

# Introduction to P vs. NP

Mika Göös



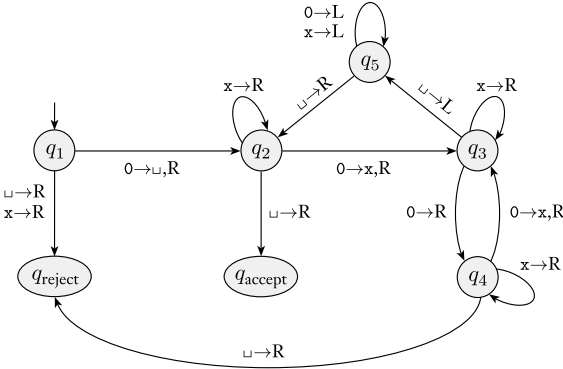
School of Computer and Communication Sciences

Intro Lecture, 11.09.2025

# Model of computation

(and why it doesn't matter)

# Remember Turing machines?



*Intuitive notion  
of algorithms*

equals

*Turing machine  
algorithms*

- ▶ All algorithms we know of can be executed on TMs
- ▶ Anything you write in C, Java, Scala, Python and so on
- ▶ The definition is also robust to variations: if we allow for many tapes instead of one, then nothing changes

This course: Suffices to describe algorithms in a high level language

# Time Complexity

# Running time of a TM

**Definition:** Let  $M$  be a TM that halts on all inputs (**decider**). The **running time** or **time complexity** of  $M$  is the function  $t : \mathbb{N} \rightarrow \mathbb{N}$  where

$$t(n) = \max_{w \in \{0,1\}^n} \text{number of steps } M \text{ takes on } w$$

- ▶  $M$  runs in time  $t(n)$
- ▶  $n$  represents the input length

$$t(0) = 2$$

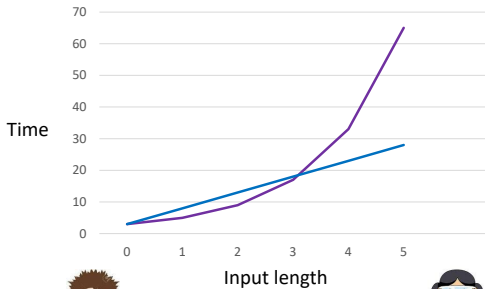
$$t(1) = 10$$

$$t(2) = 45$$

$$t(3) = 85$$

$\epsilon$	2	}	2
0	9		
1	10	}	10
00	21		
01	30	}	45
10	45		
11	33		
000	73	}	85
001	77		
010	85		
011	80		
$\vdots$			

$$t_1(n) = 2^{n+1} + 1 \quad \text{vs} \quad t_2(n) = 5n + 3$$



How to compare running time functions?

# Big-O and Small-o notation

**Definition (Big-O):** Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ . We say  $f(n) = O(g(n))$  if

$$\exists C > 0, n_0 \in \mathbb{N} \text{ s.t. } \quad \forall n \geq n_0 \quad f(n) \leq C \cdot g(n)$$

**Examples:**

- ▶  $5n^3 + 1 = O(2^n)$  ? YES
- ▶  $5n^3 + 1 = O(20n + 5)$  ? NO

**Definition (Small-o):** Let  $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ . We say  $f(n) = o(g(n))$  if

$$\forall c > 0, \exists n_0 \in \mathbb{N} \text{ s.t. } \quad \forall n \geq n_0 \quad f(n) < c \cdot g(n)$$

**Examples:**

- ▶  $\sqrt{n} = o(n)$  ? YES
- ▶  $f(n) = o(f(n))$  ? NO



# Decision problems

**Definition:** Let  $\Sigma$  be a finite input alphabet (typically  $\Sigma = \{0, 1\}$ )  
Denote by  $\Sigma^*$  the set of all finite words over  $\Sigma$

*Example:*  $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 10, \dots\}$

**Definition:** A *language* (or *decision problem*) is a subset  $L \subseteq \Sigma^*$ .

- ▶ **Input:**  $x \in \Sigma^*$
- ▶ **Output:**
  - ▶ 1/YES/True/Accept if  $x \in L$
  - ▶ 0/NO/False/Reject if  $x \notin L$

# Time Complexity

**Definition:** Time complexity class

$$TIME(t(n)) = \{L \subseteq \Sigma^* \mid L \text{ is decided by a TM in time } O(t(n))\}$$

**Example:**

$$TIME(n) \subseteq TIME(n^2) \subseteq \dots \subseteq TIME(2^{\sqrt{n}}) \subseteq TIME(2^n) \subseteq TIME(2^{2^n}) \dots$$

# The complexity class **P** and efficiency

**Definition:** **P** is the class of languages that are **decidable** in **polynomial time** on a (deterministic) Turing machine. In other words,

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

Some languages in **P**:

- ▶  $\{\langle A \rangle : A \text{ is a sorted array of integers}\}$
- ▶  $\{\langle G, s, t \rangle : s \text{ and } t \text{ are vertices connected in graph } G\}$   
(Breadth-First Search)
- ▶  $\{\langle G \rangle : G \text{ is a connected graph}\}$

# NP: Verification vs. Search

# SAT-verify and SAT

Conjunctive Normal Form (CNF) Formula:

$$\varphi_1 = (\bar{x} \vee \bar{y} \vee z_0) \wedge (x \vee \bar{y} \vee z_1) \wedge (\bar{x} \vee y \vee z_2) \wedge (x \vee y \vee z_3)$$

$$\varphi_2 = \bar{x}_1 \wedge (x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee x_2 \vee x_3 \vee \bar{x}_4)$$

$$\varphi_3 = \bar{x}_1 \wedge (x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2)$$

- ▶ **CNF Formula:** AND of **Clauses**
- ▶ **Clause:** OR of **Literals**
- ▶ **Literal:** variable or its negation

**Satisfying assignment:** Boolean assignment to variables which makes the formula TRUE.

Check:  $\varphi_1$  has 32 satisfying assignments,  $\varphi_2$  as only one,  $\varphi_3$  has zero.

A formula is **satisfiable** if it has at least one satisfying assignment.

# SAT-verify and SAT

$$\text{SAT-verify} = \{\langle \varphi, C \rangle : C \text{ is a satisfying assignment of } \varphi\}$$

Is SAT-verify in P? Yes!

- 1 Substitute for literals according to  $C$ .
- 2 Check that every clause has at least one TRUE literal.

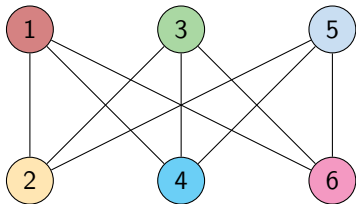
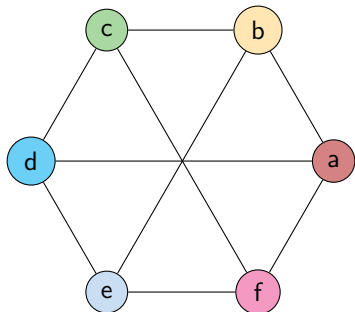
$$\begin{aligned}\text{SAT} &= \{\langle \varphi \rangle : \varphi \text{ is satisfiable}\} \\ &= \{\langle \varphi \rangle : \exists C \text{ such that } \langle \varphi, C \rangle \in \text{SAT-verify}\}\end{aligned}$$

**Is SAT in P?**

Decider for SAT:

- 1 For each assignment  $C$ :
  - ▶ If  $\langle \varphi, C \rangle \in \text{SAT-verify}$ , ACCEPT  $\varphi$ .
- 2 REJECT  $\varphi$ .

# GI-verify and GI



**Graph Isomorphism:** Bijection  $f: V(G_1) \rightarrow V(G_2)$  which preserves adjacency:  $\{u, v\} \in E(G_1) \Leftrightarrow \{f(u), f(v)\} \in E(G_2)$

Eg.  $a \rightarrow 1$   $b \rightarrow 2$   $c \rightarrow 3$   $d \rightarrow 4$   $e \rightarrow 5$   $f \rightarrow 6$  in the graphs above.

Two graphs are **isomorphic** if they have at least one graph isomorphism.

GI-verify =  $\{\langle G_1, G_2, C \rangle : C : V(G_1) \rightarrow V(G_2) \text{ is a graph isomorphism}\}$

Is GI-verify in P? Yes!

**1** Check that  $C$  is a bijection: For each  $u, v \in V(G_1)$ :

▶ Check  $\{u, v\} \in E(G_1) \Leftrightarrow \{C(u), C(v)\} \in E(G_2)$ .

GI =  $\{\langle G_1, G_2 \rangle : G_1 \text{ and } G_2 \text{ are isomorphic}\}$

=  $\{\langle G_1, G_2 \rangle : \exists C \text{ such that } \langle G_1, G_2, C \rangle \in \text{GI-verify}\}$

**Is GI in P?**

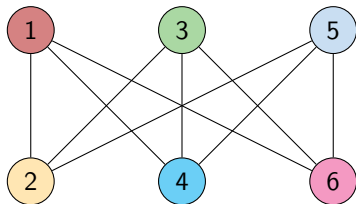
Decider for GI:

**1** For each function  $C : V(G_1) \rightarrow V(G_2)$ :

▶ If  $\langle G_1, G_2, C \rangle \in \text{GI-verify}$ , ACCEPT  $\langle G_1, G_2 \rangle$ .

**2** REJECT  $\langle G_1, G_2 \rangle$ .

# INDSET-verify and INDSET



**Independent Set:** Subset  $S \subseteq V(G)$  such that no two vertices in  $S$  are adjacent in  $G$ .

Eg.  $\{1, 3, 5\}$ ,  $\{2, 4\}$ ,  $\{6\}$ ,  $\emptyset$ , etc. in the graph above.

# INDSET-verify and INDSET

$\text{INDSET-verify} = \{ \langle G, k, C \rangle : C \text{ is an independent set of size } k \text{ in } G \}$

Is INDSET-verify in P? Yes!

- 1 Check that  $|C| = k$ .
- 2 For each  $u, v \in C$ :
  - ▶ Check  $\{u, v\} \notin E(G)$ .

$\text{INDSET} = \{ \langle G, k \rangle : G \text{ has an independent of size } k \}$   
 $= \{ \langle G, k \rangle : \exists C \text{ such that } \langle G, k, C \rangle \in \text{INDSET-verify} \}$

**Is INDSET in P?**

Decider for INDSET:

- 1 For each subset  $C \subseteq V(G)$ :
  - ▶ If  $\langle G, k, C \rangle \in \text{INDSET-verify}$ , ACCEPT  $\langle G, k \rangle$ .
- 2 REJECT  $\langle G, k \rangle$ .

# Verifiers and the class NP

**Recall:** A **decider** for language  $L$  is a TM  $M$  such that for each  $x \in \Sigma^*$

- ▶ If  $x \in L$ , then  $M$  accepts  $x$ .
- ▶ If  $x \notin L$ , then  $M$  rejects  $x$ .

**Definition:**

A **verifier** for language  $L$  is a TM  $M$  such that for each  $x \in \Sigma^*$

- ▶ If  $x \in L$ , then there exists  $C$  such that  $M$  accepts  $\langle x, C \rangle$ .
- ▶ If  $x \notin L$ , then for every  $C$ ,  $M$  rejects  $\langle x, C \rangle$ .

( $C$  is called a *certificate* or *witness*)

A verifier is a **polynomial time** verifier if its running time on any  $\langle x, C \rangle$  is **polynomial in  $|x|$** . (Thus  $|C|$  is polynomial in  $|x|$ )

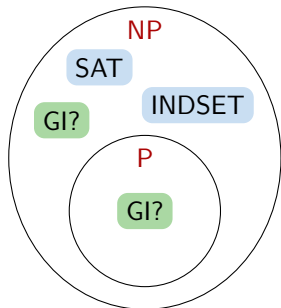
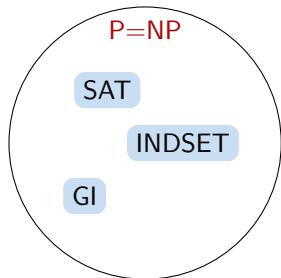
**Definition:** NP is the class of languages that have poly-time verifiers.

# P and NP

Is  $P \subseteq NP$ ? Yes (obviously)

Is  $P = NP$ ? Nobody knows ...

Find the answer and win USD 1,000,000!



**Cook-Levin Theorem** (informal):  $SAT \in P$  iff  $P = NP$ .

(Also  $INDSET \in P$  iff  $P = NP$ .)

# Why is it called NP?

# Detour: Non-deterministic Turing Machines

**Recall:** In a Turing machine,  $\delta : (Q \times \Gamma) \longrightarrow Q \times \Gamma \times \{L, R\}$ .

In a **Nondeterministic Turing Machine (NTM)**,

$$\delta : (Q \times \Gamma) \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

(several possible transitions for a given state and tape symbol)

**Definition:** A **nondeterministic decider** for language  $L$  is an NTM  $N$  such that for each  $x \in \Sigma^*$ , **every computation of  $N$  on  $x$  halts**, and moreover,

- ▶ If  $x \in L$ , then **some** computation of  $N$  on  $x$  **accepts**.
- ▶ If  $x \notin L$ , then **every** computation of  $N$  on  $x$  **rejects**.

An NTM is a **polynomial time** NTM if the running time of its **longest** computation on  $x$  is **polynomial in  $|x|$** .

# Nondeterministic deciders $\iff$ Verifiers

**Theorem:** For any language  $L \subseteq \Sigma^*$ ,  
 $L$  has a nondeterministic poly-time decider  $\iff L$  has a poly-time verifier.

**Proof Sketch ( $\Leftarrow$ ):**

Let  $M$  be the verifier. NTM  $N$  on input  $x$  does the following:

- 1 Write a certificate  $C$  nondeterministically.
- 2 Run  $M$  on  $\langle x, C \rangle$ .

**Proof Sketch ( $\Rightarrow$ ):**

Let  $N$  be the nondeterministic decider. Verifier  $M$  on  $\langle x, C \rangle$  computes:

- ▶ Simulate  $N$  on  $x$ , choosing transitions given by  $C$ .

$M$  accepts  $\langle x, C \rangle$  iff  $x \in L$  and  $C$  is an accepting path of  $N$  on  $x$ .

# Non-deterministic Polynomial-time

**Theorem:** For any language  $L \subseteq \Sigma^*$ ,

$L$  has a nondeterministic poly-time decider  $\iff L$  has a poly-time verifier.

**Definition:** **NP** is the class of languages which have poly-time nondeterministic deciders, or equivalently, have poly-time verifiers.

**Definition:**

$\text{NTIME}(t(n)) = \{L : L \text{ has a nondeterministic } O(t(n)) \text{ time decider}\}$

Then

$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k).$$