CS/ECE 374: Algorithms & Models of Computation

Nikita Borisov

University of Illinois, Urbana-Champaign

Fall 2018

Administrivia, Introduction

Lecture 1

August 28, 2018

Part I

Administrivia

Instructional Staff

1. Instructors: Chandra Chekuri (A section) and Nikita Borisov (B section)
2. 11 Teaching Assistants
3. ?? Undergraduate Course Assistants
4. Office hours: See course webpage
5. Contacting us: Use *private notes* on Piazza to reach course staff. Direct email only for sensitive or confidential information.

Section A vs B

Only lectures different for the sections.

Home work, exams, labs etc will be common.

Homework groups can be across sections.

Online resources

1. Webpage: General information, announcements, homeworks, course policies `courses.engr.illinois.edu/cs374`
2. Gradescope: Homework submission and grading, regrade requests
3. Moodle: Quizzes, solutions to homeworks, grades
4. Piazza: Announcements, online questions and discussion, contacting course staff (via private notes)

See course webpage for links

Important: check Piazza/course web page at least once each day

Prereqs and Resources

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

Grading Policy: Overview

1. Quizzes: 0% for self-study
2. Homeworks: 24%
3. Midterm exams: 44% (**2 × 22%**)
4. Final exam: 32% (covers the full course content)

Midterm exam dates:

1. Midterm 1: Mon, October 1, 7–9.30pm
2. Midterm 2: Mon, November 12, 7–9.30pm

No conflict exam offered unless you have a valid excuse.

Homeworks

1. Self-study quizzes each week on *Moodle*. No credit but stronlgy recommended.
2. One homework every week: Due on Wednesdays at 10am 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 (except Homework 0).
4. Important: academic integrity policies. See course web page.

More on Homeworks

1. No extensions or late homeworks accepted.
2. To compensate, nine problems will be dropped. Homeworks typically have three problems each.
3. Important: Read homework faq/instructions on website.

Discussion Sessions/Labs

1. 50min problem solving session led by TAs
2. Two times a week
3. Go to your assigned discussion section
4. Bring pen and paper!

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 76% 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 rigourous proofs of correctness. Refresh your 173 background on proofs.
8. Ask for help promptly. Make use of office hours/Piazza.

Homework 0

1. HW 0 is posted on the class website. Quiz 0 available on Moodle.
2. HW 0 due on Wednesady September 5th at 10am on Gradescope
3. HW 0 to be done and submitted *individually*.

Miscellaneous

Please contact instructors if you need special accommodations.

Lectures are being taped. See course webpage.

Part II

Course Goals and Overview

High-Level Questions

1. Computation, formally.
   1. Is there a formal definition of a computer?
   2. Is there a "universal" computer?
2. Algorithms
   1. What is an algorithm?
   2. What is an *efficient* algorithm?
   3. Some fundamental algorithms for basic problems
   4. Broadly applicable techniques in algorithm design
3. Limits of computation.
   1. Are there tasks that our computers cannot do?
   2. How do we prove lower bounds?
   3. Some canonical hard problems.

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, ...)

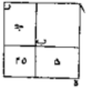
History

History

Muhammad ibn Musa al-Khwarizmi (c.780–c.850)

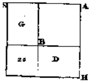
Text on Algebra

على تسعة وثلثين ليتم السطح الاعظم الذي هو سطح ره فبلغ
ذلك كله اربعة وستين فاخذنا جذرها وهو ثمانية وهو احد
اضلاع السطح الاعظم فاذا نقصنا منه مثل ما زدنا عليه وهو
خمسة بقي ثلثة وهو ضلع سطح اب الذي هو المال وهو جذره
والمال تسعة وهذه صورته



واما مال واحد وعشرون درهما يعدل عشرة اجذاره فانا
نجعل المال سطحا مربعا مجهول الاضلاع وهو سطح اد ثم نضم
اليه سطحا متوازي الاضلاع عرضه مثل احد اضلاع سطح اد وهو
ضلع دن والسطح دب فصار طول السطحين جميعا ضلع جه
وقد علمنا ان طوله عشرة من العدد لان كل سطح مربع
متساوي الاضلاع والزوايا فان احد اضلاعه مضروبا في واحد جذر
ذلك السطح وفي اثنين جذراه فلما قال مال واحد وعشرون
يعدل عشرة اجذاره علمنا ان طول ضلع هج عشرة اعداد لان
ضلع جد جذر المال فقسمنا ضلع جه بنصفين على نقطة

the first quadrate, which is the square, and the two quadrangles on its sides, which are the ten roots, make together thirty-nine. In order to complete the great quadrate, there wants only a square of five multiplied by five, or twenty-five. This we add to thirty-nine, in order to complete the great square S H. The sum is sixty-four. We extract its root, eight, which is one of the sides of the great quadrangle. By subtracting from this the same quantity which we have before added, namely five, we obtain three as the remainder. This is the side of the quadrangle A B, which represents the square; it is the root of this square, and the square itself is nine. This is the figure:—



Demonstration of the Case: "a Square and twenty-one Dirhems are equal to ten Roots."

We represent the square by a quadrate A D, the length of whose side we do not know. To this we join a parallelogram, the breadth of which is equal to one of the sides of the quadrate A D, such as the side H N. This paralellogram is H B. The length of the two

Algorithm Description

If some one says: "You divide ten into two parts: multiply the one by itself; it will be equal to the other taken eighty-one times." Computation: You say, ten less a thing, multiplied by itself, is a hundred plus a square less twenty things, and this is equal to eighty-one things. Separate the twenty things from a hundred and a square, and add them to eighty-one. It will then be a hundred plus a square, which is equal to a hundred and one roots.

Algorithm Description

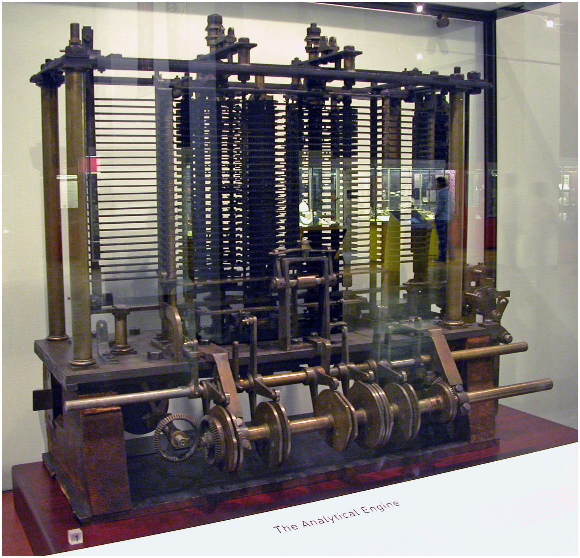
If some one says: "You divide ten into two parts: multiply the one by itself; it will be equal to the other taken eighty-one times." Computation: You say, ten less a thing, multiplied by itself, is a hundred plus a square less twenty things, and this is equal to eighty-one things. Separate the twenty things from a hundred and a square, and add them to eighty-one. It will then be a hundred plus a square, which is equal to a hundred and one roots.

$$(10 - x)^2 = 81x$$
$$x^2 - 20x + 100 = 81x$$
$$x^2 + 100 = 101x$$

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

First Computer



Babbage's analytical engine—designed in 1837, never built.

First Program

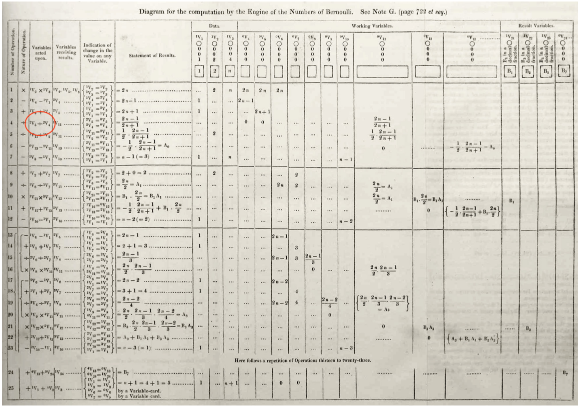


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\{^1V_2 = {}^1V_2;\ {}^1V_3 = {}^1V_3\}$	$= 2n$
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\{^1V_4 = {}^2V_4;\ {}^1V_1 = {}^1V_1\}$	$= 2n - 1$
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\{^1V_5 = {}^2V_5;\ {}^1V_1 = {}^1V_1\}$	$= 2n + 1$
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\{^2V_5 = {}^0V_5;\ {}^2V_4 = {}^0V_4\}$	$= \frac{2n-1}{2n+1}$
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\{^1V_{11} = {}^2V_{11};\ {}^1V_2 = {}^1V_2\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\{^2V_{11} = {}^0V_{11};\ {}^0V_{13} = {}^1V_{13}\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$
7	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\{^1V_3 = {}^1V_3;\ {}^1V_1 = {}^1V_1\}$	$= n - 1 (= 3)$
8	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\{^1V_2 = {}^1V_2;\ {}^0V_7 = {}^1V_7\}$	$= 2 + 0 = 2$
9	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\{^1V_6 = {}^1V_6;\ {}^0V_{11} = {}^3V_{11}\}$	$= \frac{2n}{2} = A_1$
10	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\{^1V_{21} = {}^1V_{21};\ {}^3V_{11} = {}^3V_{11}\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$
11	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\{^1V_{12} = {}^0V_{12};\ {}^1V_{13} = {}^2V_{13}\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$
12	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\{^1V_{10} = {}^2V_{10};\ {}^1V_1 = {}^1V_1\}$	$= n - 2 (= 2)$
13	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\{^1V_6 = {}^2V_6;\ {}^1V_1 = {}^1V_1\}$	$= 2n - 1$
14	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\{^1V_1 = {}^1V_1;\ {}^1V_7 = {}^2V_7\}$	$= 2 + 1 = 3$
15	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\{^2V_6 = {}^2V_6;\ {}^2V_7 = {}^2V_7\}$	$= \frac{2n-1}{3}$
16	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\{^1V_8 = {}^0V_8;\ {}^3V_{11} = {}^4V_{11}\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$
17	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\{^2V_6 = {}^3V_6;\ {}^1V_1 = {}^1V_1\}$	$= 2n - 2$
18	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\{^2V_7 = {}^3V_7;\ {}^1V_1 = {}^1V_1\}$	$= 3 + 1 = 4$
19	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\{^3V_6 = {}^3V_6;\ {}^3V_7 = {}^3V_7\}$	$= \frac{2n-2}{4}$
20	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\{^1V_9 = {}^0V_9;\ {}^4V_{11} = {}^5V_{11}\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$
21	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\{^1V_{22} = {}^1V_{22};\ {}^0V_{12} = {}^2V_{12}\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$
22	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\{^2V_{12} = {}^0V_{12};\ {}^2V_{13} = {}^3V_{13}\}$	$= A_0 + B_1 A_1 + B_3 A_3$
23	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\{^2V_{10} = {}^3V_{10};\ {}^1V_1 = {}^1V_1\}$	$= n - 3 (= 1)$
					Here follows a repetition of Operations thirteen to twenty-three.
24	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\{^4V_{13} = {}^0V_{13};\ {}^0V_{24} = {}^1V_{24}\}$	$= B_7$
25	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\{^1V_1 = {}^1V_1;\ {}^1V_3 = {}^1V_3;\ {}^5V_6 = {}^0V_6;\ {}^5V_7 = {}^0V_7\}$	$= n + 1 = 4 + 1 = 5$; by a Variable-card. by a Variable card.

Ada Lovelace's "Note G" describing how to calculate Bernouilli numbers using the analytical engine.

First Bug!

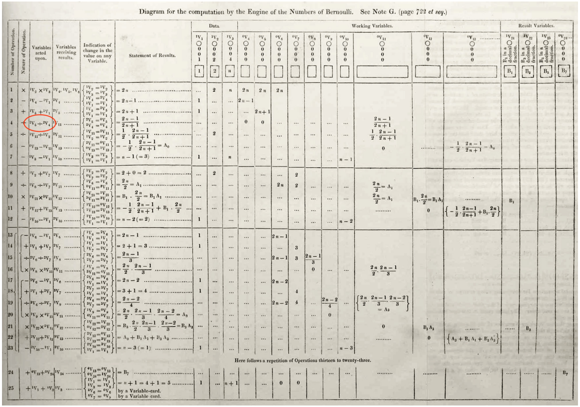


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

Number of Operation. | Nature of Operation. | Variables acted upon. | Variables receiving results. | Indication of change in the value on any Variable. | Statement of Results. | Data. | Working Variables. | Result Variables.

Here follows a repetition of Operations thirteen to twenty-three.

Ada Lovelace's "Note G" describing how to calculate Bernouilli numbers using the analytical engine.
This version contains a bug!

Models of Computation vs. Computers

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
```

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.

```
      3141
    × 2718
    ------
     25128
     3141
   21987
   6282
   -------
   8537238
```

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: $O\left(n \log n \cdot 4^{\log^* n}\right)$ by Harvey and van der Hoeven, published in 2018!

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

Fast Multiplication

Best known algorithm: $O\left(n \log n \cdot 4^{\log^* n}\right)$ by Harvey and van der Hoeven, published in 2018!
Conjecture: there exists an $O(n \log n)$ time algorithm

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

Aside about O-notation

Some previous versions of multiplication are still widely used:

- Karatsuba algorithm $O(n^{\log_2 3})$ [1962]
- Schönhage-Strassen (FFT) $O(n \log n \log \log n)$ [1971]

Why?

Aside about O-notation

Some previous versions of multiplication are still widely used:

- Karatsuba algorithm $O(n^{\log_2 3})$ [1962]
- Schönhage-Strassen (FFT) $O(n \log n \log \log n)$ [1971]

Why? Fürer's algorithm (2007) $O(n 2^{O(\log^* n)})$

Aside about O-notation

Some previous versions of multiplication are still widely used:

- Karatsuba algorithm $O(n^{\log_2 3})$ [1962]
- Schönhage-Strassen (FFT) $O(n \log n \log \log n)$ [1971]

Why? Fürer's algorithm (2007) $O(n 2^{O(\log^* n)})$

…beats Schönhage-Strassen for numbers greater than $2^{2^{64}}$.

Halting Problem

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

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"?

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!