NAME (please PRINT in large letters):

SECTION: 01 02 (circle one)

# CMSC 27200 Theory of Algorithms

Last exam — 03-15-2015

The exam is **closed book.** Do not use notes.
The use of ELECTRONIC DEVICES is strictly forbidden.

Use the paper provided. Do not use your own paper. You may continue on the reverse side of each sheet.

**SHOW ALL YOUR WORK.** Correct conclusions that come out of the blue (details of calculation missing) will not receive credit.

Please be accurate and concise. When writing **pseudocode,** keep with simple instructions and notation, do not use notation specific to some programming languages. **Define your variables** and comment on each line, what is happening.
You do NOT need to prove **correctness** of your algorithms unless specifically asked.

Warning: The **extra-credit (XC)** problems are underrated; do the ordinary problems before you attempt the extra-credit problems.

This exam contributes 40% to your course grade.
The total point value of the ordinary (non-extra-credit) problems is 400 points. The extra-credit problems, as their name suggests, add to your score.

1. (14+12 points) (Growth rates)

   (a) (Superpolynomial growth) Let $f(n)$ be a function on the positive integers. We say that $f(n)$ grows *superpolynomially* if for all $C > 0$ we have $f(n) \geq n^C$ for all sufficiently large $n$. (The threshold for "suffiently large" depends on $C$.) Prove that if $f(n)$ grows superpolynomially then for all $C > 0$ we have $n^C = o(f(n))$. (Recall the little-oh notation: $a_n = o(b_n)$ if $\lim_{n\to\infty} a_n/b_n = 0$.)

   (b) (Intermediate growth) Find a function $f(n)$ on the positive integers such that (i) for every $C > 0$ we have $n^C = o(f(n))$ and (ii) for every $c > 0$ we have $f(n) = o(\exp(n^c))$. Here $\exp(x)$ denotes $e^x$. Prove that your function satisfies both conditions.

2. (Recurrent inequalities) Consider the following model of multiplication of matrices with real entries: arithmetic operations (multiplication, addition, subtraction) of real numbers has unit cost.

   Strassen's matrix multiplication algorithm reduces the problem of multiplying two $n \times n$ matrices to 7 instances of the half-size problem (multiplying 7 pairs of $n/2 \times n/2$ matrices). The reduction is achieved by performing a constant number of additions and subtractions of matrices of size $n \times n$ and $n/2 \times n/2$ at a total cost of $O(n^2)$.

   Let $T(n)$ be the cost of multiplying two $n \times n$ matrices by Strassen's method. Observe that

$$T(n) \leq 7T(n/2) + O(n^2). \tag{1}$$

   (a) (12 points) Evaluate this recurrence. Show that $T(n) = O(n^\beta)$ for some positive real number $\beta$. Determine the smallest value of $\beta$ inferable from Eq. (1). Use the method of reverse inequalities. Do not use the "Master Theorem." You may assume that $n$ is a power of 2.

   (b) (XC 8 points) Prove that your $\beta$ is the smallest possible exponent inferable from (1).

   (c) (6 points) (Fibonacci recurrence) Let $R(n) > 0$ be a function on positive integers. Assume $R(n) \leq R(n-1) + R(n-2) + O(n^2)$. Prove: $R(n) = O(F_n)$ where $F_n$ is the $n$-th Fibonacci number ($F_0 = 0$, $F_1 = 1$, $F(n) = F(n-1) + F(n-2)$).

3. (12+8+10 points) (Generating random primes) Alice needs to generate an $n$-bit random prime number. (Initial zeros are permitted.) She performs the following experiment: she flips $n$ coins; this produces a random $n$-bit integer $X$. She checks $X$ if is prime ("success"). She repeats the experiment until she succeeds.

   (a) What is the expected number of experiments she needs to perform? Give an exact expression in terms of the function $\pi(x)$, the number of primes $\leq x$. Prove your answer.

   (b) Asymptotically evaluate this expected value. State and use the Prime Number Theorem.

   (c) Is the expected number of trials polynomially bounded as a function of $n$, i. e., is it $O(n^c)$ for some constant $c$? Prove you answer (and determine $c$ if it exists).

4. (Independence number) Let $G = (V, E)$ be an undirected graph with $n$ vertices. A subset $S \subseteq V$ is *independent* if no pair of vertices in $S$ is adjacent. In other words, $S$ is independent in $G$ exactly if $S$ is a clique (complete subgraph) in the complement of $G$. The size of the largest independent set is denoted $\alpha(G)$; this is called the *independence number* of $G$. So for instance $\alpha(K_n) = 1$ and $\alpha(C_n) = \lfloor n/2 \rfloor$ where $C_n$ denotes the cycle of length $n$.

   (a) (6 points) Finding the value of $\alpha(G)$ is an optimization problem. State the corresponding decision problem (input, question). Let IND denote this decision problem.

   (b) (8 points) Prove that IND is NP-complete, using the NP-completeness of CLIQUE, the decision version of the maximum clique problem.

   (c) (20 points) Prove that $\alpha(G)$ can be computed in time $O(F_n)$ where $F_n$ is the $n$-th Fibonacci number. Hint: build the independent set $S$ recursively. Pick a vertex $v \in V$; consider the two cases: $v \in S$ and $v \notin S$.

   (d) (6 points) Suppose every vertex in $G$ has degree $\leq 2$. Find $\alpha(G)$ in linear time. Describe your algorithm in unambiguous English, no pseudocode needed. Give sufficient detail to justify the linear time claim.

   (e) (XC 15 points) Prove that $\alpha(G)$ can be computed in time $O(\sigma^n)$ where $\sigma \approx 1.381$ is the positive root of the equation $t^4 = t^3 + 1$. (Comment: recall that $F_n = \Theta(\gamma^n)$ where $\gamma \approx 1.618$ so this is an improvement over part (c).)

5. (16 points) (Local minimum) Let $A[1 \ldots n]$ be an array of real numbers. We say that position $i$ is a *local minimum* if $A[i]$ is not greater than either of its neighbors, i. e., $A[i] \leq A[i+1]$ and $A[i] \leq A[i-1]$ for $2 \leq i \leq n-1$ and $A[1] \leq A[2]$ if $i = 1$ and $A[n] \leq A[n-1]$ if $i = n$. Find a local minimum in $O(\lg n)$ queries to the array. State the best constant hidden in the big-Oh notation that you can get. Describe your algorithm in pseudocode.

6. (20 points) (Dijkstra with a double twist) We are given a weighted digraph with a source vertex, $G = (V, E, s, w)$ where $s \in V$ is the source and $w : E \to \mathbb{R}$ is the weight function. All weights are non-negative. We are also given a partition of the edges into "red," "green," "blue" edges: $E = R \cup G \cup B$ where $R \cap G = R \cap B = G \cap B = \emptyset$. For all vertices $v \in V$ find the minimum cost of reaching $v$ from $s$ along a path that uses at most one red edge and at most one green edge. Your algorithm should run in "Dijkstra time." – Instruction: do not invent a new algorithm; apply an algorithm studied in class to inputs other than $G$. Your job is to construct the new inputs and to state how to use them. Describe your construction and your algorithm in unambiguous English (with mathematical formulas). No pseudocode required.

7. (Linear programming feasibility) Recall: The *linear programming (LP) feasibility* problem takes as input a list $L$ of linear inequalities of the form $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$. (This is the $i$-th inequality.) The $a_{ij}$ and $b_j$ are the coefficients of $L$. We say that $L$ is *feasible* if there exist real numbers $x_1, \ldots, x_n$ that satisfy all the inequalities in $L$.

   The *integer LP (ILP) feasibility* problem asks whether or not there is a solution to this system in which the variables take integer values.

   The $(0, 1)$-*LP feasibility* problem asks whether or not there is a solution to this system in which the variables take values $0, 1$ only.

   (a) (10+10 points) Prove that the following LP is (a1) feasible but (a2) not ILP-feasible (has no solution in integers):

$$\begin{aligned} x_1 + 5x_2 + 2x_3 &\leq 4 \\ 2x_1 - 2x_2 + x_3 &\leq 1 \\ -2x_1 - 2x_2 - 2x_3 &\leq -3 \end{aligned}$$

   (b) (18 points) Karp-reduce 3-colorability of graphs to $(0, 1)$-LP feasibility. Explanation: Given a graph $G$, you need to construct in polynomial time an LP $L(G)$ with integer coefficients such that $G$ is 3-colorable if and only if $L(G)$ is $(0, 1)$-feasible.

   (c) (5 points) Define the CLIQUE problem (the decision version of the maximum clique problem). Clearly state the input and the question asked.

   (d) (XC 20 points) Karp-reduce CLIQUE to $(0, 1)$-LP feasibility. State the number of variables in your LP as a function of the parameters of the input graph. Clearly state what you need to prove about the LP you construct.

8. (Complexity classes) Let NPC denote the class of NP-complete problems. Prove:

   (a) (15 points) If $\mathsf{NPC} \cap \mathsf{P} \neq \emptyset$ then $\mathsf{P} = \mathsf{NP}$.

   (b) (XC, 10 points) If $\mathsf{NPC} \cap \mathsf{coNP} \neq \emptyset$ then $\mathsf{NP} = \mathsf{coNP}$.

   (c) (XC, 15 points) If an NP-complete problem is Cook-reducible to a problem in $\mathsf{NP} \cap \mathsf{coNP}$ then $\mathsf{NP} = \mathsf{coNP}$.

9. (8+6+15+5+15 points) (Jarník vs. Dijkstra) Recall that Jarník's (a.k.a. Prim's) algorithm solves the min-cost spanning tree problem.

   (a) State exactly the input that Jarník's algorithm takes; compare it item by item with the input taken by Dijkstra's algorithm. Pay special attention to whether the graph is directed or undirected, whether the weights need to be non-negative; whether the input specifies a source node; connectivity issues.

   (b) Describe the output of (b1) Dijkstra's (b2) Jarník's algorithm

   (c) Describe Jarník's algorithm in pseudocode. Define your variables. Indicate where the code differs from Dijkstra's. (Use the reverse side of this sheet.) Even though not absolutely necessary, do use the white/grey/black status indicators.

   (d) State the three data structure operations used, and how many times each was used (upper bound) in each algorithm

   (e) Prove that Jarník's algorithm cannot be implemented to run in linear time. ("Implementation" refers to a specific realization of the data structure operations.) Specifically, show that any implementation will require $\Omega(n \log n)$ comparisons on certain connected weighted graphs with $n$ vertices and $O(n)$ edges.

10. (16+8+8+8 points) (Loop invariants)

(a) Give an exact definition of a loop invariant. Include the definition of the configuration space. Make sure to state the domain of your predicates and the domain and range of your functions. Make it explicit to what loop you are referring and what role your predicates and functions play in the loop.

(b) The body of Jarník's algorithm is a single **while** loop. For each statement below, decide whether or not it is a loop invariant. Prove your answers. (The vertices are denoted $v_1, \ldots, v_n$; the weight function is $w$, and $c$ denotes the "current cost" function.)

(b1) $c(v_7) = \min\{w(u, v_7) \mid u \text{ is a neighbor of } v_7\}$.

(b2) $c(v_7) = \min\{w(u, v_7) \mid u \text{ is a black neighbor of } v_7\}$.

(b3) The quantity $c(v_7)$ cannot increase.

11. (5 + 8 points)

    (a) Spell out the acronym "NP."

    (b) Prove: every problem in NP can be solved in time $\exp(n^c)$ for some costant $c$. ($c$ depends on the problem.)

12. (20 points) (Sorting after preprocessing) Alice needs to sort $n$ data by comparisons. Bob, her assistant, helpfully arranges the data in a heap. "I checked that the heap rule is not violated," he assures her. Prove that Alice still needs to make $\gtrsim n \lg n$ comparisons (in the worst case). (As always in this course, "lg" refers to base-2 logarithms.) – Warning. This is NOT a problem about Heapsort. While the data are arranged in a heap, Alice is free to compare any pair of data she chooses. The role of the heap is that of a preprocessing: some pairs of data have already been compared, she does not need to compare them. – Note also that you are required to prove more than an $\Omega(n \lg n)$ lower bound: you need to show that asymptotically at least $n \lg n$ comparisons are needed (i. e., more than $(1 - \epsilon)n \lg n$ for every $\epsilon > 0$ and all sufficiently large $n$). You will get partial credit for proving an $\Omega(n \lg n)$ lower bound.)

13. (12+12 points) (DFS and topological sort)

    (a) Describe DFS in pseudocode.

    (b) Given a digraph, either find a directed cycle or topologically sort the vertices. Your algorithm must run in linear time. Use DFS. Explain your algorithm in unambiguous English, no pseudocode required. No proof required.

14. (20 points) (Edit distance) We transform a word (string of characters) into another word using the following "edit" operations: delete, insert, replace. For instance, here is how to turn "NAIVE" into "FANATIC:"

    NAIVE - NAIVC - NAIC - NATIC - FNATIC - FANATIC.

    The sequence of operations was REP,DEL,INS,INS,INS. The *edit-distance* of two words is the minimum number of edit operations needed to turn one word into the other. (If the above sequence of operations is optimal, then the edit-distance of NAIVE and FANATIC is 5.)

    Describe an algorithm which finds the edit-distance of two given words in $O(km)$ steps where $k$ and $m$ are the respective lengths of the two input words.

    Describe your algorithm in pseudocode. It should be very simple, no more than a few lines. Name the algorithmic technique used. **Define the meaning of your variables.** Half the credit goes for the clear definition (the "brain" of your algorithm).

[15.] (10+8+8 points) (Euclid's algorithm)

(a) Describe Euclid's algorithm in pseudocode as a **while** loop; do not use recurrence.

(b) State the loop invariant that proves correctness.

(c) Prove that the algorithm runs in polynomial time.