zero.
3. Find consecutive primes $p_1 < p_2$, near t, such that $\Psi(x, p_1) < r\Psi(x, y) \leq \Psi(x, p_2)$.

$$1 + \sum_{p \leq p_1} \Psi(x/p, p) < r\Psi(x, y) \leq 1 + \sum_{p \leq p_2} \Psi(x/p, p),$$

   so p_2 is our largest prime divisor.
4. Output p_2.
5. Set $r' = \dfrac{r\Psi(x, y) - \Psi(x, p_1)}{\Psi(x/p_2, p_2)}$.
6. Recurse on x/p_2, p_2, and r'.

Algorithm Analysis

The running time is

$$T + D \cdot (S \log\log y + P)$$

where

- T is the cost to build a table of $\rho(u)$ values,
- D is the recursion depth, the number of prime divisors,
- S is the cost of estimating $\Psi(x, y)$,
- $\log\log y$ is the cost of finding t, and
- P is the cost to find p_1, p_2 from t.

Algorithm Details

- T: We use well-known numerical integration methods to precompute the table of ρ-values. We have $u \leq \log_2 x$.

Algorithm Details

- T: We use well-known numerical integration methods to precompute the table of ρ-values. We have $u \leq \log_2 x$.
- D: Alladi 1987, Hensley 1987, and Hildebrand 1987, show that n has on average

$$O\left(\log\log x + \frac{\log x}{\log y}\right)$$

prime divisors.

Algorithm Details

- T: We use well-known numerical integration methods to precompute the table of ρ-values. We have $u \leq \log_2 x$.
- D: Alladi 1987, Hensley 1987, and Hildebrand 1987, show that n has on average

$$O\left(\log\log x + \frac{\log x}{\log y}\right)$$

prime divisors.

- S: We estimate Ψ with either $x\rho(u)$ (if y is not too small), or with a saddle-point based method for smaller y. See [Hildebrand 1986] for the cutoff point.

Algorithm Details

- T: We use well-known numerical integration methods to precompute the table of ρ-values. We have $u \leq \log_2 x$.
- D: Alladi 1987, Hensley 1987, and Hildebrand 1987, show that n has on average

$$O\left(\log\log x + \frac{\log x}{\log y}\right)$$

  prime divisors.
- S: We estimate Ψ with either $x\rho(u)$ (if y is not too small), or with a saddle-point based method for smaller y. See [Hildebrand 1986] for the cutoff point.
- $\log\log y$: For the $x\rho(u)$ method, find t with Newton's method. Otherwise, the Illinois algorithm with bisection or Brent's algorithm works well.

Algorithm Details

- T: We use well-known numerical integration methods to precompute the table of ρ-values. We have $u \leq \log_2 x$.
- D: Alladi 1987, Hensley 1987, and Hildebrand 1987, show that n has on average

$$O\left(\log\log x + \frac{\log x}{\log y}\right)$$

  prime divisors.
- S: We estimate Ψ with either $x\rho(u)$ (if y is not too small), or with a saddle-point based method for smaller y. See [Hildebrand 1986] for the cutoff point.
- $\log\log y$: For the $x\rho(u)$ method, find t with Newton's method. Otherwise, the Illinois algorithm with bisection or Brent's algorithm works well.
- P: We can find p_1, p_2 by sieving a short interval. Average-case or Cramér's is needed here.

Running Time

$$O\left(\frac{(\log x)^3}{\log\log x}\right)$$

arithmetic operations on average.

- We assume the ERH to extend the range of applicability to the estimate $x\rho(u)$ for $\Psi(x, y)$, where ρ is the Dickman-De Bruijn function [Hildebrand 1986], and
- We use the Miller-Rabin probabilistic primality test, so that our list of "primes" all are in fact prime with probability $1 - o(1)$.

Comments:

- We can set $y = x$ to get an alternative to Bach's algorithm, which uses $O(\log x)$ prime tests.
- Conversely, to do what we do here, Bach's algorithm is expected to take

$$O\left(\frac{x}{\Psi(x,y)}(\log x)^2\right)$$

  arithmetic operations. This is interesting if $\rho(u) \sim \log\log x/\log x$, or $u \ll \log\log x$.
- Generate random *semismooth* integers with known prime factorization.

Sample Run

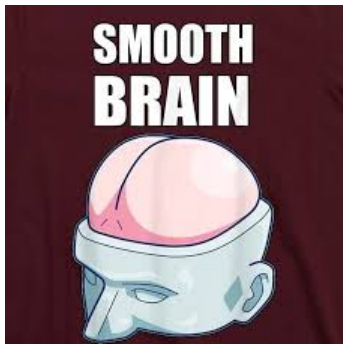
Inputs: $x = 10^{100}$, $y = 10000$, and $r = 0.5$.
Outputs:

2 3 5 7 29 31 97 113 113 113 157 223 241 503 509 569
691 727 1033 1367 1571 2141 2339 2617 2741 3041 3221
3547 3989 4021 4513 4999 5573 6577 7573 9463

This is roughly $4.29 \cdot 10^{97}$, at position near $2.05 \cdot 10^{61}$ in the enumeration, as $\Psi(10^{100}, 10000) \approx 4.10 \cdot 10^{61}$.

This took less than 0.35 seconds, wall time.

Linux desktop, C++.



Smooth

And it's just like the ocean under the moon
Oh, it's the same as the emotion that I get from you
You got the kind of lovin' that can be so smooth, yeah
Give me your heart, make it real, or else forget about it

Questions?
Thank you!