

Efficient Algorithms

Lecture 11: Binary arithmetic and random primes

Lecture overview

An integer can be enormous while its binary representation is short. We measure running time in the number of input bits.

Lecture slides and worked examples.

Primary source: Lecture 11 [Maj26]

Abstract and lecture roadmap

An integer can be enormous while its binary representation is short. We measure running time in the number of input bits.

1. Binary arithmetic: addition, multiplication, division.
2. Euclid's algorithm: why the inputs shrink quickly.
3. Random primes: how density controls waiting time.

The recurring proof plan

Give an algorithm, identify an invariant, and count bit operations.

Based on the supplied Lecture 11 [Maj26]. Background: [Sho09, Chs. 3–5, 9].

Main results

Arithmetic on binary inputs

For $a = \ell(A)$, $b = \ell(B)$, and $L = \max\{a, b\}$:

Addition	$O(a + b)$
Multiplication and division	$O((a + b)^2)$
Greatest common divisor	$O(L^3)$

These elementary bounds suffice to prove efficiency.

Random prime generation

With an exact polynomial-time primality test, rejection sampling returns a uniform n -bit prime in expected polynomial time, for $n \geq 2$.

Value and representation length

Integer X	Binary representation	Bits x
1	1	1
7	111	3
8	1000	4
13	1101	4
1024	10000000000	11

We use capital letters for integer values and corresponding lowercase letters for their bit lengths.

A 1000-bit integer

$2^{999} \leq X < 2^{1000}$, although the input contains only 1000 bits.

The exact bit-length formula

For $X \geq 1$, a binary representation has exactly x bits if and only if

$$2^{x-1} \leq X \leq 2^x - 1.$$

Therefore

Bit length

$$\ell(X) = \lfloor \log_2 X \rfloor + 1 = \lceil \log_2(X + 1) \rceil \quad (X \geq 1).$$

We represent 0 by the single bit 0 and set $\ell(0) = 1$. For $X \geq 1$, the useful approximation is $\ell(X) = \log_2 X + O(1)$.

Binary representation and integer arithmetic: [Sho09, §3.3].

The bit-operation model

Efficiency

An algorithm is efficient if its running time is bounded by a polynomial in the total input length.

- Reading, writing, or combining a bounded number of bits costs $O(1)$.
- An operation on an L -bit integer may require $O(L)$ or more bit operations.
- A loop with B iterations can take exponential time in $b = \ell(B)$, since $B \geq 2^{b-1}$ for $B \geq 1$.

Example. For inputs A, B , the input length is $a + b$. A bound $O((a + b)^2)$ is polynomial.

Computational models: [Sho09, §§3.1–3.3].

Addition by repeated increments

```
SlowAdd( $A, B$ )  
   $S \leftarrow A$   
  repeat  $B$  times:  $S \leftarrow S + 1$   
  return  $S$ 
```

The algorithm is correct, but it uses B iterations.

$$B \geq 2^{b-1} \quad (B \geq 1).$$

Even if each increment cost one step, this would not give a polynomial bound in the input length.

Proof roadmap for efficient addition

Process one binary column at a time. The carry records everything that remains to be handled.

A binary addition example

$$\begin{array}{r} \\ \\ + \\ \hline 1 \end{array} \quad 13 + 7 = 20.$$

Position i	A_i	B_i	Carry in	$(C_i, \text{carry out})$
0	1	1	0	(0, 1)
1	0	1	1	(0, 1)
2	1	1	1	(1, 1)
3	1	0	1	(0, 1)

The final carry becomes the leading 1.

The addition algorithm

Let $m = \max\{a, b\}$. Pad both representations with leading zeros to length m .

```
Add( $A, B$ )  
   $c \leftarrow 0$   
  for  $i = 0, \dots, m - 1$ :  
     $t \leftarrow A_i + B_i + c$   
     $C_i \leftarrow t \bmod 2$   
     $c \leftarrow \lfloor t/2 \rfloor$   
   $C_m \leftarrow c$   
  return  $C_m C_{m-1} \dots C_0$  in binary
```

Remove an unnecessary leading zero. Keep one digit for 0.

Schoolbook addition: [Sho09, §3.3].

Addition: proof and running time

Let c_i be the carry into position i . After i columns,

Loop invariant

$$(A \bmod 2^i) + (B \bmod 2^i) = \sum_{j < i} C_j 2^j + c_i 2^i.$$

Initially both sides are 0. The equality

$$A_i + B_i + c_i = C_i + 2c_{i+1}$$

preserves the invariant. At $i = m$, it gives the full sum. Each column uses $O(1)$ bit operations, since $t \leq 3$.

Running time $O(m) = O(a + b)$.

Multiplication from the binary expansion

$$13 \cdot 11 = 13 \cdot (8 + 2 + 1) = 104 + 26 + 13 = 143.$$

More generally, if $B = \sum_{i=0}^{b-1} B_i 2^i$, then

$$AB = \sum_{i=0}^{b-1} B_i (A 2^i).$$

Proof roadmap

Read one bit of B per iteration. Add the corresponding shifted copy of A when that bit is 1.

There are only b bits to process.

Schoolbook multiplication: [Sho09, §3.3].

Repeated addition versus binary digits

```
SlowMultiply( $A, B$ )
```

```
   $P \leftarrow 0$ 
```

```
  repeat  $B$  times:  $P \leftarrow P + A$ 
```

```
  return  $P$ 
```

After j iterations, $P = jA$, so the answer is correct. The loop still runs B times. This is exponential in b in the worst case.

The useful distinction

The value B may be exponentially larger than the number b of digits used to write it.

Multiplication by shifting and adding

Multiply(A, B)

$P \leftarrow 0, T \leftarrow A, R \leftarrow B$

while $R > 0$:

if $R \bmod 2 = 1$: $P \leftarrow P + T$

$T \leftarrow 2T$

$R \leftarrow \lfloor R/2 \rfloor$

return P

- Multiplying T by 2 shifts its digits left.
- Taking $\lfloor R/2 \rfloor$ removes its last bit.
- $R \bmod 2$ reads its last bit.

Compare the supplied Lecture 11, p. 11 [Maj26].

A multiplication trace

For $A = 13$, $B = 11$, the states at the start of successive iterations are:

P	T	R	$P + TR$
0	13	11	143
13	26	5	143
39	52	2	143
39	104	1	143
143	208	0	143

What stays unchanged?

The quantity $P + TR$ always equals the desired product AB .

Multiplication: proof and running time

Invariant. $P + TR = AB$. Write $R = 2q + \varepsilon$ with $\varepsilon \in \{0, 1\}$. One iteration replaces the state by

$$(P + \varepsilon T, 2T, q).$$

Its invariant is

$$P + \varepsilon T + (2T)q = P + TR = AB.$$

When $R = 0$, the invariant gives $P = AB$.

Bit complexity

There are at most b iterations. All stored integers have at most $a + b + 1$ bits. Even charging for full shifts,

$$O(b(a + b)) \subseteq O((a + b)^2).$$

Division with remainder

The task

Given a divisor $A \geq 1$ and dividend $B \geq 0$, find M, R with

$$B = MA + R, \quad 0 \leq R < A.$$

Example. $29 = 5 \cdot 5 + 4$.

Proof roadmap

Read the dividend from left to right. Maintain a quotient and a remainder for the prefix read so far.

Appending a bit β replaces a prefix V by $2V + \beta$.

Division is an exercise on p. 13 of [Maj26]. See also [Sho09, §3.3].

A binary division example

Divide $B = 29 = (11101)_2$ by $A = 5$.

Next bit	Prefix V	$2R + \beta$	Quotient M	New R
1	1	1	0	1
1	3	3	0	3
1	7	7	1	2
0	14	4	2	4
1	29	9	5	4

At every row, $V = 5M + R$ and $0 \leq R < 5$. The quotient digits are 00101, giving $M = 5$ and $R = 4$.

Binary long division

```
Divide( $A, B$ )  ( $A \geq 1$ )  
   $M \leftarrow 0, R \leftarrow 0$   
  for  $i = b - 1, b - 2, \dots, 0$ :  
     $M \leftarrow 2M$   
     $R \leftarrow 2R + B_i$   
    if  $R \geq A$ :  
       $R \leftarrow R - A$   
       $M \leftarrow M + 1$   
  return ( $M, R$ )
```

Why one subtraction is enough

Before reading a bit, $R < A$. Thus $2R + B_i < 2A$.

Division: proof and running time

Invariant. For the processed prefix V ,

$$V = MA + R, \quad 0 \leq R < A.$$

If the next bit is β , then

$$2V + \beta = (2M)A + (2R + \beta).$$

If $2R + \beta \geq A$, subtract A from the remainder and add 1 to the quotient. The value is unchanged. After all bits have been read, $V = B$.

Bit complexity

There are b iterations. Each uses $O(a + b)$ bit operations for shifts, comparison, and subtraction:

$$O(b(a + b)) \subseteq O((a + b)^2).$$

Euclid's algorithm: an example

$$252 = 2 \cdot 105 + 42,$$

$$105 = 2 \cdot 42 + 21,$$

$$42 = 2 \cdot 21 + 0.$$

The last nonzero remainder is 21.

Proof roadmap

1. Replacing a pair by divisor and remainder preserves its common divisors.
2. Every two divisions reduce the scale by a factor of at least 2.

Euclidean algorithm: [Sho09, §4.1].

The gcd identity

Suppose $B = MA + R$ with $A > 0$ and $0 \leq R < A$.

Euclid's identity

$$\gcd(A, B) = \gcd(R, A).$$

Proof. For every positive integer d :

- If $d \mid A$ and $d \mid B$, then $d \mid B - MA = R$.
- If $d \mid A$ and $d \mid R$, then $d \mid MA + R = B$.

The two pairs have exactly the same common divisors, hence the same greatest common divisor.

[Sho09, §4.1].

The gcd algorithm

For $A, B \geq 0$, not both zero:

```
GCD( $A, B$ )  
  while  $A \neq 0$ :  
     $R \leftarrow B \bmod A$   
     $(B, A) \leftarrow (A, R)$     (simultaneously)  
  return  $B$ 
```

Each iteration preserves the gcd. When the loop ends,

$$\gcd(0, B) = B.$$

The case $A = 0$ returns immediately. Strict decrease of positive remainders proves termination.

Why only logarithmically many divisions?

After an optional initial swap step, consider consecutive positive remainders $x > y$.

- If $y \leq x/2$, the next remainder satisfies

$$x \bmod y < y \leq x/2.$$

- If $y > x/2$, the quotient is 1, so

$$x \bmod y = x - y < x/2.$$

Halving every two steps

For a decreasing remainder sequence, $r_{j+2} < r_j/2$ whenever the terms exist. Starting below 2^L , this permits only $O(L)$ divisions.

A decreasing integer sequence alone would give a much weaker bound.

The bit cost of gcd

Let $L = \max\{a, b\}$. Throughout Euclid's algorithm, every operand has at most L bits.

Number of divisions	$O(L)$
Bit cost of one division	$O(L^2)$
Total bit cost	$O(L^3)$

What this proves

Euclid's algorithm is polynomial-time in the binary input length.

A sharper analysis is possible. The elementary cubic bound already settles the efficiency question.

For a sharper classical analysis, see [Sho09, §4.1].

A uniform random n -bit integer

For $n \geq 1$, fix the leading bit and sample the rest:

```
RandomBits( $n$ )  
   $C_{n-1} \leftarrow 1$   
  for  $i = n - 2, n - 3, \dots, 0$ :  
    sample  $C_i \in \{0, 1\}$  independently and uniformly  
  return  $C_{n-1}C_{n-2} \cdots C_0$ 
```

Exact distribution

Each integer in $I_n = \{2^{n-1}, \dots, 2^n - 1\}$ corresponds to one choice of the $n - 1$ random bits. Its probability is $2^{-(n-1)}$.

Writing the digits directly costs $O(n)$ bit operations. Here n is the requested output length.

An example of prime rejection sampling

For $n = 4$, the candidates are

$$I_4 = \{8, 9, 10, 11, 12, 13, 14, 15\}.$$

Exactly two candidates, 11 and 13, are prime.

Success probability per trial	$q_4 = 2/8 = 1/4$
Expected number of trials	$1/q_4 = 4$
Output probability of each prime	$1/2$

Proof roadmap

Count the acceptable candidates, estimate the waiting time, and then check that rejection does not bias the output.

The random prime algorithm

Assume $n \geq 2$ and an *exact* primality test `IsPrime`.

```
Prime( $n$ )  
  repeat:  
     $P \leftarrow \text{RandomBits}(n)$   (fresh random bits)  
    if IsPrime( $P$ ): return  $P$ 
```

Let $\tau(n)$ bound the bit cost of testing an n -bit candidate. Each trial costs $O(n + \tau(n))$.

The remaining question

What fraction of the interval I_n consists of primes?

Prime generation: [Sho09, §9.4]. Exact polynomial-time testing: [AKS04, AKS19].

The prime number theorem

Define $\pi(X) = \#\{p \leq X : p \text{ is prime}\}$.

Prime number theorem (external input)

$$\pi(X) \sim \frac{X}{\ln X} \quad (X \rightarrow \infty).$$

Here $\ln$ denotes the natural logarithm, and $f \sim g$ means $f/g \rightarrow 1$.

In particular,

$$\pi(2^n) \sim \frac{2^n}{n \ln 2}.$$

The factor $\ln 2$ matters for an asymptotic equivalent. It is absorbed in $\Theta(2^n/n)$ notation.

Precise input and citation appear in the final technical-input section.

Prime density in the n -bit interval

Let K_n count the primes in I_n . For $n \geq 3$, both endpoints $2^{n-1}, 2^n$ are composite, so

$$K_n = \pi(2^n) - \pi(2^{n-1}).$$

Writing each prime-counting asymptotic with its error gives

$$\begin{aligned} K_n &= \frac{2^n}{n \ln 2} - \frac{2^{n-1}}{(n-1) \ln 2} + o\left(\frac{2^n}{n}\right) \\ &\sim \frac{2^{n-1}}{n \ln 2}. \end{aligned}$$

Success probability

$$q_n = \frac{K_n}{2^{n-1}} \sim \frac{1}{n \ln 2} = \Theta(1/n).$$

Expected waiting time

Let T be the number of independent trials up to the first prime. For integers $t \geq 0$,

$$\Pr(T > t) = (1 - q_n)^t \leq e^{-q_n t}.$$

The tail-sum formula and a geometric series give

$$\mathbb{E} T = \sum_{t \geq 0} \Pr(T > t) = \frac{1}{q_n} \sim n \ln 2.$$

Expected bit complexity

$$O(n(\tau(n) + n)).$$

This is polynomial in n when $\tau(n)$ is polynomial.

The tail tends to 0, so termination has probability 1. For $n \geq 3$, no finite worst-case trial bound exists.

Uniformity and a bounded-time version

Let $H = |I_n|$ and $K = K_n$. For any prime $p \in I_n$,

$$\Pr(\text{output } p) = \sum_{j \geq 1} (1 - K/H)^{j-1} \frac{1}{H} = \frac{1}{K}.$$

Thus the output is exactly uniform over the n -bit primes.

Stopping after a fixed budget

The quantitative prime-density input gives $q_n > 1/(3n \ln 2)$. A budget

$$t = \lceil 3n \ln 2 \ln(1/\delta) \rceil$$

gives failure probability at most δ , for $0 < \delta < 1$.

Return a failure symbol if the budget expires. Conditional on success, the output remains uniform.

The algorithms at a glance

Task	Key idea	Bit cost
Addition	Carry per column	$O(a + b)$
Multiplication	One multiplier bit	$O((a + b)^2)$
Division	One dividend bit	$O((a + b)^2)$
Gcd	Remainders shrink	$O(L^3)$
Random integer	Independent digits	$O(n)$
Random prime	Rejection sampling	$O(n(\tau(n) + n))$

The prime bound is an *expected* cost. Here $L = \max(a, b)$ and $\tau(n)$ is the cost of an exact primality test.

The central habit

Count operations on bits, and relate the number of iterations to the input's bit length.

Optional reading: faster multiplication via transforms, [Sho09, §§3.5, 17.6].

Exercises

1. Trace binary addition for $15 + 1$. Where does the extra output bit come from?
2. Trace multiplication for $9 \cdot 6$. Check $P + TR$ after every iteration.
3. Divide 43 by 6 using binary long division.
4. Run Euclid's algorithm on 89 and 55. Why does a long run still have polynomial cost?
5. Suppose a rejection sampler accepts a fraction $q \in (0, 1]$ of its candidates. Derive its expected number of trials and its failure probability after t trials.

Extension. For $n \geq 3$, sample only odd n -bit candidates. What happens to success probability and uniformity?

Technical inputs: the distribution of primes

Prime number theorem

$\pi(X) \sim X/\ln X$ as $X \rightarrow \infty$.

Citation: Shoup, *A Computational Introduction to Number Theory and Algebra*, 2nd ed., Theorem 5.14, p. 116 [Sho09].

Quantitative Bertrand bound

For every integer $m \geq 1$,

$$\pi(2m) - \pi(m) > \frac{m}{3 \ln(2m)}.$$

Taking $m = 2^{n-1}$ gives $q_n > 1/(3n \ln 2)$ for every $n \geq 2$. The lower-endpoint prime 2 only improves the bound when $n = 2$.

Citation: Shoup, Theorem 5.8, p. 109 (proof pp. 109–110) [Sho09].

Technical input: exact primality testing

Polynomial-time primality testing

There is a deterministic algorithm that decides whether an integer $N \geq 2$ is prime using a number of bit operations polynomial in $\ell(N)$.

Citation: Agrawal, Kayal, and Saxena, *PRIMES is in P*, Annals of Mathematics **160** (2004), 781–793, Theorem 4.1, p. 785, and Theorem 5.1, pp. 789–790 [AKS04]. See also their 2019 correction to Lemma 4.3, pp. 317–318 [AKS19].

The lecture uses this result through the exact test `IsPrime` with cost $\tau(n)$. The arithmetic algorithms, invariants, and rejection-sampling calculations have been proved in the lecture.

References

- [AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena.
PRIMES is in P.
Annals of Mathematics, 160(2):781–793, September 2004.
- [AKS19] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena.
Errata: PRIMES is in P.
Annals of Mathematics, 189(1):317–318, 2019.
URL: <https://annals.math.princeton.edu/2019/189-1/p06>, doi:10.4007/annals.2019.189.1.6.
- [Maj26] Hemanta K. Maji.
Lecture 11: Efficient algorithms.
CS355 course slides, 2026.
Supplied lecture slides, 19 pages.
- [Sho09] Victor Shoup.
A Computational Introduction to Number Theory and Algebra.
Cambridge University Press, Cambridge, second edition, 2009.
Page and theorem references follow the author's Version 2 PDF (June 2008), corresponding to the second print edition.
URL: <https://shoup.net/ntb/ntb-v2.pdf>.