Algorithmic Essays

Michel Wermelinger
michel.wermelinger@open.ac.uk

Michael Snowden
m.densnow@protonmail.com

The Open University
School of Computing and Communications

We aim for students to consolidate learning from their programming courses and to practice essential skills, with minimal setup requirements.

https://dsa-ou.github.io/algoesup

1. Motivation

- Many CS graduates are good at coding, yet important skills like problem solving, creativity and communication may be neglected.
- Gaining skills by contributing to open source projects demands significant effort from faculty and students.
- Platforms like LeetCode and HackerRank do not typically involve teamwork and communication.

2. Approach

- **Algorithmic essays**: short reports, with code, that explain and compare alternative solutions to a computational problem.
- Our approach uses Jupyter Notebooks.
- Ideally shared with peers for code review and feedback.

3. Benefits

- reinforce learning by:
  - explaining concepts to others
  - exploring alternative solutions to the same problem
  - exercising higher level of Bloom taxonomy
- develop professional skills like:
  - problem solving
  - reviewing code
  - working collaboratively
  - writing readable, tested, documented code with appropriate tools
- produce an artefact (the essay) for portfolio for prospective employers

4. Future work

We are open to **collaborations** to

- extend the resources
- evaluate the approach
- adapt it to various contexts
- integrate it into assessment
- incorporate Generative AI, etc.

Resources

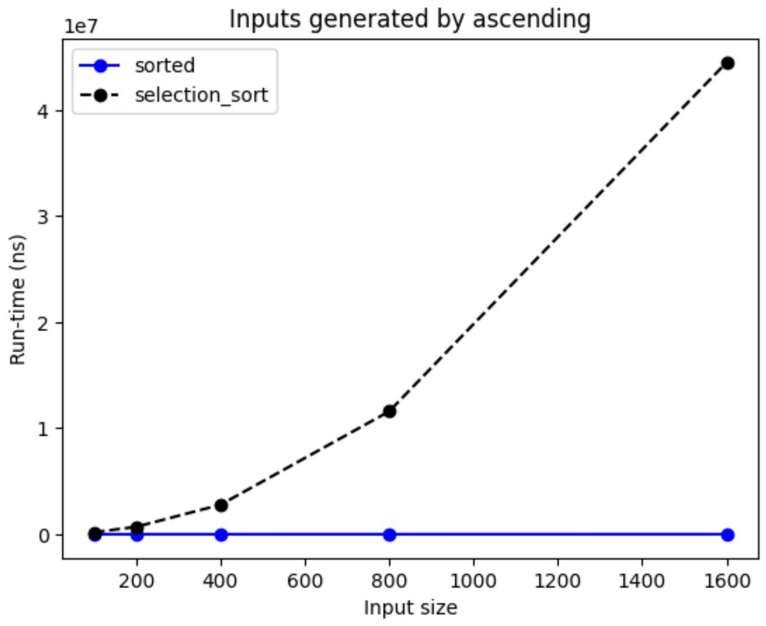
Resources are available under an **open licence** on our GitHub repository and corresponding website (scan the QR code above). They include:

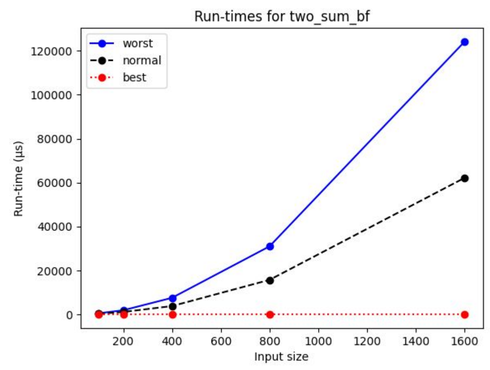
- Guidance on how to get started with **locally installed** Jupyter Notebooks as well as **two cloud-based options** (Google Colab and Deepnote), to avoid installing and configuring software.
- Guidance on **how to write and structure essays**, together with **templates** that follow the suggested structure, to help authors overcome writer's block and start their essay.
- Guidance on **how to ask for, give, and act upon feedback**.
- A small **library** for **linting** Jupyter Notebooks, **testing** code and **measuring its run-time** performance, without the learning curve of professional testing and profiling frameworks.
- A set of **example essays** that demonstrate the approach.

```
time_functions([sorted, selection_sort], ascending, 100, 4, chart=True)
```

```
Inputs generated by ascending

Input size          sorted  selection_sort
       100           488.2       182133.5 ns
       200           831.2       695633.2 ns
       400          1599.1      2779886.7 ns
       800          2847.4     11586610.4 ns
      1600          5559.0     44505450.0 ns
```






```
time_cases(sorted, [ascending, descending], 100, 4, text=False, chart=True)
```

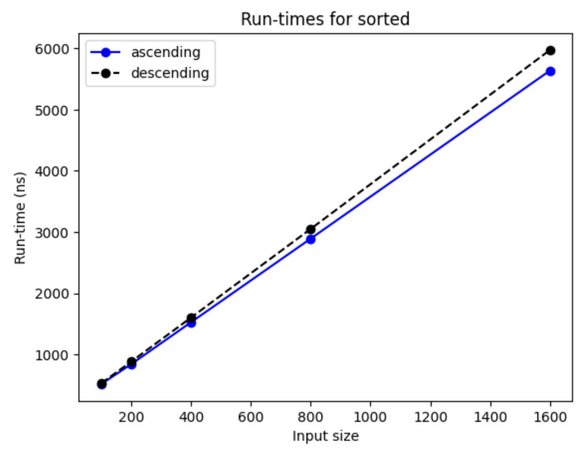


Figure 1. Measuring run-time
(a) 2 functions on 1 input *(b) 1 function on 2 or 3 inputs*

```
[6]: l = "Carol"
     def double(x: int) -> int:
         """Return twice the value of x."""
         return x * 2
     double(l)
     f"Alice and Bob met {l}"
```

```
[6]: 'Alice and Bob met Carol'
```

ruff found issues:

- 1: [E741] Ambiguous variable name: `l`

allowed found issues:

- 6: f-string

pytype found issues:

- 5: Function double was called with the wrong arguments [wrong-arg-types]

Figure 2. Linting Jupyter Notebook cells