Advanced Complexity Theory — Madhu Sudan
6.841/18.405J
Due: Monday, Feb 25, 2001

Problem Set 1

Enclosed are some sample of correct solutions to the first problem set. Solutions include some of my comments.

Problems

1. The integer factorization function takes as input an n bit integer X and outputs a list of primes $p_1, \ldots, p_\ell$ such that $X = \prod_{i=1}^{\ell} p_i$. Give a language that is "equivalent" to the integer factorization problem. (Include a precise definition of the notion of "equivalence" in your answer.)

   `Main issue here is to make sure you realize that languages capture computational problems effectively. The exercise of defining ''equivalence'' is motivated by this goal. Most people got the question right, once the definition was in place. This solution is taken from Eleni Drinea.`

   We define the following language:

   $$L = \{(x, \alpha, \beta) \mid 1 \leq \alpha \leq \beta \leq x \ \& \text{ exists a prime factor of } x \text{ in } [\alpha, \beta]\},$$

   where x, α, β are integers. This language is equivalent to the integer factorization function F under the following notion of equivalence:

   (a) If one can compute the function problem, one can efficiently decide membership in L. I.e., there exists a polynomial time TM $M^?$ that with access to an oracle for the factorization function, decides L.

   (b) If one can decide L, then one can efficiently compute F. I.e., there exists a polynomial time TM $T^?$ that with access to an oracle for L, computes F.

   We say that L is equivalent to F if and only if both of the above hold. In the following, we show why this is the case.

   - L reduces to F: $M^F =$ "On input (x, α, β)
     (a) If not $1 \leq \alpha \leq \beta \leq x$, *reject*.
     (b) Query the oracle F on the factorization of x (thus get a list of prime factors).
     (c) For every prime factor, check whether it is in $[\alpha, \beta]$.
     (d) If there is at least one prime factor in $[\alpha, \beta]$, *accept*.
     Otherwise *reject*."

     The size of the list output by F is polynomial in the size of x, the size of the prime factors of x is at most n, thus traversing the list of the prime factors and checking whether they are contained in the interval requires time polynomial in n. Thus the running time of M^F is polynomial in n.

- F reduces to L: $T^L =$ "On input x
  (a) Query oracle L on $(x, 2, x-1)$.
  (b) If answer is "no", output x and *halt*.
  (c) Do binary search for the largest prime factor p_1 of x with initial interval $[2, x]$, using the oracle to decide existence of such a factor in the proper intervals.
  (d) Output p_1.
  If $\frac{x}{p_1} > 1$, set $x = \frac{x}{p_1}$ and go to 3;
  otherwise, *halt*."

The binary search takes $O(n)$ time and we need at most n binary searches. Thus the running time of T^L is polynomial in n.

2. Given a language $L \subseteq \{0,1\}^*$, let $L_n = L \cap (\cup_{i=0}^{n}(\{0,1\}^i))$. We say that L is *self-reducible* if there exists a polynomial time oracle Turing machine M such that for every $x \in \{0,1\}^n$,

$$x \in L \quad \Leftrightarrow \quad M^{L_{n-1}}(x) \text{ accepts.}$$

   (a) Given an example of a self-reducible language.

   (b) Prove that if L is self-reducible, then L is in PSPACE.

I made an error in not asking for a self-reducible language that is NOT in P. So most people gave trivial languages, which is perfectly legitimate. More interesting answers included SAT, and TQBF. The second part, less trivial, wanted an upper bound on the complexity of self-reducible problems. Here solutions tended to be wordy, and this often results in ambiguity. If you are describing algorithms, pick the language of algorithms to describe your solution. And on at least a couple of occasions I noticed people used the phrase ``poly(x) + poly(x) is still poly(x)'' to analyze the space bound. While the phrase is correct, it does not yield the space bound! Below is a solution taken from Vahab Mirrokni.

- Any language in P is also a self-reducible language. Because in this case, the oracle TM doesn't need the oracle and can be executed in polynomial time. We can define for example this language: $L = \{x \in \{0,1\}^* |$ number of 1's in x is even $\}$. It is easy to see that $x : |x| = n$, is in L iff $x_1 \dots x_{n-1}$ is in L and x_n is 0 or $x_1 \dots x_{n-1} \notin L$ and $x_n = 1$. It means that we have an oracle Turing machine using the oracle for L_{n-1} for deciding strings of length n. Thus, L is self-reducible.
- We prove it by induction on the size of string. For induction hypothesis, we know that there exists a PSPACE TM for strings of length less than n. In order to construct a polynomial space TM for deciding strings of length n, we simulate the polynomial time oracle TM $M^{L_{n-1}}$, when this machine access the oracle L^{n-1}. From induction hypothesis we know that there exists a polynomial space turning machine for deciding it. We use this TM to find the answer yes or no of the oracle, then we follow the algorithm. Notice whenever we call the oracle L_{n-1}, we reuse the previous space and we know that M takes polynomial space(time) and this new TM for the oracle takes polynomial space as well. More precisely, from a lemma in the class we know that oracle TM M^A can be simulated with a TM M' which needs space(M)+space(A) space and by induction a TM for deciding L needs at most $n \times$space(M). Thus, the overall space, used by this algorithm is polynomial. Thus, we can decide strings of length n in PSPACE as well; and it completes the induction step. For induction basis, it's easy to see that strings of length 1 are decidable in PSPACE.

3. Prove that there exists an oracle A such that $\mathrm{NP}^A \neq \mathrm{co-NP}^A$.

`Main goal here was to make sure you went over the details that I did not go over in class; and to see where things change because we are now comparing the existential quantifier with the universal quantifier. Below is a solution taken from David Woodruff.`

We construct an oracle A such that $NP^A \neq co\text{-}NP^A$. We need a language $L \in co\text{-}NP^A$ but $L \notin NP^A$. Define $L(A)$ as follows:

$$L(A) = \{w \mid \forall x \in A : |x| \neq |w|\}$$

Intuitively, $L(A)$ is the language of words whose length differ from the length of every other word in A. It is clear that $L(A) \in co\text{-}NP^A$ since given input w, we can query A on each string of length $|w|$ and *accept* iff A rejects all such strings.

We want to show there is an A for which $L(A)$ is not in NP^A. We need to show that there is an A such that for all nondeterministic polynomial time OTMs $M^?$, $L(M^A) \neq L(A)$. First, we let $M_1, M_2, \ldots$ be an enumeration of all nondeterministic polynomial time oracle TMs. We construct a sequence
$A_0 \subseteq A_1 \subseteq \ldots \subseteq A$, with lengths $n_1 < n_2 < \ldots$, such that

(a) $M_i^{A_i}(0^{n_i}) \neq (0^{n_i} \in L(A_i))$.
(b) $M_i^{A_i}(0^{n_i}) = M_i^{A}(0^{n_i})$.
(c) $0^{n_i} \in L(A_i) \Leftrightarrow 0^{n_i} \in L(A)$.

The construction algorithm, modified slightly from last year's 6.841 Lecture 6 lecture notes, is as follows:

(a) First let $A_0 = \emptyset$.
(b) For $i = 1, 2, 3, \ldots$ do
   i. Choose n_i such that $\forall j < i$, $M_j^{A_{i-1}}$ does not ask about any strings of length n_i on input 0^{n_j}. In addition, $M_i^{A_{i-1}}$ should run for fewer than 2^{n_i} steps on input 0^{n_i}. Such an n_i can be found if taken to be large enough.
   ii. Simulate M_i with oracle A_{i-1} on input 0^{n_i}. Currently A_i has no strings of length n_i. By construction, A_{i-1} will answer no to all queries of length n_i.
   iii. If $M_i^{A_{i-1}}$ rejects, set $A_i = A_{i-1}$. (So $M_i^{A_i}$ has rejected 0^{n_i}, but $0^{n_i} \in L(A_i)$).

   If $M_i^{A_{i-1}}$ accepts, then every accepting path of $M_i^{A_{i-1}}$ makes only a polynomial number of oracle queries. Hence, for any such path we can find an x such that $|x| = n_i$ and $M_i^{A_{i-1}}$ on input 0_i^n did not ask x. Then set
   $$A_i = A_{i-1} \cup x.$$
   (So we have $M_i^{A_{i-1}}$ accepting 0^{n_i}, but $0^{n_i} \notin L(A)$)

Set A $= \cup_{i=0}^{\infty} A_i$.

4. Show that any single-tape, single-head Turing machine recognizing the "palindrome" language $\{xx^R | x \in \{0,1\}^*\}$ (where x^R denotes the reversal of the string x) must take time $\Omega(n^2)$.

Several solutions had the right intuition, that a finite space machine has to run fom one end to the other linearly many times to check palindromes, but were unable to formalize it. The solution below should help see how to formalize such intuitions. Some of you observed the connection to ''Kolmogorov complexity''. Yet claiming that ''Kolmogorov complexity'' of most strings is linear in n is an unnecesarily strong argument for this problem. It is better to resort to only as much external material as you need - and the Kolmogorov theory, relying on a universal machine etc., is unnecessary. (One solution mentioned the connection and then gave the first principles proof - I would have liked to include it - except I didn't have the source.) The solution below is a first principles argument, taken from Grant Wang.

First, note that it is easy to construct a TM M that accepts $L = \{xx^R | x \in \{0,1\}^*\}$ that takes $\Theta(n^2)$. The machine simply moves back and forth on the input string, recording the leftmost symbol not yet checked in the finite control, and checking that this symbol is the rightmost symbol not yet checked. It can determine which symbols have not been checked yet by marking a special tape symbol. The machine rejects if such a pair (leftmost, rightmost) do not contain the same symbol, and accepts if the entire string has been rewritten in the special tape symbol. Note that it moves back and forth n times, so the number of steps the machine takes is:

$$\sum_{i=1}^{n} n - i = \Theta(n^2)$$

Now, we wish to show that we can't do any better. Consider a TM M that decides L, and let $L_{\#}$ the subset of length n strings of the form $\{x\#^{\frac{n}{2}}x^R | x \in \{0,1\}^{\frac{n}{4}}\}$. Consider M operating on an input string. In particular, consider the state of the finite control as M crosses (either left or right) between the i and $i+1$th symbols on an input string. For a particular index i, let $c_i(x)$ be the sequence of finite control states M is in when crossing between index i and $i+1$, in either direction.

Consider $x \neq y, x, y \in L_{\#}$, and consider the middle portion of the two strings x, y. Then there cannot exist $\frac{n}{4} \leq i \leq j \leq \frac{3n}{4}$ such that $c_i(x) = c_j(y)$. Suppose such i, j exist. Then consider the string z that is concatenation of the prefix of x of length i, and the suffix of y of length $n - j$. Then M accepts z, because everything that occurs to the left of i exactly simulates M on the prefix of x, and everything to the right of i exactly simulates M on the suffix of y, since it enters the correct state on either side when crossing between i and $i+1$.

This provides us with a method to lower bound the running time of M. Consider each $x \in L_{\#}$, and let s_x be the shortest sequence, i.e. $s_x = \min_i c_i(x)$, for all $\frac{n}{4} \leq i \leq \frac{3n}{4}$. Let s_{max} be the maximum length sequence over all x of s_x. Since for each $x, y \in L_{\#}$ it must be the case that each $c_i(x) \neq c_j(y)$ for all $\frac{n}{4} \leq i, j \leq \frac{3n}{4}$, s_{max} must be large, or else two strings will have the same sequence. There are $2^{\frac{n}{4}}$ such strings (note that we are only considering strings of the form $x\#^{\frac{n}{2}}x^R$). What is the smallest possible value of $|s_{max}|$? Note that each such string must have a unique smallest sequence, by the above paragraph. If we let Q be the number of states of M, we must have that the number of all possible sequences of length less than

$|s_{max}|$ is at least $2^{\frac{n}{4}}$, so we have:

$$\begin{aligned}
\sum_{i=1}^{|s_{max}|} |Q|^i &\geq 2^{\frac{n}{4}} \\
\frac{|Q|^{|s_{max}|+1} - 1}{|Q| - 1} &\geq 2^{\frac{n}{4}} \\
\log \frac{|Q|^{|s_{max}|+1} - 1}{|Q| - 1} &\geq \log 2^{\frac{n}{4}} \\
|s_{max}| &\geq \Theta(n)
\end{aligned}$$

So we have that for a particular string $x \in L_{\#}$ (namely, the string x for which s_x was maximum), the smallest sequence is $\Theta(n)$. But then, for each of the $\frac{n}{2}$ positions, each sequence is $\Theta(n)$. This implies that the TM M must at least generate these sequences, which is $\frac{n}{2} \cdot \Theta(n)$ steps, i.e. $\Theta(n^2)$ steps, which is the result we desired.

5. Let LIN-SPACE be the class of languages recognizable in linear space. Show that LIN-SPACE $\neq$ P.

   `This was a relatively straightforward problem. Key is to note that while we prove inequivalence, we don't know if any side is more powerful than the other. The solution below is taken from Seth Gilbert.`

   Define the function $pad(A, k)$ as it is defined by Sipser in exercise 9.18:

$$\begin{aligned} pad(s, l) &= s\#^j \text{ where } j = \max(0, l - n \text{ and } n \text{ is the length of } s \\ pad(A, f(n)) &= \{pad(s, f(n))| \text{ where } n \text{ is the length of } s\} \end{aligned}$$

   The function *pad* effectively makes an input longer, giving a Turing machine more time (or space) with which to work. The first thing to note is that $A \in P$ if and only if $pad(A, n^k) \in P, \forall k$. Clearly if A can decide input x in polynomial time, it can decide $pad(x, n^k)$ in polynomial time: the input length is now longer, providing more time, while the amount of calculation necessary has not changed. Conversely, if $pad(A, n^k)$ can be decided in polynomial time, say $O(n^p)$, then clearly A can be decided in $O((n^k)^p))$: an input of length n to A is equivalent to an input of length n^k to $pad(A, n^k)$, which will take $O((n^k)^p)$ time.

   Assume for the sake of contradiction that LIN-SPACE $= P$. Let B be a language that can be decided in $O(n^2)$ space, but cannot be decided in $O(n)$ space. That is, B is a language that really requires $O(n^2)$ space to compute. The space-hierarchy theorem shows that such a language exists. We know that $pad(B, n^2)$ is in LIN-SPACE, since given an input to B of length n, we need $O(n^2)$ space to solve it, and given the same input to $pad(B, n^2)$, we in fact have $O(n^2)$ space available (because of the padding). (If you are worried about constant factors, use a higher order polynomial than n^2 to pad with: there exists a k such that $pad(B, n^k) \in$ LIN-SPACE.) This implies that $pad(B, n^k) \in$ P, which implies (by the reasoning above) that $B \in$ P. By our assumption, this means that $B \in$ LIN-SPACE. However this contradicts our definition of B as a language that can not be decided in linear space.