TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2016

TRIMESTER 2

COMP 361

DESIGN AND ANALYSIS OF ALGORITHMS

Time Allowed:	TWO HOURS
CLOSED BOOK	
Permitted materials:	Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination. Non-electronic foreign language to English dictionaries are permitted.
Instructions:	The exam will be marked out of 120 marks. Attempt all questions.

Questions	Marks
1. Divide and Conquer	[30]
2. Greedy Algorithms	[30]
3. Dynamic Programming	[30]
4. Guest Lectures	[10]
5. Approximation Algorithms (Hard)	[20]

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

Asymptotic notation:

$$O(g(n)) = \{f(n) \mid (\exists d)(\mathbf{aa}\, n)[0 \le f(n) \le d.g(n)]\}$$

$$\Omega(g(n)) = \{f(n) \mid (\exists c > 0)(\mathbf{aa}\, n)[f(n) \ge c.g(n) \ge 0]\}$$

$$\Theta(g(n)) = \{f(n) \mid (\exists c > 0, d)(\mathbf{aa}\, n)[0 \le c.g(n) \le f(n) \le d.g(n)]\}$$

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:

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

Matrix Multiplication: When multiplying $m \times n$ matrix A by $n \times p$ matrix B one gets an $m \times p$ matrix C where the *row by column* strategy is followed: element $c_{i,j} = a_{i,1} \times b_{1,j} + a_{i,2} \times b_{2,j} + \ldots + a_{i,n} \times b_{n,j}$.

1. Divide and Conquer **(30 marks)**

(a) **(10 marks)** How many lines, as a function of n (in $\Theta()$ form), does the following program print? You may assume n is a power of 2.

Hint: Write a recurrence that represents the number of lines printed and solve it.

```
function f(n)
  if n > 1:
    print_line(''still going'')
    f(n/2)
    f(n/2)
```

(b) **(10 marks)** You are given an array of n elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array in time $O(n\ logn)$.

(c) **(10 marks)** (**Hard**) Suppose you are given matrices A, B, C which are each n by n and you wish to check whether $AB = C$. You can do this in $O(n^{log_2 7})$ steps using Strassen's algorithm. In this subquestion we will explore a much faster, $O(n^2)$ randomised test.

(i) Let $\mathbf{v}$ be an n-dimensional vector whose entries are randomly and independently chosen to be 0 or 1 (each with probability 1/2). Prove that if M is a non-zero n by n matrix (i.e. at least one element is not a zero), then $Pr[M\mathbf{v} = 0] \leq 1/2$.

(ii) Show that $Pr[AB\mathbf{v} = C\mathbf{v}] \leq 1/2$ if $AB \neq C$. Why does this give an $O(n^2)$ randomised test for checking whether $AB = C$?

2. Greedy Algorithms **(30 marks)**

Here's a problem that occurs in automatic program analysis. For a set of variables $x_1, \ldots, x_n$, you are given some *equality* constraints, of the form "$x_i = x_j$" and some *disequality* constraints, of the form "$x_i \neq x_j$." Is it possible to satisfy all of them?

For instance, the constraints

$$x_1 = x_2,\ x_2 = x_3,\ x_3 = x_4,\ x_1 \neq x_4$$

cannot be satisfied.

(a) **(15 marks)** Give an efficient algorithm that takes as input m constraints over n variables and decides whether the constraints can be satisfied.

Hint: One possible option is to consider a graph representation of this problem where each node is a variable (e.g. x_i) and an edge represents an "equality constraint".

(b) **(10 marks)** Argue (or do a proof sketch) that your algorithm is correct.

(c) **(5 marks)** State the asymptotic cost of your algorithm and justify why it's correct.

3. Dynamic Programming **(30 marks)**

You are given a string of n characters: $s[1 \dots n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "itwasthebestoftimes..."). You wish to reconstruct the original document using a dictionary, which is available in the form of a Boolean function `dict()`. For any string w,

$$\texttt{dict}(w) = \begin{cases} \texttt{true} & \text{if } w \text{ is a valid word,} \\ \texttt{false} & \text{otherwise.} \end{cases}$$

(a) **(20 marks)** Give a dynamic programming algorithm that determines whether the string $s[]$ can be reconstituted as a sequence of valid words. The running time should be at most $O(n^2)$, assuming calls to `dict` take unit time.

(b) **(10 marks)** In the event that the string is valid, modify your algorithm to output the corresponding sequence of words.

4. Guest Lectures **(10 marks)**

(a) **(5 marks)** State what the *Tutte Polynomials* are used for as presented in David's guest lecture.

(b) **(5 marks)** Outline the main steps of the *JPEG Encoding Algorithm* as discussed in Neil's guest lecture.

5. Approximation Algorithms (Hard) **(20 marks)**

Recall the knapsack problem from the lectures and assignment 3. There are n items, where the ith item is worth v_i dollars and weighs w_i grams. We are also given a knapsack that can hold at most W grams. Here, we add the further assumptions that each weight w_i is at most W and that the items are indexed in monotonically decreasing order of their values: $v_1 \geq v_2 \geq \ldots \geq v_n$.

In the 0-1 knapsack problem, we wish to find a subset of the items whose total weight is at most W and whose total value is maximum. The fractional knapsack problem is like the 0-1 knapsack problem, except that we are allowed to take a fraction of each item, rather than being restricted to taking either all or none of each item. If we take a fraction x_i of item i, where $0 \leq x_i \leq 1$, we contribute $x_i w_i$ to the weight of the knapsack and receive value $x_i v_i$. Our goal is to develop a polynomial-time approximation algorithm for the 0-1 knapsack problem.

In order to design a polynomial-time algorithm, we consider restricted instances of the 0-1 knapsack problem. Given an instance I of the knapsack problem, we form restricted instances I_j, for $j = 1, 2, \ldots, n$, by removing items $1, 2, \ldots, j-1$ and requiring the solution to include item j (all of item j in both the fractional and 0-1 knapsack problems). No items are removed in instance I_1. For instance I_j, let P_j denote an optimal solution to the 0-1 problem and Q_j denote an optimal solution to the fractional problem.

(a) **(4 marks)** Argue that an optimal solution to instance I of the 0-1 knapsack problem is one of $\{P_1, P_2, \ldots, P_n\}$.

(b) **(4 marks)** Prove that we can find an optimal solution Q_j to the fractional problem for instance I_j by including item j and then using the greedy algorithm in which at each step, we take as much as possible of the unchosen item in the set $\{j+1, j+2, \ldots, n\}$ with maximum value per gram v_i / w_i.

(c) **(4 marks)** Prove that we can always construct an optimal solution Q_k to the fractional problem for instance I_j that includes at most one item fractionally. That is, for all items except possibly one, we either include all of the item or none of the item in the knapsack.

(d) **(4 marks)** Given an optimal solution Q_j to the fractional problem for instance I_j, form solution R_j from Q_j by deleting any fractional items from Q_j. Let $v(S)$ denote the total value of items taken in a solution S. Prove that $v(R_j) \geq v(Q_j)/2 \geq v(P_j)/2$.

(e) **(4 marks)** Give a polynomial-time algorithm that returns a maximum-value solution from the set $\{R_1, R_2, \ldots, R_n\}$, and prove that your algorithm is a polynomial-time approximation algorithm for the 0-1 knapsack problem.

* * * * * * * * * * * * * * *