Midterm Examination on Algorithms — page 1/2

Teacher: Biing-Feng Wang — April 26, 2001

Problem 1: (10%, Growth of Function)

(1) (4%) What's the definition of Θ?

(2) (6%) Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the definition of Θ-notation, prove that $\max(f(n), g(n))=\Theta(f(n)+g(n))$. $< f(n)+g(n)$

Problem 2: (20%, 5% each, Recurrences)

(1) Find an upper bound on the recurrence $T(n)=T(\lfloor 2n/3 \rfloor)+1$ by using the substitution method. (You may assume that $T(1)=1$.)

(2) Find an upper bound on the recurrence $T(n)=2T(\lfloor n/2 \rfloor)+n \lg n$ by appealing to a recursive tree. (You may assume that $T(1)=1$.)

(3) Describe the *Master Theorem*.

(4) Give an example to which the Master Theorem can not apply. Justify your answer.

Problem 3: (10%, Binary Heap) Let A=(4, 1, 3, 2, 16, 9, 10, 14, 8, 7). If we call *Build-Heap*(A) to make A a min heap, how many *exchange-operations*, each of which exchanges the contents of two elements, will be performed. Explanation is necessary.

Problem 4: (10%, Sorting) Insertion sort is efficient for sorting a small size of data items. We can modify mergesort as follows. When mergesort is called on a subarray with fewer than k elements, where k is an integer$\leq n$, we stop recursive call and run insertion sort on the subarray.

(1) (7%) What's the worst-case time complexity of the above sorting algorithm, in terms of n and k? Justify your answer. $(n-k)(k^2)$

(2) (3%) For what values of k will the above algorithm runs in $O(n\log n)$ time.

Problem 5: (10%, Selection in linear time)

(1) (7%) Let $A[1..n]$ be an array of n real numbers. Give an efficient algorithm to determine whether there is a number in A that appears more than $\lfloor n/4 \rfloor$ times. (Note that it is possible to have an $O(n\log n)$ time algorithm and you can use the $O(n)$ time selection algorithm as a procedure.)

(2) (3%) What's the time complexity of your algorithm? Justify your answer.

$O(n)$

Problem 6: (10%, Greedy algorithms)

(1) (5%) Give an efficient algorithm for the fractional knapsack problem? What's the time complexity of your algorithm? Justify your answer.

6.9 $T(n)=O(n\lg n)$

(2) (5%) Prove that your algorithm is correct.

Problem 7: (20%, Dynamic Programming) The *resource allocation problem* is defined as follows. We are given m resources and n projects. A profit $P(i, j)$ will be obtained if j, $0 \le j \le m$, resources are allocated to project i. The problem is to find an allocation of resources to maximize the total profit. For example, letting $n=4$, $m=3$ and $P(i, j)$ as below, allocating 2, 1, 0, 0 resources, respectively, to project 1, 2, 3, 4 will obtain the maximum profit 13.

	j=0	j=1	j=2	j=3
i=1	0	2	8	9
i=2	0	5	6	7
i=3	0	4	4	4
i=4	0	2	4	5

$P[i, j]$

(a) (5%) Define $X[i, l]$ as the maximum profit obtained by allocating l resources to the first i projects, where $0 \le i \le n$ and $0 \le j \le m$. Give a recurrence of $X[i, j]$, including all boundary conditions.

(b) (5%) Give an algorithm that, given m, n, and $P[i, j]$, compute the maximum profit that can be obtained.

$O(m^2 n)$

(c) (5%) What's the time complexity of your algorithm in (b)? Justify your answer.

(d) (5%) Modify your algorithm in (2) such that an allocation maximizing the profit can be output.

Problem 8: (10%, Design of Algorithms)

Given a k-bit non-negative binary integer $B=b_{k-1}b_{k-2}...b_1b_0$, its 1's complement is defined as the k-bit binary integer obtained by changing bits 1 and 0, respectively, in B into 0 and 1. For example, letting B=10110, the 1's complement of B is 01001. Let $A[1..n]$ be an array of (4log n)-bit non-negative binary integers.

(1) (6%) Given an efficient algorithm to determine for every $A[i]$ whether its 1's complement is also in A. You may assume that computing the 1's complement of an integer and comparing two integers can be done in $O(1)$ time.
(Note that an $o(n\log n)$ time algorithm is possible.)

(2) (4%) What's the time complexity of your algorithm? Justify your answer.

Bonus: (5%, Open address) What are the three techniques commonly used to compute the probe sequence required for open addressing? No explanation is required.

5, 7, 8 => 要 Algorithm

linear