Data Structures and Algorithms II
Fall 2018
Final Exam

Name:

Email:

Write all of your answers directly within this test booklet. Use the backs of pages if necessary. This is an open-book, open-notes test. You should not use any electronic devices. You may use whatever scrap paper you like, but do not hand it in.

Please also write your name on the top of each page.

(1) Consider the following mathematical function that maps two parameters to an integer in the range of 0 to 999:

$$f(x,y)=\begin{cases} 0 & x<0 \vee y<0 \\ (y*3+7)\%1000 & x=0 \wedge y\geq 0 \\ [f(x-1,y)*4+29]\%1000 & x>0 \wedge y=0 \\ [f(x-1,y)*f(x,y-1)+562]\%1000 & \textit{otherwise} \end{cases}$$

Below, write *pseudo-code* for an *efficient, recursive* function that computes value f(x,y).

Also, *state any assumptions* that must be true before the function is called.

Also, *state the complexity* of the function, in terms of x and y, using big-Theta notation.

ANNOUNCED DURING TEST: The parameters are guaranteed to be integers; they don't have a specified maximum size, but necessary data structures will fit in memory; the '%' symbol is the modulus operator, which computes the remainder when the first operand is divided by the second operand.

(2) Consider the following three NP-complete problems:

- The *Independent Set Problem*: Given an undirected graph G = (V, E) and a number k, determine if there is any set of vertices, I, with at least k vertices, such that I is an independent set (meaning that no edge connects two vertices within this subset of vertices).
- The *Clique Problem*: Given an undirected graph G and a number k, is there any set of at least k vertices that form a clique (meaning that all edges between these vertices exist).
- The *Node Cover Problem*: Given an undirected graph G and a number k, is there a set of at most k vertices that form a node cover (meaning that at least one endpoint of every edge in the graph touches one of these vertices).

Over the years, I have asked several variations of questions about these three problems on final exams; I think this one is new.

Assume that you know that the Clique Problem is NP-complete, but you do not know that the other two problems are NP-complete. Furthermore, assume you are not aware of any other NP-complete problem (i.e., at the outset of your attempt to answer this question, the Clique Problem is the only problem that you know to be NP-complete).

Given that the Clique Problem is NP-complete, prove that the Node Cover Problem is NP-complete.

HINT: The proof involves a simple reduction. If you can't figure either it out, state what you are trying to do for partial credit. Also remember that to prove that a problem is NP-complete, you need to show two things.

(Additional space for question 2…)

(3) Consider the following puzzle involving toy blocks. Every block is one unit long and one unit wide, but the heights of the blocks vary. There are four types of blocks. Some blocks are one unit high, some blocks are x units high, some blocks are y units high, and some blocks are z units high, where x, y, and z are distinct positive integers, and $1 < x < y < z$. You have unlimited supplies of each type of block.

Given a positive integer, N, *your goal is to build a tower of exactly size N using as few blocks as possible*. (Since you have an unlimited number of blocks of height 1, you know that it must be possible to produce the tower.)

(a) Consider the following greedy algorithm. Use as many z blocks as you can (without going over N). Then use as many y blocks as you can. Then use as many x blocks as you can. Then build in the remainder of the tower with blocks of height 1. Is this algorithm guaranteed to give you the optimal answer (i.e., the answer using the smallest possible number of blocks)? If so, explain why. If not, show a counter-example (i.e., an instance of x, y, z, and N for which this algorithm would not give you the optimal answer).

(b) *Show pseudo-code for a bottom-up dynamic programming solution to the problem*. The function should take x, y, z, and N as parameters, and return the smallest number of blocks necessary to create a tower of height N. Hint: Use a one-dimensional array call NB (for number of blocks), such that NB[k] = the minimum number of blocks necessary to build a tower of height k. Also specify the running time of the algorithm, in terms of N, using big-Theta notation.

(4) Consider a generalization of the block problem from (3). We will name this problem GBP. This time, you are given a finite (but possibly very large) set of blocks. There are B total blocks. Each block has height h_i, where $1 \leq i \leq B$, and each h_i is a positive integer. Note that you do NOT necessarily have any blocks of height 1, and you do NOT have an unlimited supply of blocks. Your goal is to determine if it is possible to build a tower of height N.

This is an NP-complete problem! Answer the following questions about it:

(a) Is there any non-deterministic Turing machine that could solve this problem in worst-case polynomial time? Explain your answer.

(b) Can any regular Turing machine solve this problem? Explain your answer.

(c) Recall the Boolean Satisfiability Problem (SAT) discussed in class, which Cook proved is NP-complete. Is it necessarily the case that SAT can be reduced to GBP? Is it necessarily the case that GBP can be reduced to SAT? Explain your answers.

(d) Assume that P_1 is some problem in the class NP-hard and P_2 is some problem in the class P. Also *assume that* $P \neq NP$. Answer each of the following questions (keeping in mind that *if something is necessary, it is also possible*):

I. Is it possible that P_1 can be reduced to GBP?

II. Is it necessarily the case that P_1 can be reduced to GBP?

III. Is it possible that GBP can be reduced to P_1?

IV. Is it necessarily the case that GBP can be reduced to P_1?

V. Is it possible that P_2 can be reduced to GBP?

VI. Is it necessarily the case that P_2 can be reduced to GBP?

VII. Is it possible that GBP can be reduced to P_2?

VIII. Is it necessarily the case that GBP can be reduced to P_2?

(5) Briefly answer the following questions about algorithm strategies:

(a) Suppose a program that frequently uses a pseudo-random number generator were to re-seed the generator with the time of day before every call to the generator. Why would this be a bad idea?

(b) Consider a magical oracle that asks you to guess the N-bit positive integer that it is thinking of. After each guess, the oracle will tell you if you are right or wrong, but that is all the information you will get. You have no choice, therefore, but to use exhaustive search to try all possible guesses until you get the right answer. In the worst case, how many guesses will it take you?

(c) Consider two text files, T_1 and T_2, which each contain D distinct characters and N total characters. In T_1, the frequencies of the different characters are approximately the same (i.e., each character appears approximately the same number of times). In T_2, the frequencies of the different characters vary wildly. Will one file benefit more than the other from Huffman coding? Is so, which?

(d) Why is the solution to the Tower of Hanoi puzzle, discussed in class, considered a divide-and-conquer approach?

(e) Consider the solution to the submarine/number-line puzzle, as discussed in class. If applied when the submarine starts at position s and travels at velocity v, what is the complexity of the number of guesses it would take to hit the submarine, expressed using big-Theta notation?