CSCI 347 Analysis of Algorithms
Prof. David Keil, Framingham State University, Fall 2015
SYLLABUS

Invitation
If you want to build software that performs well, this course will provide some tools for you. We’re mainly interested in time performance, but specification and verification matter to us too.

The course explores, but does not solve, a mystery about computer software performance. It concerns the kinds of problems we solve in daily life, and the kinds of problems artificial intelligence tries to solve—hard problems such as planning, or guessing what a business competitor will do. These problems have the mathematical name “NP.”

The mystery is this: Do all these hard problems have solutions that take reasonable amounts of time, not much greater than the time needed to check whether a particular possible solution is an actual one? Or do all of them require solutions that would take too much time to be worth using?

Along the way, we will look at ways to design algorithms, to prove that an algorithm works all the time, and to precisely characterize its running time.

Course description
A presentation of asymptotic time and space complexity of sequential and parallel algorithms, using big-O and related notation. Complexity classes P and NP; tractable and intractable problems; and verification of algorithms by formal methods are also discussed.

Prerequisites: CSCI 271 Data Structures;
MAT 292 Discrete Math I or MAT 215 Finite Math

Meeting times and office hours
We meet Tuesdays, 6:30-9:50 pm, in Hemenway Hall 101A.

Office hours (Hemenway Hall 418A):
Mon. 10:30-11:30 a.m., Tue. 4:30-6:30 p.m.
Others by appointment—I’m usually in office

Telephone: (508) 626-4724
Email: dkeil@framingham.edu
URL: www.framingham.edu/~dkeil/alg-mats.htm

I’ll schedule meetings with each student; I’d like to meet with you briefly at least three times in the semester to check in. I welcome questions by email and will schedule some times when I’ll be available to talk with students over lunch in the commuter caf.

Course overview
This course offers an opportunity to think in new ways about computational problems and the performance of algorithms and processes.

We use concepts that have been presented in your Finite or Discrete Math and Data Structures courses. Many of the algorithms we will discuss you’ve studied: sorting, searching, tree manipulation, hashing. The course material uses pseudocode.

The notion of mathematical proof is central to this course, as a way to obtain results in algorithm analysis and correctness. We will use the mathematical notion of recurrences to describe the run times of algorithms based on the inductive (recursive) definitions of the functions computed by the algorithms.

Half or more of the course (Topics 3-6) will cover different approaches to algorithm design, such as brute force, divide and conquer, and greedy algorithms.

In Topic 1, we present algorithmic problems that have solutions of different efficiencies, which will be analyzed in later topics.

An extension to the study of algorithm efficiency will be discussion of the computational complexity of problems. Problems whose best solutions have unacceptable running times are called intractable.

A special aspect of this course will be coverage of parallel and distributed algorithms and processes.

We will contrast algorithmic computing with interactive computing and will distinguish the different types of associated problems. I will argue for a paradigm shift in how we think about computing.

An appropriate description of the scope of the course as presented is: the design, verification, and performance analysis of algorithms and interactive processes.

Textbooks and other reading
Please use the slides and other handouts for each topic as your basic reference. Other reading, ordered from high to low priority:


I like these books because they explain in detail what we’re studying and give lots of examples. Reading text material related to course topics is vital.
Learning objectives

What the course offers is summarized by seven topic-level learning objectives, and several dozen subtopic-level objectives; see below. Some objectives, indicated by double asterisks, are essential; all students who pass the course will have shown success with these.

Other objectives I call “priority” and they are indicated by single asterisks. A third category of objectives I label “challenge.” They are optional, for extra credit. Achieving them will help you in later courses and in life.

We review and assess course objectives called “background” in the first two weeks of the course. They correspond to some of what is learned in Data Structures and Finite Math/Discrete Math.

Objectives are assessed with problems, multiple-choice quizzes, and programming exercises.

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 array-traversal and other brute-force algorithms.
4. Explain the divide-and-conquer approach and apply it to a variety of algorithm-design problems, giving the appropriate 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.

See last page of syllabus for subtopic objectives.

How the course will deliver what it offers

Agenda for class sessions

For each of the seven topics, we’ll have two to four two-hour sessions. I’ll speak about the topic for a few minutes, with slides and with problems. Students will present solutions and work in groups.

At the beginning of each class session, I ask all students to bring work they have done to prepare for the session. The best work to bring is solutions to exercise problems. This can include reflections on reading of slides or textbook chapters. I’ll track what you bring.

On the last day of each topic, I’ll collect whatever exercises students haven’t turned in, and we’ll review the topic, with problem solving by students. Quizzes will assess the topic’s learning objectives. Make-up questions on the previous two topics will be available.

How I assess learning

I use problems, several per course objective, to see what you are learning. I don’t track quiz scores, but rather best scores attained achieving objectives.

Knowing how I scored your solution, and reading my comments, can tell you how you are doing on an objective and what you can do to improve your work.

Use the back page of this syllabus as a score card, writing on it the numbers (from 1 to 6) that I write beside your quiz answers. Update these numbers as you answer questions on make-up versions of the quizzes; highest score on an objective is the one that matters.

Quizzes

My quizzes use problems and multiple-choice questions to test learning objectives; I don’t track quiz grades, but scores for objectives. This may be a new experience for you. Second chances will help you.

The best time to show attainment on an objective is right after we have discussed the relevant topic. But we all need second or third tries sometimes. For that reason several versions of quizzes will be available at different times. This is not an invitation to leave work on topics until “later.” Scores at mid-semester will matter.

Doing exercises is a precondition for taking quizzes. Because I give multiple opportunities for quizzes, I don’t value guessed answers on quizzes.

With make-up quizzes, I track the best score by objective (problem). Multiple-choice make-ups use 5-10 questions to assess an objective.

Each version of a quiz will be offered once. If you take a quiz out of the classroom and turn it in later, then your solutions will count as exercises but not toward achievement of objectives.
All quizzes are closed-book, in-class, and to be done alone without devices or any communication. The closed-book rule is necessary because all questions are available publicly. Cell phones and laptops are to be in containers during quizzes.

Problems are scored as below.

<table>
<thead>
<tr>
<th>Score</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Solution of rare quality exceeding requirements.</td>
</tr>
<tr>
<td>5</td>
<td>Solves problem thoroughly and accurately; applies relevant concepts adeptly, showing mastery of objective.</td>
</tr>
<tr>
<td>4</td>
<td>Mostly successful solution, applying concepts well. Strong success with objective.</td>
</tr>
<tr>
<td>3</td>
<td>A fair-quality solution with omissions or errors. Generally successful application of concepts.</td>
</tr>
<tr>
<td>2</td>
<td>Shows some grasp of relevant concepts, meeting minimum standards for the objective.</td>
</tr>
<tr>
<td>1</td>
<td>Unsuccessful answer that shows some understanding of problem or relevant concepts.</td>
</tr>
<tr>
<td>0</td>
<td>No answer or irrelevant answer</td>
</tr>
</tbody>
</table>

**Exercises**

Exercises to be done out of class consist of sets of problems referred to in a sheet for each topic, listed in a section for that topic in a file named “Study Questions”.

Some objectives are assessed by exercises. Those objectives will also provide questions for certain quizzes.

Some exercises are your reflections on class sessions, reading, and group work. We know that reflecting on what we’re learning is an important part of learning.

The best way to achieve an objective is to solve at least one problem for that objective, outside class. Then show your solution to a member of your group and get any help you need with it. Then take the quiz that has a problem in it that assesses the objective.

We need each student’s contributions in this class, so exercises are to be done while we discuss each topic. Exercises turned in more than two weeks late will receive half credit.

**Group work and presentations**

Every student will be in a group of about three students. We’ll have groups work on very short (multiple-choice) problems and in longer problem-solving sessions. Every professional works on a team, so one of our core program objectives in Computer Science is for every student to participate in group problem-solving activities.

Each group will prepare one of its members to present a solution to a problem on the objective. Each group member will document that student’s group work. To document a group session, briefly describe what happens and who did what.

Every professional also describes proposed solutions to problems, product designs, and so forth. For that reason all students in this course will present several solutions to problems in class, including a longer presentation of the semester project.

**Semester project**

Step by step, as the course proceeds, you will build a two-part Java programming project that will make use of control structures, debugging, class design, arrays, and file I/O. It will provide experience in coding, testing, and documentation of specifications, design, and code.

The first part is a file-maintenance program that manages a collection that you will specify. The second part is either to design and code your own application, or to modify code written by others; a still larger project with documented group contribution is an option.

I evaluate individual work even if part of a project is done by a group. Submit your own version of the project, circling your name. Label your work with your name, giving credit to others as needed.

Any research must be referenced.

**Two half-semesters**

The first seven weeks of the semester will provide a foundation for the rest. The Introduction and background topic will review the prerequisite courses. Topic 1 will explore computational problems; in topic 2 we will study the verification of algorithms; topics 3 is about the time analysis of algorithms.

Our mid-semester assessment will gather your work from these three major efforts, including optional make-up quizzes taken at the seven-week point to show success with any objectives not achieved earlier.

Plan to succeed with most of the background objectives before taking the topic-1 quiz. Succeed with all the background objectives, and most of the topic 1-3 objectives, by mid-semester. The remaining four topics will go at a faster pace based on this foundation.

The last two weeks of the semester will be for review, a summary quiz, and possible make-up quizzes on the last few topics. On final-exam day students will present their semester projects.

Thus we will have some time flexibility to achieve learning goals, and we’ll also keep up a steady pace, indicated by the following table.
Grading and assessment of learning

In evaluating the work of individual students, evidence of two kinds of accomplishment matter to me: learning of course objectives, and contribution to the learning of others. The graphs below show their relative importance to me and the relative importance of their components or methods of assessment.

Grading weights

The grading framework for this course differs from that of most college courses and reflects a valuing of learning objectives over “grades consciousness” (see B. Walvoord and J. Anderson, Effective Grading, 2010.)

I emphasize what I value (learning, preparation, contribution) over how I measure it (quizzes, exercises). You can make use of this grading system for yourself.

The following rough breakdown of grading weights will apply:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Essential*</th>
<th>Priority**</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>35</strong></td>
<td><strong>4</strong></td>
<td><strong>25</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>To be assessed</th>
<th>%</th>
<th>How measured</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Learning</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Essential objectives</td>
<td>25</td>
<td>Quizzes</td>
</tr>
<tr>
<td>Priority objectives</td>
<td>15</td>
<td>and</td>
</tr>
<tr>
<td>Challenge objectives</td>
<td>10</td>
<td>exercises</td>
</tr>
<tr>
<td>Summarizing learning</td>
<td>10</td>
<td>Summary quiz</td>
</tr>
<tr>
<td><strong>Preparation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daily preparation</td>
<td>5</td>
<td>Turn-in</td>
</tr>
<tr>
<td>Topic problem solving</td>
<td>10</td>
<td>Exercises</td>
</tr>
<tr>
<td>Summarizing preparation</td>
<td>10</td>
<td>Project</td>
</tr>
<tr>
<td><strong>Participation and contribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Steady progress</td>
<td>10</td>
<td>Mid-semester tally</td>
</tr>
<tr>
<td>Presenting</td>
<td>10</td>
<td>Blackboard work</td>
</tr>
<tr>
<td>Teamwork</td>
<td>10</td>
<td>Documentation</td>
</tr>
<tr>
<td>Attendance</td>
<td>5</td>
<td>Tally</td>
</tr>
</tbody>
</table>

I take attendance but rely more on the work that students turn in at the start of each session that reflects their preparation for the session.

Integrity and intellectual debts

The value of your degree depends on the integrity of any evaluations. To protect that value, do your own work, counsel your classmates as needed, and let me know if I should be concerned.

Framingham State University requires that signed academic work be done by the signers and that sources of ideas be acknowledged by references; use of words is to be acknowledged by quotation marks and references. In my courses, students are to help each other with ideas and explanations. Students are not to use each other’s words or the words of others in signed work.

Because the questions from which my quizzes are drawn are available publicly, students who study together to memorize answers may have identical answers on quizzes. But I seek understanding rather than memorization. I don’t accept identical answers as valid for grades. Different students who understand concepts will use different words even if they study together.

Try paraphrasing ideas. This will serve you when you are using your knowledge professionally.

Accommodations

“Students with disabilities who request accommodations are to provide Documentation Confirmation from the Office of Academic Support within the first two weeks of class. Academic Support is located in the Center for Academic Support and Advising (CASA). Please call (508) 626-4906 if you have questions or if you need to schedule an appointment.”

(See www.framingham.edu/CASA/Accommodations/accomm.htm.)
## Course Plan

<table>
<thead>
<tr>
<th>Dates</th>
<th>Topic (with handouts referenced)</th>
<th>Johnsonbaugh, Schaefer</th>
<th>Levitin</th>
<th>Kleinberg, Tardos</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/8 – 9/15</td>
<td>Introduction and background</td>
<td>Ch. 1, 2.1-2.2</td>
<td>Ch. 1</td>
<td></td>
</tr>
<tr>
<td>9/22 – 9/29</td>
<td>1. Classes of problems&lt;sup&gt;1&lt;/sup&gt;</td>
<td></td>
<td></td>
<td>Ch. 1, 3</td>
</tr>
<tr>
<td>9/29 – 10/6</td>
<td>2. Formal verification&lt;sup&gt;2,3&lt;/sup&gt;</td>
<td>Pp. 37-38; Sec. 7.2, 7.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/13 – 10/20</td>
<td>3. Algorithm analysis, recurrences, and brute force&lt;sup&gt;4&lt;/sup&gt;</td>
<td>Sec. 2.3-2.4; 11.1</td>
<td>Ch. 2-4</td>
<td>Ch. 2</td>
</tr>
<tr>
<td>10/20</td>
<td>Make-up quizzes on topics Intro to 2 Semester project preliminary version due</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10/20 – 10/27</td>
<td>4. Divide-and-conquer, decrease-and-conquer</td>
<td>Chs. 3-6</td>
<td>Ch. 5</td>
<td>Ch. 5</td>
</tr>
<tr>
<td>10/3 – 11/10</td>
<td>5. Greedy and other efficient algorithms for optimization</td>
<td>Ch. 7-8</td>
<td>Ch. 7-9</td>
<td>Ch. 4, 6</td>
</tr>
<tr>
<td>11/10 – 11/17</td>
<td>6. Intractable problems and approximate solutions</td>
<td>Sec. 9.5; Ch. 10-11</td>
<td>Ch. 10-12</td>
<td>Ch. 8-13</td>
</tr>
<tr>
<td>11/24</td>
<td>7. Parallel and distributed algorithms and processes; verification and analysis of interactive processes&lt;sup&gt;5,6&lt;/sup&gt;</td>
<td>Ch. 12</td>
<td></td>
<td>Epilogue</td>
</tr>
<tr>
<td>12/1</td>
<td>Semester project due; last make-ups on topics 1-4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/1 – 12/8</td>
<td>Summary and review Student presentations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/8</td>
<td>Summary quiz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/15</td>
<td>Final-exam make-up Student presentations</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This schedule is also posted as an Announcement at Blackboard. If we need to change the schedule, I’ll post an update there.

---

<sup>1</sup> D. Keil, Computational problems, available at course site: [www.framingham.edu/~dkeil/alg-matls.htm](http://www.framingham.edu/~dkeil/alg-matls.htm).

<sup>2</sup> Huth and Ryan, Logic in Computer Science (Cambridge, 2000).

<sup>3</sup> D. Keil, Sample problems in algorithm verification, available at course site.

<sup>4</sup> D. Keil, Sample problems in algorithm analysis, available at course site.

<sup>5</sup> P. Wegner, Why interaction is more powerful than algorithms, CACM, 5/97.

<sup>6</sup> D. Keil, Survey of scalability in concurrent computation, available at course site.
CSCI 347 Analysis of Algorithms subtopic objectives

Use this sheet to keep track as you attain objectives.

Background

- 0.0a Recall logic/proof concepts
- 0.0b Recall array concepts
- 0.0c Recall running-time concepts
- 0.0d Recall tree and hash concepts
- 0.0e Recall graph concepts
- 0.0f Design, code, and test a file management program
- 0.0g Report empirical performance of fast and slow search algorithms
- 0.0h Report empirical performance of fast and slow sort algorithms
- 0.1a Apply logical inference
- 0.1b Use a quantifier
- 0.1c Distinguish predicate from propositional logic
- 0.1d Write a direct proof
- 0.1e Describe mathematical induction
- 0.1f Use induction to prove a theorem about numbers
- 0.2a Write and explain an algorithm that traverses an array
- 0.2b Explain the linear-search algorithm
- 0.2c Explain a sorting algorithm
- 0.2d Write an algorithm with nested loops
- 0.2e Explain linked-list, stack, and queue operations
- 0.3a Explain the running time of an array traversal versus array size
- 0.3b Explain or apply a set concept
- 0.3c Describe a function
- 0.4a Describe a basic concept of trees
- 0.4b Explain why a BST or heap enables efficiency
- 0.4c Explain the binary search
- 0.5a Describe a basic graph concept
- 0.5b Perform an operation on a graph

1. Classes of problems

- 1.0a Recall basic problem concepts
- 1.0b Recall advanced problem concepts
- 1.1a Describe an array problem
- 1.1b Solve a combinatorics problem
- 1.2a Distinguish constraints from optimization
- 1.2b Evaluate a formula and determine its satisfiability
- 1.3a Describe a graph problem
- 1.3b Describe an advanced graph problem
- 1.4 Formally specify a computational problem
- 1.5 Formally specify a property of an interactive system

2. Formal verification

- 2.0a Recall basic verification concepts
- 2.0b Recall advanced verification concepts
- 2.1a Write a simple recurrence
- 2.1b Derive a *while* loop from a recurrence
- 2.1c Write a recursive method that corresponds to a *while* loop
- 2.2a Explain the basic terminology of formal verification
- 2.2b Prove that an algorithm terminates
- 2.3a Given an algorithm and a postcondition, write an appropriate loop invariant
- 2.3b Given a loop invariant, explain why an algorithm is correct
- 2.3c Write an algorithm and prove its correctness
- 2.4 Explain formal rules for algorithm correctness proofs

3. Algorithm analysis; brute force

- 3.0a Recall basic algorithm-analysis concepts
- 3.0b Recall advanced algorithm-analysis concepts
- 3.1 Define or apply the big-O, theta, and big-Θ notations
- 3.2 Write and solve a time recurrence for an algorithm
- 3.3a Explain the brute-force approach to algorithm design
- 3.3b Describe the running time of a nested-loop algorithm
- 3.3c Write and solve a time recurrence for a nested-loop algorithm
- 3.4a Describe an algorithm that performs an exhaustive search
- 3.4b Code and test an exhaustive-search algorithm

4. Divide-and-conquer

- 4.0a Recall basic divide-and-conquer concepts
- 4.0b Recall advanced divide-and-conquer concepts
- 4.1 Relate logarithmic time to very fast algorithms
- 4.2a Write a divide-and-conquer algorithm
- 4.2b Write a recurrence for a divide-and-conquer algorithm
- 4.2c Write and solve a time recurrence for a divide-and-conquer algorithm
- 4.3 Explain tree traversals
- 4.4a Explain breadth-first or depth-first search
- 4.4b Describe directed acyclic graphs

5. Greedy and other efficient algorithms for optimization

- 5.0a Recall basic greedy concepts
- 5.0b Recall advanced greedy concepts
- 5.1a Explain greedy algorithm design
- 5.1b Explain the optimal-substructure property
- 5.2a Describe a greedy algorithm for graphs
- 5.2b Apply a greedy algorithm
- 5.3 Explain an instance of dynamic programming
- 5.4 Explain an instance of transform-and-conquer

6. Intractability and approximate or randomized solutions

- 6.0a Recall basic tractability concepts
- 6.0b Recall advanced tractability concepts
- 6.1 Describe an intractable problem
- 6.2a Describe time classes for tractable and intractable problems
- 6.2b Use a reduction to explain why a problem is intractable
- 6.3 Explain an approximation algorithm
- 6.4 Describe the notion of state-space search
- 6.5 Explain a randomized approach to solving intractable problems

7. Performance of concurrent systems

- 7.0a Recall basic concurrency concepts
- 7.0b Recall advanced concurrency concepts
- 7.1 Describe two forms of concurrency
- 7.2a Design a parallel algorithm
- 7.2b Characterize the performance of a parallel algorithm
- 7.3 Describe a distributed algorithm
- 7.4 Distinguish algorithms from interactive processes
- 7.5 Describe features of multi-stream interaction

** Essential objective
* Priority objective
† Assessed only by exercises
** Assessed by multiple-choice questions
Background

0.0a Recall logic/proof concepts**
0.0b Recall array concepts**
0.0c Recall running-time concepts**
0.0d Recall tree and hash concepts**
0.0e Recall graph concepts**
0.0f Design, code, and test a file-management program**†
0.0g Report empirical performance of fast and slow search algorithms**†
0.0h Report empirical performance of fast and slow sort algorithms**†
0.1a Apply logical inference**
0.1b Use a quantifier*
0.1c Distinguish predicate from propositional logic*
0.1d Write a direct proof**
0.1e Describe mathematical induction**
0.1f Use induction to prove a theorem about numbers**
0.2a Write and explain an algorithm that traverses an array**
0.2b Explain the linear-search algorithm**
0.2c Explain a sorting algorithm**
0.2d Write an algorithm with nested loops**
0.2e Explain linked-list, stack, and queue operations**
0.3a Explain the running time of an array traversal versus array size**
0.3b Explain or apply a set concept**
0.3c Describe a function**
0.4a Describe a basic concept of trees**
0.4b Explain why a BST or heap enables efficiency**
0.4c Explain the binary search**
0.5a Describe a basic graph concept**
0.5b Perform an operation on a graph**

1. Classes of problems

1.0a Recall basic problem concepts**
1.0b Recall advanced problem concepts**
1.1a Describe an array problem**
1.1b Solve a combinatorics problem
1.2a Distinguish constraints from optimization*
1.2b Evaluate a formula and determine its satisfiability*
1.3a Describe a graph problem*
1.3b Describe an advanced graph problem
1.4 Formally specify a computational problem**
1.5 Formally specify a property of an interactive system

2. Formal verification

2.0a Recall basic verification concepts**
2.0b Recall advanced verification concepts**
2.1a Write a simple recurrence**
2.1b Derive a while loop from a recurrence
2.1c Write a recursive method that corresponds to a while loop**†
2.2a Explain the basic terminology of formal verification**
2.2b Prove that an algorithm terminates*
2.3a Given an algorithm and a postcondition, write an appropriate loop invariant*
2.3b Given a loop invariant, explain why an algorithm is correct**
2.3c Write an algorithm and prove its correctness*
2.4 Explain formal rules for algorithm correctness proofs

3. Algorithm analysis; brute force

3.0a Recall basic algorithm-analysis concepts**
3.0b Recall advanced algorithm-analysis concepts**
3.1 Define or apply the big-O, Θ, and big-Ω notations**
3.2 Write and solve a time recurrence for an algorithm**
3.3a Explain the brute-force approach to algorithm design*
3.3b Describe the running time of a nested-loop algorithm*
3.3c Write and solve a time recurrence for a nested-loop algorithm*
3.4a Describe an algorithm that performs an exhaustive search
3.4b Code and test an exhaustive-search algorithm**†

4. Divide-and-conquer
4.0a Recall basic divide-and-conquer concepts* M
4.0b Recall advanced divide-and-conquer concepts M
4.1 Relate logarithmic time to very fast algorithms*
4.2a Write a divide-and-conquer algorithm*
4.2b Write a recurrence for a divide-and-conquer algorithm*
4.2c Write and solve a time recurrence for a divide-and-conquer algorithm*
4.3 Explain trees traversals
4.4a Explain breadth-first or depth-first search
4.4b Describe directed acyclic graphs

5. Greedy and other efficient algorithms for optimization
5.0a Recall basic greedy concepts* M
5.0b Recall advanced greedy concepts M
5.1a Explain greedy algorithm design*
5.1b Explain the optimal-substructure property*
5.2a Describe a greedy algorithm for graphs*
5.2b Apply a greedy algorithm*
5.3 Explain an instance of dynamic programming
5.4 Explain an instance of transform-and-conquer

6. Intractability and approximate or randomized solutions
6.0a Recall basic tractability concepts* M
6.0b Recall advanced tractability concepts M
6.1 Describe an intractable problem**
6.2a Describe time classes for tractable and intractable problems*
6.2b Use a reduction to explain why a problem is intractable
6.3 Explain an approximation algorithm
6.4 Describe the notion of state-space search
6.5 Explain a randomized approach to solving intractable problems

7. Performance of concurrent systems
7.0a Recall basic concurrency concepts* M
7.0b Recall advanced concurrency concepts M
7.1 Describe two forms of concurrency*
7.2a Design a parallel algorithm
7.2b Characterize the performance of a parallel algorithm*
7.3 Describe a distributed algorithm
7.4 Distinguish algorithms from interactive processes*
7.5 Describe features of multi-stream interaction