BU CS 332 – Theory of Computation

Please point a browser to https://forms.gle/jmR22HpGPJzuQFtXA for in-class polls

• Lecture 1:
  • Course information
  • Overview

Reading:
Sipser Ch 0

Mark Bun
September 6, 2022
Course Information
Course Staff

• **Me:** Mark Bun (he/him/his)
  • At BU since Sept. 2019
  • Office hours: TBD
  • Research interests: Theory of computation (!)
    More specifically: Computational complexity, data privacy, cryptography, foundations of machine learning

• **TF:** Satchit Sivakumar
  • Office hours: TBD
Course Webpage

https://cs-people.bu.edu/mbun/courses/332_F22/

Serves as the syllabus and schedule

Check back frequently for updates!
Course Structure

Grading
- Homework (54%): Roughly 10 of these
- In-class tests (34%):
  - Test 1 (10%)
  - Test 2 (10%)
  - Final (14%)
- Participation (12%): In-class polls, post-lecture check-ins HW0, etc.
Homework Policies

• Weekly assignments due Thursday @ 11:59PM
• No late days
• Lowest homework score will be dropped

• Homework to be submitted via Gradescope
  • Entry code: XV2EDY

• You are encouraged to typeset your solutions in LaTeX
  (resources available on course webpage)

• HW0 out, due Th 9/8 (just some housekeeping)
• HW1 to be released on Th 9/8, due Th 9/15
Homework Policies: Collaboration

• You are encouraged to work with your classmates to discuss most of the homework problems

• HOWEVER:
  • You may collaborate with at most 3 other students
  • You must acknowledge your collaborators and write “Collaborators: none” if you worked alone
  • You must write your solutions by yourself
  • You may not share written solutions
  • You may not search for solutions using the web or other outside resources
  • You may not receive help from anyone outside the course (including students from previous years)

• Some problems will be marked “Individual Review”. No collaboration is allowed on these problems.
Collaboration Dilemma

If you worked alone on a homework assignment...

(a) You don’t need to include a collaboration statement
(b) You should invent a fake collaborator and acknowledge them appropriately
(c) You should include the collaboration statement “Collaborators: none”
(d) You should hurriedly call up three of your friends in the class at 11:55PM, briefly discuss the problems, and acknowledge them
Homework Policies: Collaboration

Details of the collaboration policy may be found here: https://cs-people.bu.edu/mbun/courses/332_F22/handouts/collaboration.pdf

**Important:** Sign this document to affirm you understand it, and turn it in via Gradescope by 11:59PM, Th 9/8
Textbook

Introduction to the Theory of Computation (Third Edition) by Michael Sipser

- It’s fine if you want to use an older edition, but section numbers may not be the same
- Other resources available on course webpage
Participation

• Your class participation score (12% of the course grade) will be determined by
  - Answering in-class polls on Google Forms
  - Completing short check-ins on Gradescope within 48 hours after each lecture
  - Handing in completed worksheets at the end of each discussion section

• You can also increase your participation score by participating thoughtfully in lecture, discussion, office hours, and on Piazza
Piazza

• We will use Piazza for announcements and discussions
  • Ask questions here and help your classmates
  • Please use private messages / email sparingly

https://piazza.com/bu/fall2022/cs332
Expectations and Advice for Succeeding in CS 332
Our (the Course Staff’s) Responsibilities

• Guide you through difficult parts of the material in lecture
• Encourage active participation in lectures / section
• Assign practice problems and homework that will give you a deep understanding of the material
• Give detailed (formative) feedback on assignments
• Be available outside of class (office hours, Piazza)
• Regularly solicit feedback to improve the course
Your Responsibilities

• Concepts in this course take some time to sink in. Keep at it, and be careful not to fall behind.

• Do the assigned reading on each topic before the corresponding lecture.

• Take advantage of office hours.

• Participate actively in lectures/sections and on Piazza.

• Allocate lots of time for the course: comparable to a project-based course, but spread more evenly.
Prerequisites

This class is fast-paced and assumes experience with mathematical reasoning and algorithmic thinking

You must have passed CS 330 – Intro to Algorithms

This means you should be comfortable with:

- Set theory
- Functions and relations
- Graphs
- Pigeonhole principle
- Propositional logic
- Asymptotic notation
- Graph algorithms (BFS, DFS)
- Dynamic programming
- NP-completeness

Come talk to me if you have questions about your preparation for the course
Advice on Homework

• Start working on homework early! You can get started as soon as it’s assigned.

• Spread your homework time over multiple days.

• You may work in groups (of up to 4 people), but think about each problem for at least 30 minutes before your group meeting.

• To learn problem solving, you have to do it:
  • Try to think about how you would solve any presented problem before you read/hear the answer
  • Do exercises in the textbook in addition to assigned homework problems
Advice on Reading

• Not like reading a novel
• The goal is not to find out the answers, but to learn and understand the techniques
• Always try to predict what’s coming next
• Always think about how you would approach a problem before reading the solution
• This applies to things that are not explicitly labeled as exercises or problems!
Academic Integrity

Extremely important: Read and understand the Collaboration and Honesty policy before you sign it

Violations of the collaboration policy...will result in an automatic failing grade and will be reported to the Academic Conduct Committee (ACC). The ACC often suspends or expels students deemed guilty of plagiarism or other forms of cheating.

If you find yourself in a desperate situation:
• Hand in as much of the assignment as you’re able to complete
• Remember the lowest HW grade is dropped
• Talk to us! We want to help

...cheating is seriously not worth it
Course Overview
Objective

Build a *theory* out of the idea of *computation*
What is “computation”

• Examples:
  • Paper + pencil arithmetic
  • Abacus
  • Mechanical calculator
  • Ruler and compass geometry constructions
  • Java/C programs on a digital computer

• For us: Computation is the processing of information by the unlimited application of a finite set of operations or rules
Other examples of computation?

Making life decisions
- computation in the brain

Bespoke devices, e.g., TV controllers

Cooking (following a recipe)

Knitting (arts & crafts)

Gambling / bookmaking

Internet/distributed systems
- Equilibrium in games
- DNA/protein folding
- Crypto/security protocols
- Quantum computation

Slime molds
What do we want in a “theory”? 

- General ideas that apply to many different systems 
- Expressed simply, abstractly, and precisely 

- **Generality**
  - Independence from Technology: Applies to the future as well as the present 
  - Abstraction: Suppresses inessential details 

- **Precision**: Can prove formal mathematical theorems 
  - Positive results (what *can* be computed): correctness of algorithms and system designs 
  - Negative results (what *cannot* be computed): proof that there is no algorithm to solve some problem in some setting (with certain cost)
Parts of a Theory of Computation

• Models for **machines** (computational devices)
• Models for the **problems** machines can be used to solve
• **Theorems** about what kinds of machines can solve what kinds of problems, and at what cost

**This course:** Sequential, single-processor computing

Not covered:
- Parallel machines
- Real-time systems
- Distributed systems
- Mobile computing
- Quantum computation
- Embedded systems
What is a (Computational) Problem?

A single question with infinitely many instances

Examples:

- **Parity**: Given a string consisting of $a$’s and $b$’s, does it contain an even number of $a$’s?

- **Primality**: Given a natural number $x$ (represented in binary), is $x$ prime? (e.g., $100$ in binary is prime)

- **Halting Problem**: Given a C program, can it ever get stuck in an infinite loop?

For us: Focus on *decision* problems (yes/no answers) on *discrete* inputs
What is a (Computational) Problem?

For us: A problem will be the task of recognizing whether a string is in a language.
What is a (Computational) Problem?

For us: A problem will be the task of recognizing whether a string is in a language

• **Alphabet**: A finite set \( \Sigma \)
  
  Ex. \( \Sigma = \{a, b, \ldots, z\} \) \[ \Sigma^* = \{\varepsilon, a, b, \ldots\} \]

• **String**: A finite concatenation of alphabet symbols
  
  Ex. \( bqr, ababb \)
  
  \( \varepsilon \) denotes empty string, length 0
  
  \( \Sigma^* = \) set of all strings using symbols from \( \Sigma \)

• **Language**: A set \( L \) of strings \[ L \subseteq \Sigma^* \]
Examples of Languages

**Parity:** Given a string consisting of $a$'s and $b$'s, does it contain an even number of $a$'s?

$\Sigma = \{a, b\}$  
$L = \{all\ strings\ over\ a\ and\ b,\ consisting\ of\ an\ even\ \#\ of\ a's\}$

**Primality:** Given a natural number $x$ (represented in binary), is $x$ prime?

$\Sigma = \{0, 1\}$  
$L = \{all\ binary\ strings\ representing\ prime\ \#s\}$

**Halting Problem:** Given a C program, can it ever get stuck in an infinite loop?

$\Sigma = \text{ASCII}$  
$L = \{C\ programs\ that\ loop\ forever\ on\ some\ input\}$
Primality language

Which best represents the language corresponding to the Primality problem? (Strings over the alphabet \{0, 1\} that are binary representations of prime numbers.) Let’s say the most significant bit is on the left, so “100” is the binary representation of 4.

(a) \{2, 3, 5, 7, \ldots \}  
(b) \{10, 11, 101, 111, \ldots \}  
(c) \{11, 111, 11111, 1111111, \ldots \}  
(d) \{11, 011, 101, 110, 111, 0111, \ldots \}
Machine Models

Computation is the processing of information by the unlimited application of a finite set of operations or rules

Input: $a b a a$ ...

Abstraction: We don’t care how the control is implemented. We just require it to have a finite number of states, and to transition between states using fixed rules.
Machine Models

- **Finite Automata (FAs):** Machine with a finite amount of unstructured memory

  Input: \[ a \ b \ a \ a \ \ldots \]

  Control scans left-to-right
  Can check simple patterns
  Can’t perform unlimited counting

  Useful for modeling chips, simple control systems, choose-your-own adventure games...
Machine Models

- **Turing Machines (TMs):** Machine with unbounded, unstructured memory

Input: \[ \text{ } a \quad b \quad a \quad a \quad \text{...} \]

Finite control

Control can scan in both directions
Control can both read and write

Model for general sequential computation

**Church-Turing Thesis:** Everything we intuitively think of as "computable" is computable by a Turing Machine