**Definition of Multiplication.**

Two $n$-bit numbers: $x, y$.

- $x = x_L x_R = 2^{n/2} x_L + x_R$
- $y = y_L y_R = 2^{n/2} y_L + y_R$

Multiplying out:

$$x \times y = \sum_{k \leq n} a_k y_{n-k}.$$

Number of “basic operations”:

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

**Recursive Algorithm for Multiplication.**

Two $n$-bit numbers: $x, y$.

$$x = x_L x_R = 2^{n/2} x_L + x_R$$

$$y = y_L y_R = 2^{n/2} y_L + y_R$$

Multiplying out:

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

$$= 2^n x_L y_L + 2^{n/2} (x_L y_R + x_R y_L) + x_R y_R$$

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

Recurrence:

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

**Recurrence for recursive algorithm.**

Recurrence:

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

$T(n)$ 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.

---

**Chapter 2.**

Divide and conquer.

**Lecture in one minute!**

Integer Multiplication: Gauss plus recursion is magic!

$$O(n^2) \rightarrow O(n^{2.8}) \approx O(n^{1.58})$$

Double size, time grows by a factor of 3.

Master’s theorem: understand the recursion tree!

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

- Branching by $a$
- Diminishing by $b$
- Working by $O(f(n))$

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

Recursive (Divide and Conquer) Matrix Multiplication:

- Idea: Think about recursion tree.
- Today: Chapter 2.
- Input size/representation really matters!
- Multiplying out $k$ th “place” of $xy$.
- $x \times y$.
- $$xy = \sum_{i=0}^{n} x_i y = \sum_{i=0}^{n} a_k 2^k.$$
Don’t worry.

But all added up.

Idea: number of base cases is

An extra addition though!

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

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 x_R y_R + x_R y_L + x_L y_R

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

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 (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.

Can you compute three terms with 3 multiplications?

(A) Yes.

(B) No

(A) Yes.

Gauss’s trick.

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

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: \(P_2 = ac, P_3 = bd\).

\((ac + bd) = P_2 - P_3\).

Only three multiplications. An extra addition though!

Which is harder of multiplication or addition?

Multiplication!

Analysis of runtime.

Recurrence for “fast algorithm”.

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

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 ..

\(T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{log_2 3}) = \Theta(n^{1.58}) \ldots \).

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. \((a^b)^c = a^{bc} = a^b = (a^c)^b\).
Definition of log: \(a = b^{\log_b a}\)
Logarithm Quiz: \(a^{\log_b a} = n^{\log_b n}\)?
Yes!

\[ a^{\log_b n} = (b^{\log_b a})^{\log_b n} = (b^{\log_b n})^{\log_b a} = n^{\log_b a} \]

Divide and Conquer: In general.

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

Recursion Tree
\[
\begin{array}{c}
T(n) \\
\text{# probs} \\
\text{sz} \\
\text{time/prob} \\
\text{time/level} \\
\hline
1 \\
\frac{n}{b} \\
\frac{n}{b^2} \\
\vdots
\end{array}
\]

\[
T\left(\frac{n}{b}\right) + T\left(\frac{n}{b^2}\right) + \cdots + T\left(\frac{n}{b^k}\right) = \sum_{i=0}^{\log_b n} a^{n/b^i} \]

\[
\frac{n^d}{\log n} = 1 \text{ when } i = \log_b n \implies \text{Depth: } d = \log_b n.
\]

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

\[ O(n^d) \]

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

\[ O(n^{dlog_b a}) \]

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

\[ O(n^d log_b n) \]

Solving recurrences.

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

Recursion Tree
\[
\begin{array}{c}
T(n) \\
\text{# probs} \\
\text{sz} \\
\text{time/prob} \\
\text{time/level} \\
\hline
1 \\
\frac{n}{2} \\
\frac{n}{4} \\
\vdots
\end{array}
\]

\[
T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + \cdots + T\left(\frac{n}{2^k}\right) = \sum_{i=0}^{\log_2 n} a^{n/2^i} \]

\[
\frac{n^2}{\log^2 n} = 1 \text{ when } i = \log_2 n \implies \text{Depth: } d = \log_2 n.
\]

Geometric series: If \(\frac{a}{b} < 1 (d > \log_2 a), \) first term dominates

\[ O(n^d) \]

if \(\frac{a}{b} > 1 (d < \log_2 a), \) last term dominates.

\[ O(n^{dlog_2 a}) \]

and if \(\frac{a}{b} = 1 (d = \log_2 a), \) then all terms are the same

\[ O(n^d log_2 n) \]

Fast multiplication.

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

Recursion Tree
\[
\begin{array}{c}
T(n) \\
\text{# probs} \\
\text{sz} \\
\text{time/prob} \\
\text{time/level} \\
\hline
1 \\
\frac{n}{2} \\
\frac{n}{4} \\
\vdots
\end{array}
\]

\[
T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + \cdots + T\left(\frac{n}{2^k}\right) = \sum_{i=0}^{\log_2 n} a^{n/2^i} \]

\[
\frac{n^2}{\log^2 n} = 1 \text{ when } i = \log_2 n \implies \text{Depth: } d = \log_2 n.
\]

Geometric series: If \(\frac{a}{b} < 1 (d > \log_2 a), \) first term dominates

\[ O(n^d) \]

if \(\frac{a}{b} > 1 (d < \log_2 a), \) last term dominates.

\[ O(n^{dlog_2 a}) \]

and if \(\frac{a}{b} = 1 (d = \log_2 a), \) then all terms are the same

\[ O(n^d log_2 n) \]

Master's Theorem: examples.

For a recurrence \( T(n) = aT(n/b) + O(n^d) \)
We have
\[ d > \log_b a \quad T(n) = O(n^d) \]
\[ d = \log_b a \quad T(n) = O(n^{dlog_b a}) \]
\[ d < \log_b a \quad T(n) = O(n^{dlog_b a}) \]

\[
\begin{align*}
T(n) &= 4T\left(\frac{n}{2}\right) + O(n^2) \\
&= O(n^d) \\
&= O(n^{dlog_2 a}) \\
&= O(n^{dlog_2 n}) \\
&= O(n^{dlog_2 n}) \\
&= O(n^2 log_2 n) \\
&= O(n^{2log_2 n}) \\
&= O(nlog n)
\end{align*}
\]
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,...,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: \( O(n^{\log_2 8}) = O(n^3). \)

Lecture in five!
Gauss plus recursion is magic!
\( O(n^2) \rightarrow O(n^{\log_2 7}) = O(n^{1.81...}) \)
Double size, time grows by a factor of 3.
Master’s theorem: understand the recursion tree!
Branching by \( a \) diminish by \( b \)
working by \( O(f(n)) \).
Leaves: \( n^{a \log_b \frac{a}{b}} \).
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 \( n/2 \times n/2 \)
\( \rightarrow O(n^{\log_2 7}) \approx O(n^{2.81...}). \)

Current State of the Art: Matrix multiplication.

\( k \times k \) multiplication in \( k^\omega \) multiplications where \( \omega = 2.36... \)
E.g., Strassen: \( 2 \times 2 \) multiplication in \( 2^{\log_2 7} = 7 \) multiplications.

\[ T(n) = k^\omega T\left(\frac{n}{k}\right) + O(n^2) \]
Masters: \( O(n^{\log_2 k}) = O(n^{\log_2 \omega}) = O(n^{\omega}) \)

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.

Strassen

Matrix Multiplication

\[ X \text{ and } Y \text{ are } n \times n \text{ matrices.} \]
\[ Z = XY, \]
\[ Z_{ij} \text{ is dot product of } i \text{th row with } j \text{th column.} \]
\[ Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}. \]

Runtime? \( O(n^2) ? \ O(n^3) ? \ldots O(n^3)! \)

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

Commonly used in practice!

Strassen

Matrix multiplication.
Strassen, 1968, visiting Berkeley.
Berkeley... Unite! Resist!
Strassen: Divide! conquer!

Strassen

Compute
\[ P_1 = A(F - H), \quad P_5 = (A + D)(E + H) \]
\[ P_2 = (A + B)H, \quad P_6 = (B - D)(G + H) \]
\[ P_3 = (C + D)E, \quad P_7 = (A - C)(E + F) \]
\[ P_4 = D(G - E) \]

\[ \begin{bmatrix}
AE + BG & P_1 + P_2 - P_3 + P_6 \\
AF + BH & P_1 + P_2 \\
CE + DG & P_3 + P_2 \\
AF + BH & P_1 + P_3 - P_2 + P_7 \\
\end{bmatrix} \]

7 multiplies! Recurrence?
\[ T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) \]
From Masters:
(A) \( O(n^2) ? \ (B) O(n^{p \log_2 n}) ? \ (C) T(n) = O(n^{p \log_2 n}) \)

Leaf subproblems dominate runtime!
\[ \begin{align*}
(A) \ O(n^2) \ ? \\
(B) \ O(n^{p \log_2 n}) \ ? \\
(C) \ O(n^{p \log_2 n}) \ = \ O(n^{2.81...}). \ \text{Way better than } O(n^2). \\
\end{align*} \]

Commonly used in practice!