CSE491/596 Course Information Fall 2020

Instructor

Dr. Kenneth W. Regan, 326 Davis Hall, 645-4738, regan@buffalo.edu;

Office hours: Still TBA.

Lectures

(LEC) MWF 3:00pm–3:50pm in Knox 20

Reading—notes to be given in class, no textbook purchase

1. Notes by Arun Debray for Stanford's undergraduate course as taught by Ryan Williams (who is now at MIT). They are based on the textbook by Sipser listed below (which is used even for graduate classes at MIT) but Williams's priorities match my own very closely.

2. Chapters 27 and 28 of the *CRC Handbook on Algorithms and Theory of Computing*, co-authored by me with Professors Eric W. Allender and Michael C. Loui. These are for the second half of the course and will be given out in class.

3. Excerpts from my textbook with Richard Lipton, *Quantum Algorithms Via Linear Algebra*. These will also be given out.

4. The weblog "Gödel's Lost Letter and P=NP" may be used for assigned readings.

Optional Alternate Sources

1. Steven Homer and Alan Selman, *Computability and Complexity Theory*. The previous textbook.

2. M. Sipser, *Introduction to the Theory of Computation*, 3rd. ed., Thompson SW International, 2012. The popular text used in the undergraduate course, CSE396.

3. J. Hopcroft and J. Ullman, *Introduction to Languages, Automata Theory, and Computation*, Addison-Wesley, 1979. *The* classic text. This course will mostly parallel the material in chapters 7–13 of this text; all assumed background and much more is in chapters 1–6.

4. H. Lewis and C. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981. Has more examples and illustrations and neat little tidbits than Hopcroft-Ullman, but some messier notation.

5. N. Cutland, *Computability*, Cambridge University Press, 1980. A short-but-comprehensive and crystal-clear treatment of computability theory, the main topic of the first part of the course.

Examinations:

- Two prelim exams held in class period.
- One *cumulative* 3-hr. final.

Organization:

A novel feature this year is the introduction of a class-participation component in the manner of British-system tutorials. This was in fact the original purpose of "recitations" at UB and was practiced by several departmental colleagues when I arrived 30 years ago and our enrollment was smaller. The expectation will be for each student in groups of 2 or 3 to present a designated part of a bi-weekly assignment, mixed in with saying how the answer or method would change when aspects of the problem change and general participation in discussions about the material. These required meetings will alternate weekly with optional homework review sessions.

The course will be graded on a total-points system. Letter grades will also be given for individual exams and possibly some assignments, as a help in telling where you stand, but only point totals will have official significance. The weighting of grades in this course shall be:

Homework:	40%
Tutorial:	15%
Prelims:	18%
Final:	27%

I reserve the right to 5% leeway in weighting while assigning the final letter grade. This is most typically done for students who do markedly well on the final exam, when it may be treated as if it were worth 32% for that student. This will only be done to an individual student's advantage, and will have no effect on others' grades. The assignments, examinations, and grading rubric will be the same for both sections. See notes on course material and philosophy below—which is partly designed to allow for differential background between undergrad and grad and also for those within both sections having had or not-had coverage of automata and formal languages in previous courses.

The first prelim exam is tentatively set for **Fri. Oct. 16** in class period. Arrangements for those who are remote in different time zones will be worked out individually.

The homework will consist of bi-weekly problem sets. All submissions will be via *CSE Autograder* in the form of PDF files. The first assignment will be given on **Fri. Sept. 11**. The logistics of exam submission will be similar to that of homeworks but in a timed period. As also discussed, I am legislating a feature that was approved after a week-13 weather event upended schedules in 2018 and again in 2019: an optional extra homework to compensate for a mishap on the final exam.

Problem set submissions must be *your own individual work*. No joint submissions will be accepted. In an early lecture I will explain the purpose of individual work, academic integrity, and the "qualitative" nature of exercises in this course. I will give guidelines on how work can be done and what can be discussed among you. Any cheating will be punished per the department policy at

https://engineering.buffalo.edu/computer-science-engineering/information-for-students/policies/academic-integrity.html

Course Coverage and Approximate Calendar ("the" syllabus)

The plan is to cover finite automata and (non-)regular languages in the first three weeks (plus a day), then computability and undecidability through mid-term. The second half will feature computational complexity (using my chapters with Allender and Loui and Debray's notes as parallel texts): time and space complexity defined, why we emphasize P and NP, NP-hardness and completeness, other salient complexity classes and the (known and unknown) relationships among them. Basics of randomized algorithms and quantum computing will round out the coverage. Homeworks or Piazza posts will give indication from week to week of exactly what to read. I cannot spell out a timetable in greater detail now because my lectures will adjust to the needs of the class. I *welcome* feedback *to me personally*.

The material is chosen as a blend of undergraduate and graduate content. Here are five "logical units" of a "CS theory" course (apart from Algorithms):

1. Regular languages and finite automata;
2. Context-free grammars and pushdown automata (grad extension: other grammars);
3. Computability and undecidability (grad extension: logic and recursion theory);
4. Computational complexity (can branch into information complexity and much else);
5. Quantum computing.

The old CSE496/596 had the first three. In the 1990s we found that almoat all graduate students had the first two areas, so the stand-alone CSE596 modernized with complexity in their place, while CSE396 kept the first three with a brief introduction to P and NP (often CSE331 has covered NP-complete problems in more detail). Now the first is delegated to CSE191 and the second is attended to by coverage of BNF grammars in CSE305 and/or coverage of compilers in other courses. The present design for CSE491/596 skips grammars but includes everything else, with complexity expressly at graduate level.

The classic textbook by Sipser actually has this nature: it is labeled for both undergrad and grad even at MIT and usually the former does chapters 1–5 and 7 while grad emphasizes chapters 6–9. The course notes by Arun Debray are from a Stanford undergraduate course that used Sipser but accentuated what we here regard as graduate content. The quantum material is from a textbook (mine with Richard J. Lipton) that also targets both upperclass undergraduate and graduate courses, and has been used for undergrad here (by Prof. Knepley, not yet myself). Thus by philosophy the course is blended.

For outcomes and assessment objectives, the course includes ABET CSE undergraduate criterion (6): apply computer science theory and software development fundamentals to produce computing-based solutions. The particular indicator is to apply computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices. The first example of such a tradeoff and design choice will come in the discussion of using the spirit but *not* the letter of the NFA-to-DFA construction when matching strings to regular expressions. The presence of tradeoffs among time, space, and other resources (in fact or conjecture) is a backbone of complexity theory. This also blends with the philosophy of emphasizing the algorithmic content of proofs. Many homework exercises and examination questions will embody such design choices and assess the understanding of them.