Introduction to Asymptotic Analysis and Big O

Asymptotic analysis is a way to classify the running time complexity of algorithms.

We'll cover the following

We have seen that the time complexity of an algorithm can be expressed as a polynomial. To compare two algorithms, we can compare the respective polynomials. However, the analysis performed in the previous lessons is a bit cumbersome and would become intractable for bigger algorithms that we tend to encounter in practice.

Asymptotic Analysis

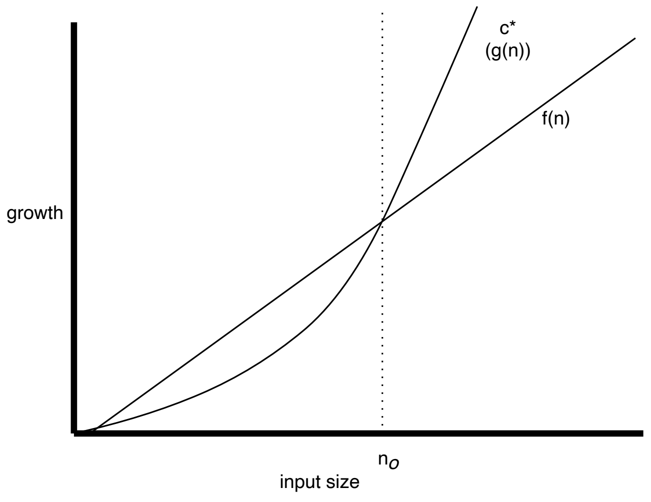
One observation that helps us is that we want to worry about large input sizes only. If the input size is really small, how bad can a poorly designed algorithm get, right? Mathematicians have a tool for this sort of analysis called the asymptotic notation. The asymptotic notation compares two functions, say, $f(n)$ and $g(n)$ for very large values of n. This fits in nicely with our need for comparing algorithms for very large input sizes.

Big O Notation

One of the asymptotic notations is the Big O notation. A function $f(n)$ is considered $O(g(n))$, read as **big oh** of $g(n)$, if there exists some positive real constant c and an integer $n_0 \geq 0$, such that the following inequality holds for all $n \geq n_0$:

$$f(n) \leq cg(n)$$

The following graph shows a plot of a function $f(n)$ and $cg(n)$ that demonstrates this inequality.



Note that the above inequality does not have to hold for all n. For $n < n_0$, $f(n)$ is allowed to exceed $cg(n)$, but for all values of n beyond some value n_0, $f(n)$ is not allowed to exceed $cg(n)$.

What good is this? It tells us that for very large values of n, $f(n)$ will be at most within a constant factor of $g(n)$. In other words, $f(n)$ will grow no faster than a constant multiple of $g(n)$. Put yet another way, the rate of growth of $f(n)$ is within constant factors of that of $g(n)$.

> People tend to write $f(n)$ = $O(g(n))$, which isn't technically accurate. A whole lot of functions can satisfy the $O(g(n))$ conditions. Thus, $O(g(n))$ is a set of functions. It is OK to say that $f(n)$ belongs to $O(g(n))$.

Example

Let's consider an algorithm whose running time is given by $f(n) = 3n^3 + 4n + 2$. Let us try to verify that this algorithm's time complexity is in $O(n^3)$. To do this, we need to find a positive constant c and an integer $n_0 \geq 0$, such that for all $n \geq n_0$:

$$3n^3 + 4n + 2 \leq cn^3$$

The above inequality would still be true if we re-wrote it while replacing cn^3 with $3n^3 + 4n^3 + 2n^3$. What we have done is replacement of the variable part in all terms with n^3, the variable-part of the highest order term. This gives us:

$$3n^3 + 4n + 2 \leq 3n^3 + 4n^3 + 2n^3 = 9n^3$$

This does not violate the inequality because each term on the right hand side is greater than the corresponding term on the left hand side. Now, comparing it with the definition of Big-O, we can pick c = 9.

That leaves n_0. For what values of n is the inequality $9n^3 \leq cn^3$ satisfied? All of them, actually! So, we can pick $n_0 = 1$.

The above solution $(c = 9, n_0 = 1)$ is not unique. We could have picked any value for c that exceeds the coefficient of the highest power of n in $f(n)$. Suppose, we decided to pick $c = 4$. The reader can verify that the inequality $3n^3 + 4n + 2 \leq cn^3$ still holds for $n_0 = 3$ or higher.

Note that it is not possible to find a constant c and n_0 to show that $f(n) = 3n^3 + 4n + 2$ is $O(n^2)$ or $O(n)$. It is possible to show that $f(n)$ is $O(n^4)$ or $O(n^5)$ or any higher power of n. Mathematically, it is correct to say that $3n^3 + 4n + 2$ is $O(n^4)$, but from a computer science point of view it is not very useful. It gives us a loose bound on the asymptotic running time of the algorithm. When dealing with time and space complexities, we are generally interested in the tightest possible bound when it comes to the asymptotic notation.

> Suppose algorithm A and B have running time of $O(n)$ and $O(n^2)$, respectively. For sufficiently large input sizes, algorithm A will run faster than algorithm B. That **does not** mean that algorithm A will **always** run faster than algorithm B.

> Algorithm A and B both have running time $O(n)$. The execution time for these algorithms, for very large input sizes, will be within constant factors of each other. For all practical purposes, they are considered equally good.

Simplified Asymptotic Analysis

Once we have obtained the time complexity of an algorithm by counting the number of primitive operations as discussed in the previous two lessons, we can arrive at the Big O notation for the algorithm simply by:

- Dropping the multiplicative constants with all terms
- Dropping all but the highest order term

Thus, $n^2 + 2n + 1$ is $O(n^2)$ while $n^5 + 4n^3 + 2n + 43$ is $O(n^5)$.

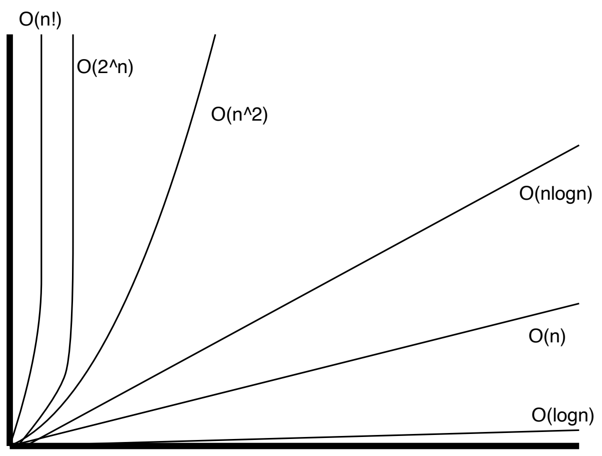
A Comparison of Some Common Functions

It is easy to work with simple polynomials in n, but when the time complexity involves other types of functions like $log()$, you may find it hard to identify the “highest order term”. The following table lists some commonly encountered functions in ascending order of rate of growth. Any function can be given as Big O of any other function that appears later in this table.

Function	Name
Any constant	Constant

Function	Name
$logn$	Logarithmic
log^2n	Log-square
$\sqrt{n}$	Root-n
n	Linear
$nlogn$	Linearithmic
n^2	Quadratic
n^3	Cubic
n^4	Quartic
2^n	Exponential
e^n	Exponential
$n!$	n-Factorial

The following graph visually shows some of the functions from the above table.



Quick quiz on Big O!

1 e^{3n} is in $O(e^n)$

A) True

Correct Answer

[✓ B) False]

Explanation

$e^{3n} \leq ce^n$ must be satisfied and $e^{3n} \leq ce^n \Rightarrow e^n e^{2n} \leq ce^n$ so such a constant c cannot be found.

2 10^n is in $O(2^n)$

A) True

Correct Answer

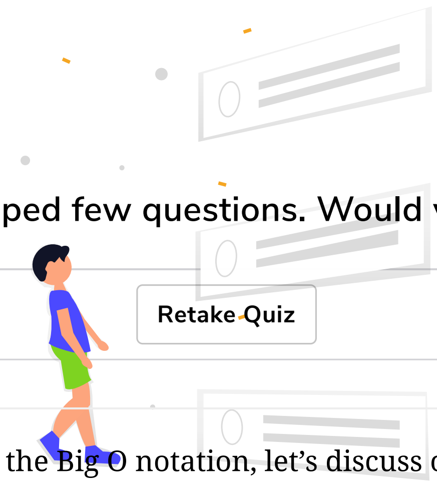
[✓ B) False]

Explanation

$10^n \leq c2^n$ must be satisfied and no such constant c can be found.

SUMMARY

Correct	0
Incorrect	0
Unanswered	2



Seems like you skipped few questions. Would you like to try again?

Retake Quiz

Now that you are familiar with the Big O notation, let’s discuss other notations in the next lesson.