INTERNATIONAL JOURNAL OF TECHNOLOGY AND ACADEMIC RESEARCH

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar

METHODS FOR ESTIMATING THE ASYMPTOTIC COMPLEXITY OF ALGORITHMS AND ANALYSIS OF THEIR EFFICIENCY INDICATORS

Kamalova Nilufar Ilxomovna
Associate Professor of the Department of "Applied Mathematics and Programming Technologies," Bukhara State University, Doctor of Philosophy (PhD)
Nuritdinov Rasul Ravshanovich
Student of the Department of "Applied Mathematics and Programming Technologies," Bukhara State University
https://doi.org/10.5281/zenodo.23243536

Abstract

This article examines the theoretical and practical aspects of evaluating the efficiency and resource economy of algorithms in software development. In an era of rapid growth of information technology and big data, the processing speed of software systems and the rational use of memory resources have become a matter of priority. Within the scope of the study, the content of the asymptotic analysis apparatus used to determine the time and space complexity of algorithms, namely the mathematical notations Big O, Big Omega, and Big Theta, is covered extensively.

The article provides a comparative analysis of the effect of various asymptotic classes $O(1)$,$O(\log n)$, $O(n)$, $O(\frac{n}{\log n})$, $O(n^2)$, $O(2^n)$ on algorithm performance. It is mathematically substantiated that increasing the power of hardware is not always the optimal solution when the volume of input data grows sharply. In addition, using sorting and search algorithms as examples, behavior in the worst, average, and best cases is evaluated, and scientific and practical recommendations have been developed for optimizing computational resources and selecting efficient algorithms in practical programming.

Keywords: algorithmic complexity, asymptotic analysis, Big O notation, time complexity, space complexity, algorithm optimization, computational speed.

Introduction

Today, the rapid development of information and communication technologies and the digital economy creates a need for fast and accurate processing of huge volumes of data (Big Data). Algorithms form the core of any software system, mobile application, or complex intelligent suite. In software development, it is not enough for an algorithm merely to produce the correct result; how economically and efficiently it uses computational resources is also a decisive criterion.

Determining the complexity of an algorithm resolves many pressing scientific and practical problems. Examples include hardware-independent objective evaluation, predicting behavior at large scale, optimizing resources and costs, and other tasks. The actual execution time of a program depends on the processor frequency, RAM, and operating system load of the computer used. Asymptotic estimation of algorithm complexity, by contrast, makes it possible to evaluate mathematically the conceptual perfection of an algorithm, free of technical indicators. It makes it possible to determine in advance how the running time of a program will grow when the problem size increases several-fold or millions of times. In high-load systems, an inefficient algorithm requires increasing technical capacity, which leads to higher financial costs. An optimal algorithm, on the other hand, ensures the use of available computing power with maximum efficiency.

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar

The apparatus for evaluating the time and space complexity of algorithms is not merely a theoretical concept; it serves as an important practical instrument in all major areas of modern information technology.

In particular, in high-load information systems, including the financial and banking sector, international payment platforms, and search engines, processing millions of requests arriving every second in real time relies directly on asymptotically optimal algorithms. Here, even a slight increase in algorithm complexity leads to a slowdown of the entire system and to huge financial losses.

Likewise, analyzing computational complexity is a priority in the rapidly developing fields of artificial intelligence and machine learning today. This is because, when training multidimensional arrays and neural networks, using algorithms of high complexity such as $O(n^2)$ or $O(2^n)$ can stretch computation time to days or weeks. For this reason, optimizing models and choosing fast algorithmic solutions is the key to improving model performance.

The concept of mathematical complexity also forms the theoretical foundation of cryptography and information security. Criteria of asymptotically high complexity are used in encrypting information; that is, the strength of an encryption algorithm is based on the exponential time complexity required to break and decrypt it.

In addition, algorithmic analysis plays an unparalleled role in bioinformatics and genomics. In comparing the human genome and sequences of millions of DNA/RNA strands and searching for similarities among them, the expected result can be achieved only with optimal $O(\frac{n}{\log n})$ or linear algorithms.

Finally, this knowledge is the basic foundation of competitive programming and the educational process. The strict time and memory limits set in olympiad problems require participants to apply the most efficient algorithmic method suited to each stream of input data.

Thus, the apparatus for asymptotic evaluation of algorithm complexity serves as a universal mathematical mechanism for processing information efficiently, securely, and quickly across the diverse areas of information technology.

The theoretical foundations of computational complexity theory and the asymptotic analysis of algorithms took shape in the second half of the twentieth century and have been consistently developed by distinguished scholars over the intervening period.

The very first cornerstone of this field was laid by Alan Turing through his abstract model of computation, the concept of the Turing machine, with which he mathematically established the boundaries of what is solvable and unsolvable in principle. Later, in 1965, Juris Hartmanis and Richard Stearns published their fundamental paper “On the Computational Complexity of Algorithms,” formally founding computational complexity theory and introducing the concept of time and space complexity classes into scientific usage. Later, for these contributions, they were awarded the Turing Award in 1993.

Donald Knuth, regarded as the father of modern algorithmic analysis, made an incomparable contribution to universalizing the practical and mathematical apparatus of the field. In his famous multivolume work “The Art of Computer Programming,” he introduced the Big O, Big Omega, and Big Theta notations into computer science as a single academic standard for the mathematical evaluation of algorithm complexity. As a logical continuation of the field’s progress, Stephen Cook and Leonid Levin classified algorithms into classes by level of complexity and discovered the theory of NP-complete problems, thereby defining the theoretical limits of modern algorithmic analysis.

Researchers such as Srinivasan Arunachalam, Arkopal Dutt, Sabee Grewal, and Aparna Gupte have successfully investigated, in their recent work, the application of Marton’s

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar

polynomial Freiman–Ruzsa conjecture to practical programming. Previously, this theory was merely existential in nature and in practice would require exponential time complexity ($O(2^K)$). This group of scientists solved this difficult problem by developing a separate randomized algorithmic model that finds a basis of subspaces covering the input data in polynomial time complexity of the form poly(n,K). The asymptotically optimal algorithm they proposed made it possible to sharply reduce execution time in fields that demand high computational power, such as quantum machine learning, quadratic Goldreich–Levin problems, and agnostic tomography of stabilizer states.

Likewise, scholars such as David Martínez-Rubio and Cristóbal Guzmán have achieved significant scientific results in convex optimization and in solving complex problems under limited resources. On the l_1-unit ball, for l_2-Lipschitz functions, classical minimization methods require a very slow time complexity of $O(\frac{1}{\sqrt{T}})$. To solve this complex nondual optimization problem, they proposed a new suite of algorithms, based on a synthesis of chasing nested convex sets and bundle methods, with asymptotically optimal time complexity of the form $O^{\sim}(GR/T)$. As a result of their research, efficient algorithmic solutions were developed that sharply reduce oracle complexity in high-dimensional computing environments and can be implemented fully in polynomial time.

Methods

One of the most important criteria in evaluating the efficiency of algorithmic methods is computational complexity. The total execution time of a program can be represented as T=k*t, where k is the number of elementary operations performed and t is the unit of time needed to perform each operation. However, in modern computational theory the parameter t is disregarded. The reason is that the execution time of a single operation depends directly on the hardware used (processor power, memory speed, and other technical characteristics) and differs across computing systems. Therefore, in algorithmic analysis the main attention is focused on the input parameter (n) of the problem and the number of basic operations performed as a function of it. This dependence is expressed by the function k(n).

If, for values of n greater than some n_0, the growth rate of the number of operations k(n) does not exceed the product C* f(n) (where C > 0 is a constant independent of n), then the algorithm is said to have complexity O(f(n)), which is formalized as k(n) = O(f(n)).

By the formal definition: if, for every $n>n_0$, the inequality $k(n)<C*f(n)$ is satisfied for some constants C and n_0, then the operation-count function is regarded as having O(f(n)) asymptotics.

The main asymptotic classes that characterize the execution speed of algorithms include the following:

O(1) — Constant complexity: The volume of operations performed never exceeds a bounded constant value, regardless of the size n of the input data (for example, conditions for separating odd and even numbers).

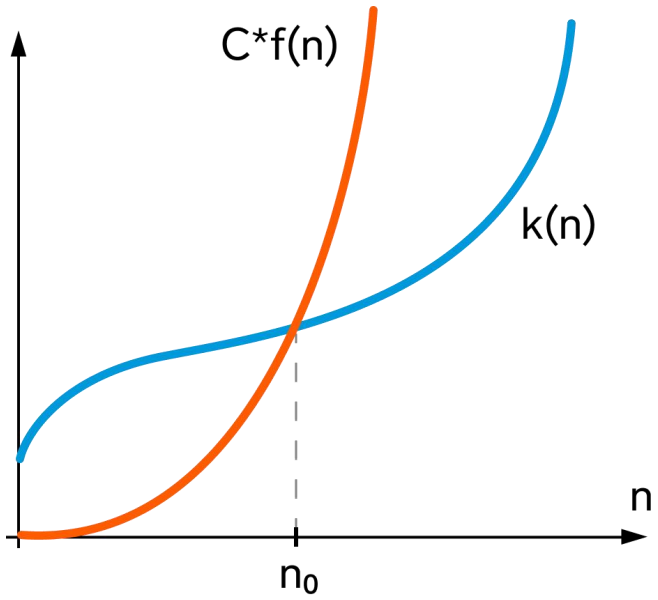
O(n) — Linear complexity: The number of operations performed grows in direct proportion to the problem size.

$O(n^2)$ — Quadratic complexity: The volume of operations grows in proportion to the square of the problem size.

$O(2^n)$ — Exponential complexity: A class of algorithms in which, with a slight increase in problem size, the computational volume multiplies in geometric-progression fashion.

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar



The complexity of most practical algorithms takes the form of a polynomial of degree k:

$a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0, \ (a_k > 0)$

In such cases, because the highest-degree term plays the decisive role, the overall asymptotics of the algorithm is taken to be equal to $O(n^k)$.

When calculating computational complexity, it is important to define the notion of an "elementary operation" correctly. Usually, the operation that is repeated most often in the algorithm, or the one most demanding in terms of time, is chosen as the basic unit. For example, if multiplication and addition occur in an algorithm in equal proportion, but multiplication takes more time at the hardware level, the computational complexity is evaluated only by the number of multiplications. Conversely, if the number of additions is far greater than the number of multiplications, the computational volume is measured by the number of additions.

In algorithms with a complex structure, for example in loops that repeat n times and contain several different operations inside (logical comparisons, arithmetic operations), all actions within the loop are generalized into a single characteristic operation, and the algorithm's complexity is evaluated by the execution frequency of this loop.

It is worth understanding from a practical standpoint why constants are discarded and only the asymptotics is studied. Suppose we lack the technical capacity to perform 2n operations, but can afford to compute n operations. This problem can be solved by increasing computing power 2-fold or by introducing parallel computing systems. However, if an algorithm has quadratic $O(n^2)$ complexity, even strengthening the computing device several-fold will not make it possible to solve the problem for larger values of n. For algorithms of the exponential $O(2^n)$ class, even when computing power is increased 100-fold, the maximum size (n) of a problem that can be solved within a given time can increase by only a few units.

Results

Problem 1. The number of operations k(n) of an algorithm is given in the form of the following arithmetic sum:

$k(n)=n+2n+3n+4n+\ldots+n^2$

Taking the common factor n out of the parentheses in this sum, we get:

$k(n)=n(1+2+3+\ldots+n)$

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar

The expression 1 + 2 + 3 + … + n in parentheses is the sum of an arithmetic progression of the first n natural numbers, and its value is calculated by the well-known formula:

$$1+2+3+\ldots+n=\frac{n(n+1)}{2}$$

We substitute this expression into the general function k(n):

$$k(n)=n\ \frac{n(n+1)}{2}$$

As a result, the function $k(n)=\frac{1}{2}n^3+\frac{1}{2}n^2$ is a polynomial of degree 3 in the variable n.

As noted in the theoretical part, when estimating the complexity of polynomial-type algorithms, the highest-degree term plays the decisive role, and constant coefficients are ignored in O-notation. Since, as $n\to\infty$, n^3 is the term whose growth rate is sharply higher than that of the n^2 term, the overall computational complexity of the algorithm has cubic asymptotics:

$k(n) = O(n^3)$

Problem 2. Analyzing the asymptotic complexity of an algorithm for finding the longest substring of ones in a binary string

Problem statement: A string of length n consisting only of the digits 0 and 1 is given. It is required to determine the length of the longest consecutive substring of the string consisting only of 1s. For example, for the string 101101001001111011, the longest sequence of ones is 4.

The presented C++ program code for solving this problem looks as follows:

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int main()
{
   string S;
   cin >> S;
   int n = S.size();
   int ans = 0;
   for (int i = 0; i < n; ++i)
   {
      int t = 0;
      while (i + t < n && S[i + t] == '1')
         ++t;
      ans = max(ans, t);
   }
   cout << ans << endl;
   return 0;
}
```

The presented program code has $O(n^2)$ time complexity. The main reason is the use, inside the outer for loop, of an inner while loop. In the worst case (for example, when the string consists entirely of 1s), at each step of the for loop the while loop re-examines all characters up to the end of the string. As a result, the number of operations performed

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar

$$n+(n-1)+(n-2)+\ldots+1=\frac{n(n+1)}{2}$$

forms an arithmetic progression, which asymptotically gives $O(n^2)$ complexity.

Discussion

The theoretical asymptotic analysis methods considered in this study, and their application to practical problems and algorithmic models, clearly demonstrate the role of an in-depth study of algorithm complexity in software engineering and the computational sciences. The results obtained confirm that estimating the time and space complexity of algorithms in advance, and optimizing them, is a matter not only of academic significance but also of practical and economic necessity. To evaluate comprehensively the possibilities of analyzing algorithmic complexity and putting it into practice, its positive and negative aspects should be examined in a coherent logical sequence.

One of the main advantages of studying algorithm complexity and applying it in practice is the possibility of significantly optimizing computational resources. Determining the asymptotics of an algorithm allows the programmer to predict how efficient it is even before running the code on hardware. For example, in the problem of finding the longest sequence of ones in a binary string considered in our study, choosing, instead of the nested-loop solution of complexity $O(n^2)$, a linear algorithm with $O(n)$ time complexity reduces the computation time for large values of n (for example, $n=10^6$) from several hours to milliseconds, which in turn sharply saves server capacity and electrical energy. At the same time, in the present era of exponentially growing information volume, since algorithms of complexity $O(2^n)$ or $O(n^2)$ lead to system failure under large data streams, choosing algorithms of the class $O(\frac{n}{\log n})$ or $O(n)$ through asymptotic analysis ensures the scalability of systems. Moreover, by disregarding the time parameter (t) when counting operations and focusing on the number of characteristic operations k(n), algorithms can be evaluated objectively, independent of the performance of a particular computer. Recent studies, in particular the reduction of the PFR conjecture from exponential time $O(2^K)$ to polynomial time poly(n,K) by Arunachalam and colleagues, and the reduction of the speed of nondual optimization to $O^{\sim}(GR/T)$ by Martínez-Rubio and Guzmán, have multiplied the possibilities of putting quantum computing and artificial intelligence models into practice.

Despite its wide-ranging advantages, asymptotic analysis of algorithmic complexity is not free of certain limitations and practical difficulties. One of the biggest shortcomings of asymptotic analysis is that constants (C) and lower-order terms are ignored in the formula $O(f(n))$. For example, if we consider two algorithms with $k_1(n) = 1000n$ and $k_2(n) = n^2$, then although theoretically the first is $O(n)$ and the second is $O(n^2)$, in practical problems with $n<1000$ the $O(n^2)$ algorithm actually runs faster. Moreover, most asymptotic analyses rely on the worst case; for example, although the worst-case complexity of the quicksort algorithm is $O(n^2)$, in practice its average running time is $O(\frac{n}{\log n})$ and it runs faster than many algorithms with a guaranteed $O(\frac{n}{\log n})$ bound. On the other hand, since algorithms with low asymptotic complexity require extremely complex mathematical and software structures, implementing them in code and debugging them requires a great deal of resources. Finally, time-optimized $O(n)$ or $O(1)$ algorithms often require additional memory resources, which can cause problems on devices with limited memory.

Conclusion

This scientific article has thoroughly examined the computational complexity of algorithms, methods of asymptotic analysis, and their role in practical programming and optimization problems. In the course of the research, the relationship between time and the number of

ISSN: 2224-6878 | Pakistan Volume 6, No. 10.

IF: 14.91 | 2026 http://waerasia.org/index.php/ijtar

operations was studied, and the advantages of the O-notation (Big O) theory, which is independent of hardware constraints, were substantiated using practical examples. It was shown that reducing quadratic $O(n^2)$ algorithms to linear O(n) form, and correctly evaluating constants and the trade-off between memory and time, can reduce computation time many-fold in processing large volumes of data. In conclusion, analyzing algorithmic complexity is not merely a theoretical calculation but a decisive factor ensuring the performance and scalability of software systems.

References.

1. Arunachalam S., Dutt A., Grewal S., Gupte A. Marton's conjecture in polynomial time // arXiv preprint arXiv:2609.20771. – 2026.
2. Martínez-Rubio D., Guzmán C. Stable Movement for Nondual Lipschitz Convex Optimization: Efficiency and Nearly Optimal Oracle Rates // arXiv preprint arXiv:2609.20701. – 2026.
3. Miloschewsky D., Podder S. Hardness of Pathfinding in a Welded Tree // arXiv preprint arXiv:2609.20651. – 2026.
4. Ahmadi A. A., Chaudhry A., Narang I., Tang Y. Complexity Of Output Feedback Stabilization // arXiv preprint arXiv:2609.20636. – 2026.
5. Chen Z., Lv S., Min C., Xu Y., Sun J., Quan Q. Time-Efficient Iterative Learning Planning for Safety-Critical Dynamic Obstacle Avoidance // arXiv preprint arXiv:2609.20435. – 2026.