1. What this course offers
2. How the course will deliver
3. Course plan

Invitation

• This course explores, but does not finally answer, a question about computer performance solving hard algorithmic problems

• *The mystery*: Do many problems we consider hard have solutions that take reasonable amounts of time?

• Or do *all* solutions to *all* these problems take too much time to be worth using?

• *Example*: visiting a set of cities, each once
**Invitation**

*We will also look at*

- ways to design computational procedures
- ways to prove that an algorithm works correctly for all inputs
- ways to precisely characterize the running time of an algorithm
- Ways to design for performance

**Inquiry**

- How useful is the study of computational *problems*?
- Does it pay to *prove* claims about the behavior of systems?
- Is *time scalability* a good way to measure the performance of computing systems?
- How much does studying *design approaches* help us create efficient solutions?
- How hard is *optimization*?
Inquiry

• How do you know what you know?

Inquiry: CS and mathematics

• What is computer science about?
• What is math about?
• In your experience, what do they have to do with each other?
• What mathematical concepts must you, as a computer-science major or a computing professional, be able to apply?
Who this course comes from

Kurt Gödel (1906-1978) established a relation between computation and recursively definable functions

Stephen Cook (L) and Leonid Levin (R) defined “difficult” problems and described one (1971)

1. What this course offers

• Practical software performance skills
• Self-confidence about quantitative problem solving
• A review of the software structures and algorithms introduced in Data Structures
• An engineer’s attitude toward verification
• A chance to have insights that don’t depend on the latest product or language
Why study algorithms?

- Specification of problems is precise if expressed mathematically
- Algorithms compute *recursively definable* mathematical functions
- Saying how *fast* an algorithm is or how hard a problem is, can be expressed precisely
- We can prove correctness of our designs and our time analysis of an algorithm

Lofty goal: software that works

- *Physics* gives theory for structural engineering
- *Bridges* built without application of physics … *collapse*

- *Mathematics* provides theory for software development
- Software built without application of relevant math … *collapses*
**Forms of network attack**

- Theft of information
- Destruction of data
- Seizing control of computer for use in attacks on other computers (zombies, botnets, distributed denial of service)

**Do verification, analysis, and modeling matter for software development?**

- **Process** is central to software engineering:
  - Version control
  - Design patterns
  - Refactoring
  - Teamwork
  - QA by testing
- So is **mathematics**:
  - Quality assurance by use of logic and inference
  - Performance analysis using big-O
  - State-based and transition-system modeling
Why theory matters

- Every field of study, including computer science, has basic assumptions and concepts
- Example: the concepts of computing in CS, society in sociology, value in economics, etc.
- In CS, we assume that people want to reliably transform numbers and symbols
- These assumptions and concepts are the theoretical foundation of the field of study

The origin of “algorithm”

- Arabic mathematics (based in part on Hindu math) used place values, more efficient than Roman numerals
- Mohammed ibn Musa Al-Kuwarizmi published the text Al-Jabr in the tenth century
- It used algorithms to calculate arithmetic
- Hence the names “algorithm” and “algebra”
**Algorithm:**
A precise plan to solve a *functional* problem in a finite number of steps

- Program designs use algorithms that compute functions from pre-specified *input* to *output*
- *function*: “mapping,” not “subprogram”
- Java *batch* programs execute algorithms
- Interactive software executes algorithms over and over

---

**Algorithms compute functions**

\[ x \xrightarrow{f} y \]

- Methods, procedures, and some high-level-language subprograms *implement* algorithms
- We distinguish functions from the algorithms that *compute* them
- Some functions are *uncomputable*
- All computable functions are *recursively definable*
Defining “algorithm”

• An algorithmic procedure is equivalent to
  – A run of a C++ or Java program with all input specified at beginning and all output at end
  – Computation of a Turing machine that always halts
  – Computation of a Random Access Machine
  – Evaluation of a μ-recursive function

• Not algorithms:
  – “Online algorithms” (non-functionality)
  – “Probabilistic algorithms” (nondeterminism)
  – Event loop (interleaved input/output)

Ways of thinking about algorithms

• Correctness of algorithms may be proven
• Performance of algorithms may be precisely characterized
• Problem complexity may be precisely characterized
• Computing is more than algorithms; it includes interaction
Scope of course

- We look mostly at *algorithmic* problems, problems of computing a *function* from input to output
- *Exceptions:*
  - *Problem specification* (Topic 1) includes *interactive* processes
  - Parallel and distributed computation (T7) uses *concurrent* processes that may interact via shared memory
- *Interaction* (T7) is in contrast to algorithms

Interactive computation

- Feature of computing today: Computation as an ongoing *service*, not assumed to terminate
- Must solve problems whose inputs cannot be completely specified a priori
- Dynamic input and output *during* computation
- *Persistence of state* between interaction steps
- *Environment* is an active partner in computation
Course focus

- Methods of design, verification, and analysis are our focus, not particular examples of algorithms and problems.
- We go from simple to complex design, and from tractable to intractable problems.
- The problems and algorithms most difficult to analyze are those involving interaction, especially multi-agent interaction.

CSCI 347 in the CS-G concentration

- This course satisfies ABET accreditation requirements for mathematics content and is part of the theoretical component of the software-development-oriented CS-G concentration.
- It meets standards of the ACM/IEEE CS2013 curriculum.
- It will help you design systems that meet time performance requirements.
How Analysis of Algorithms fits into the CS-G concentration

Proposed goals of this course section

• For all students in this section …
• to reach a quality level of skills, grasp of concepts, and demonstration of capabilities
• To discuss what you will need to know as a software professional
2. Course plan

- **Organization** of topics 3-7 is by *algorithm* type: brute-force, divide-and-conquer, greedy, transform-and-conquer, dynamic-programming, approximation, probabilistic, parallel-distributed
- Expanded set of topics includes:
  - Discussion of problem classes – T1
  - Formal verification – T2
  - Problem complexity (intractability) – T6
  - Parallel and distributed algorithms – T7
  - Interaction – T7

Topics

1. Classes of problems
2. Formal verification, including recurrences
3. Big-O notation, algorithm analysis, and brute-force algorithms
4. Divide-and-conquer, decrease-and-conquer
5. Greedy and other fast optimization algorithms
6. Intractable problems; computational complexity; approximation and random algorithms
7. Parallel and distributed algorithms; interactive processes; multi-stream interaction
Some topic objectives

1. Describe, categorize, and formally specify a variety of computational problems.
2. Explain and apply the method of proving correctness of an algorithm inductively, using postcondition and loop invariant.
3. Use recurrences and asymptotic complexity notation to analyze a variety of array-traversal and brute-force algorithms.

More topic objectives

4. Explain the divide-and-conquer approach and apply it to a variety of algorithm-design problems, giving the time analysis for each.
5. Explain and apply the greedy method of algorithm design.
6. Describe the notion of intractability mathematically, including complexity classes, and ways to overcome this obstacle.
7. Describe design and performance issues in parallel, distributed, and interactive computation.
Multi-topic objectives

0a. Participate in class activities throughout the semester
0b. Solve problems as part of a team
0c. Present results in the classroom
0d. Present written results
0e. Show knowledge of facts and concepts
0f. Summarize the semester’s learning
0g. Relate mathematical concepts to applications

Topic 1: Classes of problems

• String manipulation
• Searching, sorting
• Graph (search, shortest path, spanning tree)
• Constraint vs. optimization
• Geometric (convex hull, closest pair)
• Satisfiability of logical formulas
• Sequential-interactive problems (services)
• Multi-stream-interactive problems (missions)
Topic 2: Formal verification

- **Tools:**
  - Preconditions
  - Postconditions
  - Loop invariants
- **Goal:** To show that given a set of preconditions, a particular algorithm satisfies a postcondition (spec)
- *Backtracking* for rigorous proof
- *Temporal logic* helps verify interactive systems

```
Largest(A)
largest ← A[1]
for i ← 2 to |A| do
  if A[i] > largest
    largest ← A[i]
return largest
```

Topic 3: Big-O notation, algorithm analysis, and brute force

- Derive inductive *time* functions from inductive definitions of functions computed
- *Asymptotic analysis* abstracts from physical installation details and additive or multiplicative constants
- Example of a recurrence:
  
  \[
  \text{sum}(a, b) = \begin{cases} 
    a & \text{if } b = 0 \\
    1 + \text{sum}(a, b - 1) & \text{otherwise}
  \end{cases}
  \]
Topic 4: Divide and conquer

- We “conquer” by defining the recursive case as a simpler version of the original problem
- We solve problems that correspond to trees
- Examples:
  - Binary search
  - Binary-search-tree search, insertion
  - Quicksort
  - Heap algorithms

Topic 5: Greedy and other efficient optimization algorithms

- Greedy algorithms for optimization make use of a property that a solution’s components are also solutions to subproblems
- We may use space for faster algorithms
- Tables or larger tree nodes may be used to improve running times of some algorithms
- Transform-and-conquer: Simplify instance, change representation, or reduce problem
**Topic 6: Intractable problems and approximate solutions**

- Complexity of a *problem* is complexity of the most efficient algorithm that solves it
- We prove complexity by *reducing* a problem to another
- Problems may be categorized as *tractable* ($P$, polynomial-time) or *intractable* (NP-hard)
- Some algorithms can *approximate* an optimal solution very closely, sometimes using randomization

**Topic 7: Parallel and distributed algorithms; interaction**

- Sequential computing vs. concurrency
- Parallel algorithms and models
- Time, work, efficiency, speedup
- Parallel architectures
- Parallel vs. distributed computing
- Synchrony, asynchrony
- Communications complexity
- Scalability of concurrent systems
3. How the course will deliver

• What are your expectations?
• Inquiry before grades
• Classroom format
  – presentations
  – discussion
  – teamwork
• The growth mindset
• Assessment of learning

This course is an inquiry

• *FSU motto*: “Live to the truth”
• What does this mean?
• What is the truth?
• What does *truth* mean in software development and computer science?
• *Possible ways to the truth*:
  – Authority of experts
  – Our own experience, including experiments
  – Discussion
This course is about using professional tools

• **Program goal:** To prepare you for professional work solving software-development problems
• Knowledge is only one tool in problem solving
• **Others:**
  – Teamwork
  – Design
  – Coding
  – Testing and verification
  – Predicting evolution of a system
  – Ethical concerns

Course organization

• We have seven *topics* plus *introduction*
• Each topic has an *objective* and 3-5 *subtopics*
• Each subtopic has 1-3 *objectives*
• Some objectives are called *essential* or *priority*; others, *challenge*
• A set of *problems* for each subtopic objective is available
• Exercises and quizzes draw from these
Classroom format

• Emphasis is on *inquiry, activity,* and *collaboration*

• *Slides and short presentations* summarize the content of the course

• We seek to create a *natural critical learning environment*

• *Your contribution and participation* are essential

Classroom discussion

• *Goal: everyone* participates without exception

• *Methods:*
  – All agree to prepare for class
  – I call on students at random with open-ended questions

• *Guarantee: all answers are appreciated*

• What are pros and cons of this method, for you?

• Can we agree to use it?
Guidelines

• Facilitated democracy based on syllabus
• Full academic freedom
• Mutual respect and support
• Staying on topic
• Time limits on speaking
• One conversation
• Phone-free classroom
• Nondisruptive coming and going OK

Group work in classroom

• Form groups of 3 students to take up a problem
• Each group should have
  – Facilitator – keeps discussion on track
  – Recorder – writes results of discussion
  – Reporter – presents results to class
• Participation by all in group work is one of our basic objectives in this course
Attendance, participation, and preparation

- Please prepare for class discussion; e.g., by solving an exercise problem
- Bring evidence of preparation to each class
- Some students in each class session will show problem solutions

A possible classroom session

6:30  Slides and discussion of 1-2 subtopics
7:00  Exercises / group work
7:30  Students work at board
8:00  Slides and discussion of 1-2 subtopics
8:20  Exercises / group work
8:40  Students work at board
9:00  Turn in exercises
      Quiz time
Exercises, group work, and presentations

- *Exercises* consist of individual and group problem solving
- Each student solves problems from as many essential and priority objectives as possible; I check off work received
- Each student prepares to *lead a group* to find and present solutions for some topics
- Exercises give you practice with quiz questions

Semester project

- Brings together work done in several assigned exercises
- Summarizes your understanding of *design approaches* and *time analysis*
- Invites you to solve one or two problems in different ways
- An essential part of the course
**Due dates**

- Exercises are due on the last day of a topic
- We’ll have a quiz then too
- Week 7 is deadline for mid-semester assessment of all work
- Submit projects at mid-semester and near end; I give feedback so students may fill gaps and make corrections
- *You and your life* organize your time
- Half credit for very late exercises

**Assessing objectives in class**

- After doing exercises on a topic, a student may show attainment of an objective/objective by solving a quiz problem, in writing, in class
- More opportunities will be available for each objective
- The main factor in success is attaining objectives and objectives
- Another factor: contribution to everyone’s learning
Breakdown of objectives

<table>
<thead>
<tr>
<th>Topic</th>
<th>Essential**</th>
<th>Priority*</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>exer†</td>
<td>exer†</td>
<td></td>
</tr>
<tr>
<td>backgd</td>
<td>27</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>35</strong></td>
<td><strong>4</strong></td>
<td><strong>29</strong></td>
</tr>
</tbody>
</table>

*Suggested goal:* attain at least 5 essential or priority objectives per week; 10-15/wk in first 2 weeks

Six-week tally

- Master topics Intro to 3 by mid-semester
- Your ability to contribute will depend on that
- So will success on topics 3 to 7
- Mid-semester assessment:
  - Scores for objectives
  - Exercises and project work
  - Solutions presented in class
  - Documented group work
  - Preparation
Summary quizzes and final-exam day

• During the last week of classes, we’ll have a summary quiz of multiple-choice questions and multi-topic problems
• On final exam day, students will present elements of their semester projects

What is a successful solution?

• Reflects understanding of
  – question or problem
  – relevant concepts
  – relevant procedures
• Usually reflects study and work on an exercise
Challenge: expression evaluator

- I would like teams to develop an expression evaluator for propositional-logic formulas
- **Input:** formula plus a set of variable assignments
- **Example:** \((p \land (q \lor \neg r)), (t, f, t)\)
- **Output:** true or false
- I would like you to use this tool to investigate the satisfiability problem to see if any solutions, or solutions to any subsets of this problem, run efficiently

Failing quizzes is undefined

- I score questions, not quizzes
- Overall scores can only improve as you learn more and try again
- Leaving a question blank or “I don’t know” often shows discernment
- It is normal to be not yet successful when learning
**Background pretest**

- This pre-quiz, in 10 parts, will assess some knowledge from Finite/Discrete Math and from Data Structures
- Each pre-test question prepares you for one or more topics of this course
- Many of these objectives are considered essential for CSCI 347
- There will be multiple chances on the objectives assessed

**Assessment and grading**

- *To measure:*
  - Individual achievement of learning objectives
  - Contribution to the learning of the class
- *Breakdown: 60/40*
- *Assumptions: Learning is shared and measurable*
Assessment of learning objectives

Assumptions:
• Application of concepts is measurable via core and other topic objectives
• Facts about concepts matter
• We learn by summarizing and reflecting

Assessment of contribution and participation

We assume that learning happens by:
• Inquiring and sharing inquiry
• Being present
• Solving problems together
• Activity throughout the semester
Hidden curiosity and talent

• You were born curious; it’s in your nature
• Schooling can suppress curiosity
• One option is to allow our curiosity to re-emerge as part of our true selves
• For me, this can enable effortless effort and helps me to be present with what I study

The growth mindset

• Research tells us:
  – People can learn new skills when they believe that their effort matters
  – Learning takes effort
  – Intelligence can grow with effort
• Alternative mindset: fixed
• The fixed mindset says that innate talent, not effort, is decisive and changeless
Quality and learning

- People enjoy doing quality work
- This requires freedom of choice and control of work environment
- Coercion and boredom may discourage quality work
- In school, part of learning is recognition of quality work
- Please evaluate your work and get your friends’ evaluations!

Who can judge quality?

- Everyone likes quality
- You can be a good judge if you know what to look for
- Other students may help you to see quality
- Professors may help you to see quality
- Research predicts that if we know quality, we will produce it
Grading weights

<table>
<thead>
<tr>
<th>Objective</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Essential objectives</td>
<td>30 %</td>
</tr>
<tr>
<td>Priority objectives</td>
<td>20</td>
</tr>
<tr>
<td>Challenge objectives</td>
<td>10</td>
</tr>
<tr>
<td>Mid-semester tally of work</td>
<td>10</td>
</tr>
<tr>
<td>Summary quiz</td>
<td>10</td>
</tr>
<tr>
<td>Exercises</td>
<td>10</td>
</tr>
<tr>
<td>Semester project</td>
<td>10</td>
</tr>
<tr>
<td>Blackboard work</td>
<td>10</td>
</tr>
<tr>
<td>Documented group work</td>
<td>5</td>
</tr>
<tr>
<td>Attendance and preparation</td>
<td>5</td>
</tr>
</tbody>
</table>

All, except challenge objectives, are required to complete the course

Grades, learning, and effort

- Learning requires *curiosity, intention to learn*, and *undistracted effort*
- Attention to grades distracts from what we’re learning
- If grades measure learning, then:
  - *Getting higher grades requires not paying too much attention to grades!*
Which matters more: grades or what you learn?

- Grades are opinions
- Opinions are fallible
- Course content could change our lives
- Questions:
  - Whose opinion defines what’s true?
  - When is it right to challenge a student?
  - When is it right to challenge a teacher?

Academic integrity

- Directly lifted text must be quoted and credited
- Use of ideas or other information must be credited by citations or references
- Citation standards for MLA and APA are given at [www.citationmachine.net](http://www.citationmachine.net)
- Plagiarism: “occurs when you use someone else’s ideas or words and represent them as your own.”
- See catalog for FSU policy
A proposed agreement

I commit to:
• know the course material, present it clearly
• return submitted work within a week.
• answer questions helpfully

Students commit to:
• ask questions
• answer reasonable questions, risking error
• submit work on time, even if incomplete
• work sometimes in groups
• present results or lead discussions

References