Running time (or time complexity) of a T.M.

A T.M. has time complexity $T(n)$ if it halts in at most $T(n)$ steps (accepting or not) for all input strings of length $n$.

**Polynomial time**: $T(n) = O(n^c)$ for some fixed $c$ (fixed means independent from $n$, i.e. the input size)

Examples:

- $O(n^2)$
- $O(n \cdot \log n)$
- $O(n^{3.14})$
- $O(n \cdot \log n)$
- $O(2^n)$

Complexity theory considers tractable all problems with poly-time algorithms.

Motivations:

1) robustness wrt the computation model

   All general computation models can simulate each other in poly-time $\Rightarrow$ they define the same class of tractable problems.

2) robustness wrt combining algorithms

   ($\text{e polynomial of } \text{e polynomial is still } \text{e polynomial}$)

3) going from polynomial to non-polynomial is drastic also in practice (e.g. compare $10 \cdot n^2$ with $0.1 \cdot 2^n$, when $n$ grows)
4. Most practically used algorithms that are polynomial are so with a low coefficient (i.e. \( T(n) = O(n^c) \), with \( c \) typically \( \leq 3 \).

**Time complexity classes:**

- Definition: \( P = \{ L \mid L = L(M) \text{ for some poly-time DTM } M \} \)
- \( NP = \{ L \mid L = L(N) \text{ for some poly-time NTM } N \} \)

Note: both DTMs and NTMs must be halting T.M.s.

From the definition we have immediately: \( P \subseteq NP \) (every NTM is also a DTM).

Note: being in \( P \) corresponds to the intuition that the problem can be solved efficiently.

Instead, being in \( NP \) means intuitively that, given a solution, we can check efficiently whether it is correct.

**Satisfiability:**

- Boolean formulae: operands: \( x_1, \ldots, x_n \)
  - operators: \( \land, \lor, \lnot \)
  - formula: \( F(x_1, \ldots, x_n) \)

Satisfiability problem: given a boolean formula \( F(x_1, \ldots, x_n) \), is there a truth assignment (i.e., an assignment of true/false values) for \( x_1, \ldots, x_n \) that satisfies \( F \) (i.e., makes \( F \) evaluate to true)?
Example: \( F(x_1, x_2) = (x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2) \) is satisfiable: \( x_1 = 1, x_2 = 1 \)
\( F(x_1, x_2) = x_1 \land (\neg x_1 \lor x_2) \land \neg x_2 \)

is not satisfiable.

We first show how we can convert it to a language problem:

- We must encode formulas as strings
- \( \Sigma = \{ \land, \lor, \neg, (,), x, 0, 1 \} \)
- Variable \( x_i \): \( x(i)-i\text{-binary} \)
- E.g. \( x_5 \) is encoded as \( 101 \)

\( \Rightarrow \) we obtain that \( F(x_1, \ldots, x_n) \) can be encoded as a string over \( \Sigma \).

\( L_{\text{SAT}} = \{ w \mid w \text{ encodes a satisfiable formula} \} \)

Theorem: \( L_{\text{SAT}} \in \text{NP} \) (i.e., satisfiability is in \( \text{NP} \))

Proof:
It suffices to show a poly-time NTM \( N \) s.t. \( L(N) = L_{\text{SAT}} \)

\( N \) runs in two steps:

1) "guess" a truth assignment \( F \) for \( x_1, \ldots, x_n \)
2) evaluate \( F \) on truth assignment and whether it has value true.

We have: \( F \text{ satisfiable} \iff \exists \text{ satisfying TA} \)

\( \Rightarrow N \) has accepting \( \text{execution} \)

Running time: step 1) \( O(m) \)
step 2) \( O(n^2) \) with multiple steps \( \Rightarrow O(n^4) \)
Note: All decision problems can be converted to language problems, by encoding the input as a string.

We know that \( L_{SAT} \subseteq NP \), but we do not know whether \( L_{SAT} \subsetneq P \):

- we cannot exploit the conversion \( NTM \rightarrow DTM \), since it causes an exponential blowup in running time.
- Under the standard \( NTM \rightarrow DTM \) conversion, the DTM will have to try all possible truth assignments \( (2^k) \).

In fact: open whether \( L_{SAT} \subsetneq P \).

Special case of SAT: \( CSAT \):

- Conjunctive Normal Form:
  - Note: we use + for \( \lor \)
  - and \( \cdot \) for \( \land \)

- Literal: variable \( x_i \) or its negation \( \neg x_i \)
- Clause: \( \lor \) of literals: \( C_j = x_i + \neg x_i \)
- CNF formula: \( \land \) of clauses: \( F = C_1 \cdot \ldots \cdot C_m \)

Thus \( F = \prod_{j=1}^{m} C_j \) with \( C_j = \sum_{a=1}^{n} l_{ja} \).

CSAT problem: given a CNF formula \( F \), decide whether \( F \) is satisfiable.

Since \( SAT \subseteq NP \), we have also \( CSAT \subseteq NP \).
\textbf{CNF formula:} each clause has exactly \( k \) literals.

1-SAT: \((\bar{x}_1) \cdot (\bar{x}_2) \cdot (x_3)\)

2-SAT: \((x_4 + \bar{x}_2) \cdot (\bar{x}_1 + x_2)\)

3-SAT:

\textbf{Note:} 1-SAT \in P (trivial)

2-SAT \in P (not so easy - one graph reachability)

3-SAT \in P is still open

There are many (thousands) problems like SAT and CSAT that can be easily established to be in \( \text{NP} \) as follows:

\textbf{Step 1:} "guess" some solution \( S \)

\textbf{Step 2:} verify that \( S \) is a correct solution

\textbf{Note:} \text{Step 1} exploits non-determinism, and is clearly polynomial (running time of a \( \text{NTM} \))

\text{Step 2}, for the problem to be in \( \text{NP} \), must be carried out deterministically in \( \text{poly-time} \) (polynomial verifiability)

\textbf{Examples:}

- Traveling salesman problem (TSP)

  \text{input: graph } G = (V,E) \text{ with edge lengths } d(u,v)
  \text{integer } k

  \text{problem: does } G \text{ have a tour (visiting each node exactly once) of length } \leq k \text{?}

\text{TSP } \in \text{NP}

\text{Step 1: guess a tour}

\text{Step 2: check that length of tour is } \leq k
- Clique: input: graph $G = (V, E)$
  - integer $k$
  problem: does the graph have a clique of size $k$?
  (a clique is a subgraph of $G$ in which each pair of nodes is connected by an edge)

- Knapsack: input: set of items, each with an integer weight
  - capacity $k$ of a knapsack
  problem: is there a subset of the items whose total weight matches the capacity $k$?

This property explains why so many practical problems are NP:
- problems ask for the design of mathematical objects
  (paths, truth assignments, solutions of equations, VLSI routes)
- sometimes we look for the best solution (or a solution that
  matches some condition) that matches the specification
- the solution is of small (polynomial) size, otherwise it
  would be useless
- it is simple (poly-time) to check whether it matches the spec.
  But, there are exponentially many possible solutions

If we had $P = NP$, all these problems would have efficient
(poly-time) solutions.

But we currently believe that $P \neq NP$.

Assuming $P \neq NP$, how do we determine which problems of NP
are not in $P$ (i.e., we know they don't have an efficient
algorithm)?
NP-completeness

Key idea: we define NP-completeness in such a way that if we show that an NP-complete problem is in P, then all problems in NP would be in P, (i.e., we would have P = NP)

It follows: assuming P ≠ NP, an NP-complete problem cannot be in P

Poly-time reduction:
Problem \( \mathcal{X} \) reduces to problem \( \mathcal{Y} \) in poly-time \((\mathcal{X} \leq_{\text{poly}} \mathcal{Y})\)
if there is a function \( R \) (the poly-time reduction) s.t.
1) \( w \in \mathcal{L}_X \iff R(w) \in \mathcal{L}_Y \)
2) \( R \) is computable by a poly-time DTM
\( (\mathcal{L}_X \) is the language encoding of problem \( \mathcal{X} \))

Theorem: \( \mathcal{X} \leq_{\text{poly}} \mathcal{Y} \) and \( \mathcal{Y} \in P \implies \mathcal{X} \in P \)

Proof: Let \( M_R \) be a poly-time DTM for \( R \)
\( M_Y \) \( \rightarrow \) \( \mathcal{Y} \)

We construct a DTM \( M_x \) for \( \mathcal{X} \) as follows

\[
\begin{align*}
\text{Input: } w & \quad \rightarrow \quad \text{output } w, |w| = m \\
M_R & \quad \rightarrow \quad R(w) \\
M_Y & \quad \rightarrow \quad \begin{cases} 
\text{yes} & \text{if } w \in \mathcal{L}_x \\
\text{no} & \text{if } w \notin \mathcal{L}_x 
\end{cases}
\end{align*}
\]

Running time of \( M_x \):
Suppose \( M_R \) runs in time \( T_R(m) \leq m^a \)
\( M_Y \) \( \rightarrow \) \( \mathcal{Y} \) in time \( T_Y(m) \leq m^b \)

1/12/2015
Let $|w| = n$

Then $|R(w)| \leq n^a$

$\Rightarrow M_x$ runs in time

$$T_x(n) \leq T_R(m) + T_y(T_R(m)) = n^a + (n^a)^b = O(n^{a+b})$$

$q.e.d.$

**Corollary:** $X \preceq_{p} Y$ and $X \notin P \Rightarrow Y \notin P$

**Definition:** Problem $Y$ (or language $L_Y$) is NP-hard if

$\forall X \in NP \text{ we have } X \preceq_{p} Y$

**Intuitively:** an NP-hard problem is at least as hard as any problem in NP

**Immediate:** $Y$ is NP-hard and $Y \notin P \Rightarrow P = NP$

**Definition:** $Y$ is NP-complete if

1) $Y \in NP$ and
2) $Y$ is NP-hard

**Intuitively:** NP-complete problems are the hardest problems in NP.

If one of them is in $P$, then all problems in $NP$ are in $P$.

Hence: NP-completeness is a strong evidence of intractability.
Note: relationship between $P$, $NPC$, and $NP$:

either $P = NP$ or $P \neq NP$

in this case we know there are problems in $NP$ that are neither in $P$ nor $NPC$  
(proof is complicated)

How do we prove problems to be $NP$-complete?

Theorem: $X$ is $NP$-hard and $X \preceq_{poly} Y \implies Y$ is $NP$-hard

Proof:

$NP \preceq_{poly} X \preceq_{poly} Y$

But, to exploit this result, we need a first $NP$-hard problem:

Cook's theorem: $CSAT$ is $NP$-hard

Proof idea: we must show: $V \leq_{NP} L \leq_{poly} L \text{ CSAT}$

Fix $L \in NP$ and let $M_L$ be a poly-time NTM for $L$.

We must show a poly-time reduction $R_L$:

input: a string $w$

output: CNF formula $F = R_L(w)$ such that $w \in L(M_L) \iff F$ is satisfiable

Idea: $F$ encodes the computation of $M_L$ on $w$. Let $P_L(m)$ be the (polynomial) running time of $M_L$. 
Suppose \( w \in L(M_L) \) and \( |w| = m \).

Then there exists a sequence of IDs of \( M_L \):
\[
ID_0 \rightarrow ID_1 \rightarrow \ldots \rightarrow ID_T
\]
with \( ID_0 = \sigma_0^m \)

\( ID_T \) is an accepting ID (i.e. \( M_L \) is in a final state).

We assume that \( T = P(m) \) by adding
\[
ID_{T+1}, ID_{T+2}, \ldots, ID_{P(m)} \text{ same as } ID_T
\]

Idea: encode computation as matrix \( X \)

\[
\begin{array}{cccc|ccc|ccc|ccc}
0 & q_0 & q_1 & q_2 & q_3 & \ldots & q_m & \mathbb{B} & \mathbb{B} & \ldots & \mathbb{B} & \mathbb{B} \\
1 & q_1 & q_2 & q_3 & \ldots & q_m & \mathbb{B} & \mathbb{B} & \ldots & \mathbb{B} & \mathbb{B} \\
2 & q_1 & q_2 & q_3 & \ldots & q_m & \mathbb{B} & \mathbb{B} & \ldots & \mathbb{B} & \mathbb{B} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
P(m) & q_1 & q_2 & q_3 & \ldots & q_m & l_1, l_2, l_3 & q_x & \mathbb{B} \end{array}
\]

\( M \) cannot use more than \( P(m) \) cells

\( \mathbb{B} \) : contents of tape cell \( i \) in \( ID_T \)

except for composite symbol \( q_x \) to denote state and head position.

We have that \( w \in L(M_L) \) iff

a) the matrix \( X \) is properly filled in

b) row 0 in \( ID_0 \)

c) row \( P(m) \) has final state

d) successive rows are related through legal transitions of \( M_L \)
$M_L$ is NTM. Let be be the maximum degree of nondeterminism, i.e., for all $q, x : |\delta(q, x)| \leq k$.

To encode which of the possible transitions is chosen when going from $1D_i$ to $1D_{i+1}$ for the accepting sequence:

We use an array $C$ of $P(m)$ elements (call array)

\[
\begin{array}{c|c|c}
\text{TIME} & 0 & 1 \\
\downarrow & C_0 & C_1 \\
& \vdots & \vdots \\
& C_{P(m)-1} & C_{P(m)}
\end{array}
\]

$1 \leq C_i \leq k$

To represent $X$ and $C$ we use boolean variables

$X_{itA} = \text{true if cell } i \text{ in } 1D_t \text{ contains } A$

$C \equiv \text{true if } C_t = t$

where $1 \leq i \leq P(m)$

$0 \leq t \leq P(m)$

$A \in \Gamma' = \Gamma \cup \Gamma^Q$

$1 \leq l \leq k$

Total number of variables is $O(P(m)^2)$, i.e., polynomial

To construct the CNF formula $F$, we use 4 types of formulas (that are conjunctions of clauses)

Type C) $X$ and $C$ are properly filled in:

Cell $i$ at time $t$ is properly filled

$\text{UNIQUE}_{it} = \sum_{A \in \Gamma'} X_{itA} \land \bigwedge_{A, B \in \Gamma'} (\overline{X_{itA}} \lor \overline{X_{itB}})$
$C[1]$ is properly filled

$$UNIQUE_t^i = \sum_{1 \leq l \leq k} C_{tl} \land \prod_{1 \leq m \leq k} (\bar{C}_{tm} \lor C_{km})$$

$$UNIQUE = \prod_{1 \leq i \leq P_i(m)} UNIQUE_{i,t} \land \prod_{0 \leq t \leq P_i(m)-1} UNIQUE^{\ast}_{i,t}$$

$$\Rightarrow O(P_i(m)^2)$$ clauses, each of length 1 or 2.

**Type b)** \( ID_0 = q_0 w = q_0 a_n \ldots a_m \)

$$INIT = X_0 X_{g_0}, X_{20}, a_n, \ldots, X_{m0}. a_m$$

$$X_{m+1, 0}, X_{m+2, 0}, \ldots, X_{P_i(m), 0}, \psi$$

$$\Rightarrow O(P_i(m))$$ clauses, each of length 1

**Type c)** \( ID_{P_i(m)} \) is accepting

$$ACCEPT = \sum_{q \in F} X_{i, P_i(m), q} \left[ \begin{array}{c} \delta \end{array} \right]$$

$$\Rightarrow 1$$ clause of length \( O(P_i(m)) \)

**Type d)** legal transitions

Consider \( ID_4 \) and \( ID_{4+1} \)

<table>
<thead>
<tr>
<th>( t )</th>
<th>( A_1, A_2 )</th>
<th>( \ldots )</th>
<th>( A_{j-1}, A_{j+1} )</th>
<th>( \ldots )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t+1 )</td>
<td>( B_1, B_2 )</td>
<td>( \ldots )</td>
<td>( B_{j-1}, B_{j+1} )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

In \( ID_{4+1} \), cell \( j \) depends only on 3 cells above it and on \( C_t \)
Various cases: (we assume that there are no stay moves)

1) \(A_{j-1}, A_j, A_{j+1}\) are not composite symbols then \(B_j = A_j\)

2) \(A_{j-1}\) is \(\overline{A}\) and \(q\) th move in \(S(q, X)\) is \((q, Y, R)\)
then \(B_j = \overline{A_j}\)

3) \(A_j\) is \(\overline{A}\) and \(q\) th move in \(S(q, X)\) is \((q, Y, L)\)
then \(B_j = Y\)

4) \(A_{j+1}\) is \(\overline{A}\) and \(q\) th move in \(S(q, X)\) is \((q, Y, L)\)
then \(B_j = \overline{A_j}\)

We use clauses that forbid illegal moves: \(\text{LEGAL}(k, j)\)

\[\begin{align*}
\text{T}(D, \overline{E}, \overline{F}, G, H) \\
&= \left( \overline{C_{j, D}} + \overline{K_j x, t, E} + \overline{K_{j, t, F}} + \overline{K_{j+1, x, G}} \right) \\
&\quad + \overline{K_{j+1, x, H}}
\end{align*}\]

s.t. with clue \(D\)
and \(E, F, G, H\) we have an illegal move.

(NB: the illegal moves are those that do not correspond to 1-4 above)

\[\Rightarrow O(p(n)^2)\] clauses, each of constant length

(since \(0 \leq t < p(n)\), \(1 \leq j \leq p(n)\))

Denote \(F\) is the conjunction of all above clauses.
We can prove that \(w \in \mathcal{L}(M_k)\) iff \(F\) is satisfiable.
It is easy to see that the reduction is poly-time \(\text{q.e.d.}\)
Exercise: Let \( G = (V, E) \) be an undirected graph.

A vertex cover \( C \) of \( G \) is a subset of the vertices \( V \) such that every edge of \( G \) touches at least one of the nodes of \( C \).

**The vertex cover problem:**

**Input:**
- graph \( G = (V, E) \)
- integer \( k \)

**Output:** yes iff \( G \) has a vertex cover of size \( \leq k \).

**Vertex cover is NP-complete:**

**Proof:**

in \( \text{NP} \): easy

- guess a subset \( C \) of \( V \) of size \( \leq k \)
- check in poly-time that it is a vertex-cover.

\( \text{NP} \)-hard: by reduction from \( 3\text{-SAT} \)

We define a poly-time reduction \( R \) that:

- takes as input a 3-CNF formula \( F \)
- constructs a graph \( G = (V, E) \) and an integer \( k \) such that:

\( F \) is satisfiable \( \iff \) \( G \) admits a vertex cover with \( k \) nodes.

Let \( F = C_1 \land \cdots \land C_m \) be a 3-CNF formula over variables \( \{x_1, \ldots, x_n\} \)

We construct \( G = (V, E) \) as constituted by various components,
- For each variable $x_i$, we have a truth-setting component $T_i = (V_i, E_i)$ with $V_i = \{ x_i, \overline{x_i} \}$

$$E_i = \{ x_i, \overline{x_i} \}$$

note: at least one of $x_i, \overline{x_i}$ will be in every vertex cover to cover $\{ x_i, \overline{x_i} \}$

- For each clause $c_j$ in $F$ we have a satisfaction testing component $S_j = (V_j', E_j')$

$$S_j = \{ v_1, v_2, v_3, o_1j, o_2j, o_3j \}$$

note: at least two of $V_j'$ will be in every vertex cover to cover $E_j'$

- We have a communication component, which in the only part that depends on which literals are in which clauses.

Let $c_j = l_{1j} + l_{2j} + l_{3j}$

then we have $E''_j = \{ o_{1j}, l_{1j}, o_{2j}, l_{2j}, o_{3j}, l_{3j} \}$

We then set $K = n + 2m$

# variables

# clauses
Example: \( F = (\overline{x_1} + \overline{x_3} + x_4) \cdot (\overline{x_1} + x_2 + \overline{x_4}) \)

\[ k = m + 2m = 4 + 2 \cdot 2 = 8 \]

We show that \( F \) is satisfiable \( \Rightarrow \) \( G \) has a vertex cover of size \( \leq k \)

\[ \leq \]

Let \( V' \subseteq V \) be a vertex cover for \( G \) with \( |V'| \leq k \).

We said that \( V \) contains at least one vertex for each variable and at least two vertices for each clause.

This is already \( k = m + 2m \)

\( \Rightarrow \) at least is actually exactly.

We are \( V' \) to obtain the truth assignment \( \gamma \).

We set \( \gamma(x_i) = \text{true} \) if \( x_i \in V' \)

\( \gamma(x_i) = \text{false} \) if \( \overline{x_i} \in V' \) (i.e., \( \overline{x_i} \in V' \))

To show that \( \gamma \) is a truth assignment that satisfies \( F \), we exploit that all clauses of the communication components are covered by \( V' \).

Consider a clause \( C_j = \overline{x_{j1}} \lor x_{j2} \lor \overline{x_{j3}} \).

Two of the arcs in \( E'' \) are covered by the choice of \( \gamma \) for \( x_{j1}, \overline{x_{j2}}, \overline{x_{j3}} \in V' \).

\( \forall j \in [N] \), let there be \( x_{ji}, \overline{x_{ji}} \)
Let $\Phi$ be a truth assignment that satisfies $F$.

We define a subset $V' \subseteq V$ as follows:

- $x_i \in V'$ iff $\Phi(x_i) = \text{true}$
- $\bar{x}_i \in V'$ iff $\Phi(x_i) = \text{false}$

Since $\Phi$ satisfies $F$, for each communication component $E_j = \{e_{i,j}, l_{i,j}, e_{2i,j}, l_{2i,j}, e_{3i,j}, l_{3i,j}\}$, one of the three edges $\{e_{i,j}, l_{i,j}\}$ is covered in $V'$ by $l_{i,j}$.

We set $i = 1$. Then $\{e_{2i,j}, l_{2i,j}\}, \{e_{3i,j}, l_{3i,j}\}$ can be covered by having $e_{2i,j} \in V'$ and $e_{3i,j} \in V'$.

We get that $V'$ contains $n + 2m$ vertices.
In a collection of NP-complete problems with discussion of variants see

Garey & Johnson.


Burremann & Co. 1973

**coNP - completeness**

Set us consider the complement of a problem in NP.

E.g. unsatisfiability

\[ UNSAT = \{ F \mid F \text{ is a propositional formula that is not satisfiable} \} \]

Given a prop. formula \( F \), how can we check whether \( F \in UNSAT \)?

- try all possible truth assignments for the vars in \( F \)
- if for none of these, \( F \) evaluates to true, answer yes

Intuitively, this is very different from a problem in NP.

Note: in general, a NTM cannot answer yes to such a problem in polynomial time

**Definition:** \( coNP = \{ L \mid \overline{L} = \Sigma^* \setminus L \in NP \} \)

Note: many problems in \( coNP \) do not seem to be in NP.
We might conjecture NP ≠ coNP

This conjecture is stronger than P ≠ NP.

Indeed, since P = coP, we have that NP ≠ coNP implies P ≠ NP.

- but we might have P ≠ NP, and still NP = coNP

The following result shows a strong connection between NP-complete problems and the conjecture that NP ≠ coNP.

**Theorem:** If for some NP-complete problem/language L we have \(\overline{L} \in \text{NP} \) (i.e., \(L \in \text{coNP}\)), then NP = coNP.

**Proof:** Assume \(L \in \text{NPC} \) and \(\overline{L} \in \text{NP}\).

1) We show NP \(\subseteq\) coNP.

Let \(L' \in \text{NP}\). We show \(\overline{L'} \in \text{coNP}\), i.e. \(\overline{L'} \in \text{NP}\).

Since \(L' \in \text{NP}\), there is a poly-time NTM \(N_E\) s.t. \(L(N_E) = \overline{L'}\).

Since \(L' \in \text{NPC}\) and \(L \in \text{NPC}\), \(L' \leq_{\text{poly}} L\), i.e.

there is a polytime reduction \(R\) s.t.

\[w \in L' \iff R(w) \in L\]
\[w \notin L' \iff R(w) \notin L\]

We can construct a poly-time NTM \(N_{E'}\) for \(\overline{L'}\).

2) CONP \(\subseteq\) NP. Similar

q.e.d.
We get the following picture (assuming $P \neq NP$

\[ NP \neq coNP \]

Note: it is not known whether $P = NP \land coNP$. 