Big O and Big Omega Notations

Discusses the Big O notation with examples

In the previous section, we defined Θ notation. In this section, we'll discuss big O and big Ω notations. The reader would notice the use of the adjective "big" with these notations and rightly suspect that there exist complementary small o and small ω notations, which we'll discuss in the next lesson.

Big O

Big O is expressed as O(g(n)) and is pronounced as "big oh of g of n". We observed that Θ provides both an asymptotically upper and lower tight bound. Big O only provides an asymptotic upper bound which may not necessarily be a tight bound.

Big O is the most commonly talked-about notation when discussing algorithms in industry settings or in interviews. You won't see Θ or other notations being discussed, and for good reason too. Generally, if we are guaranteed that an algorithm will perform no worse than a certain threshold, and that threshold is acceptable for the problem at hand, then knowing its best-case performance or finding a very tight upper bound may not be productive.

Formal Definition

Similar to Θ we define O(g(n)) as a set of functions and a function f(n) can be a member of this set if it satisfies the following conditions:

$$0 \le f(n) \le cg(n)$$

where c is a positive constant and the inequality holds after n crosses a positive threshold n_0

Explanation

- If $I(n) = \Theta(g(n))$ then it also implies that I(n) = O(g(n))
- The notation means that the function f(n) is bounded by above to within a constant factor of g(n).
- Note that the inequality requires f(n) to be greater than 0 after n crosses n_0 .
- The set O(g(n)) may also contain other functions also that satisfy the inequality. There could be several values of c that can be used to satisfy the inequality.

Insertion Sort

Previously, we came up with the following f(n) for the insertion sort algorithm

$$f(n) = [2*(n+1)+2n] + [2n+7[\frac{n(n-1)}{2}]]$$

Now we can attempt to find the O(g(n)) for this expression. As a general rule of thumb, when working with big O we can drop the lower order terms and concentrate only on the higher order terms. The term $\frac{n(n-1)}{2}$ is the highest order term, since it involves a square of n. The above expression is thus quadratic, and we can ignore all the constants and linear terms. Thus a tight bound on the expression would be $O(n^2)$. We can prove it below:

$$\frac{n(n-1)}{2} \le cn^2$$

$$let \ c = 10 \ and \ n_0 \ = 10$$

You can always pick a different set of constants as long as it satisfies the inequality for all $n > n_0$.

As a general rule of thumb, for an expression consisting of polynomials the expression/function is *O(the highest polynomial degree in the expression)*.

The complementary notation for big O is the *big omega* notation. The big omega notation provides an asymptotic lower bound and is expressed as $\Omega(g(n))$.

Formal Definition

As before, $\Omega(g(n))$ is a set of functions and any function f(n) that satisfies the below constraints belongs to this set and f(n) is said to be big omega of g(n).

$$0 \le g(n) \le f(n)$$

where c is a positive constant and the above inequality holds after r n crosses a positive threshold n_0

Explanation

- Notice that similar to big O, the definition mandates we only consider positive values for f(n).
- Big Ω is not necessarily a tight lower bound.
- If $f(n) \Theta(g(n))$ then it also implies $f(n) \Omega(g(n))$.

Relation to Θ

One can see that if a function $\mathbf{f(n)}$ is $\Theta(\mathbf{g(n)})$ then it follows that $\mathbf{f(n)}$ must be $O(\mathbf{g(n)})$ and $\Omega(\mathbf{g(n)})$. In the previous lesson we proved that $\mathbf{f(n)} = 2n^2-1$ was $\Theta((n^2))$. We can simply consider the right and left handsides of the following inequality in isolation to prove that $\mathbf{f(n)}$ is also $O(\mathbf{g(n)})$ and $\Omega(\mathbf{g(n)})$.

$$1(n^2+2)<2n^2-1<2(n^2+2)$$

