Complexity. Chapter 7.

Home Work 7.1, 7.2 p. 221.

Definition. Let \( M \) be a Turing machine that halts on all inputs. For each input word \( x \), define

\[
T(x) = \text{number of steps that } M \text{ runs on input } x.
\]

The running time or time complexity of \( M \) is the function \( f: N \rightarrow N \) given by

\[
t(n) = \max \{ T_M(x) \mid |x| = n \}.
\]

We will be interested in certain estimates of the running time of algorithms for interesting problems.

E.g. highest order term of an expression: given

\[
t(n) = 6n^3 + 2n^2 + 20n + 45,
\]
we are interested in the order of magnitude \( n^3 \).

N.B. We measure complexity in terms of length of input.
Big-O notation:
Let \( f \) and \( g \) be two functions, \( f, g : \mathbb{N} \to \mathbb{R}^+ \). Define 
\[
f = O(g)
\]
if there exist positive integers \( c \) and \( n_0 \) so that for every integer \( n \geq n_0 \), 
\[
f(n) \leq c g(n) .
\]

\( \text{e.g.} \)

(1) \( 5n^3 + 2n^2 + 2n + 6 = O(n^3) \),
because 
\[
\forall n \geq 10 , \quad 5n^3 + 2n^2 + 2n + 6 \leq 6n^3 .
\]
(2) \( n^3 = O(n^4) \)
(3) \( n^3 \) is not \( O(n^2) \).

Since Big-O notation suppresses constants, the base of logarithms is unimportant.

\[
\log_b n = \frac{1}{\log_b a} \log_a n ,
\]

So, \( \log_b n = O(\log_a n) \).
we just write $O(\log n)$ -- don't care about the base.

Especially, we care about polynomial bounds:

$\alpha(n^c)$, for $c > 0$

and exponential bounds

$2^{n^c}$, $c > 0$.

When we write

$f(n) = O(g(n))$,

$g(n)$ is an upper bound for $f(n)$.

Little-o notation.

Let $f$ and $g$ be two functions $f, g : N \rightarrow \mathbb{R}^+$. Say that $f(n) = o(g(n))$ if

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

This means: $\forall \epsilon > 0 \exists n_0 \forall n \geq n_0$, $f(n) \leq \epsilon g(n)$. 
f grows more slowly than every linear combination of g.

\[ f \sim n^2 = o(n^3). \]

\[ \lim_{n \to \infty} \frac{n^2}{n^3} = \lim_{n \to \infty} \frac{1}{n} = 0. \]

Read pages 251-256. We use the multi-tape Turing machine as the standard model for complexity.

(1) Let \( L = \{0^n \mid n \geq 1 \}. \) L can be decided in \( O(n^2) \), even \( O(n \log n) \), steps on a 1-tape Turing machine. Any algorithm for \( L \) on a 1-tape TM is \( O(n \log n) \).

\[ L \leq T(n) \text{ on a 2-tape TM.} \]

(2) A multi-tape TM can simulate a RAM in time \( O(n^2) \). I.e., if an algorithm runs in \( T(n) \) steps on a RAM, it runs in \( T^2(n) \) steps on a multi-tape TM.
Theorem. Let \( T(n) = n \). Every \( T(n) \) time-bounded Turing machine has an equivalent \( O(T^2(n)) \) single-tape TM.

Theorem. Let \( T(n) = n \). Every \( T(n) \) time nondeterministic Turing machine has an equivalent \( 2^{O(T(n))} \) time deterministic TM.

A nondeterministic Turing machine runs in time \( f(n) \) if for all input strings \( x \), \( |x| = n \), every computation path eventually halts within \( f(n) \) steps.
\[
\text{TIME}(T(n)) = \{ L \mid L \text{ can be decided by some Turing machine in time } O(T(n)) \}.
\]

Recall point (2) above: All reasonable deterministic computational models are polynomial equivalent. Church's thesis states that a TM can simulate every computational model. This extension states that the simulation is reasonably efficient.

We focus attention on the famous class:

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

Class of languages that are decidable in polynomial time on a Turing machine.

Note: This definition blurs distinction between say time \( n \) and time \( n^{100} \). For practical purposes, these would be important distinctions. We know of no problems having polynomial time algorithms that don't already...
have \( n^3 \) time algorithms.

Two important points about \( P \):

1. \( P \) is invariant under different computational models

2. \( P \) corresponds to the class of problems that are realistically solvable on a computer.

Compare \( n^3 \) with \( 2^n \) for reasonable size inputs:

\[
100^3 = 1 \text{ billion} \\
2^{100} > \text{no. of atoms in the universe}
\]
Some typical problems in P.

HW 7.5, 7.6, 7.7

Path problem:

instance digraph G; nodes
s and t of G (G = (V, A), II V)

question Is there a path from s to t?

algorithm:

1. input G, s, and t;
2. mark node s
3. repeat
   for all nodes b of G
   if there is an arc (a, b)
    where a is marked,
    then mark b
   until no new nodes are marked;
4. if t is marked
   then accept else reject.

Analysis: Steps 2 and 4 execute once.
Let m be the number of nodes in G.
The repeat-until loop marks at least one new node every time it executes. Thus steps 3 runs no more than m times. Marking a node
takes constant time. The algorithm runs in \( O(m) \) steps. The length of the input is polynomial in \( m \), (Recall how we encoded graphs.)

Letting \( n = \text{length of input} \), for some \( c > 0 \), \( n = m^c \). Algorithm can be implemented on a TM in time a polynomial in length of input, so

\[
\text{PATH} = \{ (G, s, t) \mid G \text{ is a digraph}, s \text{ and } t \text{ are nodes, there is a path from } s \text{ to } t \} \in P.
\]
Let $HAM = \{ G \mid G$ has a Hamiltonian Circuit $\}$.

A verifier for a language $A$ is an algorithm $V$ s.t. $A = \{ w \in \{0,1\}^* \mid V accepts w \}$. A polynomial time verifier runs in polynomial time in the length of $w$.

A string $c$ is called a certificate if it is polynomially verifiable, i.e., if $c$ has a polynomial time verifier.

Recall the Hamiltonian Circuit problem.

Instance: Graph $G$.

Question: Does $G$ contain a Hamiltonian Circuit?
$V(<G, p)$ is an algorithm that verifies that $p$ is a Ham. circuit of $G$.

Let $\text{COMP} = \{ x \mid \exists p, q > 1, x = pq \}$

$\text{COMP}$ has a polynomial time verifier.
A certificate is a value $p > 1$ that divides $x$.

**Defn.** $\text{NP} = \{ L \mid L \text{ has a polynomial time verifier} \}$.

$\text{NTIME}(T(n)) = \{ L \mid L \text{ can be accepted by some nondeterministic Turing machine in time } O(T(n)) \}$.

**Theorem.** $\text{NP} = \bigcup_{k} \text{NTIME}(n^{k})$.

**Proof.** $\Rightarrow$: Let $A \in \text{NP}$ and let $V$ be a polynomial time verifier for $A$. Assume $V$ runs in time $n^k$.

Design $N$ to accept $A$ as follows:

1. Input $w$, $|w| = n$
2. Nondeterministically select $c$, $|c| = n^k$
3. Run $V$ on $<w, c>$
4. If $V$ accepts then accept.
Clearly $N$ runs in time $O(n^k)$.

$\leq$: Recall that an accepting computation of a Turing machine $N$ is a sequence of ID's,
$I_0, \ldots, I_s$
s.t. $I_0$ is an initial ID,
$I_s$ is an accepting ID,
and $I_i \vdash I_{i+1}$, for $0 \leq i < s$.

Let $N$ be a nondeterministic TM that accepts $A$ in time $n^k$.

A certificate $c$ for a string $w$, $|w| = n$, is an accepting sequence of ID's,
$c = \langle I_0, \ldots, I_s \rangle$.

A polynomial time verifier is an algorithm $V$ that takes $w$ and $c$ as input and accepts iff $c$ is an accepting computation of $N$ on $w$. 
Clique. A clique is a complete subgraph, i.e., every two nodes is connected by an edge.

Clique:

instance, graph $G$, $k \geq 0$
question: Is there a clique of size $k$?

The size of a clique is the number of nodes.

The clique is NP.

Two views:

verifier proof:
1. input to $V$, graph $G$, size $k$,
alleged clique $C$
2. test whether $C$ contains $k$ nodes of $G$
3. test whether $G$ contains an edge for every pair of nodes in $C$
4. if both tests pass, then accept
else reject
nondeterminism view:

1. Nondeterministically choose a subset of $k$ nodes $c$.
2. Test whether $G$ contains all edges connecting nodes in $c$.
3. If yes, accept.
   (note: we don't use reject)

Open question: $P = NP$?

We know

$$P \leq NP \leq \text{DTIME}(2^{n^k})$$
Def. A is polynomial time reducible to B if there is a polynomial time computable fn. f such that for every w, 
\[ w \in A \iff f(w) \in B. \]

We write \( A \leq_p B \).

Theorem. If \( A \leq_p B \) and \( B \in P \), then \( A \in P \).
\[ x \in A \land f(x) \in B \]

Suppose \( f \) is computable in \( n^2 \) steps.

Let \( B \in \text{P} \). Find \( M \)'s t. \( B = L(M) \).

Suppose \( M \) runs in time \( n^3 \).

\[ \text{alg to decide } A: \]
1. input \( w, |w| = n \).
2. compute \( f(w) \).
3. run \( M \) on \( f(w) \).
   \[ \{ \text{takes time } \leq (n^2)^3 = n^6 \} \]
4. if \( M \) accepts then accept else reject

Total time is \( O(n^2 + n^6) = O(n^6) \).

So \( A \in \text{P} \).

HW: Look at problem 7.27. Explain why 3-color \( \notin \text{P} \).