233

Midterm Examination on Algorithms

Teacher: Biing-Feng Wang

Nov. 9, 2016

Remark. For each problem, you must justify your answer. All time complexities are in worst-case, unless otherwise specified. It is suggested that your algorithms are described in words and examples (instead of in pseudo codes), unless pseudo codes are asked to be provided.

Problem 1: (10%, Growth of Functions)

(1) (4%) Use the definition of Ω to show that $n^3/1000 + 50n^2 - 100n + 3 = \Omega(n^3)$.

(2) (6%) Let $f(n)$ be an arbitrary positive non-decreasing function. Prove or disprove $f(n) = \Theta(f(n/3))$.

$S(m) = \lfloor \frac{m}{3} \rfloor S(\lfloor \frac{m}{3} \rfloor) + \log m$

Problem 2: (15%, Recurrences)

(1) (5%) Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = \lfloor \sqrt{n} \rfloor T(\lfloor \sqrt{n} \rfloor) + n$ for $n \geq 2$ and $T(1) = 1$.

(2) (10%) Let $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1$ for $n \geq 2$ and $T(1) = 1$. Prove that $T(n) = O(n)$ by the substitution method

$an - b$

Problem 3: (10%, Priority Queues) Give an implementation of a priority queue S that supports the following three operations efficiently. (Initially, $S = \varnothing$.)

Insert(S, x): Insert an element x into S. $O(\log n)$

Max(S): Return the value of the maximum element in S. $O(1)$

Extract-Max(S): Return and remove the maximum element in S. $O(\log n)$

Describe your implementation in detail. How much time is required for each of the operations? Justify your answer.

$\log n$

build heap $O(n)$ $n \log n$

Problem 4: (10%, Divide-and-Conquer) Explain and compare the following algorithm design strategies: divide-and-conquer, partition, prune-and-search. Also, give an example for each of them.

Merge sort Q.Sort

Problem 5: (10%, Fractional Knapsack Problem)

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

(2) (5%) Prove that your algorithm in (1) is correct.

$f(c, k) = \begin{cases} f(c, k-1) \\ f(c - W_k, k-1) + V_k \end{cases}$

Problem 6: (15%, Dynamic programming) Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n positive integers. Given an integer i, we say that S can *create* i if there is a subset $S' \subseteq S$ such that $\sum_{x \in S'}\{x\} = i$. For example, letting $S = \{5, 1, 3, 7\}$, S can create 12 (= 5+7) but can not create 2.

(1) (5%) Define a function $C(i, j)$ as follows:

C(i j)

$$C(i, j) = \begin{cases} true & \text{if } \{s_1, s_2, s_3, \ldots, s_i\} \text{ can create } j; \\ false & \text{otherwise,} \end{cases}$$

where $0 \le i \le n$ and $0 \le j \le (s_1 + s_2 + \ldots + s_n)$. Describe a recurrence of $C(i, j)$, including boundary conditions.

(2) (4%) Give an efficient algorithm that, given S and an arbitrary positive integer k, determine whether S can create k or not.

(3) (2%) What is the time complexity of your algorithm in (2)?

(4) (4%) Modify your algorithm in (2) such that in case that S can create k, a subset $S' \subseteq S$ with $\sum_{x \in S'}\{x\} = k$ is returned.

Problem 7: (10%, Design of Algorithms)

(1) (7%) Let x be an integer and $S = (s_1, s_2, \ldots, s_n)$ be a sequence of n integers ranging from 0 to $6n^{\lg\lg n}$. Design an $o(n \lg n)$-time algorithm that determines whether or not there exist two integers s_i and s_j, where $i \ne j$, in S whose sum is exactly x. Describe your algorithm in detail.

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

Problem 8: (10%, design of algorithms) Describe a linear-time algorithm that, given a set S of n distinct real numbers, a number x, and a positive integer $k \le n$, finds the k numbers in S that are closest to x. (These k numbers can be outputted in any order.) For example, if $S = (1, 4, 6, 8, 7, 2)$, $x = 5$, and $k = 3$, the output is $\{4, 6, 7\}$. Describe your algorithm in details. Briefly explain why your algorithm runs in linear time.

Problem 9: (10%, Homework) This problem is to verify whether you had done homework by yourself. Please answer either of the following. (If you answer both, only the one getting less score will be counted.)

(a) Give a linear-time algorithm for the maximum-subarray problem. Show that your algorithm does run in linear time.

(b) Show that QUICKSORT can be modified so that the worst-case stack depth is $O(\lg n)$. Show that your algorithm does require $O(\lg n)$ stack space.

Bonus: (10%, self-taught) What are the three techniques commonly used to compute the probe sequence required for open addressing? Describe the techniques briefly.