International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 09 | Sep 2023 www.irjet.net p-ISSN: 2395-0072

A Plagiarism Checker: Analysis of time and space complexity

Mr. Prashanth Kumar HM[1] & Dr. Subramanya Bhat S[2]

[1]Research Scholar, College of Computer Science, Srinivas University, Mangalore, India
[2] Professor, College of Computer Science, Srinivas University, Mangalore, India

---

Abstract - In Plagiarism processing, the time and space complexity are important concepts in software internal process, that describe the efficiency and resource requirements of algorithms and programs. The time of the plagiarism process called time complexity, refers to the amount of time an algorithm takes to complete its document execution as a function of the input size. It provides an upper bound on the number of basic operations (such as searching data, matching, arithmetic operations, indexing etc.) an algorithm performs in relation to the input size. Plagiarism time complexity is usually expressed using big O notation, which represents the worst-case scenario. Space complexity refers to the amount of memory space an algorithm or program uses to solve a problem as a function of the input size. It includes the memory required for variables, data structures, recursion call stacks, and other memory allocations during the algorithm's execution. Like time complexity, space complexity is often expressed using big O notation and describes the worst-case scenario. It's important to note that time and space complexity are interrelated but distinct concepts. An algorithm that is very time-efficient might have a higher space complexity, and vice versa. When designing algorithms or choosing between different algorithms to solve a problem, you often aim to strike a balance between time and space complexity based on the specific requirements and constraints of the problem and the underlying hardware. Here we expose accuracy and fast execution of plagiarism document checking is dependent on two complexity levels.

Key Words: Time Complexity, Space Complexity, Searching, File Processing, Indexing.

1. INTRODUCTION

In plagiarism, the accuracy refers to the degree to which a software system produces correct and reliable results that align with the intended functionality and result expectations. In the realm of software development, achieving high accuracy is a fundamental goal, as inaccuracies can lead to errors, misinformation, financial losses, compromised security, and user dissatisfaction. Ensuring software accuracy involves a combination of thorough design, meticulous testing, continuous validation, and diligent maintenance. In our process relies heavily on plagiarism to perform an array of tasks, from basic calculations to complex data analysis, critical infrastructure management, healthcare diagnostics, financial transactions, and more. As such, the accuracy of these systems holds immense.

Significance in various domains. Accuracy starts with clear and comprehensive requirements gathering. An accurate understanding of user needs, and system functionality is essential to building software that meets expectations. A well-thought-out design phase of plagiarism considers potential sources of errors and inefficiencies, aiming to minimize them. Design decisions influence how data is stored, processed, and communicated within the software, directly affecting its accuracy. Skilled coding practices and adherence to design principles are crucial for translating a design into accurate software. Inaccurate coding can introduce bugs, security vulnerabilities, and unexpected behaviors. Rigorous testing is a cornerstone of ensuring software accuracy. Then testing includes unit tests to examine individual components, integration tests to assess interactions between components, and system-level tests to verify the software. The validation covers whether the software satisfies user needs, while verification confirms that the software meets its specified requirements. These processes involve comparing plagiarism outputs to expected results and validating its behavior under various input documents. The result of delay or failure often interacts with software in ways unforeseen by developers, leading to the discovery of issues and improvements that enhance accuracy. Accuracy is an ongoing effort. Regular updates, bug fixes, and security patches are essential to maintaining accuracy as the software evolves and new challenges arise. The challenges and strategies for ensuring plagiarism accuracy are defined in different levels, but both levels are dependent on duration of process and memory usage. Software systems are becoming increasingly complex. Managing this complexity requires careful design and modularization to reduce the likelihood of errors and ease debugging. Inaccurate input data can lead to inaccurate outputs. Implementing data validation and error-handling mechanisms can help mitigate this issue. In multi-threaded or distributed systems, managing concurrency and ensuring accuracy in shared data access is a significant challenge. Proper synchronization and coordination mechanisms are necessary. Security breaches can compromise the accuracy and integrity of software systems. Implementing strong security measures, encryption, and authentication helps maintain accurate data and functionality. Employing a mix of testing strategies, including automated testing, manual testing, and exploratory testing, can help uncover different

types of issues and improve overall accuracy. Monitoring software performance and behavior in real-world scenarios helps identify and address accuracy issues as they arise. Comprehensive documentation aids in understanding the software's functionality, inputs, outputs, and potential limitations, facilitating accurate usage and troubleshooting. Software accuracy is a critical aspect of software development that directly impacts its usefulness, reliability, and user satisfaction. Achieving and maintaining accuracy involves a holistic approach that encompasses every phase of the software development lifecycle, from requirements gathering to maintenance. Through meticulous design, rigorous testing, continuous validation, and ongoing improvement efforts, software developers strive to deliver accurate systems that meet user needs and contribute positively to various aspects of society. Time complexity is a fundamental concept in computer science that measures the efficiency of an algorithm by analyzing the amount of time it takes to run as a function of the input size. It provides a way to compare and classify algorithms based on their performance characteristics. In essence, time complexity quantifies the relationship between the input size and the number of basic operations an algorithm performs. These operations could include comparisons, assignments, arithmetic computations, and other fundamental actions. By understanding how an algorithm's execution time scales with larger inputs, we can make predictions about its performance in real-world scenarios. Time complexity is commonly expressed using big O notation, which offers a simplified representation of an algorithm's upper bound performance. For instance, an algorithm with O(1) time complexity exhibits constant time behavior; its execution time remains consistent, regardless of input size. This is often seen in operations where a fixed number of steps are executed. Algorithms with logarithmic time complexity, denoted as O(log n), are notably efficient for large datasets. They divide the input into smaller portions with each step, reducing the problem size significantly. Binary search is a classic example of such an algorithm, as it divides the search space in half during each iteration. Linear time complexity (O(n)) implies that an algorithm's execution time grows linearly with the input size. For instance, iterating through an array to find a specific element requires a number of operations proportional to the array's length. Quadratic time complexity (O(n^2)) signifies that an algorithm's execution time increases quadratically with input size. This is often observed in nested loops, where each iteration of the outer loop triggers a complete run of the inner loop. Such algorithms can become inefficient for larger inputs. As time complexity increases further (e.g., cubic, exponential), algorithms become progressively less efficient and suitable for practical use due to their rapid growth in execution time. Selecting the right algorithm for a specific problem involves analyzing its time complexity and considering the trade-offs between different approaches. An algorithm that excels in time efficiency might require higher memory usage, leading to increased space complexity. It's a balancing act that considers the constraints of the problem, available hardware resources, and the desired performance. In our plagiarism time complexity provides a valuable framework for understanding an algorithm's performance characteristics as the input size changes. It aids in making informed decisions and updating code about implementation part, design, and optimization, ultimately contributing to the development of efficient and effective solutions for respected documents.

2. OBJECTIVES

File Conversion: Converting a document from one word processing format (like Microsoft Word's .docx) to another (like Adobe PDF). File conversion can be performed at an initial level of plagiarism processing step. File conversion is available where the user can upload a file in any format, and we have it converted to a single format for plagiarism similarity matching purpose without installing additional software using some document converted algorithm. When performing file conversions, it's important to ensure that the quality and content of the original file are retained to the greatest extent possible in the converted file.

Data Extraction: Data extraction refers to the process of retrieving specific information or data from a larger dataset or source. This process is commonly used to gather relevant data for analysis, reporting, transformation, or migration. Data extraction can involve a wide range of sources, such as databases, websites, documents, spreadsheets, and more. There are different methods for data extraction, depending on the source and the complexity of the data. Some common methods include querying databases using SQL, using APIs to retrieve structured data, using web scraping tools to extract information from websites, and using data integration tools for more complex extraction tasks.

Internet Search: Internet searching, also called programmable content searching, refers to the act of using API search engines and other user defined tools to find information, resources, websites, and content on the World Wide Web. It's the process of entering keywords, phrases, or questions into a search engine and receiving a list of relevant results that match the query. Internet searching is a fundamental skill in the digital age, enabling individuals to access a vast array of information quickly and easily. It's used for everything from looking up simple facts to conducting in-depth research on complex topics.

Data Indexing: Data indexing is the process of organizing and optimizing the structure of data to improve the speed and efficiency of data retrieval operations. Indexing involves creating data structures that allow for rapid access to specific pieces of information within a dataset, making data retrieval faster and more efficient compared to scanning the entire dataset. Data indexing is a crucial technique in database management and other data storage systems to ensure efficient data retrieval. It involves creating and managing data structures that provide quick access to

specific data entries, contributing to better overall system performance.

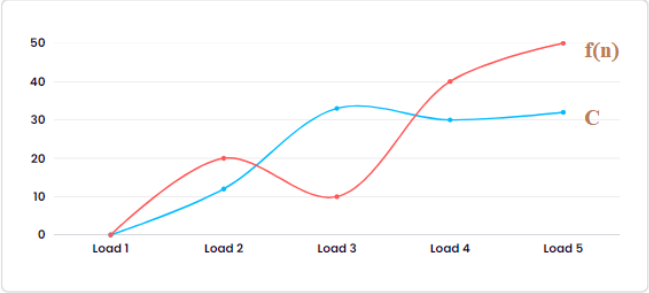
Comparison: Text comparison refers to the process of comparing two or more pieces of text to identify similarities, differences, or patterns between them. Text comparison is often used in various levels in our plagiarism detection and internal content management. in our Plagiarism comparing texts at the word level involves identifying similarities and differences between words. This is common in natural language processing tasks and document analysis.

Text Highlighting: It's a process of showing detected matched content in a same user document and called overlapping matched text. Highlighted text in document showing different color system in background. The two levels of document we are generating here, those html and pdf colored document. Advantage of this process is user can easily to navigate the highlighted content which is showed like similarity content.

Reporting: Reporting in a document refers to the process of presenting information, data, or findings in a structured and organized manner. It involves creating written or visual documents that convey specific information to a particular audience, often with the purpose of informing, analyzing, or making decisions based on the presented information. Effective reporting requires clear and concise communication, accurate data representation, proper context, and consideration of the target audience's needs and expectations. Well-structured reports make it easier for readers to understand the information presented and make informed decisions based on that information.

3. ANALYZING TIME

Omega Notation: Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm. The execution time serves as a lower bound on the algorithm's time complexity. It is defined as the condition that allows an algorithm to complete statement execution in the shortest amount of time.:



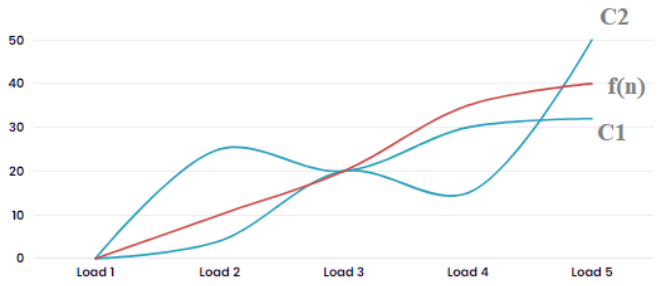
Let C and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that c ≤ f(n) for all n ≥ n0.

In our assumption, all process is done by the strength of CPU is 4 core and RAM capacity 8 giga byte of virtual server. Here every load duration time file can be automatically uploaded concurrently. The load duration was not fixed but it was less than 10 Sec and increased 100% of File processing load.

Process Name	File Process	Load Duration	Started	Completed	Processing	Avg. Time
Load 1	10	<10 Sec	10	10	0	10 Sec
Load 2	20	<10 Sec	20	20	0	20 Sec
Load 3	30	<10 Sec	30	30	0	30 Sec
Load 4	40	<10 Sec	40	40	0	40 Sec
Load 5	50	<10 Sec	50	50	0	50 Sec

The outcomes of this experiment expressed by Omega Notation c1*g(n) <= f(n), hence c1 value is minimum of load value. So, value expressed by mathematically called 10 <=50 <= f(n), so average outcome is Ω (g)=f(n), so Ω (g)=31, here 31 is file process, finally we can say according to our virtual system assumption, every less than 10 Sec we can process 31 files (best Case) concurrently (1 file=20 pages, or 4,500 words).

Theta Notation: Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm. Theta (Average Case) You add the running times for each possible input combination and take the average in the average case. Here we are analyzing 'theta' notation of overall time consumption between upload file from user end and until result of similarity of the document.



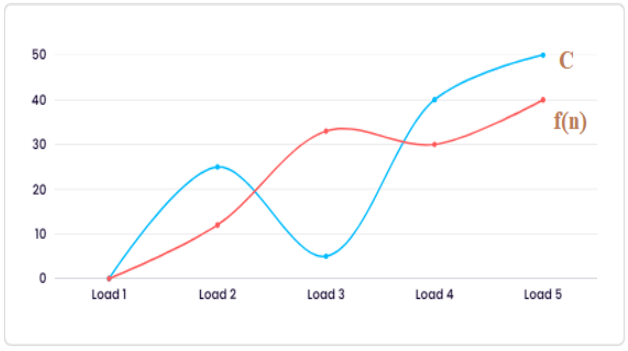
The above expression can be described as if f(n) is theta of g(n), then the value f(n) is always between c1 * g(n) and c2 * g(n) for large values of n (n ≥ n0). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0. In our assumption, all process is done by the strength of CPU is 4 core and RAM capacity 8 giga byte of virtual server. Here every load duration time file can be automatically uploaded concurrently. The load duration fixed every 10 Sec and increased 100% of File processing load.

Process Name	File Process	Load Duration	Started	Completed	Processing	Avg. Time
Load 1	10	10 Sec	10	10	0	10 Sec
Load 2	20	10 Sec	20	18	2	26 Sec
Load 3	30	10 Sec	32	26	6	20 Sec
Load 4	40	10 Sec	46	37	9	36 Sec
Load 5	50	10 Sec	59	48	11	50 Sec

Process Name	File Process	Load Duration	Started	Completed	Processing	Avg. Time
Load 1	10	10 Sec	10	10	0	10 Sec
Load 2	20	10 Sec	25	12	13	25 Sec
Load 3	30	10 Sec	48	33	15	48 Sec
Load 4	40	10 Sec	55	30	25	55 Sec
Load 5	50	10 Sec	75	40	35	75 Sec

The outcomes of this experiment expressed by Theta Notation c1*g(n) <= f(n) <= c2*g(n), hence c1 value is minimum of load value and c2 value is maximum of load value. So, value expressed by mathematically called 10 <= f(n) <= 50, so average outcome is Θ(g)=f(n), so Θ(g)=20, here 20 is file process, finally we can say according to our virtual system assumption every 10 Sec we can process 20 files concurrently (1 file=20 pages, or 4,500 words).

Big O Notation: Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm. It is the most widely used notation for Asymptotic analysis. It specifies the upper bound of a function. The maximum time required by an algorithm or the worst-case time complexity. It returns the highest possible output value(big-O) for a given input. Big-O (Worst Case) It is defined as the condition that allows an algorithm to complete statement execution in the longest amount of time possible.

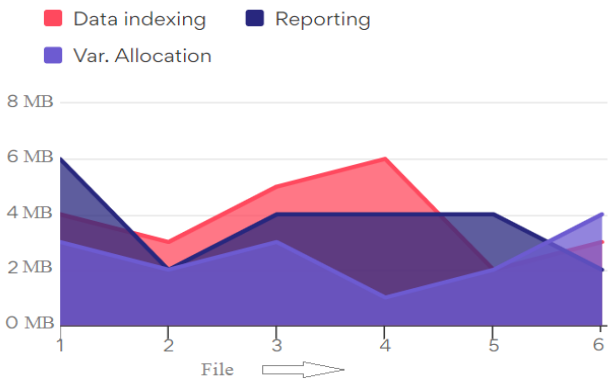


If f(n) describes the running time of an algorithm, f(n) is O(g(n)) if there exist a positive constant C and n such that, 0 ≤ f(n) ≤ cg(n) for all n ≥ n0. It returns the highest possible output value (big-O) for a given input. The execution time serves as an upper bound on the algorithm's time complexity. According to our above virtual server assumption, we processed files with below values.

The outcomes of this experiment expressed by Big O Notation f(n) <= c*g(n), hence c value is minimum of load value. So, value expressed by mathematically called 0 <= f(n) <= 50, so average outcome is O(g)=f(n), so O(g)=30, here 30 is pending or processing file after load duration.

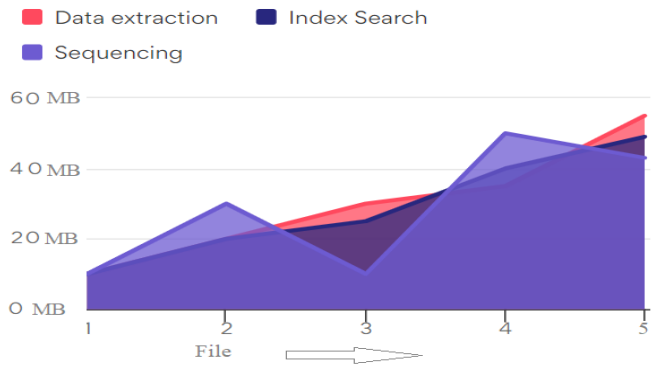
4. ANALYZING SPACE/MEMORY

Constant: We consider constant space complexity when the program doesn't contain any loop, recursive function, or call to any other functions. It's also called O (1) Complexity and auxiliary space is the extra/temporary space used by an algorithm.



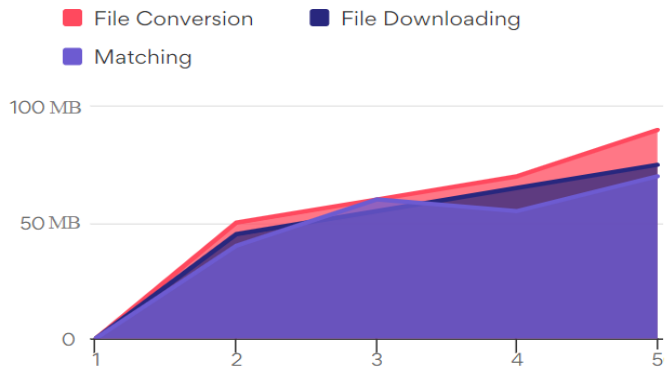
In our plagiarism checking software we are categorized at different levels of the process. Every process has a different looping and recursive stages, some stages execute a completed loop execution and some others just memory allocation using complier. In constant space complexity we summarize three main levels called data indexing, reporting and variable allocation. Those operations are observed during similarity check operation, and allocation thread is 1.

Linear: We consider Linear space complexity when the program contains only a single iterative loop or single recursive function which is not over more than 5 levels (Ex: factorial of n, where n=4 means, 4!=4*3*2*1 operation). It's also called O (n) where n means number of looping levels.



The complexity and auxiliary space are the extra/temporary space used by an algorithm. In linear space complexity we summarize three main levels called data extraction, index searching and sequence allocation or sequencing. Those operations are observed during similarity check operation, and allocation thread is 5 with respect to file size is 5 with delay of 0.5 millisecond.

Quadratic: We consider quadratic space complexity when the program contains an infinite iterative loop or multilevel recursive function which is over more than 5 level (Ex: complex operation). It's also called O (n^2) where n means multiple of looping levels.



The complexity and auxiliary space are the extra/temporary space used by an algorithm. In quadratic space complexity we summarize three main levels called file conversion which is having text with complex images, graphs or special symbols, file downloading and matching process. Those operations are observed during similarity check operation, and allocation thread is 5 with respect to file size is 25 with delay of 1 millisecond.

5. CONCLUSION

Time and space complexity are essential concepts in the field of plagiarism checker analysis. These measures help us understand the efficiency and resource requirements of algorithms and programs. Time complexity allows us to estimate how plagiarism algorithm's execution time grows with input size. It helps us make informed decisions about selecting the most efficient algorithm for a given problem and predicts how well an algorithm will perform as data scales. Space complexity, on the other hand, helps us evaluate an algorithm's memory usage, which is crucial in resource-constrained environments. Understanding an algorithm's space complexity can lead to optimized memory management and more efficient program execution. Balancing time and space complexity is often a trade-off, as reducing one may increase the other. Therefore, it's essential to consider both aspects when designing and analyzing algorithms.

REFERENCES

1]. Aho Alfred, V. and John E. Hopcroft, "*The design and analysis of computer algorithms*", Pearson Education, 1974.

2]. Wikipedia article: Counting sort, [online] Available: http://en.wikipedia.org/wiki/Counting_sort.

3]. Cederman Daniel and Philippas Tsigas, "*Gpu-quicksort: A practical quicksort algorithm for graphics processors*", Journal of Experimental Algorithmics (JEA), vol. 14, 2009.

4]. Geeksforgeeks article: Sorting Algorithms Code, [online] Available: https://www.geeksforgeeks.org/sorting-algorithms.

5]. W. Sun and Z. Ma, "*Count Sort for GPU Computing*", 2009 15th International Conference on Parallel and Distributed Systems, pp. 919-924, 2009.

6]. Faujdar Neetu and Satya Prakash Ghrera, "*Performance evaluation of parallel count sort using GPU computing with CUDA*", Indian Journal of Science and Technology, vol. 9, 2016.

7]. T. Umeda and S. Oya, "*Performance Comparison of Open-Source Parallel Sorting with OpenMP*", 2015 Third International Symposium on Computing and Networking (CANDAR), pp. 334-340, 2015.

8]. Han Yijie, "*Deterministic sorting in O (n log log n) time and linear space*", Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 602-608, 2002.

9]. Wikipedia article: Integers, [online] Available: https://en.wikipedia.org/wiki/.

10]. Y. Li, F. Sha, S. Wang and T. Hu, "*The improvement of page sorting algorithm for music users in Nutch*", 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), pp. 1-4, 2016.

11]. Yao Yuan, Virtual Memory Streaming and Sorting in MapReduce Applications, 2018.