eecs 281 DATA STRUCTURES AND ALGORITHMS

Lecture 1: Introduction

What 281 Covers

Algorithmic Thinking: "Conceptualizes problems with digital representations and seeks algorithms that express or find solutions" – *P.J. Denning*

Solutions are not simple or even known

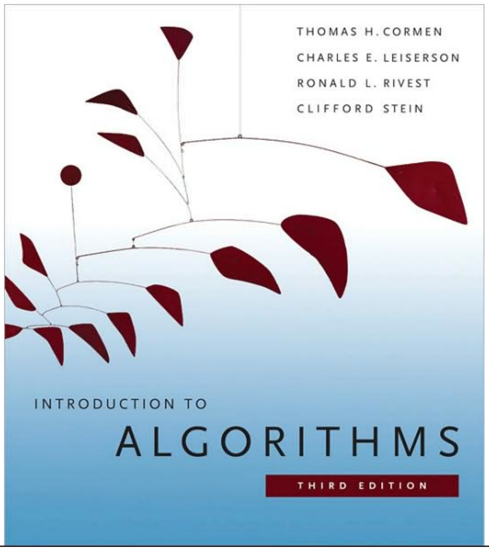
Require efficient algorithms for large datasets

- data must be wisely stored (efficient data structures)
- data must be quickly processed (fast algorithms)

Can be broken down into two fundamental topics:

1. Abstraction and Representations
2. Algorithmic Patterns and Performance Analysis

Required Reading



Prerequisites:

- EECS 280
- EECS 203
- C++

Recommended Readings

www.cplusplus.com/doc/tutorial/
www.cplusplus.com/reference/
http://www.cprogramming.com/ (but too much ads)
http://www.codeguru.com/Cpp/

Other books:

- Stroustrup, *C++ Programming Language*, latest ed.
- Lippman, Lajoie, and Moo, *C++ Primer*, latest ed., gentler and more verbose than Stroustrup
- Josuttis, *The C++ Standard Library*, latest ed.

Two Parts of 281

Instructor for first half: Sugih Jamin
email: jamin@eecs.umich.edu
Office: 4737 CSE
Office hours: TT right after lecture, Fri 9:00-10:00 and by appt.

GSIs/IAs:
Max Corman (cormamax), Gaurav Kulkarni (kgaurav), Rong Ling (lingrong), Peter Terlep (pterlep), Steve Wishnousky (swish)

GSIs/IAs

Discussion sessions:

W	3:30-4:30	1500 EECS	Rong Ling
W	4:30-5:30	1006 DOW	Pete
Th	5:00-6:00	1010 DOW	Pete
F	1:30-2:30	3150 DOW	Max
F	2:30-3:30	1005 DOW	Gaurav & Steve
F	3:30-4:30	1005 DOW	Rong Ling

Office hours (held in 1695 CSE)

Steve: M 2-3
Pete: M 3-4, W 1-2
Rong Ling: M 4:30-5:30, W 4:30-5:30
Gaurav: F 3:30-4:30
Max: Tu 3:30-4:30, Th 3:30-4:30

Two Parts of 281

Instructor for second half: Prof. David Chesney

GSIs/IAs stay the same

Best to think of the two parts of 281 as two separate courses, each with its own course policies and grading criteria

Most of what follows apply only to the first half of 281

Course Web Site

http://www.eecs.umich.edu/~sugih/courses/eecs281/
and on ctools

Most lecture slides will be posted on web site before lecture, but some will be updated *after* lecture
- always grab a fresh copy if you need to consult a lecture note
- don't print it out, save a tree

Course Announcements

Announcement page on ctools

Both course web site and ctools site (linked to each other) are "required readings"

Email both your GSI/IA and myself if you have any questions

We will post FAQ's on the Announcements page, check it first before asking your questions

Grading Policy (50% of Course Grade)

• 1 Midterm Exam: 13%	Thu, 10/20, 6-8 pm
• 2 Homeworks: 10%	Hand in hardcopies
• 2 Programming Assignments: 26%	Turn in online
• Class Participation: 1%	

Do not email us any of your assignments!

Scheduling Conflicts

Only documented medical or personal emergency allowed

If you need extra time to complete an exam due to personal disability, please inform us 1 week in advance

Other scheduling conflicts will not be considered two weeks after the start of the term (today)

Outside commitments, e.g., job interviews, top-coder contests, are not considered valid reasons for missing a deadline or exam

Grading Policy

Regrade:
- within 5 working days
- written request
- whole work will be regraded

Late days:
- 2 free late days in total for programming assignments 1 and 2
  - including weekends
  - NOT per assignment
- no need to inform us to use any of your free late days
- keep track of your own free late day usage

Help with PAs stops 2 days before due date

Late Penalty

Applied to PAs *after* free late days are used up
Applied to HW1 immediately
HW2 will not be accepted late (solution handed out same day)

Penalty schedule:

- $\leq$ 24 hours: 4%
- $\leq$ 48 hours: 8+4=12%
- $\leq$ 72 hours: 12+12=24%
- $\leq$ 96 hours: 16+24=40%
- $\leq$ 120 hours: 20+40=60%
- work more than 5 days late will not be accepted

Example:

- PA worths 100 points, work late by 24 hours and 10 mins
  - if no free late days left: 12 points late penalty
  - if 1 free late day left: 8 points late penalty
- turning in HWs after lecture has started is considered one day late

Collaboration

- All work must be done individually
- Cheating and plagiarizing are not tolerated
- To pass off the implementation of an algorithm as that of another is also considered cheating:
  - e.g., insertion sort is *not* heap sort
  - if you can not implement a required algorithm, *you must inform the teaching staff when turning in your assignment*, e.g., have the program print out an error msg
- We take the Honor Code seriously!
- There is a cost to being honest, but it's less than the cost of being caught dishonest

Collaboration

HWs 1 and 2: consultation of online and offline sources allowed, but must not be copied verbatim, you need to show that you have understood the material

- Cite your sources, including classmates and roommates, but not teaching staff or required readings

Midterm exam: open-book, open-notes, see course Grading Policy web page (under Resources on ctools)

Programming Assignments (PAs) 1 & 2

No group project, all work must be done individually

No STL

Must not include external materials (e.g., open-source code downloads or code from others (this or previous terms) or found online)

The autograder platform is Linux, with g++ 4.1.2

- develop and test locally
- compile with g++ remotely
- then submit
- no other platform supported

PAs 1 & 2 Grading Criteria

Code compiles

Code runs correctly

Code is readable, well-documented, e.g.,

- no use of literals
- code re-use instead of cut-and-paste

Algorithm is efficient

Implementation is efficient, e.g.,

- no unnecessary copying
- no loop invariant statement in loop

Code Reuse Simple Example

```
const int C=10, K=1;
int b, c;

int f()
{
  if (x < C) {
    a = x-2;
    b = a+K;
    c = (x%2)+b;
    d = b/2+c*2;
  } else {
    a = x+1;
    b = a+K;
    c = (x%2)+b;
    d = b*2+c/2;
  }

  return(d);
}
```

```
void g(a, x)
{
  b = a+K;
  c = (x%2)+b;
}

int h()
{
  if (x < C) {
    a = x-2;
    g(a, x);
    d = b/2+c*2;
  } else {
    a = x+1;
    g(a, x);
    d = b*2+c/2;
  }

  return(d);
}
```

Tips For Success

Start homework and projects early

If in doubt, make a small program and test

Experiment with code and learn from mistakes

Utilize software development tools

Standard Tools

- Compiler+Linker
- Make utility (Makefile)
- Debugger
- Profiler

Personal Preference

- Text editor
- IDE (Integrated Development Environment)
- Source code version control
- Visualization software

How to Write Clear Code Quickly

Know C++ syntax

- study it (or look it up if you've forgotten)

Develop algorithm first, check it, implement later

- trying 1-2 variants to see what works is OK with syntax, but not with algorithms
- bad idea: tweak the algorithm in the hope that it will just work at some point

Work on your skills (practice, practice, practice)

Study good code examples

Avoiding Errors

You will spend more time testing & debugging than coding

⇒ don't wait until the last two days before a deadline to start coding! or testing!

Reuse code as much as possible
- reuse will reduce the amount of code to debug
- design for reuse

Backup often
- new features & bug fixes may introduce new bugs
- use a source code version control system such as cvs, svn, or git (but careful not to make your code public!)

Finding Errors - Testing

Compile often (`g++ -Wall -Werror`)
- try not to type > 20 lines of code without compiling
- compiling often narrows down source of bugs

Generate a wide variety of test cases
- no single test will catch all errors
- most errors are only triggered by some test cases

Begin testing as early as possible
- test each function/class separately if possible (unit test)
- Fail Early, Fail Often

The Autograder

Your code is required to be 100% correct
- a 100% pass rate from the autograder does not mean your code is 100% correct and does not guarantee 100% grade on the assignment

The autograder tests only a few test cases
- you are expected to generate more test cases on your own
- your code may contain bugs not caught by the autograder
  - bugs range from poor algorithm design to careless typos
  - grades needlessly suffer from lack of testing ("last-day coding")

The autograder is NOT a debugger, don't use it as one
- you will slow down the system for everyone
- you may crash the system
- the autograder won't be available until a couple of days before a deadline

What 281 Covers

Algorithmic Thinking: "Conceptualizes problems with digital representations and seeks algorithms that express or find solutions" – *P.J. Denning*

Can be broken down into two fundamental topics:
1. Abstraction and Representations
2. Algorithmic Patterns and Performance Analysis

Algorithmic Patterns

What are algorithmic patterns?

- approaches to problems, ways of doing things, tips and tricks of the trade
- in the olden times, what you learn from guild apprenticeship

Performance Analysis and Tuning

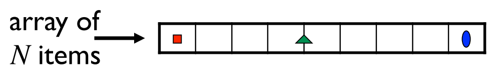
Performance analysis: how some tools are better for certain tasks than other tools

- how do you pick the right data structure?
- how do you know if an algorithm is efficient?
- how do you design an algorithm to be efficient?

"When you have a hammer every problem looks like a nail." Not

Performance tuning: how to improve the performance of the tools and solutions

Performance Analysis of Name Search



Best-Case: 1 step
Worst-Case: N steps
Average-Case: $N/2$ steps

Using a linear search over N items, how many steps will it take to find item x?

Best-Case 🟥
- data is found in the first place you look
- least number of steps required, given ideal input

Worst-Case 🔵
- data is found in the last place you could possibly look
- most number of steps required, given difficult input

Average-Case ▲
- average number of steps required, given any input
- average performed over all possible inputs of a given size

Execution Cost

Algorithms take resources to execute, such as:
- Space: memory, bandwidth
- Time
- Energy

Two types of resource cost:
- fixed cost
- variable cost

Space-Time Tradeoff

If you can load all your data into memory, you can sometimes come up with very fast algorithm

Examples of memory-constrained problem/system?

Timing Cost

Fixed cost (one-time cost):
- coding time
- compile time
- variable initializations, etc.

Variable cost:
- run time: depends on the size of the problem

Worst-case Runtimes of Our Name Search Problem

Assume can search 10 names/ms

Population (size)	Linear	Binary
EECS 281 (230)	23 ms	0.8 ms (lookup 8 names)
UM (40 000)	4 secs	1.5 ms
MI (9 mil)	15 mins	2.3 ms
Shanghai (23 mil)	38 mins	2.4 ms
US (311 mil)	8 hours 38 mins	2.8 ms
China (1.3 bil)	1 day 13 hours	3.0 ms
World (6.9 bil)	8 days	3.3 ms

There is usually more than one ways to solve a problem
Want: the most efficient way

Algorithm Design

Typical approach:
- define problem to be solved
- understand its complexity
- decompose it into smaller subtasks

Creative refinements:
- think outside the box: exploit any particular characteristics of the workload or problem
- don’t lose the forest for the tree:
  - find the real performance bottleneck of the whole program (how?)
  - also note the human labor cost
- optimize for the common case

Algorithm Performance Analysis

Given two algorithms, how do you determine which is more efficient?

Example: roofing problem

- a new house is being built in the village
- the builders are currently working on the roof
- there are three different methods for moving roof-tiles from the truck to the roof

Three Roofing Methods

Method 1:

A builder can carry two tiles on his shoulder as he climbs up the ladder. He then climbs down and carries two more tiles up the ladder. Each round trip (up and down the ladder) costs $2

Method 2:

The builder rents a lift for $10. The lift can move 20 tiles up the roof at one time, at a cost of $1 per round trip

Method 3:

The builder rents a super-lift for $40. Unfortunately, the lift has a slow leak in its hydraulic system. It is able to lift half of the necessary tiles to the roof on the first round-trip. However, on the second trip, it is only able to lift half of the remaining tiles, then half of the remaining . . . down to the minimum of one tile per round trip. Each round trip costs $2

In all three methods, it costs $4/tile to set

Back-of-the-Envelope "Analysis"

Which is the cheapest of the three methods to build a shed (8 tiles)?

Which is the cheapest for a gazebo (128 tiles)?

Which is the cheapest for a house (2048 tiles)?