NAME (please PRINT in large letters):

SECTION: 01 02 (circle one)

CMSC 27200 Theory of Algorithms

First Midterm — 01-29-2015

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 in each line on what is happening.

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

This exam contributes 15% to your course grade.
The total point value of the ordinary (non-extra-credit) problems is 150 points. The extra-credit problems, as their name suggests, add to your score.

1. (a) (25 points; lose 5 points for each incorrectly sorted pair) **(Asymptotic growth)** List the 9 functions in the list below by their asymptotic nondecreasing growth order. First identify functions that have the same growth order (are in Θ relation with each other), and then, from each block of functions with the same growth order, put just one in your linear order. "lg" is to the base 2, "ln" is to the base e.

   Note: $\ln 2 \approx 0.69$.

$$n^{1/3},\ n^{1/2},\ n^{0.9},\ 5n,\ 2^{(\lg n)^{1/2}},\ 2^{\ln n},\ 2^{\lg n},\ n/\ln n,\ n/\lg n$$

   Use the "little-oh" notation: we say that $f(n) = o(g(n))$ ("$f(n)$ is little-oh of $g(n)$") if $\lim_{n\to\infty} f(n)/g(n) = 0$. For instance, $n^{3/2} = o(n^2)$ and $n! = o(n^n)$ but $n^2 \neq o(100n^2 + 10^6)$.

   So if a function $f(n)$ precedes a function $g(n)$ in your ordering then you must have $f(n) = o(g(n))$.

   Briefly justify the claimed relationship (Θ or little-oh) among consecutive functions on your list. The penalty for omitting a function from your list is 8 points. If you list a function twice, we consider only the first occurrence.

   (b) (6 XC points) Place the function $\dfrac{4^n}{\binom{2n}{n}}$ in your list above. Prove your answer.

2. (20 points) **(Recurrence)** Let $T(n)$ be a function on the positive integers such that

   (i) $T(n) \geq 0$ for all $n \geq 1$ and

   (ii) $T(n) \leq 2T(\lceil n/2 \rceil) + O(n)$

   Prove: $T(n) = O(n \lg n)$.

   Use the method of reverse inequalities. (Do not use the Master Theorem or any other theorem on recurrences not proved in class.) You may assume that n is always a power of 2.

3. (12+15 points)

   (a) **(Merge)** Given two sorted arrays $A[1 \dots k]$ and $B[1 \dots \ell]$ of real numbers, describe in **pseudocode** how to merge them.
   So the assumption is that $A[1] \leq A[2] \leq \cdots \leq A[k]$ and $B[1] \leq B[2] \leq \cdots \leq B[\ell]$ and you need to output an array $C[1] \leq C[2] \leq \cdots \leq C[k+\ell]$ that includes each entry of the arrays A and B. State the number of comparisons made by your procedure.

   (b) **(k-way merge)** Given k sorted arrays $A_1, \dots, A_k$ (of variable lengths) of a total of n items, use **heap** to merge them in time $O(n \log k)$. (So your output will be a sorted array of length n.) Describe your algorithm in unambiguous English, pseudocode is not required. Justify the time bound. (You get half the credit if you solve the problem without using a heap.)

4. (8+16+4+18 points) **(Dijkstra's algorithm)**

   (a) State the computational task solved by Dijkstra's algorithm. Be exact in specifying the input and the output.

   (b) Describe Dijkstra's algorithm in **pseudocode**. **Define your variables.**

   (c) Suppose we modify the "RELAX(u, v)" routine so that it will try to update the cost of v even if v is black. Will this change the outcome? Reason your answer. (Note added after the test: this problem was not intended to ask you to prove the correctness of Dijkstra's algorithm but rather to take that fact for granted and then answer the question of what happens if we modify the algorithm as above.)

   (d) Give an example of a weighted DAG (directed acyclic graph) with some negative edges where even this modified version of Dijkstra's algorithm fails to find the optimum costs. Make your example as small as you can (in terms of the number of edges). You do not need to prove that your example is the smallest. Describe the progress of your algorithm in a table, stating the status, current cost, and parent node of each node in each phase (each execution of the **while** loop).

5. (10 XC points) **(Uphill-downhill Dijkstra)** We say that a sequence of real numbers in *unimodal* if the sequence is nondecreasing until it reaches its maximum (ascent phase) and then it is nonincreasing (descent phase). For instance, these sequences of single-digit integers are unimodal: 1562, 5779998331, 34789, or 99933333. (Note that we allow sequences that are monotone; either the ascent or the descent can have length zero.) The sequences 656, 4325, 12121 are not unimodal.

   Consider a weighted digraph $G = (V, E, w)$ with non-negative edge weights $w(u, v) \geq 0$ and a source $s \in V$ with the following additional information: every node v has an *elevation* $e(v)$ (specified as part of the input). We say that a path is unimodal if the sequence of nodes along the path have unimodal elevations (first we go uphill and then downhill). Find the minimum cost of unimodal paths from a given source vertex s to a given target vertex t in "Dijkstra time," i. e., time $O((|V| + |E|) \log |V|)$ if we use the heap implementation of priority queues.

   Describe your algorithm in unambiguous English, no pseudocode required. Justify the timing.

6. (14 points) **(Switchover)** Let $A[1 \dots n]$ be a $(0,1)$-array (every entry is 0 or 1). Assume $A[1] = 0$ and $A[n] = 1$. Design an algorithm that finds a value i such that $A[i] = 0$ and $A[i+1] = 1$ $\quad (1 \le i \le n-1)$. You can access the array by making queries of the form $A[i] \overset{?}{=} 0$; the answer you receive to each query is "Yes" or "No." Each query carries unit cost; all other operations (bookkeeping, computation) are free. Your goal is to minimize the cost. Your algorithm should work with cost $\lg n + O(1)$. Describe your algorithm in clear pseudocode. Name the method used.

7. (18 points) **(Interval sum)** We are given an array $A[1 \dots n]$ of real numbers. The sum of the interval $[i,j]$ is the quantity $S[i,j] := \sum_{k=i}^{j} A[k]$. Find the maximum interval sum

$$S_{\max} = \max_{1 \le i,j \le n} S[i,j].$$

Find this value in *linear* time (i. e., the number of operations should be $O(n)$). Describe your solution in elegant and simple **pseudocode**. (Note: you are not required to output the interval with the maximum sum, just the value of the maximum sum.) Observe the following convention:

Convention: If $j < i$, we say that the interval $[i,j]$ is *empty*; the sum of the empty interval is zero. Empty intervals are admitted in the problem. Therefore $S_{\max} \ge 0$ even if all the $A[i]$ are negative.

Instructions. Use dynamic programming. Bear in mind that in dynamic programming exercises, half the credit goes for the clear and simple definition of the array of problems you use (the "brain" of the solution). Do not confuse the "brain" of the solution with the "heart" of the solution (the recurrence). **Explain the meaning of your variables!** *Elegance* and *simplicity* count. Do not use notation specific to some programming language, just the generic pseudocode instructions appearing in handouts.

8. (10 XC points) Consider the following game against the bank. The bank stores a number V (your "vitality" in the game); initially, V is set to $V = n$. Here is the process:

```
**while** $V > 0$
        roll die
        **if** die $< 6$, win $V$ dollars from the bank
        **else** (die $= 6$)   $V := \lfloor V/2 \rfloor$
**end(while)**
```

What is the fair value of entering this game, i. e., what is your expected win? Your answer should be a very simple formula in terms of n. Prove your answer. You may assume n is a power of 2.