MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 6

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Manal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55
Class Room : 2101

Binary search

Find an element in a sorted array:

Divide: Check middle element.

Conquer: Recursively search 1 subarray.

Combine: Trivial.

Example: Find 9

Binary search

Find an element in a sorted array:

Divide: Check middle element.

Conquer: Recursively search 1 subarray.

Combine: Trivial.

Example: Find 9

Binary search

Find an element in a sorted array:

Divide: Check middle element.

Conquer: Recursively search 1 subarray.

Combine: Trivial.

Example: Find 9

3 5 7 8 9 12 15

Binary search

Find an element in a sorted array:

Divide: Check middle element.

Conquer: Recursively search 1 subarray.

Combine: Trivial.

Example: Find 9

Binary search

Find an element in a sorted array:

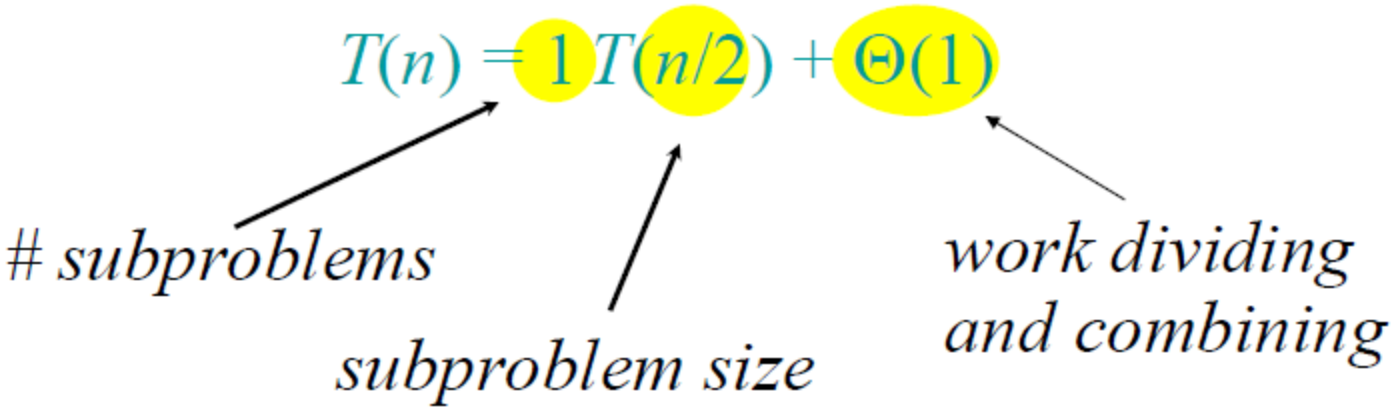
Divide: Check middle element.

Conquer: Recursively search 1 subarray.

Combine: Trivial.

Example: Find 9

Recurrence for binary search



$$n^{\log_b a} = n^{\log_2 1} = n^0 = 1 \Rightarrow \text{CASE 2 } (k = 0)$$
$$\Rightarrow T(n) = \Theta(\lg n) .$$

Powering a number

Problem: Compute a^n, where $n \in \mathbb{N}$.

Naive algorithm: $\Theta(n)$.

Divide-and-conquer algorithm:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & \text{if } n \text{ is even;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & \text{if } n \text{ is odd.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n) .$$

Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

0 1 1 2 3 5 8 13 21 34 ⋯

Naive recursive algorithm: $\Omega(\phi^n)$ (exponential time), where $\phi = (1+\sqrt{5})/2$ is the ***golden ratio***.

Fibonacci numbers

Recursive definition:

$$F_n = \begin{cases} 0 & \text{if } n = 0; \\ 1 & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n \geq 2. \end{cases}$$

0 1 1 2 3 5 8 13 21 34 ⋯

Naive recursive algorithm: $\Omega(\phi^n)$ (exponential time), where $\phi = (1+\sqrt{5})/2$ is the ***golden ratio***.

Golden Ratio

- Two quantities are in the **golden ratio** if the ratio of the sum of the quantities to the larger one equals the ratio of the larger one to the smaller.
- Two quantities a and b are said to be in the **golden ratio** ϕ if: $a+b/a = a/b = \phi$, if $a= \phi b$ then
- $\phi^2 - \phi - 1 = 0 \Rightarrow \phi = 1+\text{sqrt}(5) / 2$

  $= 1.6180339887\ldots$
- $F(n) = \phi^n - (1-\phi)^n / \text{sqrt}(5)$.

Computing Fibonacci numbers

Bottom-up:

- Compute F_0 , F_1 , F_2 , …, F_n in order, forming each number by summing the two previous.
- Running time:Θ(n).

Naive recursive squaring:

$F_n = \phi^n/\text{Sqrt}(5)$ rounded to the nearest integer.

- Recursive squaring: Θ(lg*n)* time.
- This method is unreliable, since floating-point arithmetic is prone to round-off errors.

Matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$

$\Big\}\ i, j = 1, 2, \ldots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Standard algorithm

for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do $c_{ij} \leftarrow 0$
      for $k \leftarrow 1$ **to** n
        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$

Divide-and-conquer algorithm

IDEA: n×n matrix = 2×2 matrix of (n/2)×(n/2) submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$r = ae+bg$

$s = af+bh$

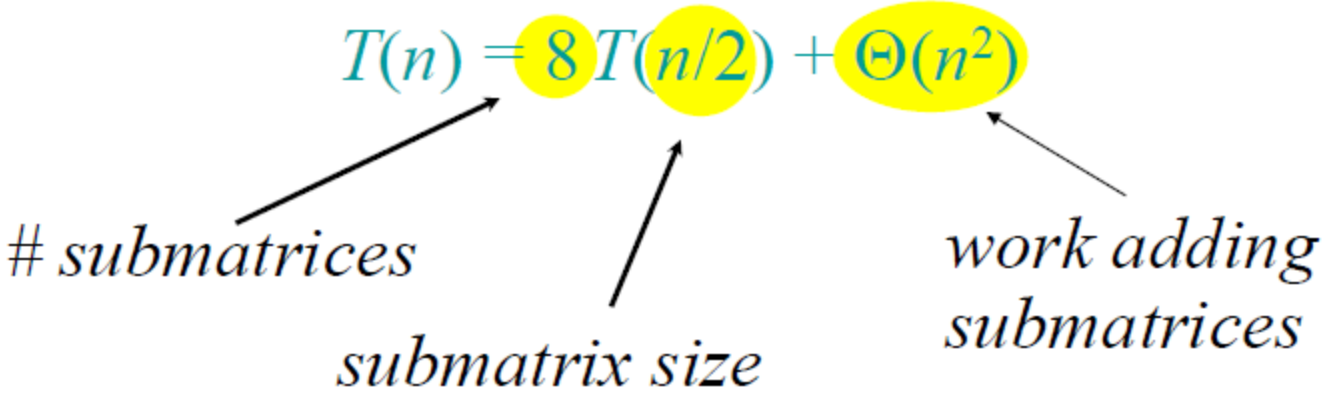
$t = ce+dg$

$u = cf+dh$

- 8 mults of (n/2)×(n/2) sub-matrices
- 4 adds of (n/2)×(n/2) sub-matrices

recursively

Analysis of D&C algorithm



$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow CASE1 \Rightarrow T(n) = \Theta(n^3)$.

- No better than the ordinary algorithm.

Strassen's idea

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

- Multiply 2×2 matrices with only 7 recursive mults.

$P_1 = a \cdot (f - h)$

$P_2 = (a + b) \cdot h$

$P_3 = (c + d) \cdot e$

$P_4 = d \cdot (g - e)$

$P_5 = (a + d) \cdot (e + h)$

$P_6 = (b - d) \cdot (g + h)$

$P_7 = (a - c) \cdot (e + f)$

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

Strassen's idea

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

- Multiply 2×2 matrices with only 7 recursive mults.

$P_1 = a \cdot (f - h)$

$P_2 = (a + b) \cdot h$

$P_3 = (c + d) \cdot e$

$P_4 = d \cdot (g - e)$

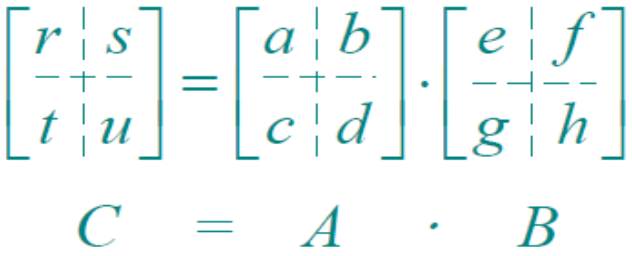
$P_5 = (a + d) \cdot (e + h)$

$P_6 = (b - d) \cdot (g + h)$

$P_7 = (a - c) \cdot (e + f)$

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ &= (a + d)(e + h) \\ &\quad + d(g - e) - (a + b)h \\ &\quad + (b - d)(g + h) \\ &= ae + ah + de + dh \\ &\quad + dg - de - ah - bh \\ &\quad + bg + bh - dg - dh \\ &= ae + bg \end{aligned}$$

Strassen's algorithm (1969)



$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

- **Divide:** Partition A and B into (n/2)×(n/2) submatrices. Form terms to be multiplied using + and –.
- **Conquer:** Perform 7 multiplications of (n/2)×(n/2) submatrices recursively.
- **Combine:** Form C using + and – on (n/2)×(n/2) submatrices.

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

Analysis of Strassen

$$T(n) = 7T(n/2) + \Theta(n^2)$$

- $n^{\log_b a} = n^{\log_2 7} = n^{2.807} \Rightarrow CASE\ 1 \Rightarrow T(n) = \Theta(n^{\lg 7})$.
- The number 2.807 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant.
- In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 32$ *or so.*

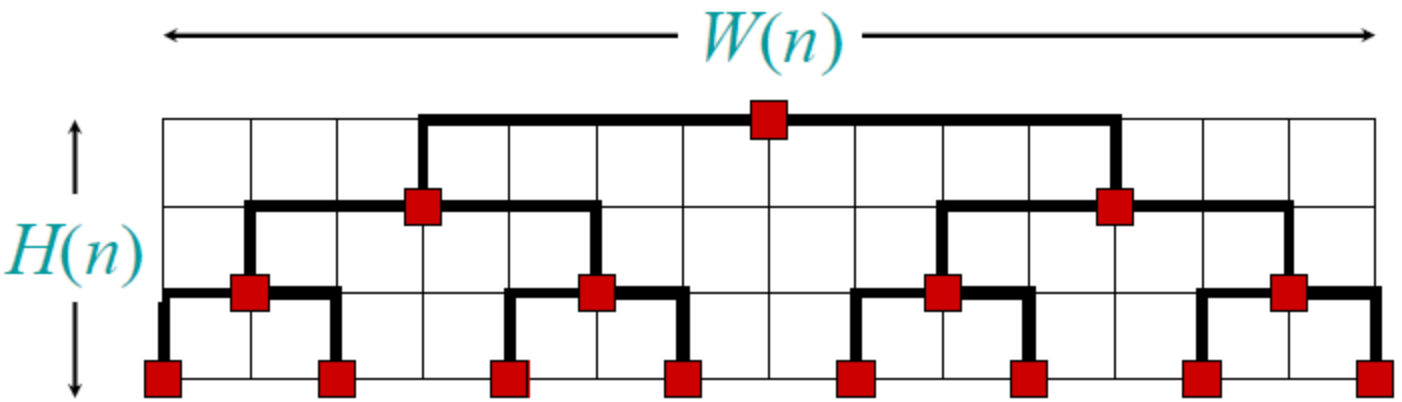
Coppersmith–Winograd algorithm (1987)

- Best to date (of theoretical interest only): $\Theta(n^{2.376\ldots})$.
  - asymptotically fastest known algorithm for square matrix multiplication as of 2008.
  - It is frequently used as a building block in other algorithms to prove theoretical time bounds.
  - it is not used in practice because it only provides an advantage for matrices so large that they cannot be processed by modern hardware.

"Toward an Optimal Algorithm for Matrix Multiplication", *SIAM News, Volume 38, Number 9, November 2005*

VLSI layout

Problem: Embed a complete binary tree with n leaves in a grid using minimal area.



$$H(n) = H(n/2) + \Theta(1) \qquad W(n) = 2\,W(n/2) + \Theta(1)$$
$$= \Theta(\lg n) \qquad = \Theta(n)$$

$$\text{Area} = \Theta(n \lg n)$$