CMSC 303 Introduction to Theory of Computation, VCU
Spring 2017, Assignment 7
Due: Thursday, April 27, 2017 in class

Matthew Bowers

Total marks: 52 marks + 5 marks bonus for typing your solutions in LaTeX using the provided A7 template file.

Unless otherwise noted, the alphabet for all questions below is assumed to be $\Sigma = \{0, 1\}$. This assignment focuses on the complexity classes P and NP, as well as polynomial-time reductions.

1. [12 marks]

   (a) [4 marks] Let $f(n) = 4n^2 - 30n + 6$. Prove that $f(n) \in O(n^2)$. In your proof, give explicit values for c and n_0.

   Proof. $f(n) = 4n^2 - 30n + 6 = O(n^2)$ That is $4n^2 - 30n + 6 <= cn^2$ for some c and $n \geq n_0$ Let $c = 4$ Let $n_0 = 1$ Taking the derivative of both functions shows that the slope of $f(n)$ is always less than $8n$ which is the slope of our comparison function. Further since these are both second order polynomials with positive leading coefficients we know that they each have exactly 1 minimum and zero points of inflection. Therefore if $4n^2 > 4n^2 - 30n + 6$ at any point beyond $4n^2$'s global minimum it will continue to be greater forevermore. The global minimum of $4n^2$ is at $n = 0$. At $n = 0$ $f(n) > 4n^2$. $f(1) = -20 < 4 * 1^2$ At this point we know that our reference function will never decrease and will always grow faster than $f(n)$ so it will always be greater. □

   (b) [4 marks] Let $f(n) = n^{999}$ and $g(n) = (\sqrt{\log n})^{\sqrt{\log n}}$. Precisely one of the following two claims is true: $f(n) \in O(g(n))$, $g(n) \in O(f(n))$. Prove whichever one is true. In your proof, give explicit values for c and n_0. (Hint: Note that an arbitrary function $f(n)$ can be rewritten as $2^{\log_2(f(n))}$.)

   To simplify Let $x = \log n$ then we can rewrite $f(x) = 10^{x^{999}}$ and $g(x) = \sqrt{x}^{\sqrt{x}}$. Applying the $2^{\lg}$ we can compare $x^{999} \lg 10$ to $\frac{1}{2}x^{\frac{1}{2}} \lg x$ It seems that x^{999} is going to win out so to simplify further: we know that $\lg x = O(x^{\frac{1}{2}})$ so the second expression can be written $\frac{1}{2}x^{\frac{1}{2}}x^{\frac{1}{2}} = \frac{1}{2}x^{\frac{1}{4}}$ So $g(n) \in O(f(n))$ $c = 1$ and $n_0 = 1$

   (c) [4 marks] This question tests an important subtlety in the definition of "polynomial-time". One of the most famous open problems in classical complexity theory is whether the problem of factoring a given integer N into its prime factors is solvable in polynomial time on a classical computer[1]. Given positive integer N as input, why is the following naive approach to the factoring problem not polynomial-time?

   1: Set $m := 2$.
   2: Set $S := \emptyset$ for S a multi-set.
   3: **while** $m \leq N$ **do**
   4: **if** m divides N **then**
   5: Set $S \cup \{m\}$.
   6: Set $N = N/m$.

[1] In fact, many popular cryptosystems are based on the assumption that this problem is *not* efficiently solvable. Recall from class, however, that in 1994 it was shown by Peter Shor that *quantum* computers can efficiently solve this problem!

```
    else
        Set m = m + 1.
    end if
end while
Return the set S of divisors found.
```

The input of this problem should not be considered as the number you are entering to factor, but rather the size of the encoding of the number you are entering. By increasing the size of the input by a single base 2 digit you are doubling the number of divisions which the program must carryout.

2. [6 marks] Let co-NP denote the complement of NP. In other words, intuitively, co-NP is the class of languages L for which if input $x \notin L$, then there is an efficiently verifiable proof of this fact, and if $x \in L$, then no proof can cause the verifier to accept.

   Prove that if P = NP, then NP is closed under complement, i.e. NP = co-NP. How can closure of NP under complement hence potentially be used to resolve the P vs NP question?

   If $P = NP$ then any NP problem can be decided in poly-time. This would mean that the machine M which decides L_{NP} could be used as a verifier for $\overline{L_{NP}}$. If $L_{NP} = \overline{L_{NP}}$ then we could combine the poly-time verifiers to decide L_{NP}

3. [10 marks] In class, we introduced the language

$$\text{CLIQUE} = \{\langle G, k\rangle \mid G \text{ is a graph containing a clique of size at least } k\}.$$

   Consider now two further languages:

$$\begin{aligned} \text{INDEPENDENT-SET} &= \{\langle G, k\rangle \mid G \text{ is a graph containing an independent set of size at least } k\} \\ \text{VERTEX-COVER} &= \{\langle G, k\rangle \mid G \text{ is a graph containing a vertex cover of size at most } k\}. \end{aligned}$$

   Here, for graph $G = (V, E)$, an *independent set* $S \subseteq V$ satisfies the property that for any pair of vertices $u, v \in S$, $(u, v) \notin E$. A *vertex cover* $S \subseteq V$ satisfies the property that for any edge $(u, v) \in E$, at least one of u or v must be in S.

   (a) [4 marks] Prove that CLIQUE $\leq_p$ INDEPENDENT-SET. (Hint: Given a graph G, think about its *complement*. Google "complement graph" for a definition.)

   Claim CLIQUE(G, k) has the same answer as INDEPENDENT-SET$(\overline{G}, k)$

   The mapping is in poly-time. It is well established that the complement of a graph can be taken in $O(V + E)$.

   proof of correctness -

   Assume we have a clique of size k. This means that we have k nodes which are mutually connected. When we take the compliment of the graph we remove every edge between these connected nodes meaning that they are now disconnected, so we have INDEPENDENT-SET of size k. The reverse argument follows in the exact same way.

   (b) [6 marks] Prove that INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

   Claim INDEPENDENT-SET(G, k) has the same answer as VERTEX-COVER$(G, n-k)$ where n is the number of nodes.

   If there is a set $S \subset V$ s.t. $\forall u, v \in S (u, v) \notin E$ then it must be that $\forall (u, v) \in E$ either $u \in S$ and $v \notin S$ $u, v \notin S$. This means that a vertex cover is $C = V - S$.

   The other direction requires a minimum vertex cover to ensure that no two members are connected to each other.

4. [8 marks] Let $CNF_k = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable CNF-formula where each variable appears in at most } k \text{ places}\}$. Show that CNF_3 is NP-complete. (Hint: Suppose a variable x appears (say) twice. Replace the first and second occurrences of x with new distinct variables y_1 and y_2, respectively. Next, add clauses to ensure that $y_1 = y_2$.

To do this in CNF form, recall that $y_1 = y_2$ is logically equivalent to $(y_1 \implies y_2) \wedge (y_2 \implies y_1)$, and that $(a \implies b)$ can be rewritten as $(a \vee \bar{b}) \wedge (\bar{b} \vee a)$. How can this trick be applied more generally when x appears $m > 2$ times?)
Claim: SAT $\leq_p$ CNF$_k$
Examine ϕ if a particular variable appears more than 3 times use the above mentioned trick repeatedly until each variable appears no more than 3 times. Since the above uses only CNF form we will still be in CNF form and each variable will appear less than 3 times.

5. [8 marks] It is possible for a problem to be NP-hard without actually being in NP itself. For example, the halting problem from class is clearly not in NP, since it is undecidable. However, in this question, you will show that the language $HALT = \{\langle M, x\rangle \mid M \text{ is a TM which halts on input } x\}$ is NP-hard. Is the halting problem hence NP-complete?

   Assume a decider for HALT
   Construct machine M which will attempt to decide 3SAT by enumeration. The machine is designed to halt only if it finds a valid assignment and loop otherwise. Feed this machine and ϕ into our HALT decider.

6. [8 marks] In class, we have focused on *decision* problems, i.e. deciding whether $x \in L$ or not. For example, given a 3-CNF formula ϕ, recall that the decision problem 3SAT asks whether ϕ is satisfiable or not. In practice, however, we may not just want a YES or NO answer, but also an actual assignment which satisfies ϕ whenever ϕ is satisfiable. It turns out that in certain settings, being able to solve the *decision* version of the problem (e.g. 3SAT) in polynomial time implies we can also solve the *search* version (e.g. find the satisfying assignment itself) in polynomial time. Problems satisfying this property are called *self-reducible*.

   Your task is as follows: Show that 3SAT is self-reducible. In other words, given any 3-CNF formula ϕ and a polynomial-time black-box M for determining if ϕ is satisfiable, show how to find a satisfying assignment for ϕ in polynomial time. You may assume that M is able to decide all CNF formulae with at most 3 variables per clause.

Claim 3SAT is self-reducible. Here is a polytime algorithm which can be used to find a satisfying assignment given M from above.
Let $x = \{x_1, x_2, ...x_n\}$ be the set of literals for ϕ

for $i1 \to n$ set $x_i = T$ run M on input ϕ with the assignments included if M rejects change x_i to False and run again if M rejects again return False i++ return x
We have assumed M runs in polytime. Worst case this alorithm will run M $2n$ times.