VICTORIA UNIVERSITY OF WELLINGTON

*Te Whare Wananga o te Upoko o te Ika a Maui*

# EXAMINATIONS — 2002

MID-YEAR

COMP 303

Design and Analysis of Algorithms

Time Allowed: 3 Hours

Instructions: There are 6 questions of various weights.
Answer all questions.
Available marks total 180, so allow about one minute per mark.

Calculators are **not** permitted.
Foreign language dictionaries **are** permitted.

## Useful results

**Asymptotic notation**

$$\begin{aligned} O(f(n)) &= \{g(n) \mid (\exists c)(\mathbf{aa}\, n)[g(n) \leq c.f(n)]\} \\ \Omega(f(n)) &= \{g(n) \mid (\exists c)(\mathbf{aa}\, n)[g(n) \geq c.f(n)]\} \\ \Theta(f(n)) &= O(f(n)) \cap \Omega(f(n)) \end{aligned}$$

**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 $(\exists c < 1)(\mathbf{aa}\, n)[a.f(n/b) \leq c.f(n)]$ then $T(n) \in \Theta(f(n))$.

## Question 1. [30 marks]

**(a)** [4 marks] Explain the use of the asymptotic notations $O$, $\Omega$, and $\Theta$ in analysis of algorithms and problems.

**(b)** [3 marks] Explain how the *principle of invariance* justifies our use of asymptotic notation to analyse algorithms.

**(c)** [3 marks] Why might asymptotic notation sometimes be inadequate for comparing algorithms?

**(d)** [5 marks] Using the definitions for $O$, $\Omega$, and $\Theta$ given on the front page of this paper, show that

$$2n^2 + 4n \in \Theta(n^2)$$

**(e)** [15 marks] For each of the following recurrence relations, give the asymptotic complexity ($\Theta$ bound) of $T(n)$. **Justify your answers**, using the Master Theorem (provided on the front page of this paper) or mathematical induction, as appropriate.

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

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

**(iii)** $T(n) = \begin{cases} 1, & \text{if } n = 0 \\ 4T(\lceil \frac{n}{2} \rceil) + n^2, & \text{otherwise} \end{cases}$

## Question 2. [34 marks]

Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task $a_i$ requires $p_i$ seconds of processing time to complete, once it has started. You have one computer on which to run these tasks, and the computer can run only one task at a time.

Assume that the clock is set to 0 at the start of processing, and let $f_i$ be the **finish time** of task $a_i$, that is the time at which task $a_i$ completes processing. Your goal is to schedule all $n$ tasks in an order that minimizes the **average** finish time: that is, to minimize $\frac{1}{n}\sum_{i=1}^{n} f_i$.

For example, suppose there are two tasks, $a_1$ and $a_2$, with $p_1 = 6$ and $p_2 = 4$. If we schedule $a_1$ first, the finish times are $f_1 = 6$ and $f_2 = 10$, and the average completion time is $(6 + 10)/2 = 8$. If we schedule $a_2$ first, the finish times are $f_2 = 4$ and $f_1 = 10$, and the average finish time is $(4 + 10)/2 = 7$.

**(a)** [6 marks] Describe a **greedy algorithm** that will solve this problem, returning the optimal sequence of tasks.

**(b)** [8 marks] Prove that your algorithm always minimizes the average finish time. (Hint: follow the pattern for proving the correctness of a greedy algorithm.)

Now suppose that there is a further restriction on the possible orderings of the tasks, given by a partial order relation $\prec$ on the tasks. That is, we will say that $a_i \prec a_j$ if task $a_i$ **must be completed** before task $a_j$ can commence. For example, if $a_1 \prec a_2$, the only valid ordering of the tasks is $a_1$ followed by $a_2$. Note that the ordering is not necessarily **total**: there may be pairs $a_i$ and $a_j$ such that neither $a_i \prec a_j$ nor $a_j \prec a_i$. In that case, activities $a_i$ and $a_j$ may be scheduled in either order.

**(c)** [4 marks] Describe a suitable data structure for representing the partial order $\prec$.

**(d)** [8 marks] Describe a modification of your algorithm from **(b)** above that will produce the optimal sequence of tasks while respecting all ordering constraints.

**(e)** [4 marks] Describe how to modify your proof from **(c)** above for the new algorithm.

**(f)** [4 marks] What is the asymptotic complexity of your algorithm from **(d)**?

## Question 3. [26 marks]

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

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

**(c)** [6 marks] 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.

**(d)** [6 marks] 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)**.

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

## Question 4. [20 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)** [8 marks] Describe an efficient data structure for your algorithm to use to represent the set of subsets.

**(b)** [8 marks] Describe how the **compare** and **merge** operations can be efficiently implemented using your data structure.

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

## Question 5. [48 marks]

The senior partner in a large firm of criminal lawyers is trying to plan her case-load for the next week. As the senior partner, she gets to choose any cases she wants from the large number available: she will choose the cases that maximize her income while not taking longer in total than the number of hours she wishes to work.

Our experienced lawyer has developed several specialties over the years, and she knows exactly how long those cases will take and what they will earn. She knows, for example, that defending a drug dealer takes 5 hours and earns $1000; defending a jewel thief takes 12 hours and earns $1800; while defending a poor student who has been arrested for busking without a licence takes 2 hours and earns $200. The lawyer does not want to work more than 24 hours in the week (she needs two afternoons free for golf). Currently awaiting trial in the police cells are 3 drug dealers, 4 jewel thieves, and 6 buskers.

The above information is summarized in the following table, in which the cases are numbered $i = 1, 2, 3$; $t[i]$ gives the time per case $i$; $e[i]$ gives the earnings per case $i$; $k[i]$ gives the number of cases $i$ available; and $T$ is the total number of hours available:

| Case | $i$ | $t[i]$ (hours) | $e[i]$ (dollars) | $k[i]$ | $e[i]/t[i]$ |
|---|---|---|---|---|---|
| Drug dealers | 1 | 5 | 1000 | 3 | 200 |
| Jewel thieves | 2 | 12 | 1800 | 4 | 150 |
| Buskers | 3 | 2 | 200 | 6 | 100 |

$T = 24$

Our lawyer (who did not study COMP303) thinks she can choose the optimal case-load using a greedy algorithm:

> **Greedy:** Look at each case in decreasing order of earnings per hour, taking the case only if it can be completed in the available time.

**(a)** [4 marks] Define **acceptable solution** and **correct solution** for the general problem: that is, assuming that the values $n$ and $T$ and the arrays $t$, $e$, and $k$ are inputs.

**(b)** [4 marks] Show that the greedy algorithm does **not** give the optimal solution for this problem instance.

**(c)** [8 marks] Show that the general problem has the **optimal substructure** property.

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

**(e)** [4 marks] Briefly outline a proof that your algorithm is correct.

**(f)** [4 marks] State the asymptotic complexity of your algorithm. **Justify your answer.**

Another approach to the lawyer's problem is to use a **backtracking search**.

**(g)** [8 marks] Describe a backtracking search algorithm for the general problem. You may find it useful to illustrate your algorithm using the example problem instance above.

**(h)** [8 marks] Describe a related problem that gives an upper bound on the value of a solution to the lawyer's problem, and explain how solutions to the related problem may be used to improve the efficiency of the backtracking search.

## Question 6. [22 marks]

**(a)** [4 marks] Describe, in informal terms, the complexity classes $\mathcal{P}$, $\mathcal{NP}$, **$\mathcal{NP}$-Complete**.

**(b)** [8 marks] Suppose that $\mathcal{P} \neq \mathcal{NP}$. Draw a Venn diagram showing the following complexity classes:

**(i)** $\Theta(n)$

**(ii)** $O(n^2)$

**(iii)** $\mathcal{P}$

**(iv)** $\mathcal{NP}$

**(v)** **$\mathcal{NP}$-Complete**.

**(c)** [10 marks] The **Travelling Salesperson** problem is defined as follows.

Given a connected, directed graph $G$ with weighted edges, find a way of visiting every node of the graph by traversing its edges in a way that minimizes the total weight of the edges traversed.

**(i)** Compare and contrast this problem with the **Hamiltonian Cycle** problem.

**(ii)** Do you think the **Travelling Salesperson** problem has a polynomial solution? Justify your answer.

**(iii)** Do you think **Travelling Salesperson** is **$\mathcal{NP}$-Hard**? Justify your answer.

**********