Big-O Notation

Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann–Landau notation or asymptotic notation.

Big O notation describes the complexity of your code using algebraic terms.

To understand what Big O notation is, we can take a look at a typical example, O(n²), which is usually pronounced “Big O squared”. The letter “n” here represents the input size, and the function “g(n) = n²” inside the “O()” gives us an idea of how complex the algorithm is with respect to the input size.

A typical algorithm that has the complexity of O(n²) would be the selection sort algorithm. Selection sort is a sorting algorithm that iterates through the list to ensure every element at index i is the ith smallest/largest element of the list

When we calculate big O notation, we only care about the dominant terms, and we do not care about the coefficients. Thus we take the n² as our final big O. We write it as O(n²), which again is pronounced “Big O squared”.

Big O Notation Cases

O(1) has the least complexity ,often called “constant time”, if you can create an algorithm to solve the problem in O(1), you are probably at your best. In some scenarios, the complexity may go beyond O(1), then we can analyze them by finding its O(1/g(n)) counterpart. For example, O(1/n) is more complex than O(1/n²).

O(log(n)) is more complex than O(1), but less complex than polynomials.As complexity is often related to divide and conquer algorithms, O(log(n)) is generally a good complexity you can reach for sorting algorithms. O(log(n)) is less complex than O(√n), because the square root function can be considered a polynomial, where the exponent is 0.5.

Complexity of polynomials increases as the exponent increases .For example, O(n⁵) is more complex than O(n⁴). Due to the simplicity of it, we actually went over quite many examples of polynomials in the previous sections.

Exponentials have greater complexity than polynomials as long as the coefficients are positive multiples of n. O(2ⁿ) is more complex than O(n⁹⁹), but O(2ⁿ) is actually less complex than O(1). We generally take 2 as base for exponentials and logarithms because things tends to be binary in Computer Science, but exponents can be changed by changing the coefficients. If not specified, the base for logarithms is assumed to be 2.

Factorials have greater complexity than exponentials. If you are interested in the reasoning, look up the Gamma function, it is an analytic continuation of a factorial. A short proof is that both factorials and exponentials have the same number of multiplications, but the numbers that get multiplied grow for factorials, while remaining constant for exponentials.

Time & Space Complexity

The space complexity is related to how much memory the program will use, and therefore is also an important factor to analyze.

The space complexity works similarly to time complexity. For example, selection sort has a space complexity of O(1), because it only stores one minimum value and its index for comparison, the maximum space used does not increase with the input size.

Some algorithms, such as bucket sort, have a space complexity of O(n), but are able to chop down the time complexity to O(1). Bucket sort sorts the array by creating a sorted list of all the possible elements in the array, then increments the count whenever the element is encountered. In the end the sorted array will be the sorted list elements repeated by their counts.

References :

<https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/>