Algorithms for arithmetic

Two reasons for studying this

despite computers having built-in arithmetic operations

- How to design circuits for arithmetic (to use in computers) that are as efficient as possible

- Big-number arithmetic with numbers much bigger than a machine word

  

  mathematics

  modern cryptography
"Polynomial time" means that running time is $O(n^2)$ where $n$ is input length (#digits, or #bits in binary), i.e. $\text{length}(x) = \log_2(x) = n$.

Example problem: Testing whether an input number is a prime number (only divisors are $1$ and $p$).

Algorithm (trial division):
- for $x = 2, 3, \ldots, p-1$:
  - if $x$ evenly divides $p$:
    - return False
- return True

Time (#of arithmetic operations) $= O(p) = O(2^n)$ not polynomial.
Faster trial division:
\[
\text{for } k = 2, 3, \ldots, \lfloor \sqrt{p} \rfloor:
\begin{align*}
\text{if } k & \text{ evenly divides } p: \\
\text{return False}
\end{align*}
\]
\[
\text{return True}
\]

\[
O(\sqrt{p}) = O(2^{\sqrt{2}})
\]

Still not polynomial

AKS primality test (2002):
\[
O((\log p)^{12}) = O(n^{12})
\]

Polynomial but not practical

Improved to:
\[
O((\log p)^6) = O(n^6)
\]

Fast practical randomized algorithms are known with smaller polynomial time bounds

Still unknown: can we factor composite numbers in polynomial time?
Binary addition

Algorithm:
- Work right to left at each step
- 3 bits input: digits of 2 input numbers + carry from previous digit
- 2 bits output: output digit + new carry
inputs:  1 0 1 1 1 0 1
        y  0 1 0 1 1 0 0
        z  1 1 1 0 1 0 1

1's bits of output: 0 0 0 0 1 0 0
2's bits of output: 1 1 1 1 1 0 1

\[ x + y + z = \]
\[ \begin{array}{cccccccc}
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
+ & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
\end{array} \]
long multiplication

\[
\begin{array}{c}
1101 \\
\times 1011 \\
\hline
\end{array}
\]

for each bit position in 2nd number: write either 0 or (first num) \( \ll \) pos.

\( O(n^2) \)

\( O(n) \) bit numbers

\[
\begin{array}{c}
1101 \\
+ \quad 1101000 \\
\hline
10001111 \\
\end{array}
\]

because it reduces to \( O(n) \) additions
Karatsuba multiplication

product \((x, y)\) of two \(n\)-bit numbers:

- **Base case (optimized)**: if \(n \leq 32\), use built-in machine multiplication instructions.
- **Represent**
  \(x = x_h \times b + x_e\)
  \(y = y_h \times b + y_e\)
- **2-digit numbers in base 2** \(x_h y_h = b\)
- **Recursively compute**
  \(p = x_h y_h\)
  \(q = x_e y_e\)
  \(r = (x_h y_e + x_e y_h)\)
- **Return**
  \(p = 2^{\lceil n/2 \rceil}\)
  \((r - p - q) = 2^{\lfloor n/2 \rfloor}\)
  \(+ q\)

\[ T(n) = 3T(\frac{n}{2}) + O(n \log n) = O(n^{\log_2 3}) \]
assoc. law \[(a + b)(c + d) = ac + bc + ad + bd\]

so \[xy =\]
\[x_h y_h b^2 + (x_h y_e + x_e y_h) b + x_e y_e\]

trick:
let \[p = x_h y_h\]
\[q = x_e y_e\]
\[r = (x_h + y_h)(x_e + y_e)\]
\[x_h y_e + x_e y_h = r - p - q\]