Computer Science 170 (Summer Term, 2018)
Introduction to the Theory of Computation

Who, Where, When

Course instructor: Harry Mairson (mairson@brandeis.edu). I'm not permanent faculty at Tufts, so I especially encourage you to communicate with me via electronic mail, for the most reliable responses to your questions. I like e-mail.

Because this term is so short (it's like time-lapse photography...), and the lectures are so long, it is *really important* that you do the reading for class, attend class, and that you come with *questions* to ask me about what you do not understand. Doing the homework will give you more clues as to what questions you should be asking.

Teaching assistant: Sara Amin (sara.amin@tufts.edu), hours TBA.
Grader: Yashvardhan Sathe (yashvardhan.sathe@tufts.edu).
Course website: www.cs.brandeis.edu/~mairson/Courses/cs170.

Time and place: Halligan Hall 111B, Mondays and Wednesdays, 6–9.30pm. Office hours before and after class, and by appointment. (That's probably the most convenient for everybody.)

Required Textbook

Michael Sipser.
Introduction to the Theory of Computation. (*Second or third* edition)

Required Poetry *(for fun)*

Martin Cohn and Harry Mairson.
New proofs of old theorems.
Unpublished manuscript: see www.cs.brandeis.edu/~mairson/poems/poems.html

What is this course about?

A computer isn't just a kind of *machine*—it is also a kind of *idea* for organizing "thoughts" and computation. But what is it that computers can do? What kinds of problems are they incapable of solving? When we look at the kinds of problems computers *can* solve, what *computational resources* (the usual parameters being *time* and *memory*) are necessary and sufficient? In what precise sense can we say that a certain problem is *harder* than another?

We will look at various models of computation, and try to analyze them in terms of their *universality*—in what sense can one model *simulate* another?—and in terms of their computational power and flexibility. Whenever possible, we will try to relate the ideas in

this class to more "real world" issues of computing: language design, recursion, compilers, circuit design, etc.

How Hard Will This Course Be?

This course will not consume hours and hours like other computer science classes, since there are no hands-on programming problems. But some occasional *real thinking* will be necessary, since it's a very math-oriented course. Some familiarity with discrete mathematics would be very helpful, particularly if you have seen induction arguments. As always, that good old "mathematical maturity" goes a long way.

What work will be required?

There will be 4 problem sets, a one hour *in-class* midterm on Wednesday, June 13, and a three hour *in-class* final examination on Wednesday, June 27, the last day of class. No programming will be required.

I encourage you to discuss the problem sets with your fellow students in the class, and to work collaboratively, particularly on difficult problems. Such collaboration *must* be mentioned explicitly in handed-in solutions, and each student is responsible for handing in individual problem sets: no "group copies" will be accepted. When you discuss problems individually with the teaching assistant or the course instructor, that counts for collegial collaboration that must be recognized.

Grading and homework policy

Homework, 20 percent; midterm, 35 percent; final, 45 percent. The percentage for homework is low because it is hard to figure out who did what in collaborations. Instead, think of homework as the best way of preparing for exams, where you have to do your own work. When it comes to learning, there is no substitute for working problems—reading is a poor alternative. Doing homework yourself is the way you learn what the course is about!

There will be a problem set just about every week.

Late homework: If you can get your homework to the TA before the TA grades the homework, no late penalty will be made. After, it is completely at the convenience of the TA. Do your homework on time!

Smartphone policy: Students who are reading email, etc., on their smartphones, tablets *et al.* may be asked to leave the lecture.

Honor code: Cheating is serious and will not be tolerated. The instructor and staff will make every attempt to be reasonable about assignments, due dates, etc., but infractions of the honor code will be dealt with severely.

Tentative syllabus

10 lectures overall, including in-class midterm and final examination.

Introduction; Finite Automata *[3 lectures]*

May 23: Introduction, course administration. Survey of relevant discrete mathematics, especially arguments based on induction, and their relation to recursive programming. Deterministic finite automata: examples, formal definition, programming style. [Sipser 0.1–0.4, 1.1] *Problem Set 1 handed out.*

May 30: Nondeterministic finite automata: examples, formal definition, programming style. Equivalence with deterministic finite automata. Regular expressions. Equivalence with nondeterministic finite automata. Regular expressions as a programming language. [Sipser 1.2, 1.3]

June 4: Pumping lemma for regular languages. Proving languages are not regular. Path problems. Using linear algebra to understand automata and regular languages. [Sipser 1.4]

Context-free languages *[2 lectures]*

June 6: Context-free grammars: examples, formal definition, programming style. Chomsky normal form. Parsing context-free languages using dynamic programming. [Sipser 2.1, class handout] *Problem Set 1 due. Problem Set 2 handed out.*

June 11: Pumping lemma for context-free languages. Proving languages are not context-free. Pushdown automata. Equivalence with context-free grammars. [Sipser 2.2, 2.3]

Turing Machines and Undecidability *[2 lectures]*

June 13: Midterm examination *(first hour only).* Turing machines and variants. Algorithms. Church-Turing thesis. Invariance thesis. [Sipser 3.1, 3.2, 3.3] *Problem Set 2 due. Problem Set 3 handed out.*

June 18: Decidable properties of languages, both regular and context-free. Undecidability of the halting problem. Reducibility. [Sipser 4.1, 4.2, 5.1 through p. 192]

Complexity Theory: P, NP, PSPACE, Completeness *[2 lectures]*

June 20: Measuring complexity. Polynomial time computation: the class **P**. PTIME-completeness. The Circuit Value Problem. Nondeterministic polynomial-time computation: the class **NP**. [Sipser 7.1–7.4, class handout] *Problem Set 3 due. Problem Set 4 handed out.*

June 25: Definition of **NP**-completeness. The Cook-Levin theorem: satisfiability is **NP**-complete. [Sipser 7.4, class handout]

June 27: Final Examination *Problem Set 4 due.*