Lecture 1: Introduction

Michael Dinitz

August 26, 2025
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*

Instructors: Michael Dinitz, Jessica Sorrell

- First time with two instructors!

About me

- 10th time teaching this class (Fall 2014 - Fall 2025).
  - 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: TBD.

Administrative Stuff

- TA: Nate Robinson and Yan Zhong (CS PhD students). Office hours TBD
- CAs: Many, still finalizing.
- Website: `https://introalgorithmsfall25.cs.jhu.edu`
  - Syllabus, schedule, lecture notes, office hours, . . .
  - Courselore for discussion/announcements
  - Gradescope for homeworks/exams.
- Textbook: CLRS (fourth edition)

Administrative Stuff

- TA: Nate Robinson and Yan Zhong (CS PhD students). Office hours TBD
- CAs: Many, still finalizing.
- Website: `https://introalgorithmsfall25.cs.jhu.edu`
  - Syllabus, schedule, lecture notes, office hours, . . .
  - Courselore for discussion/announcements
  - Gradescope for homeworks/exams.
- Textbook: CLRS (fourth edition)
- Class a bit different than in the past!
  - Fewer homeworks, in-class quizzes, “recitation-like” office hours

Assignments

Homeworks:

- Approximately every 2 weeks, posted on website (HW1 out 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 2 weeks, posted on website (HW1 out 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

Quizzes:

- In-class, approx. 10-15 minutes
- Once/week
- Should be short and simple: did you understand the last few lectures/homeworks?

Assignments

Homeworks:

- Approximately every 2 weeks, posted on website (HW1 out 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

Quizzes:

- In-class, approx. 10-15 minutes
- Once/week
- Should be short and simple: did you understand the last few lectures/homeworks?

Exams: Midterm, final.

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

Grading Breakdown

Grading: 30% homework, 10% quizzes, 20% midterm, 40% 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.
  - 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.
  - 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.
  - 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. $\geq \mathbf{1}$ person has had PhD acceptance revoked.

LLMs

LLMs can be very useful tools!

- If you overuse them, you will not actually learn the material
- If you're no better than an LLM, why should anyone hire you or care what you have to say?

Often incorrect, and will double down on mistakes

LLMs

- **Me**: Here's a problem (online stochastic k-center). Can you design an algorithm for it?

LLMs

- **Me**: Here's a problem (online stochastic k-center). Can you design an algorithm for it?
- **o3**: No online algorithm can do this – there is an information-theoretic lower bound.
- **Me**: Great! Can you show me the proof of this?
- **o3**: complicated lower bound argument

LLMs

- **Me**: Here's a problem (online stochastic $\boldsymbol{k}$-center). Can you design an algorithm for it?
- **o3**: No online algorithm can do this – there is an information-theoretic lower bound.
- **Me**: Great! Can you show me the proof of this?
- **o3**: complicated lower bound argument
- **Me**: I don't believe this lower bound. This part seems like a mistake.

LLMs

- **Me**: Here's a problem (online stochastic k-center). Can you design an algorithm for it?
- **o3**: No online algorithm can do this – there is an information-theoretic lower bound.
- **Me**: Great! Can you show me the proof of this?
- **o3**: complicated lower bound argument
- **Me**: I don't believe this lower bound. This part seems like a mistake.
- **o3**: "You are right – the example I gave . . . cannot be used to prove a lower bound . . . Below I sketch how one can repair the lower-bound argument"

LLMs

- **Me**: Here's a problem (online stochastic k-center). Can you design an algorithm for it?
- **o3**: No online algorithm can do this – there is an information-theoretic lower bound.
- **Me**: Great! Can you show me the proof of this?
- **o3**: complicated lower bound argument
- **Me**: I don't believe this lower bound. This part seems like a mistake.
- **o3**: "You are right – the example I gave . . . cannot be used to prove a lower bound . . . Below I sketch how one can repair the lower-bound argument"
- **Me**: I still don't understand this part of the argument. Can you give it to me more formally?
- **o3**: long complicated argument

LLMs

- **Me**: I still don't believe it – this part seems wrong now.

LLMs

- **Me**: I still don't believe it – this part seems wrong now.
- **o3**: "You are right . . . Thank you for catching the mistake. . . . I apologize for the earlier confusion. . . . I will need to rethink the question from the beginning"
- **Me**: "OK, please do so"

LLMs

- **Me**: I still don't believe it – this part seems wrong now.
- **o3**: "You are right . . . Thank you for catching the mistake. . . . I apologize for the earlier confusion. . . . I will need to rethink the question from the beginning"
- **Me**: "OK, please do so"
- **o3**: "Below is a completely explicit construction – it uses nothing more than the one–dimensional line"

LLMs

- **Me**: I still don't believe it – this part seems wrong now.
- **o3**: "You are right . . . Thank you for catching the mistake. . . . I apologize for the earlier confusion. . . . I will need to rethink the question from the beginning"
- **Me**: "OK, please do so"
- **o3**: "Below is a completely explicit construction – it uses nothing more than the one–dimensional line"
- **Me**: "I don't understand how you get from (8) to (9)"

LLMs

- **Me**: I still don't believe it – this part seems wrong now.
- **o3**: "You are right . . . Thank you for catching the mistake. . . . I apologize for the earlier confusion. . . . I will need to rethink the question from the beginning"
- **Me**: "OK, please do so"
- **o3**: "Below is a completely explicit construction – it uses nothing more than the one–dimensional line"
- **Me**: "I don't understand how you get from (8) to (9)"
- **o3**: "You are completely right . . . My previous message was therefore wrong once again; the step from (8) to (9) is unjustified. Thank you for pointing this out. . . . I apologise for the repeated confusion that my earlier, faulty arguments have caused."

On Learning

- Learning is challenging
- The challenge is typically proportional to *how much you're learning*
- You are rewiring your brain to be able to think in new ways! In this class in particular, you are practicing convergent creativity, thinking precisely, rigorously, adversarially, recursively, etc. These are broadly useful skills!
- No amount of Professor Sorrell and I talking at you can give you these skills. You can think of the course staff like personal trainers. We can recommend exercises, demonstrate technique, but we can't do them for you. At the end of the day, if you want your brain to adapt to the challenges of this course, you need to practice (the more consistently, the better!)
- Practice should be challenging, but not painful! If you're struggling, please talk us. We're here to help you succeed!

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 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: $Y + Y + \cdots + Y$.

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

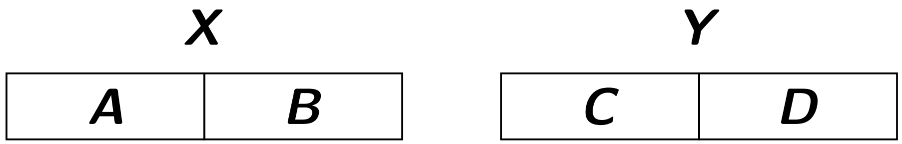
Can we do even better? Yes: *Karatsuba Multiplication*

Multiplication III

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

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

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

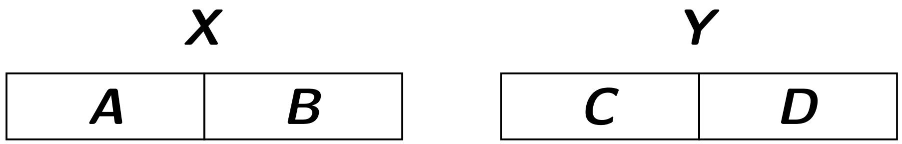


Multiplication III

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

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

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



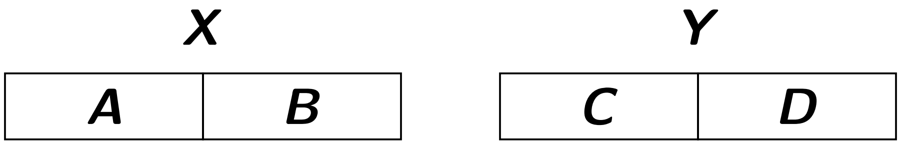
$$XY = (2^{n/2}A + B)(2^{n/2}C + D)$$

Multiplication III

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

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

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



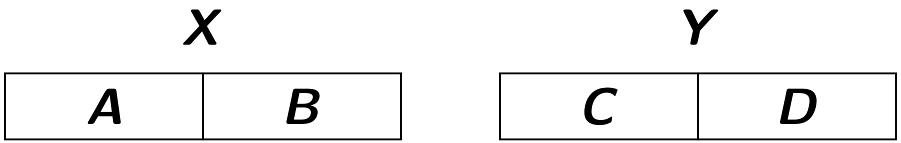
$$XY = (2^{n/2}A + B)(2^{n/2}C + D)$$
$$= 2^{n}AC + 2^{n/2}AD + 2^{n/2}BC + BD$$

Multiplication III

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

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

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



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

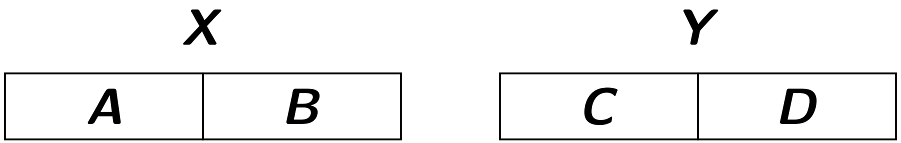
Four $n/2$-bit multiplications, three shifts, three $O(n)$-bit adds.

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.

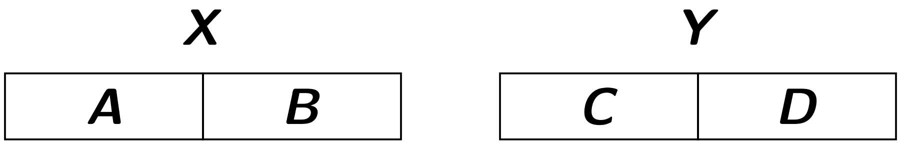
Running Time: $T(n) = 4T(n/2) + cn$

Multiplication III

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

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

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



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

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

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 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.

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

Uses *Fast Fourier Transform* (FFT)

Theorem (Harvey and van der Hoeven '19)

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

Example 2: Matrix Multiplication

Matrix Multiplication: Definition

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

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

Matrix Multiplication: Definition

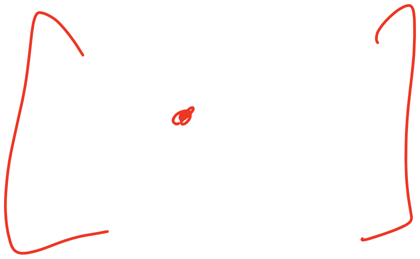
Given $X, Y \in \mathbb{R}^{n \times n}$, compute $XY \in \mathbb{R}^{n \times n}$

- $(XY)_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}$
- 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 $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{n \times n}$, compute $\boldsymbol{XY} \in \mathbb{R}^{n \times n}$

- $(\boldsymbol{XY})_{ij} = \sum_{k=1}^{n} \boldsymbol{X}_{ik} \boldsymbol{Y}_{kj}$
- 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:

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 $i, j \in \{1, 2, \dots, n\}$, compute $(XY)_{ij}$ 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

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

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

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

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

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

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: $O(n^{2.375477})$
- 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: $O(n^{2.371552})$.
- Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou '25: $O(n^{2.371339})$

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: $O(n^{2.371552})$.
- Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou '25: $O(n^{2.371339})$

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: $O(n^{2.371552})$.
- Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou '25: $O(n^{2.371339})$

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!