CS 331, Fall 2024

Lecture 3 (9/4)

Today: - Inversions
- Matrices
- FFT

Inversions (Part II, Section 6.1)

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

Output: # of (i,j): $1 \leq i < j \leq n$

$L[i] > L[j]$ (inversion)

Example

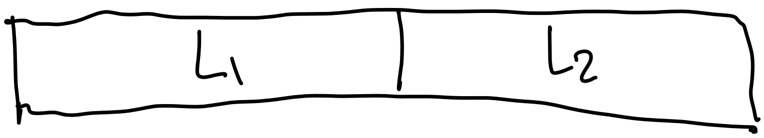
$L = [3, 5, 6, 2, 4, 1]$

inversions: $(i,j) = (1,4)\ (2,4)\ (3,4)$
$(1,6)\ (2,5)\ (3,5)$
$(4,6)\ (2,6)\ (3,6)$
$(5,6)$

Application: Metric between rankings "Kendall τ"
(e.g. comparing preferences)

Observation: n^2 time algo: compare all pairs.

Idea: recursion?

$L =$ 

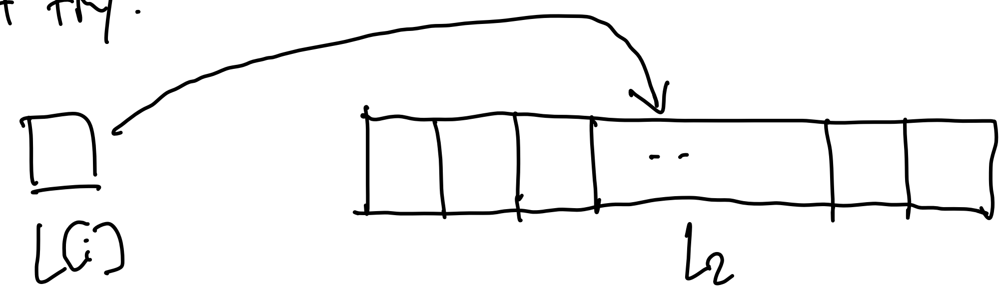
$$\text{Inversions}(L) = \text{Inversions}(L_1) \qquad T\left(\tfrac{n}{2}\right)$$
$$+ \text{Inversions}(L_2) \qquad T\left(\tfrac{n}{2}\right)$$

"stitching" step $\rightarrow$ $+ \left|\left\{ i \in \left[\tfrac{n}{2}\right], j \in [n] \setminus \left[\tfrac{n}{2}\right] : L[i] > L[j] \right\}\right|$

???

How to do stitching faster than n^2?

First try:



Time to compute $\left|\left\{ j \in [n] \setminus \left[\tfrac{n}{2}\right] \mid L[j] < L[i] \right\}\right|$

- $O(n)$ per i (naive)
- $O(\log(n))$ per i (binary search, if <u>L_2 sorted</u>)

Inversions recurrence:

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

recurse on halves

1) Sort L_2
2) binary search for each $L_1(i)$

Balanced case: $T(n) = O(n\log^2(n))$

Improvement: piggy back off MergeSort stitching

... // L_1, L_2 sorted

$i_1 \leftarrow 1$, $i_2 \leftarrow 1$, count $\leftarrow 0$ // pointers to smallest eles, # of inversions

For $i \in [n]$:

  If $L_1[i_1] \le L_2[i_2]$: $L[i] \leftarrow L_1[i_1]$, $i_1 \leftarrow i_1 + 1$,

    count $\leftarrow$ (count $+ (i_2 - 1)$) $O(1)$ time!

  Else: $L[i] \leftarrow L_2[i_2]$, $i_2 \leftarrow i_2 + 1$

Example

$$L_1 \; (1 \quad 6 \quad \overset{\boxed{i_1}}{7} \quad 18)(0 \quad 3 \quad \overset{\boxed{i_2}}{9} \quad 45) \; L_2$$

$$L \; (0 \quad 1 \quad 3 \quad 6 \quad \ldots$$

how many inserted?)

Improved runtime:

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

Takeaway 1: Can enforce stronger recursive guarantees

e.g. Sorted sublists + inversion counts.

Takeaway 2: For arrays, pointer / queue tricks save time

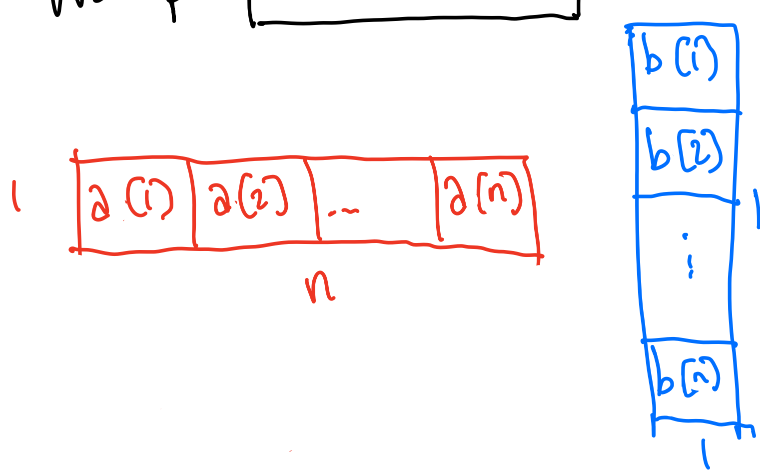
e.g. $O(\log(n)) \to O(1)$ per index.

e.g. "Sliding window" techniques: will revisit in Part III, Section 2

Matrix multiplication (Part II, Section 5.1)

Warmup: $\boxed{\text{Vector-vector}}$

"inner product"
aka
"dot product"



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

$O(n)$ time.

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

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

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

$\boxed{\text{Matrix-matrix}}$: dot every row/col pair

$$\begin{array}{cccc} & & \text{jth column} & \\ \text{ith row} & \begin{pmatrix} \vdots \\ \text{——} \; A_{i:}^T \; \text{——} \\ \vdots \end{pmatrix} & \begin{pmatrix} \Big| \\ \cdots \; B_{:j} \; \cdots \\ \Big| \end{pmatrix} & = \begin{pmatrix} \\ A_{i:}^T B_{:j} \\ \\ \end{pmatrix} \\ & A & B & AB \\ & (n \times d) & (d \times k) & (n \times k) \end{array}$$

Applications: entirety of modern ML/data science.

regression, feature learning, deep learning, ...

Intuition:

$$n \begin{pmatrix} \boxed{\text{cat}} \\ \boxed{\text{dog}} \\ \boxed{\text{bird}} \\ \vdots \end{pmatrix}_{d}$$

n = # examples

d = # features

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

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

$\boxed{\text{Aside}}$ 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

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

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

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

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

When can we do better?

$\boxed{\text{Matrix-Vector}}$ 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:

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

Linear time ☺

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

$$\overset{\text{output}}{\begin{pmatrix} \\ Ab \\ \\ \end{pmatrix}} = \underbrace{\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix}}_{\text{preprocess somehow...}} \overset{\text{input}}{\begin{pmatrix} \\ b \\ \\ \end{pmatrix}}$$

HW I, Problem 2: A is WHT, faster than n^2!

Today: A is FFT, $O(n \log(n))$ time.

Fast Fourier Transform (Part II, Section 5.2)

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

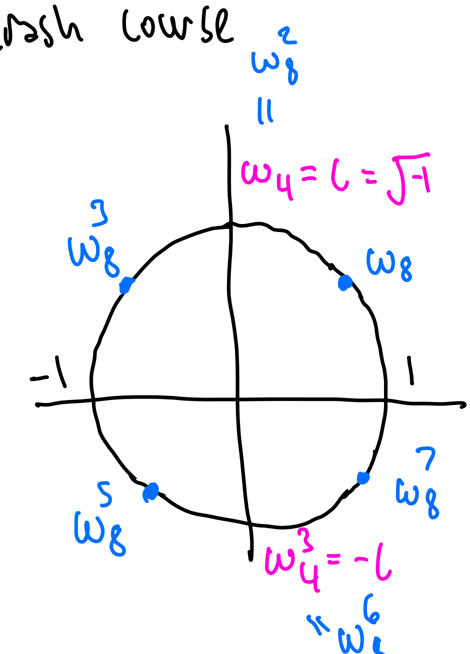
F_n is $n \times n$ complex matrix.

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

Applications: faster multiplication

signal processing (learn "spectrum" of waves)

Aside Complex numbers crash course



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

$$\omega_n = e^{i \cdot \frac{2\pi}{n}}$$ "nth root of unity"

$\frac{2\pi}{n}$ = rotate $\frac{1}{n}$th 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.

$$F_1 = \begin{pmatrix} 1 \end{pmatrix} \qquad \omega_1 = 1$$

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

$$F_4 = \begin{pmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^0 & \omega^2 \\ \omega^0 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix} \qquad \omega \equiv \omega_4 = i$$

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

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

Recall

$$\omega_8^2 = \omega_4$$

Claim: $\underbrace{T(n)}_{\text{time to compute } F_n b} = 2T\left(\frac{n}{2}\right) + O(n)$

$\boxed{\text{Aside}}$ Two ways of thinking about Ab

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

Way 2:

note: order doesn't matter!

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

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

$\boxed{\text{Observations}}$

1) $F_n b = \underbrace{\underbrace{(F_n)_{odd}}_{n \times \frac{n}{2}} \underbrace{b_{odd}}_{\frac{n}{2} \times 1}}_{T1} + \underbrace{\underbrace{(F_n)_{even}}_{n \times \frac{n}{2}} \underbrace{b_{even}}_{\frac{n}{2} \times 1}}_{T2}$

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

Can compute T1 in:

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

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

"twiddle factors"

Can compute T2 in $T\left(\frac{n}{2}\right) + O(n)$ time.

Putting it together:

$$F_n b = \begin{pmatrix} F_{\frac{n}{2}} b_{odd} \\ \\ F_{\frac{n}{2}} b_{odd} \end{pmatrix} \qquad T\left(\tfrac{n}{2}\right) + O(n)$$

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

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

About multiplication...

$$a = a_{n-1} 10^{n-1} + a_{n-2} 10^{n-2} + \dots a_1 10^1 + a_0$$

$$= p_a(10) \qquad (\text{coeffs} = \text{digits of } a)$$

$$b = b_{n-1} 10^{n-1} + b_{n-2} 10^{n-2} + \dots b_1 10^1 + b_0$$

$$= p_b(10) \qquad (\text{coeffs} = \text{digits of } b)$$

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

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

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 $p_a p_b$!