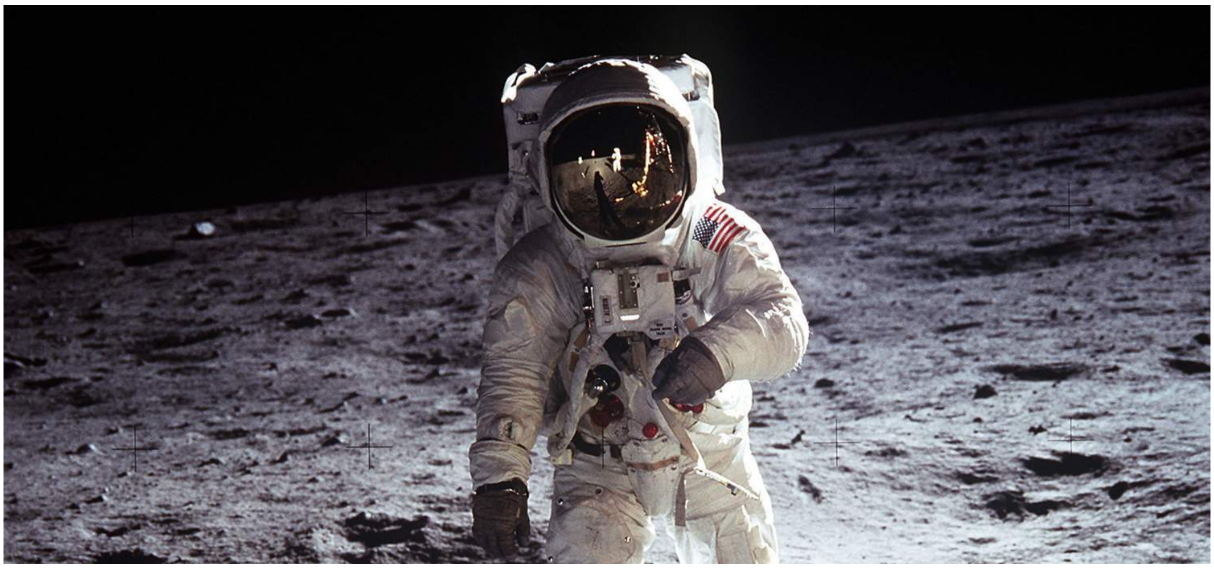


Complexity Of An Algorithm

Algorithm

Well-defined sequential computational technique that accepts a value or a collection of values as input and produces the output(s) needed to solve a problem.

- The algorithm should be clear and unambiguous.
- There should be 0 or more well-defined inputs in an algorithm.
- An algorithm must produce one or more well-defined outputs that are equivalent to the desired output. After a specific number of steps, algorithms must ground to a halt.
- Algorithms must stop or end after a finite number of steps.
- In an algorithm, step-by-step instructions should be supplied, and they should be independent of any computer code.

Complexity

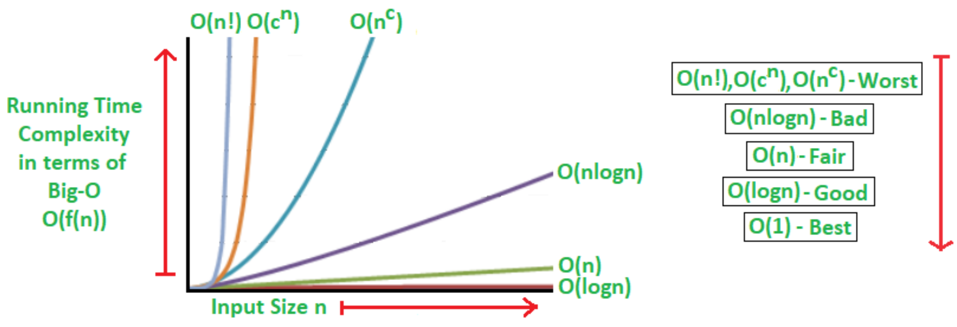
Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of length of the input. While, the space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Complexity Analysis

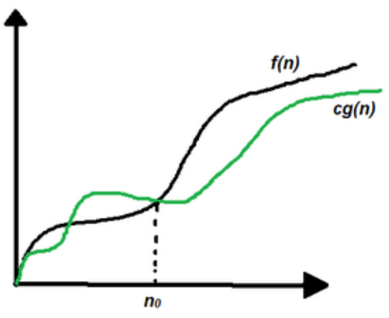
Complexity analysis is defined as a technique to characterise the time taken by an algorithm with respect to input size (independent from the machine, language and compiler). It is used for evaluating the variations of execution time on different algorithms.

Asymptotic Notations in Complexity Analysis

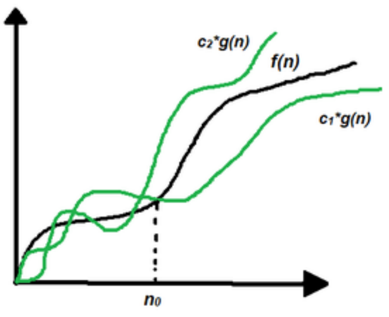
1. **Big-O Notation *O* :** It represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm. By using big O- notation, we can asymptotically limit the expansion of a running time to a range of constant factors above and below. It is a model for quantifying algorithm performance.



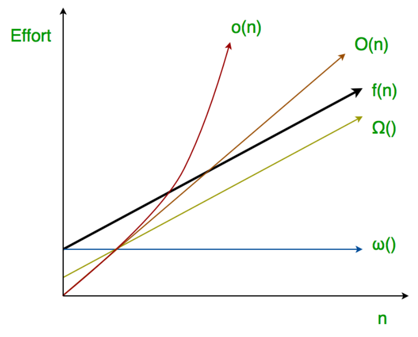
2. **Omega Notation *Ω* :** It represents the lower bound of the running time of an algorithm. Thus, it provides the best-case complexity of an algorithm. The execution time serves as a lower bound on the algorithm's time complexity. It defined as the condition that allows an algorithm to complete statement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



3. **Theta Notation *Θ* :** It 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. The execution time serves as both a lower and upper bound on the algorithm's time complexity. It exists as both, the most, and least boundaries for a given input value.



4. **Little O Asymptotic Notation *o* :** It is used as a tight upper bound on the growth of an algorithm's effort (this effort is described by the function f(n)),even though, as written, it can also be a loose upper bound. "Little-o" (o()) notation is used to describe an upper bound that cannot be tight.

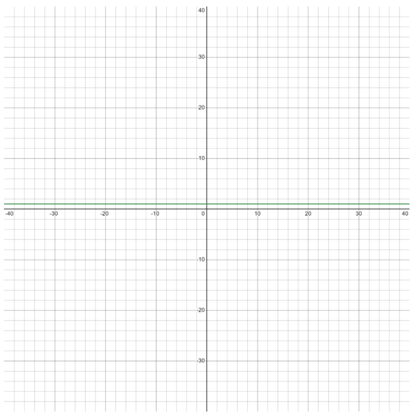


How To Measure Complexity?

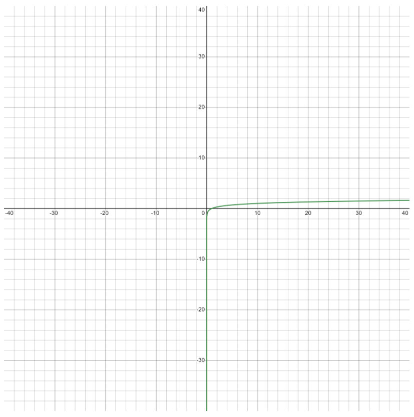
1. **Time Complexity :** The time complexity of an algorithm is defined as the amount of time taken by an algorithm to run as a function of the length of the input. Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.
    - To estimate the time complexity, we need to consider the cost of each fundamental instruction and the number of times the instruction is executed.
    - If we have statements with basic operations like comparisons, return statements, assignments, and reading a variable. We can assume they take constant time each O(1).
    - For any loop, we find out the runtime of the block inside them and multiply it by the number of times the program will repeat the loop.
        - T(N)= n *(t(cout statement))
          = n * O(1)
          =O(n)
    - For 2D arrays, we would have nested loop concepts, which means a loop inside a loop.
        - T(N)= n * m *(t(cout statement))
          = n * m * O(1)
          =O(n*m)
2. **Space Complexity :** The amount of memory required by the algorithm to solve a given problem is called the space complexity of the algorithm. Problem-solving using a computer requires memory to hold temporary data or final result while the program is in execution.
    - The space Complexity of an algorithm is the total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.
    - Space complexity is a parallel concept to time complexity. If we need to create an array of size n, this will require O(n) space. If we create a two-dimensional array of size n*n, this will require O(n^2) space.
3. **Auxiliary Space :** The temporary space needed for the use of an algorithm is referred to as auxiliary space. Like temporary arrays, pointers, etc. It is preferable to make use of Auxiliary Space when comparing things like sorting algorithms. for example, **sorting algorithms** take **O(n)** space, as there is an input array to sort but **auxiliary space is O(1)** in that case.

📌 `total time = time(statement1) + time(statement2) + ... time (statementN)`

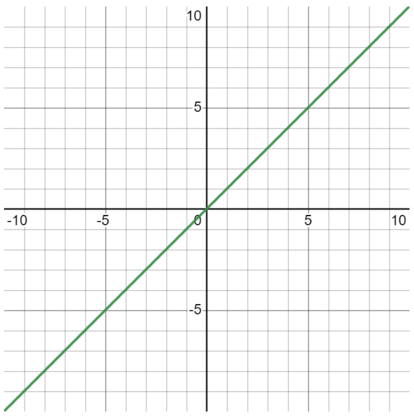
Constant Complexity O(1) : If the function or method of the program takes negligible execution time. Then that will be considered as constant complexity. comparisons, assignments, and simple instructions



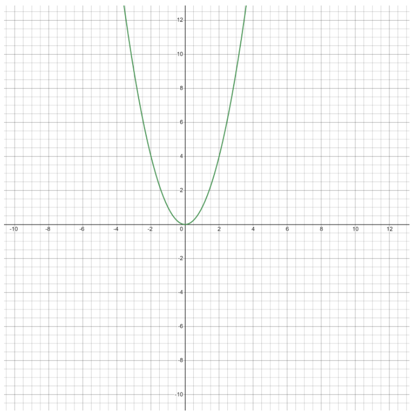
Logarithmic Complexity O(log(n)) : It undergoes the execution of the order of **log(N)** steps. To perform operations on N elements, it often takes the logarithmic base as 2.



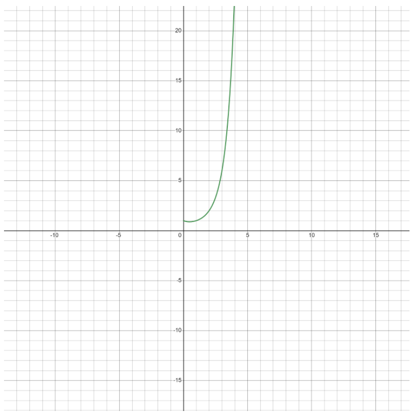
Linear Complexity O(n) : It encompasses the same number of steps as that of the total number of elements to implement an operation on N elements. Loops and repeated instructions



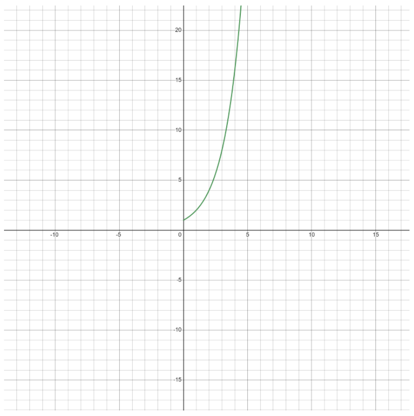
Quadratic Complexity O(n*m) : For N input data size, it undergoes the order of N2 count of operations on N number of elements for solving a given problem. Nested loop



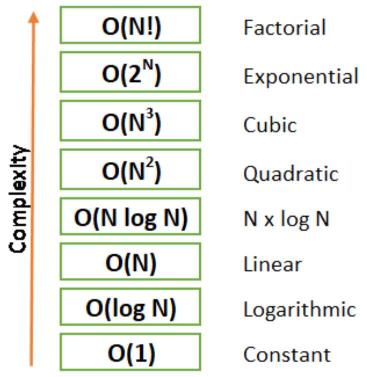
Factorial Complexity O(n!) : For N input data size, it executes the order of N! steps on N elements to solve a given problem.

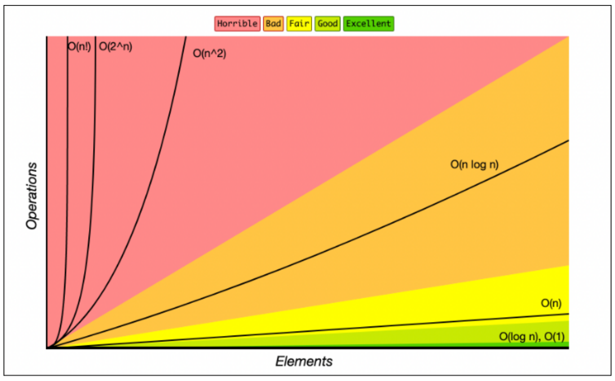


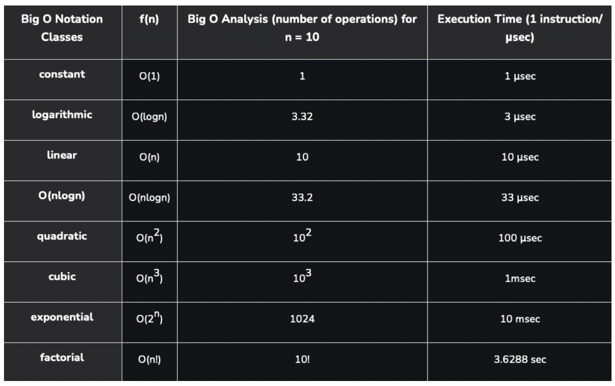
Exponential Complexity O(2^N), O(nk) : For N elements, it will execute the order of the count of operations that is exponentially dependable on the input data size.



Comparison of Complexity







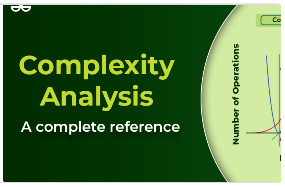
Big O Notation Classes	f(n)	Big O Analysis (number of operations) for n = 10	Execution Time (1 instruction/ μsec)
constant	O(1)	1	1 μsec
logarithmic	O(logn)	3.32	3 μsec
linear	O(n)	10	10 μsec
O(nlogn)	O(nlogn)	33.2	33 μsec
quadratic	$O(n^2)$	10^2	100 μsec
cubic	$O(n^3)$	10^3	1msec
exponential	$O(2^n)$	1024	10 msec
factorial	O(n!)	10!	3.6288 sec

📚 References

Complete Guide On Complexity Analysis - Data Structure and Algorithms Tutorial - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

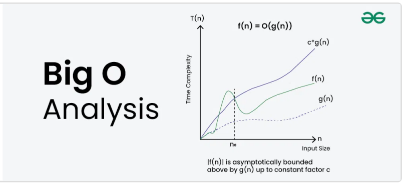
https://www.geeksforgeeks.org/complete-guide-on-complexity-analysis/



Big O Notation Tutorial - A Guide to Big O Analysis - GeeksforGeeks

Big O notation is a mathematical tool used in computer science to express the upper bound of an algorithm's time or space complexity, allowing for the comparison of algorithm efficiency in worst-case scenarios.

https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/



Big O Analysis
f(n) = o(g(n))
c*g(n)
f(n)
g(n)
Time Complexity
Input Size
|f(n)| is asymptotically bounded above by g(n) up to constant factor c

Big O Cheat Sheet – Time Complexity Chart

An algorithm is a set of well-defined instructions for solving a specific problem. You can solve these problems in various ways. This means that the method you use to arrive at the same solution may differ from mine, but we should both get the same r...

https://www.freecodecamp.org/news/big-o-cheat-sheet-time-complexity-chart/

✍🏼 Author → Serhat Kumas

https://www.linkedin.com/in/serhatkumas/

SerhatKumas - Overview

Computer engineering student who loves coding in different fields instead of focusing on a one spesific area. - SerhatKumas

https://github.com/SerhatKumas