

Integer Algorithms: Prime Numbers

p is *prime* \equiv

$p > 1$ and $\forall d ((d > 0 \wedge d \text{ divides } p) \rightarrow (d = 1 \vee d = p))$

p is *composite* $\equiv p > 1$ and p is not prime.

If n is composite, then $\exists d (d \text{ divides } n \text{ and } 1 < d \leq \sqrt{n})$.

procedure $prime(n: \mathbf{Z}$ with $n > 1)$

for $d := 2$ **to** $\lfloor \sqrt{n} \rfloor$

if n is divisible by d

then return false

return true

$prime(n)$ performs $O(\sqrt{n})$ divisions.

Greatest Common Divisor

d is the *greatest common divisor* of a and $b \equiv$

d is the largest integer that divides a and b .

$a \bmod m = r \equiv$

$m > 0$, $0 \leq r < m$, and $\exists q (a = mq + r)$.

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

procedure $gcd(a, b: \mathbf{Z}^+)$

while $b \neq 0$

$r := a \bmod b$

$a := b$

$b := r$

return a

$gcd(a, b)$ performs $O(\log(a + b))$ mod operations.

Representation of Integers

If $b > 1$ and $n \geq 1$, n can be uniquely represented by:

$$n = \sum_{i=0}^k a_i b^i \quad \text{where } k \geq 0, 0 \leq a_i < b, \text{ and } a_k > 0$$

procedure *base_expansion* (n, b : \mathbf{Z}^+ with $b > 1$)

$k := 0$

while $n \neq 0$

$a_k := n \bmod b$

$n := \lfloor n/b \rfloor$

$k := k + 1$

return (a_{k-1}, \dots, a_0)

base_expansion(n, b) performs $O(\log_b n)$ mods and divisions, e.g., n can be represented with $O(\log n)$ bits.

Binary Addition

Binary addition adds one column at a time.

procedure *add*(a, b : \mathbf{Z}^+)

$(a_{n-1}, \dots, a_0) := \text{base_expansion}(a, 2)$

$(b_{n-1}, \dots, b_0) := \text{base_expansion}(b, 2)$

carry := 0

for $j := 0$ **to** $n - 1$

$s_j := (a_j + b_j + \text{carry}) \bmod 2$

$\text{carry} := \lfloor (a_j + b_j + \text{carry})/2 \rfloor$

$s_n := \text{carry}$

return (s_n, \dots, s_0)

add(a, b) performs $O(n)$ bit operations, where a and b are represented with n bits.

Binary Multiplication

Binary multiplication performs multiple additions.

```
procedure multiply(a, b:  $\mathbf{Z}^+$ )  
    ( $a_{n-1}, \dots, a_0$ ) := base_expansion(a, 2)  
    ( $b_{n-1}, \dots, b_0$ ) := base_expansion(b, 2)  
    p := 0  
    for j := 0 to n - 1  
        if  $b_j = 1$  then p := add(p, a)  
        shift a one place adding one 0  
    return p
```

multiply(*a*, *b*) performs $O(n^2)$ bit operations, where *a* and *b* are represented with *n* bits.