Handout 12: Complexity Classes

12.1 O notation. Let $f\colon \mathbb{N} \to \mathbb{N}$ be a function over the natural numbers $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$. Then $O(f)$ is the *set* of functions over $\mathbb{N}$ that are almost everywhere dominated by a constant multiple of f:

$$O(f) \;=\; \{g : \mathbb{N} \to \mathbb{N} \mid (\exists n_0, m)(\forall n \geq n_0)(g(n) \leq m \cdot f(n))\}$$

If $g \in O(f)$, then f is an *asymptotic* upper bound for g; we also say that g grows *asymptotically* no faster than f. An expression $e(O(f))$ stands for the set

$$\{g : \mathbb{N} \to \mathbb{N} \mid (\exists n_0, m)(\forall n \geq n_0)(g(n) \leq e(m \cdot f(n)))\}.$$

For example,

$$2^{O(f)} \;=\; \{g : \mathbb{N} \to \mathbb{N} \mid (\exists n_0, m)(\forall n \geq n_0)(g(n) \leq 2^{m \cdot f(n)})\}.$$

12.2 Time complexity. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ be a Turing decider (possibly nondeterministic); that is, for every input w, every run of M over w halts (in q_a or q_r) in a finite number of steps. (One step is one application of the transition relation δ.) The *time complexity* (or *running time*) of M is a function $t_M\colon \mathbb{N} \to \mathbb{N}$ over the natural numbers, namely,

$$t_M(n) \;=\; \max\{k \mid (\exists w \in \Sigma^n)(\exists \text{ run } r \text{ of } M \text{ on input } w)(r \text{ halts in } k \text{ steps})\}$$

for all $n \geq 0$. Thus, the TD M decides all inputs of length n in at most $t_M(n)$ steps.

12.3 Complexity classes. For every function $f\colon \mathbb{N} \to \mathbb{N}$, define:

$$\text{TIME}(f) \;=\; \{L(M) \mid M \text{ is a deterministic Turing decider and } t_M \in O(f)\}$$
$$\text{NTIME}(f) \;=\; \{L(M) \mid M \text{ is a nondeterministic Turing decider and } t_M \in O(f)\}$$

Note that $\text{TIME}(f) \subseteq \text{NTIME}(f)$.

12.4 Polynomial time. Complexity classes like $\text{TIME}(f)$ are not robust with respect to the exact definition of Turing machines (or algorithms). For example, with two tapes, the language $\{\mathtt{0}^k\mathtt{1}^k \mid k \geq 0\}$ can be decided in deterministic time $O(n)$, but with one tape, $O(n \cdot \log n)$ is the best we can do. Note that the difference is polynomial. In general: for every deterministic Turing decider M with two tapes, there is a deterministic Turing decider M' with one tape such that $L(M') = L(M)$ and $t_{M'} \in O(t_M(n)^2)$. Hence we define:

$$\text{P} \;=\; \bigcup_{k \geq 0} \text{TIME}(n^k)$$
$$\text{NP} \;=\; \bigcup_{k \geq 0} \text{NTIME}(n^k)$$

These classes are robust with respect to sensible definitions of algorithms. However, we do not know if they are robust with respect to nondeterminism. Whether P = NP is perhaps the most famous open problem in computer science. The best difference we can prove is exponential: for every nondeterministic Turing decider M, there is a deterministic Turing decider M' such that $L(M') = L(M)$ and $t_{M'} \in 2^{O(t_M)}$.

12.5 Deterministic upper bounds. To prove a deterministic upper bound for a language (or problem) A, we must construct a deterministic Turing decider (or algorithm) for A whose time complexity is as desired. For example, to prove that $A \in \text{TIME}(f)$, we must construct a deterministic Turing decider M such that $L(M) = A$ and $t_M \in O(f)$; or to prove that $B \in \text{P}$, we must construct a deterministic Turing decider M and a natural number $k \geq 0$ such that $L(M) = B$ and $t_M \in O(n^k)$.

Consider, for example, the following problem:

PATH $= \{\langle G, s, t\rangle \mid$ the directed graph G has a directed path from node s to node $t\}$.
The following DTD M proves that PATH $\in$ P: On input $\langle G, s, t\rangle$ where G is a directed graph with nodes s and t, M (i) marks node s; (ii) until no more nodes are marked, it scans all edges of G and if an edge (a, b) is found going from a marked node a to an unmarked node b, then it marks node b; (iii) if t is marked, then ACCEPT; otherwise REJECT. Clearly, M runs in polynomial time in the size of G (the loop in step (ii) runs in polynomial time in the numbers of nodes of G).

12.6 Nondeterministic upper bounds. To prove a nondeterministic upper bound, we first *guess* a certificate (nondeterministic) and then *verify* that the guess is correct (deterministic). To prove the upper bound NP, both the guess and the verification must happen in polynomial time. Formally, a TM M_1 *decides* a language A if all runs of M_1 halt and

$$A = \{w \in \Sigma^* \mid (\exists r)(r \text{ is an accepting run } r \text{ of } M_1 \text{ over } w)\} = L(M_1);$$

a TM M_2 *verifies* a language A if all runs of M_2 halt and

$$A = \{w \in \Sigma^* \mid (\exists c \in \Sigma^*)(\exists r)(r \text{ is an accepting run } r \text{ of } M_2 \text{ over } w\#c)\}.$$

The word c is called a *certificate* (or *witness*) for the fact that $w \in A$. By definition, a language A is recursive iff there is a (deterministic or nondeterministic) decider for A. We can show that

A is r.e. iff there is a (deterministic or nondeterministic) verifier for A

(take the number of steps of a deterministic Turing recognizer for A as certificate). The decider M_1 is *polynomial-time* if there exist $n_0, m, k \geq 0$ such that for all inputs w with $|w| \geq n_0$, all runs of M_1 over w halt in $m \cdot |w|^k$ steps. The verifier M_2 is *polynomial-time* if there exist $n_0, m, k \geq 0$ such that for all inputs $w\#c$ with $|w| \geq n_0$, all runs of M_2 over $w\#c$ halt in $m \cdot |w|^k$ steps. By definition, a language A is in P iff there exists a deterministic polynomial-time decider for A, and A is in NP iff there exists a nondeterministic polynomial-time decider for A. We have the following equivalent characterization of NP in terms of *deterministic* TMs:

The language A is in NP iff there exists a deterministic polynomial-time verifier for A.

(Take the sequence of choices made by a nondeterministic polynomial-time Turing decider for A as certificate.) So, in order to prove that $A \in$ NP, we need to (1) specify a (polynomial-size) certificate c for every word $w \in A$, and (2) construct a deterministic polynomial-time verifier for A, which checks on input $w\#c$ that c is a proper certificate for w. (The certificate c needs to be necessarily polynomial-size in w, because a polynomial-time verifier can only look at polynomial-size certificates.)

12.7 Examples of certificates. Consider the following problems:

HAMPATH $= \{\langle G, s, t\rangle \mid$ the directed graph G has a Hamiltonian path from vertex s to vertex $t\}$.
CLIQUE $= \{\langle G, k\rangle \mid$ the undirected graph G has a clique with k vertices$\}$.
SAT $= \{\phi \mid$ the boolean formula ϕ has a satisfying truth-value assignment to its variables$\}$.
COMPOSITES $= \{k \mid$ the integer k has two factors greater than 1$\}$.
SUBSETSUM $= \{\langle X, k\rangle \mid$ the multiset X of natural numbers has a submultiset whose elements add up to $k\}$.

A certificate for HAMPATH is a list $c = v_0 v_1 \ldots v_m$ of vertices. The corresponding verifier checks (in time polynomial in G) that c contains every vertex of G exactly once, that $v_0 = s$, that $v_m = t$, and that for all $0 \leq i < m$, there is an edge from v_i to v_{i+1} in G. In other words, the certificate for a YES-instance $\langle G, s, t\rangle$ is a Hamiltonian path in G from s to t.

A certificate for CLIQUE is also a list $c = v_1 \ldots v_m$ of vertices. The corresponding verifier checks (in time polynomial in G) that $m = k$ and that for all $1 \leq i < j \leq m$, there is an edge between v_i and v_j in G. In other words, the certificate for a YES-instance $\langle G, k\rangle$ is a k-clique of G.

A certificate for SAT is a list $c = (x_1, b_1) \ldots (x_m, b_m)$ of pairs of variables x_i and truth values $b_i \in \{0, 1\}$.. The corresponding verifier checks (in time polynomial in ϕ) that $\{x_1, \ldots, x_m\}$ is the set of variables that

occur in ϕ, and that ϕ evaluates to true if each variable x_i is replaced by the boolean b_i, for all $1 \leq i \leq m$. In other words, the certificate for a YES-instance ϕ is a satisfying truth-value assignment for the variables in ϕ.

A certificate for COMPOSITES is a pair $c = (p, q)$ of numbers. The corresponding verifier checks (in time polynomial in $\log k$) that $p > 1$ and $q > 1$ and $k = p \cdot q$. In other words, the certificate for a YES-instance k is a pair of nontrivial factors of k.

A certificate for SUBSETSUM is a list $c = y_1 \dots y_m$ of numbers. The corresponding verifier checks (in time polynomial in the encoding of X and k) that $\{y_1, \dots, y_m\}$ is a submultiset of X, and that $y_1 + \cdots + y_m = k$. In other words, the certificate for a YES-instance $\langle X, k \rangle$ is a submultiset of X whose elements add up to k.