Growth of Functions

1 Asymptotic Notation

In this section we will consider how to describe the behaviour of functions on the **natural numbers**, $f(n), n \in \mathbb{N}$. These functions are typically used to describe the **running time** of an algorithm, where the input size is n. For example, $f(n)$ could be a function which describes how many steps are required to sort a list of numbers of size n.

When considering functions that represent the running time of algorithms, we are often interested in how the function *grows* as n increases without bound. This is called the *asymptotic behaviour* of the function, since we are letting $n \to \infty$. By describing functions in terms of the asymptotic behaviour, we are able to compare two algorithms to determine which is more efficient for large values of n. For example, if one algorithm uses $f(n) = 3n^2 + 14n$ operations, and another uses $g(n) = n^3$, then certainly the second algorithm seems faster if $n = 1, 2, 3$ etc.. but as soon as $n \geq 6$ the first algorithm is faster and will remain faster as $n \to \infty$.

Furthermore, when describing the running time of an algorithm, we also seek a definition that is *indifferent* to the particular hardware or software being used. Such changes may alter the overall time by a multiplicative factor, or by an additive constant. Thus the definition that we use to describe asymptotic running time should also be indifferent to such changes. For example, an algorithm that takes n^2 steps should somehow be in the same category as one that takes $2n^2$ steps.

In the definitions that follow, we will assume that the functions take in values from the **natural numbers**, (i.e. the *input size* of an algorithm), and that they return values in the **positive real numbers**, (i.e. the *running time*) So assume for now that $f : \mathbb{N} \to \mathbb{R}^+$. This assumption is valid when f represents the number of steps in an algorithm. It is important to note that similar definitions of Big-O etc, can *also* be defined for functions over $\mathbb{R}$ and may allow also for negative valued $f(x)$.

2 Big-O Notation

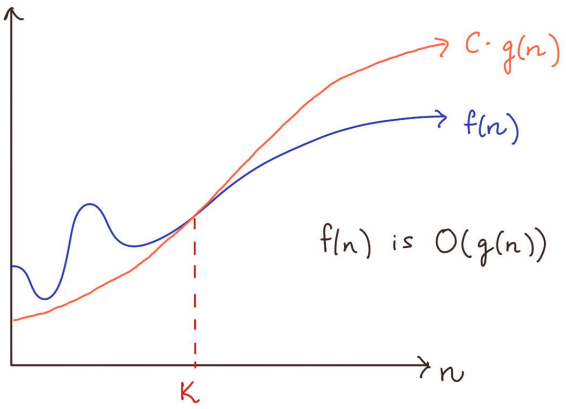
Our first definition will give us an *upper bound* for function $f(n)$ when n is very large, in other words, *asymptotically*. This is used extensively to describe the *worst case* scenario for the number of operations used by an algorithm. The notation used in the definition below is: $O(g(n))$ which is read "big-oh of g of n".

Definition. *Let $f(n) : \mathbb{N} \to \mathbb{R}^+$. For a given function $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are constants C and k such that*

$$f(n) \leq Cg(n)$$

for all $n > k$.

Let's start off by looking at a picture of what this means:



In the figure above we can see that *beyond* the value k, the function $f(n)$ is bounded above by $Cg(n)$, in other words $f(n)$ is bounded *above* by a constant factor of the function $g(n)$. Essentially, the function $g(n)$ describes the asymptotic behaviour of $f(n)$ because it provides an upper bound on the function $f(n)$ as n gets larger and larger.

Example 1. *Show that* $f(n) = n^2 + 3n + 1$ *is* $O(n^2)$.

Proof: Of the three terms in the above equation, the term n^2 is the term of the *highest order*, which means that as n gets very large, it will have the largest value. We will discuss the order of terms in more detail below, but more now you can verify this by simply plugging in a large n value, such as $n = 100$, and you will notice that the first term is the largest. In proving a big-Oh bound, we simply *identify* the highest order term, and then bound the entire function by some constant multiple of that term:

$$\begin{aligned} f(n) &= n^2 + 3n + 1 \\ &\leq n^2 + 3n^2 + n^2 \\ &= 5n^2 \end{aligned}$$

for all $n \geq 1$. It is important that in the process of upper bounding the function, we note for which n values the bound is valid. In this example, we used the fact that $3n \leq 3n^2$ and $1 \leq n^2$. Both of these statements are true for all $n \geq 1$. This means that our upper bound is valid for all $n \geq 1$, and thus we can conclude the that function $f(n) \leq Cn^2$ for $C = 5$ and $k = 1$.

Example 2. *Show that* $f(n) = 2\sqrt{n} + 5n$ *is* $O(n)$

Proof: Of the two terms above, the *linear* term, $5n$, is asymptotically larger than the term $\sqrt{n}$. Therefore as n gets large, it is reasonable to assume that we can show that the function $f(n)$ is less than a multiple of n. We use the same idea as in the previous example, and bound each tern by a multiple of n:

$$\begin{aligned} f(n) &= 2\sqrt{n} + 5n \\ &\leq 2n + 5n \\ &= 7n \end{aligned}$$

We used the fact that $\sqrt{n} \leq n$, which is true for all $n \geq 1$. Therefore, by setting $C = 7$, we have shown that $f(n)$ is $O(n)$.

2.1 Bounding Logarithmic functions

In order to continue with further examples on big-Oh, we need the following important fact:

$$\log_2 n \leq n \text{ for all } n \geq 1$$

Thus any $\log_2 n$ term that appears as part of a function $f(n)$ can be bounded above by the function n. In fact, it turns out that a more general version of this inequality is true for logarithms of any base $b > 0$:

For a base $b > 1$ and any exponent $a > 0$ we have that

$$\log_b n \leq n^a$$

for large enough n, in other words for $n \geq k$ for some constant k.

The above inequality provides us with several useful upper bounds for logarithmic functions. For example, we can say that $\log_2 n \leq \sqrt{n}$ for n large enough, also that $\log_2 n \leq n^2$ and $\log_2 n \leq n^{0.1}$ for large enough n values.

We will now show how these facts can be used to find the big-Oh value of functions.

2.2 Examples

Example 3. *Show that* $f(n) = 7n + \log(n)$ *is* $O(n^2)$

Proof: When asked to show that a function is specifically big-Oh of n^2, the goal is the upper bound $f(n)$ by a multiple of n^2. Certainly $7n \leq 7n^2$, and as discussed above, $\log(n) \leq n^2$ (when $n \geq 1$). Thus $f(n) \leq 7n^2 + n^2 = 8n^2$ for $n \geq 1$. So $f(n)$ is $O(n^2)$ with $C = 8$ and $k = 1$.

The example above shows that $f(n)$ is asymptotically bounded above by n^2. This does *not* mean that there is not another *tighter* upper bound. We now show that for the same function we could have chosen a "smaller" upper bound:

Example 4. *Show that* $f(n) = 7n + \log(n)$ *is* $O(n)$.

Proof: $f(n) \leq 7n + n$ for $n \geq 1$. Thus $f(n) \leq 8n$, and $f(n)$ is $O(n)$ with $C = 8$ and $k = 1$.

Note that in the above examples, we used the fact that $\log n \leq n$. This was done in order to compare the logarithmic function to other *larger* terms that appeared in the function. However if the dominating term in a function is itself logarithmic, then it can be used as the upper bound directly.

Example 5. *Show that* $f(n) = \log_2 n + 4$ *is* $O(\log n)$.

Proof: We could use the fact that $\log n \leq n$, and conclude that $f(n)$ is $O(n)$, as this is a true statement. However it is *also* true that $f(n) \leq \log_2 n + 4 \log_2 n = 5 \log_2 n$ for all $n \geq 2$, and so we can *also* conclude that $f(n)$ is $O(\log n)$, which is a much tighter bound. Notice that the *base* of the logarithm is omitted from the big-Oh notation. Changing the base of a logarithm only changes its value by a multiplicative constant. Thus the specific base of a logarithm is usually omitted from the big-Oh notation, since it does not change the definition of big-Oh.

Example 6. *Show that the function* $f(n) = 3n^4 + n^3 - 5n + 6 + \log_5 n$ *is* $O(n^4)$.

Solution: The goal here is to bound the function by a multiple of n^4. The positive terms can each be bounded above by n^4 and the negative terms can be simply omitted, since doing so only makes the

function larger. Thus for $n \geq 1$,

$$\begin{aligned} f(n) &= 3n^4 + n^3 - 5n + 6 + \log_5 n \\ &\leq 3n^4 + n^4 + 6n^4 + n^4 \\ &= 11n^4 \end{aligned}$$

Thus by setting $C = 11$ we have shown that $f(n)$ is $O(n^4)$

Example 7. *Show that $f(n) = n^4$ is not $O(n^3)$.*

Proof: To show that this function is *not* $O(n^3)$ we need to show that *no pair* of constants C and k exist such that $n^4 \leq Cn^3$ for $n > k$. If this were true, then $n^4 \leq Cn^3$ would imply that $n \leq C$ (by dividing both sides by n). But it is impossible that $n \leq C$ for all n values, where $n > k$, since n can be *any* large number and C is just some constant. So it is impossible to have $n^4 \leq Cn^3$ and thus $f(n)$ is not $O(n^3)$.

Example 8. *Show that $n!$ is $O(n^n)$*

Proof: The definition of the factorial sign is $n! = 1 \cdot 2 \cdot 3 \cdots n$. We can make each of these terms bigger and create the following upper bound:

$$n! \leq n \cdot n \cdot n \cdots n = n^n$$

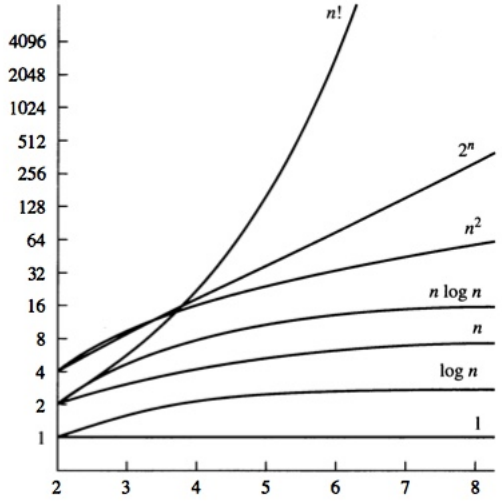
Thus $n! \leq n^n$ for all $n \geq 1$. So $n!$ is $O(n^n)$.

2.3 Comparative orders of growth

You may have noticed from the above examples that the big-Oh function can be determined by identifying the "*highest-order term*" in the equation. In Example 1, the highest-order term is n^2, and in Example 6, the highest-order term is n^4. Once the highest-order term is identified, it is usually possible to upper bound all the remaining terms by a multiple of this term, thus establishing a big-Oh upper bound for your function.

Suppose for example that $f(n) = \log n + n$. Recall that $\log n \leq n$ for large enough n, and that the "*highest-order term*" in this function is n. Therefore, we should aim to prove that $f(n)$ is $O(n)$. If $h(n) = 2^n + n^5$, what is the highest-order term is this function? Is $h(n) \leq C2^n$ or is $h(n) \leq Cn^5$? We can answer this question by understanding the relative order of growth of certain types of functions.

The figure below shows the relative order of growth of certain common function in computer science:



This figure shows that that for large n values, the function 2^n dominates polynomial functions like $n^2, n^3,$, etc. It also illustrates that logarithmic functions like $\log_b n$ are much smaller than polynomials $n, n^2, n^3 \ldots$ etc.

We will now establish a general order for asymptotic behaviour of functions. This order will help you identify the dominating terms in a function and thus identify the upper bound. The table below represents some of the most common complexity functions seen in computer science, listed in *increasing order of growth* as you go down the table:

Type	Big-Oh	Asymptotic Order
Logarithmic:	$O(\log n)$	$\log n$
Poly Logarithmic:	$O((\log n)^2), O((\log n)^3), \ldots$	$(\log n)^2 \leq (\log n)^3 \leq (\log n)^4 \leq \ldots$
Fractional Power:	$O(n^c)$ for $0 < c < 1$	$n^{0.1} \leq n^{0.2} \leq n^{0.3} \leq \ldots$
Linear:	$O(n)$	n
$n \log n$ time :	$O(n \log n)$	$n \log n \leq n(\log n)^2 \leq n(\log n)^3 \leq \ldots$
Polynomial time:	$O(n^a)$ for $a > 1$	$n^2 \leq n^3 \leq \ldots$
Exponential time:	$O(2^n)$	$1.5^n \leq 2^n \leq 3^n \leq \ldots$

Let's look at some examples, and practice identifying the dominating function:

- $f(n) = 2n^3 + n \log n$

  Applying again the fact that $\log n \leq n$, the second term is bounded by n^2. The term n^3 is of a higher order than n^2, and thus we should try to prove that $f(n)$ is $O(n^3)$.

  Proof: $f(n) \leq 2n^3 + n^2 \leq 2n^3 + n^3 = 3n^3$ for $n \geq 1$.

- $f(n) = 3n^4 + (\log n)^3$

  From the table above, the term $(\log n)^3$ is of a lower-order than the term n^4. Thus we should try to prove that $f(n)$ is $O(n^4)$.

  Proof: Using the fact that $\log n \leq n^a$ for any a, this allows us to say that $(\log n)^3 \leq n$ for $n \geq k$. Therefore, $f(n) \leq 3n^4 + n \leq 3n^4 + n^4 = 4n^4$ for $n \geq k$. Therefore $f(n)$ is $O(n^4)$.

- $f(n) = (n^3 + 2)/(n^2 + n + 1)$.

  In a case like this, we can identify the dominating terms from the numerator and denominator and estimate an upper bound from this simplified function. $f(n)$ is *approximately* n^3/n^2 for large enough n values. Therefore we should attempt to show that $f(n)$ is $O(n)$.

  Proof: Since $(n^3 + 2)/(n^2 + n + 1) \leq (n^3 + 2n^3)/n^2 = 3n$, then $f(n)$ is $O(n)$.

- $f(n) = (n \log n + \sqrt{n})$

  The dominating function out of the two is $n \log n$, as seen in the figure above. Thus we show that $f(n)$ is $O(n \log n)$.

  Proof:. Notice that $f(n) \leq n \log n + n \log n$ and so clearly $f(n)$ is $O(n \log n)$.

- $f(n) = \log n^2$

  Recall the properties of logarithms that $a \log x = \log x^a$. So $\log n^2 = 2 \log n$ and this means that $f(n)$ is $O(\log n)$.

- $f(n) = \sqrt{n} \log n + n$

  Since we know that $\log n \leq n^a$ for large enough n, then the first term is bounded by $\sqrt{n} \cdot \sqrt{n} = n$ for large enough n. Thus $f(n) \leq n + n = 2n$ and so it is $O(n)$.

3 Big-Ω and Big-Θ

If we truely want to get a notion of how a function behaves asymptotically then we need not only an upper bound, but also a lower bound. One reason for this is that our upper bound may be much greater than our actual function, and as n gets larger it may drastically overestimate the behavior of $f(n)$.

The notation big-Ω is defined similarly to that of big-O, but for a *lower* bound.

Definition. *Let $f(n)$ and $g(n)$ be functions on the natural numbers to the positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ if there is a constant C such that*

$$f(n) \geq Cg(n)$$

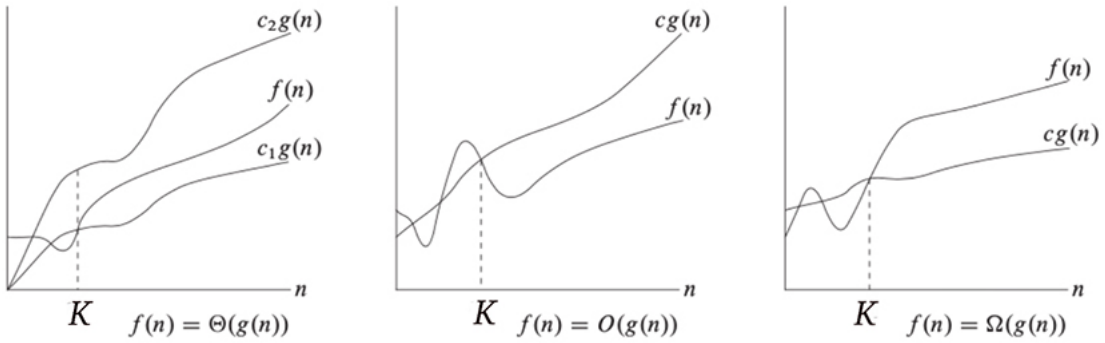
whenever $n > k$.

This is read "big omega". Note that in this definition we defined $f(n)$ to be a function over the *natural numbers* and recall as mentioned early that a similar definition works over the real numbers.

As a first example, let's take $f(n) = n^3 + n^2$. Techniques similar to those we used in big-O can be applied here in the same way, except now with a lower bound in mind. Note $f(n) \geq n^3$, and so $f(n)$ is $\Omega(n^3)$. The same function is also $O(n^3)$. So it is bounded above and below asymptotically in the same way. In this case, we use the notation Big-Θ to specify that the function $f(n)$ can be *sandwiched* between multiples of $g(n)$.

Definition. *Let $f(n)$, and $g(n)$ be functions from the natural numbers to the positive reals. If $f(n)$ is $O(g(n))$ **and** $f(n)$ is $\Omega(g(n))$ then we say that $f(n)$ is of the **order** of $g(n)$ and use the notaion $\Theta(g(n))$.*

Ideally we search for situations where we can determine a function $g(n)$ for which $f(n)$ is $\Theta(g(n))$ since this provides an asymptotically tight bound for our function. In the figure below, we can compare the three definitions that we have seen so far:



Example 9. *Show that $f(n) = n^3 + 1 + \log n$ is $\Theta(n^3)$.*

Proof: Let's start with the upper bound. Since $\log n \leq n^3$,

$$f(n) \leq n^3 + n^3 + n^3 = 3n^3$$

for $n \geq 1$. Thus $f(n)$ is $O(n^3)$. For the lower bound we create a inequality in the opposite direction:

$$f(n) = n^3 + 1 + \log n \geq n^3$$

which is true for all $n \geq 1$, and so $f(n)$ is $\Omega(n^3)$. The order of $f(n)$ is therefore n, $f(n)$ is $\Theta(n^3)$.

Example 10. *Describe the asymptotic behaviour of $f(n) = n^2 \log n + 2^n$.*

Solution: The function is dominated by the highest-order term 2^n. Thus we expect that the asymptotic behaviour to be $\Theta(2^n)$. Let's check this in detail, starting with the upper bound. For the purpose of analysis, we can assume that the logarithm is base 2. Note that for $n \geq 7$ we have that $2^n \geq n^2 \log n$ (you can verify this directly, although you can deduce that it is true from the table). Therefore,

$$n^2 \log n + 2^n \leq 2^n + 2^n = 2 \cdot 2^n$$

for $n \geq 7$. Therefore $f(n)$ is $O(2^n)$. For the lower bound, we seek to show that $f(n) \geq C \cdot 2^n$. The idea is similar, we simply require that inequality is in the other direction, and thus

$$n^2 \log n + 2^n \geq 2^n$$

and therefore $f(n)$ is $\Omega(2^n)$. Therefore our function is $\Theta(2^n)$.