Student ID: .....

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2014

TRIMESTER 2

COMP 361

DESIGN AND ANALYSIS OF ALGORITHMS

Time Allowed: TWO HOURS

Instructions: Closed Book

Attempt ALL Questions.

Answer in the appropriate boxes if possible — if you write your answer elsewhere, make it clear where your answer can be found.

The exam will be marked out of 120 marks.

Non-electronic foreign language dictionaries are permitted.

Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.

Questions	**Marks**
1. Divide and Conquer	[30]
2. Greedy Algorithms	[35]
3. Dynamic Programming	[30]
4. Approximation Algorithms	[25]

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

Asymptotic notation:

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

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

Logarithms:

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

Question 1. Divide and Conquer [30 marks]

(a) [4 marks] Use pseudo code to describe the basic structure of a typical divide-and-conquer algorithm. Explain the components of your algorithm.

(b) [4 marks] State a requirement that an array is sorted which can be used to prove that a sorting algorithm is correct (similar to how we did it in the first few lectures).

Consider the following algorithm that sorts the array A between indices i and j (initially set to the first and last element indices of the array A):

THIRDS-SORT(A, i, j)

```
1 | if A[i] > A[j]
2 |     then exchange A[i] ↔ A[j]
3 | if i + 1 ≥ j
4 |     then return
5 | k ← ⌊(j−i+1)/3⌋                  // Round down.
6 | THIRDS-SORT (A, i, j − k)        // First two-thirds.
7 | THIRDS-SORT (A, i + k, j)        // Last two-thirds.
8 | THIRDS-SORT (A, i, j − k)        // First two-thirds again.
```

(c) [7 marks] Give the general structure of the proof of correctness of a divide and conquer algorithm, and use it to show that the THIRDS-SORT algorithm above correctly sorts the input array.

(d) [7 marks] Give a recurrence relation for the worst-case running time of THIRDS-SORT and a tight asymptotic (Θ) bound on the worst-case running time.

(e) Compare the worst-case running time of THIRDS-SORT with that of:

(i) [2 marks] insertion sort,

(ii) [2 marks] mergesort,

(iii) [2 marks] heapsort, and

(iv) [2 marks] quicksort.

Question 2. Greedy Algorithms [35 marks]

Consider the problem of making change of n cents using the fewest number of coins. Assume that each coin's value is an integer.

(a) [10 marks] Describe a greedy algorithm to make change consisting of 25c, 10c, 5c, and 1c coins. Prove that your algorithm yields an optimal solution. Follow the lecture slides style of presenting and proving greedy algorithms.

(b) [10 marks] Suppose that the available coins are the powers of c, i.e. values are $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k > 1$. Show that the greedy algorithm always yields an optimal solution.

(c) [5 marks] Give a set of coin values for which the greedy algorithm does not yield an optimal solution. Your set should include a 1c coin so that there is a solution for every value of n.

(d) [10 marks] Give an $O(nk)$-time algorithm that makes change of n cents using coins from any given set of k different coin values, assuming that one of the coins is 1c. Show that the time your algorithm takes is as required.

Question 3. Dynamic Programming [30 marks]

A certain string-processing language allows a programmer to break a string into two pieces. Because this operation copies the string, it costs n time units to break a string of n characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks occur can affect the total amount of time used. For example, suppose that the programmer wants to break a 20-character string after characters 2, 8, and 10 (numbering the characters in ascending order from the left-hand end, starting from 1). If she programs the breaks to occur in left-to-right order, then the first break costs 20 time units, the second break costs 18 time units (breaking the string from characters 3 to 20 at character 8), and the third break costs 12 time units, totalling 50 time units. If she programs the breaks to occur in right-to-left order, however, then the first break costs 20 time units, the second break costs 10 time units, and the third break costs 8 time units, totalling 38 time units. In yet another order, she could break first at 8 (costing 20), then break the left piece at 2 (costing 8), and finally the right piece at 10 (costing 12), for a total cost of 40.

(a) [10 marks] Design an algorithm that, given the numbers of characters after which to break, determines a least-cost way to sequence those breaks. More formally, given a string S with n characters and an array $L[1 \ldots m]$ containing the break points, compute the lowest cost for a sequence of breaks, along with a sequence of breaks that achieves this cost.

(b) [10 marks] Prove that this problem has an *optimal substructure* property.

(c) [10 marks] Prove that your algorithm is correct.

Question 4. Approximation Algorithms [25 marks]

Suppose you are given a set of positive integers $A = a_1, a_2, \ldots, a_n$ and a positive integer B. A subset $S \subseteq A$ is called *feasible* if the sum of the numbers in S does not exceed B:

$$\sum_{a_i \in S} a_i \leq B$$

The sum of the numbers in S will be called the *total sum* of S.

You would like to select a feasible subset S of A whose total sum is as large as possible.

Example. If $A = 8, 2, 4$ and $B = 11$, then the optimal solution is the subset $S = 8, 2$.

(a) [10 marks] Here is an algorithm for this problem.

```
Initially S = ∅
Define T = 0
For i = 1, 2, ..., n
  If T + a_i ≤ B then
    S ← S ∪ a_i
    T ← T + a_i
  Endif
Endfor
```

Give an instance in which the total sum of the set S returned by this algorithm is less than half the total sum of some other feasible subset of A.

(b) [15 marks] Give a polynomial-time approximation algorithm for this problem with the following guarantee:

- It returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set $S' \subseteq A$.

Your algorithm should have a running time of at most $O(n \log n)$ (note that *at most* means that a running time of $\Theta(n)$ is acceptable).

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