IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2011

BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
MSc in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER MC202

SOFTWARE ENGINEERING - ALGORITHMS

Friday 6 May 2011, 14:30
Duration: 75 minutes
(Reading time 5 minutes)

Answer ALL TWO questions

Paper contains 2 questions
Calculators required

Section A (Use a separate answer book for this Section)

1a i) Compare and contrast "Big Theta", "Big Omega", and "Big O".

ii) Consider the function A: $2^{3n} + n^2$.

Can the function B: 2^{3n} be used as a "Big Theta", "Big Omega", or a "Big O" characterization of function A? Explain your answer.

b Your Oxbridge friend is trying to sort the suffixes of a string `s` consisting of n ASCII characters, none of which is "`\0`" (the "null" character). The code calls a function `RadixSort`, which sorts an array of `\0`-terminated strings using the radix method.

```
// form the n suffixes, appending a null character
// to the end of each string
String[] suffixes = new String[n];
for (int i = 0; i < n; i++) {
  // extract a substring and concatenate
  // a null character
  suffixes[i] = s.substring(i,n) + "\0";
}

// sort the n strings
RadixSort(suffixes);
```

i) What is a good radix to use in this situation? Explain your answer.

ii) Assume that RadixSort works by creating an optimized TST. Give the pseudocode for printing the keys in alphabetic order.

iii) Unfortunately, when your friend uses this code for large n, it runs extremely slowly, even for non-pathological inputs. Briefly explain the cause of this problem.

iv) Fix the algorithm so that it runs efficiently, and explain your answer.

The two parts carry, respectively, 20%, and 80% of the marks.

Section B (Use a separate answer book for this Section)

2 Fast Fourier Transform

The DFT for the sequence of complex numbers $a_0, \ldots, a_{n-1}$ is defined as the sequence

$$A(1), A(\omega_n), \ldots, A(\omega_n^{n-1})$$

obtained by evaluating the polynomial

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

at each of the nth root of unity $1, \omega_n, \ldots, \omega_n^{n-1}$.

a Describe the Fast Fourier Transform (FFT) algorithm which computes the DFT in $O(n \log n)$ time. You can assume that n is a power of 2. Justify the running time.

Hint: The algorithm works by partitioning $A(x)$ into its even terms and odd terms to form two recursive calls of the FFT. It is useful to know that $\omega_n^k = \omega_{n/2}^{k/2}$ when k and n are both even.

b i) Compute the DFT of (2, 3, 4, 5). Do not calculate the DFT directly, but use the FFT algorithm you devised in part a of the question. Write down all intermediate steps.

ii) Multiply the polynomials $p(x) = 9 + 7x$ and $q(x) = 5 + 4x$ using the FFT algorithm. Write down the intermediate steps.

c Consider the following pattern matching problem: You are given a long text T in binary alphabet, you are also given a shorter string S in binary alphabet and you want to find all the places in T where S appears as a substring. More formally, if $T = t_0, ..., t_n$ and $S = s_0, ..., s_m$ $(m < n)$, you have to output all the is such that for every $0 \leq j \leq m, s_j = t_{i+j}$. Show how this can be done in time $O(n \log n)$.

Hints: Change the alphabet to 1, -1. Look at the reverse of S (i.e. its mirror image). Consider S reverse and T as polynomials.

The three parts carry, respectively, 30%, 45%, and 25% of the marks.