Omega and Theta Classification

Robin Dawes

February 16, 2021

Shows the importance of the Ω and Θ classification for establishing deeper understanding of the performance of algorithms.

.

Introduction

Big O classification is all about putting an **upper bound** on the growth-rate of a function (typically a function that describes the running time of an algorithm). This is just the first step into the study of algorithmic complexity - there are several other ways of classify functions based on the time and/or space they use. Two of these are of particular interest to us in our exploration of data structures: Omega (Ω) Classification and Theta (Θ) Classification.

Combinations of Functions

If $f_1(n) \in O(g_1(n))$, and $f_2(n) \in O(g_2(n))$

then $f_1(n) + f_2(n) \ \in \ O(max(g_1(n), g_2(n)))$

and $f_1(n) * f_2(n) \ \in \ O(g_1(n) * g_2(n))$

Omega Classification

Big O classification gives us an upper bound on the growth-rate of a function but it doesn't tell us anything about a *lower bound* on the growth-rate.

Your first reaction to this observation might well be "Why would we care about a lower bound on the growth-rate? We use this computational complexity stuff to measure the worst-case running time of an algorithm ... and for worst-case analysis, all we need is an upper bound."

Before we explain why lower-bound analysis is important, we will define exactly what we mean by it and how it works.

DEFINITION: Let $f(n)$ and $g(n)$ be functions. If there exist constants n_0 and c with $c > 0$ such that

$$f(n) \geq c * g(n) \quad \forall\, n \geq n_0$$

then we write $f(n) \in \Omega(g(n))$

and we say $f(n)$ is in OMEGA $g(n)$.

Note that this is almost exactly the same as the definition of Big O except that the "$\leq$" has become "$\geq$"

Ω is the Greek letter "Omega"

As with Big O classification we can see that $\Omega(g(n))$ is actually a class of functions. $\Omega(g(n))$ contains all functions that grow **at least** as fast as $g(n)$ grows. We can also see that there is a hierarchy of Omega classes, just as there is a hierarchy of Big O classes. For example, suppose $f(n) \in \Omega(n^3)$. This means "growth-rate of $f(n)$" $\geq$ "growth-rate of n^3". But since "growth-rate of n^3" $\geq$ "growth rate of n^2", we can conclude that "growth rate of $f(n)$" $\geq$ "growth rate of n^2", which is equivalent to saying that $f(n) \in \Omega(n^2)$.

In fact if $f(n) \in \Omega(n^k)$ then $f(n) \in \Omega(n^i) \quad \forall\, i \in \{0, 1, \ldots, k\}$.

Recall the related result for Big O:

if $f(n) \in O(n^k)$
then $f(n) \in O(n^i) \quad \forall\, i \geq k$.

When determining the Big O classification for $f(n)$ we try to find the smallest function $g(n)$ such that $f(n) \in O(g(n))$. Conversely, when determining the classification for $f(n)$ we try to find the **largest** function $g(n)$ such that $f(n) \in \Omega(g(n))$.

EXAMPLE:

Let $f(n) = 0.0001 * n^2 + (10^6) * n + 3$

We know that $f(n) \in O(n^2)$.

It's also very easy to see that $f(n) \in \Omega(n^2)$ ... we can let $c = 0.0001$ and it is immediately clear that $f(n) \geq c * n^2 \quad \forall\, n \geq 0$.

Now is it possible that $f(n) \in \Omega(n^3)$?

If this were true, then there would exist a value n_0 and a positive constant c such that

$$f(n) \geq c * n^3 \quad \forall n \geq n_0$$

ie.

$$0.0001 * n^2 + (10^6) * n + 3 \geq c * n^3$$
$$3 \geq n * (c * n^2 - 0.0001 * n - 10^6)$$

but we can easily see that this is impossible: even if c is very small, as n gets large there will come a point beyond which $c * n^2 - 0.0001 * n - 10^6$ is ≥ 1 so $n * (c * n^2 - 0.0001 * n - 10^6) \geq n$, which would give $3 \geq n \ \ \forall n \geq n_0$ ... which is not possible.

Thus $f(n) \notin \Omega(n^3)$.

This example illustrates a useful fact: if $f(n)$ is a polynomial, then the Big O class and the Ω class for $f(n)$ are identical.

But this is not always the case. For example, consider this algorithm:

```
A(n):
if n % 2 == 0:
    for i = 1..n^2:
        print '*'
else:
    for i = 1..n:
        print '*'
```

Let $T_A(n)$ be the time required to execute $A(n)$. If you plot $T_A(n)$ for $n = 1, 2, 3, ...$ you will see that it has a zig-zag shape. The tops of the zigs occur when n is even, and they grow at the same rate as n^2. It is easy to see that $T_A(n) \in O(n^2)$. However, the bottoms of the zags, which occur when n is odd, do not show this behaviour - they grow at the same rate as n.

Referring back to our definitions we are now able to say that $T_A(n) \in O(n^2)$ and also $T_A(n) \in \Omega(n)$ ... and neither of these can be improved: there is no lower Big O class for $T_A(n)$ and no higher Ω class for $T_A(n)$.

This example demonstrates that an algorithm's Big O class may be different from its Ω class.

Theta Classification

IF WE CAN SHOW an algorithm's complexity is in $O(g(n))$ **and** in $\Omega(g(n))$ then we get very excited - it means that $g(n)$ gives both an upper and a lower bound on the growth-rate of the time required by the algorithm. Basically it means we know exactly how fast the algorithm's time requirement grows. This is so amazingly wonderful that we give it a special name:

DEFINITION: If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$, we write

$$f(n) \in \Theta(g(n))$$

and we say $f(n)$ is in THETA $g(n)$.

Θ is the upper-case Greek letter "Theta".

From what we have seen earlier you should have no trouble proving that if $f(n) = a_t * n^t + \cdots + a_1 * n + a_0$ is a polynomial with $a_t > 0$ then

$$f(n) \in \Theta(n^t)$$