

CSCI 347 Analysis of Algorithms

Prof. David Keil, Framingham State University, Fall 2012

SYLLABUS DRAFT

Invitation

This course explores, but does not solve, a mystery about computer software performance. The mystery 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 “NPC.”

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 to investigating this mystery, we will look at ways to design computational procedures, ways to prove that an algorithm works all the time, and ways to precisely characterize its running time. We will look at the complexities of problems and explore the mystery of “NPC” problems

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

Tuesdays, 6:30-9:50 p.m., Hemenway Hall 225

To contact instructor

Office hours (Hemenway Hall 318A):

M 2:30-3:30 p.m., W 9:30-10:30 a.m.,

Th 11:30-12:30 p.m., others by appointment

Telephone: (508) 626-4724

Email: dkeil@framingham.edu

URL: www.framingham.edu/~dkeil/alg-matls.htm

Course overview

This course offers an opportunity to think in new ways about algorithms, processes, computational problems, and related performance issues. The course material is mostly language independent; examples will be in pseudocode.

Our approach will be rigorous and mathematical, using concepts that have been presented in your Data Structures course. Many of the algorithms we discuss will be known to you: sorting, searching, tree manipulation. The notion of *mathematical proof* is a central theme of 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.

We will make use of the *logic* you studied in Discrete Math, as applied to algorithms, when we take up *formal verification* of algorithms. The main tools are *preconditions*, *postconditions*, and *loop invariants*.

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

We separate *problems* from *solutions*; in Topic 1, we present algorithmic problems that each have different solutions of different efficiencies, which will be analyzed in later topics. Some of the problems relate to *search*, *reasoning*, and *knowledge representation*, which are topics in artificial intelligence.

An extension to the study of algorithm efficiency will be discussion of *computational complexity of problems*. If the running time of the *best* algorithm that solves a problem is a polynomial function of the size of input data, then the problem is in the class P , polynomial-time problems. If not, then it may at least be in the class NP , of problems whose algorithmic solutions can be *checked* in polynomial time. The hardest problems in NP are called *NP-complete*, and are thought to have no

polynomial-time solutions. NPC problems are among those considered *intractable*, that is, not worth trying to solve due to their resource requirements.

We will contrast *algorithmic computing* with *interactive computing* and will distinguish the different types of associated *problems*. We will discuss *evolutionary computation*, one of the instructor's research interests, as a mixture of algorithms and interaction. We will argue for a *paradigm shift* in how you think about computing. We present *reinforcement learning*, an interactive type of learning. Machine learning is a subfield of artificial intelligence. One special aspect of this course will be coverage of *parallel and distributed algorithms and processes*.

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

Textbook

R. Johnsonbaugh and M. Schaefer. *Algorithms*. Pearson Prentice Hall, 2004.

Reading text material related to the course topics is vital. We will refer to the required text often.

Basic skills

All students are expected to be able to do the following, which are presented in Data Structures or Discrete Math I:

1. Write an algorithm to traverse an array and explain why it works
2. State and explain the running time of an array traversal as a function of array size
3. Write an algorithm using nested loops
4. Explain why a BST or heap enables efficiency
5. Describe operations on graphs
6. Describe the inductive proof technique

Grades and classroom format

Please see the paper, "What we do in my classroom," which is part of this course's syllabus. See especially guidelines there for assignments, grading, and collaboration.

What will we investigate?

We'll look at the following questions:

1. How useful is the study of computational *problems*?

2. Does it pay to *prove* claims about the behavior of systems?
3. Is *time scalability* a good way to measure the performance of computing systems?
4. How much does studying *design approaches* help us create efficient solutions?
5. How hard is *optimization*?
6. How helpful is it to mathematically characterize some problems as *hard* or *intractable*?
7. Are the problems *solvable* in polynomial time the same problems *whose solutions can be checked* in polynomial time; i.e., is P equal to NP?
8. Are approximate *satisficing* solutions a practical way to address hard computational problems?
9. Do concurrency and interaction belong in an algorithms course?
10. How do scalability and communication time figure in measuring the efficiency of parallel and distributed solutions?

Student presentations

The textbook and instructor-prepared slides present dozens of algorithms. Each student will select at least two of these algorithms, and research results, to present to the group. Topics that may be covered for an algorithm include the problem specification, the design approach, the pseudocode of the algorithm, example executions, time and space analysis, advantages and drawbacks, and easy, hard, and surprising aspects. Students may consult others in preparing to present.

Web aspect of course

This course is listed under <http://framingham.blackboard.com>. All course materials will be available there. The site hosts a private discussion board for students enrolled in the course and a Grade Book where quiz and other grades are posted.

Course objectives

Upon completion of the course students are expected to be able to do the following:

- 1a. Specify a computational problem using preconditions and postconditions
- 1b. Distinguish constraint from optimization problems
- 1c. Formally specify properties of an interactive system

- 2a. Explain the basic terminology of formal verification
- 2b. Given an algorithm commented with a loop invariant, explain why the algorithm is correct
- 2c. Given an algorithm commented with a postcondition, write an appropriate loop invariant
- 3a. Define and use the big-O, theta, and big-omega notations
- 3b. Given an iteratively expressed algorithm, write a recursive version
- 3c. Write a recurrence that defines the function computed by a simple linear-time algorithm
- 3d. Write and solve a recurrence that defines the time function of a linear-time algorithm
- 3e. Explain and use the brute-force approach to algorithm design
- 3f. Write a recurrence that defines the function computed by a quadratic-time brute-force algorithm
- 3g. Write and solve a time recurrence for a quadratic-time algorithm
- 4a. Explain an instance of the divide-and-conquer approach.
- 4b. Write a recurrence that defines the function computed by a divide-and-conquer algorithm
- 4c. Write and solve a time recurrence for a divide-and-conquer algorithm
- 4d. Explain and use traversal methods for trees
- 4e. Explain breadth-first and depth-first graph searches
- 4f. Explain the basic binary-search-tree algorithms
- 4g. Explain the basic algorithms on heaps
- 5a. Explain the greedy approach to algorithm design
- 5b. Explain the optimal-substructure property
- 5c. Describe a greedy algorithm for graphs

- 5d. Explain an instance of the dynamic-programming approach
- 5e. Explain an instance of the transform-and-conquer approach
- 6a. Relate the main complexity classes for tractable and intractable problems
- 6b. Explain how problems may be shown to be intractable using reductions
- 6c. Explain approximation or randomized approaches to solving intractable problems
- 7a. Design a simple parallel or distributed algorithm
- 7b. Characterize the performance of a parallel or distributed algorithm
- 7c. Distinguish algorithmic from interactive computation
- 7d. Discuss performance of interactive systems
- 8a. Solve a problem as part of a team
- 8b. Present a short talk in the classroom

Semester grading weights

Objectives attained	40 %
Assignments	10
Multiple-choice quizzes	10
Problem-solving quizzes	10
Research	10
Final exam	10
Participation	<u>10</u>
	100

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 <http://www.framingham.edu/CASA/Accommodations/accomm.htm>.)

Course Plan

Dates	Topic	Reading in Johnsonbaugh-Schaefer
9/6	<i>Introduction</i>	Ch. 1, 2.1-2.2
9/13-9/20	1. Classes of problems	Handouts ¹
9/20-9/27	2. Formal verification	Handouts ^{2,3} ; pp. 37-38; Sec. 7.2, 7.4
10/4	<i>Problem-solving quizzes on topics 1-2</i>	
10/4-10/11	3. Algorithm analysis, recurrences, and brute force	Sec. 2.3-2.4; 11.1; handout ⁴ Levitin, Ch. 2-3 (optional)
10/18-10/25	4. Divide-and-conquer, decrease-and-conquer	Chs. 3-6
10/25-11/8	5. Greedy and other efficient algorithms for optimization	Ch. 7-8
11/1	<i>Problem-solving quizzes on topics 3-4</i>	
11/8-11/22	6. Intractable problems and approximate solutions	Sec. 9.5; Ch. 10; Ch. 11
11/15	<i>Make-up quizzes on topics 1-4</i>	
11/22-12/6	7. Parallel and distributed algorithms and processes; verification and analysis of interactive processes	Ch. 12; Handouts ^{5,6}
11/29	<i>Problem-solving quiz on topics 5-6</i>	
12/13	<i>Summary; problem-solving quiz on topic 7; Summary quiz</i>	
12/20	<i>Final exam (multiple choice; all topics); Optional objectives questions</i>	

¹ D. Keil, Computational problems, available at course site: www.framingham.edu/~dkeil/alg-matls.htm.

² Huth and Ryan, *Logic in Computer Science* (Cambridge, 2000).

³ D. Keil, Sample problems in algorithm verification, available at course site.

⁴ D. Keil, Sample problems in algorithm analysis, available at course site.

⁵ P. Wegner, Why interaction is more powerful than algorithms, *CACM*, 5/97.

⁶ D. Keil, Survey of scalability in concurrent computation, available at course site.