

BU CS 332 – Theory of Computation

<https://forms.gle/85nvTEZX5hhHNibT6>



Lecture 17:

- Time/Space Complexity
- Time/Space Hierarchies
- Complexity Class P

Reading:

Sipser Ch 7.1-2, 8.0, 9.1

Mark Bun & Alexander Poremba

April 2, 2026

Time and space complexity

The basic questions

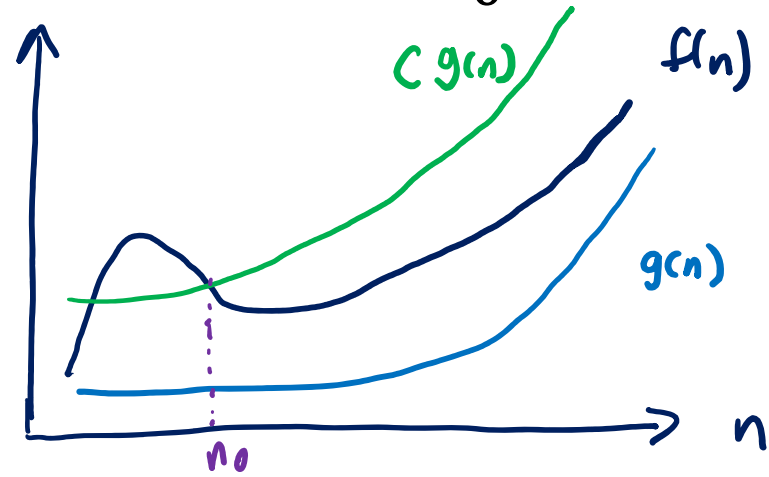
1. How do we measure complexity?
2. Asymptotic notation
3. How robust is the TM model when we care about measuring complexity?
4. How do we mathematically capture our intuitive notion of “efficient algorithms”?

Last Time: Asymptotic Notation

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

Last Time: Asymptotic Notation

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

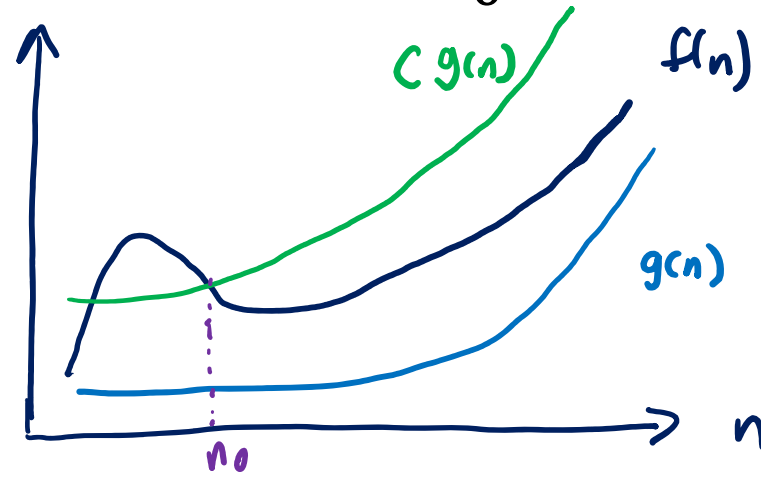


Last Time: Asymptotic Notation

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

↙ " $f \leq g$ "

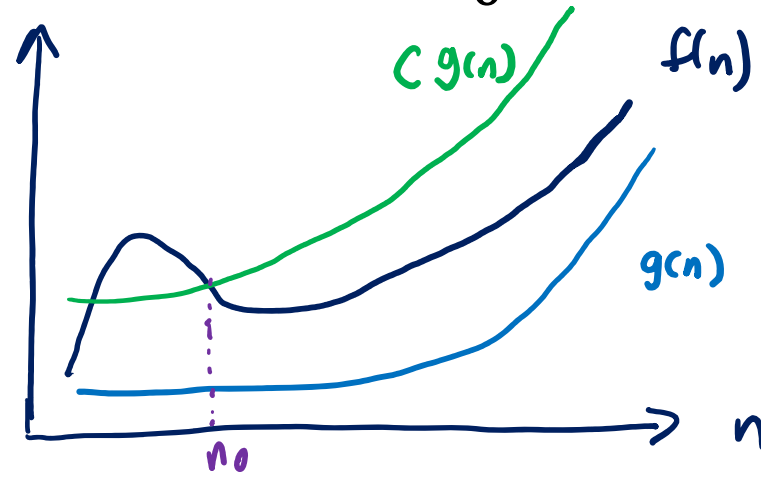
↙ " $f < g$ "



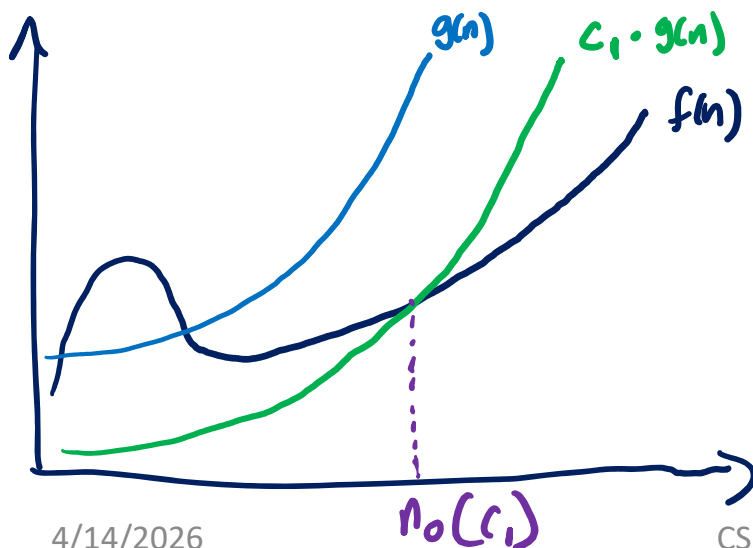
Little-Oh: $f(n) = o(g(n))$ if for every c there exists n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$

Last Time: Asymptotic Notation

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



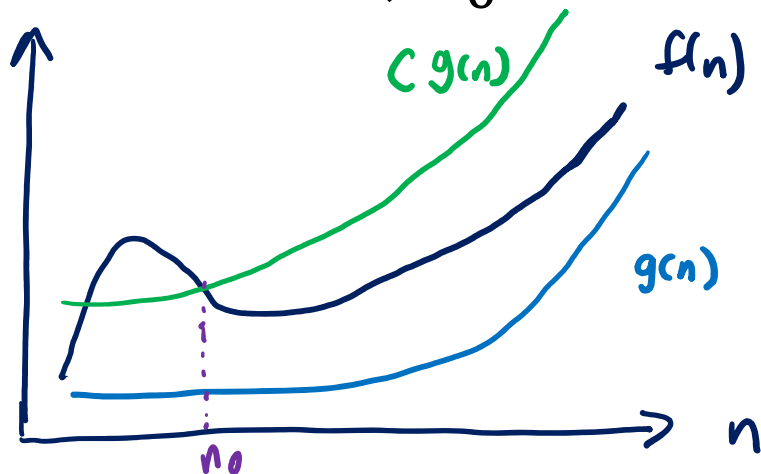
Little-Oh: $f(n) = o(g(n))$ if for every c there exists n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$



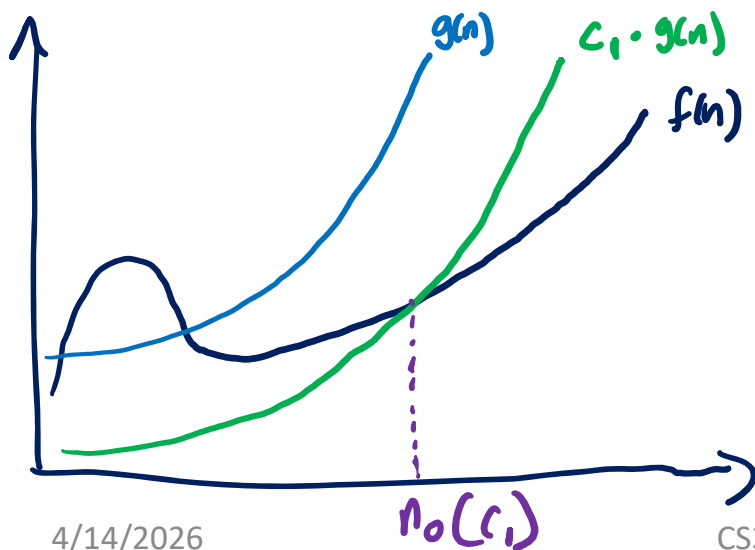
Last Time: Asymptotic Notation

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

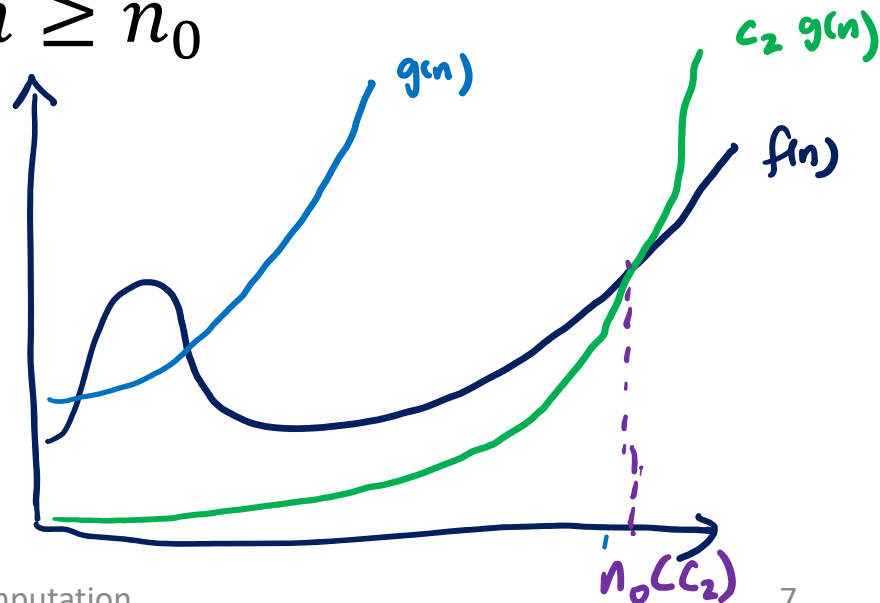
" $f \leq g$ "



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 $n \geq n_0$



" $f < g$ "



Time complexity

Time complexity of a TM (algorithm) = maximum number of steps it takes on a worst-case input

runtime bound
input length
steps

Formally: Let $f : \mathbb{N} \rightarrow \mathbb{N}$. 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 $A \in \text{TIME}(f(n))$ if there exists a basic single-tape (deterministic) TM M that

- 1) Decides A , and
- 2) Runs in time $O(f(n))$

Ex: $\text{TIME}(n^2)$
 $= \{ A \mid \text{language } A \text{ is decidable in quadratic time} \}$

Time class containment

" $f \leq g$ "

If $f(n) = O(g(n))$, then which of the following statements is always true?



- a) $\text{TIME}(f(n)) \subseteq \text{TIME}(g(n))$
- b) $\text{TIME}(g(n)) \subseteq \text{TIME}(f(n))$
- c) $\text{TIME}(f(n)) = \text{TIME}(g(n))$
- d) None of the above

Proof sketch:
 $A \in \text{TIME}(f(n))$
 $\Rightarrow \exists \text{ TM } M$ deciding A
in time $O(f(n))$.
 $\Rightarrow M$ decides A in time
 $O(g(n))$ because
 $f(n) = O(g(n))$.
 $\Rightarrow A \in \text{TIME}(g(n))$.

$\text{TIME}(f(n)) = \{ A \mid \text{language } A \text{ is decidable in } O(f(n)) \text{ time} \}$

Example

~~0~~~~0~~~~0~~~~0~~ ~~1~~~~1~~~~1~~~~1~~

$$A = \{0^m 1^m \mid m \geq 0\}$$

$M =$ "On input w :

1. Scan input and reject if not of the form $0^* 1^*$ $\} O(n)$

loops
 $O(n)$

2. While input contains both 0's and 1's:

Cross off one 0 and one 1 $\} O(n)$

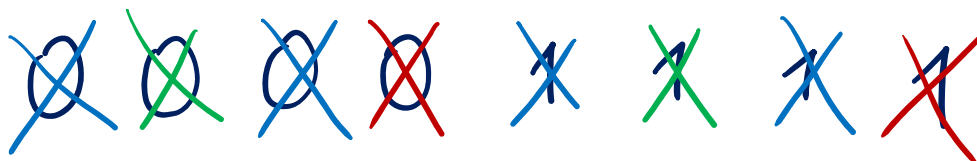
3. **Accept** if no 0's and no 1's left. Otherwise, **reject**."

$$M \text{ runs time: } O(n) + O(n) \cdot O(n) = O(n^2)$$

$$\Rightarrow \{0^m 1^m \mid m \geq 0\} \in \text{TIME}(n^2).$$

Is there a faster algorithm? -

Example



$$A = \{0^m 1^m \mid m \geq 0\}$$

M' = "On input w :

1. Scan input and reject if not of the form $0^* 1^*$ } $O(n)$

loops:
 $O(\log n)$

→ 2. While input contains both 0's and 1's:

- $O(n)$ {
- **Reject** if the total number of 0's and 1's remaining is odd
 - Cross off every other 0 and every other 1

3. **Accept** if no 0's and no 1's left. Otherwise, **reject**."

$$\begin{aligned} \text{Running time of } M' : & O(n) + O(\log n) \cdot O(n) \\ & = O(n \log n). \end{aligned}$$

$$\Rightarrow A \in \text{TIME}(n \log n).$$

Is there a faster algorithm?

Example

Running time of M' : $O(n \log n)$

Theorem (Sipser, Problem 7.49): If L can be decided in $o(n \log n)$ time on a basic single-tape TM, then L is regular

$A = \{0^m 1^m \mid m \geq 0\}$ is a non-regular language!

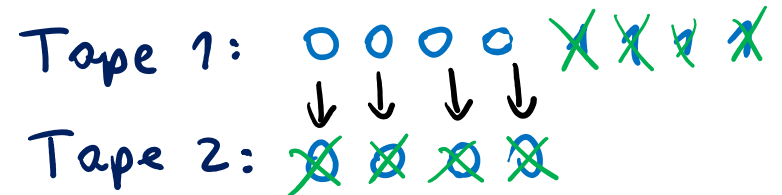
$\Rightarrow A \notin \text{TIME}(f(n))$ for any $f(n) = o(n \log n)$.

Fact: If L is regular $\Rightarrow L$ is decidable
in time $O(n)$

Does it matter that we're using the 1-tape model for this result?

It matters: 2-tape TMs can decide A faster

$M'' =$ "On input w :



- $O(n)$ {
1. Scan input and reject if not of the form 0^*1^*
 2. Copy 0's to tape 2
- $O(n)$ {
3. Scan tape 1. For each 1 read, cross off a 0 on tape 2
 4. If 0's on tape 2 finish at same time as 1's on tape 1, **accept**.
Otherwise, **reject**."

Total: $O(n)$

Analysis: A is decided in time $O(n)$ on a 2-tape TM

Moral of the story (part 1): Unlike decidability, time complexity depends on the TM model

How *much* does the model matter?

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

Proof idea:

We already saw how to **simulate** a multi-tape TM with a single-tape TM

Need a runtime analysis of this construction

Moral of the story (part 2): Time complexity doesn't depend too much on the TM model (as long as it's deterministic, sequential)

Single vs. Multi-Tape

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

Suppose B is decidable in time $O(n^2)$ on a 42-tape TM. What is the best upper bound you can give on the runtime of a basic single-tape TM deciding B ?

- a) $O(n^2)$
- b) $O(n^4)$
- c) $O(n^{84})$
- d) $2^{O(n)}$

$$t(n) = O(n^2)$$

$$O(t(n)^2) = O(n^4)$$



Single vs. Multi-Tape

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

Proof idea:

We already saw how to **simulate** a multi-tape TM with a single-tape TM

Need a runtime analysis of this construction

Simulating Multiple Tapes

(Implementation-Level Description)

On input $w = w_1 w_2 \dots w_n$

1. Format tape into $\# w_1 w_2 \dots w_n \# \dot{\square} \# \dot{\square} \# \dots \#$

2. For each move of M :

Scan left-to-right, finding current symbols

Scan left-to-right, writing new symbols,

Scan left-to-right, moving each tape head

If a tape head goes off the right end, insert blank

If a tape head goes off left end, move back right

Single vs. Multi-Tape

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

Proof: Time analysis of simulation

- Time to initialize (i.e., format tape): $O(n + k)$
- Time to simulate one step of multi-tape TM: $O(k \cdot t(n))$

- Number of multi-tape steps to simulate: $t(n)$

⇒ Total time:
$$\underbrace{O(n+k)}_{\text{initialization}} + \underbrace{t(n)}_{\text{\# steps of simulation}} \cdot \underbrace{O(k \cdot t(n))}_{\text{time per simulation step}} = O(k \cdot t(n)^2 + n + k) = O(t(n)^2)$$

constant! ↓

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)

Possible counterexamples? Randomized computation, parallel computation, DNA computing, quantum computation

Space complexity

Space complexity of a TM (algorithm) = maximum number of tape cells it uses on a worst-case input

Formally: Let $f : \mathbb{N} \rightarrow \mathbb{N}$. 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 $A \in \text{SPACE}(f(n))$ if there exists a basic single-tape (deterministic) TM M that

- 1) Decides A , and
- 2) Runs in space $O(f(n))$

How does space relate to time?



Which of the following is true for every function $f(n) \geq n$?

- a) $TIME(f(n)) \subseteq SPACE(f(n))$
- b) $SPACE(f(n)) \subseteq TIME(f(n))$
- c) $TIME(f(n)) = SPACE(f(n))$
- d) None of the above

Proof sketch:

$A \in TIME(f(n))$

$\Rightarrow \exists$ TM M deciding A
in $O(f(n))$ steps

$\Rightarrow M$ decides A in
space $O(f(n))$

$\Rightarrow A \in SPACE(f(n)).$

Back to our example

$$A = \{0^m 1^m \mid m \geq 0\}$$

M = "On input w :

1. Scan input and reject if not of the form $0^* 1^*$
2. While input contains both 0's and 1's:
Cross off one 0 and one 1
3. **Accept** if no 0's and no 1's left. Otherwise, **reject**."

$$A \in \text{SPACE}(n)$$

Back to our example

$$A = \{0^m 1^m \mid m \geq 0\}$$

M = “On input w :

1. Scan input and reject if not of the form $0^* 1^*$
2. While input contains both 0's and 1's:
Cross off one 0 and one 1
3. **Accept** if no 0's and no 1's left. Otherwise, **reject**.”

Theorem: Let $s(n) \geq n$ be a function. Every multi-tape TM running in space $s(n)$ has an equivalent single-tape TM running in space $O(s(n))$

Hierarchy Theorems

More time, more problems

We know, e.g., that $TIME(n^2) \subseteq TIME(n^3)$

(Anything we can do in quadratic time we can do in cubic time)

More time, more problems

We know, e.g., that $TIME(n^2) \subseteq TIME(n^3)$

(Anything we can do in quadratic time we can do in cubic time)

Question: Are there problems that we can solve in cubic time that we cannot solve in quadratic time?

More time, more problems

We know, e.g., that $TIME(n^2) \subseteq TIME(n^3)$

(Anything we can do in quadratic time we can do in cubic time)

Question: Are there problems that we can solve in cubic time that we cannot solve in quadratic time?

Theorem: There is a language $L \in TIME(n^3)$,
but $L \notin TIME(n^2)$

“Time hierarchy”:

$TIME(n) \subsetneq TIME(n^2) \subsetneq TIME(n^3) \subsetneq TIME(n^4) \dots$

An explicit separating language

Theorem: $L = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle \text{ within } n^{2.5} \text{ steps}\}$

is in $TIME(n^3)$, but not in $TIME(n^2)$

Proof Sketch: In $TIME(n^3)$

On input $\langle M \rangle$:

1. Simulate M on input $\langle M \rangle$ for $n^{2.5}$ steps
2. If M accepts, **reject**. If M rejects or did not yet halt, **accept**.



An explicit separating language

Theorem: $L = \{\langle M \rangle \mid M \text{ is a TM that does not accept input } \langle M \rangle \text{ within } n^{2.5} \text{ steps}\}$

is in $TIME(n^3)$, but not in $TIME(n^2)$

Proof Sketch: Not in $TIME(n^2)$

what if we feed $\langle D \rangle$ as input to D ?

Suppose for contradiction that D decides L in time $O(n^2)$

There are two cases:

- 1) D accepts $\langle D \rangle \Rightarrow D$ accepts $\langle D \rangle$ within $O(n^2) < n^{2.5}$
many steps
 $\Rightarrow \langle D \rangle \notin L \quad \downarrow$
- 2) D rejects $\langle D \rangle \Rightarrow D$ does not accept $\langle D \rangle$ within $O(n^2)$ steps.
 $\Rightarrow \langle D \rangle \in L \quad \downarrow$

Time and space hierarchy theorems

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

*“time constructible” and “space constructible”, respectively

Complexity Class P

Time and space complexity

The basic questions

1. How do we measure complexity?
2. Asymptotic notation
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

$$P = \bigcup_{k=1}^{\infty} \text{TIME}(n^k)$$

- 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

A note about encodings

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

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

... as an adjacency matrix?

... as an adjacency list?

How long is the encoding of a natural number k

... in binary?

... in decimal?

... in unary?

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.
 - ⇒ 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.)