# CSE 3101 Design and Analysis of Algorithms
# Practice Test for Unit 6
# Reductions and NP-Completeness

Jeff Edmonds

First learn the steps. Then try them on your own. If you get stuck only look at a little of the answer and then try to continue on your own.

1. What is a computational problem $P$? Give two examples.
2. What is an algorithm $A$? When does it solve problem $P$?
3. What is the time complexity of an algorithm $A$?
4. What is the time complexity of a computational problem $P$? What are upper and lower bounds on this?
5. What is constant, linear, polynomial, and exponential time? Practically why does it matter? What are these times when $c = 2$ seconds and $n = 150$?
6. What is an *Optimization Problem* $P$?
7. Give examples of optimization problems studied in class that have polynomial time algorithms. Give one for each key type of algorithm covered. Give a real world application.
8. How do you make an optimization problem into a *decision problem*? Give an example. What is a *witness*?
9. What does it mean for a decision problem $P$ to be in *NP (Non-Deterministic Polynomial Time)*? (This was defined by Jeff's dad Jack Edmonds. (Do this without defining or referring to Non-Deterministic Turing Machines).
10. Your boss gives you an instance $I$ of an NP problem that by good luck happens to be a *Yes* instance. You are blessed to have a powerful fairy god mother to help you. How do you convince your boss that the answer for his instance is *Yes* without him knowing or being affected by your fairy god mother? Similarly, how would you convince your boss that the answer for his instance is *No*?
11. For an NP problem, is there a limit on the size of a solution for $I$ and/or on the number of possible such solutions? Why?
12. What is the *brute force* algorithm for $P$ and how long does it take?
13. What is the famous problem $N = NP$?
14. What does it mean for a decision problem $P$ to be *NP-Hard* or *NP-Complete*? What does this say about the time complexity of such a problem?
15. How is this useful to you in the real world.
16. Which problem did Cook at U. of Toronto. prove was NP-Complete? Give some other examples.
17. In practice, what is done to know whether the answer for the instance is yes or no.
18. Sketch the Venn-diagram of $P$, $NP$, $NP - complete$, $Co - NP$, $NP - Complete$, $Exp$.
19. Recall in 2001 learning that problems are *computable/decidable* if there is a TM that stops on every instance $I$ with the correct answer. What was the definition of a problem being *acceptable/recognizable*? How is this similar to the definition of NP. How is it different? How big can the solution be? Explain how the *Halting Problem* fits this definition.

20. How do you prove that one computational problem is at least as hard as another? $P_1 \leq P_2$. What is an *oracle*? Note that this is used in the definition of NP-Hard.

21. Outline the basic code for $Alg_{alg}$ solving $P_{alg}$ using the supposed algorithm $Alg_{oracle}$ supposedly solving $P_{oracle}$ as a subroutine. If $Alg_{oracle}$ is kind and also provides a valid solution $S_{oracle}$ for its instance $I_{oracle}$, then you should provide a valid solution $S_{alg}$ for your instance $I_{alg}$.

22. What steps do you have to take to prove that this reduction is correct?

23. Give two purposes of reductions.

24. Name two reductions done this term (using these two purposes).