Computational Complexity Theory
Fall 2025

Logistics and Course Overview
August 28, 2025

Lijie Chen
University of California, Berkeley
lijiechen@berkeley.edu

Course Information

- **Instructor:** Lijie Chen (lijiechen@berkeley.edu)
- **Office Hours:** 2:00 - 3:00 PM, Tuesday
- **Reader:** Carl Sun (carlsxh@berkeley.edu)
- **Time:** Tuesday, Thursday 12:30 - 2:00 PM
  (we will be on Berkeley time, so 12:40 :))
- **Location:** Etcheverry 3113 (map)
- **Slides:** can be found on the course website

Grading Information

- **Homework Assignments:** 50% (4 assignments total)
- **Final Project & Presentation:** 50%
- **Project presentations will be scheduled during the final weeks of the semester.**

Final Project and Presentation (50%)

- Can team up with up to 2 other students.
  - Higher expectations for groups with more students.
- A project proposal (1-2 pages): 5% score
- A mid-term progress report (1-2 pages): 5% score
- A final project paper (at least 5 pages): 20% score
- A final presentation in class: 20% score

Two Types of Projects

- **Survey of a complexity-related topic we haven't covered in class.**
  - check with instructor if you are unsure about the connection to complexity theory.
- **Exploring an open direction related to complexity theory.**
  - You are NOT required to solve any open problems to get a good grade.
  - can always turn into a survey project halfway through.

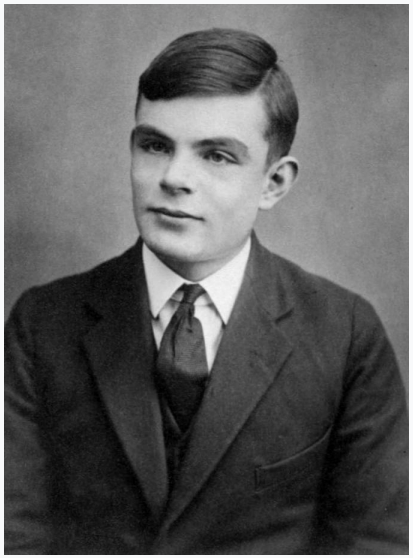
List of Suggested Projects

- Will be posted on the course website by Sept 2nd.
- You can also propose your own project!
  - check with instructor if you are unsure about the connection to complexity theory.

Course Overview

- What is complexity theory?
- What topics will be covered in the course?

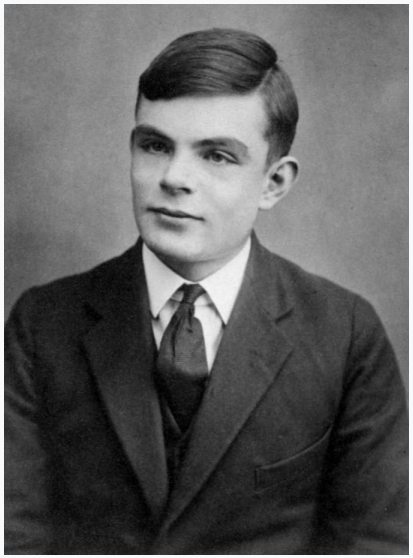
Computability: what is computable in *principle*?



Alan Turing (1912-1954)

- Turing machine: a simple model of computation

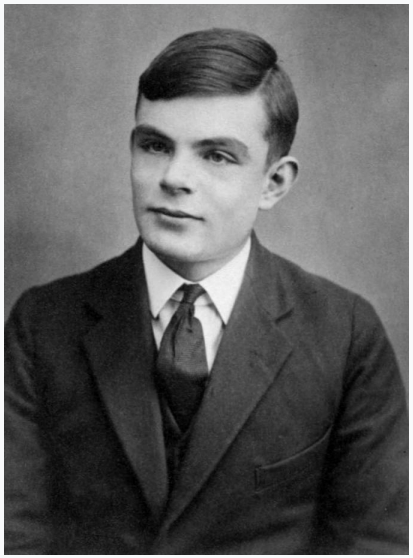
Computability: what is computable in *principle*?



Alan Turing (1912-1954)

- Turing machine: a simple model of computation
  - Church-Turing thesis: every physically computable function can be computed by a Turing machine

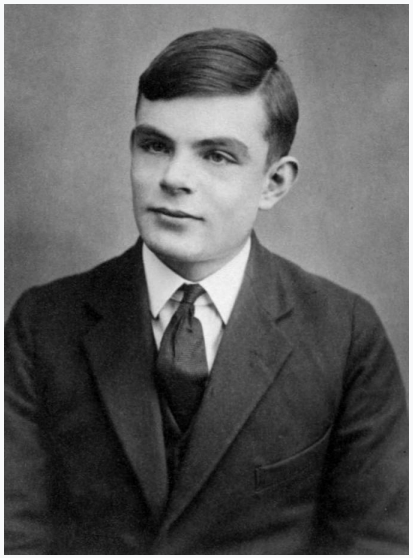
Computability: what is computable in *principle*?



Alan Turing (1912-1954)

- Turing machine: a simple model of computation
  - Church-Turing thesis: every physically computable function can be computed by a Turing machine
  - This is indeed a physical law!

Computability: what is computable in *principle*?



Alan Turing (1912-1954)

- Turing machine: a simple model of computation
  - Church-Turing thesis: every physically computable function can be computed by a Turing machine
  - This is indeed a physical law!
- Easy: finite steps; Hard: infinite steps

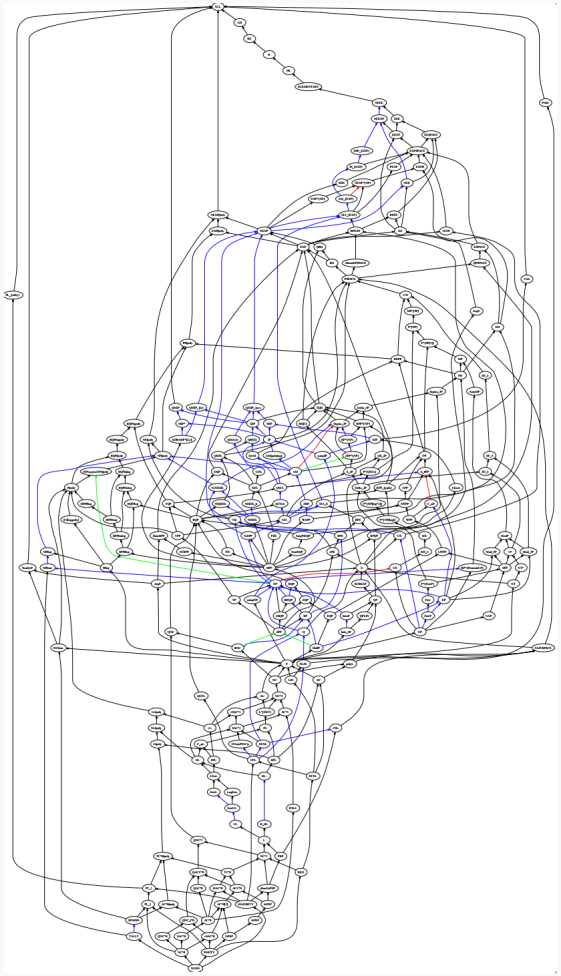
Complexity: What is computable in Practice?

- $10^{10^{10^{10}}}$ steps is too long to complete
- More precisely, complexity theory studies what is computable under resource constraints
  - **Time**: how many *time steps* are needed to solve a problem?
  - **Space**: how much *memory* is needed to solve a problem?
  - **Randomness**: how much *randomness* is needed to solve a problem?
  - **Non-determinism**: how many *nondeterministic guesses* are needed to solve a problem?

Complexity: What Type of Computation is Interesting?

- Complexity theory always love to formalize whatever exciting new type of computation is out there and study them formally.
- **Quantum complexity**: what can be computed by a quantum computer? (again, you can study time, space, and nondeterminism)
- **Parallel complexity (i.e., GPUs)**: what can be computed by a parallel computer? (here, you distinguish between *work* (total amount of computation) and *depth* (maximum number of steps))
- **Later in this course**: we will study the complexity of *transformers* and *state-space models*.

Complexity: What Type of Computation is Interesting?



The Complexity Zoo: A diagram showing relationships between complexity classes

Complexity and Physics

- As we discussed, complexity theory is fundamentally about what you can do in our physical world!
- **Fluid computers**: Finite time blowup for an averaged three-dimensional Navier-Stokes equation
- **Undecidability of the spectral gap**: Certain properties of materials are undecidable in principle.

Course Overview

- What is complexity theory?
- What topics will be covered in the course?

Time complexity: How much time is needed to solve a problem?

- **Time hierarchy theorems**: Given more time, you can always solve more problems.
- **Nondeterministic time hierarchy theorems**: The same holds for nondeterministic time.

Oracle Computation and Relativization

Oracle Computation: What can you compute if you are given a *magic box* for some function?

Relativization: Why it's so hard to separate P from NP? or NP from coNP?

Space complexity: How much space is needed to solve a problem?

- **Savitch's theorem**: Non-deterministic space is *not* more powerful than deterministic space!
- **NL = coNL**: So, for space, you get "$P = NP = coNP$"!

Space complexity: How much space is needed to solve a problem?

- **Time vs. space**: For an algorithm running in time $T(n)$, can you always run it with significantly less space? (but potentially more time)
- **T-time in $\sqrt{T}$-space**: recent breakthrough by Ryan Williams!

The space complexity of recursion

How much space is needed to compute a recursive function?

```
DFS(u):
    if u is a leaf:
        return LEAF_VALUE(u)
    out = []
    for v in CHILDREN(u):
        out.append(DFS(v))
    result = COMBINE(out)
    return result

# LEAF_VALUE(u): outputs at most m bits
# COMBINE(out): outputs at most m bits
# CHILDREN(u): at most 2 children
# Recursive depth is at most d
```

- **Question**: how much space is needed to compute DFS(root)?
- **Naïve answer**: $O(m \cdot d)$ bits
- **Cook-Mertz algorithm**: $O(m + d)$ bits!!!

Circuit Complexity

- **Circuit**: a combinatorial view of computation.
- **Circuit lower bounds**: a once-promising approach towards P vs. NP.
- **Parity not in** AC^0: constant-depth AND/OR/NOT circuits cannot compute parity!
- **The Natural Proof Barrier**: Why circuit lower bounds got stuck.
- **Recent maximum circuit lower bounds**: By finding a new *algorithm*!

Randomized Complexity and Derandomization

- **Randomized complexity**: What can be computed by a randomized algorithm?
- **Derandomization**: Can we simulate a randomized algorithm with a deterministic algorithm with similar efficiency?
- **Pseudorandomness**: efficient generators for "random strings" that are indistinguishable from true random strings! Using them one can derandomize any randomized algorithm efficiently.

Interactive Proofs and PCPs

- Complexity reimagined what it means to be a "proof"
  - **Completeness**: the honest prover can convince the verifier of a *true* statement.
  - **Soundness**: no malicious prover can convince the verifier of a *false* statement.
- **Interactive proofs**: a proof system where the verifier is interactive with the prover, with completeness and soundness as above.
- **PCP**: Mathmatical proofs that can be checked by only reading $O(1)$ bits of the proof!
- **PCP theorem**: *Every* language in NP has a PCP proof system!

Derandomization from Interactive Proofs

- **Hardness vs. randomness**: A classical framework for derandomization, turing a specific hard function into a pseudorandom generator.
- **Derandomization from interactive proofs**: Constructing such a generator from an interactive proof system, any hard function with such interactive proof system can be turned into a pseudorandom generator.
- **Constrution of Prime in Pseudo-deterministic Polynomial Time**: A beautiful application of the above.

Complexity of Modern AI Models

- Complexity theory always love to formalize whatever exciting new type of computation is out there and study them formally.
- **Transformers**: a new type of sequence models that are very powerful.
- **Complexity of Transformers**: what functions can be computed by Transformers? (constraints: depth, width, and number of attention heads)
- **Chain-of-Thought**: Transformers with a "chain-of-thought" can solve problems with "reasoning".
- **Complexity of Chain-of-Thought**: what functions can be computed by Chain-of-Thought? (constraints: length of the CoT)