Technische Universität München
Department of Computer Science
Chair for Efficient Algorithms
Prof. Dr. Sami Khuri

WS 1999/2000
Exam
February 22, 2000

Efficient Algorithms and Data Structures

Time allowed: 120 minutes

Exam rule: You can use one sheet of paper (A4) with arbitrary contents and a normal calculator (not programmable), nothing else!

Passing criterion: You must get more than 16 out of 40 credits to pass the exam.

Please do not write with a red or green pen!

Problem 1 (5 credits)

Which of the following claims are true?

a) $n^2 \log n$ is $O(n^3)$
b) $11 \log n$ is $\Omega(n)$
c) $n^2 + 10^{100} n$ is $O(n^2)$
d) 2^{n+1} is $O(2^n)$
e) 2^{2n} is $O(2^n)$
f) $\log n$ is $\Omega(\log(n^5))$

Give a brief justification for your answers.

Problem 2 (2+3+3 credits)

a) Solve the recurrence: $T(n) = 2 \cdot T(n-1) + n$
$T(1) = 1$

b) Use **generating functions** to solve the recurrence:

$$\begin{aligned} a_n &= 5a_{n-1} - 4a_{n-2}, \text{ for } n \geq 2 \\ a_0 &= 0 \\ a_1 &= 1 \end{aligned}$$

c) Solve the recurrence: $T(n) = n + \frac{1}{n} \sum_{i=1}^{n} [T(i-1) + T(n-i)]$
$T(0) = 0$

Give the solution in terms of H_n, the n^{th} Harmonic number. What is the growth rate of $T(n)$?

Problem 3 (7 credits)

Consider a priority queue that is implemented as a binomial heap. Initially, the binomial heap is empty. Then we insert elements with keys $1, 2, 3, \ldots, n$ (in that order). Draw a picture of the resulting binomial heap for $n = 11$. For arbitrary n, what is the growth rate of the total running-time for these n operations? (Use the Θ-notation!)

Problem 4 (7 credits)

The purpose of a union-find data structure is to maintain a partition of a set of n elements into disjoint subsets. It supports the operation $\textsc{Find}(x)$ that returns a representative element (canonical element) from the set that contains x, and the operation $\textsc{Union}(S_1, S_2)$ that combines the sets S_1 and S_2 to form a new set.

We study the following implementation of a union-find data structure: Each of the sets is represented by a linear list, where each element has an additional *back pointer* that points to the element at the beginning of the list (which serves as the canonical element of the set). This allows operation $\textsc{Find}(x)$ to be executed in time $O(1)$: it is sufficient to return the back pointer of x.

To perform $\textsc{Union}(S_1, S_2)$, the shorter of the two lists is appended to the end of the longer list, and the back pointers of the elements in the shorter list are updated. Each pointer update takes time $O(1)$, so the running-time for the operation $\textsc{Union}(S_1, S_2)$ is $O(\min\{|S_1|, |S_2|\})$.

Beginning with n single-element sets, we perform $n-1$ $\textsc{Union}$ operations until we have one set containing all n elements. Show that the total running-time for these $n-1$ operations is $O(n \log n)$.

Problem 5 (7 credits)

For two given sequences $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ of characters from some alphabet, a sequence $C = (c_1, \ldots, c_k)$ is a *common supersequence* if A and B are both subsequences of C. For example, MODULARITY is a common supersequence of OLATY and MULRY. Give a dynamic programming algorithm that finds a shortest common supersequence of two given sequences A and B. Analyze the running-time of your algorithm. Use the algorithm to compute a shortest common supersequence of OLATY and MULRY.

Problem 6 (6 credits)

Consider a connected undirected graph G with positive edge weights. For a spanning tree T of G, let *heaviest*(T) be the weight of the edge in T with the largest weight. Give an efficient algorithm that computes a spanning tree T of G such that *heaviest*(T) is minimal among all spanning trees of G. In other words, for any spanning tree T' of G, we want to have *heaviest*$(T) \leq$ *heaviest*(T'). Justify the correctness of your algorithm.