Matteo Marchi

12695662

17/06/2024

Individual programming exercise

The Python programming language's popularity in the automation domain can be attributed to its versatility in connecting applications, streamlining processes, and simplifying intricate tasks. This is supported by the extensive availability and active community contributing to his development and success (Raschka et al., 2020). The open-source nature of the language and the extensive number of libraries and toolkits have helped his diffusion and adoption (Zhang & Zhang, 2023).

In my organisation Python is widely used for different purposes, for example in Data Analysis, Networking, and scripting within several departments. In my day-to-day tasks I have never used Python so far, instead, I have developed a better understanding of PowerShell and Bash, due to extensive work I must perform on Mobile Device Management Platforms such as Microsoft Intune and JAMF for managing enterprise devices.

For my individual programming exercise, I have selected the Quicksort algorithm to sort all the numbers in the pidata.txt file.

In my coding exercise I have created functions that will generate 2 different .txt files that are subsequently hashed:

- The first generated .txt file contains the sorted numbers, and it is called sortedNumbers.txt
- The second .txt file contains all the requested information on top of the numbers: name, date of birth, student number and 100 randomly inserted copies of my name. This file is called: MatteoMarchi17-06-24.txt as requested for the assignment

Both files are generated using the data contained in the pidata.txt file.

I have selected Quicksort because it is known for its efficiency and speed, often considered as one of the fastest sorting algorithms, it was first developed by Tony Hoare in 1959 and published in 1961 (Hossain et al., 2020). It operates basing on the divide-and-conquer paradigm, where it recursively partitions the array by selecting a pivot element (Aslam et al., 2016):

- Divide: Select a pivot element from the array. Rearrange the array elements so that all elements less than the pivot come before it, and all elements greater than the pivot come after it (this is called partitioning).
- Conquer: Recursively apply the above step to the sub-arrays formed by partitioning.
- Combine: As the base case of the recursion, arrays of size 0 or 1 are already sorted, so no further action is needed.

On the other hand, Python's built-in sort function typically uses an algorithm like Timsort, which is a hybrid sorting algorithm derived from merge sort (divide-and-conquer paradigm) and insertion sort (builds the final sorted array one item at a time) (Zhang, 2018).

Research has shown that Timsort, which is the default sorting algorithm for Python and Java, can be outperformed by certain optimized algorithms like Quick sort. (Kristo et al., 2020). Moreover, when considering large data sizes, experimental results have shown that Quick sort is more efficient than other sorting algorithms like Selection Sort, Bubble Sort, and Insertion Sort (Frak et al., 2018).

These are the results I have obtained:

Execution time of Quick Sort: 1.3984489440917969 seconds

Execution time of Built-In Sort: 0.2834312915802002 seconds

The system used to run the application is quite powerful as the CPU is a Ryzen 9 5900X with 12 cores and 24 threads, but I have obtained comparable results with a MacBook Pro with an M1 Pro processor.

In this case the Built-In Sort algorithm has been quicker than the chosen Quick Sort algorithm, the main reason is the quantity of numbers contained in the pidata.txt file. A study comparing sorting algorithms found that TimSort is the best algorithm for N (the number of elements) greater than or equal to 64 (Jmaa et al., 2019).

In conclusion, while Quick Sort is renowned for its speed and efficiency, Python's built-in sort function offers a better balance between performance and versatility for general-purpose sorting tasks. The choice between Quick Sort and Python's built-in sorting algorithm would depend on the specific requirements of the sorting task at hand, always considering factors such as data size, time complexity, and memory constraints that are going to be crucial for the right selection of the best algorithm.

Comparing my results with the given examples, both Quicksort and the built-in sorting algorithm are much quicker than the Bubble sort algorithm showed during the seminar. The algorithm used during the demonstration, Bubble sort, has multiple times been defined as a simple and inefficient algorithm (Goodhill & Xu, 2005), not fitting the purpose

for the given amount of data.

Hash Function

For the second part of this exercise, I have used the hashlib.md5 function to create an MD5 hash for the two different files that have been generated. An MD5 (Message-Digest Algorithm 5) hash is a cryptographic hash function that generates a 128-bit hash value, typically expressed as a 32-character hexadecimal number. This hash function, designed by Ron Rivest as an enhancement of the previous MD4, was commonly used for data integrity verification in various fields.
Despite MD5 is known for its cryptographic weaknesses, it continues to be a prevalent choice for ensuring data integrity in various applications such as database partitioning and computing checksums to validate file transfers (Black et al., 2006).
An evolution of MD5 can be found in the SHA-2 and SHA-3 families of hash functions that are more secure and less vulnerable to collision attacks (Pham et al., 2022).

The first file, with only the sorted numbers (sortedNumbers.txt), has generated this hash value: d0a644cd3ec9d6e39129add82f91ba98

The second file containing both the sorted numbers and my details (MatteoMarchi17-06-24.txt) has generated this hash value: e34bcee9b570fbb0ab89e014b6b5c22b

As we can clearly see the two hashes are different, the reason is the MD5 hash produces a unique hash value for different input data (Rasjid, 2019). Even the smallest difference between the two files would have generated a unique hash value.

Raschka, S., Patterson, J., & Nolet, C. (2020). Machine learning in python: main developments and technology trends in data science, machine learning, and artificial intelligence. Information, 11(4), 193. https://doi.org/10.3390/info11040193

Zhang, Y. and Zhang, H. (2023). Construction of financial big data analysis platform in universities under big data technology. Proceedings of the 2nd International Conference

on Information, Control and Automation, ICICA 2022, December 2-4, 2022, Chongqi. https://doi.org/10.4108/eai.2-12-2022.2328061

Hossain, M., Mondal, S., Ali, R., & Hasan, M. (2020). Optimizing complexity of quick sort., 329-339. https://doi.org/10.1007/978-981-15-6648-6_26

Aslam, A., Ansari, M., & Varshney, S. (2016). Non-partitioning merge-sort. https://doi.org/10.1145/2905055.2905092

Zhang, Y., Zhao, Y., & Sanan, D. (2018). A verified timsort c implementation in Isabelle/Hol. https://doi.org/10.48550/arxiv.1812.03318

Kristo, A., Vaidya, K., Çetintemel, U., Misra, S., & Kraska, T. (2020). The case for a learned sorting algorithm. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. https://doi.org/10.1145/3318464.3389752

Frak, A. N., Saringat, M. Z., Prasetyo, Y. A., Aman, H., & Ibrahim, N. H. (2018). Comparison study of sorting techniques in static data structure. International Journal of Integrated Engineering, 10(6). https://doi.org/10.30880/ijie.2018.10.06.014

Jmaa, Y. B., Atitallah, R. B., Duvivier, D., & Jemaa, M. B. (2019). A comparative study of sorting algorithms with fpga acceleration by high level synthesis. Computación Y Sistemas, 23(1). https://doi.org/10.13053/cys-23-1-2999

Goodhill, G. J. and Xu, J. (2005). The development of retinotectal maps: a review of models based on molecular gradients. Network: Computation in Neural Systems, 16(1), 5-34. https://doi.org/10.1080/09548980500254654

Black, J., Cochran, M., & Highland, T. (2006). A study of the md5 attacks: insights and improvements. Fast Software Encryption, 262-277. https://doi.org/10.1007/11799313_17

Rasjid Zulfany Erlisa, Gunawan Witjaksono, Benfano Soewito, Edi Abdurahman et al., 2019. "Timestamp injected cryptographic hash function to reduce fabrication of hash collisions", International Journal of Recent Technology and Engineering (IJRTE)(4), 8:5568-5575. https://doi.org/10.35940/ijrte.b2169.118419

Black, J., Cochran, M., & Highland, T. (2006). A study of the md5 attacks: insights and improvements. Fast Software Encryption, 262-277. https://doi.org/10.1007/11799313_17

Pham, H. L., Tran, T. H., Le, V. T. D., & Nakashima, Y. (2022). A high-efficiency fpga-based multimode sha-2 accelerator. IEEE Access, 10, 11830-11845. https://doi.org/10.1109/access.2022.3146148