

CS 331, Fall 2026

Lecture 3 (8/31)

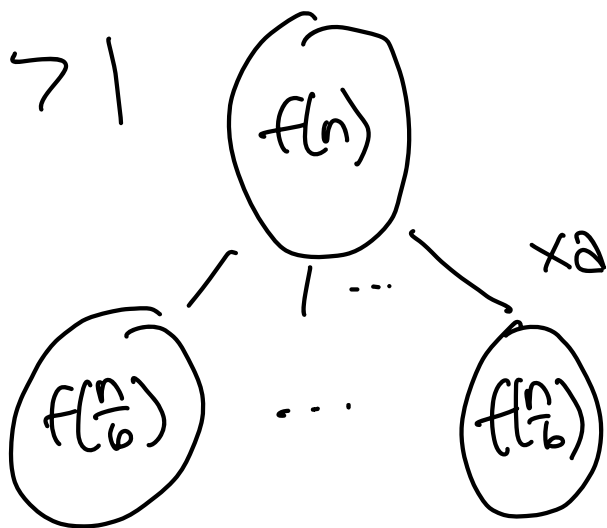
Today: - Matrices
- Fibonacci
- FFT

FAQ

The "leaves-heavy" case of recursion trees.

$$T(n) = a T\left(\frac{n}{b}\right) + \underbrace{\Theta(n^c \log^d n)}_{f(n)}$$

$$\text{Ratio} = \frac{a}{b^c} > 1$$



$$n^c \dots$$

$$a \frac{n^c}{b^c} \dots$$

$$\text{Cost} = \# \text{ leaves} = a^{\log_b n} = n^{\log_b a}$$

Key fact: $\log_b n = \frac{\log n}{\log b}$

Q: Why can't $O(\dots)$ hide the $\frac{2}{b^c}$ factor?

Warmup: Stock market ↙ prices on n days

Input: L is a list of n elements in $\mathbb{R}$

Output: (i, j) with $1 \leq i \leq j \leq n$ must buy
then sell

Maximizing $L[j] - L[i]$
sell buy

Example

	Day						
	1	2	3	4	5	6	7
GME Stocks	8	12	4	9	17	1	13

How to maximize profit?

Idea 1: Try everything.

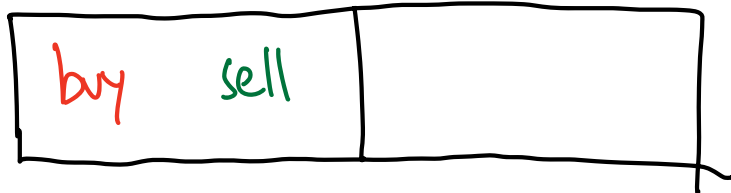
For $j \in [n]$:
For $i \in [j]$: $\left. \vphantom{\begin{matrix} \text{For } j \in [n]: \\ \text{For } i \in [j]: \end{matrix}} \right\} O(n^2)$

... # keep track of running max

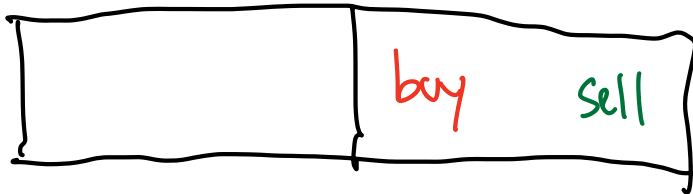
Idea 2: Divide-and-conquer

$T(n)$ = runtime on length- n input

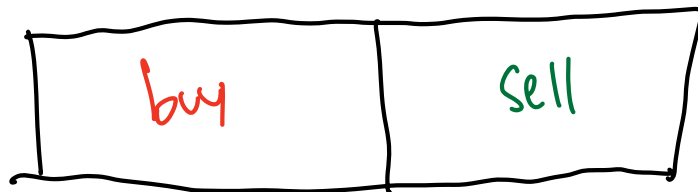
Cases:



$T(\frac{n}{2})$



+ $T(\frac{n}{2})$



+ $O(n)$

running min

running max

= $O(n \log(n))$

Idea 3: $O(n)$ next time ☺

Matrix multiplication (Part II, Section 6.1)

Warmup: Vector-vector

"inner product"
aka
"dot product"

$$1 \quad \begin{array}{|c|c|c|c|} \hline a(1) & a(2) & \dots & a(n) \\ \hline \end{array} \quad \begin{array}{|c|} \hline b(1) \\ \hline b(2) \\ \hline \vdots \\ \hline b(n) \\ \hline \end{array} \quad n = \sum_{i \in [n]} a(i) b(i)$$

n

$O(n)$ time.

linear time

Notation:

In this class, vectors are lower-case and columns ($n \times 1$)

Dot product of $a, b \in \mathbb{R}^{n \times 1}$:

$$\langle a, b \rangle = a^T b = \sum_{i \in [n]} a(i) b(i)$$

Matrix-matrix: dot every row/col pair

$$\begin{array}{c}
 \text{ith row} \left(\begin{array}{c} \vdots \\ \text{--- } A_{i:}^T \text{ ---} \\ \vdots \end{array} \right) \left(\begin{array}{c} \text{jth column} \\ \vdots \\ B_{:j} \\ \vdots \end{array} \right) = \left(\begin{array}{c} A_{i:}^T B_{:j} \end{array} \right) \\
 \begin{array}{c} A \\ (n \times d) \end{array} \quad \begin{array}{c} B \\ (d \times k) \end{array} \quad \begin{array}{c} AB \\ (n \times k) \end{array}
 \end{array}$$

Applications: entirety of modern ML/data science.

Intuition: $n \left(\begin{array}{c} \text{cat} \\ \text{dog} \\ \text{bird} \\ \vdots \end{array} \right)_d$ $n = \# \text{ examples}$
 $d = \# \text{ features}$

Examples

$$\begin{pmatrix} 1 & 2 \\ -3 & -1 \end{pmatrix} \begin{pmatrix} 4 & 1 \\ 6 & -2 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} 7 & 2 & 6 \\ 1 & 3 & 5 \\ 1 & 2 & 3 \end{pmatrix} \quad \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix} (1 \ 2 \ 3)$$

Naive matrix-matrix: $\begin{pmatrix} n \times d \end{pmatrix} \begin{pmatrix} d \times k \end{pmatrix}$

$$\underbrace{O(d)}_{\text{per dot product}} \times \underbrace{nk}_{\text{\# entries}} = O(ndk).$$

If $n=d=k$, this is $O(n^3)$. Linear time = $O(n^2)$

Aside

Block matrix multiplication works like you think!

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

$A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22}$ are matrices

Intuition: Splitting dot product

$$A_{1:}^T B_{:,1} = [A_{11}]_{1:}^T [B_{11}]_{:,1} + [A_{12}]_{1:}^T [B_{12}]_{:,1}$$

$$\begin{aligned} \text{Naive recursion: } T(n) &= 8T\left(\frac{n}{2}\right) + O(n^2) \\ &= O(n^3) \end{aligned}$$

Strassen recursion: $T(n) = 7T(\frac{n}{2}) + O(n^2) = O(n^{2.807...})$

World record: $T(n) = O(n^{2.3714...})$ (galactic algorithm...)

When can we do better?

Matrix-Vector

let A is $n \times n$
 b is $n \times 1$

(input size:
 $O(n^2 + n)$
 $= O(n^2)$)

We can compute Ab in time:

$$(O(n) \text{ per entry}) \times n = O(n^2)$$

Linear time ☺

Can we do better if A fixed, b is input?

output

Ab

=

A

input

b

preprocess somehow...

→ HW 1, P3

→ FFT (today)

Fibonacci (Part II, Section 6.3)

Classic interview question / "gotcha"

Input: $n \in \mathbb{N}$

Output: F_n , n^{th} Fibonacci number

Recursive definition:

$$F_0 = 1, F_1 = 2, F_2 = 3, F_3 = 5, F_4 = 8$$

$$\dots F_n = F_{n-1} + F_{n-2}$$

Observation: $F_n = \exp(\Theta(n))$

Aside In fact,
 $F_n \approx \phi^n$, where
 $\phi \approx 1.618$ is "golden ratio"

Naive (??): $\exp(\Theta(n))$ time

Naive: n additions... but #s are $O(n)$ digits
 $\rightarrow O(n^2)$ time

Improvement using matrix multiplication

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x+y \\ x \end{pmatrix}$$

Main claim: $A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$

Proof (induction):

$$A^{n+1} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = A \cdot A^n \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} F_n + F_{n-1} \\ F_n \end{pmatrix}$$

$\nwarrow F_{n+1}$

Algo: 1) Compute $A, A^2, A^4, \dots, A^{2^{\lfloor \log_2(n) \rfloor}}$ $O(n \log(n))$

2) Write n in binary $O(\log(n))$

e.g. $57 = 32 + 16 + 8 + 1$

3) Multiply those matrix powers $O(n \log(n))$

e.g. $A^{57} = A^{32} \cdot A^{16} \cdot A^8 \cdot A$

Fast Fourier Transform (Part II, Section 6.2)

Let $n = 2^k$, $k = 1, 2, \dots$

F_n is $n \times n$ Complex matrix.

DFT_n(b): return $F_n b$ in time $O(n \log n)$

Applications: faster multiplication
signal processing (learn "spectrum" of waves)

Aside

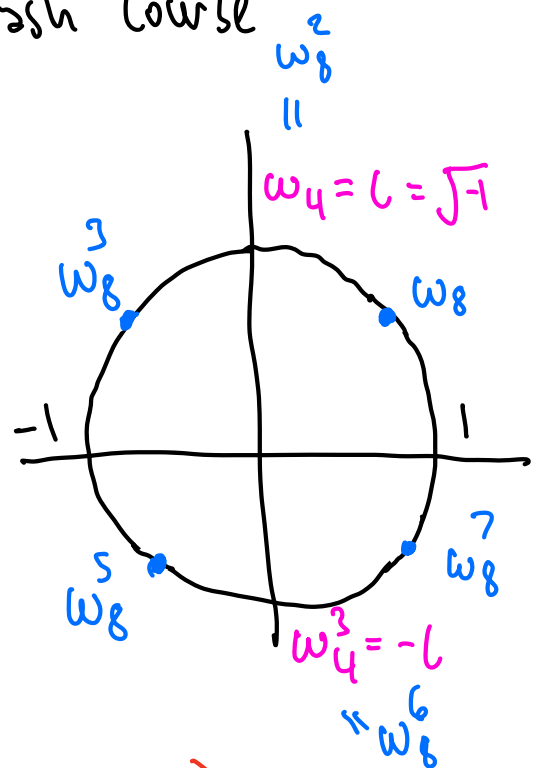
Complex numbers crash course

$$e^{i\theta} = \cos(\theta) + i \sin(\theta)$$

$$\omega_n = e^{i \cdot \frac{2\pi}{n}}$$

" n th root of unity"

$$\frac{2\pi}{n} = \text{rotate } \frac{1}{n}^{\text{th}} \text{ of a circle}$$



Remember: $\omega_n^n = 1$. (full lap around circle)

So, what is F_n ?

$$F_n(i,j) = \omega_n^{(i-1)(j-1)}$$

Interpretation:

i^{th} row of F_n
goes around unit circle,
 $\frac{(i-1)}{n} \cdot 2\pi$ at a time.

$$F_1 = (1) \quad \omega_1 = 1$$

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \omega_2 = -1$$

$$F_4 = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^0 & \omega^2 \\ \omega^0 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

$$\omega \equiv \omega_4 = i$$

$$F_8 = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^1 & \omega^4 & \omega^7 & \omega^2 & \omega^5 \\ \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 & \omega^0 & \omega^4 \\ \omega^0 & \omega^5 & \omega^2 & \omega^7 & \omega^4 & \omega^1 & \omega^6 & \omega^3 \\ \omega^0 & \omega^6 & \omega^4 & \omega^2 & \omega^0 & \omega^6 & \omega^4 & \omega^2 \\ \omega^0 & \omega^7 & \omega^6 & \omega^5 & \omega^4 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix}$$

$$\omega \equiv \omega_8 = \exp\left(i \cdot \frac{2\pi}{8}\right)$$

Recall

$$\omega_8^2 = \omega_4$$

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

time to
compute $F_n b$

Aside

Two ways of thinking about Ab

Way 1: $(Ab)(i) = A_{i,:} b$

Way 2:

note: order
doesn't matter!

$$\begin{pmatrix} | & | & & | \\ A_{:,1} & A_{:,2} & \dots & A_{:,n} \\ | & | & & | \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$= b_1 A_{:,1} + b_2 A_{:,2} + \dots + b_n A_{:,n}$$

Observations

(T1)

(T2)

$$1) F_n b = \underbrace{(F_n)_{\text{odd}}}_{n \times \frac{n}{2}} \underbrace{b_{\text{odd}}}_{\frac{n}{2} \times 1} + \underbrace{(F_n)_{\text{even}}}_{n \times \frac{n}{2}} \underbrace{b_{\text{even}}}_{\frac{n}{2} \times 1}$$

$$2) (F_n)_{\text{odd}} = \begin{pmatrix} F_{\frac{n}{2}} \\ F_{\frac{n}{2}} \end{pmatrix}$$

Can compute (T1) in:

$$T\left(\frac{n}{2}\right) + O(n) \text{ time.}$$

$$3) (F_n)_{\text{even}} = \begin{pmatrix} 1 & & & \\ & \omega_n & & \\ & & \omega_n^2 & \\ & & & \ddots \\ & & & & \omega_n^{n-1} \end{pmatrix} \begin{pmatrix} F_{\frac{n}{2}} \\ \\ \\ F_{\frac{n}{2}} \end{pmatrix}$$

"twiddle factors"

Can compute $\textcircled{T_2}$ in $T(\frac{n}{2}) + O(n)$ time.

Putting it together:

$$F_n b = \begin{pmatrix} F_{\frac{n}{2}} b_{\text{odd}} \\ F_{\frac{n}{2}} b_{\text{odd}} \end{pmatrix} \quad T(\frac{n}{2}) + O(n)$$

$$+ \begin{pmatrix} 1 & & & \\ & \omega_n & & \\ & & \omega_n^2 & \\ & & & \ddots \\ & & & & \omega_n^{n-1} \end{pmatrix} \begin{pmatrix} F_{\frac{n}{2}} b_{\text{even}} \\ \\ \\ F_{\frac{n}{2}} b_{\text{even}} \end{pmatrix} \quad T(\frac{n}{2}) + O(n)$$

FFT in $O(n \log n)$ time!

About multiplication... $O(n \log n)$ via FFT!

$$\begin{aligned} a &= a_{n-1} 10^{n-1} + a_{n-2} 10^{n-2} + \dots + a_1 10^1 + a_0 \\ &= p_a(10) \quad (\text{coeffs} = \text{digits of } a) \end{aligned}$$

$$\begin{aligned} b &= b_{n-1} 10^{n-1} + b_{n-2} 10^{n-2} + \dots + b_1 10^1 + b_0 \\ &= p_b(10) \quad (\text{coeffs} = \text{digits of } b) \end{aligned}$$

To compute ab , just need coeffs of $p_a p_b$.

Amazing fact: $F_n \overset{\text{coeffs}}{v} = \begin{pmatrix} p_v(1) \\ p_v(\omega_n) \\ p_v(\omega_n^2) \\ \vdots \\ p_v(\omega_n^{n-1}) \end{pmatrix} \overset{\text{evaluations}}{}$

FFT-based multiplication:

- 1) Evaluate $F_n a, F_n b$
- 2) Let $c = \{p_a(x) p_b(x)\}$ for $x = 1, \omega_n, \omega_n^2, \dots, \omega_n^{n-1}$
- 3) Evaluate $F_n^{-1} c$. Gives coeffs of $p_a p_b$!
- 4) Evaluate $(p_a p_b)(10) = ab$