CS/ECE 374: Algorithms & Models of Computation

Sariel Har-Peled

University of Illinois, Urbana-Champaign

Fall 2024

Administrivia, Introduction

Lecture 1

Tuesday, August 27, 2024

LaTeXed: August 27, 2024 10:04

https://courses.grainger.illinois.edu/cs374al1/fa2024/

374A

Part I

Administrivia

Instructional Staff

1. Instructor: Sariel Har-Peled
2. **360** students.
3. 9 Teaching Assistants
4. ≈20 Undergraduate Course Assistants
5. Office hours: See course webpage
6. Contacting us: Use private notes on EdStem to reach course staff. Direct email only for sensitive or confidential information.

Online resources

1. Webpage: General information, announcements, homeworks, course policies
   `https://courses.grainger.illinois.edu/cs374al1/fa2024/`
2. Gradescope: Homework submission and grading, regrade requests
3. PrairieLearn (PL): Guided homework problems.
4. EdStem: Announcements, online questions and discussion, contacting course staff (via private notes)
5. Discord: For general discussion, etc.
6. Go to class webpage, "get access" link to get access.

See course webpage for links.

Important: check EdStem regularly.

Prereqs and Resources

1. Prerequisites: CS 173 (discrete math), CS 225 (data structures)
2. Recommended books: (not required)
    2.1 Introduction to Theory of Computation by Sipser
    2.2 Introduction to Automata, Languages and Computation by Hopcroft, Motwani, Ullman
    2.3 Algorithms by Dasgupta, Papadimitriou & Vazirani.
    Available online for free!
    2.4 Algorithm Design by Kleinberg & Tardos
3. Lecture notes/slides/pointers: available on course web-page
4. Additional References
    4.1 Lecture notes of Jeff Erickson, Sariel Har-Peled, Mahesh Viswanathan and others
    4.2 Introduction to Algorithms: Cormen, Leiserson, Rivest, Stein.
    4.3 Computers and Intractability: Garey and Johnson.

Grading Policy: Overview

1. Homeworks: 28%
2. Midterm exams: 42% (**2 × 21%**)
3. Final exam: 30% (covers the full course content)

Midterm exam dates:

1. Midterm 1: See webpage (should be on the schedule).
2. Midterm 2: See webpage.

No conflict exam offered unless you have a valid excuse.

Homeworks

1. GPS problem every week: Due on Tuesday at 9am. Individually on PrairieLearn.
2. Homework every week (2 problems): Due on Wednesdays at 9am on Gradescope. Assigned at least a week in advance.
3. Homeworks can be worked on in groups of up to 3 and each group submits one written solution.
4. Important: academic integrity policies. See course web page.

More on Homeworks:

1. No extensions or late homeworks accepted.
2. To compensate, only top 26 problems will be used in computing final grade. Homeworks typically have three problems each.
3. Important: Read homework FAQ/instructions on website.

Discussion Sessions/Labs

1. Only first 7 discussion sections on schedule would be held. If you are registered to the later ones, attend an earlier discussion section.
2. 50min problem solving session led by TAs
3. Two times a week
4. Go to your assigned discussion section
5. Bring pen and paper! Or tablet, or whatever you can use to scribble on.

Advice

1. Attend lectures, please ask plenty of questions.
2. Attend discussion sessions.
3. Don't skip homework and don't copy homework solutions. Each of you should think about all the problems on the home work - do not divide and conquer.
4. Use pen and paper since that is what you will do in exams which count for 75% of the grade. Keep a note book.
5. Study regularly and keep up with the course.
6. This is a course on problem solving. Solve as many as you can! Books/notes have plenty.
7. This is also a course on providing rigorous proofs of correctness. Refresh your 173 background on proofs.
8. Ask for help promptly. Make use of office hours/EdStem.

Homework 1 + GPS 1

1. HW 1 is posted on the class website. PL GPS1 is also available.
2. GPS 1 due on Tuesday.
3. HW 1 due next Wednesday.
4. Groups of size up to 3.

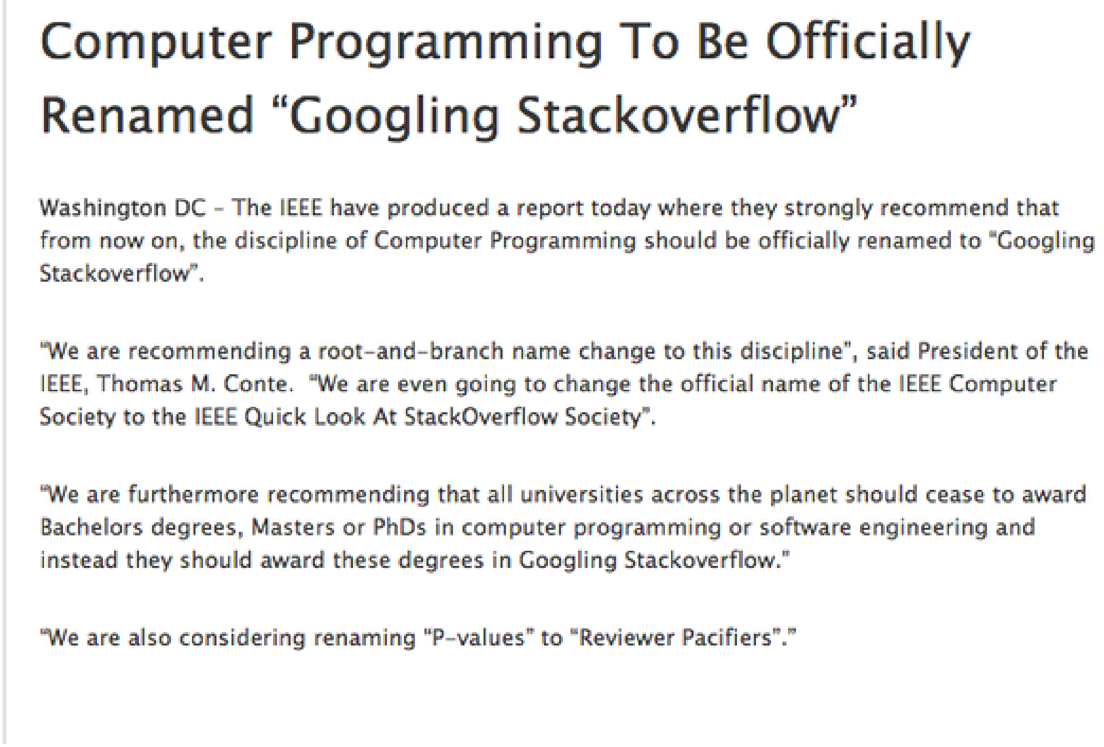
Miscellaneous

Please contact instructors if you need special accommodations.

Lectures are being taped. See course webpage.

Part II

Course Goals and Overview



Computer Programming To Be Officially Renamed “Googling Stackoverflow”

Washington DC – The IEEE have produced a report today where they strongly recommend that from now on, the discipline of Computer Programming should be officially renamed to "Googling Stackoverflow".

"We are recommending a root-and-branch name change to this discipline", said President of the IEEE, Thomas M. Conte. "We are even going to change the official name of the IEEE Computer Society to the IEEE Quick Look At StackOverflow Society".

"We are furthermore recommending that all universities across the planet should cease to award Bachelors degrees, Masters or PhDs in computer programming or software engineering and instead they should award these degrees in Googling Stackoverflow."

"We are also considering renaming "P-values" to "Reviewer Pacifiers"."

High-Level Questions

1. Modeling: States/Graphs/Recursion/Algorithms.
2. Algorithms
   2.1 What is an algorithm?
   2.2 What is an <u>efficient</u> algorithm?
   2.3 Some fundamental algorithms for basic problems
   2.4 Broadly applicable techniques in algorithm design
3. What is a mathematical definition of a computer?
   3.1 Is there a formal definition?
   3.2 Is there a "universal" computer?
4. What can computers compute?
   4.1 Are there tasks that our computers cannot do?

Course Structure

Course divided into three parts:

1. Basic automata theory: finite state machines, regular languages, hint of context free languages/grammars, Turing Machines
2. Algorithms and algorithm design techniques
3. Undecidability and NP-Completeness, reductions to prove intractability of problems

Goals

1. Algorithmic thinking
2. Learn/remember some basic tricks, algorithms, problems, ideas
3. Understand/appreciate limits of computation (intractability)
4. Appreciate the importance of algorithms in computer science and beyond (engineering, mathematics, natural sciences, social sciences, ...)

Historical motivation for computing

1. Fast (and automated) numerical calculations
2. Automating mathematical theorem proving

Models of Computation vs Computers

1. Model of Computation: an “idealized mathematical construct” that describes the primitive instructions and other details
2. Computer: an actual “physical device” that implements a very specific model of computation

Models and devices:

1. Algorithms: usually at a high level in a model
2. Device construction: usually at a low level
3. Intermediaries: compilers
4. How precise? Depends on the problem!
5. Physics helps implement a model of computer
6. Physics also inspires models of computation

Models of Computation vs Computers

1. Model of Computation: an “idealized mathematical construct” that describes the primitive instructions and other details
2. Computer: an actual “physical device” that implements a very specific model of computation

Models and devices:

1. Algorithms: usually at a high level in a model
2. Device construction: usually at a low level
3. Intermediaries: compilers
4. How precise? Depends on the problem!
5. Physics helps implement a model of computer
6. Physics also inspires models of computation

Adding Numbers

Problem Given two n-digit numbers x and y, compute their sum.

Basic addition

$$\begin{array}{r} 3141 \\ +7798 \\ \hline 10939 \end{array}$$

Adding Numbers

```
$c = 0$
for $i = 1$ to $n$ do
    $z = x_i + y_i$
    $z = z + c$
    If $(z > 10)$
        $c = 1$
        $z = z - 10$        (equivalently the last digit of $z$)
    Else $c = 0$
    print $z$
End For
If $(c == 1)$ print $c$
```

1. Primitive instruction is addition of two digits
2. Algorithm requires $O(n)$ primitive instructions

Adding Numbers

```
$c = 0$
for $i = 1$ to $n$ do
    $z = x_i + y_i$
    $z = z + c$
    If $(z > 10)$
        $c = 1$
        $z = z - 10$      (equivalently the last digit of $z$)
    Else $c = 0$
    print $z$
End For
If $(c == 1)$ print $c$
```

1. Primitive instruction is addition of two digits
2. Algorithm requires $O(n)$ primitive instructions

Multiplying Numbers

Problem Given two n-digit numbers x and y, compute their product.

Grade School Multiplication

Compute “partial product” by multiplying each digit of y with x and adding the partial products.

$$\begin{array}{r} 3141 \\ \times 2718 \\ \hline 25128 \\ 3141 \\ 21987 \\ 6282 \\ \hline 8537238 \end{array}$$

Time analysis of grade school multiplication

1. Each partial product: $\Theta(n)$ time
2. Number of partial products: $\leq n$
3. Adding partial products: n additions each $\Theta(n)$ (Why?)
4. Total time: $\Theta(n^2)$
5. Is there a faster way?

Fast Multiplication

Best known algorithm: $O(n \log n \cdot 2^{O(\log^* n)})$ time [Furer 2008]

Previous best time: $O(n \log n \log \log n)$ [Schonhage-Strassen 1971]

Conjecture: there exists an $O(n \log n)$ time algorithm

We don't fully understand multiplication!
Computation and algorithm design is non-trivial!

Fast Multiplication

Best known algorithm: $O(n \log n \cdot 2^{O(\log^* n)})$ time [Furer 2008]

Previous best time: $O(n \log n \log \log n)$ [Schonhage-Strassen 1971]

Conjecture: there exists an $O(n \log n)$ time algorithm

We don't fully understand multiplication!
Computation and algorithm design is non-trivial!

Post Correspondence Problem

Given: Dominoes, each with a top-word and a bottom-word.

b	***ba***	***abb***	***abb***	***a***
bbb	***bbb***	***a***	***baa***	***ab***

Can one arrange them, using any number of copies of each type, so that the top and bottom strings are equal?

abb	***ba***	***abb***	***a***	***abb***	***b***
a	***bbb***	***a***	***ab***	***baa***	***bbb***

Halting Problem

Debugging problem: Given a program M and string x, does M halt when started on input x?

Simpler problem: Given a program M, does M halt when it is started? Equivalently, will it print "Hello World"?

One can prove that there is no algorithm for the above two problems!

Halting Problem

Debugging problem: Given a program M and string x, does M halt when started on input x?

Simpler problem: Given a program M, does M halt when it is started? Equivalently, will it print "Hello World"?

One can prove that there is no algorithm for the above two problems!

Halting Problem

Debugging problem: Given a program M and string x, does M halt when started on input x?

Simpler problem: Given a program M, does M halt when it is started? Equivalently, will it print “Hello World”?

One can prove that there is no algorithm for the above two problems!