Umeå University
Department of Computing Science
TDBC91 — Analysis of Algorithms
Examination: January 7, 2003

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

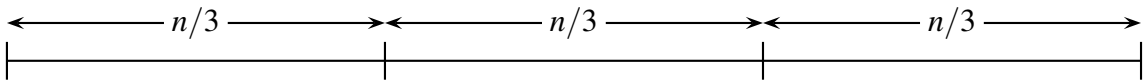
6. In many of the problem with multiple parts, the answer to later parts depends upon the answer to earlier parts. If the answer to an earlier part is wrong, and if that error makes the answer to a later part much easier or trivial, then full credit will not be awarded for that part even if the answer is correct relative to the previous error.

(1: 50 points total) Consider the following two complexity classes:

$$\begin{aligned} C_1 &= O(n \cdot \sqrt{\log(n)}) \\ C_2 &= O(n \cdot \log(\log(n))) \end{aligned}$$

Determine which of the following four conditions holds: (a) $C_1 = C_2$, (b) $C_1 \subsetneq C_2$; (c) $C_2 \subsetneq C_1$; (d) both $C_1 \not\subseteq C_2$ and $C_2 \not\subseteq C_1$. You must prove your answer to obtain credit. A correct guess without justification will not receive credit.
Note: Here "$\subsetneq$" means "is a proper subset of," while "$\not\subseteq$" means "is not a subset of." (A *proper* subset P of A satisfies both $P \subseteq A$ and $P \neq A$.)

(2: 60 points total) Consider a variation on the mergesort algorithm in which the array to be sorted is broken into three pieces of equal size, with each piece then recursively processed using the same technique, and the resulting pieces merged together. (For simplicity, assume that n is a power of 3.) The idea is recaptured in the picture below.



(a) Write down the recurrence relation which characterizes the time complexity of this algorithm. (You do **not** have to write down the algorithm; just give the recurrence relation which governs it.)

(b) Solve this recurrence relation, using whatever techniques are appropriate. You need not explicitly solve for constants, but you must produce the correct form of the solution. You must show intermediate work, including the characteristic polynomial and how you obtained it, as well as any variable transformations which were necessary in the solution.

(c) Interpret the result of part (b) in the context of the original problem. That is, use the result of part (b) to give the complexity of the sorting algorithm.

Note: If your answer to (a) is incorrect and results in a recurrence relation which is much simpler to solve than the correct one, then substantial credit will not be obtained for parts (b) and (c), even if the answers are correct for the recurrence used.

(3: 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.

(4: 60 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 minimum spanning-tree problem using Kruskal's algorithm. Also state (a proof is not required) the worst-case complexity of the algorithm using this implementation.
(Important: Kruskal's algorithm, as describe in the course, uses both a min-heap and a parition-union type. Your explanation must focus upon the use of the partition-union type, and not the min-heap.)

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: 60 points total) The following questions address various aspects of the *backtracking* search strategy.

(a) Describe, in general terms, the structure of the control strategy for backtracking. If special supporting terminology, such as *E-node*, is employed, it must be carefully defined as well.

(b) For the specific problem of discrete knapsack optimization, describe the general properties which an acceptable bounding function must have.

(c) For the discrete knapsack optimization problem, identify two nontrivial and fundamentally different examples of bounding functions which may be used in conjunction with a backtracking search strategy.

(6: 60 points total) A substantial improvement over the naïve algorithm for evaluating a polynomial

$$p(x) = \sum_{i=0}^{n} a_i \cdot x_i$$

at a particular value for x is Horner's rule. Answer the following questions.

(a) Give a pesudo-code description of Horner's rule.

(b) Show, step-by-step, how this algorithm evaluates the polynomial

$$5 \cdot x^3 + 3 \cdot x^2 + 8 \cdot x + 9$$

at $x = 4$. For each step, show both the formula and the values which are substituted. If you prefer, you may leave your answers (final and intermediate) in the form of a sum of products of integers.

(c) State the best, average, and worst-case time complexities for this algorithm, as a function of the degree n of the polynomial. (A proof is not required.)

Notes:

- It must be Horner's rule which is described. Other algorithms will not be accepted.
- Assume that the polynomial is **not** sparse. In other words, do not present the special algorithm for sparse polynomials.

(7: 60 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).