TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS — 2006

MID-YEAR

COMP 303
Design and Analysis of Algorithms

Time Allowed: 3 Hours

Instructions: There are 5 questions of varying weights.
Available marks total 180, so allow about one minute per mark.
Answer all questions.

Calculators and other electronic devices are **not** permitted.
Printed foreign language dictionaries **are** permitted.

The following definitions are provided for your convenience. You may find it useful to tear off this front page of the paper.

Asymptotic notation:

$$\begin{aligned} O(g(n)) &= \{f(n) \mid (\exists c)(\mathbf{aa}\, n)[0 \le f(n) \le c.g(n)]\} \\ \Omega(g(n)) &= \{f(n) \mid (\exists c)(\mathbf{aa}\, n)[f(n) \ge c.g(n) \ge 0]\} \\ \Theta(g(n)) &= \{f(n) \mid (\exists c, d)(\mathbf{aa}\, n)[0 \le c.g(n) \le f(n) \le d.g(n)]\} \end{aligned}$$

Master Theorem: Let $T(n)$ be defined by the recurrence $T(n) = aT(n/b) + f(n)$.
Let $\alpha = \log_b a$.

1. If $(\exists \epsilon > 0)[f(n) \in O(n^{\alpha-\epsilon})]$ then $T(n) \in \Theta(n^{\alpha})$.
2. If $f(n) \in \Theta(n^{\alpha})$ then $T(n) \in \Theta(n^{\alpha} \log n)$.
3. If $(\exists \epsilon > 0)[f(n) \in \Omega(n^{\alpha+\epsilon})]$ and $(\exists c < 1)(\mathbf{aa}\, n)[a.f(n/b) \le c.f(n)]$ then $T(n) \in \Theta(f(n))$.

Logarithms:

$$\log_a x = y \text{ if and only if } a^y = x$$

THIS PAGE INTENTIONALLY LEFT BLANK

Question 1. [38 marks]

(a) Suppose we have analysed an algorithm and discovered that it does exactly $\frac{n(n+1)}{2}$ elementary operations for any input of size n. For each of the following statements, state whether the statement is true, false, or unknown.

(i) The complexity of the algorithm is in $O(n^2)$.

(ii) The complexity of the algorithm is in $\Theta(n^2)$.

(iii) The complexity of the algorithm is in $O(n \log n)$.

(iv) The complexity of the algorithm is in $\Omega(n \log n)$.

(v) The complexity of the problem is in $O(n^2)$.

(vi) The complexity of the problem is in $\Theta(n^2)$. [6 marks]

(b) Explain how we usually justify ignoring constant factors in algorithm analysis. Describe circumstances that might require us to consider constant factors. [4 marks]

(c) Using the definitions for O, Ω, and Θ given on the front page of this paper, show that:

(i) $n^2 + n \in O(n^3)$

(ii) $n^2 + n \in \Theta(2n^2)$

(iii) $f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ [10 marks]

(d) You are given two problems A and B, and told that A is NP-complete. How would you:

(i) show that B is NP-Hard?

(ii) show that B is NP-Complete? [6 marks]

(e) For each of the following recurrence relations, give the asymptotic complexity (Θ bound) of $T(n)$. Justify your answers using either induction or the Master Theorem (given on page 1 of the paper), as appropriate.

(i) $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ T(n-1) + n, & \text{otherwise} \end{cases}$

(ii) $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 8T(\lceil \frac{n}{2} \rceil) + n^3, & \text{otherwise} \end{cases}$

(iii) $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 9T(\lceil \frac{n}{3} \rceil) + n^3, & \text{otherwise} \end{cases}$ [12 marks]

Question 2. [36 marks]

(a) Write pseudocode to define the basic structure of a typical divide-and-conquer algorithm. Explain the components of your algorithm. [6 marks]

(b) Divide-and-conquer algorithms deal with subproblems. What *two* properties must these subproblems have for a divide-and-conquer algorithm to be efficient? [4 marks]

(c) Give an efficient divide and conquer algorithm for multiplying two n-digit numbers, assuming that the only available operations are multiplication of single digits, and addition of numbers. [12 marks]

(d) Give the general structure of the proof of a divide and conquer algorithm, and use it to show your algorithm from part (c) is correct. [10 marks]

(e) Give a recurrence relation for the number of single-digit multiplications performed by your algorithm. [4 marks]

Question 3. [40 marks]

Given a *connected, undirected* graph $G = (V, E)$ with weighted edges, the single-source shortest path problem (SSSP) asks for a path between vertex 1 and every other vertex in V.

Below is **Dijkstra's algorithm** for the single-source shortest path problem. The input is the graph, represented as an adjacency matrix L, in which $L[i, j]$ is the length of the edge between vertices i and j, if one exists; otherwise ∞.

```
SSSP(L)
  C ← {2, 3, ..., n}
  D[1] ← 0
  for i ← 2 to n
    D[i] ← L[1, i]
  while C ≠ {}
    v ← the element x of C that makes D[x] minimum
    C ← C\{v}
    for each w ∈ C
      D[w] ← min(D[w], D[v] + L[v, w])
```

(a) Which of the algorithm design techniques discussed in the course does this algorithm use? Explain why. [4 marks]

(b) After this algorithm has finished executing, $D[i]$ gives the *length* of the shortest path between vertex 1 and any vertex i. Explain how to modify this algorithm so as to be able to determine the actual path, not just the length of the path. [6 marks]

(c) The most efficient implementation of Dijkstra's algorithm gives time $\Theta(e \log_2 n)$, where n is the number of vertices and e is the number of edges in the graph. Explain how to achieve this running time. [4 marks]

Let $G = (V, E)$ be an undirected graph, and $u \in V$ be a vertex. The *connected component* of u in G is the largest connected subgraph $G' = (V', E')$, with $V' \subseteq V$, $E' \subseteq E$, and $u \in V'$.

We want to label all the connected components in a graph so that each vertex has the same label as all the vertices in its connected component, but a different label from any vertex in another connected component.

(d) Describe an algorithm that uses depth-first search to produce such a labelling for an arbitrary undirected (but possibly disconnected) graph $G = (V, E)$. [9 marks]

(e) Explain how to use Dijkstra's algorithm to solve the same problem. [9 marks]

(f) What are the running times of your algorithms? Under what circumstances does one algorithm perform better than the other? Include a description of any relevant implementation details. [8 marks]

Question 4. [24 marks]

Suppose you have designed an algorithm whose solution is described in terms of a set of n elements. Initially, the set is assumed to be partitioned into n subsets of 1 element each. The algorithm needs two operations: **compare**(e_1, e_2) returns true iff the elements e_1 and e_2 are in the same subset, and **merge**(e_1, e_2) merges the subsets containing elements e_1 and e_2 into a single subset.

(a) Describe an efficient data structure for your algorithm to use to represent the set of subsets. [10 marks]

(b) Describe how the **compare** and **merge** operations can be efficiently implemented using your data structure. [10 marks]

(c) Suppose the algorithm requires a total of c **compare/merge** operations. What is the best possible asymptotic cost of the algorithm? [4 marks]

Question 5. [42 marks]

The University has a ten million dollar budget which it wishes to spend on improving student numbers. The marketing department has proposed a programme of advertising on television and radio and via glossy brochures distributed with the education supplement of the newspaper. Staff have argued that the money would be better spent on improving the attractiveness of the university via increased staff salaries and better facilities for students. Unfortunately, nobody has suggested reducing the fees.

Our problem is to find a way of selecting from among a set of proposed expenditures $i \in 1..n$, each with specified cost $c[i]$ and predicted number $p[i]$ of expected enrolments, in such a way that the total cost is no greater than the available funds M and the expected extra enrolments are maximised. For example, the following table summarizes, for each of the proposed expenditures ($i \in \{1,2,3,4,5\}$), the cost $c[i]$ (in millions of dollars); the projected number $p[i]$ of extra enrolments that would result; and, to save you the trouble of calculation, the ratio $p[i]/c[i]$ (in enrolments per million dollars). Also shown is the total money available, M.

i	Expenditure	$c[i]$ ($mill)	$p[i]$	$p[i]/c[i]$ (enrolments per $mill)
1	Television	2	1000	500
2	Radio	3	1800	600
3	Brochure	4	1200	300
4	Salaries	5	4000	800
5	Facilities	2.5	1750	700

$M = 10$ ($mill).

(a) Define **acceptable solution** and **correct solution** for the general problem. Assume the inputs for the problem are n, $c[1..n]$, $p[1..n]$, and M. Assume the output is an array $f[1..n]$ of integer values 0 or 1 ($f[i] = 1$ if item i should be spent, and $f[i] = 0$ otherwise). [6 marks]

(b) You initially try to solve the problem using a greedy algorithm: look at each item in decreasing order of enrolments per million dollars, funding it only if it can be done with the available money. Show that the greedy algorithm does **not** give the optimal solution for the problem instance described above. [4 marks]

(c) Show that the general problem has the **optimal substructure** property. [10 marks]

(d) Describe an appropriate **dynamic programming** algorithm for solving the general problem. [8 marks]

(e) Outline an algorithm for solving the general problem by **backtracking search**. You may find it useful to explain your algorithm via a small example. [8 marks]

(f) Describe how one may **prune** and **bound** the search tree to make backtracking search more efficient. [6 marks]

****