CS 350 Algorithms and Complexity

Winter 2019

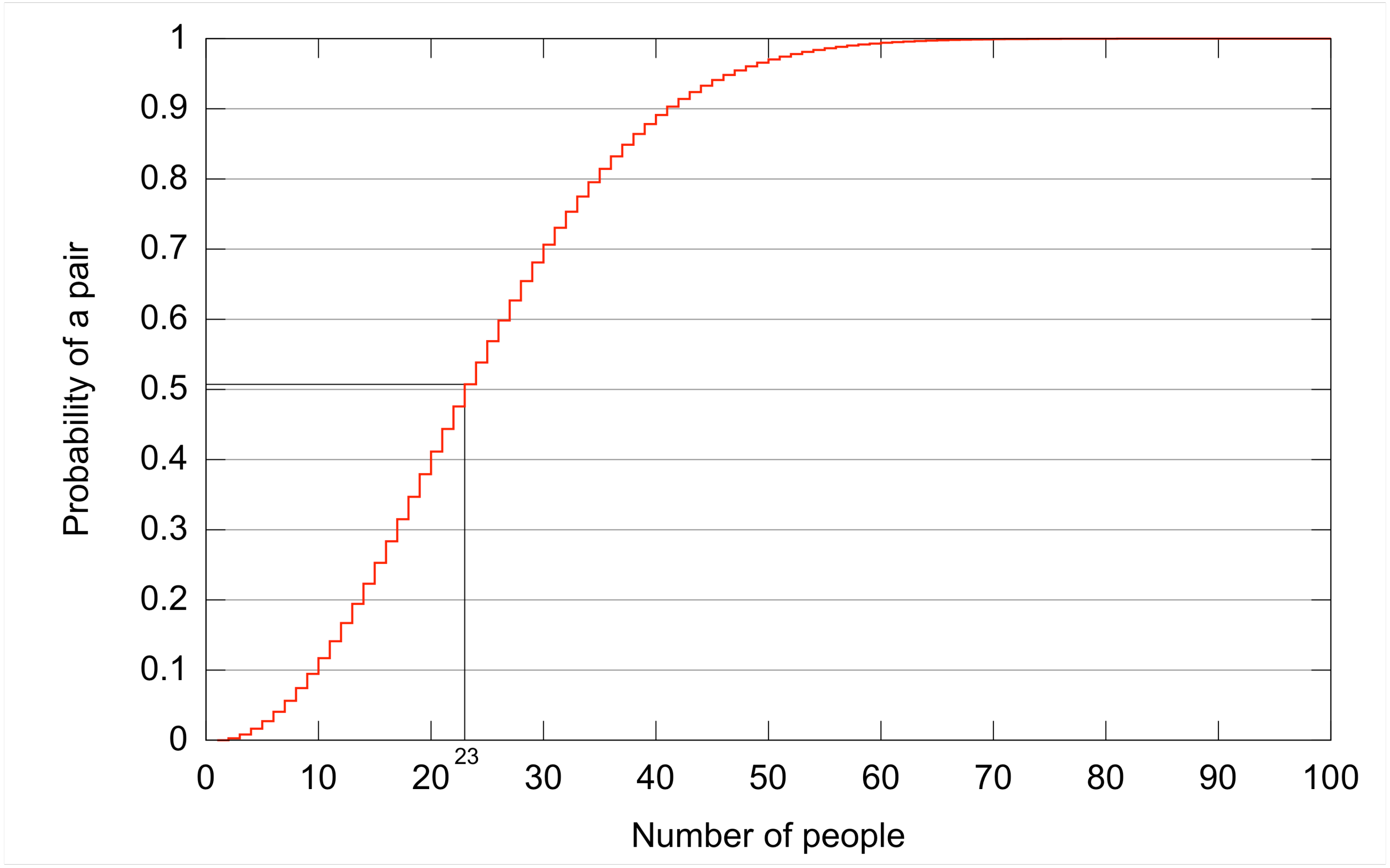
Lecture 1: Introduction

Andrew P. Black
based on material by Mark P. Jones
Department of Computer Science
Portland State University

Happy Birthday!

- Are there two people in this room with the same birthday?

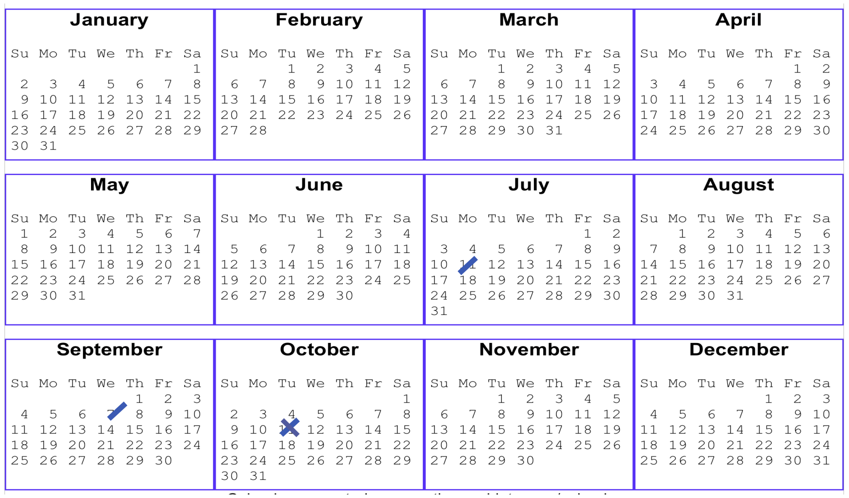
Surprisingly likely …



Happy Birthday!

- Are there two people in this room with the same birthday?
    - if so: what is that day?
- How might we go about answering this question?

One Possible Algorithm:



January

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

February

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28					

March

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

April

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

May

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

June

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

July

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11 ✓	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

August

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

September

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7 ✓	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

October

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11 ✗	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

November

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

December

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Algorithm:

```
for each person in the room:
    find each's birthday on
                        calendar
    if calendar day is marked
    then return that day
    else mark day on calendar
return false
```

Another Possible Algorithm

11 September
7 July
11 October

- Algorithm

```
for each person in the room:
    write each's birthday on the whiteboard
    for other people in the room:
        if other's birthday is on the whiteboard
            then return birthday
return false
```

Question:

- Which of these algorithms is better?
  - You decide what “better” means

- A: Calendar Algorithm

- B: Whiteboard algorithm

Question:

- True or False: there are no other algorithms for this problem

- A: True
- B: False

The Birthday Problem

- Are there two people in this room with the same birthday?
- How might we go about answering this question?
- How much effort would it take?
  - In the best case?
  - In the worst case?
- Are there faster methods?
- Are there approximate methods?

Fermat’s Last Theorem:

- Find integers, $a>0$, $b>0$, $c>0$, $n>2$, such that:

$$a^n + b^n = c^n$$

  if they exist.

- Could you write a program to “answer” this question?
  - A: Yes
  - B: No

Fermat’s Last Theorem:

- Could you write an algorithm to “answer” this question?
  - A: Yes
  - B: No

Complexity:

- How do we compare algorithms?
- How do we compare problems?
- What are the limits of computation?
- What opportunities do different models of computation provide?

Why Study Algorithms?

- Because it's a required class!

Why Study Algorithms?

- Because you want (amongst other things)
  - a job, or career, or way to support yourselves while doing what you love
- How can studying algorithms help you?

Studying Algorithms can help you:

- To recognize common patterns, or problem-solving strategies …

- To be able to analyze algorithms for time- and space-efficiency

- To strengthen your mathematical, programming, and problem-solving skills …

Studying Algorithms can help you:

- To build a “repertoire” of algorithmic building blocks
- Because there is no better optimization than replacing a bad algorithm with a good one!
- Because it’s fun and enlightening!
- Because hiring companies care about algorithms

Administrative Details

Contact Details:

Andrew P Black:

- Office: FAB 115-10
- Telephone: (503) 725 2411
- Email: apblack@pdx.edu

Web page:

- http://www.cs.pdx.edu/~black/cs350/

Piazza page:

- piazza.com/pdx/winter2019/cs350

Teaching Assistant / Grader

- We have a Teaching Assistant: Arjun Koduru
  - Looking for an office and time to hold office hours
- CS Tutors can help
- I can help too!
  - My office hours
    - Tuesday noon–12:30
    - Thursday 16:00–17:00
  - or by appointment (**telephone** to make one)

Piazza:

- Sign up at piazza.com/pdx/winter2019/cs350
- Piazza is required reading!
- The instructor, TA, and all students will see and can respond to questions posted on Piazza
  - Except for private questions to the instructor
- Interaction on Piazza counts for a (small) part of your grade.
- Please don’t send me email. Instead, send a private message to "Instructors" on Piazza.

Assessment:

20% on homeworks (most, but not all, will be formally graded; I'll be very clear about this when assignments are given out.)

20% on midterm (Provisionally, 12 Feb)

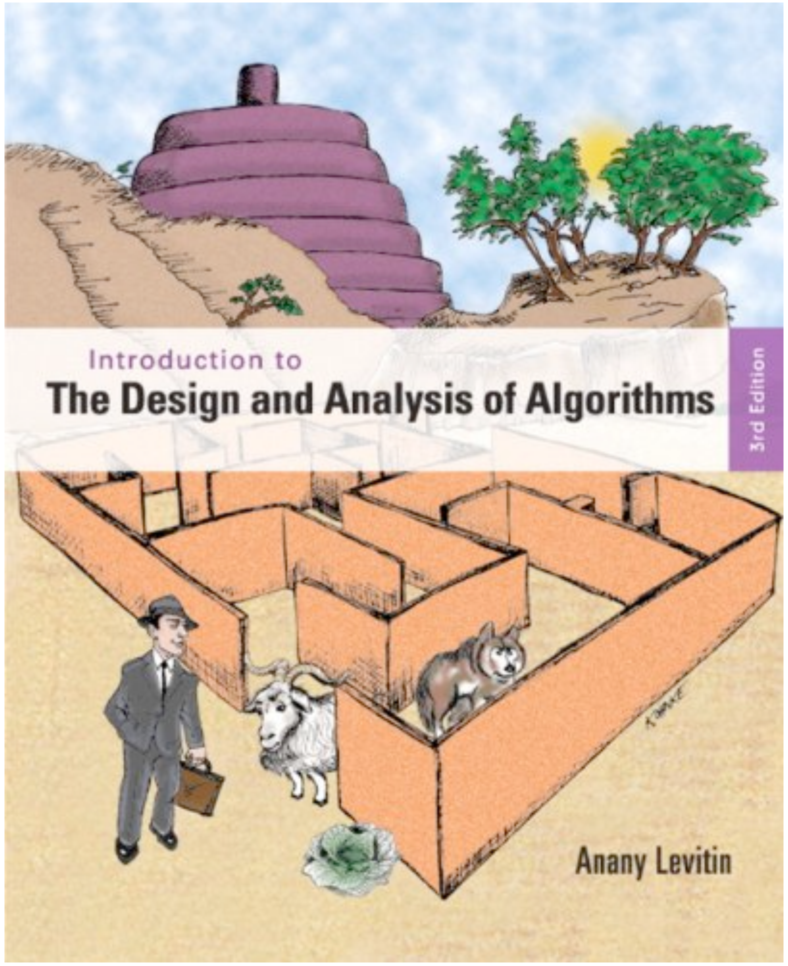
30% on Final (Tuesday 19 Mar, 10:15–12:05)

20%: Term paper/project: empirical analysis of algorithm behavior; further details after midterm

10%: Participation, in class & electronically.

Required Text:

- Introduction to the Design and Analysis of Algorithms (3rd Edition) by Anany Levitin. Addison Wesley).
- Very readable book: it's a MOC



Required Reading

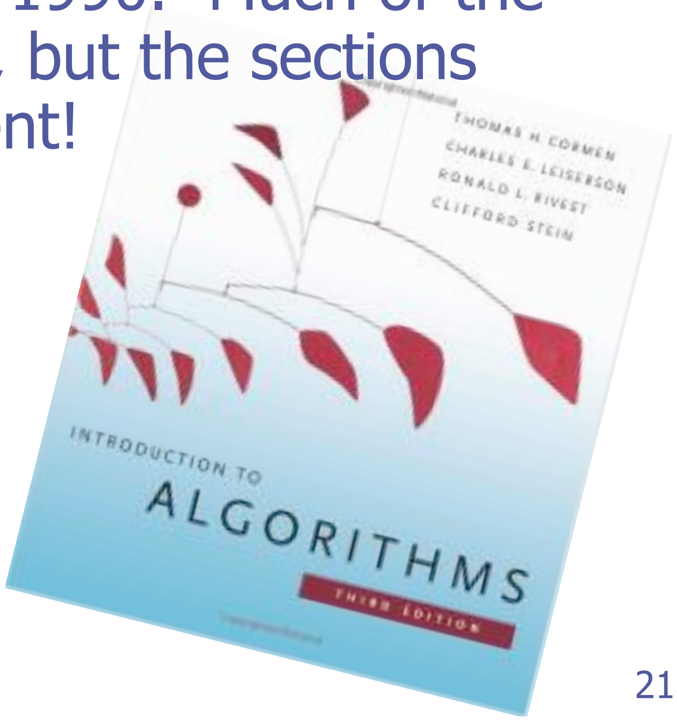
- I’m not requiring you to just *own* Levitin, I’m requiring you to *read* it too!
- After today, I won’t be lecturing on the material in the book
- Instead, class sessions will be used as an opportunity to interact with the material and to challenge your understanding.

Useful Reference Text:

- Introduction to Algorithms, 2nd or 3rd Ed

  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
  - Published by McGraw-Hill / MIT Press
  - A first edition was published in 1990. Much of the material appears in all editions, but the sections numbers are sometimes different!
  - Frequently called “CLRS”

- http://mitpress.mit.edu/algorithms/



Catalog Description:

- Techniques for the design and analysis of algorithms
- Case studies of existing algorithms:
  - sorting
  - searching
  - graph algorithms
  - dynamic programming
  - matrix multiplication
  - fast Fourier transformation
- NP-completeness

Course Goals

Upon Successful completion of this course, students will be able to:

1. Analyze the running time and space complexity of algorithms.
2. Use the big Oh notation. (e.g., O(n lg n).)
3. Describe how to prove the correctness of an algorithm.
4. Use the mathematical techniques required to prove the time complexity of a program/algorithm. (e.g., limits and sums of series.)
5. Perform inductive proofs.
6. Prove and apply the Master Theorem.
7. Describe the notions of P, NP, NPC, and NP-hard.
8. Compare the rates of growth of functions.
9. Apply algorithmic complexity principles in the design of programs.
10. Design divide and conquer and dynamic programming algorithms.

Provisional Plan:

Weeks	Topics	Levitin
2	Introduction to Algorithm Analysis, asymptotic notation, recurrence relations	§1–2
3	Brute force, Decrease & Conquer, Transform & Conquer	§3–4.5
1	Divide & Conquer, Master Theorem, Quicksort & Mergesort	§5
2	Space & Time tradeoffs. Dynamic programming, Greedy algorithms	§7–9.4
1	Introduction to complexity theory, P, NP, NPC, etc. Proof of Correctness	§11.1–11.3

I'll be flexible!

- This schedule is intended as a rough guide only.
- Based on class interaction, we'll go slower (or faster) as necessary
  - Your feedback will help to determine the pace!
- Suggestions/requests for other topics are always welcome
- If you think that the book, or your instructor are too vague: ask questions!

Policies:

- By default, all deadlines are firm. There will be penalties for unexcused late assignments.
- We will be as flexible as possible in accommodating special circumstances (talk to me in advance if you have a chance)
- We follow the standard policies for academic integrity:
  - discussion is good;
  - items turned in should be your own, individual work;
  - cheating is not acceptable.
- I put material on the website or Piazza rather than handing out paper.

Prerequisites:

- CS 250, CS251, CS311:
  - A solid grounding in discrete math
  - The ability to write down a formal proof

- This isn’t a course about programming, but some programming experience is required

- Assignment 0 — set today — is designed to remind you of some of the essential pre-requisites

Programming Languages:

- Our primary goal is to learn about <u>general</u> concepts.
- A lot of the programs in this class will be given using “pseudo-code”
- For concrete examples, I might use Java, C, Smalltalk, Grace, or Pascal.

What Questions do you have?

Don’t be shy …

… don’t be afraid to ask questions!

(… don’t be afraid to offer suggestions too!)

Use Piazza to ask questions or leave feedback. If you want to leave **anonymous** feedback for the instructor, you can do so at https://sayat.me/PSU_CS350

Algorithms: Introduction

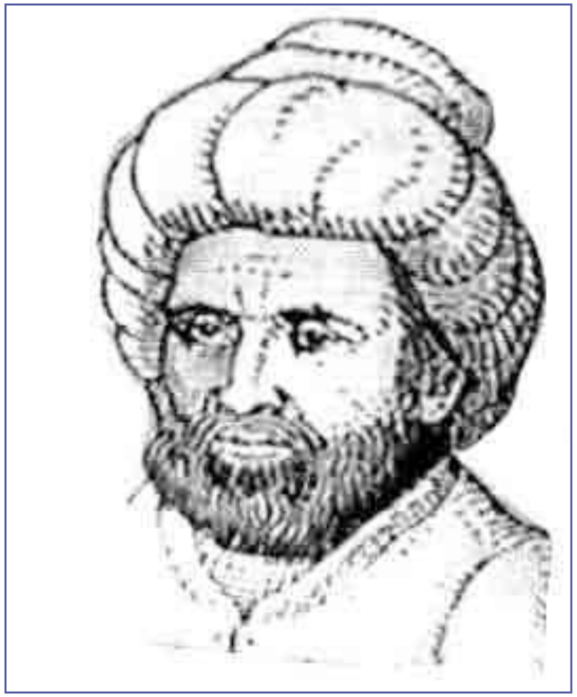
Goals for this lecture:

- To understand the structure, content, and expectations of the course
- To explore the concept of an “algorithm”, and the relationship with programs
- To appreciate how algorithms can be designed and analyzed

The origins of “Algorithm”:

Al-Khawarizmi (~790-850)

- Iranian mathematician, geographer, astronomer.
- Credited with inventing zero (!), negative numbers, the decimal system, etc …
- Famous treatise: `Hisab al-jabr w’al-muqabala’, from which we get the word `**algebra**’.



What is an "Algorithm"?

A precise set of rules specifying how to solve a particular problem.

Expressed at a level of abstraction that allows:

- A focus on essential aspects only;
- Machine and language independence;
- Rigorous scientific/mathematical analysis.

Example: Recipes in a Cookbook

- How to prepare your favorite dish.
- A recipe lists ingredients, measurements, equipment, timings, and the sequence of steps that need to be followed.
- In principle, anyone, in any kitchen, can follow the recipe.

Example Recipe

1. Line a loose-bottomed quiche tin with pastry, and bake until done.
2. Sauté vegetables and season to taste
3. Allow vegetables to cool a little, and stir into previously-prepared egg mixture
4. Pour into the pastry case and bake until firm, but not brown.
   - 2 or 3 minutes before the quiche is done, sprinkle cheese on top and return to oven to brown.

What did you notice about the recipe?

- Is it an Algorithm? (A: Yes, B: No)
- Features of a Program?

1. Line a loose-bottomed quiche tin with pastry, and bake until done.
2. Sauté vegetables and season to taste
3. Allow vegetables to cool a little, and stir into previously-prepared egg mixture
4. Pour into the pastry case and bake until firm, but not brown.
    - 2 or 3 minutes before the quiche is done, sprinkle cheese on top and return to oven to brown.

Example: Driving Directions

- Turn right out of the driveway, cross César Chávez Blvd., and turn right on 37th Ave.
- Proceed North to Gladstone.
- Turn left on Gladstone and continue down the hill to 22nd Ave.
- Turn left onto Bush, first right onto 21st Ave.
- Left on Powell; right on 21st; half-right onto Frontage Road; take Bike path on left sidewalk
- Continue under the railway tracks on Bike Path
- Cross pedestrian crossing at end of Bike Path, and turn right.
- Turn 180° left on bike path, and then continue on Gideon Street
- …

Example: Driving Directions

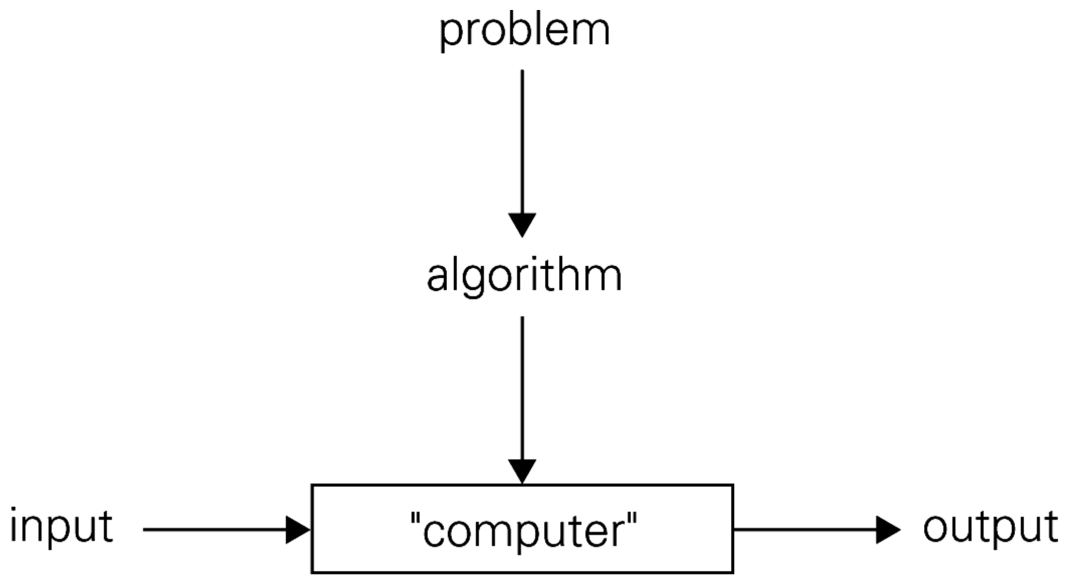
- Turn right out of the driveway, right again on César Chávez Blvd
- Turn left on Holgate, and continue to McLoughlin Blvd.
- Exit onto Ross Island Bridge.
- After crossing bridge, turn right onto Corbett.
- Turn half-left onto Caruthers St.
- Turn Right onto 6th Ave
- Hot Lips Pizza will be on your left, on the corner with Hall St.
- Park wherever you can.

Example: Driving Directions

- How to get from one place to another.
- A sequence of instructions indicating:
  - Landmarks to look for;
  - Junctions at which to turn;
  - Distances to travel; etc
- Abstracts away from inessential details: when to apply brakes, how fast to travel, etc.
- Is mode of transportation an inessential detail?

Definition: Algorithm

- An Algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite time.



Properties of this Definition:

- each step must be unambiguous
- range of allowable inputs must be specified
- process must terminate
- the same algorithm can be represented in different ways
- there can be several algorithms for the same problem
    - they can differ dramatically in concept, speed, and space requirements

Algorithms

- Which of the following are important characteristics of an algorithm?
  - A. Instructions are unambiguous
  - B. Instructions work for some inputs
  - C. The algorithm depends on the size of the input
  - D. Instructions work for all inputs
  - E. Algorithm always stops
  - F. Instructions are easy to understand

Example: GCD

the GCD of two non-negative integers, not both zero, is the largest integer that divides both exactly.

Note: careful description of legal inputs

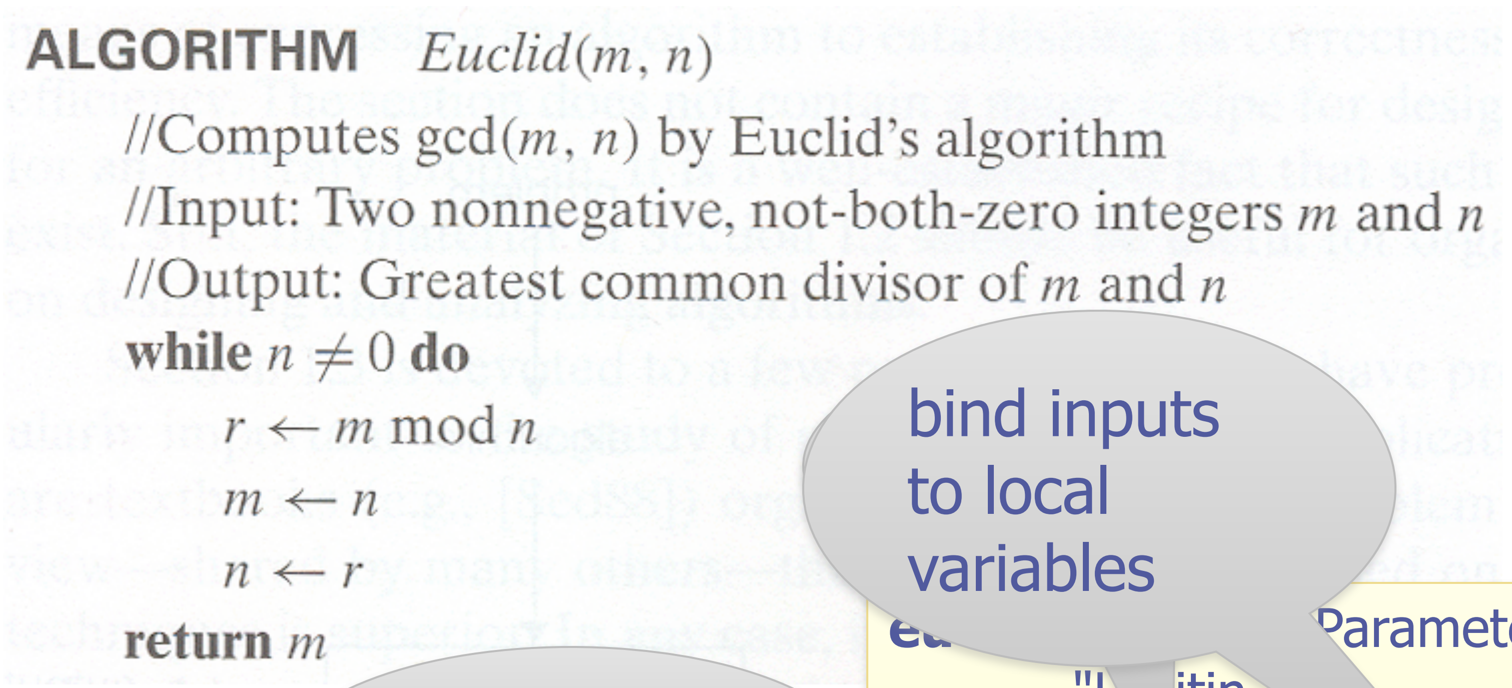
Euclid's Algorithm for GCD

Based on repeated application of

GCD(m, n) = GCD(n, m mod n)

until (m mod n) = 0

```
ALGORITHM  $Euclid(m, n)$
    //Computes gcd($m$, $n$) by Euclid's algorithm
    //Input: Two nonnegative, not-both-zero integers $m$ and $n$
    //Output: Greatest common divisor of $m$ and $n$
    while $n \neq 0$ do
        $r \leftarrow m \bmod n$
        $m \leftarrow n$
        $n \leftarrow r$
    return $m$
```



```
ALGORITHM  Euclid(m, n)
    //Computes gcd(m, n) by Euclid's algorithm
    //Input: Two nonnegative, not-both-zero integers m and n
    //Output: Greatest common divisor of m and n
    while n ≠ 0 do
        r ← m mod n
        m ← n
        n ← r
    return m
```

bind inputs to local variables

Declare local variables

```
eu              Parameter
        "L  itin g
        | m n r |
        m ← self.
        n ← anIntegerParameter.
        [n = 0] whileFalse: [
                r ← m mod: n.
                m ← n.
                n ← r ].
        ↑ m
```

Another Algorithm for GCD

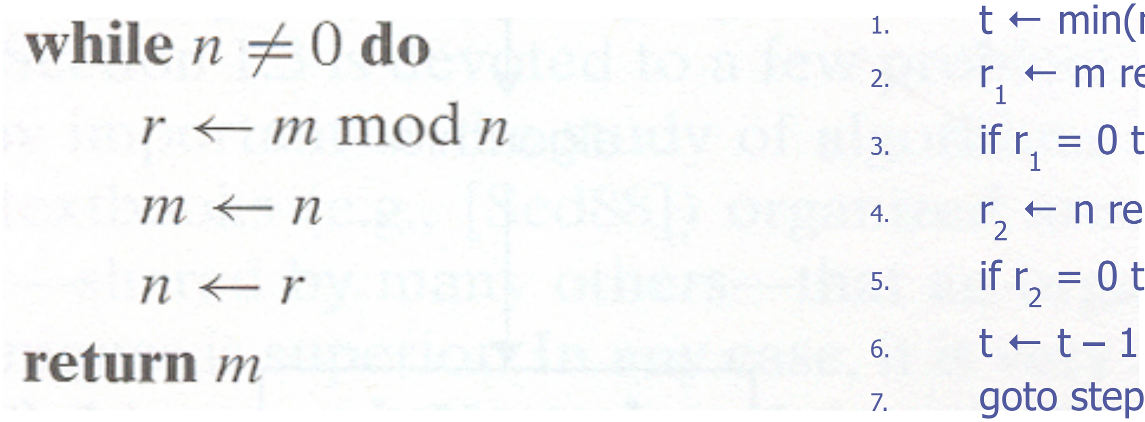
- Consecutive integer checking algorithm

```
1. t ← min(m, n)
2. $r_1$ ← m rem t.
3. if $r_1 = 0$ then goto step 4 else goto step 6
4. $r_2$ ← n rem t.
5. if $r_2 = 0$ then return t
6. t ← t – 1
7. goto step 2
```

Compare these two algorithms

- Presentation style:
  - A: Euclid’s is bad
  - B: Consecutive is bad
  - C: both are sort of OK
  - D: both are bad

Euclid



```
while $n \neq 0$ do
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return $m$
```

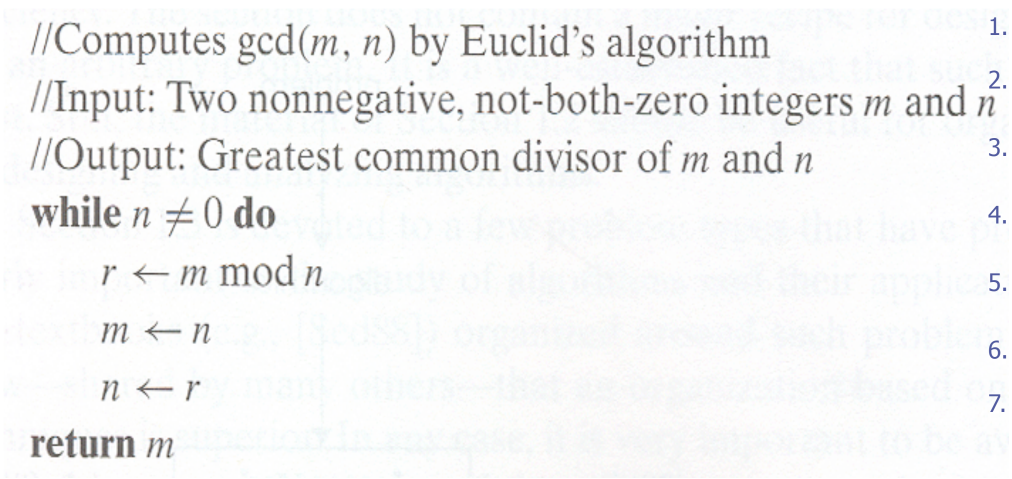
Consecutive

```
1.    $t \leftarrow \min(m, n)$
2.    $r_1 \leftarrow m$ rem $t$.
3.    if $r_1 = 0$ then goto step 4 else goto step 6
4.    $r_2 \leftarrow n$ rem $t$.
5.    if $r_2 = 0$ then return $t$
6.    $t \leftarrow t - 1$
7.    goto step 2
```

Compare these two algorithms

- Valid inputs:
  - A: Euclid's specifies validity of inputs
  - B: Consecutive specifies validity of inputs
  - C: both specify validity of inputs
  - D: neither specifies validity of inputs

Euclid



```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while $n \neq 0$ do
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return $m$
```

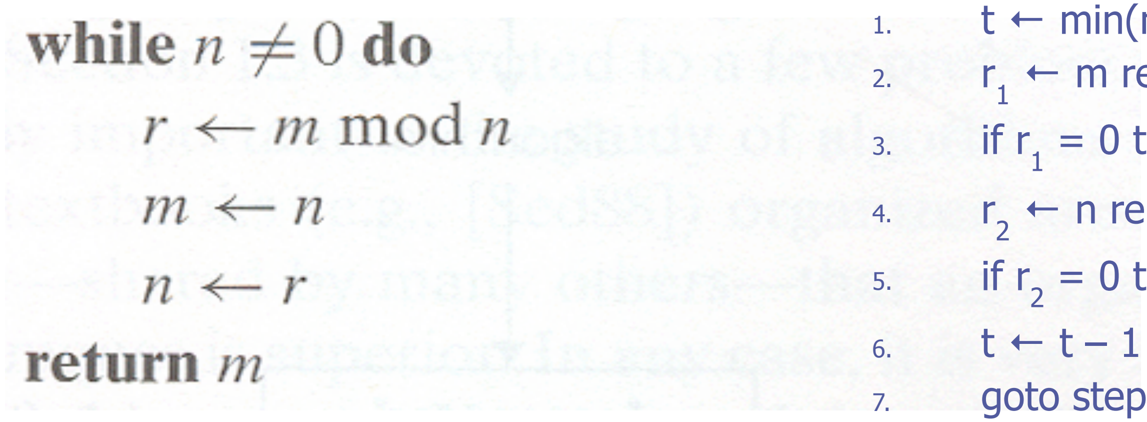
Consecutive

```
1. $t \leftarrow \min(m, n)$
2. $r_1 \leftarrow m$ rem $t$.
3. if $r_1 = 0$ then goto step 4 else goto step 6
4. $r_2 \leftarrow n$ rem $t$.
5. if $r_2 = 0$ then return $t$
6. $t \leftarrow t - 1$
7. goto step 2
```

Compare these two algorithms

- Work required: in general
  - A: Euclid's does more work
  - B: Consecutive does more work
  - C: both do the same amount of work

Euclid



```
while $n \neq 0$ do
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return $m$
```

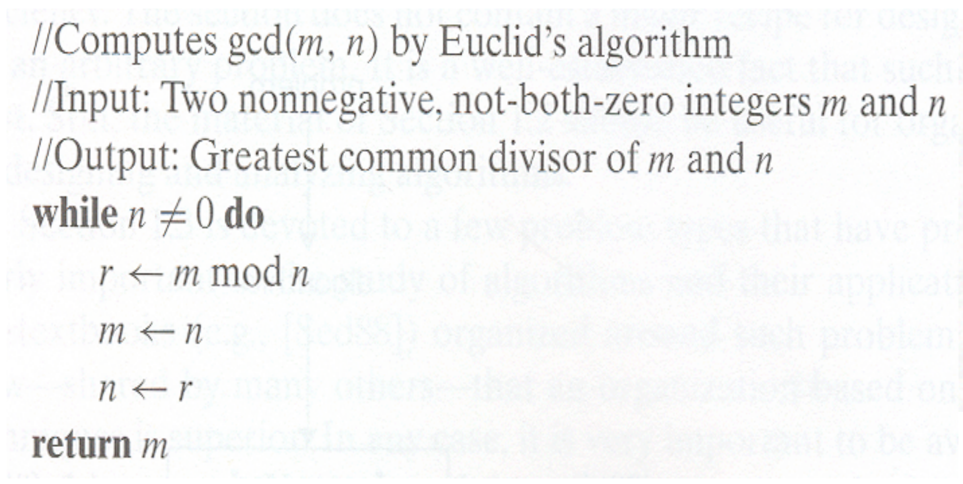
Consecutive

```
1. $t \leftarrow \min(m, n)$
2. $r_1 \leftarrow m \text{ rem } t$.
3. if $r_1 = 0$ then goto step 4 else goto step 6
4. $r_2 \leftarrow n \text{ rem } t$.
5. if $r_2 = 0$ then return $t$
6. $t \leftarrow t - 1$
7. goto step 2
```

Compare these two algorithms

- Will they terminate on all valid inputs?
  - A: Euclid's always terminates
  - B: Consecutive always terminates
  - C: both always terminate
  - D: both may run forever

Euclid



```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

Consecutive

```
1. t ← min(m, n)
2. r_1 ← m rem t.
3. if r_1 = 0 then goto step 4 else goto step 6
4. r_2 ← n rem t.
5. if r_2 = 0 then return t
6. t ← t – 1
7. goto step 2
```

Common Features:

Descriptions of algorithms often involve:

- Sequential and Parallel Execution;
- Conditionals and Decision Making;
- Looping and Repetition;
- Assumed Details (procedures);
- etc…

There are corresponding features in many programming languages.

How Good is an Algorithm?

Given a particular algorithm, we might ask:

- Does it solve the original problem?
- How fast is it?
- How much space does it require?
- Are there "better" ways to solve this problem?

First Approach: Experiment!

- Implement the algorithm;
- Run it on some test data.

But if we can’t test all possible inputs:

- We might miss cases where the algorithm fails;
- We might miss cases where performance is particularly good (or bad);
- Results will depend on implementation details.

Works well in some, but not all cases.

Second Approach: Analyze!

- Study the algorithm in general terms;
- Try to predict general trends in behavior:
  - Correctness: Does it do the “right thing”?
  - Performance: How long does it take? How much memory does it use?

Correctness:

For an informal proof of correctness of Euclid's algorithm, we begin with the equality:

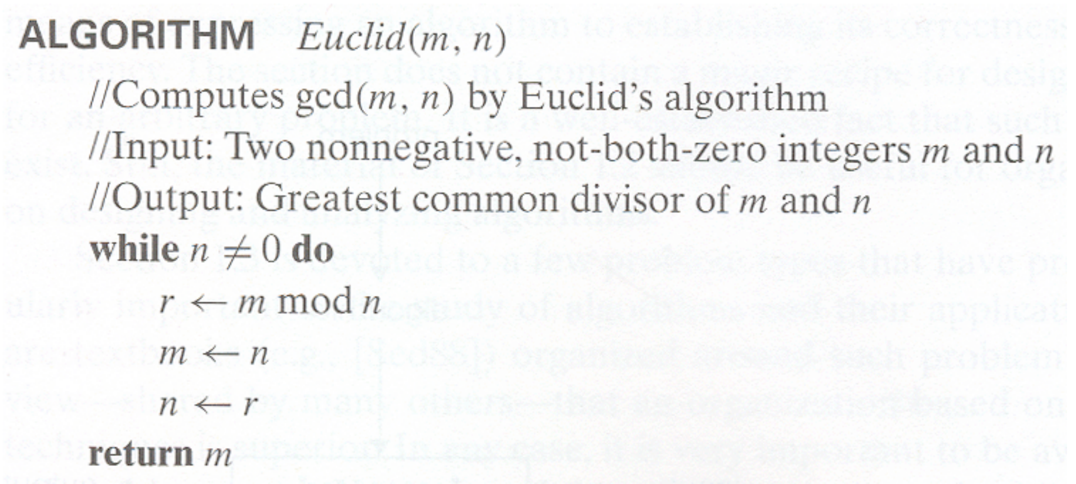
GCD(m, n) = GCD(n, m mod n)

Invariant: each time around the loop, GCD of the original two inputs = GCD(m, n)

Variant: each time around the loop, n gets smaller, but never becomes negative

Loop terminates when n = 0

GCD(m, 0) = m



```
ALGORITHM  Euclid(m, n)
    //Computes gcd(m, n) by Euclid's algorithm
    //Input: Two nonnegative, not-both-zero integers m and n
    //Output: Greatest common divisor of m and n
    while $n \neq 0$ do
        $r \leftarrow m \bmod n$
        $m \leftarrow n$
        $n \leftarrow r$
    return m
```

Performance:

Execution time depends on:

- the size of the input data;
- the input data itself;
- the machine and implementation.

What exactly do we want to measure?

- A: CPU Cycles?
- B: Wall time?
- C: Number of machine instructions?
- D: Number of primitive operations?
- E: All of the above
- F: It doesn’t really matter

- Suppose we have a program that calculates f(x) for some function f, and input x.
- We could write a program $f^{Time}(x)$ that calculates the time taken to execute f(x); this might be done by instrumenting the original code.
- In effect, we need to run the program for each given input, and measure how long it takes.
- In most cases, this is too much information to be useful!

- Solution: Abstract away from the specifics of particular input data.
- We can get useful information from a function

  T(n) = time taken to process an input of “size” n
- It’s up to us to decide what would be a good measure of “size”;
- We can make do with approximations instead of precise timings.

Summary:

- The concept of an algorithm
- Dimensions of algorithm analysis
  - Correctness
  - Efficiency
- Approximating efficiency is OK