COMPSCI 320SC 2008 Midterm Test

Attempt *all* questions. (Use of calculators is NOT permitted.)

Put the answers in the space below the questions. Write clearly and *show all your work*!

Marks for each question are shown below and just before each answer area.

This 50 minute test is worth 10% of your final grade for the course.

Question #:	1	2	3	4	Total
Possible marks:	10	10	10	20	50
Awarded marks:					

University ID: _____

Student Name: _____

Student Signature: _____

Time Finished: _____

1. Consider the following divide-and-conquer recurrence

$$t(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ t(\lfloor n/3 \rfloor) + t(\lfloor n/2 \rfloor) + 3n & \text{if } n > 1 \end{cases}$$

(a) What are the values of $t(2), t(4), t(7)$? (***5 marks***)

(b) Solve the recurrence asymptotically (upper bound) for general n.

You may want to make use of the following 'master recurrence theorem':

Assume $t(n) = a \cdot t(n/b) + \Theta(n^k)$ is the total time for a divide-and-conquer algorithm then:

$$t(n) \in \begin{cases} \Theta(n^k) & \text{if } a < b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

(***5 marks***)

2. (a) Describe and analyze the running time of a good minimum spanning tree algorithm that takes as input an edge-weighted graph $G = (V, E)$ that is represented as adjacency lists. The adjacency list for vertex u consists of a list of pairs (v, w), where v is a neighbor of u and w is a 32-bit integer edge weight.[1] Use variables $n = |V|$ and $m = |E|$ and explain any costs for other data structures you require. (***6 marks***)

(b) Consider the same algorithm for part (b), but with your algorithm operating on an edge-weighted graph that is represented as an adjacency matrix of 32-bit integer edge weights. Explain why the running time may be different (either better or worse). (***4 marks***)

[1] We assume the weight w is the same for pair (v, w) and pair (u, w) in the lists of u and v, respectively.

3. Let $M_1, M_2, \ldots, M_n$ be songs to be burned on a read-only disk. Song M_i requires K_i kilobytes of storage, and the capacity of the disk is D kilobytes, where $D < \Sigma_{i=1}^{n} K_i$.

   (a) Briefly describe a correct algorithm (not necessarily efficient) that would minimize the leftover space on the disk, assuming we can not include part of a song. (***5 marks***)

   (b) Complete the following proof that the greedy algorithm that selects songs in order of nondecreasing K_i will always maximize the number of songs that can be stored on the disk. (***5 marks***)

   Let $G = (g_1, g_2, \ldots, g_i, \ldots, g_r)$ be a selection of songs ($1 \leq g_j \leq n$) picked in order by the greedy algorithm and let $O = (o_1, o_2, \ldots, o_i, \ldots, o_s)$ be a better solution, $s > r$, that agrees with the greedy algorithm up till position i where i is as large as possible. [Note: we are using indices (j's) of the songs M_j to represent the algorithms' selection.]

   We know that $K_{g_i} < K_{o_i}$ because ...

   If $g_i = o_j$ for some $j > i$ then we know that $O' = (o_1, o_2, \ldots, o_{i-1}, o_j, o_{i+1}, \ldots, o_{j-1}, o_i, \ldots, o_s)$ is a better solution with a larger possible prefix in common with the greedy algorithm. The modified solution O' differs from the better solution O only in its ordering; and only the i-th and j-th elements have been changed. Formally, O' is related to O by the following equations: $o'_i = o_j$, $o'_j = o_i$, and $o'_k = o_k$ for all other k.

   Otherwise, if $g_i \neq o_j$ for some $j > i$ then we know that

   $O'' =$

   is a better solution with a larger possible prefix in common with the greedy algorithm.

   Thus we get a contradiction to the maximality of i. Hence the greedy algorithm must be an optimal solution. □

4. You are developing a simple version of the Pac-Man game. This game is played on a $n \times n$ grid. Each square in the grid is either empty, or else it has a cherry.

   The input is held in a 2-dimensional array C. If $c_{ij} = 1$, a cherry is in the square at row i and column j. If this square is empty, then $c_{ij} = 0$. Sample input is shown below.

   The Pac-Man starts in the upper-left corner of the grid, and can move only in two directions: right and down. It should be clear that, after exactly $2(n-1)$ moves, the Pac-Man will be in the bottom-right corner.

   The goal of the game is to have the Pac-Man eat the maximum number of cherries when making $2(n-1)$ moves. Pac-Man eats a cherry whenever it is on the same square as a cherry.

$$C = \begin{array}{|c|c|c|c|} \hline 1 & 0 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 \\ \hline \end{array}$$

   (a) Let V be an $n \times n$ array of integers. The value of v_{ij} is the maximum number of cherries that Pac-Man can eat by the time it reaches square (i, j). It should be clear that $v_{11} = c_{11}$ and $v_{12} = c_{11} + c_{12}$.

   Compute V for the sample input C, writing your answers in the space below. (***4 marks***)

$$V = \begin{array}{|c|c|c|c|} \hline 1 & 1 & & \\ \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}$$

   (b) Write a recurrence relation for v_{ij}. (***6 marks***)

(c) Write a program, in C++ or Java or pseudocode, which efficiently computes V. (***4 marks***)

(d) Let X be an $n \times n$ array of integers. The value of x_{ij} is the number of different paths on which Pac-Man can reach square (i, j) while eating a maximal number of cherries.

Develop a recurrence relation for x_{ij}. Use your recurrence to compute X for the sample input C, writing your answers into the following array.

$X =$

1	1		
1	2		

Write your recurrence relation in the space below. (***3 marks***)

(e) You are now starting to work on Pac-Man2. Pac-Man2 can move up, down, and right. However Pac-Man2 cannot move to the left.

Pac-Man2 starts in the upper-left corner of an $n \times n$ board and has the goal of eating all the cherries in a minimum number of moves. Note that Pac-Man2 can clear the board in fewer than $n(2n-1)$ moves, because an optimal Pac-Man2 can clear a column and move to any desired row in that column with at most $2n-2$ up/down moves; there are exactly n columns; and Pac-Man2 must take exactly $n-1$ rightward moves in order to reach the n-th column.

You have written a subroutine $f(i, j, k, C)$ which computes the minimum number of moves required for Pac-Man2 to eat all cherries in column j of array C, starting from square (i, j) and ending in square (k, j).

Write a recurrence relation for the Pac-Man2 problem. Hint: let w_{ij} be the minimum number of moves to collect all the cherries in $C[1..n, 1..j]$, starting from $(1, 1)$ and ending at (i, j).

(***3 marks***)