Exam Revision

COMP4128 Programming Challenges

School of Computer Science and Engineering
UNSW Sydney

Term 2, 2025

Table of Contents

Congratulations!

- Congratulations on making it to the end! There was a lot of difficult material and the problem sets were non-trivial.
- There were also a fair few rough edges, thanks for sticking through it all!
- Massive thanks to the staff team, wouldn't be possible without them.

Remaining Dates

- Problem sets due 28th August 11:59pm AEDT
- Diaries due 28th August anywhere-on-Earth, submit on Formatif
- Final exam: 23rd August, 9:45am–3:00pm
  - Instructions coming soon by email.
- Extra labs: Mon 11th, Wed 13th, Mon 18th, Wed 20th, all 2-5pm at OMB 229
- Consultation: at those labs, or email

Feedback

- This course is still iterating, and your feedback is very valuable to us.
- From student feedback so far, I intend to:
  - move contest 1 back one week, and change the format to the same as the other midterm contests
  - add more debugging advice
  - consider introducing a workshop participation incentive.

myExperience

- Please fill out the myExperience survey at https://myexperience.unsw.edu.au.
- Some aspects you might wish to give feedback on:
  - Topics. What should we add/remove?
  - Lecturing style or content, e.g. pacing, choice of examples.
  - Problem sets. Difficulty? Interesting? Deadlines?
  - Problem diaries. Should we keep doing them?
  - Contests/exam. Difficulty, format, etc.
  - Workshops and labs. How can we make them more useful?
  - Forum. How can we best help you outside of class time?
  - Anything we should include on the Tips page of the website, e.g. debugging advice, Mac setup.

Feedback

- In addition to myExperience, you can also give us feedback by:
  - posting on Discourse, anonymously if you want, or
  - emailing me.

Final Exam

- 5 hours, $\sim$7 problems worth 100 points each, with subtasks.
- All topics, *including Computational Geometry*
    - but not the Week 6 extension lectures.
- Expect similar difficulty and style to the contest problems. Problems comparable to a problem set E or F will only appear as the very hardest problem or two in the set, in order to distinguish the top couple of students.
- Problems will use the theory we covered but typically require some modifications first.

Final Exam

- Problems will be in approximate order of difficulty of the full problem.
- Average last year was about 300/800; we'll be aiming for higher (at least 350), but it's hard to predict.
- Read all the problems! Difficulty is subjective, and everyone has different strengths. Also, hard problems may have easy subtasks.
- Use the scoreboard to see what problems other students have had success with, and prioritise those. This is particularly useful to identify easy subtasks.

Final Exam

- Roughly same setup as the contests - submissions on CMS. Bring your username and password.
- Open book - make use of lecture slides and code.
- Internet access is restricted to contest server, course website and documentation from https://en.cppreference.com only; *not* https://cplusplus.com.
- You will have access to your home directory.
- A practice final exam will be made available shortly.

Finals Advice

- Be prepared for bugs to occur, don't let them throw you off.
- Try to get some realistic gauge of how much of your time is spent debugging on problem sets, what your most common bugs are and what works for you debugging wise.
- Reread your old diaries!
- If you feel something is suspicious with your submissions, feel free to send us a clarification.

Study Advice

- Perhaps the most difficult skill required is to recognise what tools to use for which problem.
- This isn't a concern in the weekly problem sets, but the contests should give you an idea of what to expect.
- This skill is best acquired through experience - practice on Codeforces by doing virtual participation in past rounds or doing assorted problems with *tags off*.

Study Advice

- Review problem sets and contests. Try to recall what the key parts of solutions were and how you derived them.
- Look through examples in lectures and tutorials, make sure you understand roughly how to do them.
- I've also created a summary of what I think were some of the key parts of each week.

Summaries Summary

- This is a list of what I think the biggest themes of each week are. So things I've tried to convey and think are important.
- Note: This does **NOT** mean if something isn't on this slide then it won't be on the exam.

Lectures 1–2 (Intro, Getting Started)

- Always ask first whether brute force is fast enough - complexity analysis.
- It’s worth remembering how to do a recursive brute force like in N-Queens.
- If brute force would TLE, instead try to make observations that could form the basis of a greedy algorithm.
- Often want to process in some order, so sorting is useful and leads naturally to linear sweep.

Lecture 3 (Problem Solving Paradigms)

- Binary search is an efficient way to search sorted data - use `lower_bound` or `upper_bound` to avoid bugs.
- We can also use it to search a monotonic function by computing function values on the fly.
- Particularly useful case is discrete binary search, where function values are true/false.
- Whenever you are asked to find the largest/smallest x for which some property holds, at least consider binary search. Consider whether the property is monotonic, and if so, can you test the property quickly for fixed x?

Lectures 4–6 (Data Structures)

- Basic data structures - vector, stack, queue, heap (pq).
- Make sure you remember what sets, maps and order statistics trees do.
- They don't just store elements, they also maintain order! Useful things they do:
  - Store elements.
  - Find the first element less than or greater than x.
  - Find number of elements less than x.
  - Find the kth element in order.

Lectures 4–6 (Data Structures)

- Union find manages disjoint sets. But remember the main limitation, they don't do well when you need to support both insertion and deletion.
- Cumulative array - allows sum over range in $O(1)$.
- Sparse table supports *idempotent* queries in constant time if there are no updates.
- Range tree supports *associative* queries and updates in log time.
  - Subtrees can be identified with subarrays using DFS order.
  - Revise LIS and Max Sum Subrange.

Lectures 4–6 (Data Structures)

- **PS2:**
    - *Classrooms*: good example of sweep with set using a greedy observation.
    - *Ancient Berland Roads*: Both the idea of considering a different order (the reverse) as well as how to maintain extra metadata along with union find (e.g: total population).
- **PS3:**
    - *The Problem Set*: routine but important.
    - *Promotion Counting*: when filtering by two criteria, try using the sweep order to account for one criterion.

Lectures 7–8 (Dynamic Programming)

- You must be able to DP problems by YOURSELF - the theory is secondary. I presented an iterative method to design an appropriate state and recurrence; not the only method, use whatever thought process works for you.
- I expect you know how to do tree DP, exponential DP and DP involving data structures.
- You should at least be comfortable with each example except Art, Key, Texture, where comfortable means you can see how you'd solve them yourself.
- Also review the week 6 revision lecture for more examples.
- **PS4:**
  - *Coloring Trees*: good exercise in designing a good state here.
  - *Wi-Fi*: typical example of DP with a data structure.

Lectures 9–10 (Graph Algorithms)

- Many basic things you will be assumed to know: representing a graph, representing a tree in particular, DFS, DAGs, SCCs, MST.
- DFS is surprisingly useful - DFS tree analysis provides structure, both for directed and undirected graphs
- For trees specifically, know how to compute LCA and the data structure for doing so. The same binary composition structure answers most path queries on trees without updates.
- **PS5:** all problems are instructive.

Lectures 11–12 (Shortest Paths)

- Single source:
  - Unweighted? BFS
  - Non-negative weights? Dijkstra
  - Negative weights, detect negative cycles? Bellman-Ford
- All pairs shortest path: Floyd-Warshall.
- The most important skill is recognising and constructing implicit graphs.
- **PS6:** Again all problems but in particular *Jzzhu and Cities* and *President's Path* for creative applications of Dijkstra and Floyd-Warshall.

Lectures 13–14 (Network Flow)

- Max flow is useful for some optimisation problems which don't permit a greedy or DP solution - clues include lack of useful order.
- Be familiar enough with Dinic's that you can copy paste and use it.
- Be familiar with the basic flow constructions (vertex capacities, undirected graphs, multiple sources and sinks, bipartite matching).
- Know that min cut is equal to max flow - commonly used for assignment problems. Understand the common constructions (Project Selection, Image Segmentation)
- Know how to construct a flow graph to fit a set of constraints (Jumping Frogs, Magic Hours).

Lectures 13–14 (Network Flow)

- **PS7:**
  - *Magic Potion* is a good exercisee in flow graph construction.
  - *Oil Skimming* is a non-obvious but very elegant application of flow. The one time you would want to edit the template.
  - *Delivery Bears* is a nice combination of flow with a technique from earlier in the course.

Lectures 15–16 (Mathematics)

- Know how to do all operations modulo a prime.
- Know how to do primality testing, prime factorisation and finding all divisors both for a single n and for all $n \leq N$.
- Know how to compute binomial coefficient modulo a prime.
- Understand the Combinatorics examples. Especially important: understand how to set up a combinatorial DP.
- Understand how to do exponentiation quickly and the covered examples of why matrix exponentiation is powerful.

Lectures 15–16 (Mathematics)

- **PS8:**
  - *Journey*: manipulate probability expressions.
  - *Math*, *Two Divisors* and *Count GCD*: understand prime factorisation properties.
  - *Magic Gems* and *Another Filling the Grid*: combinatorial DP.

Lectures 17–18 (Computational Geometry)

- Single most important tool is cross product: gives orientation (including collinearity) and areas, and underpins convex hull.
- Lots of ideas from Paradigms: both sweep and binary search are typical.
- Use integers. Never divide.
- Plan implementation before you go into the deep dark forest.
- Write your own tests, particularly for special cases.

Lectures 17–18 (Computational Geometry)

- **PS9:**
  - *Pair of Lines*: exercise in “what can you do in the time complexity”.
  - *Polygons* is quite typical: conceptually OK but a pain to get completely correct.