CPSC 411: Design and Analysis of Algorithms
Exam 1
October 16, 2008

Name: Key

Instructions:

1. This is a closed book exam. Do not use any notes or books, other than your 8.5-by-11 inch review sheet. Do not confer with any other person. Do not use any computer equipment.
2. Show your work. Partial credit will be given. Grading will be based on correctness, clarity and neatness.
3. I suggest that you read the whole exam before beginning to work any problem. Budget your time wisely—according to the point distribution.
4. There are 4 questions worth a total of 100 points, on 8 pages (including this page).

DO NOT BEGIN THE EXAM UNTIL INSTRUCTED TO DO SO. GOOD LUCK!

Please sign the academic integrity statement:

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have not received or given any assistance that is contrary to the letter or the spirit of the guidelines for this exam."

Signature: _____

1. (12 pts total) Solving Recurrences. Give tight asymptotic bounds for the following recurrences. Justify your answers by working out the details or by appealing to a case of the master theorem.

(a) (4 pts) $T(n) = 9T(n/4) + n^2$

$a = 9$

$b = 4$

$f(n) = n^2$

Compare $f(n) = n^2$ to $n^{\log_b a} = n^{\log_4 9} = n^{1.\ldots}$

Since $f(n) = n^2 = \Omega(n^{\log_4 9 + \varepsilon})$ we might be in Case 3

Check if $a \cdot f(\frac{n}{b}) \le c \cdot f(n)$ for some $c < 1$ and sufficiently large n,

i.e., $9 \cdot f(\frac{n}{4}) \le c \cdot f(n)$, or $9(\frac{n}{4})^2 \le c \cdot n^2$, or $\frac{9}{16} \cdot n^2 \le c \cdot n^2$.

Pick $c = 9/16$. So $T(n) = \Theta(f(n)) = \Theta(n^2)$.

(b) (4 pts) $T(n) = 3T(n/2) + n$

$a = 3$

$b = 2$

$f(n) = n$

Compare $f(n) = n$ to $n^{\log_b a} = n^{\log_2 3} = n^{1.\ldots}$

Since $n = O(n^{1.\ldots - \varepsilon})$, we're in Case 1.

So $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$.

(c) (4 pts) $T(n) = 3T(n/3) + \log n$

$a = 3$

$b = 3$

$f(n) = \log n$

Compare $f(n) = \log n$ to $n^{\log_b a} = n^{\log_3 3} = n$.

Since $\log n = O(n^{1-\varepsilon})$, we're in Case 1.

So $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$.

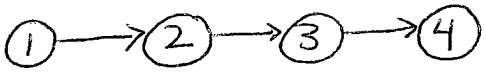
2. (24 pts total) Greedy Algorithms, Shortest Paths, and Matrix Multiplication.

Given a directed graph, the *transitive closure* problem is to determine, for all pairs of nodes u and v, if there exists a path from u to v.

Consider the following idea for how to solve the transitive closure problem.

1. Use the adjacency matrix representation for the graph, where $A[i, j]$ equals 1 if $i = j$ or if there is an edge from node i to node j, and 0 otherwise.
2. Note that $A^2 = A \cdot A$ (multiplying the matrix A times itself) gives a matrix in which the (i, j) entry is positive if there is a path of length 2 or less from i to j, and is 0 otherwise.
3. Similarly, note that A^{n-1} (the product of A with itself $n - 1$ times) is a matrix such that if entry (i, j) is positive, then there is a path of length $n - 1$ from i to j, otherwise there is not one. Since $n - 1$ is the maximum length of any loop-free path, we have now computed the transitive closure.

(a) (6 pts) Compute A^1, A^2, and A^3 for the given graph.



$$A^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 0 & 1 & 3 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(b) (6 pts) Describe an algorithm to calculate A^{n-1}.

```
A^1 := A
for i := 2 to n do
    A^i := A^(i-1) · A
end for
```

(c) **(6 pts)** What is the running time of your algorithm from part (b)? Assume the naive algorithm for matrix multiplication.

$O(n)$ iterations, each takes $O(n^3)$ time.

Total is $O(n^4)$

(d) **(6 pts)** Redo part (c) assuming Strassen's algorithm is used for matrix multiplication.

$O(n)$ iterations, each takes $O(n^{\log_2 7})$

Total is $O(n^{1+\log_2 7})$

3. (27 pts total) Dynamic Programming.

(a) (3 pts) What is the key feature of a problem that makes dynamic programming a good candidate for finding a solution?

overlapping subproblems

(b) (3 pts) List the three steps for solving a problem using dynamic programming.

1. construct recursive formulation of the solution
2. determine dependencies among subproblems
3. choose total order for solving the subproblems that respects the dependencies.

Recall that in the 0-1 knapsack problem, a thief robbing a store finds n items. The i-th item is worth v_i dollars and weighs w_i pounds, where v_i and w_i are integers. The thief wants to maximize the value of what he steals, but he cannot carry more than W pounds in his knapsack. *He must take all of an item or none of it.*

Describe a dynamic programming algorithm for determining the maximum value that the thief can steal. *Hint:* Consider any fixed order of the items. Let $C[i, p]$ be the maximum total value of any subset of the first i items whose total weight is at most p. To compute $C[i, p]$, the thief has two choices when considering item i: either he takes it or he leaves it behind.

- If he takes the i-th item, then he gets the value of the item (v_i), but it also takes up some of the capacity of the knapsack so he only has $p - w_i$ room in the knapsack in which some subset of the previous items (1 through $i - 1$) might fit.
- If he doesn't take the i-th item, then he doesn't get the value of the item, but he also doesn't use up any of the capacity of the knapsack on that item, so he can apply all of the p capacity to the previous items (1 through $i - 1$).

Take the maximum of these two choices.

(c) (4 pts) What is the mathematical definition of $C[i, p]$, based on the previous discussion?

$$C[i,p] = \max\{v_i + C[i-1, p-w_i], C[i-1,p]\}$$

(d) (2 pts) What are the basis cases for defining $C[i, p]$?

$C[0,p] = 0, \forall p$

$C[i,0] = 0, \forall i$

(e) (2 pts) Which element of $C[i, p]$ is the goal?

$C[n, W]$

(f) **(4 pts)** What are the dependencies among the subproblems?

C[i,p] depends on C[i-1, q] for all $q \leq p$
(previous row up to current column)

(g) **(4 pts)** What order for filling in the C table would respect the dependencies?

row major order

(g) **(5 pts)** What is the running time of your algorithm, and why?

$O(n \cdot W)$ since there are $O(n \cdot W)$ entries in C and each takes $O(1)$ time to compute.

4. **(27 pts total) Amortized Analysis**

(a) **(3 pts)** What is the purpose of amortized analysis?

To get an accurate upper bound on the worst-case running time of a sequence of operations on a data structure.

(b) **(3 pts)** What is the definition of the amortized cost of an operation using the potential method?

$\hat{c}_i \triangleq c_i + \Phi(D_i) - \Phi(D_{i-1})$

c_i: actual cost; $\Phi(D_i) - \Phi(D_{i-1})$: change in potential

Consider the implemention of the disjoint sets data structure that uses forests with path compression and weighted union. Let s be a sequence of operations in which first n Make-Sets are done, then $n-1$ Unions are done (creating a single tree), and then n Find-Sets are done, one for each element. Assume that the arguments to the Unions are roots (so no Find-Set is done inside a Union). Analyze the running time of the entire sequence using the potential method. Use the potential function $\Phi(D_i) =$ the number of nodes with a grandparent.

(c) **(2 pts)** Check that $\Phi(D_0) = 0$ and that $\Phi(D_i) \geq \Phi(D_0)$ for all $i > 0$.

$\Phi(D_0) = 0$ since initially there are no nodes at all.

$\Phi(D_i) \geq \Phi(D_0)$ since the number of nodes with a grandparent can never be negative.

(d) **(4 pts)** Calculate the amortized cost for each Make-Set.

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

$= 1 + 0 - 0$

1: actual cost is constant; $0 - 0$: during the Make Sets, no nodes have grandparents

(e) **(5 pts)** Note that the Unions are operations $n+1$ through $2n-1$. Verify that the amortized cost for *all* the Unions is $n - 1 + \Phi(D_{2n-1})$. *Hint:* Use the formula defining amortized cost and apply the telescoping sums trick.

$$\sum_{i=n+1}^{2n-1} \hat{c}_i = \sum_{i=n+1}^{2n-1} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \quad \text{by def.}$$

$$= \sum_{i=n+1}^{2n-1} c_i + (\Phi(D_{2n-1}) - \Phi(D_n)) \quad \text{by telescoping sums trick}$$

$$= (n-1) + (\Phi(D_{2n-1}) - \Phi(D_n)) \quad \text{since each union's actual cost is constant}$$

$$= (n-1) + \Phi(D_{2n-1}) \quad \text{since no nodes have grandparents after the MakeSets}$$

$$= O(n) \quad \text{since in worst case (almost) all } n \text{ nodes have grandparents}$$

(f) **(5 pts)** Calculate the amortized cost for each Find-Set.

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \quad \text{by def.}$$

$$= d + \Phi(D_i) - \Phi(D_{i-1}) \quad \text{where } d \text{ is distance from root to arg. of Find Set}$$

$$= d + (d-2) \quad \text{since all but the top 2 nodes on that path lose their grandparent}$$

$$= 2$$

(g) **(5 pts)** What is the total time for the entire sequence of operations? Explain your answer.

$$n + O(n) + 2 \cdot n = O(n)$$

↑ for the MakeSets; ↑ for the Unions; ↑ for the FindSets