Umeå University
Department of Computing Science
TDBC91 — Analysis of Algorithms
Examination: October 28, 2002

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 with your solutions. Collate the answer 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) Suppose that the sorting algorithm *mergesort* is implemented in a particularly poor way, so that the time required to merge a list of size n_1 with a list of size n_2 is

$$10 \cdot (n_1 + n_2) \cdot \log_2(n_1 + n_2)$$

instead of the usual $k \cdot (n_1 + n_2)$ for some constant k.

(a) Write down the recurrence relation which describes the time complexity of this modified mergesort algorithm. It may be assumed that the number of elements in the list to be sorted is a power of two.

(b) Transform the answer of (a) to a linear (inhomogeneous) recurrence.

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

(d) Solve the recurrence of (c), up to constant multiples. Specific numerical values of the constants need not be found.

(e) Translate the answer of (d) back to the original problem, thus providing the detailed complexity of this sorting algorithm.

(f) Give the complexity of this algorithm in $\Theta(-)$ notation, based upon the result of (e).

<u>Note</u>: 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: 50 points total) Consider the problem of searching a rooted, acyclic, directed graph using best-first branch-and-bound search. Each leaf node is labelled with a nonnegative integer, and the goal is to find a leaf node with maximum value. The following definitions are made.

(d1) For each leaf vertex x of the graph, $c(x)$ denotes the value of that vertex.

(d2) For each non-leaf vertex x of the graph,

$$c(x) = \max(\{c(y) \mid y \text{ is a leaf vertex under } x\})$$

(a) Let $\hat{c}$ be an approximation of c. State necessary and sufficient conditions on $\hat{c}$ such that under best-first branch-and-bound search, the first leaf node visited is a greatest-cost leaf.

(b) Approximations which satisfy the conditions of (a) are typically not practical. State a simpler but more useful condition on $\hat{c}$ which guarantees that an optimal solution will be found using best-first branch-and-bound search. State both the conditions relating c and $\hat{c}$ and the criteria which guarantee that a leaf node which is found in the search is optimal. If the answer includes specific search terminology (*e.g.*, *E-node*), these concepts must be defined clearly as well.

(3: 60 points total) Consider the following "characterization" of a matroid.

(m*) All maximal independent sets of the associated subset system have the same number of elements.

While every matroid has this property, it is not sufficient in itself to characterize them. The following sequence of questions expands this idea.

(a) Define carefully the notion of a *subset system*. The definition must be formal enough to remove any possibility of confusion. It should furthermore be a general definition, not tied to the travelling salesman problem in any way.

(b) Define carefully the *travelling salesman problem*. Make sure that the context in which the problem is considered is presented. It may be assumed that basic notions such as *vertex*, *edge*, *directed graph*, *undirected graph*, are understood, but notions such as *tour* and the like must be explained.

(c) Give a subset system for which the maximal independent sets are precisely the feasible solutions for the travelling salesman problem.

(d) Show that every maximal set in the answer to (c) has the same number of elements.

(e) Show by example that the subset system of (c) is not a matroid.

(f) Give an expanded version of the faulty characterization (m*) above which does provide necessary and sufficient conditions for a subset system to be a matroid. (Just state the condition; no proof is required.)

(4: 60 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:

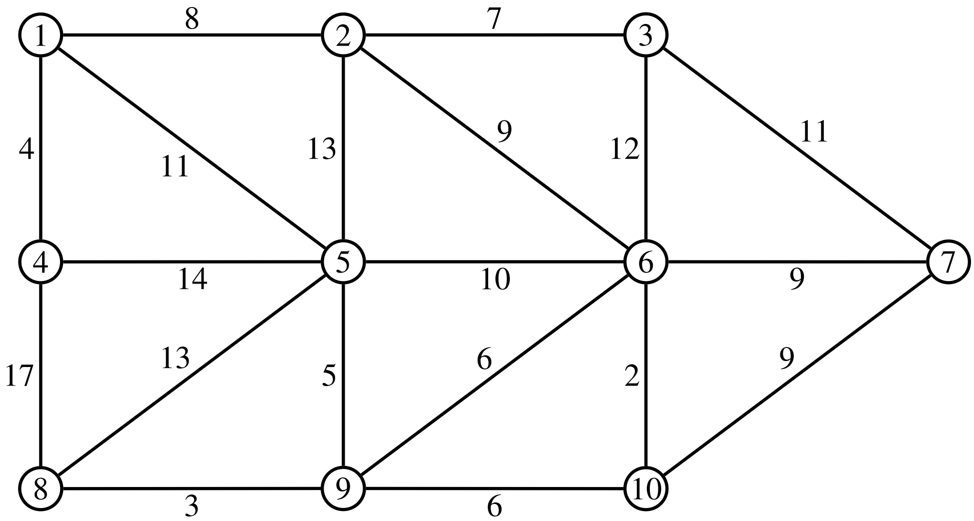
  (i) how it is implemented using arrays; and
  (i) how the operations of a priority queue are represented within it.

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

(d) State (proof is not required) the worst-case complexity for creating a min-heap from an appropriate set of data.

(e) Explain briefly how a min-heap may be used as the core of a sorting algorithm. Based upon the complexities identified in (c) and (d) above, give the worst-case complexity of such an algorithm.

(5: 60 points total)



(a) For the graph shown above, show all possible sequences of edges which may be considered by Kruskal's algorithm for the minimum spanning-tree problem. (Since there are several edges with the same weights, the choices are not always unique.) In each list, draw a line through those which are rejected. The sequence may end when a minimal spanning tree is found. Represent each answer as a distinct sequence, as illustrated below.

(b) Repeat (a), this time for Prim's algorithm.

<u>Note</u>: The format for the answers should be as was used in the course notes. An example is shown below. This is an example to illustrate the correct notation only; it is clearly not a correct answer for either of the above cases.

Edge:	Cost:
$\{1,2\}$	8
$\{3,6\}$	12
$\{4,5\}$	14
$\{2,6\}$	9
$\cancel{\{1,4\}}$	4
$\{3,7\}$	11
$\cancel{\{2,5\}}$	13
$\cancel{\{1,5\}}$	11
$\cancel{\{2,3\}}$	7
$\cancel{\{5,6\}}$	10
$\{9,10\}$	6

(6: 60 points total) It is easy to see how a priority queue may be used to manage the selection process in Kruskal's algorithm. This is less clear for Prim's algorithm. Describe clearly how a priority queue may be used to obtain a worst-case time complexity of $\Theta(n_E \cdot \log(n_v))$ for Prim's algorithm, with n_E (resp. n_v) denoting the number of edges (resp. vertices). The answer contain a clear explanation of why $\Theta(n_E \cdot \log(n_v))$ is a correct characterization of the complexity.

(7: 50 points total) The standard dynamic programming algorithm for the discrete knapsack problem is *pseudo polynomial time*. Define clearly what this means, and explain its significance in terms of the fact that this problem is known to be NP-complete.