

---

# COSC 3101 - Design & Analysis of Algorithms

Summer 2005

---

## Course Outline

*This document contains combined information from the COSC 3101 webpage. You may wish to visit the course webpage regularly for updated info <http://www.cs.yorku.ca/~papakons/>.*

**Instructor:** Periklis A. Papakonstantinou, e-mail: [papakons@cs.yorku.ca](mailto:papakons@cs.yorku.ca), office: CSEB 2016

**Teaching assistant:** Hoda Dehmeshki, e-mail: [hoda@cs.yorku.ca](mailto:hoda@cs.yorku.ca)

**Lectures:** Tuesday 7-10pm, CB 115.

**Office hours:** Tuesday 5pm or by appointment.

**Course web-page:** <http://www.cs.yorku.ca/~papakons/>

**Course news-group:** [news://york.cs.course.3101](mailto:news://york.cs.course.3101)

## 1 Objectives & Prerequisites

This is an introductory course in the analysis and design of combinatorial algorithms. Emphasis is given on (i) familiarizing the students with fundamental algorithmic paradigms and (ii) rigorous analysis of combinatorial algorithms.

We (quickly) review the prerequisites needed from Discrete Mathematics (counting arguments, induction, recurrence relations and discrete probability); but the students are expected to take care (with the help of the instructor) for a more in-depth acquisition of the subjects taught in (prerequisite) courses at York University.

This is a modern introduction to combinatorial algorithms and it maintains some consistency with previous COSC 3101 courses.

**Primary textbook:** *Introduction to Algorithms (2nd edition)*, by Cormen, Leiserson, Rivest and Stein, MIT Press.

## 2 Grading

The final grade is calculated as follows:

- **Assignments (40%):** 5 assignments 8% each.
- **Midterm (20%):** closed book.
- **Final (40%):** with crib paper.

You have the chance to improve your final grade by adding an extra of at most 15%. There are some optional bonus (that count as a surplus to your final grade):

- 2-3 short quizzes each of 15 minutes long. All together will count as a total surplus of 5%.
- One project involving both theory and programming. This will count as a 10% surplus.

### 3 Tentative Calendar

Here is the list of topics we are going to cover together with an approximate timetable. The details of this schedule are subject to change.

- **Lecture 1** (*May 3*)  
**Introduction & (mathematical) Prerequisites Review:** overview, complexity measures & models of computation, asymptotic notation, elementary counting, graphs, induction & proofs of correctness.
- **Lecture 2** (*May 10*)  
**Greedy algorithms:** general framework, correctness of greedy algorithms, exchange lemmas (elements of matroid theory), minimum spanning tree, unweighted interval scheduling, currency denomination, greedy algorithms with reordering: Dijkstra's algorithm (label setting shortest path).
- **Lecture 3** (*May 17*)  
**Recurrence relations and Divide and Conquer:** solving recurrence relations (bounding techniques, characteristic polynomials etc), Master theorem, introduction to divide and conquer: correctness of D&C algorithms, integer multiplication, Strassen's algorithm, FFT & application of FFT to integer multiplication.
- **Lecture 4** (*May 24*)  
**Divide & Conquer and Sorting lower bounds:** quicksort, mergesort, heapsort, optimality of mergesort and heapsort (on comparison-based models).
- **Lecture 5** (*May 31*)  
Midterm review.
- **Lecture 6** (*June 7*)  
**Dynamic programming I:** general framework, correctness of DP algorithms, all-pairs shortest path (Floyd-Warshall - label correcting), weighted interval scheduling.
- **Lecture 7** (*June 14*)  
**Dynamic programming II:** longest common subsequence, pseudopolynomial time algorithms: knapsack, subset-sum, makespan.
- **Lecture 8** (*June 21*)  
**Network flows and cuts:** general framework, basic flow and cut problems (Ford-Fulkerson & Edmonds-Karp), applications: matching, resource allocation.
- **Lecture 9** (*June 28*)  
**Amortized analysis:** incrementing a binary number, Graham's scan convex hull algorithm, applications of aggregate, accounting and potential method.

- **Lecture 10** (July 5)

**Computationally intractable problems:** definition of NP, reductions and NP-hardness, satisfiability, vertex cover, "canonical" NP-complete problems.

- **Lecture 11** (July 12)

**Approximation algorithms:** greedy approximation of makespan, vertex and set cover, fully-polynomial time approximation schemes for knapsack.

- **Lecture 12** (July 19)

**Randomized algorithms:** elements of discrete probability, randomized algorithms vs average case analysis (quicksort and randomized quicksort), Monte Carlo and Las Vegas algorithms, Miller-Rabin primality testing.

- **Lecture 13** (July 26)

**Algorithms that run forever & Course Review:** this slot is also reserved in case of slow instructor.

## 4 Collaboration & remarking requests

- The assignments should be done by a group of at most two students. You are not only allowed but also encouraged to form study groups. Your assignment report must be prepared solely by you (avoid plagiarism).
- Every assignment comes with a cover page. Please mention every person you collaborated with or write "Collaborators: none". Fill-in every field and sign the cover page in the designated area.
- No late assignments unless there is an acceptable and documented reason. *In case of a late assignment you must inform the instructor before the deadline.*
- Remarking requests: Every remarking request should be addressed directly to the instructor of the course. You should attach an extra cover page mentioning the reason for remarking.

## 5 Assignments preparation

This is a suggestion of how to prepare your assignment reports (and partly how to answer your tests). Any other way of presenting your work is totally acceptable given that conforms to the nature and requirements of this course. Communicating technical mathematical material is a skill required in (or should be developed during) this course. Here are some suggestions that hopefully will help you to improve the presentation of your work (and reduce the risk of losing credits - or help you to get partial marks):

- In most questions you will be asked to devise an algorithm (for a given problem) and analyze its (time) performance. In this course every algorithm should also be accompanied with a proof of correctness and proper time analysis.
- It helps your presentation if you begin every answer by stating (within 2-3 sentences) what is the problem you are solving and briefly state your results. Then you can elaborate by giving the (actual) answer.
- **Present your algorithm in English.** Alternatively (or additionally) you can give the description of your algorithm in **pseudo-code**. Please be careful when expressing your algorithm in English. This means that you shouldn't be vague or ambiguous about its structure. You are *strongly discouraged* in presenting algorithms using specific programming language primitives (e.g. avoid using ++, - or !=).

- There are specific problems where the algorithm might be obvious (or more obvious) but it appears that it gets more involved to prove it correct. Apart from convincing yourself that your algorithm works correctly; the ability to provide convincing (mathematically valid) arguments on the correctness of an algorithm has to do with the ability to grasp its very structure. A goal of this course is to make you develop this ability. Unless the question explicitly specifies that you should merely give an indication of the correctness of your algorithm you are required to carry the proof of correctness in detail.
- It's good to include (at least) one input example and show how your algorithm works on this input. Note that no example is sufficient, *not even as an indication*, to show the correctness of your algorithm.
- Stating the running time should always be accompanied with a counting argument for the time analysis. Any assumptions about the data-structures you use (as a black-box) should be stated explicitly.
- When arguing rigorously you should decide on how much detail you should give. It would be nice if you try to find a balance between detail and clarity. Moreover, the simpler (and shorter) the better your proof is. Note that messy, unnecessarily long and unnecessarily complicated answers involve the risk of losing marks.

## 6 Related readings

Apart from the CLRS (primary textbook) and the notes posted in the handouts no other text or set of notes is required for this course. Please note that using any other source apart from the recommended texts (see below) requires you to contact the instructor beforehand.

You can use the references mentioned below. A short description for each can be found in the course webpage <http://www.cs.yorku.ca/~papakons/texts.html> or <http://www.cs.yorku.ca/~papakons/texts3101.pdf>

- **Algorithmics: The Spirit of Computing (2nd Edition)**, by *D. Harel*, Pearson Education.
- **Algorithms**, by *R. Sedgewick*, Addison-Wesley, 1988.
- **Introduction to Combinatorial Mathematics**, by *C.L. Liu*, McGraw-Hill.
- **Concrete Mathematics: A Foundation for Computer Science (2nd Edition)**, by *R.L. Graham, D.E. Knuth, O. Patashnik*, Addison-Wesley.
- **The Design and Analysis of Computer Algorithms**, by *A.V. Aho, J. E. Hopcroft, J.D. Ullman*, Addison Wesley.
- **The Art of Computer Programming (3 volumes)**, by *D.E. Knuth*, Addison-Wesley.
- **Data Structures and Algorithms (3 volumes)**, by *K. Melhorn*, Springer-Verlag.
- **Network Flows: Theory, Algorithms, and Applications**, by *R.K. Ahuja, T.L. Magnanti, J.B. Orlin*, Prentice Hall.
- **Combinatorial Optimization : Algorithms and Complexity**, by *C.H. Papadimitriou and K. Steiglitz*, Dover Publications.
- **Introduction to the theory of computation**, by *M. Sipser*, PWS Publishing.
- **Computers and Intractability: A Guide to the Theory of NP-Completeness**, by *M.R. Garey and D.D. Johnson*, W.H. Freeman & Co.