COMP90038 Algorithms and Complexity

Introduction and Welcome

Michael Kirley

Lecture 1

Semester 1, 2019

Welcome to COMP90038

- Data structures, including stacks, queues, trees, priority queues and graphs.
- Algorithms for various problems, including sorting, searching, string manipulation, graph manipulation, and more.
- Algorithmic techniques, including including brute force, decrease-and-conquer, divide-and-conquer, dynamic programming and greedy approaches.
- Analytical and empirical assessment of algorithms.
- Complexity classes.

Anany Levitin. *Introduction to the Design and Analysis of Algorithms* Pearson, 2012.

Staff; Learning Management System

Lecturer and subject coordinator:

A/Prof Michael Kirley

Tutors: *see LMS for details*

Other support is provided by your classmates, for example via the LMS Discussion Board.

The LMS is our notice board, repository, and discussion forum.

The Timetable

Timetable changes may occur (our class size has increased significantly)

We use lecture capture which is useful for revisiting points from a lecture; and it can, sort of, be a substitute for the lecture proper, even if it is a poor one.

Tutorials start in Week 2.

Time Commitment

For the 12 weeks of semester, expect

- 34 hours in class,
- 30 hours on assignments
- 24 hours of reading and reviewing
- 24 hours of tute preparation
- 8-12 hours on quizzes and discussion

That is an average of 10 hours per week.

The commitment is well worth it: Knowledge of algorithms is essential for any computing professional, it expands your mind, improves complexion, and contains all the minerals and vitamins essential for developing boundless wisdom.

Assessment

- Assignment 1, due around Week 7, worth 15%.
- Assignment 2, due around Week 11, worth 15%.
- A 3-hour exam, worth 70%.

To pass the subject you must obtain at least 35/70 in the written exam (and 50 overall).

Hurdle requirement: weekly quizzes (complete 8/11)

All these details, and more, can be found on the LMS.

Expectations

You need to catch up on any "assumed background knowledge" that you may not have:

- An understanding of sets and relations;
- A grasp of recursion and recurrence relations; a short tutorial on the latter is in Levitin's book, Appendix B.
- Knowledge of basic data structures, such as arrays, records, linked lists, sets and dictionaries.
- Knowledge of some programming language that has a concept of "pointer".

“assumed background knowledge”

Recommended background knowledge - mathematics

- Quiz Week 1A

Prerequisites - some programming experience

- Quiz Week 1B

How to Succeed

Understand the material, don't just memorize it (apart, perhaps, from the formulas in Levitin's Appendix A).

If you fall behind, try to catch up as fast as possible.

Don't procrastinate. Start assignments before you are ready. Put in the necessary time.

Attempt the tutorial questions every week, before you attend the tutorial, if at all possible.

How to Succeed

Support the learning of your fellow students and expect their support, in class and through the LMS discussion board.

Remember that we are all on the same "learning journey" and have the same goal.

Participate in the discussions on the subject's LMS site and check regularly for announcements.

Over to You—Introductions

Please introduce yourself to your neighbours.

Tell them where you are from, what degree program you are enrolled in, whatever.

Over to You—A Maze Problem

A maze (or labyrinth) is contained in a 10×10 rectangle; rows and columns are numbered from 1 to 10.

It can be traversed along rows and columns: up, down, left, right.

The starting point is (1,1), the goal point is (10,10).

These points are obstacles that you cannot travel through:

(3,2)	(6,6)	(2,8)	(5,9)	(8,4)	(2,4)	(6,3)	(9,3)	(1,9)
(3,7)	(4,2)	(7,8)	(2,2)	(4,5)	(5,6)	(10,5)	(6,2)	(6,10)
(7,5)	(7,9)	(8,1)	(5,7)	(4,4)	(8,7)	(9,2)	(10,9)	(2,6)

Find a path through the maze.

What Is a Problem?

My ODE says: "doubtful or difficult question or task."

In computer science we use the term like that too, but there is a more technical concept of algorithmic problem.

We usually want to find a single generic solution to a bunch of similar questions.

For example, the "maze problem" is to come up with a mechanical solution to any particular maze.

So to us, a "problem" usually has many instances, sometimes infinitely many.

Algorithmic Problems

So a problem in computer science typically means a family of instances of a general problem.

An algorithm for the problem has to work for all possible instances (input).

Example: The sorting problem—an instance is a sequence of items.

Example: The graph colouring problem—an instance is a graph.

Example: Equation solving problems—an instance is a set of, say, linear equations.

What Is an Algorithm?

My ODE says: "process or rules for (esp. machine) calculation etc."

A finite sequence of instructions

- No ambiguity, and each step precisely defined
- Should work for all (well-formed) input
- Should finish in a finite (reasonable) amount of time

The (single) description of a process that will transform arbitrary input to the correct output—even when there are infinitely many possible inputs.

A cookbook recipe?

What Is an Algorithm?

Not long ago, "algorithm" was synonymous with "numeric algorithm".

Mathematicians had found many clever algorithms for all sorts of numeric problems.

The following algorithm for calculating the greatest common divisor of positive integers m and n is known as "Euclid's Algorithm".

To find $gcd(m, n)$:

Step 1: If $n = 0$, return the value of m as the answer and stop.
Step 2: Divide m by n and assign the value of the remainder to r.
Step 3: Assign the value of n to m, and the value of r to n; go to Step 1.

Non-Numeric Algorithms

350 years ago, Thomas Hobbes, in discussing the possibility of automated reasoning, wrote:

> *"We must not think that computations, that is, ratiocination, has place only in numbers."*

Today, numeric algorithms are just a small part of the syllabus in an algorithms course.

The kind of computation that Hobbes was really after was mechanised reasoning, that is, algorithms for logical formalisms, for example, to decide "does this formula follow from that?"

Computability

In 2012 we celebrated Alan M. Turing's 100th birthday.

At the time of Turing's birth, a "computer" was a human employed to do tedious numerical calculations.

Legacy: "Turing machine", the "Church-Turing thesis", "Turing reduction", the "Turing test", the "Turing award"

One of Turing's great accomplishments was to put the concept of an algorithm on a firm foundation and to establish that certain important problems do not have algorithmic solutions.

Abstract Complexity

In a course like this, we are only interested in problems that do have algorithmic solutions.

However, amongst those, there are many that provably do not have efficient solutions.

Towards the end of this subject we discuss complexity theory briefly—this theory is concerned with the inherent "hardness" of problems.

Why Study Algorithms?

Computer science is increasingly an enabler for other disciplines, providing useful tools for these.

Algorithmic thinking is relevant in the life sciences, in engineering, in linguistics, in chemistry, etc.

Today computers allow us to solve problems whose size and complexity is vastly greater than what could be done a century ago.

The use of computers has changed the focus of algorithmic study completely, because algorithms that work well for a human (small scale) usually do not work well for a computer (big scale).

Why Study Algorithms and Their Complexity?

To collect a number of useful problem solving tools.

To learn, from examples, strategies for solving computational problems.

To be able to write robust programs whose behaviour we can reason about.

To develop analytical skills.

To learn about the inherent difficulty of some types of problems.

Problem Solving Steps

- Understand the problem
- Decide on the computational means (sequential/parallel, exact/approximate)
- Decide on method to use (algorithm design technique or strategy, use of randomization)
- Design the necessary data structures and algorithm
- Check for correctness, trace example input
- Evaluate analytically (time, space, worst case, average case)
- Code it
- Evaluate empirically

What we will study

Algorithm analysis

Important algorithms for various problems, primarily

- Sorting
- Searching
- String processing
- Graph algorithms

Approaches to algorithm design

- Brute force
- Decrease and conquer
- Divide and conquer
- Transform and conquer

Study Tips

Before the lecture, as a minimum make sure you have read the introductory section of the relevant chapter.

Always read (and work) with paper and pencil ready; run algorithms by hand.

Always have a go at the tutorial exercises; this subject is very much about learning-by-doing.

After the lecture, reread and consolidate your notes.

Identify areas not understood and use the LMS Discussion Forum.

Rewrite your notes if that helps.

Things to Do in the First Two Weeks

Buy the text, read Chapter 1, and skim Chapter 2.

Make sure you have a unimelb account.

Visit the COMP90038 LMS pages and check any announcements.

Use the LMS Discussion Board; for example, if you are interested in forming a study group with like-minded people, the Discussion Board is a useful place to say so.