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 (excepting 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:

\[
\begin{align*}
0(n^2) & \quad \text{polynomial time} \\
O(n \cdot \log n) & \quad \text{polynomial time} \\
O(n^{3.44}) & \quad \text{polynomial time} \\
O(n \cdot \log n) & \quad \text{non-poly} \\
O(2^n) & \quad \text{non-poly}
\end{align*}
\]

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

Motivations:

1) Robustness w.r.t. the computation model

(all general computation models can simulate each other in poly-time \(\Rightarrow\) they define the same class of tractable prob.)

2) Robustness w.r.t combining algorithms

(e polynomial of a polynomial is still a polynomial)

3) Going from polynomial to non-polynomial is drastic also in practice (e.g., compare $100n^2$ with $0.1 - 2^n$ 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 typically $c \leq 3$.

**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: $\land, \lor, \neg$
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: \( F(\kappa_1, \kappa_2) = (\kappa_1 \lor \neg \kappa_2) \land (\neg \kappa_1 \lor \kappa_2) \)  

is satisfiable: \( \kappa_1 = 1, \kappa_2 = 1 \)

\( F(\kappa_1, \kappa_2) = \kappa_1 \land (\neg \kappa_1 \lor \kappa_2) \land \neg \kappa_2 \)

is not satisfiable  

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

- we must encode formulas as strings

\[ \Sigma = \{ \land, \lor, \neg, \lnot, (, ), 0, 1 \} \]

variable \( \kappa_i \): \( \kappa \text{(im-binary)} \)

e.g. \( \kappa_5 \) is encoded as \( \times 101 \)

\[ \Rightarrow \text{we obtain that } F(\kappa_1, \ldots, \kappa_n) \text{ can be encoded as a string over } \Sigma. \]

\[ L_{\text{SAT}} = \{ w \mid w \text{ encodes a satisfiable formula} \} \]

Theorem: \( L_{\text{SAT}} \in \text{NP} \) (i.e. satisfiability is in 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 \( \kappa_1, \ldots, \kappa_n \)

2) evaluate \( F \) on truth assignment and whether it has a value true.

We have: \( F \text{ satisfiable } \iff \exists \text{ satisfying T.A.} \)

\( \Rightarrow N \) has accepting procedure

Running time: step 1) \( O(n) \)

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 \text{SAT} \in \text{NP}, but we do not know whether \text{SAT} \in \text{P}.
- We cannot exploit the conversion \text{NTM} \rightarrow \text{DTM}, since it causes an exponential blowup in running time.
- Under the standard \text{NTM} \rightarrow \text{DTM} conversion, the \text{DTM} will have to try all possible truth assignments \(2^{2^k}\).

In fact: open whether \text{SAT} \in \text{P}.

Special case of \text{SAT} : \text{CSAT}.

Conjunctive normal form:

- \text{literal} : variable \(x_i\) or its negation \( \bar{x}_i \).
- \text{clause} : \text{sum} / \text{or} of literals : \(C_j = x_i + \bar{x}_2\).
- \text{CNF-formula} : \text{product} / \text{and} of \text{clauses} : F = C_1 \cdot C_2 \cdot \ldots \cdot C_m.

Thus \( F = \bigwedge_{j=1}^{m} C_j \) with \( C_j = \bigvee_{i=1}^{k} x_{j,i} \).

\text{CSAT} \text{ problem:} \ given a \text{CNF} \text{ formula} \ F, \ decide \ whether \ F \ \text{is satisfiable}.

Since \text{SAT} \in \text{NP}, we have also \text{CSAT} \in \text{NP}.
A CNF-formula: each clause has exactly k literals.

1-SAT: \((\overline{x_1}) \cdot (\overline{x_2}) \cdot (x_3)\)

2-SAT: \((x_1 + \overline{x_2}) \cdot (\overline{x_1} + x_2)\)

3-SAT:

Remarks:

1-SAT \(\in \mathbb{P}\) (trivial)
2-SAT \(\in \mathbb{P}\) (not so easy - via graph reachability)
3-SAT \(\in \mathbb{P}\) is still open

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

Step 1: "guess" some solution \(S\)
Step 2: verify that \(S\) is a correct solution

Note: Step 1 exploits nondeterminism, and is clearly polynomial (running time of a NTM)
Step 2, for the problem to be in \(\mathbb{NP}\), must be carried out deterministically in poly-time (polynomial verifiability)

Examples:

- Traveling salesman problem (TSP)

  Input: graph \(G=(V,E)\) with edge lengths \(d(m,v)\)
  Integer \(k\)

  Problem: does \(G\) have a tour (visiting each node exactly once) of length \(\leq k\)?

  \(TSP \in \mathbb{NP}\)

  Step 1: guess a tour
  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 \( G \) on 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 \( \text{NP} \) - 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 \( P = \text{NP} \), all these problems would have efficient (poly-time) solutions.

But we currently believe that \( P \neq \text{NP} \).

Assuming \( P \neq \text{NP} \), how do we determine which problems of \( \text{NP} \) are not in \( P \) (i.e., we know they don’t have an efficient algorithm)?
Key idea: we define NP-completeness in such a way that if we show that an NP-complete problem \( \mathcal{X} \) is in \( \mathcal{P} \), then all problems in \( \mathcal{NP} \) would be in \( \mathcal{P} \). (i.e. we would have \( \mathcal{P} = \mathcal{NP} \))

It follows: assuming \( \mathcal{P} \neq \mathcal{NP} \), an NP-complete problem cannot be in \( \mathcal{P} \).

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. \( w \in L_x \iff R(w) \in L_y \)
2. \( R \) is computable by a poly-time DTM

\( L_x \) is the language encoding of problem \( \mathcal{X} \).

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

Proof: let \( M_R \) be a poly-time DTM for \( R \)

\[ M_Y \]

\[ M_X \]

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

Running time of \( M_x \):

- Suppose \( M_R \) runs in time \( T_R(m) \leq m^a \)
- \( M_Y \) runs in time \( T_Y(m) \leq m^b \)
Set \( |w| = n \)

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

\( \Rightarrow M_X \) runs in time

\[ T_X(n) \leq T_R(n) + T_Y(T_R(n)) = n^a + (n^a)^b = O(n^{a+b}) \]

q.e.d.

**Corollary:** \( X \leq_{\text{poly}} Y \) and \( X \notin P \Rightarrow Y \notin P \)

**Definition:** Problem \( Y \) (or language \( L_Y \)) is NP-hard if \( \forall X \in \text{NP} \) we have \( X \leq_{\text{poly}} Y \)

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

**Immediate:** \( Y \) is NP-hard and \( Y \in P \Rightarrow P = \text{NP} \)

**Definition:** \( Y \) is NP-complete if

1) \( Y \in \text{NP} \) 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 \( \text{NP} \) 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 $X \leq_{p} Y \Rightarrow Y$ is NP-hard

**Proof:** $NP \leq_{p} X \leq_{p} Y$

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

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

**Proof idea:** we must show: $\forall L \in NP : L \leq_{p} L_{CSAT}$

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: string $w$
- output: CNF formula $F = R_L(w)$ s.t. $w \in L(M_L) \iff F$ is satisfiable

Idea: $F$ encodes the computation of $M_L$ on $w$. 

2/12/2008
Suppose \( w \in L(M_L) \) and \( |w| = n \).

Then there exists a sequence of IDs of \( M_L \):
\[
ID_0 \to ID_1 \to \ldots \to ID_T
\]
with \( ID_0 = q_0w \)

\( ID_T \) 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 \( X \)

\[
\begin{array}{cccccccc}
1 & 2 & 3 & m & m+1 & m+2 & P(m) & w = a_1 \ldots a_n \\
\hline
0 & a_0 & a_1 & a_2 & a_3 & \ldots & a_m & b & b & \ldots & b & b \\
1 & b_1 & a_0 & a_1 & a_2 & \ldots & a_m & b & b & \ldots & b & b \\
2 & b_2 & b_1 & a_0 & a_1 & a_2 & \ldots & a_m & b & b & \ldots & b & b \\
\end{array}
\]

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

\( \forall i \): contents of tape cell \( i \) in \( ID_0 \)
except for composite symbol [2]
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 $k$ 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 $1D_t$ to $1D_{t+1}$ for the accepting sequence:

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

<table>
<thead>
<tr>
<th>TIME</th>
<th>$C_0$</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>$\vdots$</th>
<th>$C_{P(n)-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$C_0$</td>
<td>$C_1$</td>
<td>$C_2$</td>
<td>$\vdots$</td>
<td>$C_{P(n)-1}$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$1 \leq C_t \leq k$

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

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

$$C_{it} = \text{true if } C_t = l$$

where

$1 \leq i \leq P(n)$

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

$A \in \Gamma' = \Gamma \cup \Gamma \times \{0\}$

$1 \leq l \leq k$

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

To construct the CNF formula $F$, we use 4 types of formulas:

- $K$ and $C$ are properly filled in:
  (that are conjunctions of clauses)

- $\text{UNIQUE}(i, t)$: for each $i$ and $t$, cell $i$ in $1D_t$ is uniquely filled

\[
(\sum_{A \in \Gamma'} X_{itA}) \land \bigwedge_{A, B \in \Gamma'} (X_{itA} + X_{itB})
\]
$\text{UNIQUEC}(k)$ : for each $k$, $C[k]$ is uniquely filled

$\left( \sum_{l \in S_k} \prod_{m \in S_m, m \neq k} (\bar{C}_{k,m} + \bar{C}_{m,k}) \right)$

$\Rightarrow O(P(m)^2)$ clauses, which is still polynomial
(since $1 \leq i \leq P(m)$ and $0 \leq k \leq P(m)$)
Each clause has constant length (i.e., independent of $m$).

Type b) \hspace{1cm} ID$\_0$ = $q_0$, $\omega = q_0, a_1 \cdots a_n$

\begin{align*}
\text{INIT} & : x_1^{q_0} \cdots x_{i_0}^{q_{i_0}} a_i \cdots x_{i_m}^{q_{i_m}} a_i \\
& \hspace{1cm} \cdots \hspace{1cm} x_{i_m}^{q_{i_m}} a_i \hspace{1cm} \cdots \hspace{1cm} x_{i_{p(n)}}^{q_{i_{p(n)}}} a_i
\end{align*}

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

Type c) \hspace{1cm} ID$\_P(m)$ is accepting

\begin{align*}
\text{ACCEPT} & : \sum_{q \in F} x_{i_q} P(m), \bar{q}/A \\
& \hspace{1cm} \text{for all } i_q \in \{1, \ldots, P(m)\}
\end{align*}

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

Type d) legal transitions

consider \hspace{1cm} ID$\_A$ and \hspace{1cm} ID$\_A_{i+1}$

\begin{tabular}{c c c c}
\hline
$A_1$ & $A_2$ & $\cdots$ & $A_{i+1}$ \\
\hline
$B_1$ & $B_2$ & $\cdots$ & $B_{i+1}$ \\
\hline
\end{tabular}

On \hspace{1cm} ID$\_A_{i+1}$, cell $j$ depends only on $3$ cells above it and on $C_k$. 

\begin{tabular}{c c c}
$A_{i-1}$ & $A_j$ & $A_{i+1}$ \\
$B_j$ \\
\end{tabular}
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 \( \text{X} \) and \( i \)’th move in \( S(q, x) \) is \( (q, y, r') \)
   then \( B_j = X_{\text{A}_j} \)

3) \( A_j \) is \( \text{X} \) and \( i \)’th move in \( S(q, x) \) is \( (q, y, r, -) \)
   then \( B_j = Y \)

4) \( A_{j+1} \) is \( \text{X} \) and \( i \)’th move in \( S(q, x) \) is \( (q, y, l) \)
   then \( B_j = \overline{X}_{A_j} \)

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

\[
\bigwedge_{D, E, F, G, H} \left( \overline{C}_{k, D} + \overline{K}_{j, r, k, E} + \overline{K}_{j, l, F} + \overline{K}_{j+1, k, G} \right) + \overline{X}_{j, k+1, H}
\]

\( \text{A.H. with clause D} \)
and \( \overline{E}_{F, G} \) are
then an illegal move

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

\( \Rightarrow O(P(m)^2) \) clauses, each of constant length
(since \( 0 \leq t < P(m), 1 \leq j \leq P(m) \))

Formula \( F \) is the conjunction of all above clauses.
We can prove that \( W \in \mathcal{L}(M) \) iff \( F \) is satisfiable.
It is easy to see that the reduction is poly-time q.e.d.
In a collection of NP-complete problems with discussion of variants see

Garey & Johnson,

*Computers and Intractability: A Guide to the Theory of NP-Completeness*

Greene & Sol, 1979

\[ \text{coNP-completeness} \]

Let us consider the complement of a problem in NP.

E.g. unsatisfiability

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

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

- 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:** \( \text{coNP} = \{L \mid \exists \Sigma^* \setminus L \in \text{NP} \} \)

"Note: many problems in \( \text{coNP} \) do not seem to be in \( \text{NP} \)."
We might conjecture $NP \neq coNP$.

This conjecture is stronger than $P \neq NP$.

Indeed, since $P = coP$, we have that $NP \neq coNP$ implies $P \neq NP$.

But we might have $P \neq NP$, and still $NP = coNP$.

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

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

**Proof:** Assume $L \in NP$ and $L \in conP$.

1) We show $NP \subseteq conP$.

Let $L \in NP$. We show $L' \in conP$, i.e., $\overline{L} \in NP$.

Since $L \in NP$, there is a poly-time NTM $N_L$ s.t. $L(N_L) = L$.

Since $L' \in NP$ and $L \in NP$, $L' \leq_{poly} L$, i.e., there is a polytime reduction $R$ s.t.

\[ w \in L' \iff R(w) \in L \quad \text{i.e.} \]

\[ w \in \overline{L} \iff R(w) \in \overline{L} \]

We can construct a poly-time NTM $N_{\overline{L}}$ for $\overline{L}$.

2) $conP \subseteq NP$. Similar
We get the following picture (assuming \( P \neq NP \) and \( NP \neq coNP \)).

Note: it is not known whether \( P = NP \cap coNP \).