SORTING ALGORITHMS IN PYTHON

Mark Collins

12696434, mc24952@essex.ac.uk

12 June 2024

Word count: 1122

1.0 Introduction

Sorting algorithms are an important area of Computer Science research. As trends to utilise "Big data" become more prevalent, efficient sorting algorithms are essential (Naeem et al., 2022). Here, a comparison of sorting algorithms, implemented using Python 3.12.4 will be performed, looking at how efficiently they sort a given dataset ($n > 500000$).

Three algorithms will be used. The first, "Bubble-sort" (Friend, 1956) is a simple algorithm comparing two adjacent elements in the array and switching place depending on which is the higher. This process is repeated for each pair of numbers in the array. After an initial pass, the data may not be sorted, and the process is repeated. In a worst-case scenario, this process, of comparing every element in the array of size n will be repeated n times, meaning the efficiency of bubble sort will tend to n^2 (Dev Mishra & Garg, 2008; Hammad, 2015).

The second algorithm, "Merge-sort" (Knuth, 1987), has been suggested as being efficient for very large datasets (Dev Mishra & Garg, 2008). Merge-sort works on the principle of "divide and conquer". An array of length n is recursively split into n sub-arrays of length 1. These sub-arrays are then combined, or merged, into a sorted order, as shown in Figure 1 (Lobo & Kuwelkar, 2020). Rather than in the case of Bubble-sort, where each element i is compared to $i + 1$, Merge-sort only needs to compare $array_1[i]$ with $array_2[j]$. This efficiency over Bubble-sort means the efficiency of Merge-sort will tend to $n \cdot \log n$ (Dev Mishra & Garg, 2008; Hammad, 2015).

Finally, the built-in sorting function in Python will be used. Until v3.10, the Python sort algorithm used Tim-sort (Peters, 2002). From v3.11 onwards, Python has used Power-sort (Munro & Wild, 2018; The Python Software Foundation, 2021). Both methods further optimise the Merge-sort. The former by combining with Insertion-sort (Mauchly, 1985), and the latter by using adaptive "on-the-fly" merging.

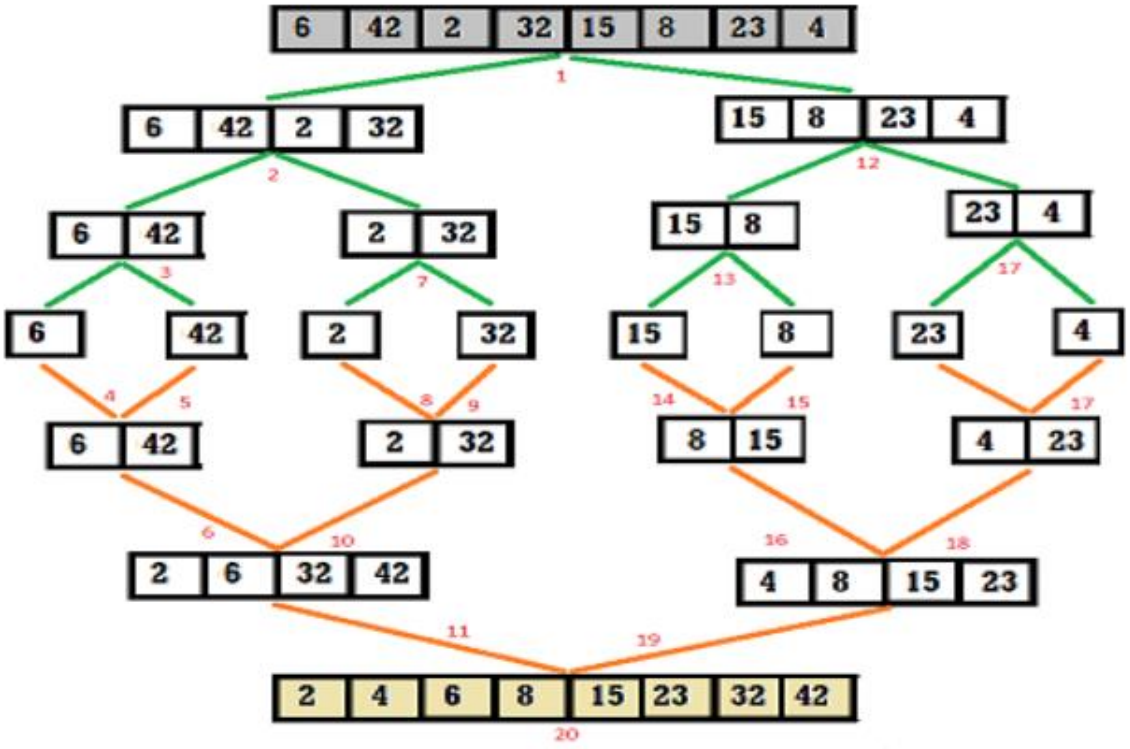

FIGURE 1: EXAMPLE OF MERGE-SORT FOR N=8, TAKEN FROM (LOBO & KUWELKAR, 2020)

2.0 Method, Materials, and Results

The three Python scripts were written using the latest version of Python available (v3.12.4). The time to run the scripts was compared using the NumPy function datetime64.

The three scripts were run for n = 10, 50, 100, 500, 1000, 5000, 10000, 50000, 100000, and 500000. The time taken to sort was recorded for three runs and the average was taken. Results are plotted in Figure 2.

The equipment used was an AMD Ryzen 5 3550H, 2.1GHz CPU with 8GB memory, running Windows 11 23H2. The computer underwent light use while the scripts were running, mainly word processing and connection to a remote Citrix session.

As expected, the Bubble-sort algorithm took the longest ($9 \cdot 10^{10}$ microseconds, or 25 hours) to sort 500,000 elements. This is significantly longer than Merge-sort ($1.4 \cdot 10^{7}$ microseconds, or 10.4 seconds) or the built-in sort function ($4.7 \cdot 10^{4}$ microseconds, or 0.047 seconds).

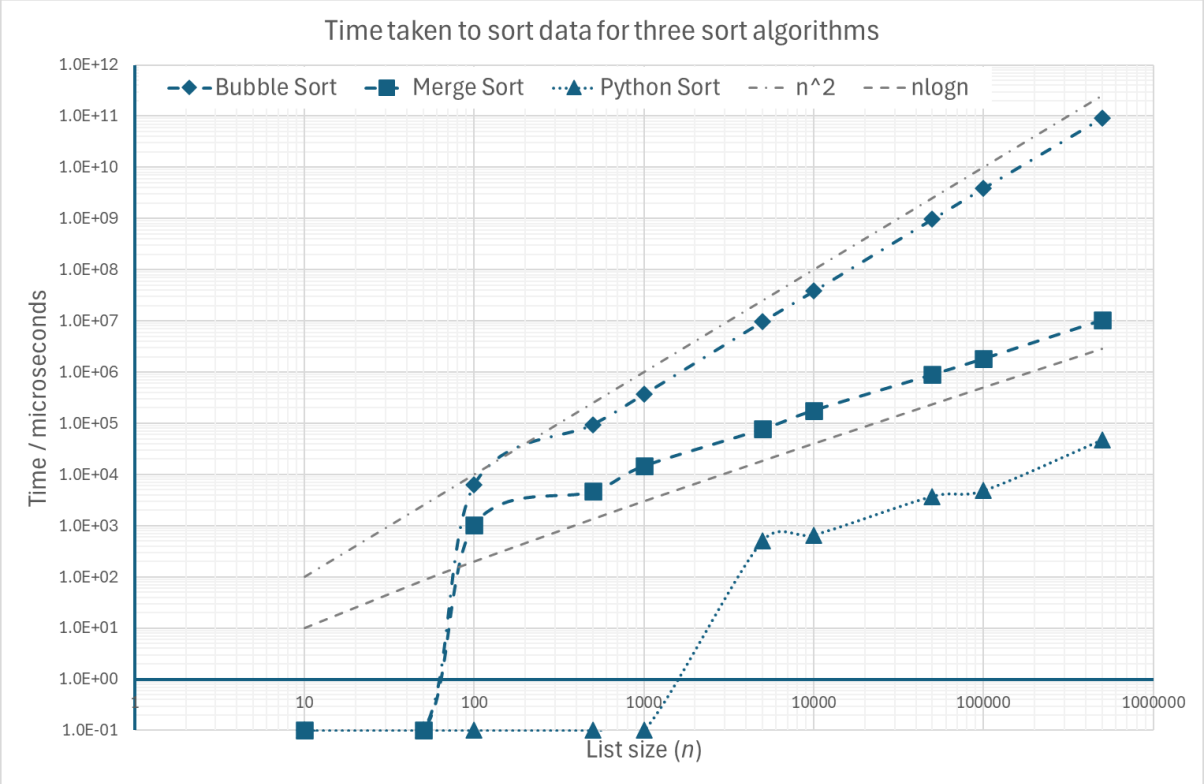

FIGURE 2: TIME TAKEN TO SORT DATA FOR THREE ALGORITHMS

3.0 Discussion or Results

Figure 2 shows the time taken to complete the three sorting algorithms. Also shown is the theoretical efficiency of the sort algorithms of n^2 and $n \cdot \log n$. For $n > 100$ Bubble-sort agrees very well with n^2 and Merge-sort agrees well with $n \cdot \log n$.

For the Bubble-sort, a list size of $n = 100000$ took approximately $4 \cdot 10^9$ microseconds, or just over one hour. Increasing the sample size to $n = 500000$, is a five-fold increase. By following n^2 the increase in time should be $5^2 = 25$ times longer. This is indeed what is seen, and the time taken for $n = 500000$ is approximately 25 hours.

It is perhaps not surprising that the built-in sort function is faster than the standard Merge-sort, since it is essentially a further optimised Merge-sort (Munro & Wild, 2018). What is surprising however is that all the algorithms performed better than theory for small n. For the two "written" algorithms (Bubble and Merge), the time taken to sort $n < 100$ is faster than the time for the internal clock to register time passing. For the built-in sort algorithm, this does not occur until after $n > 1000$. This is most likely due to the speed of the modern CPU. Once the list size becomes more complex, and more comparisons are required, the time taken to sort the data tends towards the theoretical limits.

4.0 Thoughts on Python

Python is a powerful, yet easy to learn and understand programming language. There is a great deal of support documentation available online, and many communities that will support Python programming (Stack Overflow, 2024). The ability to debug and fault find Python script varies between the various integrated development environments (IDE's). The IDE chosen for this work was Visual Studio Code (Microsoft, 2024). In order to debug effectively, one needs to be considerate of where to place breakpoints, however Visual Studio does provide the user the ability to view the current content of the stack. This can be extremely useful when unexpected results are obtained, and one can step through the calculation stages.

This contrasts with previous experience writing programs to solve similar problems using MATLAB (MathWorks, 2024), where the ability to run a line of code directly from the command line, and to have a variable viewer and editor built-into the environment makes fault finding extremely easy.

Given the ease of use and access to Python and Python materials (Python Software Foundation, 2024), it is an ideal tool for automation within organisations. Recently, Portsmouth Hospitals University NHS Trust has invested in a new Radiotherapy Treatment Planning System (TPS). This new system allows custom automation scripts to be written in Python (RaySearch Laboratories, 2024), allowing repetitive and time consuming tasks to be automated. This in effect makes Python essential to modern Radiotherapy treatment planning.

The built-in functions of Python are extremely useful (such as the `list.sort()` function), however the ability to record time to the microsecond requires the additional functionality provided by NumPy. Also, personal experience, additional libraries for PyDICOM have been used (Mason, 2023). This is because Python is provided as a base, for which a user can build on, using the core functionality of Python, with the advantages of additional plug-ins and libraries to enhance the user experience.

5.0 Conclusion

In conclusion, of the three sorting algorithms looked at here, Bubble-sort was the slowest with an efficiency of n^2, shown both theoretically and experimentally. Merge-sort was significantly faster, with proven efficiency of $n \cdot \log n$. Finaly the fastest algorithm, was the built-in Python `list.sort()`, which is built on Merge-sort, but with additional optimisation.

Python is a powerful tool for automating processes and has many advantages making it useful for the modern organisation. However, when implementing python within an organisation, security must be considered. The ease of use and access to Python and Python resources, means a malicious user can easily exploit vulnerabilities left by inexperienced programmers.

6.0 References

Dev Mishra, A., & Garg, D. (2008) Selection of Best Sorting Algorithm. *International Journal of Intelligent Information Processing* 2(December): 363–368. http://www.gdeepak.com/pubs/Selection of best sorting algorithm.pdf

Friend, E. H. (1956) Sorting on Electronic Computer Systems. *Journal of the ACM (JACM)* 3(3): 134–168. DOI:https://doi.org/10.1145/320831.320833

Hammad, J. (2015) A comparative study between various sorting algorithms. *International Journal of Computer Science and Network Security* 15(3): 11–16.

Knuth, D. E. (1987) 'Von Neumann's First Computer Program', In: W. Aspray & A. Burks (Eds.), *Papers of John von Neumann on Computing and Computer Theory*. MIT Press. (pp. 89–95).

Lobo, J., & Kuwelkar, S. (2020) Performance Analysis of Merge Sort Algorithms. *Proceedings of the International Conference on Electronics and Sustainable Communication Systems, ICESC 2020*, 110–115. DOI: https://doi.org/10.1109/ICESC48915.2020.9155623

Mason, D. (2023) *Pydicom*. Available from: https://pydicom.github.io/ [Accessed 12 June 2024].

MathWorks. (2024) *MATLAB*. Available from: https://uk.mathworks.com/products/matlab.html [Accessed 12 June 2024].

Mauchly, J. W. (1985) 'Sorting and Collating', In: M. Cambell-Kelly & M. R. Williams (Eds.), *The Moore School Lectures: theory and techniques for design of electronic digital computers*. MIT Press. 271–287.

Microsoft. (2024) *Visual Studio Code*. Available from: https://code.visualstudio.com/ [Accessed 12 June 2024].

Munro, J. I., & Wild, S. (2018) Nearly-optimal mergesorts: Fast, practical sorting methods that optimally adapt to existing runs. *Leibniz International Proceedings in Informatics, LIPIcs* 112(63): 1–15. DOI: https://doi.org/10.4230/LIPIcs.ESA.2018.63

Naeem, M., Jamal, T., Diaz-Martinez, J., Butt, S. A., Montesano, N., Tariq, M. I., De-la-Hoz-Franco, E., & De-La-Hoz-Valdiris, E. (2022) Trends and Future Perspective Challenges in Big Data. *Smart Innovation, Systems and Technologies* 253: 309–325. DOI: https://doi.org/10.1007/978-981-16-5036-9_30

Peters, T. (2002) *[Python-Dev] Sorting*. Available from: https://mail.python.org/pipermail/python-dev/2002-July/026837.html [Accessed 12 June 2024].

Python Software Foundation. (2024). *Python History and License*. Available from: https://docs.python.org/3/license.html [Accessed 12 Jine 2024].

RaySearch Laboratories. (2024). *Raystation Automated Treatment Planning*. Available from: https://www.raysearchlabs.com/automated-treatment-planning/ [Accessed 12 June 2024].

Stack Overflow. (2024). *Stack Overflow, Questions tagged [Python]*. Available from:

https://stackoverflow.com/questions/tagged/python [Accessed 12 June 2024].

The Python Software Foundation. (2021). *Python 3.11 Changelog*. Available from: https://docs.python.org/3.11/whatsnew/changelog.html [Accessed 12 June 2024].