Lecture 1: Introduction

Michael Dinitz

August 31, 2021
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 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

- 8th time teaching this class (Fall 2014 - Fall 2021).
  - I'm still learning – let me know if you have suggestions!
  - I'd appreciate it if you watched lectures synchronously with your webcam on if possible, but not required.
- 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: Wednesdays 2 - 4pm (zoom link on webpage; not lecture link).

Administrative Stuff

- TA: Isabel Cachola (CS PhD student). Office hours TBD
- Head CA: Tanuj Alapati (senior undergraduate). Office hours: Mondays 10am-12pm
- CAs: Many, still finalizing.
- Website: `http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2021/`
  - Syllabus, schedule, lecture notes, ...
  - Campuswire for discussion/announcements
  - Gradescope for homeworks/exams.

Administrative Stuff

- TA: Isabel Cachola (CS PhD student). Office hours TBD
- Head CA: Tanuj Alapati (senior undergraduate). Office hours: Mondays 10am-12pm
- CAs: Many, still finalizing.
- Website: `http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2021/`
  - Syllabus, schedule, lecture notes, . . .
  - Campuswire for discussion/announcements
  - Gradescope for homeworks/exams.

- Expectation is that you are in Baltimore and attending synchronously, even though course is online.
  - If you're not in Baltimore, must be because of either disability or visa-related issue.
  - Undergrads: official accommodations.
  - Grad students: permission from your department, email me, fill out WSE survey (see Campuswire post)

Course Size and Waitlist

Course is very full!

- 433: 75 enrolled, 31 waitlist
- 633: 75 enrolled, 28 waitlist

Problem with expanding: grading.

- Grading has to be done individually, by hand!
- Don't know how many CAs will be assigned, but seems like fewer than past few years

Course Size and Waitlist

Course is very full!

- 433: 75 enrolled, 31 waitlist
- 633: 75 enrolled, 28 waitlist

Problem with expanding: grading.

- Grading has to be done individually, by hand!
- Don't know how many CAs will be assigned, but seems like fewer than past few years

If enough CAs get assigned, would like to allow junior/senior CS majors and CS MSE students to enroll off of waitlist

Course Size and Waitlist

Course is very full!

- 433: 75 enrolled, 31 waitlist
- 633: 75 enrolled, 28 waitlist

Problem with expanding: grading.

- Grading has to be done individually, by hand!
- Don't know how many CAs will be assigned, but seems like fewer than past few years

If enough CAs get assigned, would like to allow junior/senior CS majors and CS MSE students to enroll off of waitlist

In meantime, if you're on the waitlist:

- Take course in the spring!
- Or attend lectures, do homeworks.

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 up ≤ 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 up ≤ 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: final. No midterm.

- Used to have midterm, but I don't like exams for online courses. Trying without midterm this year.
- 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 up ≤ 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: final. No midterm.

- Used to have midterm, but I don't like exams for online courses. Trying without midterm this year.
- Final: in person, scheduled by registrar. 3 hours, traditional, closed book.

Grading: 65% homework, 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 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 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 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. $\geq \mathbf{1}$ 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 $\mathbf{X}$, want to output $\mathbf{f(X)}$. 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 $\mathbf{X}$, want to output $\mathbf{f(X)}$. 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 $\mathbf{X}$, want to output $\mathbf{f(X)}$. 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 $\mathbf{n}$-bit integers $\mathbf{X}$ and $\mathbf{Y}$. Compute $\mathbf{XY}$.

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

How to do this?

Definition of multiplication:

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

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 $\mathbf{n}$-bit integers $\mathbf{X}$ and $\mathbf{Y}$. Compute $\mathbf{XY}$.

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

How to do this?

Definition of multiplication:

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

Running time:

- $\Theta(\mathbf{Y})$ or $\Theta(\mathbf{X})$.

Multiplication I

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

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

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

How to do this?

Definition of multiplication:

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

Running time:

- $\mathbf{\Theta(Y)}$ or $\mathbf{\Theta(X)}$.
- Could be $\mathbf{\Theta(2^n)}$. *Exponential* in size of input $(\mathbf{2n})$.

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:

- $\mathbf{O(n)}$ column additions, each takes $\mathbf{O(n)}$ time $\implies$ $\mathbf{O(n^2)}$ 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*

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

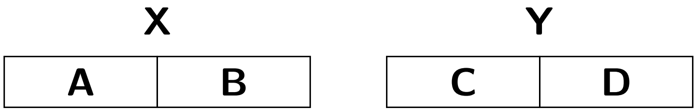
$$Y = 2^{n/2}C + D$$

Multiplication III

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

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

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



Multiplication III

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

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

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

X

A	**B**

Y

C	**D**

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

Multiplication III

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

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

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

X

A	**B**

Y

C	**D**

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

Multiplication III

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

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

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

X

A	B

Y

C	D

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

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

Multiplication III

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

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

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

X		Y	
A	**B**	**C**	**D**

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

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

Running Time: $\mathbf{T(n) = 4T(n/2) + cn}$

Multiplication III

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

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

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

X			Y	
A	B		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**:

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

Karatsuba Multiplication

Rewrite equation for $\mathbf{XY}$:

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

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

Karatsuba Multiplication

Rewrite equation for $\mathbf{XY}$:

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

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

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

Karatsuba Multiplication

Rewrite equation for $\mathbf{XY}$:

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

Three $\mathbf{n/2}$-bit multiplications, $\mathbf{O(1)}$ shifts and $\mathbf{O(n)}$-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 $\mathbf{O(n \log^2 n)}$*-time algorithm for multiplication.*

Uses *Fast Fourier Transform* (FFT)

Even Better Multiplication?

Can we do even better than Karatsuba?

Theorem (Karp)

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

Uses *Fast Fourier Transform* (FFT)

Theorem (Harvey and van der Hoeven '19)

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

Example 2: Matrix Multiplication

Matrix Multiplication: Definition

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

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

Matrix Multiplication: Definition

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

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

Algorithm from definition:

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

Matrix Multiplication: Definition

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

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

Algorithm from definition:

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

Running time:

Matrix Multiplication: Definition

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

- $(\mathbf{XY})_{ij} = \sum_{k=1}^{n} \mathbf{X}_{ik} \mathbf{Y}_{kj}$
- 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 $(\mathbf{XY})_{ij}$ 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 **X** and **Y** each into four $(\mathbf{n}/2) \times (\mathbf{n}/2)$ matrices:

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

Strassen I

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

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

So can rewrite $\mathbf{XY}$:

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

Strassen I

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

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

So can rewrite $\mathbf{XY}$:

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

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

Strassen II

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

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

Strassen II

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

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

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

Strassen II

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

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

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

Strassen III

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

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

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

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

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: $\mathbf{O(n^{2.375477})}$
- Virginia Vassilevska Williams '13: $\mathbf{O(n^{2.3728642})}$
- François Le Gall '14: $\mathbf{O(n^{2.3728639})}$
- Josh Alman and Virginia Vassilevska Williams '21: $\mathbf{O(n^{2.3728596})}$

Further Progress

- Coppersmith and Winograd '90: $\mathbf{O(n^{2.375477})}$
- Virginia Vassilevska Williams '13: $\mathbf{O(n^{2.3728642})}$
- François Le Gall '14: $\mathbf{O(n^{2.3728639})}$
- Josh Alman and Virginia Vassilevska Williams '21: $\mathbf{O(n^{2.3728596})}$

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

Further Progress

- Coppersmith and Winograd '90: $\mathbf{O(n^{2.375477})}$
- Virginia Vassilevska Williams '13: $\mathbf{O(n^{2.3728642})}$
- François Le Gall '14: $\mathbf{O(n^{2.3728639})}$
- Josh Alman and Virginia Vassilevska Williams '21: $\mathbf{O(n^{2.3728596})}$

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

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

See you Thursday!