Chapter 7: Computational Complexity

A. Introduction.

1. Motivation
   - We have seen that some problems are not solvable if the model of computation is too simple.
   - We have seen that some problems are not solvable by any reasonable finite model of computation.
   - But what about efficiency?! Are some problems intrinsically harder than others? Are there interesting problems that are theoretically solvable but practically unsolvable?
2. Time-bounded Turing machines.
   - **Definitions.** A TM M decides a language L in time T if M returns **Y** or **N** in a number of steps $\leq T(|w|)$ for input w. We say L is decidable in time T, or $L \in TIME(T)$.
   - Example.
   - Remarks:
     - these are "upper bounds", or "worse case complexities."
     - sometimes "average case" is of interest.
     - an upper bound is "tight" if it is achieved by some specific known example.
     - an algorithm is "optimal" if it achieves some problem-dependent lower bound.
3. Rate of growth of functions.
   - **Definition.** $f = O(g)$ iff there is a constant $c > 0$ and an integer n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
   - Examples.
   - **Theorem 7.2.1.** If $f(n)$ is a polynomial then $f = O(r^n)$ for any $r > 1$.
   - Remarks:
     - For our purposes, constants don't matter because ...
       - **Theorem 7.1.1.** If $L \in TIME(T)$ and $\varepsilon > 0$, then $L \in TIME(T')$, where $T'(n) = 2n + 18 + \lceil \varepsilon T(n) \rceil$.
       - If $f = O(g)$ then $cf = O(g)$ for any constant c.
     - For other purposes, constants can matter very much.
     - For our purposes, the distinction between polynomial and exponential time complexities is the only one that matters.
     - For other purposes, distinctions among polynomial complexities matter very much.
4. Simulating one TM with another.
   - If a computation takes time $T_1(n)$ on a TM with a 2-way infinite tape, it can be simulated in time _____on a standard TM.
   - If a computation takes time $T_1(n)$ on a k-tape machine, it can be simulated in time _____on a standard TM.
   - Remark: the simulation cost does not take us out of polynomial complexity.

B. $\mathcal{P}$ vs. $\mathcal{NP}$

1. Definitions.
   The set of polynomial-time decidable languages is

$$\mathcal{P} = \bigcup\{TIME(n^d) \mid d > 0\}.$$

   The set of (nondeterministic) polynomial-time decidable languages is

$$\mathcal{NP} = \bigcup\{NTIME(n^d) \mid d > 0\},$$

   where $NTIME(T)$ is set of languages decided by some nondeterministic TM in time T.

2. Examples:
   Integer programming. Let $A \in \mathcal{N}^{m \times n}$ and $b \in \mathcal{N}^m$. Then

$$\{(A,b) \mid \exists x \in \{0,1\}^n, \text{ such that } Ax = b\} \in \mathcal{NP}$$

$$\{(A,b,x) \mid x \in \{0,1\}^n, \text{ and } Ax = b\} \in \mathcal{P}$$

   Traveling salesman. Given a list of n cities, find a route that visits them all and whose length is $\leq M$.

3. Remarks:
   - A "Church-like" thesis: $\mathcal{P}$ corresponds to the class of realistically solvable problems.
   - $\mathcal{P} \subseteq \mathcal{NP}$. But does $\mathcal{P} = \mathcal{NP}$?
   - There is much evidence that $\mathcal{P} \neq \mathcal{NP}$, but no proof as yet.
   - A typical hard problem in $\mathcal{NP}$:
     - The problem is to find an x that has some property.
     - The only known algorithm is exponential.
     - But if you could guess an x, checking whether it solves the problem can be done in polynomial time.

C. NP-completeness

1. Definitions.
   - A **polynomial-time transformation** is a function which transforms one string into another in polynomial time.
   - A language $L \in \mathcal{NP}$ is **$\mathcal{NP}$-complete** iff for every $L' \in \mathcal{NP}$ there exists a polynomial-time transformation from L' to L.
2. **Theorem 7.5.1.** If L is $\mathcal{NP}$-complete, then $\mathcal{P} = \mathcal{NP}$ iff $L \in \mathcal{P}$.
3. **Theorem 7.5.2.** N_0 is $\mathcal{NP}$-complete, where

$$N_0 = \{\rho(M)\rho(w)@I^t \mid \text{the NDTM } M \text{ accepts input } w \text{ in at most } t \text{ steps}\}.$$

4. Examples of $\mathcal{NP}$-complete problems.