Lecture 1: Introduction

Michael Dinitz

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.
  ▶ Real prerequisite: mathematical maturity
About me

- 7th time teaching this class (Fall 2014 - Fall 2020).
  - First time online, first time with slides!
  - 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 theory (algorithmic game theory, complexity theory) and theory of networking.

- Office hours: Wednesdays 4 - 6pm
Administrative Stuff

- TA: Yasamin Nazari (theory PhD student). Office hours Monday 10:30 - 11:30am, 8:15 - 9:15pm
- CAs: Many, still finalizing.
- Website: http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2020/
  - Syllabus, schedule, lecture notes...
  - Piazza for discussion/announcements
  - Gradescope for homeworks/exams.
Administrative Stuff

- TA: Yasamin Nazari (theory PhD student). Office hours Monday 10:30 - 11:30am, 8:15 - 9:15pm
- CAs: Many, still finalizing.
- Website: [http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2020/](http://www.cs.jhu.edu/~mdinitz/classes/IntroAlgorithms/Fall2020/)
  - Syllabus, schedule, lecture notes...
  - Piazza for discussion/announcements
  - Gradescope for homeworks/exams.

- Grading: 50% homework, 35% final exam, 15% midterm
- “Curve”: Historically, average ≈ 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).
Assignments

Homeworks: Approximately every 1.5 weeks, posted on website (HW1 out now!)

- *Must* be typeset (\LaTeX\ preferred, not required)
- Work in groups up ≤ 3 (highly recommended). But *individual* writeups.
  - Work together at a (virtual) 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 now!)

- *Must* be typeset (*\LaTeX* preferred, not required)
- Work in groups up ≤ 3 (highly recommended). But *individual* writeups.
  - Work together at a (virtual) whiteboard to solve, then write up yourself.
  - Write group members at top of homework
- 120 late hours (5 late days) total

Exams: midterm and final.

- Original plan: Midterm in class, final as scheduled by registrar.
- New plan:
  - Midterm: takehome. Exact format TBD, but individual.
    - One week? 24 hours? 1.25 hours?
  - Final: TBD, based on midterm experience
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. $\geq 1$ 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 $X$, want to output $f(X)$. 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 $X$, want to output $f(X)$. 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 $X$, want to output $f(X)$. 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.

Experiments can (maybe) convince you that something is true. But can’t tell you why!
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: \( Y + Y + \cdots + Y \).
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 \( \Theta(X) \).
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 \( \Theta(X) \).
- Could be \( \Theta(2^n) \). Exponential in size of input (2\( n \)).
Better idea?
Better idea? Grade school algorithm!
Better idea? Grade school algorithm!

\[
\begin{align*}
110110 & \quad = 54 \\
x & \quad 101001 \quad = 41 \\
\hline
110110 \\
110110 \\
+ 110110 \\
\hline
100010100110 \quad = \quad 2 + 4 + 32 + 128 + 2048 \quad = \quad 2214
\end{align*}
\]
Better idea? Grade school algorithm!

\[\begin{align*}
110110 & = 54 \\
x & \quad 101001 = 41 \\
\hline
110110 \\
110110 \\
+ & \quad 110110 \\
\hline
100010100110 & = 2 + 4 + 32 + 128 + 2048 = 2214
\end{align*}\]

Running time:
Better idea? Grade school algorithm!

\[
\begin{align*}
110110 & = 54 \\
x & \quad 101001 = 41 \\
\hline
110110 & \\
110110 & \quad \rightarrow \quad 30(4) \\
+ & \quad 110110 \\
\hline
100010100110 & \quad \rightarrow \quad 0(4) \\
& \quad \rightarrow \quad 2 + 4 + 32 + 128 + 2048 = 2214
\end{align*}
\]

Running time:
- \(O(n)\) column additions, each takes \(O(n)\) time \(\rightarrow O(n^2)\) time.
- Better than obvious algorithm!
Can we do even better?

\[
XY = \left(2^n A + B\right) \left(2^n C + D\right) = 2^n AC + 2^n AD + \ldots
\]

Four \(n\) bit multiplications, three shifts, three \(O(n)\) bit adds.

Running Time:

\[
T(n) = 4T(n/2) + cn \Rightarrow T(n) = \mathcal{O}(n^2)
\]
Can we do even better? Yes: Karatsuba Multiplication
Can we do even better? Yes: *Karatsuba Multiplication*

\[
X = 2^{n/2} A + B \\
Y = 2^{n/2} C + D
\]
Can we do even better? Yes: *Karatsuba Multiplication*

\[
\begin{align*}
X &= 2^{n/2}A + B \\
Y &= 2^{n/2}C + D
\end{align*}
\]
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*

\[
X = 2^{n/2}A + B
\]
\[
Y = 2^{n/2}C + D
\]

\[
XY = (2^{n/2}A + B)(2^{n/2}C + D)
= 2^nAC + 2^{n/2}AD + 2^{n/2}BC + BD
\]
Can we do even better? Yes: *Karatsuba Multiplication*

\[
X = 2^{n/2}A + B
\]
\[
Y = 2^{n/2}C + D
\]

\[
XY = (2^{n/2}A + B)(2^{n/2}C + D)
\]
\[
= 2^nAC + 2^{n/2}AD + 2^{n/2}BC + BD
\]

Four \(n/2\)-bit multiplications, three shifts, three \(O(n)\)-bit adds.
Can we do even better? Yes: *Karatsuba Multiplication*

\[ X = 2^{n/2}A + B \]
\[ Y = 2^{n/2}C + D \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

\[
XY = (2^{n/2}A + B)(2^{n/2}C + D)
= 2^nAC + 2^{n/2}AD + 2^{n/2}BC + BD
\]

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

Running Time: \(T(n) = 4T(n/2) + cn\)
Can we do even better? Yes: *Karatsuba Multiplication*

\[
X = 2^{n/2}A + B
\]
\[
Y = 2^{n/2}C + D
\]

\[
XY = (2^{n/2}A + B)(2^{n/2}C + D)
\]
\[
= 2^nAC + 2^{n/2}AD + 2^{n/2}BC + BD
\]

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 $XY$:

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

$$= 2^{n/2} (A + B)(C + D) + (2^n - 2^{n/2}) AC + (1 - 2^{n/2}) BD$$
Rewrite equation for $XY$:

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

$$= 2^{n/2} (A + B)(C + D) + (2^n - 2^{n/2}) AC + (1 - 2^{n/2}) BD$$

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

Rewrite equation for $XY$:

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

$$= 2^{n/2}(A + B)(C + D) + (2^n - 2^{n/2})AC + (1 - 2^{n/2})BD$$

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

$$\Longrightarrow T(n) = 3T(n/2) + c'n$$
Karatsuba Multiplication

Rewrite equation for $XY$:

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

$$= 2^{n/2} (A + B)(C + D) + (2^n - 2^{n/2}) AC + (1 - 2^{n/2}) BD$$

*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 $X, Y \in \mathbb{R}^{n \times n}$, 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 $X, Y \in \mathbb{R}^{n \times n}$, 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:
- $O(n^2)$ entries, each entry takes $n$ multiplications and $n - 1$ additions $\implies O(n^3)$ time.
Break $X$ and $Y$ each into four $(n/2) \times (n/2)$ matrices:

\[
X = \begin{bmatrix}
A & B \\
C & D
\end{bmatrix}, \quad
Y = \begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
\]
Break \( X \) and \( Y \) each into four \((n/2) \times (n/2)\) matrices:

\[
X = \begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\quad \quad \quad \quad \quad \quad
Y = \begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
\]

So can rewrite \( XY \):

\[
XY = \begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]
Strassen I

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

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

So can rewrite $XY$:

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

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

\[ XY = \begin{pmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{pmatrix} \]

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

\[ XY = \begin{array}{cc}
AE + BG & AF + BH \\
CE + DG & CF + DH \\
\end{array} \]

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

\[
\begin{array}{|c|c|}
\hline
AE + BG & AF + BH \\
\hline
CE + DG & CF + DH \\
\hline
\end{array}
\]

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**

\[
XY = \begin{array}{cc}
AE + BG & AF + BH \\
CE + DG & CF + DH \\
\end{array}
\]
Strassen III

\[
\begin{array}{c|c}
AE + BG & AF + BH \\
\hline
CE + DG & CF + DH \\
\end{array}
\]

\[
XY = M_1 + M_2 - M_3 + M_4 + M_5 - M_6 + M_7
\]

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

Strassen's algorithm for matrix multiplication:

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

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

\[ XY = \begin{array}{c|c}
  \begin{array}{c}
    M_1 + M_4 - M_5 + M_7 \\
    M_2 + M_4
  \end{array} & \begin{array}{c}
    M_3 + M_5 \\
    M_1 - M_2 + M_3 + M_6
  \end{array}
\end{array} \]
Strassen IV

\[
\begin{align*}
M_1 &= (A + D)(E + H) \\
M_2 &= (C + D)E \\
M_3 &= A(F - H) \\
M_4 &= D(G - E) \\
M_5 &= (A + B)H \\
M_6 &= (C - A)(E + F) \\
M_7 &= (B - D)(G + H)
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\
\hline
M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \\
\hline
\end{array}
\]

Only seven \((n/2) \times (n/2)\) matrix multiplies, \(O(1)\) additions
Strassen IV

\[
\begin{align*}
M_1 &= (A + D)(E + H) & M_2 &= (C + D)E & M_3 &= A(F - H) \\
M_4 &= D(G - E) & M_5 &= (A + B)H & M_6 &= (C - A)(E + F) \\
M_7 &= (B - D)(G + H)
\end{align*}
\]

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

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: $O(n^{2.3728639})$
Further Progress

- Coppersmith and Winograd ’90: $O(n^{2.375477})$
- Virginia Vassilevska Williams ’13: $O(n^{2.3728642})$
- François Le Gall ’14: $O(n^{2.3728639})$

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: $O(n^{2.3728639})$

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!