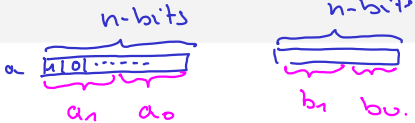
Plan for today

- Review of Karatsuba algorithm and its analysis ✓ / School method for division.
- Division with remainder
- The Euclidean algorithm
- Modular arithmetic
- Fast exponentiation.

Last Exercise : Subtraction.

Karatsuba: Main idea



- a and b two n-bit natural numbers, $n = 2^l$ for some $l \in \mathbb{N}_0$.

- Divide: $a = a_1 \cdot 2^{n/2} + a_0$, $b = b_1 \cdot 2^{n/2} + b_0$

$a \cdot b = (a_1 \cdot b_1) \cdot 2^n + (a_1 \cdot b_0 + a_0 \cdot b_1) \cdot 2^{n/2} + a_0 \cdot b_0$

$S_3 - S_1 - S_2$

- Compute recursively: $s_1 = a_1 \cdot b_1$, $s_2 = a_0 \cdot b_0$, $s_3 = (a_1 + a_0) \cdot (b_1 + b_0)$

$S_3 = a_1 \cdot b_1 + a_1 \cdot b_0 + a_0 \cdot b_1 + a_0 \cdot b_0$

- Return $s_1 \cdot 2^n + (s_3 - s_1 - s_2) \cdot 2^{n/2} + s_2$

3. Multiplications of $\frac{n}{2}$-bit numbers + $O(n)$ time for re-arrangement.

Karatsuba: The algorithm

$T(n)$ is time required by Multiply on two n-bit numbers.

```
function Multiply (a, b)
Input: Two n-bit integers a, b ∈ ℕ0
Output: Their product a · b
if n = 1 return a · b                    O(1)
else
```

$T(n)$ satisfies the recursion. (c some constant.)

$n \geq 1$: $T(n) \leq 3 \cdot T(n/2) + c \cdot n$

a_1, a_0 leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of a
b_1, b_0 leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of b

} $O(n)$ time.

$s_1 = \text{Multiply}(a_1, b_1)$ $T(n/2)$
$s_2 = \text{Multiply}(a_0, b_0)$ $T(n/2)$
$s_3 = \text{Multiply}(a_1 + a_0, b_1 + b_0)$ $T(n/2) + O(n)$

$(a_1+a_0)(b_1+b_0)$
$= a_1 \cdot b_1 \cdot 2^2$
$+ a_1 \cdot 2 \cdot b_0 +$
$b_1 \cdot 2 \cdot a_0 + a_0 \cdot b_0$

return $s_1 \cdot 2^n + (s_3 - s_1 - s_2) \cdot 2^{n/2} + s_2$ $O(n)$

Analysis

Theorem

The Karatsuba algorithm runs in time $O(n^{\log_2 3})$.

Proof: We want to show that $T(n) \le d \cdot n^{\log_2 3}$.

$$T(n) \le 3 \cdot T(n/2) + c \cdot n$$

$$\le 3\left[3T\left(\frac{n}{2^2}\right) + c \cdot \frac{n}{2}\right] + c \cdot n$$

$$= 3^2 \cdot T\left(\frac{n}{2^2}\right) + c \cdot n \cdot \frac{3}{2} + c \cdot n \cdot \left(\frac{3}{2}\right)^0$$

$$\dots \le \underbrace{3^{\log_2 n}}_{n^{\log_2 3}} + c \cdot n \sum_{i=0}^{\log_2 n - 1} \left(\frac{3}{2}\right)^i \le n^{\log_2 3} + c \cdot 2 \cdot n^{\log_2 (3)} = O(n^{\log_2 3})$$

$$= \frac{\left(\frac{3}{2}\right)^{\log_2 n} - 1}{3/2 - 1} \le 2 \cdot \left(\frac{3}{2}\right)^{\log_2 n} = 2 \cdot n^{\log_2 (3/2)} = 2 \cdot n^{\log_2 3 - 1}$$

Division with remainder

Theorem

For $a, b \in \mathbb{N}$, $b \neq 0$, there exists $q, r \in \mathbb{N}$ with

i) $a = q \cdot b + r$

ii) $0 \leq r < b$.

$q = \left\lfloor \frac{a}{b} \right\rfloor \qquad r = a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b$

e.g. $10 = 3 \cdot 3 + 1$

$a \quad q \quad b \quad r$

Algorithm

q r

$a = q \cdot b + r \qquad 0 \le r < b$

```
function divide(a,b)   # b>= 1

if a = 0: return (q,r)=(0,0)
                    O(1)-time
(q,r) = divide(⌊a/2⌋,b)
q = 2q,  r = 2r
if a is odd:
   r = r+1
if r>=b:
   r = r-b
   q = q+1
return (q,r)
```

$$\left\lfloor \frac{a}{2} \right\rfloor = \begin{cases} \frac{a}{2} - \frac{1}{2} & \text{if } a \text{ odd} \\ \frac{a}{2} & \text{if } a \text{ is even.} \end{cases}$$

$\left\lfloor \frac{a}{2} \right\rfloor = q \cdot b + r$

1. Case: a even.

$\frac{a}{2} = q \cdot b + r$

$\Leftrightarrow \quad a = 2q \cdot b + 2r$ if $r \ge b$, $a = (2q+1) \cdot b + (2r - b)$

$r < b$

$\Rightarrow 2r - b < b$

Algorithm

```
function divide(a,b)  # b>= 1

if a = 0: return (q,r)=(0,0)

(q,r) = divide([a/2],b)
q = 2q,  r = 2r
if a is odd:
   r = r+1
if r>=b:
   r = r-b
   q = q+1
return (q,r)
```

$$\left\lfloor \frac{a}{2} \right\rfloor = \begin{cases} \frac{a}{2} - \frac{1}{2} & \text{if } a \text{ odd} \\ \frac{a}{2} & \text{if } a \text{ is even.} \end{cases}$$

$$\left\lfloor \frac{a}{2} \right\rfloor = q \cdot b + r$$

a is odd. $\left\lfloor \frac{a}{2} \right\rfloor = \frac{a}{2} - \frac{1}{2}$.

$$\frac{a}{2} - \frac{1}{2} = q \cdot b + r$$

$$a = 2q \cdot b + 2r + 1$$

Algorithm

```
function divide(a,b)  # b>= 1

if a = 0: return (q,r)=(0,0)

(q,r) = divide(⌊a/2⌋,b)
q = 2q,  r = 2r
if a is odd:
   r = r+1
if r>=b:
   r = r-b
   q = q+1
return (q,r)
```

O(1)

O(1)

O(1)

O(e)

O(1) [since last bit of q = 0]

number of bits of q increases by exactly one!

Analysis

Theorem

Let $a, b \in \mathbb{N}$, $b \geq 1$. The algorithm `divide` *runs in time $O(\text{size}(q) \cdot \text{size}(b))$ on Input a, b.*

Proof: Of k each recursive call:

- $O(1)$ operations.
- # bits of q increases by one.

$\Rightarrow O(\#\text{ bits of } q \cdot \#\text{ bits of } b)$ operations.

∎

The greatest common divisor

- $a, b \in \mathbb{N}_0$ not both equal to zero.
- $\gcd(a, b) = \max\{d \in \mathbb{N} : d \mid a,\ d \mid b\}$.

$\gcd(16, 12) = 4$

Suppose: $a \geq b \geq 1$

Lemma

Let $a, b \in \mathbb{N}$ $b \neq 0$ and let $q, r \in \mathbb{N}$ with

$$a = q \cdot b + r, \quad 0 \leq r < b,$$

then $\gcd(a, b) = \gcd(b, r)$.

One has: $q \geq 1$

$q \cdot b > r \Rightarrow a > 2 \cdot r$

$\Rightarrow$ # bits of r < # bits of a.

proof: we show: $d \in \mathbb{N}$ divides a and b $\Leftrightarrow$ d divides b and r.

$\Rightarrow$: $d \mid a$ and $d \mid b$ then there exist $a', b' \in \mathbb{N}$ with $a = d \cdot a'$, $b = d \cdot b'$

$r = a - q \cdot b = d(\underbrace{a' - q \cdot b'}_{\in \mathbb{N}}) \Rightarrow d \mid r$

$\Leftarrow$: Similar.

The Euclidean algorithm

```
function gcd(a, b)

Input: a, b ∈ ℕ₀ with a ≥ b and a > 0
Output: gcd(a, b)

if b = 0 return a
else
    Compute q, r ∈ ℕ with a = q · b + r, 0 ≤ r < b.
    return gcd(b, r)
```

$O(\text{size}(q) \cdot \text{size}(b)) = O(\text{size}(a)^2)$

↑ recursive call.

We have seen that every second recursive call, the first parameter looses at least one bit.

$\Rightarrow O(\text{size}(a))$ recursive calls. $\Rightarrow O(\text{size}(a)^3)$ alg.

Analysis: Lower bound

Fibonacci numbers: $F_0 = 0, F_1 = 1, \quad F_N = F_{N-1} + F_{N-2}, \quad N \geq 2.$

$$F_N = q \cdot F_{N-1} + r \quad , \quad q = 1$$

$r = F_{N-2}$.

On input F_N and F_{N-1}

Euclid generates all Fibonacci numbers up to F_2.

if $F_{N-1} > F_{N-2}$

holds for $N \geq 4$.

$\text{Size}(F_N) = \Theta(N)$

$F_N \leq 2 \cdot F_{N-1} \leq 2^2 F_{N-2} \leq \cdots 2^N \cdot F_1$

has at most N - Bits.

lower bound: # of bits to write down. $F_N, F_{N-1}, \ldots, F_1$

$= \Theta(n^2)$

$F_N \geq 2 \cdot F_{N-2} \geq 2^2 \cdot F_{N-4} \geq 2^3 F_{N-3\cdot 2}$

$\geq 2^{\left\lfloor \frac{N-1}{2} \right\rfloor} F_1$

Analysis: Lower bound

Observation: There are N-bit numbers a and b (with $a = F_N$, $b = F_{N-1}$) such that

Euclid(a, b) needs space $\Theta(n^2)$

Alg. writes down: $F_N, F_{N-1}, \ldots F_1$

$\geq \frac{N}{2} + \frac{N-1}{2} + \ldots + \frac{1}{2}$ bits.

$= \frac{1}{2} \frac{N(N+1)}{2} = \Theta(N^2)$.

Analysis: Upper bound

Thm: Euclid(a,b) requires time $O(\text{size}(a) \cdot \text{size}(b))$

proof: Suppose $a \geq b \geq 1$. We show: There exists a constant d such that Euclid runs in time $d \cdot \log_2(a+1) \cdot \log_2(b+1)$.

Proof: Induction on number of recursive calls.

$n=1$ Choose d large enough such that runtime $\leq d \cdot \log_2(a+1) \cdot \log_2(b+1)$

$n>1$ Alg computes $a = q \cdot b + r$ in time $d \cdot \log_2(q+1) \cdot \log_2(b+1)$

recursive call on (b,r): $d \cdot \log_2(b+1) \cdot \log_2(r+1)$ $(r \geq 1)$

All together: running time $\leq d \left[\log_2(b+1) \cdot \log_2(r+1) + \log_2(q+1) \cdot \log_2(b+1) \right]$

$= d \cdot \log_2(b+1) \left[\log_2(q+1) + \log_2(r+1) \right]$

$\leq \log_2(a+1)$

The extended Euclidean Algorithm

function exgcd(a, b)

Input: $a, b \in \mathbb{N}_0$ with $a \geq b$ and $a > 0$
Output: $(\gcd(a, b), x, y)$ with

if $b = 0$ return $(a, 1, 0)$
else
Compute $q, r \in \mathbb{N}$ with $a = q \cdot b + r$, $0 \leq r < b$.
$(d, x', y') = \text{exgcd}(b, r)$

return

Analysis

Theorem

The extended Euclidean algorithm runs in time $O(size(a) \cdot \text{size}(b))$

Computing in $\mathbb{Z}_N$

- $N \in \mathbb{N}$, $a \in \mathbb{Z}$: $[a] = \{x \in \mathbb{Z} \colon N \mid (a - x)\}$
- $\mathbb{Z}_N = (\{[a] \colon a \in \mathbb{Z}\}, \oplus, \odot)$