5DV022 Analysis of Algorithms Fall 2008

1 Course Staff

Instructor:	Name:	Stephen J. Hegner
	Office:	C444 MIT-huset
	Telephone:	090 786 5760
	E-mail:	hegner@cs.umu.se
	URL:	`http://www.cs.umu.se/~hegner`
	Office hours:	1200-1300 on days when there is a lecture, and by appointment

2 Course Language

All lectures will be given in English, and all written work must be submitted in English. Exceptions regarding written submissions may be made upon special arrangement with the instructor. For the final examination, it will be permitted to use an XX-English / English-XX dictionary, where XX is a natural language of the student's choice.

3 Course Literature

There is no requirement that the student purchase any particular book. There are three books which surround this course; each has its strengths and weaknesses.

1. Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasekaran, *Computer Algorithms*, W. H. Freeman, 1998; ISBN: 0-7167-8316-9.

   This is the official textbook for the course. It presents a design-oriented approach to the subject, and, by far, follows the course outline and topics most closely. Unfortunately, the presentation is not as clear as that of some other texts. Only this book has been ordered by local bookstores.

   There are two versions of this book which are in print. One expresses the algorithms in a high-level pseudocode, and the other in C++. For clarity of concept without unnecessary detail, the pseudocode version of the book has been selected as the course text. However, for each pseudocode program in the text, the corresponding C++ program may be found on-line, at the home page of the textbook: `http://www.cise.ufl.edu/~raj/BOOK.html`.

2. Thomas R. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, Second edition, MIT Press, 2001; ISBN: 0-262-03293-7.

   This book is a standard reference on the analysis of algorithms, and is very well written. It is very strongly recommended as a reference book. Unfortunately, it uses a problem-by-problem organization, rather than a design-strategy organization, and so is not well suited for this course. Also, it does not cover search algorithms, a major topic of this course.

3. Gilles Brassard and Paul Bratley, *Fundamentals of Algorithmics*, Prentice-Hall, 1996; ISBN: 0-13-335068-1.

   This book was used in a previous offering of the course. It presents a design-oriented approach, and is reasonably well written. However, its coverage of topics is not as extensive as that of Horowitz *et al.*

In addition to the course text, there will be relatively detailed overhead slides, as well as a few papers from the literature. These materials will be available for download on the course web page.

4 Course Content and Outline

The official *kursplan* (an official document identifying the topics and goals of the course) is available on the course web site. A more offering-specific outline follows.

The course will be divided into two parts.

Analysis and Design of Algorithms (75%): In this part of the course, some of the basic techniques used in the analysis and design of non-numerical algorithms which have become central to the study of computer science in recent years will be examined. Both the theoretical and experimental aspects of the subject will be covered. In particular, the following questions will be considered.

- Both formal characterization and software profiling of algorithm performance will be given strong consideration, with special emphasis upon how the two methods complement one another.
- The presentation will be organized around design paradigms, so that the design as well as the analysis of algorithms will form a central focus.

Complexity of Problems and Problem Classes (25%): In this part of the course, the intrinsic complexity of problems (as opposed to algorithms) will be examined. After briefly examining the inherent complexity of solving certain ubiquitous problems (such as sorting and searching), attention will be turned to the important class of NP-complete problems. These are often considered "intractable" in the literature. In this course, the focus will be upon various approximation and heuristic techniques which have proven useful in solving such intractable problems in practice will be studied. A theoretical presentation of the properties of the class NP is now part of the course Complexity Theory, and so this topic is no longer covered in detail in this course on the Analysis of Algorithms.

Due to time constraints, and because there are other courses available, material on algorithms for parallel computation will not be covered in this course.

An outline of the course is shown below. The numbers shown in the rectangular brackets (i.e., [..]) identify chapters and sections in the textbook. The numbers in angle brackets (i.e.,

⟨...⟩) indicate the approximate number of 45-minute lecture periods which will be devoted to the topic.

- Reasonably detailed overhead slides will be available for many topics. The authoritative source for relevant (i.e., possible examination material) is the course lectures and these slides. In many cases, material not covered in the textbook may nonetheless be covered in lecture presentations.
- The number of 45-minute lecture "hours" to be devoted to each topic is approximate. Adjustments may be made as the course progresses. In addition, some class time will be used to discuss the solutions to obligatory exercises.

1. Introduction [] ⟨1⟩
2. Basic tools for algorithm analysis [1] ⟨3⟩
   - 2.1 Mathematical techniques
   - 2.2 Experimental techniques
3. Algorithm Analysis and Design Strategies
   - 3.1 Divide-and-conquer [3] ⟨4⟩
   - 3.2 The greedy method [4] ⟨5⟩
   - 3.3 Dynamic programming [5] ⟨4⟩
   - 3.4 Basic search and traversal techniques [6] ⟨2⟩
   - 3.5 Backtracking [7] ⟨1⟩
   - 3.6 Branch-and-bound [8] ⟨2⟩
   - 3.7 Algebraic simplification and transformations [9] ⟨3⟩
4. Problem Classes and Problem Complexity
   - 4.1 Lower bound theory [10] ⟨1⟩
   - 4.2 NP-complete and NP-hard problems [11] ⟨2⟩
   - 4.3 Approximation algorithms for NP-hard problems [12] ⟨4⟩

5 Course Materials Outline

5.1 Textbook Materials Outline

The following is a section-by-section outline of the textbook. For each section, a symbol is given which indicates the nature of coverage in the course. The meaning of these symbols is provided in the table below.

✔	Material will be covered in the course.
✘	Material will not be covered in the course.
✙	Review material, prior knowledge is expected.
✥	Material will be covered partially or selectively.
✻	Tentative material, may be covered, time permitting.

Entries have not been provided for sections entitled ”References and Readings” or ”Additional Exercises.” In addition, no section-by-section breakdown is given for chapters which are to be omitted entirely.

1. Introduction
   - 1.1. What is an algorithm? ✔
   - 1.2. Algorithm specification ✔
   - 1.3. Performance analysis ✔
   - 1.4. Randomized algorithms ✘
2. Elementary Data Structures
   - 2.1. Stacks and queues ✙
   - 2.2. Trees ✙
   - 2.3. Dictionaries ✙
   - 2.4. Priority queues ✔(with Kruskal's algorithm: 4.5.2)
   - 2.5. Sets and disjoint union ✔(with job sequencing with deadlines: 4.4)
   - 2.6. Graphs ✙
3. Divide and Conquer
   - 3.1. General method ✔
   - 3.2. Binary search ✔
   - 3.3. Finding the maximum and minimum ✘
   - 3.4. Merge sort ✔
   - 3.5. Quicksort ✔
   - 3.6. Selection ✔
   - 3.7. Strassen's matrix multiplication ✘
   - 3.8. Convex hull ✔
4. The Greedy Method
   - 4.1. The general method ✔

- 4.2. Knapsack problem ✔
- 4.3. Tree vertex splitting ✘
- 4.4. Job sequencing with deadlines ✔
- 4.5. Minimum-cost spanning trees ✔
- 4.6. Optimal storage on tapes ✘
- 4.7. Optimal merge patterns ✘
- 4.8. Single-source shortest paths ✔

5. Dynamic Programming
   - 5.1. The general method ✔
   - 5.2. Multistage graphs ✔
   - 5.3. All pairs shortest paths ✔
   - 5.4. Single-source shortest paths: general weights ✔
   - 5.5. Optimal binary search trees ✘
   - 5.6. String editing ✘
   - 5.7. 0/1 knapsack ✔
   - 5.8. Reliability design ✘
   - 5.9. The travelling salesperson problem ✔
   - 5.10. Flow shop scheduling ✘
6. Basic Search and Traversal Techniques
   - 6.1. Techniques for binary trees ✔
   - 6.2. Techniques for graphs ✔
   - 6.3. Connected components and spanning trees ✘
   - 6.4. Biconnected components and DFS ✘
7. Backtracking
   - 7.1. The general method ✔
   - 7.2. The 8-queens problem ✔
   - 7.3. Sum of subsets ✘
   - 7.4. Graph coloring ✘
   - 7.5. Hamiltonian cycles ✘
   - 7.6. Knapsack problem ✔
8. Branch-and-Bound
   - 8.1. The method ✔
   - 8.2. 0/1 knapsack problem ✔
   - 8.3. Travelling salesperson ✔
   - 8.4. Efficiency considerations ✔
9. Algebraic Problems
   - 9.1. The general method ✔
   - 9.2. Evaluation and interpolation ✔
   - 9.3. The fast Fourier transform ✔
   - 9.4. Modular arithmetic ✘

Version: August 17, 2008

- 9.5. Even faster evaluation and interpolation ✘
- 10. Lower Bound Theory
  - 10.1. Comparison trees ✔
  - 10.2. Oracles and adversary arguments ✔
  - 10.3. Lower bounds through reductions ✔
  - 10.4. Techniques for algebraic problems ✘
- 11. NP-Hard and NP-Complete Problems
  - 11.1. Basic concepts ✔
  - 11.2. Cook's theorem ✥
  - 11.3. NP-hard graph problems ✥
  - 11.4. NP-hard scheduling problems ✥
  - 11.5. NP-hard code-generation problems ✥
  - 11.6. Some simplified NP-hard problems ✥
- 12. Approximation Algorithms
  - 12.1. Introduction ✔
  - 12.2. Absolute approximations ✔
  - 12.3. ε–approximations ✔
  - 12.4. Polynomial-time approximation schemes ✔
  - 12.5. Fully polynomial-time approximation schemes ✔
  - 12.6. Probabilistically good algorithms ✘
- 13. PRAM Algorithms ✘
- 14. Mesh Algorithms ✘
- 15. Hypercube Algorithms ✘

5.2 Supplementary Materials Outline

Several topics which are not covered in the textbook, or are covered insufficiently, will be included in the course. For these topics, supplementary materials (which may be part of the lecture notes) will be made available. The major topics, together with the associated topic number from Section 4 of this syllabus, are as follows.

1. Documentation on the gprof profiling package [2.2]
2. Recurrence relations [2.1]
3. Matroids and the formalization of the greedy method [3.2]
4. AND/OR graphs and game trees [3.4]
5. Cook's theorem [4.2]

6 Laboratory Schedule

There is no official laboratory booking for the course, nor any in-laboratory instruction. In general, when not reserved by a course, the computer laboratories of the department are open for use by students for their coursework.

7 Course Schedule

The table below identifies the course meeting times and places, together with the nature of the meeting. The key ”L” denotes a lecture, while ”E” denotes an examination.

For each lecture, the topics to be covered are identified via the outline header number of Section 4 of this syllabus. So, for example, on September 20 the topics of 3.3, dynamic programming, will be covered. This is only an approximate assignment of meeting times to topics, and it may be altered as the course progresses.

For the final examinations, *ÖP* stands for Östra Paviljongen.

Week	Type	Date	Time	Room	Topics
36	L	Sep 01	0815-1000	MA226	1, 2.1
36	L	Sep 04	0815-1000	MA226	2.1, 2.2
37	L	Sep 08	0815-1000	MA226	3.1
37	L	Sep 11	0815-1000	MA226	3.1
38	L	Sep 15	0815-1000	MA226	3.2
38	L	Sep 18	0815-1000	MA226	3.2
39	L	Sep 22	0815-1000	MA226	3.2, 3,3
39	L	Sep 25	0815-1000	MA226	3.3
40	L	Sep 29	0815-1000	MA226	3.3, 3.4
40	L	Oct 02	0815-1000	MA226	3.4, 3.5
41	L	Oct 06	0815-1000	MA226	3.6
41	L	Oct 09	0815-1000	MA226	3.7
42	L	Oct 13	0815-1000	MA226	3.7, 4.1
42	L	Oct 16	0815-1000	MA406	4.2
43	L	Oct 20	0815-1000	MA226	4.3
43	L	Oct 23	0815-1000	MA226	4.3
44	E	Oct 30	0900-1500	Skrivsal 3 ÖP	Final examination
02	E	Jan 07	0900-1500	Skrivsal 1 ÖP	Final examination
13	E	Apr 14	0900-1500	Skrivsal 7 ÖP	Final examination

8 Prerequisites

The formal requirements are listed in the course plan, which may be found at the following link. They include the following.

1. A basic knowledge of programming and data structures.
2. A knowledge of programming in C in the Unix/Linux environment.
3. A knowledge of discrete mathematics and the formal foundations of computer science.
4. A knowledge of single-variable differential and integral calculus.

Although not listed, it is also expected that the student has some basic knowledge of matrices and linear algebra (product of matrices, inverse of a matrix). It is unlikely that anyone who has progressed to this level would not have at least a nodding acquaintance with these topics, but if so, the instructor can recommend some background reading.

9 On-Line Resources

Whenever feasible, images of overhead transparencies and the like will be made available on-line, at the web page for the course.

http://www.cs.umu.se/kurser/5DV022/H08/index.html.

10 Grading System

There will be a total of 1000 points in the course grading system. These points will be distributed as follows.

Problem exercises (6 @ 50 points each):	300 points
Obligatory software exercises (3 @ 100 points each):	300 points
Final examination:	400 points

Grade boundaries are as follows:

Number p of points	Grade
$p \geq 800$	5 (med beröm godkänd – excellent)
$650 \leq p \leq 800$	4 (icke utan beröm godkänd – very good)
$500 \leq p \leq 650$	3 (godkänd – satisfactory)
$p < 500$	U (underkänd – unsatisfactory)

In addition, it is necessary to receive at least 100 points on the final examination in order to pass the course.

The final examination is offered on three different dates. Examinations may be written until a passing grade is obtained. However, once a passing grade is obtained, it is not permitted to write an additional examination in order to improve it.

11 Problem Exercises

The course includes six weekly problem exercises. The following conditions apply.

- No collaboration whatever is permitted on the problem exercises, as they are part of the course examination. With each such problem set, a cover sheet describing the rules will be distributed. To receive credit, the student must sign the cover sheet and submit it with the problem set.
- The problem exercises are not "obligatory;" failing to complete them results only in a loss of points. Thus, it is quite possible to pass the course, even though all exercises have not been completed. See the further notes under item 10 below.
- Each problem exercise will be distributed approximately one week before its due date. These problem sets will be graded promptly and returned to the student. The problems will then be discussed during a class meeting, so that all students may understand fully how to solve the problems. To receive full credit, solutions must be submitted on or before the date and time stipulated. The rules are as follows:
  - For each working day or fraction thereof that a problem exercise is late, 5 points will be deducted from the score.
  - Once solutions are returned and the problems discussed in class, submissions will no longer be accepted.

12 Software Exercises

12.1 General Remarks

- The software exercises are obligatory. To receive a passing grade in the course, all three software exercises must be completed satisfactorily (godkänd). Note, however, that late submissions will lose some or all points. See the additional remarks under item 11 below.
- The software exercises are *laboratory exercises (laborationer)*, and may be completed in small groups. Collaboration is permitted roughly as described in the documents *Policy for Obligatory Exercises* and *Honor Code*. More details will be provided later, when the descriptions of these exercises are distributed.
- Each software exercise will be distributed approximately two weeks before its due date. To receive full credit, solutions must be submitted on or before the date and time stipulated. The rules are as follows:
  - For each working day or fraction thereof that a software exercise is late, 5 points will be deducted from the score.

  - In addition to the point total, each submission will also be marked as satisfactory or unsatisfactory. Unsatisfactory submissions may be resubmitted to obtain a satisfactory evaluation. However, the point total cannot be increased by a resubmission.
  - To receive a point score greater than 0, a software exercise must be submitted (for the first time) on or before the date of the first final examination. After that, submissions will only be graded as satisfactory or unsatisfactory.

12.2 Obligatory Work Completed in Previous Years

- Points for obligatory exercises do not carry over from previous years.
- If all of the obligatory exercises from a previous year were completed, then credit for completing the obligatory exercises, with zero points, will be awarded for the current year upon explicit request on the part of the student. This will not be done automatically; the student must make an explicit request.
- Submissions from different years cannot be mixed. Either credit is awarded for all exercises completed in a previous year, or else all exercises for the current year must be completed.
- To the extent that the problems are similar, work from solutions developed in previous years may be used in solutions for the current year. Such re-use must be explicitly acknowledged, and may not be shared with partners who were not part of development of the original solutions.