CS170: Lecture 2

Chapter 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 = Recursive.

Definition of Multiplication.

Two n-bit numbers: x, y.

\[
\begin{array}{c|cc}
\hline
& x & y \\
\hline
x_L & x & y_L \\
\hline
x_R & x & y_R \\
\hline
\end{array}
\]

kth "place" of xy: coefficient of \(2^k\):

\[a_k = \sum_{i=k}^{n} x_i y_{k-i} \]

\[x \times y = \sum_{k=0}^{2^n} 2^k a_k \]

Number of "basic operations":

\[\sum_{k=0}^{2^n} \min(k, 2n - k) = \Theta(n^2)\]

Recursive Algorithm for Multiplication.

Integer Multiplication: Gauss plus recursion is magic!
\(O(n^2) \rightarrow O(n^{\log_2 3}) = O(n^{1.58...})\)
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: \(\sum a^i f\left(\frac{n}{b^i}\right)\).

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 \(n/2 \times n/2\)
\(\rightarrow O(n^{\log_2 7}) \approx O(n^{2.8})\).
Recurrence for recursive algorithm.

Recurrence:

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

\( T(n) \) is

(A) \( \Theta(n) \).

(B) \( \Theta(n^2) \).

(C) \( \Theta(n^3) \).

Idea: Think about recursion tree.
A degree 4 tree of depth \( \log_2 n \).

4\( \log_2 n \) = \( 2^4 \log_2 n \) = \( 2^{\log_2 n} \) = \( n^2 \).

\( \Theta(n^2) \) 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.

TBH, unfolded recurrence in head. Don’t remember formulas.

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

\((ad + bc) = P_1 - P_2 - P_3 \)

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.

Demo

As number of bits double:

Elementary School Multiply:

\[ O(n^2) \]

\( n \rightarrow 2n \)

Runtime: \( T = cn^2 \rightarrow T' = c(2n)^2 = 4(cn^2) = 4T \)

Python multiply:

\( n \rightarrow 2n \)

Runtime: \( T \rightarrow 3T \).

Asymptotics: \( T = cn^m \rightarrow c(2n)^m = T' = 3T = 3(cn^m) \).

\( \cdots \rightarrow 2^n = 3 \) or \( w = \log_2 3 \approx 1.58 \).

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

Much better than grade school.

Multiply Complex Numbers

Recall, \( i^2 = -1 \), so simplifying

\[
(12 - 10) + 22i = 2 + 22i.
\]

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

\[
(3 + 2i)(4 + 5i) = 12 + (15 + 8)i + 10i^2
\]

Recurrence for recursive algorithm.

Recurrence:

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

\( T(n) \) is

(A) \( \Theta(n) \).

(B) \( \Theta(n^2) \).

(C) \( \Theta(n^3) \).

Idea: Think about recursion tree.
A degree 4 tree of depth \( \log_2 n \).

4\( \log_2 n \) = \( 2^4 \log_2 n \) = \( 2^{\log_2 n} \) = \( n^2 \).

\( \Theta(n^2) \) 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.

TBH, unfolded recurrence in head. Don’t remember formulas.

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

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

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.
Analysis of runtime.
Recurrsence for “fast algorithm”.

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

Runtime is

- (A) \( \Theta(n) \)
- (B) \( \Theta(n^2) \)
- (C) \( \Theta(n \log n) \)

(C) Idea: number of base cases is \( n^{\log_2 3} \),
\[
3^{\log_2 n} = (2^{\log_2 3})^{\log_2 n} = 2^{\log_2 3 \cdot \log_2 n} = n^{\log_2 3}.
\]

So multiplication algorithm with...

\[
T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log n) = \Theta(n^{1.58...})!!!
\]

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. \((a^b)^c = a^{bc} = a^b \cdot (a^c)^b\).
Definition of \(\log\): \(a = b^{\log_b a}\).
Logarithm Quiz: \(\log_2 n = \log_{2^2} n^2\)?
Yes!
\[
\log_2 n = (\log_{2^2} n^2) = \log_{2^{\log_2 n}} (2^{\log_2 n}) = n^{\log_2 2} = n.
\]

Solving recurrences.
\[ T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) \]
\[ T(1) = c \]

Recursion Tree
1. \(n\) time/prob time/level
2. \(cn\)
3. \(cn\)
4. \(cn^2\)

\[
\sum_{i=1}^{\log_2 n} i \cdot cn^i = O(n^2). \text{ Geometric series.}
\]

Fast multiplication.
\[ T(n) = 3T\left(\frac{n}{2}\right) + \Theta(n) \]
\[ T(1) = c \]

Recursion Tree
1. \(n\) time/prob time/level
2. \(cn\)
3. \(cn^2\)
4. \(cn^3\)

\[
\sum_{i=1}^{\log_2 n} i \cdot cn^i = O(n^2). \text{ Geometric series.}
\]

Divide and Conquer: In general.
\[ T(n) = aT\left(\frac{n}{2}\right) + \Theta(n^d) \]
\[ T(1) = c \]

Recursion Tree
1. \(n\) time/prob time/level
2. \(cn^d\)
3. \(cn^{2d}\)
4. \(cn^{3d}\)

\[
\sum_{i=1}^{\log_2 n} i \cdot cn^{di} = O(n^d). \text{ Geometric series.}
\]

Solving recurrences.
\[ T(n) = 4T\left(\frac{n}{2}\right) + cn \]
\[ T(1) = c \]

Recursion Tree
1. \(n\) time/prob time/level
2. \(cn\)
3. \(cn^2\)
4. \(cn^3\)

\[
\sum_{i=1}^{\log_2 n} i \cdot cn^{di} = O(n^d). \text{ Geometric series.}
\]
### Master's Theorem

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

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

- **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 $O(n^{(\log_b a)}d)$.
  - If $\frac{a}{b} = 1 (d = \log_b a)$, then all terms are the same $O(n^{(\log_b a)}d)$.

### 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.

$$i \times j = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$  

Runtime? $O(n^3)$? $O(n^2)$? $n^2$ entries in $Z$, $O(n)$ time per entry. $O(n^3)$

### Divide and Conquer

**Strassen**


Berkeley...Unite! Resist!

Strassen: Divide! conquer!

**Strassen**

$$P_1 = A(F - H) \quad P_2 = (A + D)(E + H)$$

$$P_3 = (A + B)H \quad P_4 = (B - D)(G + H)$$

$$P_5 = (C + D)E \quad P_6 = (A - C)(E + F)$$

$$P_7 = D(G - E)$$

$$[AE + BG = P_5 + P_4 - P_2 + P_6 \quad AF + BH = P_1 + P_3]$$

$$[CE + DG = P_3 + P_4 \quad AF + BH = P_1 + P_3]$$

$$P_2 + P_3 + P_4 = \begin{bmatrix} AE + AH + DE + DH \quad (DG - DE) - AH - BH + BG - BH - DG - DH \quad AE + BG \end{bmatrix}$$

$7$ multiplies! Recurrence?

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

From Masters:

(A) $O(n^3)$? (B) $O(n^{\log_2 7})$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

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

Commonly used in practice!

### Master's Theorem: examples.

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

- We have $d > \log_b a$ $T(n) = O(n^d)$
- $d < \log_b a$ $T(n) = O(n^{(\log_b a)}d)$
- $d = \log_b a$ $T(n) = O(n^{(\log_b a)}d)$

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

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

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

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

$1 = \log_2 2 \rightarrow T(n) = O(n \log n)$
Current State of the Art: Matrix multiplication.

\[ k \times k \text{ multiplication in } k^\omega \text{ multiplications where } \omega = 2.37... \]

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

\[ T(n) = k^\omega T(\frac{n}{k}) + O(n^2) \]

Masters: \( O(n^{\log_2 k} ) = O( n^{\omega \log_k k} ) = O(n^\omega) \)

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 one minute!

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: \( n^{\log_2 a} \), Work: \( \sum_a f(\frac{n}{b^i}) \).

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 \( n/2 \times n/2 \)

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