Polynomial Time

Definition A program or algorithm has worst case running time $T(n)$ if for all inputs of size $n$ it completes its computation in at most $T(n)$ steps. (Strictly, in addition, on some input of size $n$, the full $T(n)$ steps are performed.)

By a step we intend a basic operation such as an addition, a comparison, a branch (due to a goto, in a while loop, in an if-statement, to perform a procedure call, etc.). We do not allow more complex steps such as $B \leftarrow 0$ where $B$ is an $n \times n$ array to count as a single step. Finally, we also limit the word size to $O(\log n)$ bits, so repeated squaring of a number would not be a legitimate series of steps.

Definition An algorithm runs in polynomial time if its running time $T(n)$ is bounded by a polynomial function of $n$.

Definition A problem or language is in $\mathbf{P}$ if there is an algorithm solving the problem or deciding the language that runs in polynomial time.

We view polynomial time as corresponding to the class of problems having efficient algorithms. Of course, if the best algorithm for an algorithm ran in $n^{100}$ time this would not be efficient in any practical sense. However, in practice, the polynomial time algorithms we find tend to have modest running times such as $O(n), O(n \log n), O(n^2), O(n^3)$.

Another interesting way of viewing polynomial time algorithms is that a moderate increase in resources suffices to double the size of the problems that can be solved. For example, given that algorithm $A$ can solve a problem of size $m$ in one hour on a particular computer, if $A$ runs in linear time, then given a new computer that runs twice as fast in the one hour one could solve twice as large a problem with algorithm $A$. By running twice as fast I mean that it performs twice as many operations in the given time. While if $A$ runs in quadratic time ($\theta(n^2)$), then give a new computer that runs four times as fast, in one hour one could solve a problem that is twice as large; again, if $A$ runs in cubic time ($\theta(n^3)$), then given a new computer that runs eight times as fast, in one hour one could solve twice as large a problem; and so forth.

This contrasts with the effect of an exponential running time ($2^n$, for instance). Here doubling the speed of the computer increases the size of the problem that can be solved in
one hour by one. As a result, exponential time algorithms are generally considered to be infeasible.

**The Class NP**

This class of languages is characterized by being “verifiable” in polynomial time. We begin with some examples.

**Example 1.** Hamiltonian Circuit.

Input: A directed graph \( G = (V, E) \).

Question: Does \( G \) have a Hamiltonian Circuit, that is a cycle that goes through each vertex in \( V \) exactly once?

Output: “Yes” or “no” as appropriate.

This problem is polynomial time verifiable in the following sense. Given a sequence of \( n = |V| \) vertices which is claimed to form a Hamiltonian Circuit this is readily checked in linear time (it suffices to check that each vertex appears exactly once in the list and that if the list is the sequence \( v_1, v_2, \ldots, v_n \), then \((v_i, v_{i+1} \mod n) \in E\) for each \( i, 1 \leq i \leq n \).

**Example 2.** Clique.

Input: \((G, k)\) where \( G \) is an undirected graph and \( k \) an integer.

Question: Does \( G \) have a clique of size \( k \)? A clique is a subset of vertices such that every pair of vertices in the subset is joined by an edge.

Output: “Yes” or “no” as appropriate.

Again this is polynomial time verifiable, given the following additional information, or **certificate**: a list of \( k \) vertices which are claimed to form a clique. To verify this it suffices to check that the list comprises \( k \) distinct vertices and that each pair of vertices is joined by an edge. This is readily done in \( O(k^2 n) \) time, and indeed in \( O(n^3) \) time (for if \( k > n = |V| \), then the graph does not have a \( k \)-clique).

**Example 3.** Satisfiability

Input: \( F \), a CNF Boolean formula.

\( F \) is in CNF form if \( F = c_1 \lor C_2 \lor \cdots \lor C_m \), where each clause \( C_i \), \( 1 \leq i \leq m \), is an ‘or’ of literals:

\[
C_i = l_{i1} \lor l_{i2} \lor \cdots \lor l_{ij},
\]

where each \( l_{ik} \) is a Boolean variable or its complement (negation).

\[
e.g. \quad F_1 = (x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor x_3) \land (x_3 \lor x_2); \quad F_2 = x; \quad F_3 = x \land \overline{x}.
\]

Question: Is \( F \) satisfiable? That is, is there an assignment of truth values to \( F \)’s variables that causes \( F \) to evaluate to True?

\[
e.g. \quad \text{For } F_1, \quad x_1 = \text{True}, \quad x_2 = \text{False}, \quad x_3 = \text{True}, \quad F_1 \text{ evaluates to True}; \quad \text{For } F_2, \quad x = \text{True} \text{ causes it to evaluate to True}; \quad \text{for } F_3, \quad \text{no setting of the variables causes it to evaluate to True.}
\]

**Definition** A language \( L \in \text{NP} \) if there are polynomials \( p, q \) an algorithm \( V \) called the verifier, and there is a certificate \( C(x) \) of length at most \( p(|x|) \) such that \( V(x, C(x)) \) outputs
“yes”, while if $x \notin L$, for every string $s$ of length at most $p(|x|)$ $V(x, s)$ outputs “No”; and finally, $V$ runs in time $q(|x| + |C(x)|) = O(q(p(n)))$, where $|x| = n$.

**Comment** If $x \notin L$ there is no certificate for $x$; in other words, any string claiming to be a certificate is readily exposed as a non-certificate.

**Reductions among NP Problems**

We begin with a number of reductions among pairs of problems in NP. The basis form is the following. Given a polynomial time membership algorithm for language $A$ we use it as a subroutine to give a polynomial time algorithm for problem $B$.

**Example 4** Independent Set

Input: Undirected Graph $G$, integer $k$.

Question: Does $G$ have an independent set of size $k$, that is a subset of $k$ vertices with no edges between them?

**Claim 1** Given a polynomial time algorithm for Clique there is a polynomial time algorithm for Independent Set. Let $G = (V, E)$ be the input to the Independent Set problem. Consider the graph $\overline{G} = (V, \overline{E})$. We note that $S \subseteq V$ is an independent set in $G$ if and only if it is a clique in $\overline{G}$.

Thus the Independent Set algorithm is simply to run the Clique algorithm on the pair $(\overline{G}, k)$, and report its result.

□

The next claim is similar and is left to the reader.

**Claim 2** Given a polynomial time algorithm for Independent Set there is a polynomial time algorithm for Clique.

**Example 5** Hamiltonian Path

Input: $(G, s, t)$, where $G = (V, E)$ is a directed graph and $s, t \in V$.

Question: Does $G$ have a Hamiltonian Path from $s$ to $t$, that is a path that goes through each vertex exactly once?

**Claim 3** Given a polynomial time algorithm for Hamiltonian Circuit there is a polynomial time algorithm for Undirected Hamiltonian Path.

**Proof** Let $(G, s, t)$ be the input to the Hamiltonian Path Problem. We build the following graph $H$ such that $H$ has a Hamiltonian Circuit if and only if $G$ has a Hamiltonian Path from $s$ to $t$. $H$ consists of $G$ plus one new vertex, $z$ say, together with new edges $(z, s), (t, z)$.

If $H$ has a Hamiltonian Circuit it includes edges $(z, s), (t, z)$ as these are the only edges incident on $z$; so the circuit has the form $z, s = v_1v_2, \cdots, v_n = t$, and then $s = v_1, v_2, \cdots, v_n = t$ is the corresponding Hamiltonian Path in $G$. Conversely, if $G$ has Hamiltonian Path $s = v_1, v_2, \cdots, v_n = t$ then $H$ has Hamiltonian Circuit $z, s = v_1, v_2, \cdots, v_n = t$. 
Thus the Hamiltonian Path algorithm is to build $H$, run the Hamiltonian Circuit algorithm on $H$ and report its report. Clearly it runs in polynomial time.

**Example 6** Degree $d$ Bounded Spanning Tree

Input: Undirected Graph $G$.

Question: Does $G$ have a spanning tree $T$ such that in $T$ each vertex has degree at most $d$?

**Example 7** Undirected Hamiltonian Path

Input: $(G, s, t)$, where $G = (V, E)$ is an undirected graph, and $s, t \in V$.

Question: Does $G$ have a Hamiltonian Path from $s$ to $t$, that is a path going through each vertex exactly once?

**Claim 4** Given a polynomial time algorithm for Degree $d$ Bounded Spanning Tree there is a polynomial time algorithm for Undirected Hamiltonian Path.

**Proof** Let $(G, s, t)$ be the input to the Hamiltonian Path Problem. We build a graph $H$ such that $H$ has a degree $d$ bounded spanning tree if and only if $G$ has a Hamiltonian Path from $s$ to $t$.

$H$ is the following graph: $G$ plus vertices $v'$ for each $v \in V$, plus vertices $s'', t''$, together with edges $(v', v)$ for each $v \in V$ and edges $(s'', s)$, $(t'', t)$. Note that all the new vertices in $H$ have degree 1 and consequently all the new edges must be in any spanning tree $T$ of $H$. This means that the remaining edges in $T$ form a Hamiltonian Path in $G$ from $s$ to $t$, as each of $s$ and $t$ can have at most one more incident edge, and edge $v \neq s, t$ can have at most two more incident edges; further, to achieve connectivity, these additional edges must be present.

Thus the algorithm is to form $H$, run the degree 3 bounded spanning tree algorithm on $H$ and report its result. Clearly this runs in polynomial time if the degree 3 bounded spanning tree algorithm runs in polynomial time.

□

**Reductions** We observe that all these constructions have the following form, which we denote by $A \leq_P B$, or $A \leq B$ for short. Given a polynomial time algorithm for membership in $B$ we construct a polynomial time algorithm for membership in $A$ as follows.

Input to $A$: $I$.

Step 1. Construct $J(I)$ in polynomial time.

Step 2. Run the membership algorithm for $B$ on input $J(I)$.

Step 3. Report the answer from Step 2.

In order for Step 3 to be correct we need that:

$I \in A \iff J(I) \in B$.

This is called a polynomial time reduction of problem $A$ to problem $B$. It shows that if there is a polynomial time membership algorithm for $B$ then there is also a polynomial time membership algorithm for $A$. The converse is true also: if there is no polynomial time membership algorithm for $A$ then there is not one for $B$ either.