Journal of Applied Mathematics and Computation, 2022, 6(1), 1-3
https://www.hillpublisher.com/journals/JAMC/
ISSN Online: 2576-0653
ISSN Print: 2576-0645

The Big-O of Mathematics and Computer Science

Firdous Ahmad Mala[1,*], Rouf Ali[2]

[1]Department of Mathematics, Govt. Degree College Sopore, J&K, India.
[2]Department of Computer Applications, Govt. Degree College Sopore, J&K, India.

How to cite this paper: Firdous Ahmad Mala, Rouf Ali. (2022) The Big-O of Mathematics and Computer Science. *Journal of Applied Mathematics and Computation*, 6(1), 1-3.
DOI: 10.26855/jamc.2022.03.001

Received: November 23, 2021
Accepted: December 18, 2021
Published: January 4, 2022

[*]**Corresponding author:** Firdous Ahmad Mala, Department of Mathematics, Govt. Degree College Sopore, J&K, India.
Email: firdousmala@gmail.com

Abstract

In this paper, we review the basic notion of the Big-O notation, also known as the Bachmann-Landau notation, that is frequent and prevalent in the study of the computational complexity of algorithms. Though this notation has been in use for quite some time, the authors feel that there is still scope for some literature in this direction. And this paper is an attempt for the same. We start with the discussion of a very brief chronological history of the various attempts made at understanding and calculating the computational complexities of algorithms. We show, using examples, how the Big-O notation could turn out to be a better, easier and more informative notation compared to the limit notation of a function. We point out some common Big-O orders that one comes across during the evaluation of the computational complexities of algorithms and functions. Finally, we show how the Big-O notation follows transitivity.

Keywords

Big-O notation, computational complexity, Analysis of Algorithms, Limits of functions

1. Introduction

The history of what and how attempts were made at the optimization of arithmetic algorithms can be traced back to the Middle Ages. Methods employed to reduce the number of steps needed for various calculations can be found in the work of Ibn al-Majdi, a fourteenth century Egyptian astronomer [1].

Regarding the analysis of the Euclidean algorithm, Gabriel Lamé proved, in 1844, that if $\alpha > \beta > 0$, then the number of steps required for division employed in the Euclidean algorithm $E(\alpha, \beta)$ is, somewhat invariable, and less than five times the number of digits. Later on, several improvements were made regarding the number of required steps in the Euclidean algorithm [2].

It is pertinent to mention that algorithmic thinking has been studied in connection with geometrical constructions since antiquity. In fact, in the words of Schreiber, "the first studies on the unsolvability of problems by special instruments (that is, by special classes of algorithms) and the first attempts to measure, compare, and optimize the complexity of different algorithms for solving the same problem were in the field of geometrical constructions" [3].

Later, in 1864, it was Charles Babbage who predicted the importance of the analysis and study of algorithms. In his words, "As soon as an Analytical Engine (that is, a general purpose computer) exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise—By what course of calculation can these results be arrived at by the machine in the shortest time?" [4].

The advent of a Turing machine, in 1937, raised a lot of eyebrows. The question of which problems could be or could not be solved by a computer came to the fore. More specifically, it led to the question of the relative computational difficulty of computational functions that is now the part of computational complexity. The question of exactly what it meant to say that a function is more difficult to compute in comparison to another one was first addressed, in 1960, by Michael

Rabin in his paper “Degree of Difficulty of Computing a Function and Hierarchy of Recursive Sets” [5].

2. Main Discussion

To compare efficiencies of competing algorithms in context of a given problem, it seems essential to consider the number of operations performed by each algorithm, especially for large inputs. This is carried out by classifying and comparing the growth rates of the complexity function of each algorithm.

The Big-O notation, that was introduced by the German mathematician, Paul Bachmann in 1894, has been used extensively in connection with the analyses of algorithms to understand and appreciate the order of growth of a complexity function [6].

In particular, Big-O gives an upper bound on the order of growth of a function.

3. Definition

For functions f and g defined on some common domain D, we write $f(x) = O\big(g(x)\big)$ as $x \to \infty$ to mean that there exists some positive number k and some $x_0 \in D$ such that $|f(x)| \le k\, g(x), \forall x \ge x_0$.

In other words, this simply means that the absolute value of $f(x)$ is at most a positive constant multiple of $g(x)$ for all sufficiently large $x \in D$. The Big-O notation is also known as the Bachmann-Landau notation.

Consider, for example, the function $f: \mathbb{R} \to \mathbb{R}$ defined by $f(x) = x^6 - x^2 + x - 1$ and the function $g: \mathbb{R} \to \mathbb{R}$ defined by $g(x) = x^6$.

Clearly, for all $x \ge 1$, $|f(x)| = |x^6 - x^2 + x - 1| \le |x|^6 + |x|^2 + |x| + 1 \le x^6 + x^6 + x^6 + x^6 = 4x^6$ so that $|f(x)| \le 4\, g(x)$. Consequently, $f(x) = O\big(g(x)\big)$.

In the context of the analysis of algorithms, the domain of all the functions under consideration will be a subset of the set of positive integers. This is because the variable with invariably be the input size or the discrete time variable.

Consider, for example, $f: \mathbb{N} \to \mathbb{N}$ given by $f(n) = \sum_{r=1}^{n} (2r - 1)$.

Clearly, $f(n) = 1 + 3 + 5 + \cdots + (2n - 1) = n^2$ so that $f(n) = O(n^2)$.

Similarly, if $u(n) = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n}$, then using the fact that $\sum_{r=0}^{n} \binom{n}{r} = 2^n$, we conclude that $u(n) = O(2^n)$.

4. Big-O compared to the limit notation

In comparison with the limit notation of functions, it suppresses lesser information and is easy to manipulate. In the words of N. G. de Brujin, “It does not suppress a function, but only a number. That is to say, it replaces the knowledge of a number with certain properties by the knowledge that such a number exists. The O notation suppresses much less information than the limit notation, and yet it is easy enough to handle” [7].

To appreciate what Brujin said, consider the case of a sequence $\{f_n\}$ that satisfies $|f_n - 1| \le n^{-1}, \forall n \in \mathbb{N}$. In the limit sense of functions, we have $f_n \to 1$ as $n \to \infty$ or $\lim_{n\to\infty} f_n = 1$. In the $\epsilon - \delta$ sense of a function, it means that for every $\epsilon > 0$, $\exists m \in \mathbb{N}$ such that $|f_n - 1| < \epsilon$ whenever $n \ge m$. In terms of the Big-O notation, $f_n - 1 = O(n^{-1})$. Notice how the Big-O notation mentions not only the fact that the function is sort of dominated by another function, but it also mentions, without scale, the function that dominates f.

To appreciate how much more informative the Big-O notation is in comparison to the limit notation of a function, observe that if a sequence $\{g_n\}$ satisfies $|g_n - 1| \le n^{-2}$, then $g_n \to 1$as $n \to \infty$ or $\lim_{n\to\infty} g_n = 1$. However, in terms of the Big-O notation, $g_n - 1 = O(n^{-2})$.

Thus, the sequences $\{f_n\}$ and $\{g_n\}$, despite being different, have the same limit but different orders.

As recent work in the direction of computation of complexities and the Bachmann-Landau notation, there is good literature out there [8-11].

5. Some Special Cases

$O(1)$: If a function f is $O(1)$, it would simply mean that there exists some $M \in \mathbb{R}$ such that $\forall n \ge n_0, f(n) \le M$. This simply means that the function f is bounded on its domain. Examples include $f(x) = e^{-n}, n \in \mathbb{N}$. Also, hash tables are $O(1)$ that means that a hashed key would take one directly to what one is looking for.

$O(n)$: If a function f is $O(n)$, it would simply mean that there exists some $M \in \mathbb{R}$ such that $\forall n \ge n_0, f(n) \le M\, n$. This simply means that the function f is dominated by a linear function. Examples include $f(n) = 3n + 2, n \in \mathbb{N}$. Scanning a list is $O(n)$ for one needs to visit each item in the list separately.

$\mathbf{O(n^2)}$: If a function f is $O(n^2)$, it would simply mean that there exists some $M \in \mathbb{R}$ such that $\forall x \geq x_0, f(x) \leq M\, n^2$. Examples include $f(n) = 1 + 2 + 3 + \cdots + n, n \in \mathbb{N}$.

$\mathbf{O(2^n)}$: If a function f is $O(2^n)$, it would simply mean that there exists some $M \in \mathbb{R}$ such that $\forall x \geq x_0, f(x) \leq M\, 2^n$. Examples include $f(n) = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n}, n \in \mathbb{N}$.

$\mathbf{O(\log n)}$: If a function f is $O(\log n)$, it would simply mean that there exists some $M \in \mathbb{R}$ such that $\forall x \geq x_0, f(x) \leq M \log n$. Examples include the binary search algorithm.

$\mathbf{O(n \log n)}$: If a function f is $O(n \log n)$, it would simply mean that there exists some $M \in \mathbb{R}$ such that $\forall x \geq x_0, f(x) \leq M\, n\log n$. Examples include sorting which is normally $O(n \log n)$.

6. Transitivity of Big-O

Theorem: The Big-O notation follows transitivity. In other words, if for functions f, g, h on some appropriate domain, $f(x) = O\big(g(x)\big)$ and $g(x) = O\big(h(x)\big)$ then $f(x) = O\big(h(x)\big)$.

Proof: Given the fact for functions f, g, h defined on an appropriate domain, if $f(x) = O\big(g(x)\big)$ and $g(x) = O\big(h(x)\big)$, then there exist numbers $M_1, M_2, x_0{}', x_0{}''$ such that $f(x) \leq M_1\, g(x), \forall x \geq x_0{}'$ and $g(x) \leq M_2\, h(x), \forall x \geq x_0{}''$.

Consequently, for $M = \max\{M_1, M_2\}$ and $x = \max\{x_0', x_0''\}$, $f(x) \leq M\, h(x), \forall x \geq x_0$.

7. Conclusion

In this paper, we reviewed, very briefly, the history that led to us to the current understanding of the computational complexity of algorithms. We discussed the concept of the Big-O notation used frequently in connection with the computation of the complexity of an algorithm. We also discussed some common Big-O orders, the usefulness of the Big-O notation over the limit notation of functions and finally observed the transitivity of the Big-O notation.

References

[1] Barbin, É., Borowczyk, J., Guillemot, M., and Michel-Pajus, A. (1999). A history of algorithms: from the pebble to the microchip (Vol. 23). J. L. Chabert (Ed.). Berlin: Springer. https://10.1007/978-3-642-18192-4.

[2] Shallit, J. (1994). Origins of the analysis of the Euclidean algorithm. Historia Mathematica, 21(4), 401-419. https://doi.org/10.1006/hmat.1994.1031.

[3] Schreiber, P. (1994). Algorithms and algorithmic thinking through the ages. Companion Encyclopedia of the History and Philosophy of the Mathematical Sciences. New York: Routledge. https://doi.org/10.4324/9780203014585.

[4] Knuth, D. E. (1976). Big omicron and big omega and big theta. ACM Sigact News, 8(2), 18-24. https://doi.org/10.1145/1008328.1008329.

[5] Rabin, M. O. (1960). Degree of difficulty of computing a function and hierarchy of recursive sets. Tech. Rep. 2, Hebrew University, Jerusalem.

[6] Rosen, K. H. and Krithivasan, K. (2012). Discrete mathematics and its applications: with combinatorics and graph theory. Tata McGraw-Hill Education.

[7] De Bruijn, N. G. (1981). Asymptotic methods in analysis (Vol. 4). Courier Corporation.

[8] Iqbal, N., Hasan, O., Siddique, U., and Awwad, F. (2019, February). Formalization of Asymptotic Notations in HOL4. In 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS) (pp. 383-387). IEEE. https://doi.org/10.1109/CCOMS.2019.8821642.

[9] Sergeev, I. (2020). On the asymptotic complexity of sorting. In Electron. Colloquium Comput. Complex. (Vol. 27, p. 96).

[10] De Micheli, G., Gaudry, P., and Pierrot, C. (2020, August). Asymptotic complexities of discrete logarithm algorithms in pairing-relevant finite fields. In Annual International Cryptology Conference (pp. 32-61). Springer, Cham. https://doi.org/10.1007/978-3-030-56880-1_2.

[11] Guéneau, A., Charguéraud, A., and Pottier, F. (2018, April). A fistful of dollars: Formalizing asymptotic complexity claims via deductive program verification. In European Symposium on Programming (pp. 533-560). Springer, Cham. https://doi.org/10.1007/978-3-319-89884-1_19.