Umeå University
Department of Computing Science
TDBC91 — Analysis of Algorithms
Examination: January 10, 2004

Name and ID number: ______________________________

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 for Problems 1–7. 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. 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 15% 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 15% 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

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 part even if the answer is correct relative to the previous error.

(1: 60 points total) The behavior of an algorithm is governed by the following recurrence relation, with initial conditions $T(1) = 2$ and $T(2) = 7$.

$$T(n) = 3 \cdot T(n/2) - 2 \cdot T(n/4) + n \cdot \log_2(n)$$

(a) Using a change of variable, transform this into a linear (inhomogeneous) recurrence.

(b) Give the characteristic polynomial for the answer to part (a).

(c) Find the roots of the polynomial obtained in (b).

(d) Give the solution to this linear recurrence. This answer may contain constants to be determined later.

(e) Translate the answer of (d) back to the original problem.

(f) Write down a system of linear equations which must be solved to find the "constants to be determined later" in the translated answer of part (e). It is not necessary to find these constants; however, all strictly numerical multiplications must be carried out, so that the answer is a simple linear system of n equations in n variables, for some n. In other words, your answer should look like

$$\begin{array}{ccccccccc} a_1 & = & b_{11} \cdot c_1 & + & b_{12} \cdot c_2 & + & \cdots & + & b_{1n} \cdot c_n \\ a_2 & = & b_{21} \cdot c_2 & + & b_{22} \cdot c_2 & + & \cdots & + & b_{2n} \cdot c_n \\ \vdots & = & \vdots & + & \vdots & + & \vdots & + & \vdots \\ a_n & = & b_{n1} \cdot c_n & + & b_{n2} \cdot c_n & + & \cdots & + & b_{nn} \cdot c_n \end{array}$$

with the a_i's and b_{ij}'s *numbers* (not formulas) and the c_i's variables representing the constants to be determined.

Note: The comments in item 6 of the cover page are particularly relevant here. Answers to parts of this question which are based upon incorrect answers to earlier parts may not receive much credit, particularly if the earlier answers result in a much simpler problems for the later parts.

(2: 60 points total) Matroids were used to formalize the notion of a greedy solution to a combinatorial optimization problem. Answer the following fundamental questions on this topic. For (a)-(c), you are to give **formal** definitions, not "intuitive" ones.

(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 *matroid*.

(d) State the fundamental result characterizing matroids in terms of the comparison of two independent sets of differing sizes. (Just state it; do not prove it.)

(e) State the fundamental result characterizing matroids in terms of maximal independent subsets of a set A. (Just state it; do not prove it.)

(3: 40 points) Consider the classical discrete knapsack problem, as studied in the course. More precisely, given is a knapsack with weight capacity M, and n objects $\{\mathsf{obj}_1, \mathsf{obj}_2, \ldots, \mathsf{obj}_n\}$, each with a weight w_i and a value v_i. In this case, M, the w_i's, and the v_i's are all taken to be positive integers. The goal is to find a vector $(x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ such that $\sum_{i=1}^{n} x_i \cdot \mathsf{v}_i$ is a maximum, subject to the constraint that $\sum_{i=1}^{n} x_i \cdot \mathsf{w}_i \leq M$. It is also assumed that $w_i \leq M$ for each object obj_i.

In the dynamic-programming solution to this problem, a sequence $S_0, S_1, S_2, \ldots, S_{n-1}$ of sets of pairs of the form (p,w) is computed, with each such $(p,w) \in S_i$ representing the profit and weight of a partial solution for the objects $\{\mathsf{obj}_1, \mathsf{obj}_2, \ldots, \mathsf{obj}_i\}$. A key feature of this algorithm is the *purging* of suboptimal pairs. Relative to this feature, answer the following questions.

(a) Under the assumption that one is interested in finding only a single optimal solution, describe precisely the conditions under which a pair (p,w) may be purged from S_i.

(b) Repeat part (a), this time assuming that one is interested in finding *all* optimal solutions.

Note: The answers to (a) and (b) must describe strategies which purge as many elements as possible, consistent with the problem. Answers which retain pairs which should be purged will not be considered correct.

(4: 70 points) In this question, the discrete knapsack problem described in question 3 is again considered, this time in the context of using branch-and-bound search to find a solution.

(a) Give a precise definition of the true value $c(x)$ of a vertex x in the search tree. Your definition must cover all vertices, leaf and interior, including vertices which define inadmissible solutions.

(b) State the formal properties which an approximation function $\hat{c}$ must have to be *admissible* relative to c.
Note: This is the condition which is easily computed and often used in practice; it is not the *ideal* condition, usually unrealizable in practice, which guarantees that c and $\hat{c}$ behave identically.

(c) Describe a specific choice for $\hat{c}$ which works well in the context of this problem.

(d) State the general property which a *vertex-killing function* u should have within this context.

(e) Describe a specific choice for u which works well in the context of this problem.

(f) Describe how the search process proceeds for this problem under branch-and-bound, with particular emphasis upon how the functions $\hat{c}$ and u are used to limit the search space.

Note: In parts (c) and (e), the expected answers for $\hat{c}$ and u are those which are used in the course notes and textbook. if you give different choices, then they must be comparable in its quality to limit the search space as the one used in the course presentation. In other words, it is not considered correct to give choices for $\hat{c}$ and u which, while leading to a correct solution, do relatively little to limit the search.

(5: 50 points total) Decision trees are a useful tool for establishing lower bounds over all algorithms which solve a particular problem. Answer the following questions.

(a) Draw the decision tree for the three element set $\{a, b, c\}$ with reference list $A = \langle a, b, c \rangle$. (That is, assume that the list to be sorted is A, and that the decision tree is to be used to determine the sorted version.) Make sure that the decisions (branches of the tree) as well as the nodes are labelled clearly.

(b) Show how a decision tree may be used to establish that the worst-case time complexity for a comparison-based sorting algorithm can be no better than $\Theta(n \cdot \log(n))$, with n the number of elements in the list to be sorted.

(c) Show how a decision tree may be used to establish that the average-case time complexity for a comparison-based sorting algorithm can be no better than is $\Theta(n \cdot \log(n))$, with n the number of elements in the list to be sorted. (A reasonably comprehensive proof is required here.)

Note: In principle, part (c) subsumes part (b), so if you are sure of your answer to (c), you may say "see the answer to part (c)" as your answer to (b). However, part (b) is much easier than part (c), so in case you are unsure of your answer to (c), or if you skip (c), then provide an independent answer to part (b).

(6: 70 points total) Answer the following questions in the context of the study of NP-completeness. In all cases, give precise, formal answers. Purely verbal answers which do not elucidate the underlying mathematical concepts will receive relatively little credit.

(a) Define the concept of an *abstract problem with length*.

(b) Define the concept *encoding scheme* for an abstract problem with length.

(c) State the condition which is necessary for an encoding scheme to be *polynomial*.

(d) Give a definition of the notion of a *structured string*.

(e) Define the concept of a *reasonable encoding* of an abstract problem.

(f) It is clear that the dynamic-programming solution to the discrete knapsack problem (described in question 3) has worst-case time complexity $O(M \cdot n)$. Yet, it is well known that this problem is NP-complete. Using the ideas developed in (a)-(e) above, explain why this does not prove that $P = NP$.

(g) There is a name for the type of NP-complete problem identified in part (f). Give that name.

(7: 50 points total) In experimental profiling of algorithm performance, two of the principal methods for clock-based timing are *fixed position timing*, as represented by the software tools developed in the first software exercise of the course, and *random-sample logging*, as represented by the `gprof` package. Describe each of these approaches, highlighting in particular their differences. In addition, identify the principal advantages and disadvantages of each approach.