Introduction to Theoretical Computer Science

Lecture 10: (Polynomial) Complexity

Richard Mayr

University of Edinburgh

Semester 1, 2025/2026

Time Complexity

We have looked into whether problems can be computed or not.
But are they easy to compute or hard to compute?

Time Complexity

The *time complexity* of a (deterministic) machine M that halts on all inputs is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps that M uses on any input of size n.

Example

Example

Recall that $\{0^i 1^i \mid i \in \mathbb{N}\}$ is a CFL and decidable by, e.g., a TM M_1 that given input w:

1. Scan w and reject if anything not in $\{\sqcup, 0, 1\}$ or 10 is found.
2. While there are 0s and 1s left in the tape:
   - Scan across and replace with blanks both the leftmost 0 and the rightmost 1.
3. If any 0s or 1s are left on the tape, reject. Else, accept.

Time complexity measure:

w	ε	01	$0^2 1^2$	$0^3 1^3$	$0^4 1^4$	$0^5 1^5$
$f(\|w\|)$	2	8	19	34	53	76

Big Letters

Recall from previous courses...

Big O and Ω

Let $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Say that $f(n) \in \mathcal{O}(g(n))$ if there exists $c, n_0 > 0$ such that for all $n \geq n_0$:

$$f(n) \leq c \cdot g(n)$$

Similarly $f(n) \in \Omega(g(n))$ if:

$$f(n) \geq c \cdot g(n)$$

Example

- $f(n) = 5n^3 + 2n^2 + 22n + 6$ is $\in \mathcal{O}(n^3)$.
- M_1's complexity is $\mathcal{O}(n^2)$.

Logarithms

Recall that comparison-based sorting has $\Omega(n \log n)$ time complexity, and we have an $\mathcal{O}(n \log n)$ algorithm.

Omitting the bases

We may safely omit the base of the logarithms here because:

$$\log_a n = \frac{log_b n}{log_b a}$$

Model Concerns

Addition of two numbers is $\mathcal{O}(n)$ in our RM models.

Why is this bad?

In TMs, addition is $\mathcal{O}(\log n)$ (e.g. consider binary addition).
$\Rightarrow$ exponential penalty for RMs!

If we extend our RMs with ADD(i, j) SUB(i, j) which instantly add/subtract R_j from/to R_i, putting the result in R_i:

Less inaccurate.. but

Now addition is $\mathcal{O}(1)$ instead of $\mathcal{O}(\log n)$, but this is a smaller inaccuracy than the exponential penalty from before.

Variations in Models

Problem?

For sorting, we counted the number of comparisons as our time measure: we assumed comparison of small numbers and big numbers take the same time.

- What about control flow or memory access costs? In RMs this can be fast, but in TMs we have to move symbol by symbol.
- As we've seen, addition has different complexities based on the model.

Question

Can we ignore these differences? How?

What counts as different?

While complexity is useful, the measures are slightly bogus:

- If a problem is $\mathcal{O}(n)$ on some model, it's surely easy on any model.
  Not really: If n is a petabyte..
- If a problem is $\Omega(2^n)$ on some model, it's surely hard on any model.
  Actually: There are problems that are much worse than this, but still solvable for real examples. We'll see later.
- What about something that is $\mathcal{O}(n^{10})$ or $\Omega(n^{10})$?
  An $\Omega(n^{10})$ problem seems practically insoluble.
  However: Maybe a new algorithm or fancy model makes it $\Omega(n^2)$?
- There's also our coefficients. If $f(n) \geq 10^{100} \log n$, that's only $\mathcal{O}(\log n)$.
  However this isn't common.

Complexity Classes

Definition

Let $t : \mathbb{N} \to \mathbb{R}_{\geq 0}$. A *time complexity class* **TIME**$(t(n))$ to be the collection of all problems that are decidable by a machine in $\mathcal{O}(t(n))$ time.

We'll give a more precise definition of time in terms of bounded machines later.

Example

Recall $A = \{0^i 1^i \mid i \in \mathbb{N}\}$. Our TM M_1 can decide this in $\mathcal{O}(n^2)$. Therefore $A \in$ **TIME**(n^2).

Can we do better?

Can we come up with a machine M_2 that shows A is in **TIME**$(t(n))$ for some $t(n)$ that is asymptotically $< n^2$?

Example

Given w input:

1. Scan w left to right and reject if `10` is found.
2. Repeat as long as there are `0`s and `1`s on the tape:
   1. Scan from right to left and reject if there is an odd number of non-`X`s on the tape.
   2. Scan from left to right and replace every other `0` by an `X`, beginning from the first `0`. Then, do the same for `1`s.
3. If neither `0`s nor `1`s are left, accept. Else, reject.

Steps 1, 2.1, 2.2, and 3 are all $\mathcal{O}(n)$. Step 2 runs the substeps $\mathcal{O}(\log n)$ times. So this is $\mathcal{O}(n \log n)$.

Comparing real times

Comparing the running times of M_1 and M_2:

w	ε	01	$0^2 1^2$	$0^3 1^3$	$0^4 1^4$	$0^5 1^5$
$f_{M_1}(\lvert w\rvert)$	2	8	19	34	53	76
$f_{M_2}(\lvert w\rvert)$	1	15	45	63	117	141

M_2 has “better” complexity, but M_1 performs better for small n.
(M_2 will be faster for $0^{20}1^{20}$.)

Doing better

Could we do still better for A? I.e. a sub-$\mathcal{O}(n \log n)$ algorithm for A?

Example (Two tape TMs)

The answer is no, for a single-tape TM. But in a two tape TM, we can copy all 0s onto the second tape and then compare the number of 0s to 1s by moving the second tape head synchronously with the first.

Polynomial Time

Definition

$$\mathbf{P} = \bigcup_{k \in \mathbb{N}} \mathbf{TIME}(n^k)$$

That is, the class of problems decidable with some (deterministic) polynomial time complexity.

- Problems in **P** are called *tractable*.
- The class is robust: “Reasonable” changes in model don’t change it, and “reasonable” translations between problems preserve membership in **P**.
- Any problem not in **P** is $\Omega(n^k)$ for every k, e.g. 2^n or $2^{\sqrt{n}}$.

Outside P

Definition

A *polynomially-bounded RM* is an RM together with a polynomial (wlog n^k for some k), such that given an input w, it will always halt after executing $|w|^k$ instructions.

A problem Q is in **P** iff it is computed by polynomially-bounded RM.

Polynomial Reductions

Recall:
To prove that a problem P_2 is *hard*, show that there is an *easy* reduction from a known hard problem P_1 to P_2.

Definition

A *polynomial reduction* from $P_1 = (D_1, Q_1)$ to $P_2 = (D_2, Q_2)$ is a **P**-computable function $f : D_1 \to D_2$ such that $d \in Q_1$ iff $f(d) \in Q_2$.

- If P_2 is in **P**, then P_1 is in **P** straightforwardly.
- Therefore: To prove that a problem P_2 is not in **P**, show that there is a polynomial reduction from a known non-**P** problem P_1 to P_2.

Question: Is this more like a mapping or Turing reduction?

Apparently Intractable Problems

These problems appear to be non-**P**, so if they are, we could use them as our known non-**P** problems.

Example (Hamiltonian Path Problem)

Given a graph $G = (V, E)$, is there a path that visits every vertex in V exactly once?
We could solve this in $\mathcal{O}(|V|!)$, but this is not ideal..

Example (Timetabling)

Given students taking exams, and timetable slots for exams, is it possible to schedule the exams so that there are no clashes? It also apparently requires looking at exponentially many possible assignments.

(That's why Registry starts timetabling exams 9 weeks in advance...)

Open problem: Are they really not in **P**?

Checking

Consider *HPP* (the Hamiltonian Path Problem) or timetabling. Both are apparently not in **P**.

However..

They are easy to check:
Given a claimed solution, it's tractable to check if the solution is indeed a correct solution.

Theorem

Any problem that can be checked in polynomial time on a deterministic RM/TM can be computed in polynomial time on a *nondeterministic RM/TM*.

Nondeterminism

We can have nondeterministic RMs just like we have nondeterministic finite automata.

The Change

Add a special instruction MAYBE(j) that will nondeterministically either do nothing or jump to I_j.

Example (generating a nondetermined number)

	CLEAR	R_0
beg :	MAYBE	*end*
	INC	0
	GOTO	*beg*
end :		

Non-nondeterminism

Acceptance

An NRM accepts if there is *some run* (sequence of instructions through the choices) halts and accepts.
"Accepts" could mean halting, halting with 1 in R_0 or anything else.

- Nondeterminism is **NOT** probability. No randomness is involved.
- The presence of infinite runs doesn't matter if there are also accepting finite runs.
- I sometimes like to think of MAYBE as FORK: the machine forks a copy of itself which takes the jump. If any copy accepts, it signals the OS, which kills off all the others.

Question: Do NRMs have the same deciding power as RMs?

Comparing RMs and NRMs

Power

NRMs have the same deciding power as RMs, because we can use the interleaving technique to simulate all runs of an NRM.

Sipser has the same result for TMs.

However!

In time n, an RM can explore only $\mathcal{O}(n)$ possibilities, but an NRM can explore $2^{\mathcal{O}(n)}$ possibilities.

NRMs are potentially exponentially faster than RMs.

NP

Definition

Let $t : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Define $\textbf{NTIME}(t(n))$ to be the collection of all problems that are decidable by an NRM in $\mathcal{O}(t(n))$ time.

Definition

$$\textbf{NP} = \bigcup_{k \in \mathbb{N}} \textbf{NTIME}(n^k)$$

That is, the class of problems decidable with some nondeterministic polynomial time complexity.

Is *HPP* in **NP**? Nondeterministically “guess” any path and check if it is Hamiltonian ($\mathcal{O}(n)$).

A Short Aside

Can we implement nondeterminism or is it just a theoretical exercise?

Quantum Computing

Quantum computers can achieve a similar effect: an n-qubit computer computes on all 2^n values simultaneously.
However, one cannot really access all these 2^n values in an arbitrary way.
Not every **NP** algorithm is quantum-computable (as far as we know).
Not every problem (e.g., odd/even parity of bitstrings) can be solved faster by quantum computers.
Moreover, in practice it is hard to get many qubits..

Is NP All?

Is every exponentially-bounded problem in **NP**? probably No!

Tough problem

Given a machine M and input w, determine if M halts in less than $2^{|w|}$ steps.
There doesn't seem to be anything to do but run the machine M for an exponential number of steps $\Rightarrow$ *Probably* not in **NP**.