UNIVERSITY OF EDINBURGH

FACULTY OF SCIENCE AND ENGINEERING

COMPUTER SCIENCE THREE

ALGORITHMS AND DATA STRUCTURES

Saturday 27 May 2000

9.30 to 11:00

Examiners:

Prof. K. Turner (External)

Prof. G. Brebner (Chairman)

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

1. (a) Recall that the procedure Randomised-Partition is used as part of the randomised version of the Quicksort algorithm. Like Partition, Randomised-Partition splits the input array into two parts. Denoting the input for the procedure by the sub-array $A[p \dots r]$ the procedure rearranges the elements in A and returns an index q such that $A[p \dots q]$ and $A[q+1 \dots r]$ are both non-empty and such that each element in $A[p \dots q]$ is smaller than or equal to each element in $A[q+1 \dots r]$.

   i. Explain in detail how Randomised-Partition achieves the above. If you find it more convenient you may write pseudo-code for it but this is not required; if you do, you should still explain the general idea. [*5 marks*]

   ii. Denote the size of the low part of the partition by L and the size of the input array by n so that $L = q - p + 1$ and $n = r - p + 1$. Show that if all elements in A are distinct (i.e. there are no repetitions) then for Randomised-Partition

$$\begin{aligned} Pr[L = 1] &= 2/n \\ Pr[L = i] &= 1/n \qquad \text{for } 2 \leq i \leq n-1 \end{aligned}$$

   You may assume in your argument that Partition is correct. [*12 marks*]

(b) Recall that the expected runtime $T(n)$ of Quicksort satisfies the recurrence

$$T(n) = \frac{1}{n}\left(T(1) + T(n-1) + \sum_{q=1}^{n-1}(T(q) + T(n-q))\right) + Q(n)$$

for $n > 1$ where $Q(n) = \Theta(n)$, and that $T(1) = \Theta(1)$. We have seen in lectures that $T(n) = O(n \log n)$. Show that this is not an overestimate, i.e. show that $T(n) = \Omega(n \log n)$.

[**Hints:** Prove by induction that $T(n) \geq cn \log n$ for some $c > 0$. You may assume that $Q(n)$ satisfies $a_1 n \leq Q(n) \leq a_2 n$ for all $n \geq 1$ and for some positive constants a_1, a_2. You may also use without proof the following inequality: $\sum_{k=1}^{n-1} k \log k \geq \frac{1}{2}n^2 \log n - n^2$.] [*8 marks*]

2. Recall that the DFT of vector of n elements $\bar{a} = (a_0, \ldots, a_{n-1})$ is defined as

$$\mathrm{DFT}(\bar{a}) = (y_0, \ldots, y_{n-1})$$

where $y_j = \sum_{k=0}^{n-1} a_k \omega_n^{kj}$, and the inverse DFT of $\bar{y} = (y_0, \ldots, y_{n-1})$ is

$$\mathrm{IDFT}(\bar{y}) = (a_0, \ldots, a_{n-1})$$

where $a_j = \frac{1}{n} \sum_{k=0}^{n-1} y_k \omega_n^{-kj}$, where throughout ω_n is a principal n'th root of unity.

(a) Let $n = 2^k$ for any integer $k > 2$. Write down values for ω_n^0, $\omega_n^{n/4}$, $\omega_n^{n/2}$, ω_n^n. Make sure to be clear about which value corresponds to each of the expressions.

(Note: you do not have to prove that these values are correct, just to specify them. The correct values are among: 1, -1, i, -i, $\sqrt{2}i$, $-\sqrt{2}i$, $\frac{\sqrt{2}}{2}i$, $-\frac{\sqrt{2}}{2}i$) [*4 marks*]

(b) Let $\bar{a} = (a_0, a_1, a_2, a_3) = (5, 1, 3, 2)$. Compute $DFT(\bar{a})$.

(Note: you are *not* required to refer to the Fast Fourier Transform (FFT) algorithm. The question can be answered by evaluating the relevant formulae directly.) [*4 marks*]

(c) Give pseudo-code for an algorithm for multiplying two polynomials of degree $n-1$ in time $O(n \log n)$ and explain briefly why the algorithm is correct. (Note: you are *not* required to prove that the algorithm is correct; just explain the general reasoning.)

You may assume subroutines that compute DFT and IDFT correctly in time $O(n \log n)$ and do not need to describe their details. You may also assume that n is a power of 2. [*8 marks*]

(d) Given a vector $\bar{a} = (a_0, \ldots, a_{n-1})$ the cyclic difference of $\bar{a}$ is

$$\Delta(\bar{a}) = (a_0 - a_{n-1}, a_1 - a_0, \ldots, a_{n-1} - a_{n-2}).$$

Let $\bar{y} = DFT(\bar{a})$. Show that $DFT(\Delta(\bar{a})) = (z_0, \ldots, z_{n-1})$ where $z_j = (1 - \omega_n^j) y_j$. [*9 marks*]

3. For this question let $G = (V, E)$ be a flow network with capacity function c, source vertex s, and sink vertex t.

   (a) A flow in G is a real valued function $f : V \times V \rightarrow R$ which satisfies the three properties known as *capacity constraint*, *skew symmetry*, and *flow conservation*. Define each of these properties. [*3 marks*]

   (b) Explain what is meant by a cut (S, T) of a flow network. [*2 marks*]

   (c) Recall that for subsets X, Y of V we defined the implicit summation notation by $f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$, and similarly for the capacity $c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y)$. Show that for any flow f, and any cut (S, T) in G, $|f| \leq c(S, T)$.

   You may use without proof the fact that for any flow f and any cut (S, T), $|f| = f(S, T)$. [*5 marks*]

   (d) Let f be a flow in G, and let G_f be the residual flow graph of f in G. Recall that the capacity in G_f is $c_f(u, v) = c(u, v) - f(u, v)$.

   Show that if G_f contains no augmenting path then there is a cut (S, T) such that $|f| = c(S, T)$. [*7 marks*]

   (e) A safe-path graph $G = (V, E)$ is a directed graph with a source vertex s and sink vertex t. The graph G is k-safe if there are k vertex disjoint (and hence also edge disjoint) paths from s to t. That is, there are k paths from s to t such that no two paths share a vertex (except for s, t). Such paths can be traversed independently so that if less than k of the paths are blocked at any vertex we can still traverse the graph from s to t.

   Give an efficient algorithm that, on input G, computes the maximal k such that G is k-safe, and argue that it is correct. (Hint: the naive application of flow to the graph will fail since it does not produce vertex disjoint paths but something slightly more complex does work.) [*8 marks*]