Victoria University of Wellington

DEGREE EXAMINATIONS — 2001

MID-YEAR

COMP 303

DESIGN AND ANALYSIS
OF ALGORITHMS

Time Allowed: 3 Hours

Instructions:

- This exam consists of seven (7) questions of varying weights. Answer all questions.
- Read each question completely before starting it.
- The total mark for each question is given in square brackets "[]". The mark for each part of a question is given in round brackets "()".
- The exam is out of 180 marks so allocate about 1 minute for each mark.
- *Show all working details.* Partial credit will be given for sensible but incomplete attempts, provided intent is clear.
- When you are asked to describe an algorithm, give the most efficient (for time) algorithm possible, unless otherwise stated.

Question 1. [24 marks]

(a) Consider the following definitions for asymptotic notation.

$$\Omega(g(n)) = \left\{ \; h \;\mid\; (\exists c)(\textbf{aa}\ n)[c.g(n) \leq h(n)] \; \right\}$$

$$O(g(n)) = \left\{ \; h \;\mid\; (\exists c)(\textbf{aa}\ n)[h(n) \leq c.g(n)] \; \right\}$$

$$\Theta(g(n)) = \left\{ \; h \;\mid\; (\exists c, d)(\textbf{aa}\ n)[c.g(n) \leq h(n) \leq d.g(n)] \; \right\}$$

In all cases, c and d are positive real numbers.

(i) Using your own words, explain the three definitions. (5 marks)

(ii) Explain how Ω, O, and Θ are typically used to express bounds on the best, average, and worst-case behaviours of algorithms, and on the complexities of problems. (5 marks)

(b) Consider the recurrence:

$$T(n) \leq \begin{cases} cn, & 0 \leq n < 70 \\ T(\lfloor \frac{n}{5} \rfloor) + T(\lfloor \frac{3n}{4} \rfloor) + cn, & n \geq 70 \end{cases}$$

(i) Prove by mathematical induction that $T(n) \leq 20cn$. (8 marks)

(ii) Prove that $T(n) \in O(n)$ using only the definition of "O" given above (that is, find appropriate constants and demonstrate that the appropriate relationships exist). (6 marks)

Question 2. [20 marks]

(a) Describe the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-**Complete** and $\mathcal{NP}$-**Hard**. (6 marks)

(b) Under each of the following assumptions, give the relationships between the four complexity classes mentioned above, perhaps by showing them on a Venn diagram.

(i) $\mathcal{P} = \mathcal{NP}$ (3 marks)

(ii) $\mathcal{P} \neq \mathcal{NP}$ (3 marks)

(c) You are required to solve some problem which you know to be $\mathcal{NP}$-**Hard**. Write a paragraph outlining several ways in which you might use *probability* and/or *heuristics* to find a tractable solution for some instances of the problem. (8 marks)

Question 3. [30 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) Outline the structure of a proof of a typical divide-and-conquer algorithm. In your answer, you should refer to *assumptions* **A** and *requirements* **R** for the subproblems. (8 marks)

(d) Give a *recurrence relation* that describes the running time of a typical divide-and-conquer algorithm. Explain how each part of your recurrence relates to the components of your divide-and-conquer algorithm in **(a)**. (8 marks)

(e) Explain how the recurrence relation from **(d)** can have three different forms of solution, depending on the relative values of the parts. (4 marks)

Question 4. [22 marks]

The following algorithm is intended to sort an array A, whose index range is $a..c$.

Quicksort(A, a, c)
  if $a < c$ then
    Partition(A, a, b, c)
    Quicksort(A, a, b)
    Quicksort$(A, b + 1, c)$

(a) Give a specification for **Partition**, making sure you precisely define the assumptions it makes and the requirements on its result. (Do **not** write an algorithm.) (8 marks)

(b) Give an informal argument that, for any n, there is an input of size n such that **Quicksort** takes time $\Omega(n^2)$. (6 marks)

(c) Prove that the worst-case running time for **Quicksort** is $\Theta(n^2)$ for a list of size n, assuming that **Partition** takes time $O(m)$ for lists of size m. (8 marks)

Question 5. [36 marks]

A thief, known for his ability to steal gemstones, decides to change to art objects. He breaks into an art gallery and is once again confronted with more to steal than he can carry. Unlike the jeweller's shop, where all the gemstones were of the same weight (although of different values), the paintings he can steal from the art gallery are of different weights as well as different values.

(a) Unfortunately, his greedy algorithm (shown below) that worked so well in the jeweller's shop doesn't work in the art gallery.

> **Greedy:** Look at each painting in decreasing order of value, and take it only if it will fit in the bag.

Give an example that demonstrates why the greedy algorithm does not work in the art gallery, explaining clearly what goes wrong. (4 marks)

(b) Since the greedy algorithm doesn't work in this case, the thief switches to Plan B: a *dynamic programming* algorithm.

Let the maximum capacity of the thief's sack be M kilogrammes. For $i \in 1..n$, let V_i be the *value* of the ith painting (listed in no particular order), and let W_i be the *weight* of the ith painting. Assume that the weights of all the paintings are integers (kilogrammes) and their values are also integers (dollars).

To develop an algorithm, we must first decide what a *subproblem* is. In this case, a subproblem consists of reducing the number of choices (paintings) available, and reducing the amount the thief is able to carry. So, let c_{iw} be the value of the maximum haul that fits in a sack that can carry only w kilogrammes (where $0 \leq w \leq M$), looking only at paintings $1, 2, \ldots, i$. What we want to do is build up a recursive definition for c_{iw}.

(i) Show that the art gallery problem has the *optimal substructure* property. [Hint: use the decomposition into subproblems suggested above.] (8 marks)

(ii) Give a *dynamic programming* algorithm to solve the art gallery problem. (8 marks)

(c) Another approach to the art gallery problem is to use a *backtracking search*.

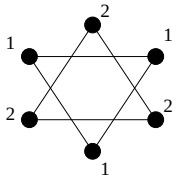
(i) Outline a backtracking search algorithm for the art gallery problem. You may find it useful to explain your algorithm via a small example. (8 marks)

(ii) Describe a related problem that gives an upper bound on the value of a solution to the art gallery problem, and explain how solutions to the related problem may be used to improve the efficiency of the backtracking search. (8 marks)

Question 6. [24 marks]

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

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



(a) Explain how to use **Dijkstra's algorithm** (for the single-source shortest path problem) to produce such a labelling for an arbitrary undirected (but possibly disconnected) graph $G = (V, E)$. (8 marks)

(b) Describe a backtracking search algorithm for the same problem. (8 marks)

(c) 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 7. [24 marks]

Suppose you have n magnetic tapes, each containing records in sorted order. Let m_i (for $1 \leq i \leq n$) be the number of records on tape i.

You want to create a master tape that contains the contents of these tapes and is also in sorted order. Your system allows you to merge the records on two tapes containing p and q records respectively with $p+q$ record movements, but you can only perform this operation on two tapes at a time to produce a third tape containing both sets of records (i.e., the primitive operation is *2-file merge*).

(a) Design an algorithm to determine the order to merge the n tapes that will **minimise** the number of record movements required to actually merge the tapes. You may assume as many extra tapes as you need. [Hint: How would you represent a series of tape-merges pictorially? Like the data compression problem discussed in class, this problem is an instance of the **minimum weighted external path length** problem.] (16 marks)

(b) What is the asymptotic running time for your algorithm? Informally justify your answer. Include any relevant implementation details. (8 marks)

****