Unit introduction
COMS20017 (Algorithms and Data)

John Lapinskas, University of Bristol

Course ethos

This half of the unit focuses on algorithms, **not** programming — there will be very little overlap with the TB1 half.

Why bother if there's no programming? Four reasons:

- One day you might need to implement these algorithms.
- One day you might need to understand how these algorithms work.
- One day you might need to come up with your own algorithms. (Much more likely than the above two!)
- One day you might have to go through a job interview...

What about AI?

Classical algorithms have a lot of advantages over an AI-based approach:

- They're guaranteed to work!
- Other people don't have to take this guarantee on trust.
- The IP situation (and ethics!) are clear.

What about AI?

Classical algorith ased approach:

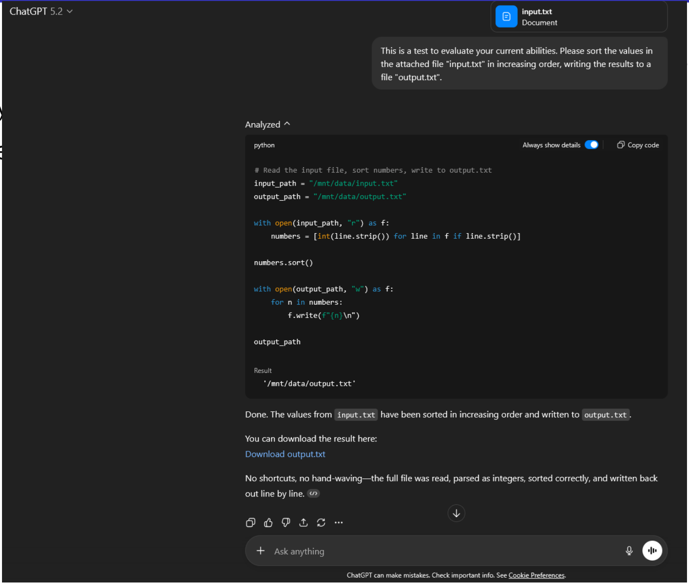
- They're gua
- Other peopl ust.
- The IP situ

```python
from pathlib import Path
from random import randrange, seed

LENGTH = 1000000
UNSORTED_PATH = Path.cwd() / "input.txt"
SORTED_PATH = Path.cwd() / "sorted_output.txt"

seed()
integers = []
for i in range(LENGTH):
    integers.append(randrange(1, 10**9 + 1))

with UNSORTED_PATH.open("w") as output:
    for i in integers:
        output.write(str(i) + "\n")

integers.sort()

with SORTED_PATH.open("w") as output:
    for i in integers:
        output.write(str(i) + "\n")
```

What about AI?

Classical proach:

- They
- Othe
- The



What about AI?

Classical algorithms have a lot of advantages over an AI-based approach:

- They're guaranteed to work!
- Other people don't have to take this guarantee on trust.
- The IP situation (and ethics!) are clear.
- They run *much* faster, cheaper, and more energy-efficiently. (That test took 30 seconds not counting upload/download...)
- The AI probably uses a classical algorithm anyway...!

AI has two major advantages over classical algorithms:

- AI can solve problems which can't be mathematically specified, like "Does this picture contain a cat?".
- AI is good at finding heuristics for problems we don't have good algorithms for, like protein folding.

So if you *can* use a classical algorithm for a problem, you always should.

Assessment and expectations

Bad news: Algorithms are hard! Getting a 2.i is something to be proud of.
Bad news: You need to pass the algorithms half in order to pass the unit.

Good news: Getting a pass in the algorithms half isn't too hard!

Your final grade for the algorithms half will be determined by:

- **90%** from the final exam.
- **10%** from weekly Blackboard quizzes.

The exam questions will start out easy, asking about algorithms you've already seen, then get harder, asking you to design new algorithms.

Good news: You can bring a page of notes into the exam!

Good news: The quizzes are free marks!

Blackboard quizzes

These are auto-marked questions worth **10%** of your final grade:

- One per week, to be done before the week's problem class. (**Including this week!**)
- They should take roughly 1 hour each, but no time limit.
- You can start a quiz and then finish it later.
- Collaboration, online resources etc. are all fine. Study together!
- The usual late policy for coursework applies, so don't miss the deadline or you'll lose a lot of marks very quickly.

Important: If you get 50% or more on a blackboard quiz, this will count as **full marks** in the final grade calculation!

Last year **almost everyone** got above 90% final marks for quizzes. **More than half** got 100%. Free marks!

After a quiz, you get immediate answers and feedback. Don't abuse this. They're important exam prep, so you'd only be cheating yourselves...

Unit schedule

Schedule for week n material:

- Lecture and quiz release: Friday evening, week $n-1$.
  - Lectures are asynchronous videos.
- Do quiz by: 10am Friday morning, week n.
  - Formal deadline (from which UoB late penalties apply) is noon April 27th for all of them. Ignore that though, you'll need them to keep up with the unit.
- In-person problem class: 90 minutes Friday afternoons, week $n+1$.
  - These will be half-lab, half-lecture, all-important.
  - You don't have to try the sheet first! (See unit page...)
  - You do have to have tried your best to understand the week's material. **Do the quiz first!**
- Problem sheet answers release: Friday evening, week $n+1$.
- Q&A session: 2PM Monday, Week $n+1$, in-person in Powell.
  - You can ask questions anonymously (but moderated) via Padlet.
  - Vote on which questions you want me to answer!
  - Alternatively, ask questions on the unit team (1 working day response).

Planning your time

During term, aim to spend at least **7 hours per week** on this unit:

- 2 hours watching the week's lecture videos.
- 2.5 hours *understanding* the week's lecture videos. This could, but doesn't have to, include:
  - Attending the one-hour Q&A session;
  - Attending drop-in sessions (times TBD);
  - Asking questions on the unit Team;
  - Reading textbooks and other sources;
  - Working together with other students;
  - Trying the problem sheet.
- 1 hour finishing the week's Blackboard quiz.
- 1.5 hours attending the week's problem class.

Further details about unit organisation are on the unit page.

Useful references

Proofs on slides are hard, so I provide recommended readings each week on the unit page as an alternative source.

These are all available **as free eBooks** from the university library at https://www.bristol.ac.uk/library/. The most common three will be:

- **Introduction to Algorithms (Cormen et. al.)**
  - Exhaustive reference, classic in the field.
  - As an undergrad I found it quite dry, technical and difficult...
- **Algorithm Design (Kleinberg & Tardos)**
  - Moves very slowly and spells things out in great detail.
  - Does a great job at teaching underlying principles — "how did anyone come up with this?"
  - The book I wish I'd had as an undergrad. Read it!
- **The Algorithm Design Manual (Skiena)**
  - For engineers, by an engineer.
  - The least technical option — great if you're having trouble with proofs.

Mindset for the unit

This unit is hard, because solving problems is hard.

But like most things, you get **much** better at it with practice.

Case in point...

So keep at it, and climb the mountain. ;-)