Umeå University
Department of Computing Science
5DV022 — Analysis of Algorithms
Examination: December 19, 2007

Name (printed): _____

Swedish ID number: _____

Computer User-ID: _____

Signature: _____

1. All answers must be written in English.
2. A pocket engineering calculator (miniräknare) and English/X - X/English dictionary may be used. No other help materials are allowed.
3. You must turn in this cover sheet together with the pages containing your answers. Collate the sheets in numerical order of the problems, and write on only one side of the paper.
4. The examination has a total of 400 points, which constitutes 40% of the course grade. The other 60% is based upon the obligatory exercises and problem sets, as explained in the course syllabus.
5. As noted in the course syllabus, it is necessary to obtain at least 100 points on the examination in order to receive a passing grade, regardless of the number of points obtained on the laboratory exercises and problem assignments.
6. In many of the problem with multiple parts, the answer to later parts depends upon the answer to earlier parts. If the answer to an earlier part is wrong, and if that error makes the answer to a later part much easier or trivial, then full credit will not be awarded for that later part even if the answer is correct relative to the previous error.
7. This examination consists of eight (8) problems on a total of three (3) pages.
8. You have the option of skipping a problem for partial credit. In the table below, place an X in the position for any problem for which you have attempted a solution, and which you wish to have graded. It is extremely important that you fill in this table properly, because of the following option. For any box which is left blank, the associated question will not be graded, and you will instead be awarded 12% of the points for that question. Your decision to leave a box blank is definitive, so be very careful. For example, If you leave box 5 blank, your answer to that question will not be graded, even if it is completely correct. On the other hand, if you place an X in box 5, but provide no answer whatsoever to that question, you will not receive 12% of the points for that question. It is strongly recommended that you use a pencil, in case you change your mind! Note that it is not possible to opt in or out of parts of problems; only entire problems may be skipped for partial credit.

Prob	1	2	3	4	5	6	7	8

(1: 50 points total) Two distinct notions of asymptotic complexity were defined in the course notes and lectures, each recapturing the notion that function f is at least as complex as function g. They were represented as $f \in \Omega_1(g)$ and $f \in \Omega_2(g)$. Define each of Ω_1 and Ω_2, and briefly discuss the advantages and disadvantages of each. (It is *not* necessary that you remember which was called Ω_1 and which was called Ω_2. Just define and compare the two notions.) For full credit, your answer must include formal mathematical definitions, as well as less formal explanations of their respective meanings. Illustrate each with a simple example.

(2: 50 points total) The worst-case complexity of an algorithm is characterized by the following recurrence relation

$$T(n) = 9 \cdot T(n/3) + 2 \cdot n$$

with initial condition $T(1) = 1$.

(a) Using a change of variable, obtain a linear recurrence relation.

(b) Give the characteristic polynomial of this linear recurrence.

(c) Solve for the roots of this polynomial.

(d) Give the solution to this linear recurrence.

(e) Translate this solution back to the original variable. For full credit, this answer must be in a simplified form.

(f) Give a tight bound on the worst-case complexity of the algorithm based upon this solution, in $\Theta(-)$ format.

(3: 50 points total) Answer the following questions. For (a)-(c), you are to give **formal** definitions, not "intuitive" ones. Furthermore, for parts (a) and (b), illustrate the concepts using the discrete knapsack problem.

(a) Define *subset system* and *profit function*.

(b) Define *combinatorial optimization problem*, *feasible solution*, and *optimal solution*, with respect to your answer of (a).

(c) Define the *formal greedy algorithm* with respect to a combinatorial optimization problem.

(d) Using the notions of (a)-(c), define the concept of *matroid*.

(e) Give two further conditions which are equivalent to a subset system being a matroid. There is no need to prove the equivalence; simply state the conditions.

(4: 50 points total) Consider the following variation on the job scheduling problem. Each job job_j has not only a deadline $d(\mathsf{job}_j)$, but also a *startline* $s(\mathsf{job}_j)$. To be scheduled legally, a job must not only be completed before its deadline, but it must also be started at or after its startline. It is assumed that $0 \leq s(\mathsf{job}_j) < d(\mathsf{job}_j)$ for each job job_j. Each job has unit running time and an associated profit $p(\mathsf{job}_j)$, as before. (Note that in the special case that each $s(\mathsf{job}_j)$ is 0, the simple scheduling problem identified in the course notes is obtained.) Legal and optimal solutions are defined in the obvious way. Prove or disprove: this problem defines a matroid.

Note: A statement such as "It is a matroid" or "It is not a matroid", without a correct proof, will not receive credit.

(5: 50 points total)

(a) Give a concise but clear definition of the abstract data type *priority queue*. The following information must be provided.

(i) The underlying type on which it operates.

(ii) Each of its operations, including the domain and range (*i.e.*, input and output types), as well as a clear description of what the operation does.

Do not describe implementation here in part (a); that issue is addressed in (b) below.

(b) Describe the data structure *min-heap*, including:

(i) how it is implemented using arrays; and

(i) how the operations of a priority queue are represented within it.

(c) State (proof is not required) the *worst case* and *average case* time complexity for each of the operations of a priority queue which is implemented as a min-heap as described in (b) above.

(d) State what is meant by an *adjustable priority queue*, emphasizing how it extends the notions defined in part (a). In particular, identify the operations which it adds to an ordinary priority queue.

Note: For part (b) above, the implementation must be the one presented in the book and course notes, or else an implementation for which all operations have a worst-case and average case time complexity which is no worse than for that of the implementation presented in the course.

(6: 50 points total) Consider the problem of evaluating a polynomial of degree n

$$\sum_{i=0}^{n} a_i \cdot x^i = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + a_{n-2} \cdot x^{n-2} \ \ldots \ a_i \cdot x_i \ \ldots \ a_2 \cdot x^2 + a_1 \cdot x + a_0$$

at some value $x = b$.

(a) Give high-level pseudocode for an algorithm which solves this problem in worst-case time $O(n)$.

(b) Illustrate the operation of this algorithm on the polynomial

$$8 \cdot x^4 + 7 \cdot x^2 + 6 \cdot x + 3$$

evaluated at $x = 3$.

(7: 50 points) Answer the following questions in the context of the 0/1-knapsack optimization problem.

(a) Define what is meant by an ε-*relative* approximation algorithm.

(b) Sketch a $1/(k+1)$-relative approximation for the 0/1-knapsack optimization problem which has worst-case time complexity $\Theta(n/(k+1))$. Here n is the problem size, while k is a parameter of the algorithm which you should define carefully. No proofs of complexity or correctness are required; just sketch the algorithm.

(c) Define what is meant by a *fully polynomial-time* approximation scheme.

(d) Explain clearly why the algorithm identified in (b) is not fully polynomial time.

(8: 50 points) Describe an experimental procedure to determine the precise granularity of the interval timer on a Unix/Linux system. It is not necessary to recall or use the exact syntax of the `getitimer` function or the `itimerval` record structure, or to write detailed C code, but your description must be complete and unambiguous enough that a skilled C programmer with access to that documentation for that syntax could construct the corresponding program without difficulty. In other words, you must provide a high-level description which shows convincingly that you understand how to solve this problem.