



School of Computer Science and Engineering
COMP3121/9101 Algorithm Design and Analysis

Module 0: Welcome

Raveen de Silva
K17-202

Course Admins:
cs3121@unsw.edu.au
(Liam, Leith, Hugo, Stephen, James, Brendan)

2026 T3

Table of Contents

1. What is this course about?

2. Course overview

3. Admin

Why should you study algorithms?

- ▶ This course is about **problem-solving** and **communication**
- ▶ Two pillars:
 - **algorithm design** is the task of designing a process which, when executed, solves a particular problem;
 - This process might be a way of calculating something, selecting something, etc.
 - **algorithm analysis** is the task of assessing such a process (either your own work or someone else's) with regard to the problem.
 - Does it always give the intended result?
 - How quickly does it run?

Why Study Algorithm Design and Analysis?

Algorithm design

- ▶ Algorithm design is the “planning stage”, where we decide what our program should do.
- ▶ We have all tried to code programs only to find a fundamental flaw in our approach later on.
- ▶ A systematic approach to algorithm design separates engineers from “code monkeys”.

Correctness

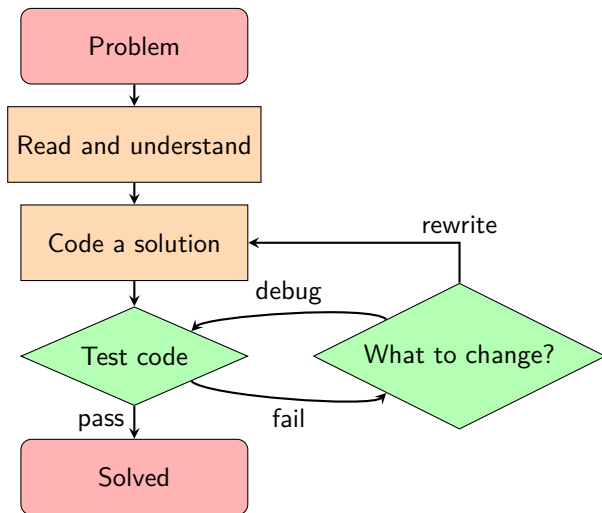
- ▶ A program which fails a test case is wrong, but a program which passes a test case isn't **necessarily** correct.
- ▶ Failure on an obscure case can be catastrophic!
- ▶ Testing is good, good testing is better, proof is best.
- ▶ How do you prove (or at least justify **in general terms**) correctness and explain it to a peer?

Efficiency

- ▶ Using better hardware might save a small (or even large) percentage of the running time of a program.
- ▶ However, a more efficient algorithm can make staggering improvements!
- ▶ Sorting one million numbers could take an hour using bubble sort, or a second using merge sort.
- ▶ Details, such as the choice of data structures, can affect efficiency too.

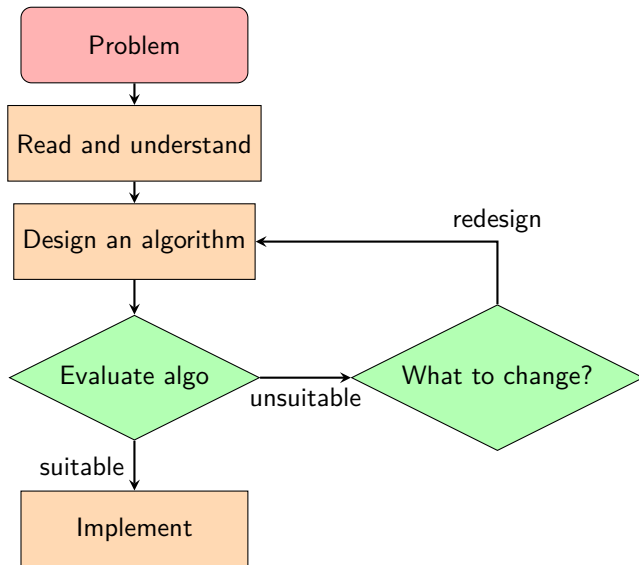
Code-first approach

Solving the problem as you code can lead you into a “debugging vortex”, applying patches without fixing the underlying issue, or to accept incorrect code that passes weak tests.



Design-first approach

Separating design from implementation allows high-level reasoning to confirm that the algorithm meets the requirements of the problem before making implementation decisions.



Separating design from implementation

- ▶ We will talk about algorithms in **plain English**. We will avoid pseudocode, and we will especially avoid using code in any programming language.
- ▶ Implementation is outside the scope of this course.
- ▶ We'll focus on the big picture, to prepare you to:
 - work in teams on large projects
 - communicate clearly and persuasively with stakeholders including colleagues, clients and the public
 - compile design briefs and documentation,which are all important skills for the modern computing professional.

Course Learning Outcomes

1. Explain how standard design techniques are used to develop algorithms.
2. Solve problems by creatively applying algorithm design techniques.
3. Communicate algorithmic ideas at different levels of abstraction.
4. Evaluate the efficiency of algorithms and justify their correctness.
5. Apply the $\text{\LaTeX}$ typesetting system to produce high-quality technical documents.

What is an algorithm?

- ▶ An algorithm is a collection of precisely defined steps that can be executed **mechanically**, i.e., without intelligent decision-making.
- ▶ Designing a recipe involves creativity, executing it does not; the same is true of algorithms.
- ▶ The word “algorithm” comes by corruption of the name of **Muhammad ibn Musa al-Khwarizmi**, a Persian scientist 780–850, who wrote an important book on algebra, “**Al-kitab al-mukhtasar fi hisab al-gabr wa'l-muqabala**”.

Scope

- ▶ In this course we will deal only with algorithms that are:
 - **sequential**: comprised by a sequence of steps, which can only be executed one at a time, and
 - **deterministic**: each step always gives the same result for the same input.
- ▶ Parallel algorithms and randomised algorithms are interesting and powerful, but beyond the scope of this course.

Problem structure

- ▶ Problems may have “flavour text”: a brief story for context. Abstracting this information away is a key first step.
- ▶ Algorithm design tasks may also include an input and output specification if it is not otherwise clear.
- ▶ In this course, we are usually not concerned with the format in which this information is presented, nor with the minute details of how an implementation of the algorithm accesses input or produces output.
- ▶ However, it is crucially important that our algorithm **always** transforms valid input into the appropriate output **efficiently**.

Problem structure

- ▶ We'll usually measure the efficiency of algorithms using **asymptotic analysis**, rather than the exact time or number of operations taken.
- ▶ You will have already encountered big-Oh notation.
 - For example, a process consisting of two nested loops “for each i from 1 to n , for each j from 1 to n ” takes $O(n^2)$ time.
 - We'll formalise these ideas in the next lecture.
- ▶ We'll usually use the **unit-cost** model of computation, where each machine operation ($+$, $-$, $\times$, $\div$ and accessing memory) takes constant time.
 - The only exception is when we discuss divide-and-conquer algorithms to multiply large integers: there we'll use the **logarithmic-cost** model, where these operations take time dependent on the width (number of bits) of the input numbers.

Example problem: Geese in a Trench Coat

Problem: Geese in a Trench Coat

There is a gaggle of n geese, who have heights $h_1, \dots, h_n$. Each h_i is a positive integer.

They need to select two of them to wear a trench coat together, with one standing on top of the other. The trench coat will only fit if the two geese have total height exactly equal to T .

Design an algorithm with worst-case running time in $O(n \log n)$ that determines whether there are two geese who can wear the trench coat together.



Artwork credit: Larissa Yip

Example problem: Geese in a Trench Coat

Abstraction

Design an algorithm which runs in $O(n \log n)$ time in the worst case and, given an array h consisting of n positive integers and a positive integer T , determines whether there are two indices $i < j$ such that $h[i] + h[j] = T$.

- ▶ If we fix index i (one of the geese), then we know that a matching element must contribute the rest of the total (make up the rest of the required height).
- ▶ Let's try all values of i .

Example problem: Geese in a Trench Coat

Attempt 1

For each goose i , check whether a matching goose exists, and answer **yes** if one is found or **no** otherwise.

- ▶ This algorithm is underspecified.
- ▶ It does not describe how to actually perform this check.

Example problem: Geese in a Trench Coat

Attempt 2

For each goose i , a matching goose must have height $T - h_i$. For each other goose j , check whether h_j equals this quantity, and answer **yes** if one is found. In the end, answer **no** if no matching pair was found.

- ▶ It is a correct algorithm.
 - It correctly answers all **yes** instances.
 - It correctly answers all **no** instances.
- ▶ Unfortunately it's too slow: "for each i (of which there are n), for each j (of which there are n), do something" is already n^2 iterations.
- ▶ Can we search efficiently for the correct j , rather than trying them all? Perhaps if the heights are well-organised.

Example problem: Geese in a Trench Coat

Algorithm

Sort the geese in ascending order of height, using merge sort. For each goose i , a matching goose must have height $T - h_i$.

Since the heights are now non-decreasing, we can search for this required length using binary search, and answer yes if it is found.

In the end, answer no if no matching pair was found.



Artwork credit: Larissa Yip

Example problem: Geese in a Trench Coat

- ▶ Is this algorithm correct?
 - Does it correctly answer all **yes** instances?
 - Does it correctly answer all **no** instances?
- ▶ What is the time complexity of this algorithm?
 - How long does the sort take?
 - How many times is the binary search run?
 - How long does each binary search take?
- ▶ You will usually have to justify that your algorithm is correct and that it runs within the specified time complexity.

Algorithm design:

- ▶ **Clarity:** the algorithm is communicated clearly, unambiguously and concisely. All terms used in the algorithm are well-defined. Where appropriate, clearly annotated diagrams supplement the algorithm description.
- ▶ **Style:** the algorithm is written in plain English and not written in (pseudo-)code.

Algorithm analysis:

- ▶ **Time complexity:** the time complexity is correctly measured, with clear reasoning and correct use of asymptotic notation. The time complexity analysis is internally consistent.
- ▶ **Correctness:** the proof is correct. The proof does not leave major gaps for the reader to complete. Computations are correct. The conclusion to the proof is present.
- ▶ **Clarity:** the proof is communicated clearly, unambiguously and concisely. All terms and notations used in the proof are well-defined. Where appropriate, clearly annotated diagrams supplement the proof.

Table of Contents

1. What is this course about?

2. Course overview

3. Admin

Modules

1. Foundations (lectures 2–3)
 - Foundations of algorithm design (data structures, sorting and searching)
 - Foundations of algorithm analysis (correctness, time complexity)
2. Greedy algorithms (lectures 4–7)
 - When local considerations align with global
 - Applications: file compression, graphs
3. Dynamic programming (lectures 8–10)
 - Recursion with **overlapping** subproblems
 - Applications: graphs, string matching
4. Flow networks (lectures 11–12)
 - Routing “flow” through a graph
 - Applications: allocation and matching problems
5. Divide and conquer (lectures 13–14)
 - Recursion with **independent** subproblems
 - Applications: big integer multiplication, signal processing
6. Intractable problems (lectures 15–16)
 - Recognising, analysing and relating “hard” problems

Textbooks

Recommended textbook

Kleinberg and Tardos: **Algorithm Design**
paperback edition available at UNSW Bookshop

- ▶ very readable textbook

Alternative textbooks

- ▶ Cormen, Leiserson, Rivest and Stein: **Introduction to Algorithms**
4th edition now available at UNSW Bookshop
 - good reference manual;
 - somewhat formalistic and dry.
- ▶ Erickson: **Algorithms**, 2019. Available [online](#)
 - fun to read, modern

Prerequisites

- ▶ COMP2521 or COMP9024
 - Fundamental data structures
 - Fundamental algorithms
- ▶ MATH1081¹ or COMP9020
 - Proofs
 - Written communication skills

¹Students in program 3785 Engineering (Honours) / Computer Science are exempted, but advised to self-study, particularly on proof methods

Extended courses

- ▶ The extended courses COMP3821/9801 run once a year.
- ▶ Differences in content and assessment

Courses to consider

- ▶ COMP4121 Advanced Algorithms
 - Real-world algorithms, including randomisation (and therefore probability and statistics)
- ▶ COMP4128 Programming Challenges
 - Algorithms and C++ implementation for small, self-contained problems, as in contests.
- ▶ COMP[29]111 Foundations of Formal Methods, COMP[39]153 Algorithmic Verification
 - Formal and rigorous methods for verifying software
- ▶ COMP4141 Theory of Computation
 - Computability, complexity classes, languages and automata
- ▶ COMP6741 Algorithms for Intractable Problems
 - Preprocessing, approximation, parameterized, exponential-time, quantum, and heuristic algorithms
- ▶ Various MATH courses also build on the content covered in this course.

Table of Contents

1. What is this course about?

2. Course overview

3. Admin

Portfolio

- ▶ Tasks released weekly on [Formatif](#)
- ▶ In each task, incorporate lab instructor's feedback until task complete
- ▶ Contract grading
 - Choose your target grade PS, CR, DN, HD
 - Be realistic; discuss with lab instructor in task 0.0 Introduce Yourself
 - You can change this at any time
 - Attempt tasks **up to this level inclusive**
 - Be ambitious – try some tasks beyond your target level
- ▶ At end of term, complete the Create Portfolio task and include a Learning Summary Report to recap what you found instructive and highlight your attainment of learning outcomes

- ▶ Portfolio is evaluated holistically by your lab instructor

- Mostly on the basis of task completion (**quantity** of satisfactory work)

Grade level	PS	CR	DN	HD
Total weight	55	15	15	15

- Possible adjustment for **quality** (subjective)
 - Includes writing, demonstrations, partial progress on incomplete tasks and more.
 - Document what learning you have achieved (beyond just fully-completed tasks) in the Learning Summary Report of the portfolio submission.
- ▶ Weighted 60% of course mark

- ▶ Task types: (W)ritten, (M)oodle, (D)emonstration, (E)ngagement

Task type	(W)	(M)	(D)	(E)
Relative weight	5	2	10	10

- ▶ Task sheets on Formatif
- ▶ Task resources on Moodle, suitable for import to Overleaf
- ▶ Start with tasks 0.1 *Getting Started* and 0.2 *Preparing for Success*

Deadlines

Written week x tasks

- ▶ are released on Monday of Week x ,
- ▶ a first submission is due on Sunday of week x , and
- ▶ resubmissions are possible after feedback.

An oral demonstration of week x tasks is due

- ▶ by the end of **your lab** in week $x + 1$.

Extensions to deadlines of

- ▶ up to 1 week: use the inbuilt Formatif functionality to request extensions;
- ▶ more than 1 week: make an application for [Special Consideration](#).

Engagement tasks

Recent feedback:

- ▶ Make the course more engaging
- ▶ Mid-term exam?
- ▶ Peer support

As of 26T2:

- ▶ Lab-based, peer-based exercises
- ▶ Intended to prepare, motivate and challenge you
- ▶ Small groups, no computers
- ▶ Board-work problems, discussion points, ...

Assessment:

- ▶ In your timetabled lab only
- ▶ Complete by demonstrating “active participation throughout the activity, including contributing one’s own ideas and/or providing feedback on others’ ideas”
- ▶ Best 7 (out of 9) will count

Final Exam

- ▶ Multiple choice, short answer and algorithm design problems
- ▶ Shorter than comparable portfolio tasks
- ▶ Hurdle: 40% required to pass the course
- ▶ INSPERA (on campus): [more info here](#)
- ▶ Weighted 40% of course mark

Bonus Marks

- ▶ Reward **contributions to other students' learning**, in classes, labs, and on the forum
- ▶ About 3-5% of students will get at least 1 mark, half as many for each extra mark up to a maximum of 5

Course Delivery

Lectures

- ▶ Weeks 1–5, 7–10 (no lecture in flex week)
 - Monday week 4 cancelled due to public holiday
- ▶ Face to face
 - WEB stream students are welcome to attend in person
- ▶ Recording and live stream on Echo360
- ▶ Slides on Moodle

Seminar: tutor presents worked examples

- ▶ Weeks 1–5, 7–10 (no seminar in flex week)
- ▶ Reinforcement: revise recent lecture content and demonstrate problem-solving process
- ▶ Read the tutorial sheet in advance to get the most benefit
- ▶ Online seminar is recorded

Labs: demonstrators support you with preparing for the course

- ▶ Weeks 1–5, 7–10 (no labs in flex week)
- ▶ (E)ngagement tasks: Collaborate with classmates on (unseen) exercises
- ▶ (D)emonstration tasks: Discuss portfolio tasks with your lab demonstrator
- ▶ Lab demonstrators will spend a maximum of 8 minutes for discussion with one student per lab
- ▶ Both tasks working in parallel

Support

In addition to classes, we provide the following support services and resources:

- ▶ Consultation: for questions about lecture content
 - Mon 1pm–2pm and Thu 1pm–2pm at K17-202
 - Weeks 1–5, 7–10 (no consultation in flex week)
 - Monday week 4 cancelled due to public holiday
- ▶ Forum [Discourse link](#)
- ▶ Drop-in help sessions:
 - Tue 6pm–8pm in CSE Help (K17-G05)

AI assistance

- ▶ This course does not allow the use of generative AI tools in assessments (portfolio and exam).
- ▶ Simple editing assistance (spellchecker, translation of single words and short phrases) is allowed in the portfolio.

Equitable Learning Services

- ▶ [Equitable Learning Services](#) is a free and confidential service that provides practical support to ensure that your health condition doesn't adversely affect your studies.
- ▶ If you have an Equitable Learning Plan, please provide it to us [by email](#) within the first two weeks of the term.

Student welfare resources (with links)

How We Help You at UNSW

General Health and Wellbeing	Educational Adjustments
Feelings and Mental Health	Academic and Study Skills
Sexual Health & Relationships	Special Consideration
Uni and Life Pressures	Aboriginal & Torres Strait Islander Community
Reporting Harassment and Gendered Violence	Legal & Advocacy Services

This course is evaluated each session using the myExperience system.

- ▶ Changes from last term:
 - Reduced difficulty of engagement tasks
 - Simplified assessment of engagement tasks
 - Extended module 1 to solidify foundational knowledge
- ▶ Feedback is always welcome, e.g.
 - myExperience survey
 - feedback category in Discourse
 - email

You are not alone!

What to do in the event of a problem or concern with the course

1

In the first instance, please try to resolve the issue with the **immediate party** - which in most cases will be your **tutor**

2

If unresolved, please escalate to the **course admins and lecturer**

3

If unresolved, please escalate to the CSE Student Representatives at **stureps@cse.unsw.edu.au** (or anonymously through our website)

4

If unresolved, please escalate to the CSE Grievance Officers at **grievance-officer@cse.unsw.edu.au**

5

If unresolved, please escalate to **UNSW Complaints** via the UNSW website

Brought to you by the CSE Student Representatives

Find us at <https://cgi.cse.unsw.edu.au/~stureps/>



That's All, Folks!!