Lecture 1: Introduction

Michael Dinitz

August 27, 2024
601.433/633 Introduction to Algorithms

Welcome!

Introduction to (the theory of) algorithms

- How to design algorithms
- How to analyze algorithms

Prerequisites: Data Structures and MFCS/Discrete Math

- Small amount of review next lecture, but should be comfortable with asymptotic notation, basic data structures, basic combinatorics and graph theory.
- Undergrads from prereqs.
- "Informal" prerequisite: *mathematical maturity*

About me

- 9th time teaching this class (Fall 2014 - Fall 2021).
  - I'm still learning – let me know if you have suggestions!
  - Fall 2022: Sabbatical at Google Research-New York
  - Fall 2023: Parental leave

- Research in theoretical CS, particularly algorithms: approximation algorithms, graph algorithms, distributed algorithms, online algorithms.
- Also other parts of math (graph theory) and CS theory (algorithmic game theory, complexity theory) and theory of networking.

- Office hours: Mondays 2pm–3pm (zoom), Wednesdays 2pm–3pm (Malone 217).

Administrative Stuff

- TA: Shruthi Prusty (CS PhD student). Office hours TBD
- Head CA: Tian Zhou (senior undergraduate). Office hours TBD
- CAs: Many, still finalizing.
- Website:
  `http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2024/`
  - Syllabus, schedule, lecture notes, office hours, ...
  - Courselore for discussion/announcements
  - Gradescope for homeworks/exams.
- Textbook: CLRS (third or fourth edition)

Administrative Stuff

- TA: Shruthi Prusty (CS PhD student). Office hours TBD
- Head CA: Tian Zhou (senior undergraduate). Office hours TBD
- CAs: Many, still finalizing.
- Website: `http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2024/`
  - Syllabus, schedule, lecture notes, office hours, ...
  - Courselore for discussion/announcements
  - Gradescope for homeworks/exams.
- Textbook: CLRS (third or fourth edition)
- Class not the same as has been taught the last few years by Gagan Garg!
  - I tend to go faster, cover more material.
  - A little less hand-holding, a little more traditional
  - Grade distribution should be approximately the same.

Assignments

Homeworks:

- Approximately every 1.5 weeks, posted on website (HW1 out, due next Tuesday!)
- *Must* be typeset (LaTeX preferred, not required)
- Work in groups of ≤ 3 (highly recommended). But *individual* writeups.
  - Work together at a whiteboard to solve, then write up yourself.
  - Write group members at top of homework
- 120 late hours (5 late days) total

Assignments

Homeworks:

- Approximately every 1.5 weeks, posted on website (HW1 out, due next Tuesday!)
- *Must* be typeset (LaTeX preferred, not required)
- Work in groups of ≤ 3 (highly recommended). But *individual* writeups.
  - Work together at a whiteboard to solve, then write up yourself.
  - Write group members at top of homework
- 120 late hours (5 late days) total

Exams: Midterm, final.

- Midterm: In-class (75 minutes), traditional, closed book
- Final: in person, scheduled by registrar. 3 hours, traditional, closed book.

Assignments

Homeworks:

- Approximately every 1.5 weeks, posted on website (HW1 out, due next Tuesday!)
- *Must* be typeset (LaTeX preferred, not required)
- Work in groups of ≤ 3 (highly recommended). But *individual* writeups.
  - Work together at a whiteboard to solve, then write up yourself.
  - Write group members at top of homework
- 120 late hours (5 late days) total

Exams: Midterm, final.

- Midterm: In-class (75 minutes), traditional, closed book
- Final: in person, scheduled by registrar. 3 hours, traditional, closed book.

Grading: 50% homework, 15% midterm, 35% final exam,

- "Curve": Historically, average $\approx$ B+. About 50% A's, 50% B's, a few below.
  - Curve only helps! Someone else doing well does not hurt you.
  - Be collaborative and helpful (within guidelines).

Academic Honesty

- Cheating makes you a bad person. Don't cheat.

Academic Honesty

- Cheating makes you a bad person. Don't cheat.
- Cheating includes:
  - Collaborating with people outside your group of three.
  - Collaborating *with* your group on the writeup.
  - Looking online for the solutions/ideas to the problem *or related problems*, rather than to understand concepts from class.
  - Using ChatGPT or other LLMs.
  - Using Chegg, CourseHero, your friends, . . . , to find back tests, old homeworks, etc.
  - Uploading anything to the above sites.
  - etc.

Academic Honesty

- Cheating makes you a bad person. Don't cheat.
- Cheating includes:
  - Collaborating with people outside your group of three.
  - Collaborating *with* your group on the writeup.
  - Looking online for the solutions/ideas to the problem *or related problems*, rather than to understand concepts from class.
  - Using ChatGPT or other LLMs.
  - Using Chegg, CourseHero, your friends, . . . , to find back tests, old homeworks, etc.
  - Uploading anything to the above sites.
  - etc.
- Just solve the problems with your group and write them up yourself!
  - Use the internet, classmates, other resources to understand concepts from class, not to help with assignments.

Academic Honesty

- Cheating makes you a bad person. Don't cheat.
- Cheating includes:
  - Collaborating with people outside your group of three.
  - Collaborating *with* your group on the writeup.
  - Looking online for the solutions/ideas to the problem *or related problems*, rather than to understand concepts from class.
  - Using ChatGPT or other LLMs.
  - Using Chegg, CourseHero, your friends, . . . , to find back tests, old homeworks, etc.
  - Uploading anything to the above sites.
  - etc.
- Just solve the problems with your group and write them up yourself!
  - Use the internet, classmates, other resources to understand concepts from class, not to help with assignments.
- In previous years, punishments have included zero on assignment, grade penalty, mark on transcript, etc. ≥ 1 person has had PhD acceptance revoked.

Course Overview

- Introduction to *Theory* of Algorithms: math not programming.
- Two goals: how to *design* algorithms, and how to *analyze* algorithms.
  - Sometimes focus more on one than other, but both important

Course Overview

- Introduction to *Theory* of Algorithms: math not programming.
- Two goals: how to *design* algorithms, and how to *analyze* algorithms.
  - Sometimes focus more on one than other, but both important
- Algorithm: "recipe" for solving a computational problem.
  - Computational problem: given input $\boldsymbol{X}$, want to output $\boldsymbol{f(X)}$. How to do this?

Course Overview

- Introduction to *Theory* of Algorithms: math not programming.
- Two goals: how to *design* algorithms, and how to *analyze* algorithms.
  - Sometimes focus more on one than other, but both important
- Algorithm: "recipe" for solving a computational problem.
  - Computational problem: given input $\boldsymbol{X}$, want to output $\boldsymbol{f(X)}$. How to do this?
- Things to prove about an algorithm:
  - Correctness: it does solve the problem.
  - Running time: worst-case, average-case, worst-case expected, amortized, ...
  - Space usage
  - and more!

Course Overview

- Introduction to *Theory* of Algorithms: math not programming.
- Two goals: how to *design* algorithms, and how to *analyze* algorithms.
  - Sometimes focus more on one than other, but both important
- Algorithm: "recipe" for solving a computational problem.
  - Computational problem: given input $\boldsymbol{X}$, want to output $\boldsymbol{f(X)}$. How to do this?
- Things to prove about an algorithm:
  - Correctness: it does solve the problem.
  - Running time: worst-case, average-case, worst-case expected, amortized, ...
  - Space usage
  - and more!
- This class: mostly correctness and asymptotic running time, focus on worst-case

Why analyze? Why worst case?

- Obviously want to prove correctness!
  - Testing good, but want to be 100% sure that the algorithm does what you want it to do!

Why analyze? Why worst case?

- Obviously want to prove correctness!
  - Testing good, but want to be 100% sure that the algorithm does what you want it to do!
- What is a "real-life" or "average" instance?
  - Especially if your algorithm is "low-level", will be used in many different settings.

Why analyze? Why worst case?

- Obviously want to prove correctness!
  - Testing good, but want to be 100% sure that the algorithm does what you want it to do!
- What is a “real-life” or “average” instance?
  - Especially if your algorithm is “low-level”, will be used in many different settings.
- We will focus on how algorithm “scales”: how running times change as input grows. Hard to determine experimentally.

Why analyze? Why worst case?

- Obviously want to prove correctness!
  - Testing good, but want to be 100% sure that the algorithm does what you want it to do!
- What is a “real-life” or “average” instance?
  - Especially if your algorithm is “low-level”, will be used in many different settings.
- We will focus on how algorithm “scales”: how running times change as input grows. Hard to determine experimentally.
- Most importantly: want to *understand*.
  - Experiments can (maybe) convince you that something is true. But can’t tell you why!

Example 1: Multiplication

Multiplication I

Often an obvious way to solve a problem just from the definition. But might not be the right way!

Multiplication I

Often an obvious way to solve a problem just from the definition. But might not be the right way!

Multiplication: Given two n-bit integers X and Y. Compute XY.

- Since n bits, each integer in $[0, 2^n - 1]$.

How to do this?

Multiplication I

Often an obvious way to solve a problem just from the definition. But might not be the right way!

Multiplication: Given two $\boldsymbol{n}$-bit integers $\boldsymbol{X}$ and $\boldsymbol{Y}$. Compute $\boldsymbol{XY}$.

- Since $\boldsymbol{n}$ bits, each integer in $[\mathbf{0}, \mathbf{2}^{\boldsymbol{n}} - \mathbf{1}]$.

How to do this?

Definition of multiplication:

- Add $\boldsymbol{X}$ to itself $\boldsymbol{Y}$ times: $\boldsymbol{X} + \boldsymbol{X} + \cdots + \boldsymbol{X}$. Or add $\boldsymbol{Y}$ to itself $\boldsymbol{X}$ times: $\boldsymbol{Y} + \boldsymbol{Y} + \cdots + \boldsymbol{Y}$.

Multiplication I

Often an obvious way to solve a problem just from the definition. But might not be the right way!

Multiplication: Given two n-bit integers X and Y. Compute XY.

- Since n bits, each integer in $[0, 2^n - 1]$.

How to do this?

Definition of multiplication:

- Add X to itself Y times: $X + X + \cdots + X$. Or add Y to itself X times: $Y + Y + \cdots + Y$.

Running time:

Multiplication I

Often an obvious way to solve a problem just from the definition. But might not be the right way!

Multiplication: Given two n-bit integers X and Y. Compute XY.

- Since n bits, each integer in $[0, 2^n - 1]$.

How to do this?

Definition of multiplication:

- Add X to itself Y times: $X + X + \cdots + X$. Or add Y to itself X times: $Y + Y + \cdots + Y$.

Running time:

- $\Theta(Y)$ or $\Theta(X)$ (assuming constant-time adds).

Multiplication I

Often an obvious way to solve a problem just from the definition. But might not be the right way!

Multiplication: Given two n-bit integers X and Y. Compute XY.

- Since n bits, each integer in $[0, 2^n - 1]$.

How to do this?

Definition of multiplication:

- Add X to itself Y times: $X + X + \cdots + X$. Or add Y to itself X times: $Y + Y + \cdots + Y$.

Running time:

- $\Theta(Y)$ or $\Theta(X)$ (assuming constant-time adds).
- Could be $\Theta(2^n)$. *Exponential* in size of input ($2n$).

Multiplication II

Better idea?

Multiplication II

Better idea? Grade school algorithm!

Multiplication II

Better idea? Grade school algorithm!

```
          110110   = 54
x         101001   = 41
----------------
          110110
       110110
+    110110
----------------
    100010100110   = 2 + 4 + 32 + 128 + 2048 = 2214
```

Multiplication II

Better idea? Grade school algorithm!

```
          110110   = 54
x         101001   = 41
----------------
          110110
       110110
+    110110
----------------
    100010100110   = 2 + 4 + 32 + 128 + 2048 = 2214
```

Running time:

Multiplication II

Better idea? Grade school algorithm!

```
          110110   = 54
x         101001   = 41
----------------
          110110
       110110
+    110110
----------------
    100010100110   = 2 + 4 + 32 + 128 + 2048 = 2214
```

Running time:

- $O(n)$ column additions, each takes $O(n)$ time $\implies O(n^2)$ time.
- Better than obvious algorithm!

Multiplication III

Can we do even better?

Multiplication III

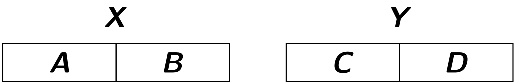
Can we do even better? Yes: *Karatsuba Multiplication*

Multiplication III

Can we do even better? Yes: *Karatsuba Multiplication*

$$X = 2^{n/2}A + B$$

$$Y = 2^{n/2}C + D$$

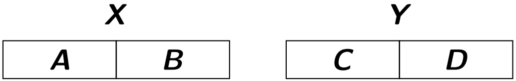


Multiplication III

Can we do even better? Yes: *Karatsuba Multiplication*

$$X = 2^{n/2}A + B$$

$$Y = 2^{n/2}C + D$$



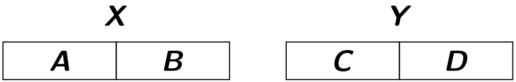
$$XY = (2^{n/2}A + B)(2^{n/2}C + D)$$

Multiplication III

Can we do even better? Yes: *Karatsuba Multiplication*

$$X = 2^{n/2}A + B$$

$$Y = 2^{n/2}C + D$$



$$XY = (2^{n/2}A + B)(2^{n/2}C + D)$$

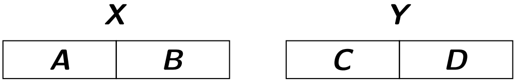
$$= 2^n AC + 2^{n/2}AD + 2^{n/2}BC + BD$$

Multiplication III

Can we do even better? Yes: *Karatsuba Multiplication*

$X = 2^{n/2}A + B$

$Y = 2^{n/2}C + D$



$$XY = (2^{n/2}A + B)(2^{n/2}C + D) = 2^{n}AC + 2^{n/2}AD + 2^{n/2}BC + BD$$

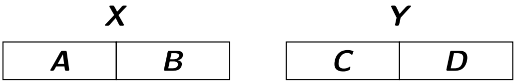
Four $n/2$-bit multiplications, three shifts, three $O(n)$-bit adds.

Multiplication III

Can we do even better? Yes: *Karatsuba Multiplication*

$$X = 2^{n/2}A + B$$

$$Y = 2^{n/2}C + D$$



$$\begin{aligned} XY &= (2^{n/2}A + B)(2^{n/2}C + D) \\ &= 2^{n}AC + 2^{n/2}AD + 2^{n/2}BC + BD \end{aligned}$$

Four $n/2$-bit multiplications, three shifts, three $O(n)$-bit adds.

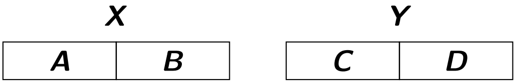
Running Time: $T(n) = 4T(n/2) + cn$

Multiplication III

Can we do even better? Yes: *Karatsuba Multiplication*

$X = 2^{n/2}A + B$

$Y = 2^{n/2}C + D$



$$
\begin{aligned}
XY &= (2^{n/2}A + B)(2^{n/2}C + D) \\
&= 2^{n}AC + 2^{n/2}AD + 2^{n/2}BC + BD
\end{aligned}
$$

Four $n/2$-bit multiplications, three shifts, three $O(n)$-bit adds.

Running Time: $T(n) = 4T(n/2) + cn \implies T(n) = O(n^2)$

Karatsuba Multiplication

Rewrite equation for XY:

$$\begin{aligned} XY &= 2^n AC + 2^{n/2} AD + 2^{n/2} BC + BD \\ &= 2^{n/2}(A+B)(C+D) + (2^n - 2^{n/2})AC + (1 - 2^{n/2})BD \end{aligned}$$

Karatsuba Multiplication

Rewrite equation for $\boldsymbol{XY}$:

$$\begin{aligned} \boldsymbol{XY} &= 2^{\boldsymbol{n}}\boldsymbol{AC} + 2^{\boldsymbol{n}/2}\boldsymbol{AD} + 2^{\boldsymbol{n}/2}\boldsymbol{BC} + \boldsymbol{BD} \\ &= 2^{\boldsymbol{n}/2}(\boldsymbol{A}+\boldsymbol{B})(\boldsymbol{C}+\boldsymbol{D}) + (2^{\boldsymbol{n}} - 2^{\boldsymbol{n}/2})\boldsymbol{AC} + (1 - 2^{\boldsymbol{n}/2})\boldsymbol{BD} \end{aligned}$$

Three $\boldsymbol{n/2}$-bit multiplications, $\boldsymbol{O(1)}$ shifts and $\boldsymbol{O(n)}$-bit adds.

Karatsuba Multiplication

Rewrite equation for XY:

$$\begin{aligned} XY &= 2^n AC + 2^{n/2} AD + 2^{n/2} BC + BD \\ &= 2^{n/2}(A+B)(C+D) + (2^n - 2^{n/2})AC + (1 - 2^{n/2})BD \end{aligned}$$

Three $n/2$-bit multiplications, $O(1)$ shifts and $O(n)$-bit adds.

$\implies T(n) = 3T(n/2) + c'n$

Karatsuba Multiplication

Rewrite equation for XY:

$$\begin{aligned} XY &= 2^n AC + 2^{n/2} AD + 2^{n/2} BC + BD \\ &= 2^{n/2}(A+B)(C+D) + (2^n - 2^{n/2})AC + (1 - 2^{n/2})BD \end{aligned}$$

Three $n/2$-bit multiplications, $O(1)$ shifts and $O(n)$-bit adds.

$\implies T(n) = 3T(n/2) + c'n$

$\implies T(n) = O(n^{\log_2 3}) \approx O(n^{1.585})$

Even Better Multiplication?

Can we do even better than Karatsuba?

Even Better Multiplication?

Can we do even better than Karatsuba?

Theorem (Karp)

There is an $O(n \log^2 n)$-time algorithm for multiplication.

Uses *Fast Fourier Transform* (FFT)

Even Better Multiplication?

Can we do even better than Karatsuba?

Theorem (Karp)

There is an $\boldsymbol{O(n \log^2 n)}$*-time algorithm for multiplication.*

Uses *Fast Fourier Transform* (FFT)

Theorem (Harvey and van der Hoeven '19)

There is an $\boldsymbol{O(n \log n)}$*-time algorithm for multiplication.*

Example 2: Matrix Multiplication

Matrix Multiplication: Definition

Given $X, Y \in \mathbb{R}^{n \times n}$, compute $XY \in \mathbb{R}^{n \times n}$

- $(XY)_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$
- Don't worry for now about representing real numbers
- Assume multiplication in $O(1)$ time

Matrix Multiplication: Definition

Given $X, Y \in \mathbb{R}^{n \times n}$, compute $XY \in \mathbb{R}^{n \times n}$

- $(XY)_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$
- Don't worry for now about representing real numbers
- Assume multiplication in $O(1)$ time

Algorithm from definition:

- For each $i, j \in \{1, 2, \ldots, n\}$, compute $(XY)_{ij}$ using formula.

Matrix Multiplication: Definition

Given $X, Y \in \mathbb{R}^{n \times n}$, compute $XY \in \mathbb{R}^{n \times n}$

- $(XY)_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$
- Don't worry for now about representing real numbers
- Assume multiplication in $O(1)$ time

Algorithm from definition:

- For each $i, j \in \{1, 2, \ldots, n\}$, compute $(XY)_{ij}$ using formula.

Running time:

Matrix Multiplication: Definition

Given $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{n \times n}$, compute $\boldsymbol{XY} \in \mathbb{R}^{n \times n}$

- $(\boldsymbol{XY})_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$
- Don't worry for now about representing real numbers
- Assume multiplication in $O(1)$ time

Algorithm from definition:

- For each $i, j \in \{1, 2, \ldots, n\}$, compute $(\boldsymbol{XY})_{ij}$ using formula.

Running time:

- $O(n^2)$ entries, each entry takes n multiplications and $n - 1$ additions $\implies O(n^3)$ time.

Strassen I

Break $\boldsymbol{X}$ and $\boldsymbol{Y}$ each into four $(\boldsymbol{n}/2) \times (\boldsymbol{n}/2)$ matrices:

$$\boldsymbol{X} = \begin{array}{|c|c|} \hline \boldsymbol{A} & \boldsymbol{B} \\ \hline \boldsymbol{C} & \boldsymbol{D} \\ \hline \end{array} \qquad \boldsymbol{Y} = \begin{array}{|c|c|} \hline \boldsymbol{E} & \boldsymbol{F} \\ \hline \boldsymbol{G} & \boldsymbol{H} \\ \hline \end{array}$$

Strassen I

Break $\boldsymbol{X}$ and $\boldsymbol{Y}$ each into four $(\boldsymbol{n}/2) \times (\boldsymbol{n}/2)$ matrices:

$$\boldsymbol{X} = \begin{array}{|c|c|} \hline \boldsymbol{A} & \boldsymbol{B} \\ \hline \boldsymbol{C} & \boldsymbol{D} \\ \hline \end{array} \qquad \boldsymbol{Y} = \begin{array}{|c|c|} \hline \boldsymbol{E} & \boldsymbol{F} \\ \hline \boldsymbol{G} & \boldsymbol{H} \\ \hline \end{array}$$

So can rewrite $\boldsymbol{XY}$:

$$\boldsymbol{XY} = \begin{array}{|c|c|} \hline \boldsymbol{AE} + \boldsymbol{BG} & \boldsymbol{AF} + \boldsymbol{BH} \\ \hline \boldsymbol{CE} + \boldsymbol{DG} & \boldsymbol{CF} + \boldsymbol{DH} \\ \hline \end{array}$$

Strassen I

Break $\boldsymbol{X}$ and $\boldsymbol{Y}$ each into four $(\boldsymbol{n}/2) \times (\boldsymbol{n}/2)$ matrices:

$$\boldsymbol{X} = \begin{array}{|c|c|} \hline \boldsymbol{A} & \boldsymbol{B} \\ \hline \boldsymbol{C} & \boldsymbol{D} \\ \hline \end{array} \qquad \boldsymbol{Y} = \begin{array}{|c|c|} \hline \boldsymbol{E} & \boldsymbol{F} \\ \hline \boldsymbol{G} & \boldsymbol{H} \\ \hline \end{array}$$

So can rewrite $\boldsymbol{XY}$:

$$\boldsymbol{XY} = \begin{array}{|c|c|} \hline \boldsymbol{AE} + \boldsymbol{BG} & \boldsymbol{AF} + \boldsymbol{BH} \\ \hline \boldsymbol{CE} + \boldsymbol{DG} & \boldsymbol{CF} + \boldsymbol{DH} \\ \hline \end{array}$$

Recursive algorithm: compute eight $(\boldsymbol{n}/2) \times (\boldsymbol{n}/2)$ matrix multiplies, four additions

Strassen II

$$XY = \begin{array}{|c|c|} \hline AE + BG & AF + BH \\ \hline CE + DG & CF + DH \\ \hline \end{array}$$

Recursive algorithm: compute eight $(n/2) \times (n/2)$ matrix multiplies, four additions

Strassen II

$$XY = \begin{array}{|c|c|} \hline AE + BG & AF + BH \\ \hline CE + DG & CF + DH \\ \hline \end{array}$$

Recursive algorithm: compute eight $(n/2) \times (n/2)$ matrix multiplies, four additions

Running time: $T(n) = 8T(n/2) + cn^2 \implies T(n) = O(n^3)$.

Strassen II

$$XY = \begin{array}{|c|c|} \hline AE + BG & AF + BH \\ \hline CE + DG & CF + DH \\ \hline \end{array}$$

Recursive algorithm: compute eight $(n/2) \times (n/2)$ matrix multiplies, four additions

Running time: $T(n) = 8T(n/2) + cn^2 \implies T(n) = O(n^3)$.

Improve on this?

Strassen III

$$XY = \begin{array}{|c|c|} \hline AE + BG & AF + BH \\ \hline CE + DG & CF + DH \\ \hline \end{array}$$

Strassen III

$$XY = \begin{array}{|c|c|} \hline AE + BG & AF + BH \\ \hline CE + DG & CF + DH \\ \hline \end{array}$$

$$M_1 = (A + D)(E + H) \qquad M_2 = (C + D)E \qquad M_3 = A(F - H)$$

$$M_4 = D(G - E) \qquad M_5 = (A + B)H \qquad M_6 = (C - A)(E + F)$$

$$M_7 = (B - D)(G + H)$$

Strassen III

$$XY = \begin{array}{|c|c|}\hline AE + BG & AF + BH \\ \hline CE + DG & CF + DH \\ \hline \end{array}$$

$$M_1 = (A + D)(E + H) \qquad M_2 = (C + D)E \qquad M_3 = A(F - H)$$

$$M_4 = D(G - E) \qquad M_5 = (A + B)H \qquad M_6 = (C - A)(E + F)$$

$$M_7 = (B - D)(G + H)$$

$$XY = \begin{array}{|c|c|}\hline M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ \hline M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \\ \hline \end{array}$$

Strassen IV

$$M_1 = (A + D)(E + H) \qquad M_2 = (C + D)E \qquad M_3 = A(F - H)$$
$$M_4 = D(G - E) \qquad M_5 = (A + B)H \qquad M_6 = (C - A)(E + F)$$
$$M_7 = (B - D)(G + H)$$

$$XY = \begin{array}{|c|c|} \hline M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ \hline M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \\ \hline \end{array}$$

Only *seven* $(n/2) \times (n/2)$ matrix multiplies, $O(1)$ additions

Strassen IV

$$M_1 = (A+D)(E+H) \qquad M_2 = (C+D)E \qquad M_3 = A(F-H)$$
$$M_4 = D(G-E) \qquad M_5 = (A+B)H \qquad M_6 = (C-A)(E+F)$$
$$M_7 = (B-D)(G+H)$$

$$XY = \begin{array}{|c|c|} \hline M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ \hline M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \\ \hline \end{array}$$

Only *seven* $(n/2) \times (n/2)$ matrix multiplies, $O(1)$ additions

Running time: $T(n) = 7T(n/2) + c'n^2 \implies T(n) = O(n^{\log_2 7}) \approx O(n^{2.8074})$.

Further Progress

- Coppersmith and Winograd '90: $\boldsymbol{O(n^{2.375477})}$
- Virginia Vassilevska Williams '13: $\boldsymbol{O(n^{2.3728642})}$
- François Le Gall '14: $\boldsymbol{O(n^{2.3728639})}$
- Josh Alman and Virginia Vassilevska Williams '21: $\boldsymbol{O(n^{2.3728596})}$
- Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou '24: $\boldsymbol{O(n^{2.371552})}$.

Further Progress

- Coppersmith and Winograd '90: $O(n^{2.375477})$
- Virginia Vassilevska Williams '13: $O(n^{2.3728642})$
- François Le Gall '14: $O(n^{2.3728639})$
- Josh Alman and Virginia Vassilevska Williams '21: $O(n^{2.3728596})$
- Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou '24: $O(n^{2.371552})$.

Is there an algorithm for matrix multiplication in $O(n^2)$ time?

Further Progress

- Coppersmith and Winograd '90: $O(n^{2.375477})$
- Virginia Vassilevska Williams '13: $O(n^{2.3728642})$
- François Le Gall '14: $O(n^{2.3728639})$
- Josh Alman and Virginia Vassilevska Williams '21: $O(n^{2.3728596})$
- Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou '24: $O(n^{2.371552})$.

Is there an algorithm for matrix multiplication in $O(n^2)$ time?

If you answer this (with proof!), automatic A+ in course and PhD

See you Thursday!