Benchmarking sorting algorithms in Python

INF221 small project, NMBU, Spring 2024

Fadi Al Machot
Data Science Section, Faculty of Science and Technology, Norwegian University of Life Sciences

ABSTRACT

Your work on the small project shall train you in performing, analysing and presenting benchmark experiments. As practical examples, sorting algorithms are used and benchmarked against different types of data. All algorithms are implemented in Python. As a reference, also Python's build-in sorting algorithms are benchmarked.

1 INTRODUCTION

In your small project, you will perform benchmarks on several sorting algorithms for test data of various sizes and structures, analyze your results, visualize them graphically, present them, and discuss your results in relation to expectations.

You shall collaborate in teams of three as assigned for the regular exercises in the course. Both students in a team shall contribute equally to all parts of the project, i.e., running benchmarks, analyzing and visualizing results, and writing the paper. You shall provide by

24 March, 23.59

This document will give information about the scope of this project in Sec. 2, the overall structure of the small project in Sec. 3, provide instructions for installation of necessary software in Sec. 4 followed by technical suggestions for benchmarking and analysis in Sec. 5. Finally, Sec. 6 contains information about writing a paper using LaTeX.

For example scientific papers presenting benchmarking results, see Edelkamp and Weiß [2019] for an analysis of a sorting algorithm and Ippen et al. [2017] for an analysis of a completely different problem.

2 SCOPE

This small project aims to compare the real-life behavior of sorting algorithms with theoretical expectations. This requires *measurements*: real-life, physical experiments on computer code and data of repeated tests run on sorted data.

In your paper, you shall implement and benchmark the following sorting algorithms, using the algorithms presented in pseudocode in the course:

- Insertion sort
- Merge sort
- Quicksort

In your benchmarks, you shall use test data suitable to test the the behavior of the algorithms under the worst-case, best case and average-case scenarios. In order to study the scaling behavior of algorithms with problem size, one usually increases problem size n by a factor, e.g., 2, 10 or 16 instead of increasing the problem size linearly.

You can limit your largest problem size so that the full set of all benchmarks do not execute long on your computer. Drop test cases that take a too long time (e.g., large sizes for quadratic algorithms).

2.1 Evaluation and Grading

This small project will be handled as same as an assignment. However, it will be graded (5 points).

3 PAPER STRUCTURE AND LANGUAGE

The paper shall be **at most 3 pages** in the prescribed format (including references). You should merge your pdf paper (3 pages) with your source code (Jupyter notebook + 3 pages paper) in PDF format.

Your paper shall follow the general structure of scientific papers in experimental disciplines and have the following sections in this order:

Abstract Give a concise, single-paragraph overview of the motivation for, goals of, and results of the work presented. It should not reference the literature or figures in the main text.

Introduction Provide a brief introduction to the topic and describe the purpose of the paper, the questions investigated, and the key results obtained. Further, give an overview of the structure of the remainder of the paper.

Methods Describe how you implemented and tested the algorithms, how you generated test data, and how you performed the benchmarks and analyzed results. Provide information on all hardware and software used, but should not include the source code of algorithms tested.

Results Present your results using figures and tables. Figures should be briefly described in figure captions and more thoroughly in the text. In this section, the focus should be on a factual description of the results, not on the interpretation.

Discussion Summarize your findings, compare results obtained for different algorithms and different data, related them to expectations from theory, explain observations and place them in context. You may also suggest further investigations to answer open questions.

References Provide bibliographical information for all references cited in the paper. This section is usually automatically generated from your bibliography database using BibTeX or a similar reference management software.

Most sections should be subdivided into subsections. Further levels of hierarchy are possible, but be careful not to introduce too many levels. A subdivision of a paper should usually have more than one paragraph.

In a paper, you shall try to present a coherent story in a factual style. It is not a diary of your journey from project start to the completed document, but rather an idealized report: Imagine that

you would do the work once again and take your reader through the process from problem definition via data collection (methods) to results and interpretation.

3.1 How to get started

Start by structuring your project. Develop code to perform benchmarks for one algorithm and visualize the running time. Integrate these early results into your paper, so that you have an initial version of your paper with all parts in place. Then add more algorithms, cases, and data incrementally.

Some further suggestions:

- Write in active form, but use the impersonal scientific "we".
- You can choose present or past tense, but be consistent.
- Be precise; give actual values instead of terms such as "large" or "tiny".

Most importantly, make sure that you have (and show) the data to back up any claims you make.

The NMBU Writing Centre provides resources for academic writing at https://www.nmbu.no/en/students/writing/resources.

3.2 Plagiarism

Your small project shall be *your* work. You are encouraged to consult sources from textbooks via scientific papers to online material, but you must properly cite your sources. If you directly copy material from other sources, you must mark it as a quote. For long passages and for figures, you may need to secure the right to include material by others in your texts. You are strongly encouraged to work through the *Write & Cite* online course provided by NMBU's Writing Centre at https://www.nmbu.no/en/students/writing/resources.

4 SOFTWARE

4.1 Software for benchmarking

Use Python for running all benchmarks. You can use your usual Python setup, but it is good practice to create a dedicated Conda environment for each research project to ensure that software versions remain fixed during the course of a project, even if you move to newer versions of some packages elsewhere.

4.2 Software for writing

For writing, you should use LaTeX, as it is the most widely used tool for writing professional-quality texts in math-heavy fields. The following packages provide complete installation packages for LaTeX. Overleaf[1] has become a popular platform for (co-)working on LaTeX documents.

5 BENCHMARKING AND ANALYSIS

Your work should be split into three logical parts:

- generating data;
- analyzing and visualizing data;
- describing and discussing your work and results.

You can use Jupyter Notebooks or normal Python scripts for generating, analyzing and visualizing data. The data generating step shall store benchmark results on disk, where the analysis and visualization step can collect this data. In this way, you are flexible to optimize visualization without interfering with data generation.

[1] https://www.overleaf.com

5.1 Benchmarking

Listing 1 Example script for timinig with `timeit.Timer`. See text for details.

```
import numpy as np
import timeit
import copy

rng = np.random.default_rng(12235)
test_data = np.random.uniform(size=100)

clock = timeit.Timer(stmt='sort_func(copy(data))',
         globals={'sort_func': sorted,
                  'data': test_data,
                  'copy': copy.copy})

n_ar, t_ar = clock.autorange()

t = clock.repeat(repeat=7, number=n_ar)
```

5.1.1 Taking times. We define some terminology first:

- Repetition:
  - a single independent experiment
  - we collect data from multiple repetitions to understand fluctuations in measurement process
- Execution:
  - a single call to function to be timed
  - repeated many times to get sufficient interval for timing
  - each execution must work on pristine data

Our measurement process then looks like this

- Single repetition
  1. Start timer
  2. Call timed function n times (n executions)
  3. Stop timer
  4. Report time difference as a result of repetition
- Selection of the number of executions
  1. Start with some n, e.g. $n = 1$ executions
  2. Perform single repetition
  3. If repetition took less than t_{min}, set $n = 10n$ and go back to 2.
  4. We now have n_E, the number of executions required per repetition (may depend on problem and problem size)
- Multiple repetitions
  1. Once a suitable number of executions has been found, perform r repetitions
  2. Collect results
  3. Report
- Choosing t_{min} and r
  - Python's `titmeit` uses by default $t_{\text{min}} = 0.2$s, but for more reliable measurements 1s seems more appropriate

– r should be at least 3, but 5 or 7 does not hurt

The most important Python tool for benchmarking is the Python `timeit` module[2], which is part of standard Python. In order to have most control over the benchmarking, use the `Timer` class from this module.

Listing 1 shows an example of how to do a single benchmark with the `Timer` class. Note the following points:

(1) We create test data using the new NumPy random interface (NumPy v1.17 and later).
(2) We create a `Timer` an object called `clock`, which we then use to run timing experiments.
(3) The `stmt` argument is a string containing a Python statement. Execution of this statement will be timed. The statement applies `sort_func` to a fresh copy of data.
(4) We need to provide a fresh copy of the data to be sorted on every call to the `sort_func`; otherwise, when doing in-place sorts, only the first execution of the sorting function would sort the test data.
(5) The `globals` dictionary passes variables from the scope of our script to the scope in which the `stmt` is executed.
(6) `clock.autorange()` figures out how many times `stmt` needs to be executed so that total time taken is at least 0.2 ms; this number is returned as `n_ar`.
(7) `clock.repeat` performs `repeat` repetitions of a benchmark experiment, executing `stmt` `number` times for each experiment. It returns a list `repeat` execution times.

You may want to take the source code of the `timeit` module to see how it works.

5.1.2 Managing code and results. Benchmarks should be executed for different data sizes and differently structured data as described in Sec. 2. It may be a good idea to split benchmarks across different scripts or notebooks. For each algorithm, start with a small problem size and increase it until the time for a single run becomes so long that increasing further makes little sense. Try to do this automatically in a loop.

Collect benchmark results in a Pandas dataframe in each script and store them in files using `to_pickle()`. Ideally, your script or notebook should write data to file automatically, no manual action should be required. Choose filenames systematically.

5.2 Analysis and visualization

A Jupyter notebook is probably the most useful approach to analyzing and visualizing benchmark results. The notebook should read the pickled benchmark results from a file, where users combine data from different benchmarks and in the end generate plots displaying results. Choose figures that present the scaling properties of the algorithms well and also indicate variations in measurement results, using error bars or box plots. Plots should be saved in PDF format for integration in the final report document. For best results, figures should be created in the size they will have in the final paper (84 mm wide for a single column figure), with clear lettering in not-too-small fonts (8 points or larger). A sample plotting script is provided with material for this paper, showing how to set font and figure sizes.

[2] https://docs.python.org/3/library/timeit.html

Choose colors wisely and avoid clutter. In scientific articles, one usually avoids legends in graphs as well as figure titles on the top of figures. Line styles are instead explained in the figure caption. Try to be consistent in the use of colors across figures and remember that many of us cannot distinguish red from green.

6 WRITING USING LaTeX

Hear and use the materials of the recorded lecture 9.

REFERENCES

Stefan Edelkamp and Armin Weiß. 2019. BlockQuicksort: Avoiding Branch Mispredictions in Quicksort. *ACM J. Exp. Algorithmics* 24, Article 1.4 (2019), 22 pages. https://doi.org/10.1145/3274660

Tammo Ippen, Jochen Martin Eppler, Hans Ekkehard Plesser, and Markus Diesmann. 2017. Constructing neuronal network models in massively parallel environments. *Front. Neuroinform.* 11 (2017), 30. https://doi.org/10.3389/fninf.2017.00030