

## LECTURE 2 • USEFUL FACTS

8/28/18

$$(a^b)^c = (a^c)^b = a^{bc}$$

$$a^{\log_b c} = c^{\log_b a}$$

→ proof

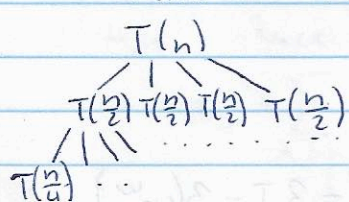
- note  $a = b^{\log_b a}$

$$(b^{\log_b a})^{\log_b c} = (b^{\log_b c})^{\log_b a}$$

## • RECURRENCE

- think of tree  $c \rightarrow$  helpful to introduce constant

$$T(n) = 4 \cdot T\left(\frac{n}{2}\right) + O(n)$$



→ problem of size  $n$  at cost  $c \cdot n$  |  $1 \cdot cn$

→ 4 problems of size  $\frac{n}{2}$ : each  $c \cdot \frac{n}{2}$  |  $4 \cdot c \cdot \frac{n}{2}$

→ 16 problems of size  $\frac{n}{4}$ : each  $c \cdot \frac{n}{4}$  |  $16 \cdot c \cdot \frac{n}{4}$

⇒ generalize

→  $4^i$  problems of size  $\frac{n}{2^i}$  each at cost  $c \cdot \frac{n}{2^i}$  |  $4^i \cdot c \cdot \frac{n}{2^i}$

- runtime

$$T(n) = 1cn + 4c\frac{n}{2} + 16c\frac{n}{4} + \dots + 4^i c \frac{n}{2^i}$$

$$= cn \left( 1 + \left(\frac{4}{2}\right)^1 + \left(\frac{4}{2}\right)^2 + \dots + \left(\frac{4}{2}\right)^i + \dots + \left(\frac{4}{2}\right)^{\log_2 n} \right)$$

$$= cn (1 + 2 + 4 + \dots + n) \leq cn (2n) = O(n^2)$$

$\frac{1}{2} \log_2 n$  bound because  $1 + \frac{1}{2} + \frac{1}{4} + \dots \leq 2$

→ last term is  $\log_2 n$

- every time we go down the layer, problem size halves

→ base case can be solved linearly

- when  $T(n) = 3T\left(\frac{n}{2}\right) + cn$

$$= cn \left( 1 + \left(\frac{3}{2}\right)^1 + \left(\frac{3}{2}\right)^2 + \dots + \left(\frac{3}{2}\right)^i + \dots + \left(\frac{3}{2}\right)^{\log_2 n} \right)$$

$$= cn \left( 1 + \left(\frac{3}{2}\right)^1 + \left(\frac{3}{2}\right)^2 + \dots + \frac{n^{\log_2 3}}{n} \right)$$

$$\leq cn \left( \frac{n^{\log_2 3}}{n} \right) = cn^{\log_2 3}$$

## BINARY SEARCH

- divide and conquer

- find key  $k$  in  $a[0, 1, \dots, n-1]$

in sorted order

→ compare  $k$  with  $a\left[\frac{n}{2}\right]$  and depending on result repeat it on lower or upper half

$$T(n) = T\left(\frac{n}{2}\right) + O(1) = O(\log n)$$

## • NOTE

$$1 + v + v^2 + \dots + v^m$$

$$v > 1 \quad O(v^m)$$

$$v = 1 \quad O(m)$$

$$v < 1 \quad O(1)$$



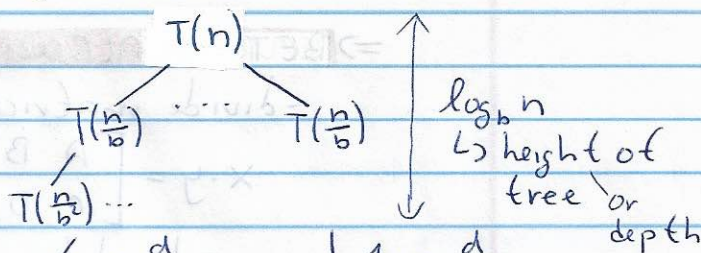
$$a^{\log_b n} = n^{\log_b a}$$

↪ width

## • GENERAL RECURSION

$$T(n) = a T\left(\frac{n}{b}\right) + O(n^d)$$

||  
cnd



- 1 problem of size  $n$  at cost  $cn^d$
- $a$  problems of size  $\frac{n}{b}$  at cost  $c\left(\frac{n}{b}\right)^d$
- $a^2$  problems of size  $\frac{n}{b^2}$  at cost  $c\left(\frac{n}{b^2}\right)^d$
- $a^i$  problems of size  $\frac{n}{b^i}$  at cost  $c\left(\frac{n}{b^i}\right)^d$

$$T(n) = cn^d \left( 1 + \frac{a}{b^d} + \left(\frac{a}{b^d}\right)^2 + \dots + \left(\frac{a}{b^d}\right)^i + \left(\frac{a}{b^d}\right)^{\log_b n} \right)$$

## ⇒ DISCUSS CASES - MASTER THEOREM

- CASE 1 - bottom heavy tree

$$\frac{a}{b^d} > 1 \quad cn^d \left(\frac{a}{b^d}\right)^{\log_b n} = O\left(\frac{n^d n^{\log_b a}}{b^{\log_b n d}}\right)$$

→ most of the work is done on leaves =  $O\left(\frac{n^d n^{\log_b a}}{n^d}\right) = O(n^{\log_b a})$

- CASE 2 - balanced tree → all operations cost the same

$$\frac{a}{b^d} = 1 \quad cn^d \log_b n = O(n^d \log n)$$

- CASE 3 - top heavy tree → most of work is done on top

$$\frac{a}{b^d} < 1 \quad cn^d \cdot O(1) = O(n^d)$$

because we divide  $n$  by  $b$  in every step

- trees have different asymptotic behavior

## • MATRIX MULTIPLICATION

- input: 2  $n \times n$  matrices

- output:  $n \times n$  matrix

$$i \begin{bmatrix} x \end{bmatrix} \begin{bmatrix} y \end{bmatrix}^j = \begin{bmatrix} z \end{bmatrix}_{ij}$$

$$z_{ij} = \langle x[i,*], y[* ,j] \rangle = \sum_{k=1}^n x_{ik} y_{kj}$$

→ algorithm

$$\forall i \in [n]$$

$$\forall j \in [n]$$

$$z = \sum x_{ik} y_{kj}$$

$$\left. \begin{array}{l} \begin{matrix} \times n \\ \times n \end{matrix} \\ \left. \begin{matrix} \times n \\ \times n \end{matrix} \right\} O(n^3) \\ \left. \begin{matrix} \times n \\ \times n \end{matrix} \right\} O(n) \end{array} \right\}$$



## ⇒ RECURSIVE ALGORITHM

- divide matrices to blocks

$$X \cdot Y = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

→ divided problem into 8 subproblems

$$T(n) = 8 T\left(\frac{n}{2}\right) + O(n^2)$$

↳ get  $\frac{n}{2} \times \frac{n}{2}$  matrix      ↳ adding matrices

→ use theorem

$$a=8, b=2, d=2$$

$$\text{ratio } \frac{a}{b^d} = \frac{8}{2^2} = 2 > 1$$

$$\text{runtime } O(n^{\log_2 8}) = O(n^3)$$

→ to get better runtime than  $O(n^3)$

$$\text{we have to reduce } T(n) = 8 T\left(\frac{n}{2}\right) + O(n^2)$$

## ⇒ BETTER ALGORITHM - STRASSEN

$$P_1 = A(F-H)$$

$$P_2 = (A+G)H$$

⋮

$$\begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_1 - P_7 \end{bmatrix}$$

$$T(n) = 7 T\left(\frac{n}{2}\right) + O(n^2)$$

$$= O(n^{\log_2 7}) = O(n^{2.81})$$

$$\frac{a}{b^d} = \frac{7}{2^2} > 1 \quad O(n^{\log_2 7})$$

- If you were to find a way to multiply  $k \times k$  matrices in  $k^w$  multiplications, then you can obtain

$$T(n) = k^w T\left(\frac{n}{k}\right) + O(n^2)$$

$$= O(n^w)$$