Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn

Exam Algorithm Theory

Friday, August 30 2013, 9:00 – 10:30

Name: . . . . . .

Matriculation Nr.: . . . . . .

Signature.: . . . . . .

Do not open or turn until told so by the supervisor!

Instructions:

- Write your name and matriculation number on the cover page of the exam and sign the document! Write your name on all sheets!
- Your signature confirms that you have answered all exam questions without any help, and that you have notified exam supervision of any interference.
- Write legibly and only use a pen (ink or ball point). Do **not** use **red**! Do **not** use a pencil!
- You are **not** allowed to use any material except for a dictionary, a pocket calculator, and a self-written summary of at most 5 A4 pages (corresponds to 5 single-sided A4 sheets!).
- There are 3 problems (with several questions per problem) and there is a total of 90 points. The number of points for each question is given.
- Use a separate sheet of paper for each of the 3 problems.
- Only one solution per question is graded! Make sure to strike out any solutions that you do not want to be considered!
- **Explain your solutions! Just writing down the end result is not sufficient.**

Question	Achieved Points	Max Points
1		42
2		29
3		19
Total		90

Problem 1: Short Questions **42 points**

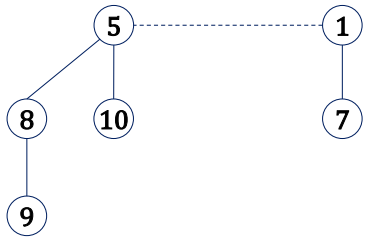
(a) **(9 points)** Consider a divide-and-conquer algorithm where solving a problem of size n takes time $T(n)$ such that

$$T(n) = 3 \cdot T\left(\frac{n}{3}\right) + \Theta(n \log n), \quad T(1) = 1.$$

What is $T(n)$ asymptotically?

(b) **(6 points)** In the lecture, we have seen the FFT algorithm to multiply two polynomials. A similar problem occurs when two n-bit integers (for arbitrary n) have to be multiplied. The single bits (or blocks of, e.g., 16 bits) can be considered as the coefficients of a polynomial. Why does the FFT algorithm as discussed in class **not** immediately give an $O(n \log n)$ time algorithm for multiplying two n-bit integers?

(c) **(6 points)** Consider the following binomial heap. How does the binomial heap look after inserting values 3, 6, and 12 (in that order)? How does it look after a subsequent *delete-min* operation?



(d) **(9 points)** We are given a data structure $\mathcal{D}$, which supports the operations `put` and `flush`. The operation `put` stores a data item in $\mathcal{D}$ and has a running time of 1. Further, if $\mathcal{D}$ contains $k \geq 0$ items, the operation `flush` deletes $\lceil k/2 \rceil$ of the k data items stored in $\mathcal{D}$ and its running time is equal to k.

Prove that both operations have constant amortized running time by using the potential function method.

(e) **(12 points)** In the lecture, we have seen a randomized primality test which for a number N, tests whether N is a prime. If N is a prime, the test always returns "yes," if N is not prime, the test returns "no" with probability at least $3/4$. The running time of the test is $O(\log^2 N \cdot \log\log N \cdot \log\log\log N)$. Your task now is to find an efficient randomized algorithm that for a given (sufficiently large) input number n, finds a prime number p between n and $2n$. Your algorithm should succeed with probability at least $1 - 1/n$. You can assume that the number of primes between n and $2n$ is at least $\frac{n}{3 \ln n}$. What is the running time of your algorithm?

Problem 2: Stacking Boxes **29 points**

We are given a set of n rectangular 3-D boxes, where the i^{th} box has length ℓ_i, width w_i, and height h_i, and where all three dimensions are *positive integers*. We further assume that for each box i, $\ell_i \geq w_i$. The objective is to create a stack of boxes which has *a total volume that is as large as possible*, but one is only allowed to stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Hence, it is only possible to put box j on top of box i if $\ell_i > \ell_j$ and $w_i > w_j$. See Figure 1 as an illustration.

(a) **(6 points)** As a first attempt, we might try to use a greedy algorithm. Because when stacking boxes on top of each other, the lengths have to be strictly decreasing, a natural strategy might be to first sort the boxes by decreasing length such that $\ell_1 \geq \ell_2 \geq \ldots \geq \ell_n$. Then, we go through the boxes in that order and add a box on the existing stack whenever possible. Show that the behavior of this greedy algorithm can be arbitrarily bad!

(b) **(9 points)** Instead of sorting the boxes by length, we could alternatively sort by decreasing area of the 2-D base. That is, we sort the boxes such that $\ell_1 w_1 \geq \ell_2 w_2 \geq \ldots \geq \ell_n w_n$. Show that now, box j can only be placed on top of box i if $j > i$ and therefore $\ell_j w_j \leq \ell_i w_i$. We again consider the greedy algorithm, but this time we go through the boxes according to the new ordering where the boxes are sorted by decreasing base area. We further simplify and assume that *all boxes have height* $h_i = 1$. Show that if there is a stack of volume V, the greedy algorithm finds a stack of volume at least $\Omega(V^{2/3})$.

(c) **(14 points)** As a greedy algorithm does not seem to lead to a solution of the problem, let us finally try another approach. We again consider the general case where the heights of the boxes are arbitrary integers. Give an efficient algorithm that solves the box stacking problem by using dynamic programming. (The algorithm should return a maximum volume stack of boxes.) What is the running time of your algorithm? (as a function of n)

Hint: *It might still make sense to go through the boxes in order of decreasing base area.*

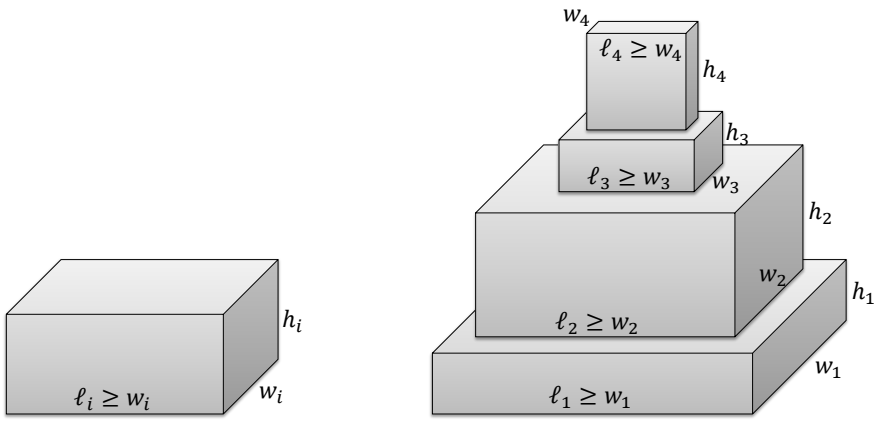


Figure 1: A single box of length ℓ_i, width w_i, and height h_i, as well as a feasible stack of boxes.

Problem 3: Network Flows **19 points**

(a) **(9 points)** In this problem, we are given a flow network with unit-capacity edges: It consists of a directed graph $G = (V, E)$, a source node $s \in V$, a sink node $t \in V$, and capacity $c_e = 1$ for every $e \in E$. We are also given a positive integer parameter k.

The goal is to delete k edges so as to reduce the maximum s-t flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum s-t-flow in $G' = (V, E \setminus F)$ is as small as possible.

Give a polynomial-time algorithm to solve this problem.

(b) **(10 points)** Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e, a source node $s \in V$, and a sink node $t \in V$. You are also given a maximum s-t flow f in G, defined by a flow value f_e on each edge e. The flow f is *acyclic*: There is no directed cycle in G on which all edges carry positive flow. The flow f is also integer-valued.

Now suppose, we pick a specific edge $e^* \in E$ and reduce its capacity by 1 unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$, where m is the number of edges in G and n is the number of nodes.