CSE 421

Divide and Conquer

Yin Tat Lee

Median

Selecting k-th smallest

Problem: Given numbers $x_1, \ldots, x_n$ and an integer $1 \leq k \leq n$
output the k-th smallest number

$$\mathrm{Sel}(\{x_1, \ldots, x_n\}, k)$$

Can we do $O(n)$ for all possible values of k?

An Idea

Note:
Finding w is like median problem

Choose a number w from $x_1, \dots, x_n$

Define

- $S_<(w) = \{x_i : x_i < w\}$
- $S_=(w) = \{x_i : x_i = w\}$
- $S_>(w) = \{x_i : x_i > w\}$

Can be computed in linear time

Solve the problem recursively as follows:

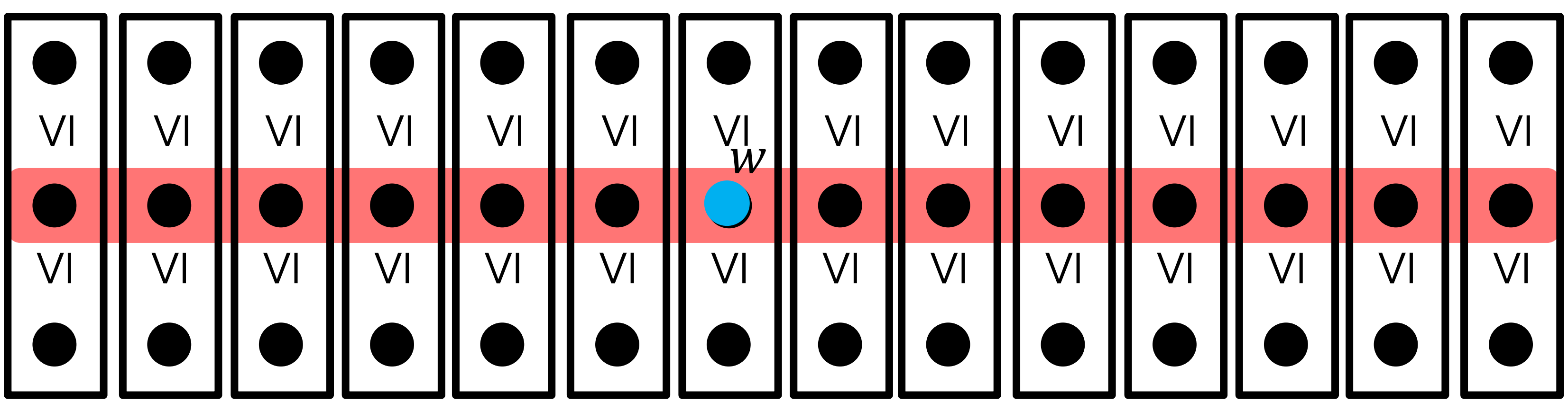
- If $k \leq |S_<(w)|$, output $Sel(S_<(w), k)$
- Else if $k \leq |S_<(w)| + |S_=(w)|$, output w
- Else output $Sel(S_>(w), k - |S_<(w)| - |S_=(w)|)$

Ideally want $|S_<(w)|, |S_>(w)| \leq n/2$. In this case ALG runs in $O(n) + O\left(\frac{n}{2}\right) + O\left(\frac{n}{4}\right) + \cdots + O(1) = O(n)$.

How to choose w?

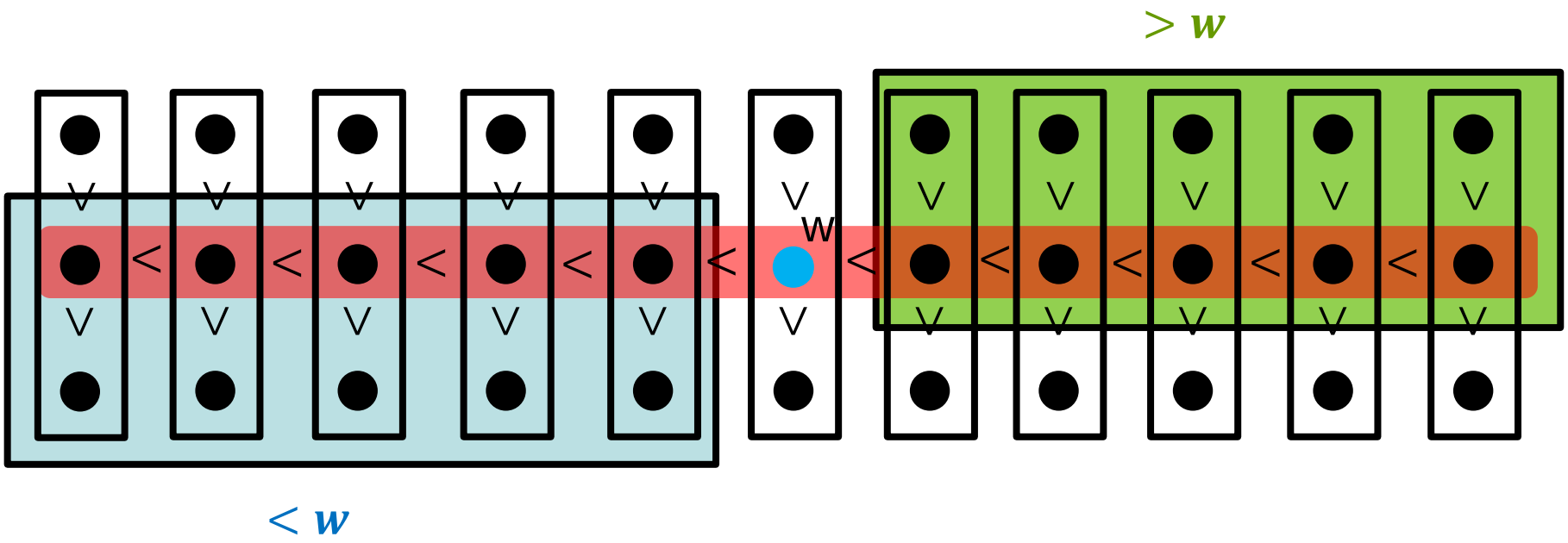
Almost correct approach:

- Partition numbers into sets of size 3.
- Sort each set (takes O(n))
- $w = Sel(\textcolor{red}{midpoints}, n/6)$



Assume all numbers are distinct for simplicity.

How to lower bound $|S_<(w)|, |S_>(w)|$?



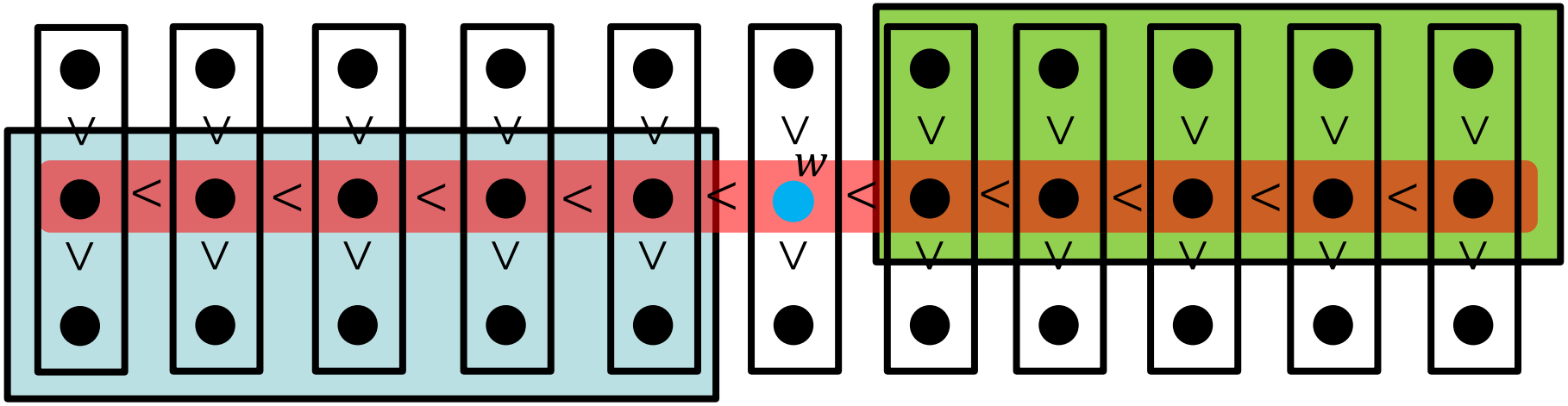
- $|S_<(w)| \geq 2\left(\frac{n}{6}\right) = \frac{n}{3}$
- $|S_>(w)| \geq 2\left(\frac{n}{6}\right) = \frac{n}{3}$.

$$\frac{n}{3} \leq |S_<(w)|, |S_>(w)| \leq \frac{2n}{3}$$

So, what is the running time?

Assume all numbers are distinct for simplicity.

Asymptotic Running Time?



- If $k \leq |S_<(w)|$, output $Sel(S_<(w), k)$
- Else if $k \leq |S_<(w)| + |S_=(w)|$, output w
- Else output $Sel(S_>(w), k - |S_<(w)| - |S_=(w)|)$

$O(n \log n)$ again?
So, what is the point?

Where $\frac{n}{3} \leq |S_<(w)|, |S_>(w)| \leq \frac{2n}{3}$

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + O(n) \Rightarrow T(n) = O(n \log n)$$

Recurrences

- $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n \Rightarrow T(n) = O(n \log n)$

The first layer cost is n.

The second layer cost is $\frac{n}{3} + \frac{2n}{3}$ which is same as the first layer.

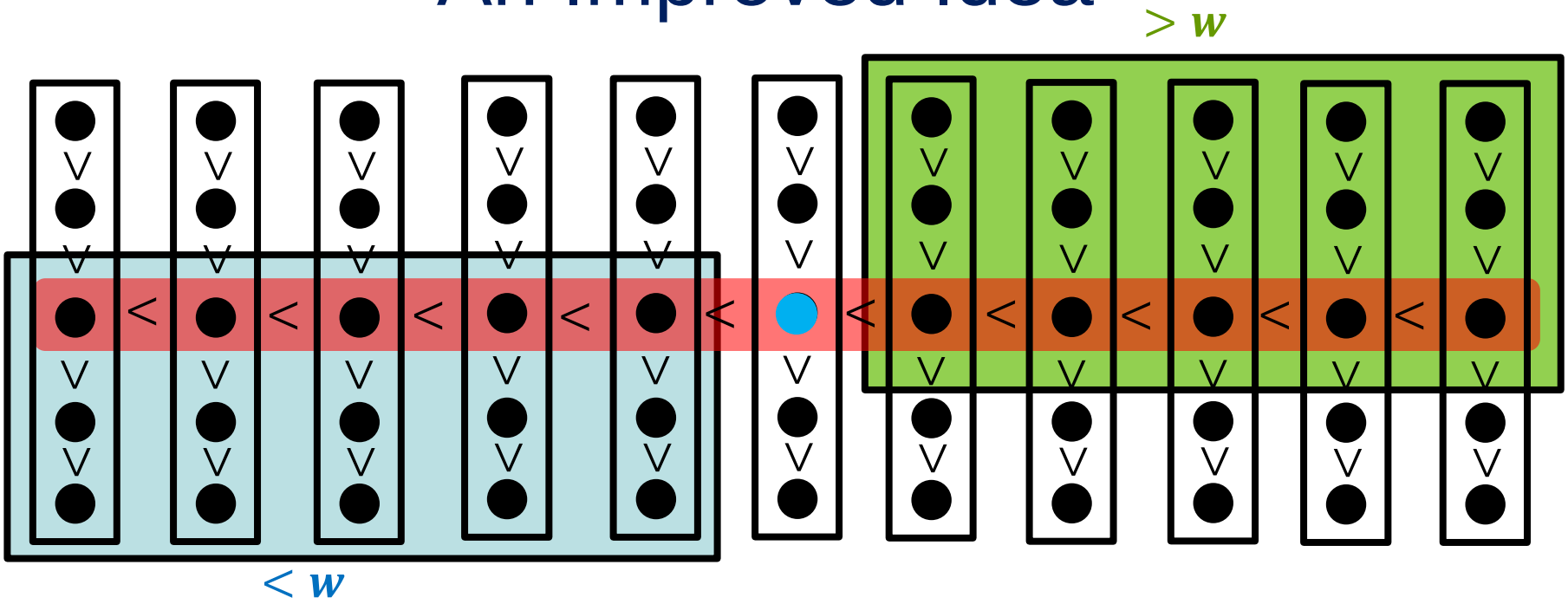
Similarly, every layer is roughly the same. Hence, it is $n \log n$.

Question 1: Where does the $n/3$ term comes from?

Question 2: How can we make it smaller?

Question 3: How to get $O(n)$ time?

An Improved Idea



Partition into n/5 sets. Sort each set and set $w = Sel(midpoints, n/10)$

- $|S_<(w)| \geq 3\left(\frac{n}{10}\right) = \frac{3n}{10}$
- $|S_>(w)| \geq 3\left(\frac{n}{10}\right) = \frac{3n}{10}$

$$\frac{3n}{10} \leq |S_<(w)|, |S_>(w)| \leq \frac{7n}{10}$$

$$T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + O(n) \Rightarrow T(n) = O(n)$$

Recurrences

- $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n \Rightarrow T(n) = O(n \log n)$

The first layer cost is n.

The second layer cost is $\frac{n}{3} + \frac{2n}{3}$ which is same as the first layer.

Similarly, every layer is roughly the same. Hence, it is $n \log n$.

- $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + n \Rightarrow T(n) = O(n)$

The first layer cost is n.

The second layer cost is $\frac{n}{5} + \frac{7n}{10}$ which is smaller than the first layer by a constant factor.

Hence, the total cost is a geometric sum and it is $O(n)$.

Integer Multiplication

Integer Arithmetic

Add: Given two n-bit integers a and b, compute $a + b$.

Add

	1	1	1	1	1	1	0	1	
		1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

$O(n)$ bit operations.

Multiply: Given two n-bit integers a and b, compute $a \times b$. The "grade school" method:

Multiply

						1	1	0	1	0	1	0	1		
					*	0	1	1	1	1	1	0	1		
						1	1	0	1	0	1	0	1		
					0	0	0	0	0	0	0	0			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	1	1	0	1	0	1	0	1							
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

$O(n^2)$ bit operations.

Divide and Conquer

Let x, y be two n-bit integers

Write $x = 2^{n/2}x_1 + x_0$ and $y = 2^{n/2}y_1 + y_0$

where x_0, x_1, y_0, y_1 are all $n/2$-bit integers.

Quiz:
What is the recursion for $T(n)$?

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0$$

Therefore,

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

We only need 3 values $x_1y_1, x_0y_0, x_1y_0 + x_0y_1$
Can we find all 3 by only 3 multiplication?

So,

$$T(n) = \Theta(n^2).$$

Key Trick: 4 multiplies at the price of 3

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0$$

$$\alpha = x_1 + x_0$$
$$\beta = y_1 + y_0$$
$$\alpha\beta = (x_1 + x_0)(y_1 + y_0)$$
$$= x_1 y_1 + (x_1 y_0 + x_0 y_1) + x_0 y_0$$
$$(x_1 y_0 + x_0 y_1) = \alpha\beta - x_1 y_1 - x_0 y_0$$

Key Trick: 4 multiplies at the price of 3

Theorem [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in $O(n^{1.585\ldots})$ bit operations.

$$x = 2^{n/2} \cdot x_1 + x_0 \Rightarrow \alpha = x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0 \Rightarrow \beta = y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= \underbrace{2^n \cdot x_1 y_1}_{A} + \underbrace{2^{n/2} \cdot (\alpha\beta - A - B)}_{\alpha\beta - A - B} + \underbrace{x_0 y_0}_{B}$$

To multiply two n-bit integers:

- Add two $n/2$ bit integers.
- Multiply three $n/2$-bit integers.
- Add, subtract, and shift $n/2$-bit integers to obtain result.

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) \Rightarrow T(n) = O\left(n^{\log_2 3}\right) = O(n^{1.585\ldots})$$

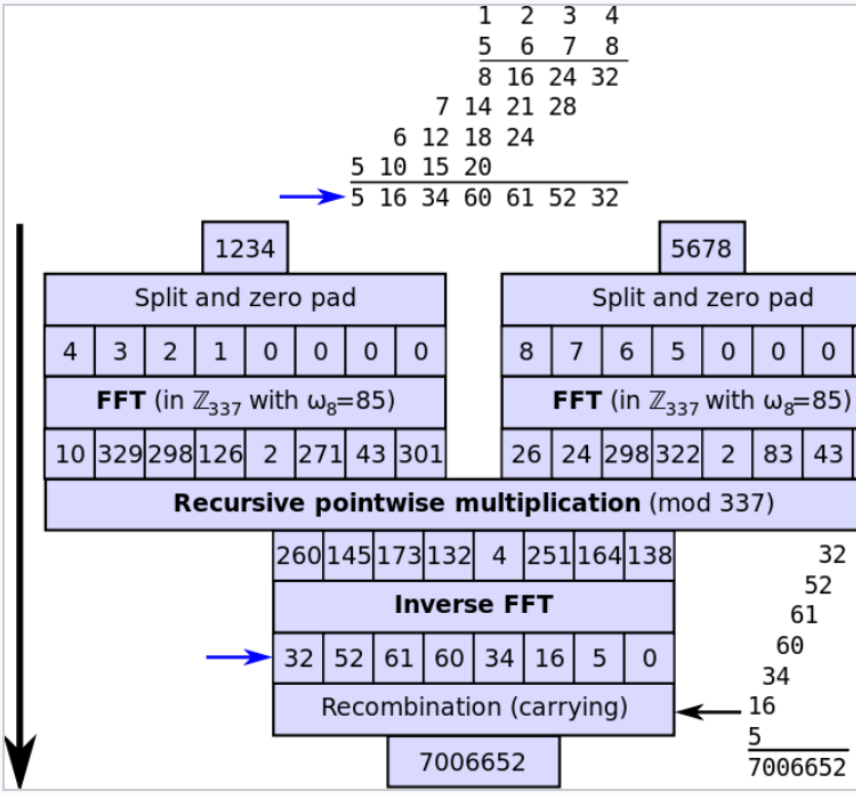
Integer Multiplication (Summary)

- Exercise: generalize Karatsuba to do 5 size $n/3$ subproblems

  This gives $\Theta(n^{1.46\ldots})$ time algorithm

Date	Authors	Time complexity
<3000 BC	Unknown	$O(n^2)$
1962	Karatsuba	$O(n^{\log 3/\log 2})$
1963	Toom	$O(n\,2^{5\sqrt{\log n/\log 2}})$
1966	Schönhage	$O(n\,2^{\sqrt{2\log n/\log 2}}(\log n)^{3/2})$
1969	Knuth	$O(n\,2^{\sqrt{2\log n/\log 2}}\log n)$
1971	Schönhage–Strassen	$O(n\log n\log\log n)$
2007	Fürer	$O\left(n\log n\,2^{O(\log^* n)}\right)$
2014	Harvey-Hoeven-Lecerf	$O(n\log n\,8^{\log^* n})$
2019	Harvey-Hoeven	$O(n\log n)$

Integer Multiplication (Summary)



Demonstration of multiplying 1234 × 5678 = 7006652 using fast Fourier transforms (FFTs). Number-theoretic transforms in the integers modulo 337 are used, selecting 85 as an 8th root of unity. Base 10 is used in place of base 2^w for illustrative purposes.

Matrix Matrix Multiplication

Multiplying Matrices

Let A be an $n \times m$ matrix, B be an $m \times p$ matrix.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}$$

Then, $C = AB$ is an $n \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj},$$

Question: Why matrix multiplication is defined in such way?

Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42} & \circ & a_{11}b_{14}+a_{12}b_{24}+a_{13}b_{34}+a_{14}b_{44} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42} & \circ & a_{21}b_{14}+a_{22}b_{24}+a_{23}b_{34}+a_{24}b_{44} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}+a_{34}b_{41} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}+a_{34}b_{42} & \circ & a_{31}b_{14}+a_{32}b_{24}+a_{33}b_{34}+a_{34}b_{44} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}+a_{44}b_{41} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}+a_{44}b_{42} & \circ & a_{41}b_{14}+a_{42}b_{24}+a_{43}b_{34}+a_{44}b_{44} \end{bmatrix}$$

Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42} & \circ & a_{11}b_{14}+a_{12}b_{24}+a_{13}b_{34}+a_{14}b_{44} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42} & \circ & a_{21}b_{14}+a_{22}b_{24}+a_{23}b_{34}+a_{24}b_{44} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}+a_{34}b_{41} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}+a_{34}b_{42} & \circ & a_{31}b_{14}+a_{32}b_{24}+a_{33}b_{34}+a_{34}b_{44} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}+a_{44}b_{41} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}+a_{44}b_{42} & \circ & a_{41}b_{14}+a_{42}b_{24}+a_{43}b_{34}+a_{44}b_{44} \end{bmatrix}$$

Multiplying Matrices

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{array}{l} \\ A_{11} \\ \\ A_{21} \end{array} \begin{array}{l} \\ A_{12} \\ \\ A_{22} \end{array} \bullet \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} \begin{array}{l} \\ B_{11} \\ \\ B_{21} \end{array} \begin{array}{l} \\ B_{12} \\ \\ B_{22} \end{array}$$

$$= \begin{bmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31}+a_{14}b_{41} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32}+a_{14}b_{42} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31}+a_{24}b_{41} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32}+a_{24}b_{42} \\ a_{31}b_{11}+a_{32}b_{21}+a_{33}b_{31}+a_{34}b_{41} & a_{31}b_{12}+a_{32}b_{22}+a_{33}b_{32}+a_{34}b_{42} \\ a_{41}b_{11}+a_{42}b_{21}+a_{43}b_{31}+a_{44}b_{41} & a_{41}b_{12}+a_{42}b_{22}+a_{43}b_{32}+a_{44}b_{42} \end{bmatrix} \begin{array}{l} \\ A_{11}B_{11}+A_{12}B_{21} \\ \\ A_{21}B_{11}+A_{22}B_{21} \end{array} \circ \begin{array}{l} \\ A_{11}B_{12}+A_{12}B_{22} \\ \\ A_{21}B_{12}+A_{22}B_{22} \end{array} \begin{bmatrix} a_{11}b_{14}+a_{12}b_{24}+a_{13}b_{34}+a_{14}b_{44} \\ a_{21}b_{14}+a_{22}b_{24}+a_{23}b_{34}+a_{24}b_{44} \\ a_{31}b_{14}+a_{32}b_{24}+a_{33}b_{34}+a_{34}b_{44} \\ a_{41}b_{14}+a_{42}b_{24}+a_{43}b_{34}+a_{44}b_{44} \end{bmatrix}$$

Simple Divide and Conquer

Quiz:
What is the recursion for $T(n)$?

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

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

- $T(n) = 8T(n/2) + 4\left(\frac{n}{2}\right)^2 = 8T(n/2) + n^2$

  So, $T(n) = \Theta\left(n^{\log_2 8}\right) = \Theta(n^3)$

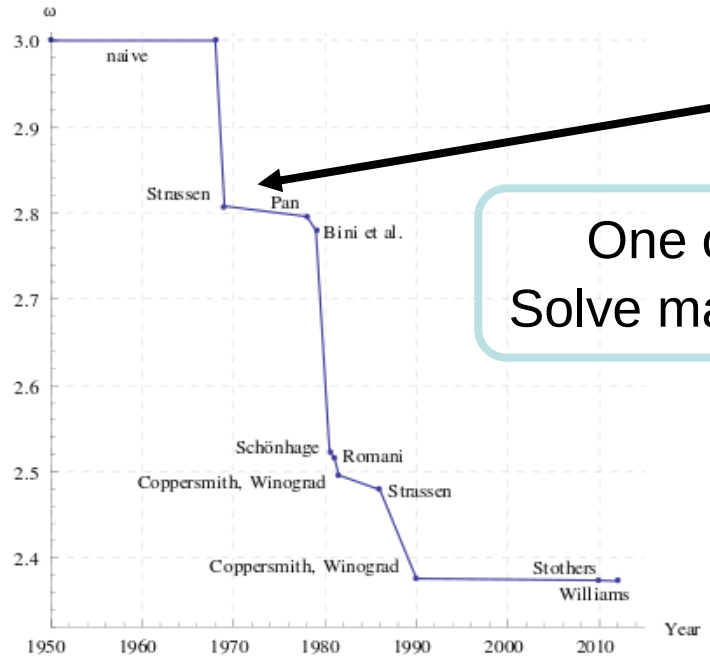
Strassen's Divide and Conquer Algorithm

- Strassen's algorithm

  Multiply $\mathbf{2 \times 2}$ matrices using $\mathbf{7}$ instead of $\mathbf{8}$ multiplications (and 18 additions)

$$T(n) = 7T\left(\frac{n}{2}\right) + 18n^2$$

  Hence, we have $T(n) = O\left(n^{\log_2 7}\right)$.



Useful when $n \sim 500$.

One of the most important open problem:
Solve matrix multiplication in $O(n^2 \log^{O(1)} n)$ time

Strassen's Divide and Conquer Algorithm

Naive

$$\mathbf{C}_{1,1} = \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{1,2} = \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2}$$
$$\mathbf{C}_{2,1} = \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1}$$
$$\mathbf{C}_{2,2} = \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2}$$

Strassen

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$
$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$
$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$
$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

Matrix Vector Multiplication

Matrix Vector Multiplication

Let A be an $n \times n$ matrix and a $n \times 1$ vector x.

How fast we can compute Ax?

Textbook explains it in a totally different way.

For general dense matrices, $\Theta(n^2)$.

For some special A, nearly linear time is possible!

Two important examples are

Fast Fourier transform: $A_{jk} = \exp(-\frac{2\pi ijk}{n})$

Fast multipole method: $A_{ij} = ||v_i - v_j||_2^{-1}$ for some vectors v_i

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

1962: Tony Hoare of Elliott Brothers, Ltd., London, presents **Quicksort**.

Putting N things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing so quickly. Hoare's algorithm uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a "pivot," separate the rest into piles of "big" and "small" elements (as compared with the pivot), and then repeat this procedure on each pile. Although it's possible to get stuck doing all $N(N-1)/2$ comparisons (especially if you use as your pivot the first item on a list that's already sorted!), Quicksort runs on average with $O(N \log N)$ efficiency. Its elegant simplicity has made Quicksort the pos-terchild of computational complexity.

James Cooley

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the **fast Fourier transform**.

Easily the most far-reaching algo-rithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ chore to an $O(N \log N)$ frolic. But unlike Quick- sort, the implementation is (at first sight) nonintuitive and less than straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

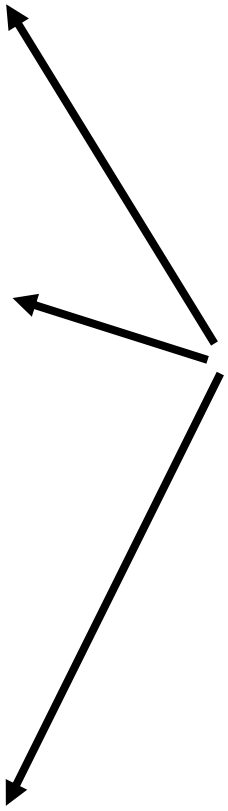
John Tukey

1977: Helaman Ferguson and Rodney Forcade of Brigham Young University advance an **integer relation detection algorithm**.

The problem is an old one: Given a bunch of real numbers, say $x_1, x_2, \ldots, x_n$, are there integers $a_1, a_2, \ldots, a_n$ (not all 0) for which $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$? For $n = 2$, the venerable Euclidean algorithm does the job, computing terms in the continued-fraction expansion of x_1/x_2. If x_1/x_2 is rational, the expansion terminates and, with proper unraveling, gives the "smallest" integers a_1 and a_2. If the Euclidean algorithm doesn't terminate—or if you simply get tired of computing it—then the unraveling procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade's generalization, although much more difficult to implement (and to understand), is also more powerful. Their detection algorithm, for example, has been used to find the precise coefficients of the polynomials satisfied by the third and fourth bifurcation points, $B_3 = 3.544090$ and $B_4 = 3.564407$, of the logistic map. (The latter polynomial is of degree 120; its largest coefficient is 257^{30}.) It has also proved useful in simplifying calculations with Feynman diagrams in quantum field theory.

1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the **fast multipole algorithm**.

This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require $O(N^2)$ computations—one for each pair of particles. The fast multipole algorithm gets by with $O(N)$ computations. It does so by using multipole expansions (net charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.



Divide and Conquer

Matrix Vector Multiplication (FFT)

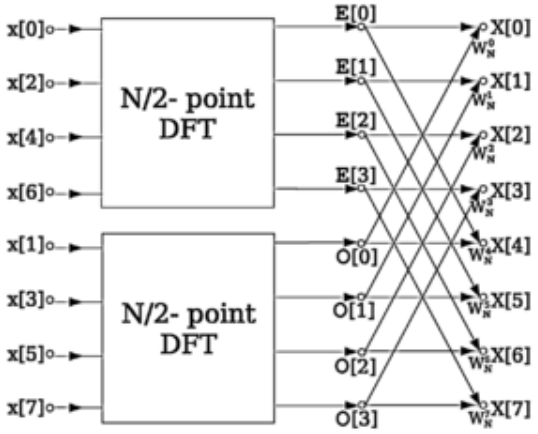
Let F_n be the $n \times n$ DFT matrix.

$$T(n) = 2T(n/2) + O(n)$$

Then, we have

$$F_{2n} = P_1 \cdot \begin{bmatrix} F_n & 0 \\ 0 & F_n \end{bmatrix} \cdot P_2 \cdot D \cdot \begin{bmatrix} F_2 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & F_2 \end{bmatrix} \cdot P_3$$

for some permutation P and some diagonal D.



See https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm for details.

Matrix Vector Multiplication (FMM)

Let M_n be the $n \times n$ green matrix ($||v_i - v_j||^{-1}$).

Then, we have

$$M_{2n} \approx \begin{bmatrix} M_{O(1)} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & M_{O(1)} \end{bmatrix} + P^{\top} M_n P.$$

for some permutation P.

See https://math.nyu.edu/~greengar/shortcourse_fmm.pdf for the real stuff.

$$T(n) = T(n/2) + O(n)$$