IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2014

EEE/EIE PART II: MEng, BEng and ACGI

Corrected Copy

ALGORITHMS AND COMPLEXITY

Monday, 16 June 2:00 pm

Time allowed: 1:30 hours

There are THREE questions on this paper.

Answer ALL questions. Each question carries 20 marks.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible   First Marker(s) :   M.M. Draief

                                      Second Marker(s) :   D.B. Thomas

© Imperial College London

ALGORITHMS AND COMPLEXITY

1. Give a tight bound for each of the following recurrence relations.
    Carefully justify your answers.

    a) $T(n) = 4T(n/2) + n^3$,

    [5]

    b) $T(n) = 17T(n/4) + n^2$,

    [5]

    c) $T(n) = 9T(n/3) + n^2$,

    [5]

    d) $T(n) = T(\sqrt{n}) + 1$.

    [5]

Master Theorem. Let $T(n)$ be the number of operations performed by an algorithm that takes an input of size n. Assume $T(n)$ satisfies, $T(n) = 0$ for $n = 1$, and for $n \geq 2$

$$T(n) = aT(n/b) + O(n^d),$$

where $a > 0, b > 1$ and $d \geq 0$. Then

$$T(n) = \begin{cases} O(n^d), & \text{if } d > \log_b(a) \\ O(n^d \log(n)), & \text{if } d = \log_b(a) \\ O(n^{\log_b(a)}); & \text{if } d < \log_b(a). \end{cases}$$

2. Consider an array containing n elements $a_1, a_2, \ldots, a_n$. The only operation you are allowed is to check whether two elements are equal, i.e. "is a_i equal to a_j?" and these will be the elementary operations.

In what follows, assume that n is a power of 2, i.e. $n = 2^k$, for some positive integer k.

An element x is the *majority* element in $a_1, a_2, \ldots, a_n$ if it occurs more than $n/2$ times in the list $a_1, a_2, \ldots, a_n$.

We will consider two distinct algorithms for finding the majority element in a list.

a) Let us first start with a naive approach. Describe, in words, what the different steps of the following algorithm do and derive the complexity of this algorithm in terms of elementary operations.

> **for** $i = 1, \ldots, n$ **do**
> $c \leftarrow 0$
>   **for** $j = 1, \ldots, n$ **do**
>   If $a_i = a_j$ then $c \leftarrow c + 1$
>   **end for**
>   If $c > n/2$ then *return* "a_i is the majority element"
>   **else** *return* "there is no majority element"
> **end for**

[5]

b) Derive, in words or using pseudocode as in the previous question, a divide-and-conquer recursive algorithm for finding the majority element in $a_1, a_2, \ldots, a_n$.

Hint: You need to distinguish a number of scenarios depending on whether there is a majority element in the first and/or second list of the divide-and-conquer procedure.

[10]

c) Compute the complexity of the algorithm you described in the question 2.b).

[5]

3.   The majority of spell-checking algorithms are based on the idea of measuring some distance, known as the *edit distance*, between two words, which evaluates the number of transformations needed to go from one word to another.

In particular, given two words $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ we are interested in the longest common sub-word between them. For example, a longest common sub-word, between *aabaababaa* and *ababaaabb* is

$$ababaaa$$

as highlighted in the initial words a*a*ba*ababaa* and *ababaa*a*bb*. However, this may not be unique.

a)   How many operations would one need to perform to find the longest sub-word between two words using brute force/exhaustive search?   [2]

b)   We will now derive a dynamic programming algorithm that if given two words $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_m$, returns the length of the longest sub-word between A and B denoted by $p(n, m)$. Let $p(i, j)$ be the length of the longest sub-word between $a_1, a_2, \ldots, a_i$ and $b_1, b_2, \ldots, b_j$, for $i \leq n$ and $j \leq m$.

   i)   if $a_i \neq b_j$, show that either $p(i, j) = p(i - 1, j)$ or $p(i, j) = p(i, j - 1)$.

[3]

   ii)   If $a_i = b_j$, show that either $p(i, j) = p(i - 1, j - 1) + 1$ or $p(i, j) = p(i - 1, j - 1)$

[3]

   iii)   Using the above cases and the notation $\delta(a_i, b_j)$ which is equal to 1 if $a_i = b_j$ and 0 otherwise, write a dynamic programme for computing $p(n, m)$.

[4]

   iv)   Derive the complexity of the above dynamic programme.

[2]

c)   Apply the above procedure to find the longest common sub-word between *abcbdab* and *bdcaba*.

[6]