

CS170: Lecture 2

Last Time: Place value is democratizing!

Like the printing press!

Reading, writing, arithmetic!

Input size/representation really matters!

Today: Chapter 2.

Divide and Conquer \equiv Recursive.

Lecture in one minute!

Integer Multiplication: Gauss plus recursion is magic!

$$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

$$T(n) = aT\left(\frac{n}{b}\right) + f(n).$$

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f\left(\frac{n}{b^i}\right)$.

Recursive (Divide and Conquer) Matrix Multiplication:

8 subroutine calls of size $n/2 \times n/2$

$$\rightarrow O(n^3).$$

Strassen:

7 subroutine calls of size $n/2 \times n/2$

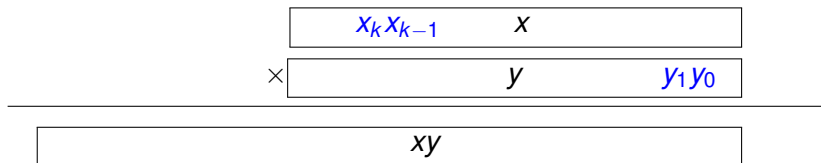
$$\rightarrow O(n^{\log_2 7}) \approx O(n^{2.8}).$$

Chapter 2.

Divide and conquer.

Definition of Multiplication.

n -bit numbers: x, y .



k th “place” of xy : coefficient of 2^k :

$$a_k = \sum_{i \leq k} x_i y_{k-i}.$$

$$x * y = \sum_{k=0}^{2n} 2^k a_k.$$

Number of “basic operations”:

$$\sum_{k \leq 2n} \min(k, 2n - k) = \Theta(n^2).$$

Recursive Algorithm for Multiplication.

Two n -bit numbers: x , y .

$$\begin{aligned}x &= \boxed{\begin{array}{c} x_L \end{array}} \boxed{\begin{array}{c} x_R \end{array}} = 2^{n/2}x_L + x_R \\y &= \boxed{\begin{array}{c} y_L \end{array}} \boxed{\begin{array}{c} y_R \end{array}} = 2^{n/2}y_L + y_R\end{aligned}$$

Multiplying out

$$\begin{aligned}x \times y &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\&= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

Four $n/2$ -bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.

Recurrence:

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

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

- (A) $\Theta(n)$.
- (B) $\Theta(n^2)$.
- (C) $\Theta(n^3)$.

Idea: Think about recursion tree.

A degree 4 tree of depth $\log_2 n$.

$\Theta(n^2)$ leaves or base cases.

One for each pair of digits!

Really? Unfolded recursion in my head?!?!

How did I really obtain bound? [Soon a formula.](#)

Demo

As number of bits double:

Elementary School Multiply:

$$O(n^2)$$

$$n \rightarrow 2n$$

$$\text{Runtime: } T = cn^2 \rightarrow T' = c(2n)^2 = 4(cn^2) = 4T$$

Python multiply:

$$n \rightarrow 2n$$

$$\text{Runtime: } T \rightarrow 3T.$$

$$\text{Asymptotics: } T = cn^w \rightarrow c((2n)^w) = T' = 3T = 3(cn^w).$$

$$\dots \rightarrow 2^w = 3. \text{ or } w = \log_2 3 \approx 1.58.$$

$$\text{Python multiply: } O(n^{\log_2 3})$$

Much better than grade school.

Multiply Complex Numbers

$$(3 + 2i)(4 + 5i) = 12 + (15 + 8)i + 10i^2$$

Recall, $i^2 = -1$, so simplifying

$$(12 - 10) + 22i = 2 + 22i.$$

What about $(32765 + 219898i)(413764 + 511110i)$?

Gauss's trick.

$$(a + b i)(c + d i) = (ac - bd) + (ad + bc) i.$$

Four multiplications: ac , bd , ad , bd .

Drop the i :

$$P_1 = (a + b)(c + d) = ac + ad + bc + bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac - bd) = P_2 - P_3.$$

$$(ad + bc) = P_1 - P_2 - P_3.$$

Only three multiplications. An extra addition though!

Which is harder of multiplication or addition?

Multiplication!

Faster Algorithm for Multiplication.

Two n -bit numbers: x, y .

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$
$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L, x_L y_R + x_R y_L, x_R y_R$.

Used four $\frac{n}{2}$ -bit multiplications: $x_L y_L, x_L y_R, x_R y_L, x_R y_R$.

Can you compute three terms with 3 multiplications?

(A) Yes.

(B) No

(A) Yes.

Three multiplications and faster algorithm.

Two n -bit numbers: x, y .

$$\begin{aligned}x &= 2^{n/2}x_L + x_R & ; & & y &= 2^{n/2}y_L + y_R \\x \times y &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R\end{aligned}$$

Need 3 terms: $x_L y_L, x_L y_R + x_R y_L, x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L, P_3 = x_R y_R. (x_L y_R + x_R y_L) = P_1 - P_2 - P_3$
3 multiplications!

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

Technically: $\frac{n}{2} + 1$ bit multiplication. Don't worry.

Analysis of runtime.

Recurrence for “fast algorithm”.

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n)$$

Runtime is

- (A) $\Theta(n)$
- (B) $\Theta(n^2)$
- (C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.

More soon.

So multiplication algorithm with ..

$$T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\dots})!!!!$$

But: all digits have to multiply each other!

They do! $(a + b)(c + d) = ac + ad + bc + bd$

4 products from one multiplication!

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

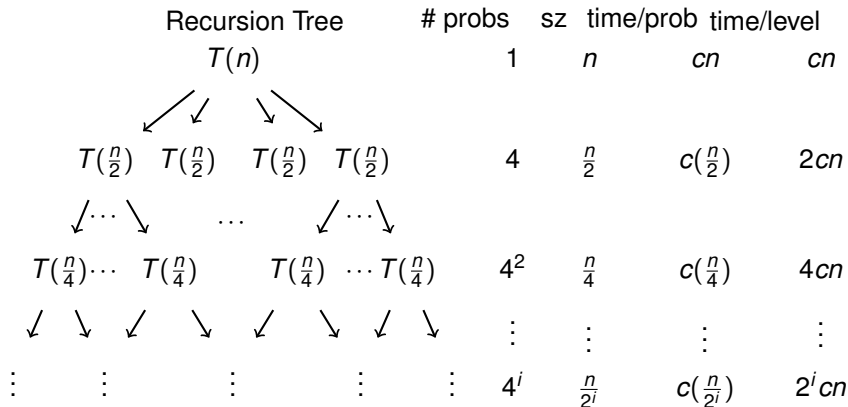
Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Yes!

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

Solving recurrences.

$$T(n) = 4T\left(\frac{n}{2}\right) + cn; \quad T(1) = c$$



$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log_2 n} = 2^{2 \log_2 n} = n^2$ base case problems. size 1. Work/Prob: c

Work: cn^2 .

Total Work: $cn + 2cn + 4cn + \dots + cn^2 = O(n^2)$. Geometric series.

Fast multiplication.

$$T(n) = 3T\left(\frac{n}{2}\right) + cn; \quad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \quad \downarrow \quad \searrow$ $T\left(\frac{n}{2}\right) \quad T\left(\frac{n}{2}\right) \quad T\left(\frac{n}{2}\right)$	3	$\frac{n}{2}$	$c\left(\frac{n}{2}\right)$	$\left(\frac{3}{2}\right)cn$
$\swarrow \cdots \searrow \quad \dots \quad \swarrow \cdots \searrow$ $T\left(\frac{n}{4}\right) \cdots T\left(\frac{n}{4}\right) \quad T\left(\frac{n}{4}\right) \cdots T\left(\frac{n}{4}\right)$	3^2	$\frac{n}{4}$	$c\left(\frac{n}{4}\right)$	$\left(\frac{3}{2}\right)^2 cn$
$\swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow$ $\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots$	\vdots	\vdots	\vdots	\vdots
$\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots$	3^i	$\frac{n}{2^i}$	$c\left(\frac{n}{2^i}\right)$	$\left(\frac{3}{2}\right)^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c . Work: $cn^{\log_2 3}$.

Total Work: $cn + \left(\frac{3}{2}\right)cn + \dots + cn^{\log_2 3} = O(n^{\log_2 3})$ Geometric series.

Divide and Conquer: In general.

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d); \quad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/lvl
$T(n)$	1	n	cn^d	cn^d
$\swarrow \quad \downarrow \quad \searrow$ $T\left(\frac{n}{b}\right) \quad T\left(\frac{n}{b}\right) \quad T\left(\frac{n}{b}\right)$	a	$\frac{n}{b}$	$c\left(\frac{n}{b}\right)^d$	$\left(\frac{a}{b^d}\right)cn^d$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$ $T\left(\frac{n}{b^2}\right) \cdots T\left(\frac{n}{b^2}\right) \quad T\left(\frac{n}{b^2}\right) \cdots T\left(\frac{n}{b^2}\right)$	a^2	$\frac{n}{b^2}$	$c\left(\frac{n}{b^2}\right)^d$	$\left(\frac{a}{b^d}\right)^2 cn^d$
$\swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow$ $\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots$	\vdots	\vdots	\vdots	\vdots
$\vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots$	a^i	$\frac{n}{b^i}$	$c\left(\frac{n}{b^i}\right)^d$	$\left(\frac{a}{b^d}\right)^i cn^d$

$\frac{n}{b^i} = 1$ when $i = \log_b n \implies$ Depth: $k = \log_b n$.

Level i work: $\left(\frac{a}{b^d}\right)^i n^d$.

Master's Theorem

Depth: $\log_b n$.

Level i work:

$$\left(\frac{a}{b^d}\right)^i n^d.$$

Total:

$$n^d \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i$$

Geometric series: If $\frac{a}{b^d} < 1$ ($d > \log_b a$), first term dominates

$$O(n^d),$$

if $\frac{a}{b^d} > 1$ ($d < \log_b a$), last term dominates.

$$O(n^{\log_b a}),$$

and if $\frac{a}{b^d} = 1$ ($d = \log_b a$), then all terms are the same

$$O(n^d \log_b n).$$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$

We have

$$d > \log_b a \quad T(n) = O(n^d)$$

$$d < \log_b a \quad T(n) = O(n^{\log_b a})$$

$$d = \log_b a \quad T(n) = O(n^d \log_b n).$$

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.

$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.

$1 > \log_2 1 = 0 \implies T(n) = O(n)$

$T(n) = 2T(\frac{n}{2}) + O(n)$ $a = 2$, $b = 2$, and $d = 1$.

$1 = \log_2 2 \implies T(n) = O(n \log n)$

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...Unite! Resist!

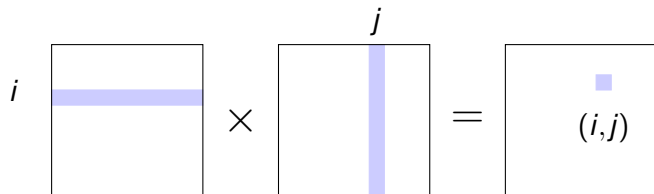
Strassen: Divide! conquer!

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of i th row with j th column.



$$Z_{ij} = \sum_{k=1}^n X_{ik} Y_{kj}.$$

Runtime? $O(n^2)$? $O(n^3)$? $O(n^3)$!

Divide and Conquer

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

A, B, C, \dots, H are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems? $AE, BG, AF, BH, CE, DG, CF, DH$.

Recurrence?

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

Masters: $O(n^{\log_2 8}) = O(n^3)$.

Strassen

Compute

$$P_1 = A(F - H) \quad P_5 = (A + D)(E + H)$$

$$P_2 = (A + B)H \quad P_6 = (B - D)(G + H)$$

$$P_3 = (C + D)E \quad P_7 = (A - C)(E + F)$$

$$P_4 = D(G - E)$$

$$\left[\begin{array}{ll} AE + BG = P_5 + P_4 - P_2 + P_6 & AF + BH = P_1 + P_2 \\ CE + DG = P_3 + P_4 & AF + BH = P_1 + P_5 - P_3 + P_7 \end{array} \right]$$

7 multiplies! Recurrence?

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

From Masters:

(A) $O(n^2)$? (B) $O(n^{\log_2 7} \log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

(C) $O(n^{\log_2 7}) = O(n^{2.81\dots})$ Way better than $O(n^3)$.

Commonly used in practice!

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36\dots$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

$$\text{Masters: } O(n^{\log_k k^\omega}) = O(n^{\omega \log_k k}) = O(n^\omega)$$

State of the art: k is very very large... e.g., 10^{100} ...but still a constant.

Based on complicated recursive constructions.

Improvement for constant + recursion gives better algorithm!

Example:

Gauss + recursion \implies faster multiplication.

Strassen's 7 multiplies + recursion \implies faster matrix multiplication.

Lecture in five!

Gauss plus recursion is magic!

$$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:

8 subroutine calls of size $n/2 \times n/2$

$\rightarrow O(n^3)$.

Strassen:

7 subroutine calls of size $n/2 \times n/2$

$\rightarrow O(n^{\log_2 7}) \approx O(n^{2.8})$.