Logarithms

- $\log_a(a^x) = x$
- $a^{\log_a x} = x$

$$x = a^y \Rightarrow a^{\log_a x} = a^{\log_a a^y} = a^y = x$$

- $a^{x+y} = a^x \cdot a^y$ $\qquad \log_a(x \cdot y) = \log_a x + \log_a y$

  $a^{x \cdot y} = (a^x)^y$ $\qquad \log_a(x^y) = y \cdot \log_a x$

- $a^{\log_b x} = x^{\log_b a}$ (Swap Rule)

$$b^{\log_b a^{\log_b x}} = b^{\log_b x \cdot \log_b a}$$

$$= \left(b^{\log_b x}\right)^{\log_b a} = x^{\log_b a}$$

- $\log_a x = \log_b\left(b^{\log_a x}\right) = \log_b(x^{\log_a b})$

$$= \log_a b \cdot \log_b x \quad \left(\text{Transformation Rule}\right)$$

Example

$$\log_2 x = \log_2 10 \cdot \log_{10} x$$

$$\approx 10/3 \cdot \log_{10} x$$

$$\Rightarrow \log_a x = \Theta(\log_b x) = \Theta(\lg x)$$

```
MergeSort(l, r)                    % data in array A
    if l < r then
        m := (l+r)/2               % integer division
        MergeSort(l, m)
        MergeSort(m+1, r)
        Merge(l, m, r)
```

```
Merge(l, m, r)
    Copy A[l, m] into new array B
    Copy A[m+1, r] into new array C
    while B and C not empty
        find smallest among first elements of B and C
        put it into A[l, r]
```

Running Time of MergeSort

$T(n)$ worst case running time for array of length n

recurrence relation

$$T(n) = \underbrace{2\,T(n/2)}_{\text{running the alg on 2 halfs}} + \Theta(n), \quad n > 1$$

$$T(1) = D = \Theta(1)$$

What is $T(n)$? In Θ-notation?

Suppose $n = 2^k \Rightarrow k = \lg n$

Substitution Method

$$T(n) = 2\,T(n/2) + \Theta(n)$$
$$= 2\,(2\,T(n/2\cdot 2) + \Theta(n/2)) + \Theta(n)$$
$$= 2\cdot 2\,T(n/2\cdot 2) + 2\,\Theta(n/2) + \Theta(n)$$
$$= 2^2\,T(n/2^2) + 2\,\Theta(n)$$
$$= 2^2\,(2\,T(n/2^3) + \Theta(n/2^2)) + 2\,\Theta(n)$$
$$= 2^3\,T(n/2^3) + \Theta(n) + 2\,\Theta(n)$$
$$\cdots$$
$$= 2^k\,T(n/2^k) + k\,\Theta(n)$$
$$= n\,\Theta(1) + \lg n \cdot \Theta(n) = \Theta(n \cdot \lg n)$$

Binary Search

Input: Sorted array A, element q to be found in A

1. Divide: compare q with middle element in A
2. Conquer: recurse in one subarray
3. Combine: return from call (no work)

Recurrence for Running Time

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 1 \cdot T(n/2) + \Theta(1) & \text{if } n > 1 \end{cases}$$

Substitution Method

$n = 2^k$

$$\begin{aligned} T(n) &= T(n/2) + \Theta(1) \\ &= T(n/2 \cdot 2) + \Theta(1) + \Theta(1) \\ &= T(n/2^k) + k\,\Theta(1) \\ &= \Theta(1) + k \cdot \Theta(1) \\ &= \Theta(1) + \lg n \cdot \Theta(1) = \Theta(\lg n) \end{aligned}$$

Powering a number

Input: number x, integer $n \geq 0$

Output: x^n

In cryptography (RSA), powering modulo some k is basic operation,

i.e., $x^n \bmod k$ $\quad$ ($x \cdot y \bmod k$ $= (x \bmod k) \cdot (y \bmod k) \bmod k$)

$\Rightarrow$ multiplication mod k is a special operation

$$5 \cdot 8 \bmod 3 = 2 \cdot 2 \bmod 3$$
$$40 \bmod 3 = 1, \quad 4 \bmod 3 = 1$$

```
pow(x, n)
    p := 1
    for i := 1 to n do
        p := p * x
    return p
```

Running time $\Theta(n)$

Ideas: $n = 2m \Rightarrow x^n = x^m \cdot x^m$

$n = 2m+1 \Rightarrow x^n = x^m \cdot x^m \cdot x$

```
powrec(x, n)
    if n = 1 then return x
    else
        p := powrec(x, n div 2)        (integer division)
        if n mod 2 = 0                 (divide)
        then return p * p              (combine)
        else return p * p * x          (combine)
```

Running Time

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(n/2) + \Theta(1) & \end{cases}$$

$$\Rightarrow T(n) = \Theta(\lg n)$$

Multiplication of binary numbers

$a = \underbrace{1\,1\,0\,\ldots\,\ldots\,\ldots}_{n \text{ bit}}$

$b = \underbrace{0\,1\,\ldots\,\ldots\,\ldots\,1}_{n \text{ bit}}$

Cost of $a * b = n^2$ digit multiplications

Example

$23 * 14 = (2 * 10 + 3) * (1 * 10 + 4)$

$= (2 * 1) * 10^2 + (3 * 1 + 2 * 4) \cdot 10^1 + (3 \cdot 4)$

Save one multiplication

$(3 \cdot 1 + 2 * 4) = (2 + 3) * (1 + 4) - 2 * 3 - 3 * 4$

In general

$$a = a_1 10^{n/2} + a_0$$

$$b = b_1 10^{n/2} + b_0$$

$$\Rightarrow \quad a \cdot b = (a_1 * b_1) 10^n + (a_1 * b_0 + a_0 * b_1) 10^{n/2} + a_0 b_0$$

Trick: $a_1 * b_0 + a_0 * b_1$

$$= (a_1 + a_0) * (b_1 + b_0) - (a_1 * b_1) - (a_0 * b_0)$$

1 multiplication instead of 2

$T(n) :=$ # digit multiplications for input n digits

$$T_1(n) = \begin{cases} 1 & n = 1 \\ 3\,T(n/2) + \Theta(n) & , \; n > 1 \end{cases}$$