BU CS 332 – Theory of Computation

https://forms.gle/gUYdeMLDG4DWRLEVA

Lecture 19:

- Complexity Class P
- Complexity Class NP

Reading:

Sipser Ch 7.2-7.3

Alexander Poremba & Mark Bun

April 14, 2026

Complexity Recap

1. Asymptotic notation

2. How do we measure complexity?

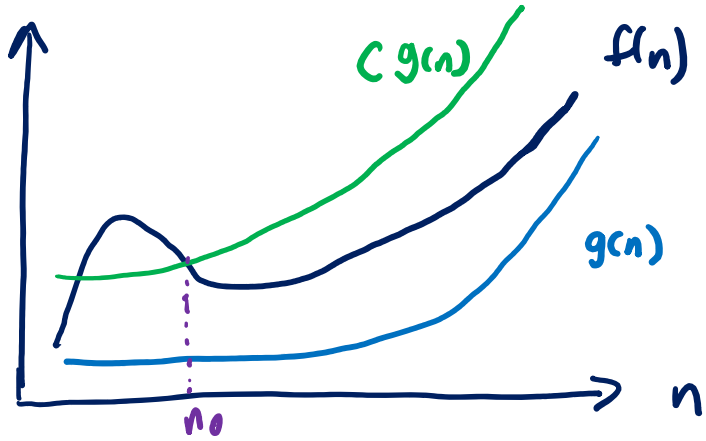
3. How robust is the TM model when we care about measuring complexity?

4. How do we mathematically capture our intuitive notion of “efficient algorithms”?

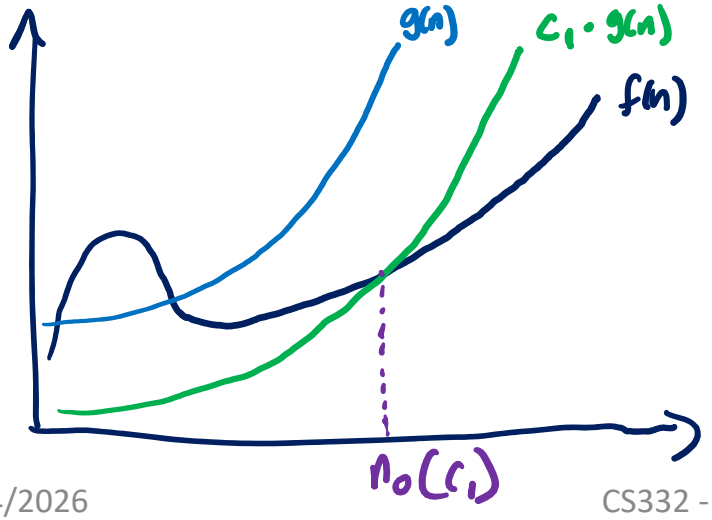
Complexity Recap

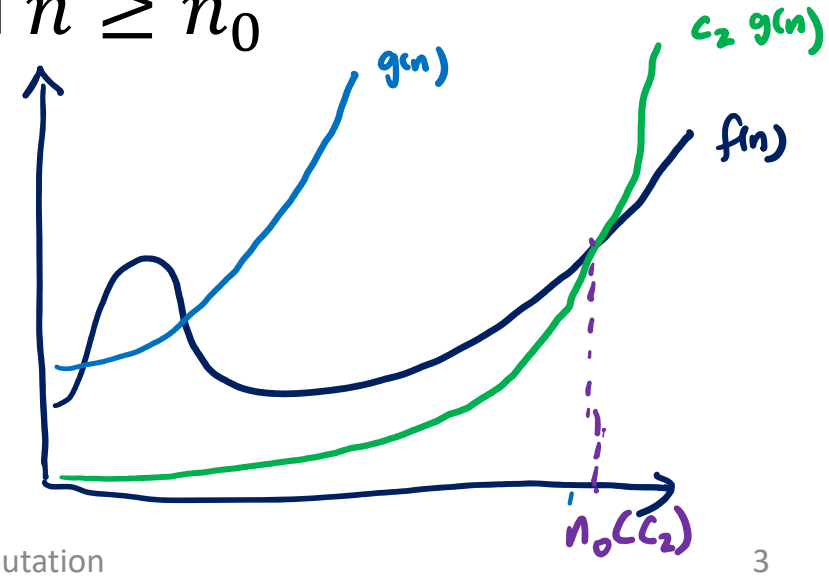
1. Asymptotic notation

Big-Oh: $f(n) = O(g(n))$ if there exist c, n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$



Little-Oh: $f(n) = o(g(n))$ if for every $c > 0$ there exists n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$





Complexity Recap

2. How do we measure complexity?

A TM M runs in time $f(n)$ if for every n and every input $w \in \Sigma^n$, M halts on w within at most $f(n)$ steps

A language $A \in \mathrm{TIME}(f(n))$ if there exists a basic single-tape (deterministic) TM M that 1) Decides A, and

2) Runs in time $O(f(n))$

A TM M runs in space $f(n)$ if for every n and every input $w \in \Sigma^n$, M halts on w using at most $f(n)$ tape cells

A language $A \in \mathrm{SPACE}(f(n))$ if there exists a basic single-tape (deterministic) TM M that 1) Decides A, and

2) Runs in space $O(f(n))$

Complexity Recap

3. How robust is the TM model for measuring complexity?

The machine model matters:

There exists a language decidable in time $O(n)$ on a two-tape TM, but which cannot be decided in $o(n \log n)$ time on a basic single tape TM

But not too much:

Theorem: Every multi-tape TM running in time $t(n)$ has an equivalent single-tape TM running in time $O(t(n)^2)$

Extended Church-Turing Thesis:

Every "reasonable" (physically realizable) model of computation can be simulated by a basic, single-tape TM with only a **polynomial** slowdown.

Complexity Recap

3. How robust is the TM model for measuring complexity?

Hierarchy Theorems: (Asymptotically) more resources allow TMs to solve strictly more problems

- For every time-constructible function $t(n) \geq n \log n$, there exists a language decidable in $t(n)$ time, but not in $o\left(\frac{t(n)}{\log t(n)}\right)$ time.

$\exists L$ s.t. $L \in TIME(t(n))$ but $L \notin TIME(f(n))$ for any $f(n) = o\left(\frac{t(n)}{\log t(n)}\right)$

- For every space-constructible function $s(n) \geq \log n$, there exists a language decidable in $s(n)$ space, but not in $o(s(n))$ space.

Complexity Recap

1. Asymptotic notation

2. How do we measure complexity?

3. How robust is the TM model when we care about measuring complexity?

4. How do we mathematically capture our intuitive notion of “efficient algorithms”?

Complexity class P

Definition: P is the class of languages decidable in polynomial time on a basic single-tape (deterministic) TM

$$\mathrm{P} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(n^k) = \{L \mid \exists \text{ TM deciding } L \text{ running in some polynomial time}\}$$

$$= \mathrm{TIME}(n) \cup \mathrm{TIME}(n^2) \cup \mathrm{TIME}(n^3) \cup \ldots$$

- Class doesn't change if we substitute in another reasonable deterministic model (Extended Church-Turing)
- Cobham-Edmonds Thesis: Roughly captures class of problems that are feasible to solve on computers

Check your type checker: P

Variants that are decision problems:

1) VERIFY_ADD = $\{\langle x, y, z\rangle \mid x + y = z\}$

2) BIT_ADD = $\{\langle x, y, i\rangle \mid$ i'th bit of $x+y$ is 1 $\}$

(index into bit representation of sum

Consider the following computational problem: Given two numbers x, y (written in binary), output their sum $x + y$ (in binary). Which of the following is true?

a) This is a problem in P

b) This problem is not in P because it cannot be solved by a Turing machine (i.e., it's undecidable)

c) This problem is not in P because it cannot be solved in polynomial time

d) This problem is not in P because it is not a decision problem (i.e., does not correspond to a language)

A note about encodings

We'll still use the notation $\langle \quad \rangle$ for "any reasonable" encoding of the input to a TM…but now we have to be more careful about what we mean by "reasonable"

Adj. matrix $V \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & & \\ 0 & & \end{pmatrix}$ V

Adj. list $V \{ v_7 \; v_{24} \; v_3 \; \cdots$; $v_8 \; v_{16} \; \cdots$

How long is the encoding of a V-vertex, E-edge graph…

… as an adjacency matrix? $O(V^2)$

… as an adjacency list? $O(V + E \log V)$ } same up to polynomial factor

How long is the encoding of a natural number k

… in binary? $n = |\langle k \rangle| \approx \log_2 k$ $(\lfloor \log_2 k \rfloor + 1)$ } same up to constant factor

… in decimal? $n = |\langle k \rangle| \approx \log_{10} k = \frac{1}{\log_2 10} \cdot \log_2 k$

… in unary? $n = |\langle k \rangle| = k$

exponentially longer encoding

$\langle k \rangle = 11\underbrace{1 \ldots 1}_{k}$

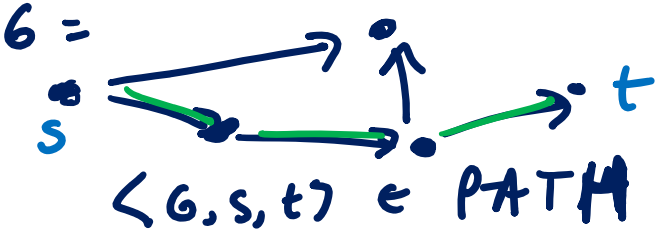
Describing and analyzing polynomial-time algorithms

- Due to Extended Church-Turing Thesis, we can still use high-level descriptions on multi-tape machines
- Polynomial-time is robust under composition: $\text{poly}(n)$ executions of $\text{poly}(n)$-time subroutines run on $\text{poly}(n)$-size inputs gives an algorithm running in $\text{poly}(n)$ time.

  $\Rightarrow$ Can freely use algorithms we've seen before as subroutines if we've analyzed their runtime

- Need to be careful about size of inputs! (Assume inputs represented in binary unless otherwise stated.)

Examples of languages in P



$PATH =$
$\{\langle G, s, t\rangle \mid G \text{ is a directed graph with a directed path from } s \text{ to } t\} \in P$

Idea: Breadth-first search

Assume G presented as adjacency matrix

$O(|V|^2)$ $\log_2|V|$ $\log_2|V|$ Input length $n = |\langle G, s, t\rangle|$
$= O(|V|^2) + O(\log_2|V|) = O(|V|^2)$

"On input $\langle G, s, t\rangle$:

1. Mark start vertex s $O(|V|^2)$
2. For $i = 1, 2, \ldots, |V|$: $O(|V|)$ loop iterations
3. Mark all neighbors of currently marked vertices $O(|V|^2)$
4. If t is marked, accept. Else, reject." $O(|V|^2)$

Runtime: $O(|V|^2) + O(|V|) \cdot O(|V|^2) + O(|V|^2) = O(|V|^3) = O(n^{3/2})$

Examples of languages in P

$E_{\text{DFA}} = \{\langle D \rangle \mid D \text{ is a DFA that recognizes the empty language}\}$

$\in P$

Idea: Reduce to PATH

On input $\langle D \rangle$:

1. Let G = graph underlying state diagram of D
   s = start vertex of D
2. For each accept state t of D:
   a) Run alg. for PATH on input $\langle G, s, t \rangle$
   b) If accepts, reject. Else, continue
3. Accept.

Examples of languages in P

- $RELPRIME = \{\langle x, y \rangle \mid x \text{ and } y \text{ are relatively prime}\}$

Euclidean algorithm

- $PRIMES = \{\langle x \rangle \mid x \text{ is prime}\}$

2006 Gödel Prize citation



The 2006 Gödel Prize for outstanding articles in theoretical computer science is awarded to Manindra Agrawal, Neeraj Kayal, and Nitin Saxena for their paper "PRIMES is in P."

In August 2002 one of the most ancient computational problems was finally solved….

A polynomial-time algorithm for $PRIMES$?

Consider the following algorithm for $PRIMES$

On input $\langle x \rangle$:

For $b = 2, 3, 4, 5, \ldots, \sqrt{x}$:

- Try to divide x by b
- If b divides x, reject

If all b fail to divide x, accept

Worst case # Divisions: $\sqrt{x}$

$= \sqrt{2^n} = 2^{n/2}$

$n = |\langle x \rangle| \approx \log_2 x$

$\Leftrightarrow x \approx 2^n$

How many divisions does this algorithm require in terms of $n = |\langle x \rangle|$?

a) $O(\sqrt{n})$ b) $O(n)$ c) $2^{O(\sqrt{n})}$ d) $2^{O(n)}$

Beyond polynomial time

Definition: EXP is the class of languages decidable in exponential time on a basic single-tape (deterministic) TM

$$\mathrm{EXP} = \bigcup_{k=1}^{\infty} \mathrm{TIME}(2^{n^k})$$ = TIME(2^n) ∪ TIME(2^{n^2}) ∪ TIME(2^{n^3}) ∪ ...

Why study P ?

Criticism of the Cobham-Edmonds Thesis:

- Algorithms running in time n^{100} aren't really efficient

  Response: Runtimes improve with more research

- Does not capture some physically realizable models using randomness, quantum mechanics

  Response: Randomness may not change P, useful principles

$TIME(n)$ vs. $TIME(n^2)$

P vs. EXP

decidable vs. undecidable

Nondeterministic Time and NP

Extended Church-Turing Thesis

Every “reasonable” (physically realizable) model of computation can be simulated by a basic, single-tape TM with only a **polynomial** slowdown.

E.g., doubly infinite TMs, multi-tape TMs, RAM TMs

Does not include nondeterministic TMs (not reasonable)

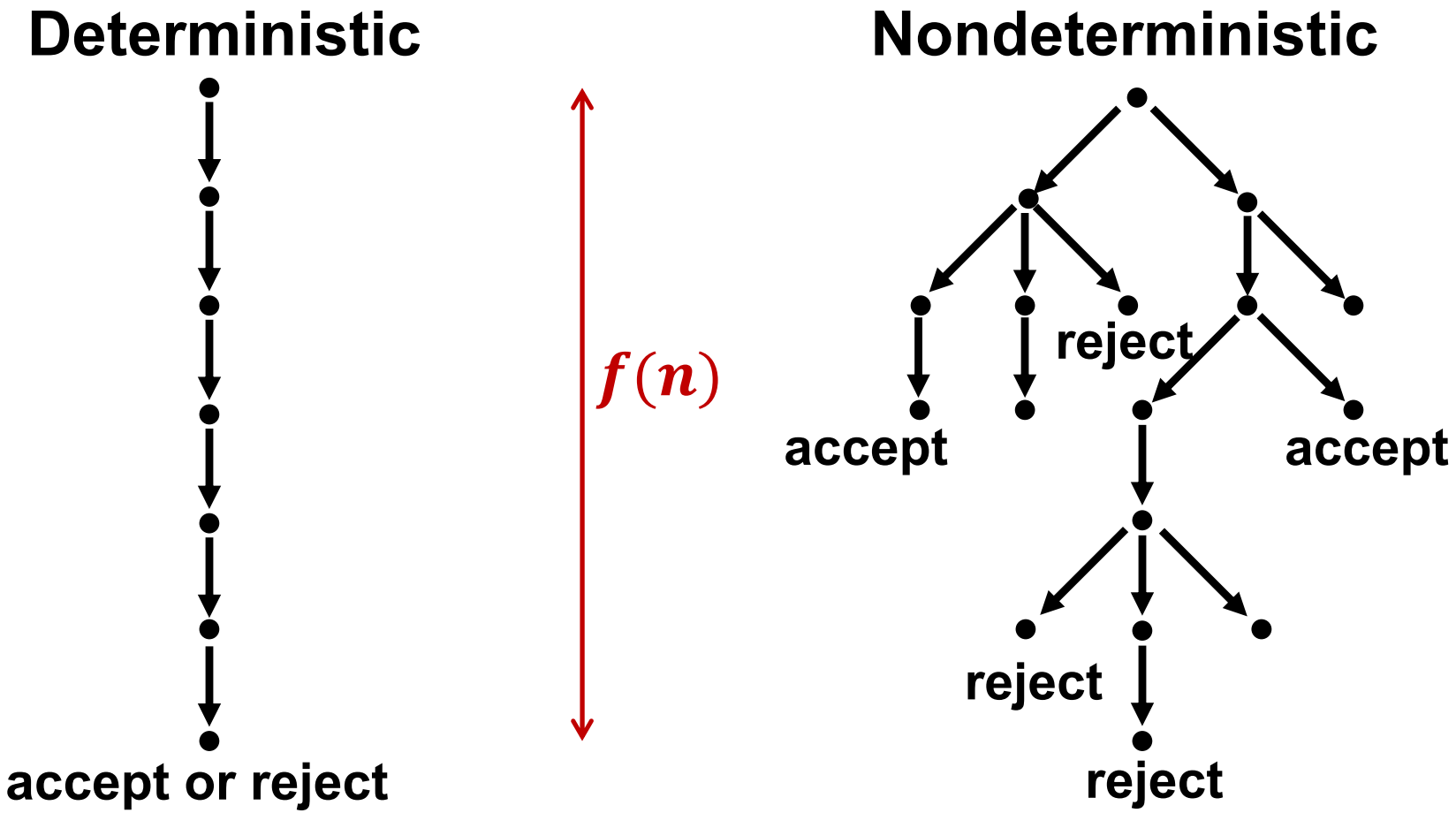
Nondeterministic time

Let t: $\mathbb{N} \rightarrow \mathbb{N}$

NTM M runs in time $t(n)$ if:

For every n and every input $w \in \Sigma^n$, M halts on w within at most $t(n)$ steps on every computational branch

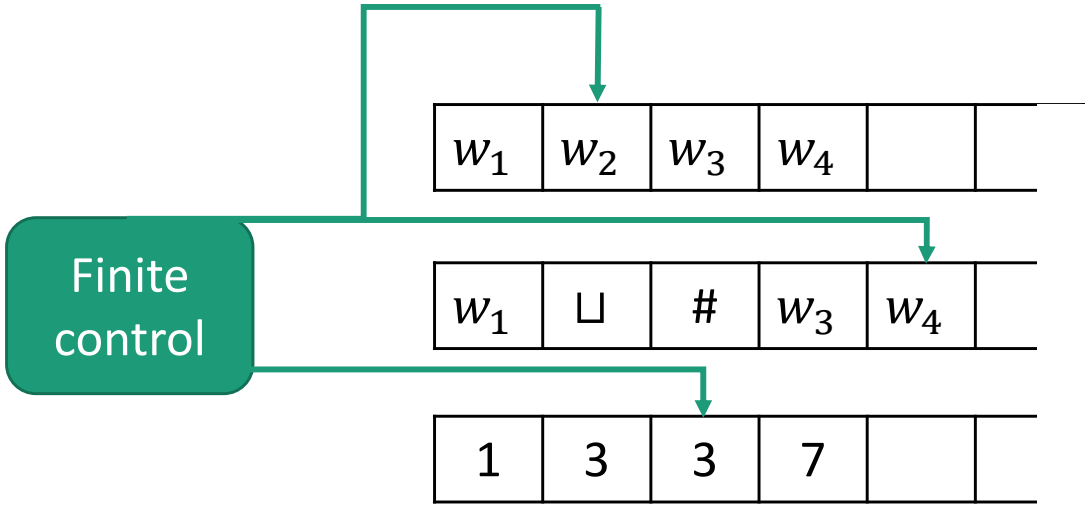
Deterministic vs. nondeterministic time



Deterministic vs. nondeterministic time

Theorem: Let $t(n) \geq n$ be a function. Every NTM running in time $t(n)$ has an equivalent deterministic single-tape TM running in time $2^{O(t(n))}$

Proof: Simulate NTM by 3-tape TM



Input w to N (read-only)

Simulation tape (run N on w using nondeterministic choices from tape 3)

Address in computation tree

Counting leaves

What is an upper bound on the maximum number of nodes in a tree with branching factor b and depth t?

a) bt

b) b^t

c) t^b

d) 2^t