Umeå University
Department of Computing Science
5DV022 — Analysis of Algorithms
Examination: October 30, 2008

Name (printed): _____

Swedish ID number: _____

Computer User-ID: _____

Signature: _____

Secret code number: _____

Instructions: / Instructioner:

This examination will be graded anonymously. This page will be removed before the instructor receives the examination for grading. The secret code number given above must therefore be written on every answer page which you turn in to the examination proctor.

Denna skrivning rättas kodad. Detta blad kommer att avskiljas innan läraren får skrivningen för rättning. Ovanstående kod måste därför finnas på samtliga svarsblad när du lämnar skrivningen till skrivvakten.

To the proctor of the examination: / Till skrivningsbevakaren:

Detach this cover sheet from the examination and put it in the envelope which is addressed to Yvonne Löwstedt, Department of Computing Science.

Avskilj detta försättsblad och stoppa i kuvert som skickas till Yvonne Löwstedt, Dataveten-skap.

Umeå University
Department of Computing Science
5DV022 — Analysis of Algorithms
Examination: October 30, 2008

Secret code number: _____

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. Write only your secret code number on these pages; do not write your name or ID.
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) In this problem, a generalization of the job scheduling problem is considered in which the deadline of a job is replaced by a finite set of admissible slots. Specifically, a *job* job_j has associated with it a *profit* p_j and a finite set $A_j = \{t_{j_1}, \ldots, t_{n_j}\}$ of positive integers, called the set of *admissible slots* for job_j. All jobs have unit running times. The idea is that job_j may only run in the time slots identified by the elements of A_j. Specifically, given a set $\mathbf{J}$ of jobs, a legal *schedule* for a subset I of $\mathbf{J}$ is a function $\sigma : I \to \mathbb{N}^{>0}$ (with $\mathbb{N}^{>0}$ denoting the positive integers) subject to the following constraints.

(i) For any $\mathsf{job}_j \in I$, $\sigma(\mathsf{job}_j) \in A_j$. (Each job is scheduled in one of its admissible slot.)

(ii) For any job_{j_1} and job_{j_2} in I, $\sigma(\mathsf{job}_{j_1}) \neq \sigma(\mathsf{job}_{j_2})$. (Jobs do not overlap.)

An *optimal* schedule is a legal schedule of maximal profit, where the profit of the schedule is the sum of the profits of its constituents. (Notice that the job scheduling problem discussed in the course notes and the text is a special case of this, in $A_j = \{1, \ldots, d_j\}$, with d_j the deadline of job_j.)

Prove or disprove: The subset system associated with this problem is a matroid. In other words, prove or disprove that the greedy algorithm provides an optimal solution to this job scheduling problem.

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

(4: 50 points total)

(a) Give a concise but clear definition of the abstract data type *partition-union*. 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.

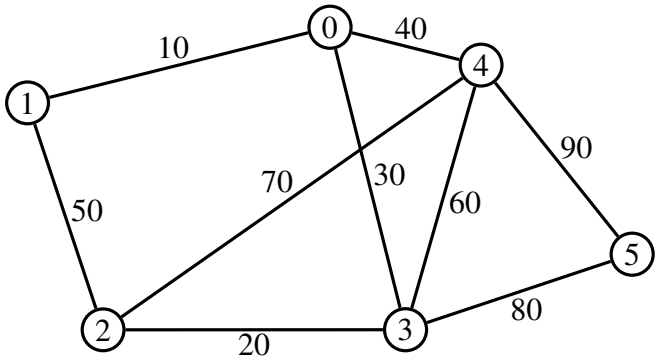
(b) Describe an implementation of this data type using rooted trees. In particular, make sure that the implementation of the operations is included. Illustrate with examples.

(c) State (a proof is not required) the *worst-case* amortized complexity for the operations of a partition-union data type which is implemented as described in the answer to (b) above.

(d) Explain briefly how a partition-union type may be used as a core data structure for solving the job-scheduling problem with unit deadlines. Also state (a proof is not required) the worst-case complexity of the algorithm using this implementation.

Note regarding this problem: It is expected that the descriptions presented in the solutions of parts (b) and (d) are those of the text and course notes. If alternate implementations are described, they must have time and space complexities for each of their operations which are no worse no worse than those of the implementations presented in the course. In other words, credit cannot be obtained for describing inferior implementations.

(5: 50 points) Depicted below is an undirected graph with weighted edges.



(a) Show the sequence of edges which are considered, and which are selected, in applying Kruskal's algorithm for identifying a minimum spanning tree. Thus, you must show the edge considered, and whether or not it is selected, at each step. Represent each edge in your solution by the numerical labels on the vertices which are its endpoints.

(b) Repeat part (a) for Prim's algorithm.

(c) Kruskal's algorithm is a *formal greedy algorithm*, while Prim's is not. Explain clearly what is meant by a formal greedy algorithm, and why this definition holds for Kruskal's but not for Prim's approach. State further the advantage of knowing that an algorithm is formal greedy.

(6: 50 points total) Consider a discrete knapsack problem P with n objects $\{\mathsf{obj}_1, \mathsf{obj}_2, \ldots, \mathsf{obj}_n\}$, each object obj_i possessing a weight w_i and a value v_i. The capacity of the knapsack is M. Answer the following questions in the context of solving such a problem using the branch-and-bound technique studied in the course.

(In the following, a *partial solution* of degree i with $1 \leq i \leq n$ is a vector $\langle x_1, x_2, \ldots, x_i \rangle \in \{0,1\}^i$ which indicates whether or not each of the first i objects is included or not. A *total solution* is a partial solution of degree n.)

(a) Given a partial solution $\langle x_1, x_2, \ldots, x_i \rangle$ of P, a solution to a (not necessarily discrete) knapsack problem Q on $\{\mathsf{obj}_{i+1}, \ldots, \mathsf{obj}_n\}$ is used to obtain a **lower** bound on the best total solution to P which extends $\langle x_1, x_2, \ldots, x_i \rangle$. Describe this problem Q clearly.

(b) Given a partial solution $\langle x_1, x_2, \ldots, x_i \rangle$ of P, a solution to a (not necessarily discrete) knapsack problem Q' on $\{\mathsf{obj}_{i+1}, \ldots, \mathsf{obj}_n\}$ is used to obtain an **upper** bound on the best total solution to P which extends $\langle x_1, x_2, \ldots, x_i \rangle$. Describe this problem Q' clearly.

(c) Explain clearly how the solutions to the knapsack problems Q and Q', which are identified in parts (a) and (b) respectively, are used to obtain the solution to the total knapsack problem on $\{\mathsf{obj}_1, \mathsf{obj}_2, \ldots, \mathsf{obj}_n\}$. In answering this part, it is appropriate to provide some details on how the branch-and-bound strategy operates.

Note: For parts (a) and (b) above, the special problems Q and Q' must be those which were presented in the book and course notes, or else ones which provides bounds which are at least as good as those which were studied in the course.

(7: 50 points total) Consider the following statement:

No sorting algorithm can have average-case complexity which is better than $O(n \cdot \log(n))$.

Sketch an argument which shows this to be the case. Make sure that you state the assumptions under which this argument holds.

(8: 50 points total) Recall that a logical formula (in propositional logic) is in *disjunctive normal form* (DNF) if it is of the following form

$$(\ell_{11} \wedge \ell_{12} \wedge \ldots \wedge \ell_{1k_1}) \vee (\ell_{21} \wedge \ell_{22} \wedge \ldots \wedge \ell_{2k_2}) \vee \ldots \vee (\ell_{n1} \wedge \ell_{n2} \wedge \ldots \wedge \ell_{nk_n})$$

with each ℓ_{ij} a literal (either an atomic proposition or the negation of an atomic proposition).

(a) Show that the satisfiability problem for a logical formula in DNF can be solved in time linear in the number of literals in that formula. (Sketch an algorithm.)

(b) It is known that the satisfiability problem for formulas in propositional logic is NP-complete. Explain briefly what this implies about the complexity of any algorithm which converts an arbitrary logical formula to an equivalent one in DNF.