COMS21400 : Time Complexity

G. Fuchsbauer

Dept of Computer Science
University of Bristol,
Room 3.53, Merchant Venturers Building

19–27 Nov 2012
Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness
BIG-O and small-o notation

Classify functions by their asymptotic growth rate

Let $f, g : \mathbb{N} \to \mathbb{R}^+$

- $f(n) = O(g(n))$ if

  $$\exists c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

  (“For some positive $c$: $f(n) \leq c \cdot g(n)$ for all sufficiently large $n$”)

- $f(n) = o(g(n))$ if

  $$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

  that is, $\forall c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : f(n) < c \cdot g(n)$

  (“For any positive $c$: $f(n) < c \cdot g(n)$ for all sufficiently large $n$”)

G. Fuchsbauer
COMS21400 : Time Complexity
BIG-O and small-o notation

Classify functions by their asymptotic growth rate

Let $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$

- $f(n) = O(g(n))$ if
  \[ \exists c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : f(n) \leq c \cdot g(n) \]
  (“For some positive $c$: $f(n) \leq c \cdot g(n)$ for all sufficiently large $n$”)

- $f(n) = o(g(n))$ if
  \[ \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \]
  that is, $\forall c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : f(n) < c \cdot g(n)$
  (“For any positive $c$: $f(n) < c \cdot g(n)$ for all sufficiently large $n$”)
Examples 1

**Polynomials.** If \( f \) is a polynomial of degree \( k \) then

\[
\begin{align*}
    f(n) & = O(n^k) \\
    f(n) & = o(n^{k+1}), \text{ but } f \text{ is not } o(n^k)
\end{align*}
\]

- In general: if \( 0 < k_1 < k_2 \) then \( n^{k_1} = o(n^{k_2}) \)
**Examples 1**

**Polynomials.** If \( f \) is a polynomial of degree \( k \) then

\[
\begin{align*}
    f(n) &= O(n^k) \\
    f(n) &= o(n^{k+1}), \text{ but } f \text{ is not } o(n^k)
\end{align*}
\]

- In general: if \( 0 < k_1 < k_2 \) then \( n^{k_1} = o(n^{k_2}) \)

**Logarithms.**

- Recall: \( a^x = y \) then \( x = \log_a y \)

- Typically \( a = 2 \): \( \lceil \log_2 n \rceil + 1 \) is \( n \)'s length in binary

- \[
    \frac{(\log n)^k}{n^c} \xrightarrow{n \to \infty} 0 \quad \text{for all } k, c > 0
\]

thus \( \log n = o(n^k) \), for all \( k > 0 \), \( n \log n = o(n^2) \), \ldots
Exponentials. Any exponential function “dominates” any polynomial:

\[ \frac{n^k}{c^n} \rightarrow 0 \quad \text{for all } k > 0, \ c > 1 \]

thus \( n^k = o(c^n) \), for any \( c > 1 \)

In general: if \( c_1 < c_2 \) then \( c_1^n = o(c_2^n) \)

Notation: \( f(n) = 2^{O(g(n))} \) iff \( \exists c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : f(n) \leq 2^{c \cdot g(n)} \)
Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness
Time complexity for TMs

Definition. Let $M$ be a TM which halts on every input. The **running time** or **time complexity** of $M$ is $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. (“worst-case time”)

Definition. Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$. $\text{TIME}(f(n))$ is the class of all languages decided by an $O(f(n))$-time TM.
Time complexity for TMs

Definition. Let $M$ be a TM which halts on every input. The **running time** or **time complexity** of $M$ is $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. (“worst-case time”)

Definition. Let $f : \mathbb{N} \rightarrow \mathbb{R}^+$. $\text{TIME}(f(n))$ is the class of all languages decided by an $O(f(n))$-time TM.

Examples.

- $\text{TIME}(n)$ (linear time)
- $P := \bigcup_k \text{TIME}(n^k)$ (polynomial time)
- $\text{EXP} := \bigcup_k \text{TIME}(2^{n^k})$ (exponential time)

Gap: super-polynomial, sub-exponential ($\forall c > 1 : f(n) = o(c^n)$), e.g.: $f(n) = n^{\log n}$. 
Non-deterministic Turing machines

A node • is a configuration

DTM:

Start configuration

NDTM:

$q_{accept}$

$q_{reject}$
Definition. Let $N$ be a NTM which is a decider. The **running time** of $N$ is $f : \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $N$ uses on any branch of its computation on any input of length $n$.

Theorem. *(Time-complexity of NTM simulation.)* Every $t(n)$-time NTM $N$ has an **equivalent** $2^{O(t(n))}$-time TM $M$.

(equivalent: $N$ and $M$ decide the same language.)
Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness
The class $P$

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

- $P$ is **robust** (not affected by model of computation)
- $P$ is a mathematical **model** of “realistically solvable” or “tractable” problems (Cobham’s thesis)
  (caveat: running time $c \cdot n^k$ with $c \gg$ or $k \gg$)

Examples
- $\text{FACTORING} \in \?P$
  - $\text{FACTORING} = \{(N, M) | N \text{ has an integer factor } 1 < k < M\}$
  - Brute force: $O \left(2^{n/2}\right)$
    - Best known algo: $2^{O \left(n^{1/3} (\log n)^{2/3}\right)}$

PATH $\in \?P$
- $\text{PATH} = \{(G, s, t) | G \text{ is directed graph with path from } s \text{ to } t\}$
The class $P$

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

- $P$ is **robust** (not affected by model of computation)
- $P$ is a mathematical **model** of “realistically solvable” or “tractable” problems (Cobham’s thesis)
  
  (caveat: running time $c \cdot n^k$ with $c \gg$ or $k \gg$)

**Examples**

- **FACTORING $\in P$**
  
  $(\text{FACTORING} = \{(N, M) \mid N \text{ has an integer factor } 1 < k < M\})$

  Brute force: $O(2^{n/2})$. Best known algo: $2^{O(n^{1/3}(\log n)^{2/3})}$
The class $P$

\[ P = \bigcup_{k} \text{TIME}(n^k) \]

- $P$ is **robust** (not affected by model of computation)
- $P$ is a mathematical **model** of “realistically solvable” or “tractable” problems (Cobham’s thesis)
  
  (caveat: running time $c \cdot n^k$ with $c \gg$ or $k \gg$)

Examples

- **FACTORING** $\in P$
  
  \[ \text{FACTORING} = \{(N, M) \mid N \text{ has an integer factor } 1 < k < M\} \]
  
  Brute force: $O(2^{n/2})$. Best known algo: $2^{O(n^{1/3}(\log n)^{2/3})}$

- **PATH** $\in P$
  
  \[ \text{PATH} = \{((G, s, t) \mid G \text{ is directed graph w/ path from } s \text{ to } t}\} \]
Examples

- $V$ is a polynomial time verifier if $V$ runs in poly time in the length of $w$ (note: not it's full input $(w, c)$!). In this case $V$ can access only poly many squares so the certificates have only poly length too (without loss of generality).

- A language $A$ is polynomially verifiable if it has a poly time verifier.

Definition: NP is the class of all polynomially verifiable languages ("languages with short certificates").

Intuitively $P = \text{class of languages that can be decided}" \text{"quickly";}$ NP = class of languages that can be verified "quickly."

Example Hamiltonian path (cf Sipser p268-9). A Hamiltonian path from vertex $s$ to vertex $t$ in a directed graph $G$ is a path that goes through every vertex exactly once.

The computational task $\text{HAMPATH}(G, s, t)$ is defined to accept iff $G$ has a Hamiltonian path from $s$ to $t$. It is easy to check if a given path from $s$ to $t$ is Hamiltonian or not but it is hard to decide if such a path exists or not – there can be generally exponentially many paths in a graph between two given vertices for a brute force search. For example, is there a Hamiltonian path from $s$ to $t$ in the following graph?

\[ \begin{array}{c}
\text{s} \\
\text{u} \\
\text{t} \\
\end{array} \]

In the case of the ordinary path problem, $\text{PATH}(G, s, t)$ (cf INSERT1), we are also faced with a potential search over exponentially many paths but there was a cleverer approach leading to a poly time algorithm. No such cleverer approach is known for $\text{HAMPATH}$ (e.g. in contrast to PATH it no longer suffices to remember just where you currently are on the path being tested). However HAMPATH is in NP – if $G$ has a Hamiltonian path from $s$ to $t$ then the certificate $c$ is just a description of this path and the (poly time) verifier $V(G, s, t, c)$ simply checks that $c$ really is Hamiltonian, starts at $s$ and ends at $t$. If $s = t$ we are asking if a graph $G$ has a closed circuit passing through each vertex exactly once. This computational task is called $\text{HAMCIRCUIT}(G)$ and it is similarly in NP (and not known or believed to be in P).

- It is obvious that $P \subseteq \text{NP}$. (Why?)

- co-NP is the class of all complements $\Sigma^* - A$ of languages $A$ in NP. (cf exercise sheet 1 for $P$ vs. co-P). Note that in general if membership of $A$ has short certificates, we do not obviously get short certificates for non-membership in $A$!! (Why?) Indeed it is not known whether $\text{NP} = \text{co-NP}$ or not (but generally believed not equal).
• $V$ is a polynomial time verifier if $V$ runs in poly time in the length of $w$ (note: not its full input $(w, c)$!). In this case $V$ can access only poly many squares so the certificates have only poly length too (without loss of generality).

• A language $A$ is polynomially verifiable if it has a poly time verifier.

Definition: $NP$ is the class of all polynomially verifiable languages (“languages with short certificates”).

Intuitively $P = \text{class of languages that can be decided “quickly”}$; $NP = \text{class of languages that can be verified “quickly”}$.

Example: Hamiltonian path (cf Sipser p268-9). A Hamiltonian path from vertex $s$ to vertex $t$ in a directed graph $G$ is a path that goes through every vertex exactly once.

The computational task $HAMPATH(G, s, t)$ is defined to accept iff $G$ has a Hamiltonian path from $s$ to $t$. It is easy to check if a given path from $s$ to $t$ is Hamiltonian or not but it is hard to decide if such a path exists or not – there can be generically exponentially many paths in a graph between two given vertices for a brute force search. For example, is there a Hamiltonian path from $s$ to $t$ in the following graph?

In the case of the ordinary path problem, $PATH(G, s, t)$ (cf INSERT1), we are also faced with a potential search over exponentially many paths but there was a cleverer approach leading to a poly time algorithm. No such cleverer approach is known for $HAMPATH$ (e.g. in contrast to $PATH$ it no longer suffices to remember just where you currently are on the path being tested). However $HAMPATH$ is in $NP$ – if $G$ has a Hamiltonian path from $s$ to $t$ then the certificate $c$ is just a description of this path and the (poly time) verifier $V(G, s, t, c)$ simply checks that $c$ really is Hamiltonian, starts at $s$ and ends at $t$. If $s = t$ we are asking if a graph $G$ has a closed circuit passing through each vertex exactly once. This computational task is called $HAMCIRCUIT(G)$ and it is similarly in $NP$ (and not known or believed to be in $P$).

• It is obvious that $P \subseteq NP$. (Why?)

• $co-NP$ is the class of all complements $\Sigma^* - A$ of languages $A$ in $NP$. (cf exercise sheet 1 for $P$ vs. $co-P$). Note that in general if membership of $A$ has short certificates, we do not obviously get short certificates for non-membership in $A$!! (Why?) Indeed it is not known whether $NP = co-NP$ or not (but generally believed not equal).

$HAMPATH = \{(G, s, t) | G \text{ is directed graph with a Hamiltonian path from } s \text{ to } t\}$
Examples

- **HAMPATH** = \{ (G, s, t) | G is directed graph with a Hamiltonian path from s to t \}

- Easy to check that \( (G, s, t) \in HAMPATH \) when given a path; easy to check that \( (N, M) \in FACTORING \) when given a factor ...
Examples

\[ HAMPATH = \{(G, s, t) \mid G \text{ is directed graph with a Hamiltonian path from } s \text{ to } t\} \]

Easy to check that \((G, s, t) \in HAMPATH\) when given a path; easy to check that \((N, M) \in FACTORING\) when given a factor ...

Many computational problems

- can be solved by brute-force, testing exp. many candidates.
- Verification of desired property on a candidate is easy.
Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness
The class NP

Polynomial-time verifiers

A verifier for a language $A$ is a TM $V$, s.t.

$$A = \{ w \mid V \text{ accepts } (w, c) \text{ for some } c \}.$$

The string $c$ is called a certificate or proof of membership in $A$. 

G. Fuchsbauer
COMS21400 : Time Complexity
The class NP

Polynomial-time verifiers

- A **verifier** for a language $A$ is a TM $V$, s.t.

  $$A = \{w \mid V \text{ accepts } (w, c) \text{ for some } c\}.$$

  The string $c$ is called a **certificate** or **proof** of membership in $A$.

- $V$ is a **polynomial-time verifier** if runs in polynomial time in the length of $w$. 

G. Fuchsbauer
COMS21400 : Time Complexity
The class NP

Polynomial-time verifiers

- A **verifier** for a language $A$ is a TM $V$, s.t.
  \[
  A = \{ w \mid V \text{ accepts } (w, c) \text{ for some } c \}.
  \]
  
  The string $c$ is called a **certificate** or **proof** of membership in $A$.

- $V$ is a **polynomial-time verifier** if runs in polynomial time in the length of $w$.

**Definition.** **NP** is the class of all **polynomially verifiable languages** i.e., all languages which have a polynomial-time verifier.
More on NP

\[ P = \text{class of languages that can be decided \textbf{“quickly”}} \]

\[ NP = \text{class of languages that can be verified \textbf{“quickly”}} \]

\[ P \subseteq NP \quad (\rightarrow \text{problems class}) \]
More on NP

P = class of languages that can be \textit{decided} “quickly”
NP = class of languages that can be \textit{verified} “quickly”

\begin{itemize}
  \item P \subseteq NP \quad (\rightarrow \text{problems class})
\end{itemize}

\textbf{Definition}. For a class of languages \( C \), we define \( \text{co-C} \) as the class of all complements \( \overline{A} \) of languages \( A \) in \( C \).

\begin{itemize}
  \item P = \text{co-P} \quad (\rightarrow \text{problems class})
  \item NP \neq \text{co-NP}
\end{itemize}
More on NP

\[ P = \text{class of languages that can be decided “quickly”} \]
\[ NP = \text{class of languages that can be verified “quickly”} \]

\[ P \subseteq NP \quad (\rightarrow \text{problems class}) \]

**Definition.** For a class of languages \( C \), we define \( \text{co-}C \) as the class of all complements \( \overline{A} \) of languages \( A \) in \( C \).

\[ P = \text{co-P} \quad (\rightarrow \text{problems class}) \]
\[ NP \neq \text{co-NP} \]

**Examples.**
- \( HAMPATH \in NP \)
- \( COMPOSITES = \{ x \mid x = pq, \text{ for integers } p, q > 1 \} \in NP \)
- \( PRIMES \in \text{co-NP} \)

Actually: \( PRIMES \in NP \) (not obvious)
\( PRIMES \in P \) (shown in 2003)
More on NP

\[ P = \text{class of languages that can be decided "quickly"} \]
\[ NP = \text{class of languages that can be verified "quickly"} \]

- \[ P \subseteq NP \ (\rightarrow \text{problems class}) \]

Definition. For a class of languages \( C \), we define \( \text{co-C} \) as the class of all complements \( \overline{A} \) of languages \( A \) in \( C \).

- \[ P = \text{co-P} \ (\rightarrow \text{problems class}) \]
- \[ NP \not\subseteq \text{co-NP} \]

Examples. \( \text{HAMPATH} \inNP \)

- \( \text{COMPOSITES} = \{ x \mid x = pq, \text{for integers } p, q > 1 \} \in NP \)
- \( \text{PRIMES} \in \text{co-NP} \)

Actually: \( \text{PRIMES} \in NP \) (not obvious)

\( \text{PRIMES} \in P \) (shown in 2003)
The name NP

**Theorem.** A language is in NP iff it is decided by some polynomial-time NTM.

**Corollary.** NP ⊆ EXP (by the theorem on Slide 9).
The name NP

Theorem. A language is in NP iff it is decided by some polynomial-time NTM.

Corollary. $\text{NP} \subseteq \text{EXP}$ (by the theorem on Slide 9).

$\text{P} \neq \text{NP}$

“Can every problem whose solution is quickly verifiable be solved quickly?”

- Implications?
The SATisfiability problem

- **Boolean variables:** $x$ take values 1 (TRUE) or 0 (FALSE)
- **Boolean operations:** AND ($x_1 \land x_2$), OR ($x_1 \lor x_2$), NOT ($\overline{x}$)
- **Boolean formulas:** e.g. $\phi = (\overline{x}_1 \land x_2) \lor ((x_1 \land \overline{x}_3) \lor x_2)$

A boolean formula is satisfiable if there exists an assignment of 0's and 1's to the variables, s.t. the formula evaluates to 1. (Formula with $n$ variables has $2^n$ possible assignments)

$\text{SAT} := \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}$

$\text{SAT} \in \text{NP}$, $\text{SAT} \in \text{co-NP}$

**Theorem.** (Cook-Levin) $\text{SAT} \in \text{P}$ iff $\text{P} = \text{NP}$
The SATisfiability problem

- **Boolean variables:** $x$ take values 1 (TRUE) or 0 (FALSE)
- **Boolean operations:** AND ($x_1 \land x_2$), OR ($x_1 \lor x_2$), NOT ($\overline{x}$)
- **Boolean formulas:** e.g. $\phi = (\overline{x}_1 \land x_2) \lor ((x_1 \land \overline{x}_3) \lor x_2)$

A boolean formula is **satisfiable** if there exists an assignment of 0’s and 1’s to the variables, s.t. the formula evaluates to 1.

(Formula with $n$ variables has $2^n$ possible assignments)
The SATisfiability problem

- **Boolean variables**: \( x \) take values 1 (TRUE) or 0 (FALSE)
- **Boolean operations**: AND \((x_1 \land x_2)\), OR \((x_1 \lor x_2)\), NOT \((\overline{x})\)
- **Boolean formulas**: e.g. \( \phi = (\overline{x}_1 \land x_2) \lor ((x_1 \land \overline{x}_3) \lor x_2) \)

A boolean formula is **satisfiable** if there exists an assignment of 0’s and 1’s to the variables, s.t. the formula evaluates to 1.

(Formula with \( n \) variables has \( 2^n \) possible assignments)

\[ SAT := \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula} \} \]

- \( SAT \in NP \), \( SAT \not\in \text{co-NP} \)
The SATisfiability problem

- **Boolean variables**: $x$ take values 1 (TRUE) or 0 (FALSE)
- **Boolean operations**: AND $(x_1 \land x_2)$, OR $(x_1 \lor x_2)$, NOT $(\overline{x})$
- **Boolean formulas**: e.g. $\phi = (\overline{x}_1 \land x_2) \lor ((x_1 \land \overline{x}_3) \lor x_2)$

A boolean formula is **satisfiable** if there exists an assignment of 0’s and 1’s to the variables, s.t. the formula evaluates to 1.

(Formula with $n$ variables has $2^n$ possible assignments)

\[
\text{SAT} := \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula} \}
\]

- $\text{SAT} \in \text{NP}$, $\text{SAT} \not\in \text{co-NP}$

**Theorem.** (*Cook-Levin*)

\[
\text{SAT} \in \text{P} \iff \text{P} = \text{NP}
\]
Outline

Big-O and small-o Notation

Time Complexity

The Class P

The Class NP

Reductions, NP-Completeness
Reducibility

Informally: If $A$ reduces to $B$ then $B$ is “harder” than $A$ (cf. undecidability)

Definition.

A language $A$ is polynomial-time reducible to $B$ if there is a poly-time computable $f: \Sigma^* \rightarrow \Sigma^*$ with $w \in A$ iff $f(w) \in B$.

We write $A \leq_p B$.

Theorem. If $A \leq_p B$ and $B \in P$ then $A \in P$. 
Reducibility

Informally: If $A$ reduces to $B$ then $B$ is “harder” than $A$ (cf. undecidability)

Definition. $f : \Sigma^* \rightarrow \Sigma^*$ is **polynomial-time computable** if there is a poly-time TM, which on input $w$ halts with $f(w)$ on its tape.
Reducibility

Informally: If $A$ reduces to $B$ then $B$ is “harder” than $A$ (cf. undecidability)

Definition. $f : \Sigma^* \rightarrow \Sigma^*$ is polynomial-time computable if there is a poly-time TM, which on input $w$ halts with $f(w)$ on its tape.

Definition. A language $A$ is polynomial-time reducible to $B$ if there is a poly-time computable $f : \Sigma^* \rightarrow \Sigma^*$ with

$$w \in A \iff f(w) \in B$$

We write $A \leq_p B$. 

G. Fuchsbauer
COMS21400 : Time Complexity
Reducibility

Informally: If $A$ reduces to $B$ then $B$ is “harder” than $A$ (cf. undecidability)

Definition. $f : \Sigma^* \rightarrow \Sigma^*$ is **polynomial-time computable** if there is a poly-time TM, which on input $w$ halts with $f(w)$ on its tape.

Definition. A language $A$ is **polynomial-time reducible** to $B$ if there is a poly-time computable $f : \Sigma^* \rightarrow \Sigma^*$ with

$$w \in A \iff f(w) \in B$$

We write $A \leq_p B$.

Theorem. If $A \leq_p B$ and $B \in P$ then $A \in P$. 
NP-completeness

Definition. A language $B$ is **NP-complete** if

- $B \in \text{NP}$, and
- every $A$ in NP is polynomial-time reducible to $B$

(*NP-complete problems are the “hardest” problems in NP*)
Definition. A language $B$ is **NP-complete** if
- $B \in \text{NP}$, and
- every $A$ in $\text{NP}$ is polynomial-time reducible to $B$

(*NP-complete problems are the “hardest” problems in $\text{NP}$*)

Theorem. If $B$ is NP-complete and $B \in \text{P}$ then $\text{P} = \text{NP}$.
Definition. A language $B$ is **NP-complete** if

- $B \in \text{NP}$, and
- every $A$ in NP is polynomial-time reducible to $B$

($\text{NP-complete problems are the “hardest” problems in NP}$)

Theorem. If $B$ is NP-complete and $B \in \text{P}$ then $\text{P} = \text{NP}$.

Theorem. If $B$ is NP-complete and $B \leq_p C$, for $C \in \text{NP}$, then $C$ is NP-complete.
Definition. A language $B$ is **NP-complete** if

- $B \in \text{NP}$, and
- every $A$ in $\text{NP}$ is polynomial-time reducible to $B$

$(\text{NP-complete problems are the “hardest” problems in } \text{NP})$

Theorem. If $B$ is NP-complete and $B \in \text{P}$ then $\text{P} = \text{NP}$.

Theorem. If $B$ is NP-complete and $B \leq_p C$, for $C \in \text{NP}$, then $C$ is NP-complete.

Theorem. *(Cook-Levin, restated)* $\text{SAT}$ is NP-complete.
3-SAT

- **Literal:** $x$ or $\overline{x}$
- **Clause:** Disjunction of literals, e.g. $(x_1 \lor \overline{x}_2 \lor x_3)$
- $\phi$ is in **conjunctive normal form** if $\phi$ is a conjunction of clauses
- **3-CNF formula:** A CNF formula with all clauses having 3 literals, e.g. $(x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_2 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4)$. 

Theorem. 3-SAT is NP-complete.
3-SAT

- **Literal**: $x$ or $\overline{x}$
- **Clause**: Disjunction of literals, e.g. $(x_1 \lor \overline{x}_2 \lor x_3)$
- $\phi$ is in **conjunctive normal form** if $\phi$ is a conjunction of clauses
- **3-CNF formula**: A CNF formula with all clauses having 3 literals, e.g. $(x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_2 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4)$.

$3\text{-SAT} := \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3-CNF formula}\}$
3-SAT

- **Literal**: $x$ or $\overline{x}$
- **Clause**: Disjunction of literals, e.g. $(x_1 \lor \overline{x}_2 \lor x_3)$
- $\phi$ is in **conjunctive normal form** if $\phi$ is a conjunction of clauses
- **3-CNF formula**: A CNF formula with all clauses having 3 literals, e.g. $(x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_2 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4)$.

$3$-$SAT := \{(\phi) \mid \phi$ is a satisfiable 3-CNF formula$\}$

**Theorem.** $3$-$SAT$ is NP-complete.
More NP-complete languages

A $k$-clique in a graph is a set of $k$ nodes in which every two nodes are connected by an edge.

$CLIQUE := \{(G, k) \mid G$ is an undirected graph with a $k$-clique$\}$
More NP-complete languages

A $k$-clique in a graph is a set of $k$ nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Theorem. $CLIQUE$ is NP-complete.
More NP-complete languages

A \textit{k-clique} in a graph is a set of \( k \) nodes in which every two nodes are connected by an edge.

\[
\text{CLIQUE} := \{(G, k) \mid G \text{ is an undirected graph with a } k-\text{clique}\}
\]

Theorem. \textit{CLIQUE} is NP-complete.

More NP-complete languages:

- \textit{HAMPATH}
More NP-complete languages

A $k$-clique in a graph is a set of $k$ nodes in which every two nodes are connected by an edge.

$$CLIQUE := \{(G, k) \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

Theorem. $CLIQUE$ is NP-complete.

More NP-complete languages:

- **HAMPATH**
- **SUBSET-SUM** = $\{\{x_1, \ldots, x_k\} \mid$ for some $\{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\}$ we have: $\sum y_i = 0\}$