Complexity Theory

Instructor: Roei Tell

Preliminaries

Bleeding edge technology

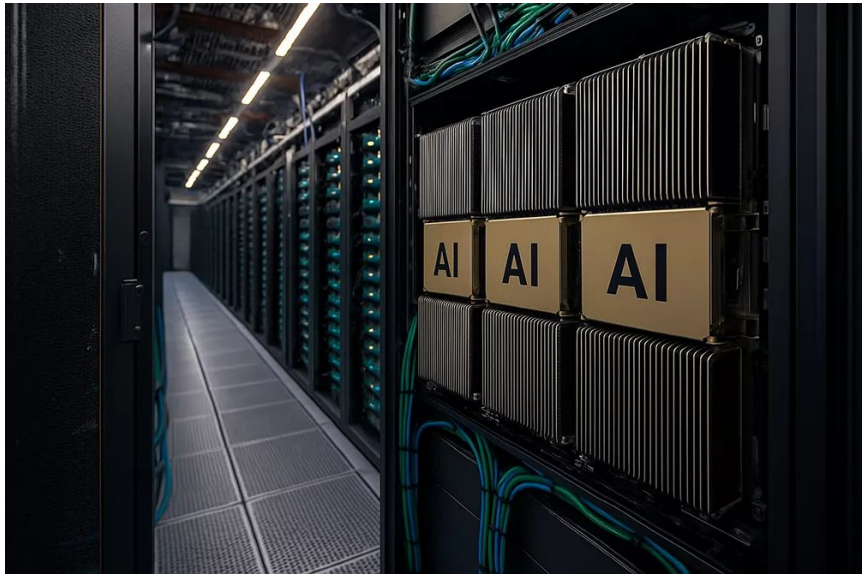
- Intel 80286
- CGA graphics
- Two floppy disks

Bleeding edge technology

- iPhone 1
- Tiny chips
- Cellular data

Bleeding edge technology

- Deep learning
- GenAI
- ???



Guiding question

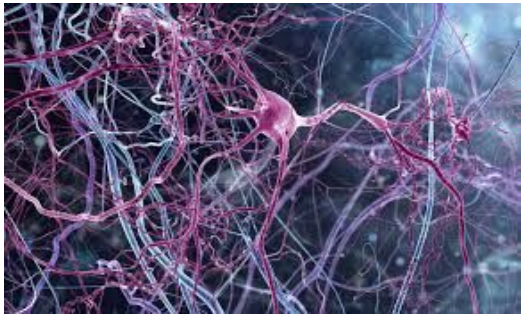
- How to argue about computation when technology changes quickly?

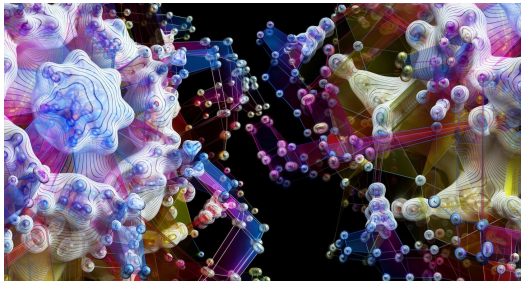
Theoretical computer science

- Understand the rules of computation in general, decoupled from any specific physical procedure.
- Study the capabilities and limitations of all possible algorithms, not only of Intel 80286 or of ChatGPT 5.2 .
- Basic science
  - analogy: theoretical physics
  - relationship with applications as any other basic science

Theoretical computer science

- understanding computation
- what rules govern all of these?





Course policies

- **Material delivered in class**

  Attendance is mandatory

- **Active learning**

  Questions bank, reading, thinking, discussing

- **Evaluation**

  Two midterms, exam

Syllabus

Resources

- **Information** in Quercus
- **Recommended textbooks**

  Introduction to the Theory of Computation (Sipser)

  Computational Complexity: A Conceptual Perspective (Goldreich)

  Computational Complexity: A Modern Approach (Arora & Barak)

Resources

- **Mental health** support

  9-8-8: Suicide Crisis Helpline

  844-451-9700: Telus Health Student Support (+ chat via web option)

  866-925-5454: Good2Talk (+ chat via web option)

  Health & Wellness Centre at 5th floor of the Sam Ibrahim building

  https://www.utsc.utoronto.ca/home/mental-health-resources

  https://mentalhealth.utoronto.ca/

  https://www.utsc.utoronto.ca/hwc/

  https://good2talk.ca/

Contact us

- Your TA
- Piazza
- Office hours
- Emailing instructor

What is complexity theory

Complexity theory

“The quest for efficiency is ancient and universal, as time and other resources are always in shortage. Thus, the question of which tasks can be performed efficiently is central to the human experience.”

(Goldreich, Computational Complexity: A Conceptual Perspective)

Complexity theory

- Main question:

  How do resource constraints (e.g., restricted time and memory) limit what algorithms can achieve?

- Core subfield of TCS

  (sibling to algorithm design, cryptography, data structures, …)

Complexity theory

- Methodology:

1. Introduce mathematical models for computation, which ideally capture any possible process.
2. Prove, mathematically, limitations for these models.
3. Conclude that these limitations govern any possible process.

Complexity theory

- Key parts:

  Understanding classical notions via the lens of efficient algorithms:

  - impossibility (aka “lower bounds”)
  - knowledge & information
  - randomness
  - interaction

Complexity theory

- Key parts:

  Understanding classical notions via the lens of efficient algorithms:

  - impossibility (aka “lower bou
  - knowledge & information
  - randomness
  - interaction

This is **not** a philosophy course! We are studying **computer science!**

Computability vs complexity

- Historical roots in **logic & computability** theory (1930s):

  What can be computed, regardless of resources?

- Introduction of **complexity theory** (1960s):

  What can be computed efficiently, i.e. with limited resources?

Lower bounds

Intuition

exam

1. Show a linear-time algorithm that finds the largest clique in a given graph

Intuition

exam

1. Show a linear-time algorithm that finds the largest clique in a given graph

hmmm, we don't know one?...

Intuition

exam

1. Show a linear-time algorithm that finds the largest clique in a given graph

- maybe one of the other students taking the exam solved it?...
- maybe one day somebody else will solve this?...

Intuition

exam

1. Show a linear-time algorithm that finds the largest clique in a given graph

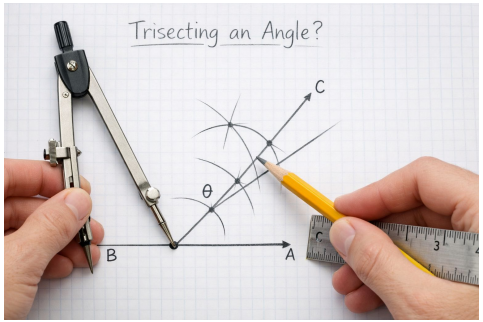
- how can we prove that there is no linear-time algorithm?
- we need to prove that something doesn't exist!

Intuition

exam

1. Show a linear-time algorithm that finds the largest clique in a given graph

- how can we prove that there is no linear-time algorithm?
- we need to prove that something doesn't exist!



Intuition

- Question [Cobham ‘64]:

  “Is it harder to multiply than to add?”

Intuition

- Question [Cobham ‘64]:

  “Is it harder to multiply than to add?”

- Best known multiplication algorithm is slower than best known addition algorithm, but maybe there’s a better one?

The notion of a lower bound

- A lower bound is a theorem of the form

  "Any computational process solving problem P must use at least T many steps"

The notion of a lower bound

- A lower bound is a theorem of the form

  "Any computational process solving problem P must use at least T many steps"

- What are we lower bounding?

The notion of a lower bound

- A lower bound is a theorem of the form

  "Any computational process solving problem P must use at least T many steps"

- What are we lower bounding?

  The amount of resources needed to solve problem P

  #steps ≥ T

The notion of a lower bound

› Differentiate between:

Complexity of a specific algorithm

vs

Complexity of a problem P ($= \min_{\text{A solves P}}\{\text{complexity of A}\}$)

The notion of a lower bound

- In particular,

  If we prove a very high lower bound on the number of steps needed to solve some problem P,

  for a convincingly general model of computation,

  then problem P cannot be efficiently solved, in any way.

The notion of a lower bound

- Understanding the limits of what can be efficiently solved
- Consequences:

  try to find ways around it (approximate, solve on average, etc.)

  give up, do something else

  turn lemons into lemonade (lower bounds ⇒ algorithms, crypto)

Toy examples

Toy example 1

- Comparison-based sorting (from CSCB63)
- Intuitive claim:

  “Sorting can’t be done in linear time by any algorithm working in a restricted model of computation”

  ⇒ model captures MergeSort, QuickSort, BubbleSort, …

  ⇒ lower bound true also for algorithms nobody thought of!

Toy example 1

- Problem:

  Given n distinct elements in [n] = { 1, ..., n }, output a permutation on [n] that sorts them

- Def:

  An algorithm is comparison-based if it makes a sequence of pairwise comparisons of input elements, and chooses a permutation based on to the outcomes of the comparisons.

Toy example 1

- Thm:

  Any comparison-based algorithm for sorting makes $\Omega(n \cdot \log n)$ comparisons.

Toy example 1

- Thm:

  Any comparison-based algorithm for sorting makes $\Omega(n \cdot \log n)$ comparisons.

- Pf reminder:

  1. There are n! different inputs, and each permutation can only sort one input correctly.
  2. k comparisons $\Rightarrow 2^k$ possible outputs, so we must have $k \geq \log(n!)$

Toy example 2

› Def:

A Disjunctive Normal Form (DNF) gets n input bits and computes an OR of ANDs over the bits and their negations.

For example:

$$F(x_1,\ldots,x_n) = (x_1 \wedge \neg x_2 \wedge x_n) \vee (x_3 \wedge x_4 \wedge \neg x_7 \wedge x_{n-1}) \vee (x_{12} \wedge x_{18})$$

Toy example 2

› Thm:

Any DNF computing $\sum_{i \in [n]} x_i \pmod 2$ has $\Omega(2^n)$ terms.

Toy example 2

› Thm:

Any DNF computing $\sum_{i \in [n]} x_i \pmod 2$ has $\Omega(2^n)$ terms.

› Pf:

Exercise in questions bank

Reminder

- These examples referred to restricted models of computation (lower bounds on comparisons, terms…)
- Our focus:
  - general models of computation
  - lower bounds on #steps of any possible algorithm