Benchmarking sorting algorithms in Python

INF221 Term Paper, NMBU, Autumn 2020

Hans Ekkehard Plesser
Data Science Section, Faculty of Science and Technology, Norwegain University of Life Sciences
hans.ekkehard.plesser@nmbu.no

ABSTRACT

Your work on the term paper 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 reference, also Python's build-in sorting algorithms are benchmarked.

1 INTRODUCTION

In your term paper, you will perform benchmarks on a number of sorting algorithms for test data of various size and structure, analyze your results, visualize them graphically and present them together with necessary theoretical background and discuss your results in relation to expectations.

You shall collaborate in teams of two 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

Tuesday, 27 Oct, 18.00 a first draft of your term paper;
Tuesday, 17 Nov, 18.00 a second draft of your term paper;
Friday, 4 Dec, 12.00 the final version of your term paper.

In this document will give information about the scope of this project in Sec. 2, the overall structure of the term paper 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 examples of 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

The goal of this term paper is to compare real-life behavior of sorting algorithms with theoretical expectations. This requires *measurements*: real-life, physical experiments on computer code and data. Important computer science issues you need to relate to in your work on the paper are the correctness of code (actual implementation, not a pseudo-code algorithm), since it makes no sense to compare incorrect code; the suitable choice of test data, reflecting relevant test cases and real-life situations; and the correctness of the test setup, where a typical error would be that all but the first of repeated tests run on sorted data.

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

- Quadratic algorithms
  - insertion sort
  - bubble sort
- Sub-quadratic algorithms
  - mergesort
  - quicksort
- Combined algorithm
  - mergesort switching to insertion sort for small data
- Built-in sorting functions
  - Python 'sort()'
  - NumPy 'sort()'

In your benchmarks, you shall use test data suitable to test the behavior of the algorithms under worst case, best case and average case scenarios. It is sufficient to test sorting of NumPy arrays with floats. Data shall be sorted *in place*.

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 problem size linearly.

You can limit your largest problem size so that the full set of all benchmarks executes in no more than two hours on your computer. Drop test cases that take too long time (e.g., large sizes for quadratic algorithms) or cannot be completed at all (e.g., recursive algorithms with excessive recursion depth).

2.1 Evaluation and Grading

You will be graded on the A–F scale based on the term paper submitted (PDF file) and the material in your GitLab repository. By default, both partners in a team will receive the same grade, but if there are clear indications that partners have made significantly different contributions to the term paper and underlying code, different grades may be given. Details on evaluation criteria will be given soon.

3 PAPER STRUCTURE AND LANGUAGE

The paper shall be **at most 8 pages** in the prescribed format (including references and acknowledgements, no appendices).

You 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 contain references to the literature or figures in the main text.

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

Theory Describe the algorithms you will benchmark and the expected run-times for the algorithms. Algorithms in pseudocode will be useful in this part. In journal papers, this part is sometimes included in the introduction or methods section and kept very short, as experts in the field are expected to be familiar with the theory.

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, including version numbers, and describe where source code used and data generated can be found. You may include brief code excerpts in this part, but should not include the complete source code of algorithms tested. Include a link to the GitLab repository for your code.

Results Present your results using figures and possibly also 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.

Acknowledgements Thank people who have helped you to prepare the paper, e.g., through technical advice or discussions. In journal papers, this part also includes information about funding sources.

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 too introduce too many levels. A subdivision of a paper should usually have more that 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 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 or two algorithms and one of two cases and use these initial cases to write code to visualise data. 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 Version control of code and text

To keep a log of your benchmarking work including all source code, results obtained and figures and text produced, you shall use the version control software Git and register all material related to this project in a GitLab source code repository. You shall also use this repository to share material between partners. Details on how to set up Git and GitLab and work with it will follow shortly.

3.3 Plagiarism

Your term paper shall be *your* work. You are encouraged to consult sources from textbooks via scientific papers to online material, but your 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.

For this project, you should just create a clone of the `inf221` conda environment that you already have using this command:

```
conda create --name inf221paper --clone inf221
```

4.2 Software for version control

You shall use Git as version control software and a GitLab repository to manage your material. Details will follow shortly.

4.3 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:

Windows MikTeX from https://miktex.org
macOS MacTeX from https://www.tug.org/mactex/
Linux You should be able to install via your package manager

You should install the full packages under Windows or macOS, even though they are rather large. If you are really pressed for disk space, you can install just a minimal system, but then you may need to install add-on packages later.

Overleaf[1] has become a popular platform for (co-)working on LaTeX documents with integration to GitLab. NMBU is about to obtain an institutional subscription at present. More information

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

will follow shortly. Until that comes in place, you can use a version with some limitations for free.

See Section 6 for more information about writing with LaTeX.

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 to disk, where the analysis and visualisation step can collect this data. In this way, you are flexible to optimize visualization without interfering with data generation.

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 measurment 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 result of repetition
- Selection of 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 `timeit` uses by default $t_{\min} = 0.2$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` 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 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.

To ensure maximum traceability of your results, commit all source code to your repository before running a benchmark, and

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

commit the result files immediately after the benchmark is complete. Tools such as Sumatra[3] [Davison 2012] can automatize this, but you are not expected to use them in this project.

5.2 Analysis and visualization

A Jupyter notebook is probably the most useful approach to analysing and visualising benchmark results. The notebook should read the pickled benchmark results from file, where useful 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.

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 use of colors across figures and remember that many of us cannot distinguish red from green.

If you want to include tables of results in your paper, remember that Pandas can generate LaTeX code for dataframes directly. To avoid any copy-paste errors, you can write the resulting LaTeX code directly to a file and include that file automatically in the LaTeX file for your paper.

6 WRITING USING LaTeX

Use LaTeX to write your paper, either with a local installation or using Overleaf. An archive containing a template for your paper is available on the course website. The archive also contains the source code for this document. By building on these example LaTeX source codes, you should be able to figure out how to write code which LaTeX then turns into a beautifully typeset document. MiKTeX comes with the TeXworks frontend for LaTeX and MacTeX with TeXShop. Those frontends should make running life reasonably easy.

Here are some links to introductions to working with LaTeX:

- https://www.latex-tutorial.com
- https://www.andy-roberts.net/writing/latex
- https://www.dickimaw-books.com/latex/novices/
- https://texfaq.org/FAQ-man-latex
- https://www.youtube.com/watch?v=fCzF5gDy60g&ab_channel=AcademicLesson

6.1 Citations in LaTeX

To refer to a source in LaTeX use the `\cite` command and its siblings as shown in Tab. 1. The argument passed to the `\cite` command is the "cite key" for the work you want to refer to, which is defined in the entry for that work in you bibliography file. For the Cormen book, this entry looks like this

```
@book{CLRS_2009,
 author = {Cormen, Thomas H. and Leiserson, Charles E.
          and Rivest, Ronald L. and Stein, Clifford},
 title = {Introduction to Algorithms, Third Edition},
 year = {2009},
 isbn = {0262033844, 9780262033848},
 edition = {3rd},
 publisher = {The MIT Press},
 address = {Cambridge, MA}
}
```

[3] http://neuralensemble.org/sumatra/

Command	Result
`\cite{CLRS_2009}`	[Cormen et al. 2009]
`\citet{CLRS_2009}`	Cormen et al. [2009]
`\cite[Ch.~2.1]{CLRS_2009}`	[Cormen et al. 2009, Ch. 2.1]
`\citet[Ch.~2.1]{CLRS_2009}`	Cormen et al. [2009, Ch. 2.1]

Table 1: Citation commands in LaTeXwith the `natbib` package and the resulting citations in text.

For more examples, see the `sample_bib.bib` file included in the template archive. You are welcome to use the file in your work. For more on bibliographies in LaTeX, see, e.g., https://www.andy-roberts.net/writing/latex/bibliographies.

If you do not run LaTeXthrough TeXworks or TeXShop, you should run the following sequence of commands to ensure that all references are up to date:

```
pdflatex myfile
bibtex myfile
pdflatex myfile
pdflatex myfile
```

REFERENCES

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press, Cambridge, MA.

Andrew P. Davison. 2012. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science and Engineering* 14 (2012), 48–56. https://doi.org/10.1109/MCSE.2012.41

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