K. Tavina Thisak Gunawardana GNumber:-G21097584

21/01/2024

Word count: 734

Module Name:-Computational Thinking

a) In straightforward terms, Big O Notation articulates algorithms' time complexity or performance. Its primary utility lies in distinguishing the efficiency of different algorithms across various scenarios, encompassing their worst, best, and average cases (Krone et al., 2003). It provides a standardized and concise representation of an algorithm's scalability as the input size varies. (Gupta et al., 2012). Big O notation enables the classification and comparison of algorithms, allowing researchers to differentiate their relative performance characteristics regardless of hardware or implementation complexities. The symbol "O" symbolizes the algorithm's time complexity and provides a high-level understanding of how the algorithm scales. At the same time, the variable "n" signifies the size of the data set manipulated by the algorithm (Jadoon et al., no date).

b) This part initiates a comprehensive exploration of Selection Sort, Merge Sort, and Quick Sort algorithms through an in-depth review of reputable journals. The analysis will present original findings and empirical results, featuring a meticulous examination of three datasets. The performance characteristics of the algorithms are quantified and visually depicted using line charts and bar charts. Determining the fastest algorithm becomes complex due to varying efficiencies across different scenarios. In average-case scenarios, selection sort has a time complexity of O (n2), merge sort operates at O (n log n), and quick sort also maintains O(n log n) (Al-Kharabsheh et al., 2013). Conversely, in worst-case scenarios, selection sort and quick sort both exhibit O(n2) time complexity, while merge sort remains at O (n log n) (Al-Kharabsheh et al., 2013). The best-case scenario of quick sort demonstrates a similar performance profile to the average-case scenario. (Anwar, N., 2016). In average scenarios, quick sort performs best, while merge sort takes the lead in worst-case and best-case scenarios (Al-Kharabsheh et al., 2013).

I present results from three datasets of hundreds, thousands, and ten thousand numbers.

Table 1. Execution times of all sorting algorithms

|  |  |  |  |
| --- | --- | --- | --- |
|  | Selection Sort | Merge Sort | Quick Sort |
| 100 Data Set Execution Time(m/s) | 1.3903 | 0.0 | 0.0 |
| 1000 Data Set Execution Time(m/s) | 20.7886 | 0.9984 | 1.0004 |
| 10000 Data Set Execution Time(m/s) | 2142.9855 | 13.808 | 11.7323 |

The above table shows the execution timings for each sorting algorithm's three data sets. This data indicates that merge sort is the most efficient sorting process for small-range data sets, and quick sort is the most efficient sorting process for larger data sets.

This is the average-case scenario of quick sort with a time complexity of O(n log n). The reason for quick sort to be in the average-case scenario is that all the datasets have a random, evenly distributed numbers array, so to keep up the efficiency of the quick sort algorithm, a random number was chosen as a pivot. The reason for having random, evenly distributed lists is because, in the marking scheme, they have specified to take randomly generated unique arrays.

The time complexity of merge sort is consistently O(n log n) for all best, worst, and average cases. This makes merge sort an efficient and reliable sorting algorithm in scenarios where reliable and predictable performance is crucial. Selection sort has the average-case scenario with time complexity of o(n2) because the dataset is neither sorted nor in reverse order and has a fixed number of comparison amounts.

C)Figure 1 shows the data from the above table more concisely, allowing us to compare all algorithms’ efficiencies on the three data sets.

A graph with a blue line

Description automatically generated

Figure 1. Line Chart for all sorting algorithms

The figure above shows selection sort with a quadratic time complexity, and Quick and Merge sort with a log-linear time complexity.

The bar charts generated for each sorting algorithm show the individual performance of each algorithm with the datasets for easy comparison.

A screen shot of a graph

Description automatically generated

Figure 2. Selection Sort Bar Chart



Figure 3. Merge Sort Bar Chart



Figure 4. Quick Sort Bar Chart

In conclusion, Quick sort was the most efficient sorting algorithm in sorting the 100th and 10,000th arrays while merge sort was most efficient in sorting the 1000th array as observed in this analysis. This is because Quick sort chooses a random pivot resulting in it being less efficient in sorting smaller datasets. Merge sort came second, followed by Selection sort for the 100th and 10,000th array. They were true to their time complexities of n(log N), O(log n) and O(n^2) respectively. Observing space complexity, Selection sort was most efficient with a notation of O(1), because it does in-place sorting, they do not require extra memory allocation when sorting. Quick has a notation of n(log n), as lastly merge sort with O(n) because it uses the most resources as additional memory is used when creating a temporary array. However, Quick sort has more comparison counts than Merge sort, this could be because Merge sort does less comparisons when sorting the arrays. Overall, Quick sort would be most efficient with the fastest execution time and better space complexity.

Reference List

1. Anwar, N. (2016) Comparison study on sorting techniques in static data structure, Edu.my. Available at: http://eprints.uthm.edu.my/926/1/24p%20ANWAR%20NASER%20FRAK.pdf (Accessed: December 29, 2023).
2. Follow, S. S. (2018) Comparisons involved in modified quicksort using merge sort tree, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/comparisons-involved-modified-quicksort-using-merge-sort-tree/ (Accessed: December 24, 2023).
3. Gupta, P., Agarwal, V. and Varshney, M. (2012) Design and analysis of algorithms, PHI Learning, Delhi
4. How do I add a comparison counter to merge sort in Python? (no date) Quora. Available at: https://www.quora.com/How-do-I-add-a-comparison-counter-to-merge-sort-in-Python (Accessed: December 24, 2023)
5. Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani & Nabeel Imhammed Zanoon (2013) International Journal of Computer Science and Security

Available at: https://d1wqtxts1xzle7.cloudfront.net/83532208/research-paper-lecture-5-bubble-sort-4-libre.pdf?1649488629=&response-content-disposition=inline%3B+filename%3DReview\_on\_Sorting\_Algorithms\_A\_Comparati.pdf&Expires=1703419603&Signature=QZhZ3t34nw1Wy3H69yPuXt8bkK~YTHDqVQLKqVOKCMkNvQZu7~k16IdbDFp6HRP8rJNFTAavdeeaEjHuY48w0M21AgbF9H7KPqpar9fZpNVDmDlSFigXHr7o91BUAJD2U0LObF819zeq2J-W8dUCRWEeTNtNrkcI-j0hgrMLlVwENreAMVnfcyC7YAb~~zwscANrkLkfCMU7IEfQaX2XglHo0iRev5mC8LaWHIX0Vc1HcrZWrhRP8ZqmgeFafkH4cbPPtADs1BfYUN2etYEy9uANg8BZeOw096P9y1dDpjPcaXF9bNbRDCzLiYTP8CjM90JlkEUDyKOjtCkD8-pqwA\_\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA (Accessed: December 29, 2023).

1. Krone, J., Ogden, W. F. and Sitaraman, M. (2003) OO Big O: A Sensitive Notation for Software Engineering, Clemson.edu. Available at: https://computing.clemson.edu/resolve/research/reports/RSRG-03-06.pdf (Accessed: December 28, 2023).
2. Merge sort (with code in Python/C++/Java/C) (no date) Programiz.com. Available at: https://www.programiz.com/dsa/merge-sort (Accessed: November 24, 2023)
3. Jadoon, S. et al. (no date) Design and analysis of optimized selection sort algorithm, Psu.edu. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0047d34b344c05ffa4c0e94ab134d58546a7c6c2 (Accessed: December 28, 2023).
4. jeeteshgavande, J. (2020) Bar plot in matplotlib, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/bar-plot-in-matplotlib/ (Accessed: December 29, 2023).
5. prernadec , P. (2020) Line chart in matplotlib - python, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/line-chart-in-matplotlib-python/ (Accessed: December 29, 2023)
6. QuickSort (with code in Python/C++/Java/C) (no date) Programiz.com. Available at: https://www.programiz.com/dsa/quick-sort (Accessed: December 12, 2023).

Bibliography

1. GeeksforGeeks (2023) 'Time and Space Complexity Analysis of Selection Sort', GeeksforGeeks. Available at: https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-selection-sort/. (Accessed: 25 December 2023).
2. Huang, S. (2022) *What is big O notation explained: Space and time complexity*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/ (Accessed: 26 December 2023).
3. JavaTpoint (no date) 'Quick Sort', www.javatpoint.com. Available at: https://www.javatpoint.com/quicksort#:~:text=Best%20Case%20Complexity%20%2D%20In%20Quicksort,O(n\*logn). (Accessed: 25 December 2023).
4. Nero, R. del (2023) *Big (O) notation explanation*, *Java Challengers*. Available at: https://javachallengers.com/big-o-notation-explanation/ (Accessed: 26 December 2023).
5. Nielsen, J. (2020) *Big O log-linear time complexity*, *jarednielsencom RSS*. Available at: https://jarednielsen.com/big-o-log-linear-time-complexity/ (Accessed: 19 January 2024).
6. Simplilearn (2023) 'Big O Notation in Data Structure: An Introduction', Simplilearn.com. Available at: https://www.simplilearn.com/big-o-notation-in-data-structure article#:~:text=Big%20O%20Notation%20is%20a%20tool%20used%20to%20describe%20the,bound%20of%20an%20algorithm’s%20runtime. (Accessed: 25 December 2023).
7. ‌Vittal (2021). *Time Complexity Analysis*. [online] Medium. Available at: https://medium.com/@vittal.a.badami0107/time-complexity-analysis-2fe312bfcfa3 [Accessed 17 Jan. 2024].