

Informatics 2: Introduction to Algorithms and Data Structures

Notes on Lecture 4

More asymptotics: Big O , big Ω , and big Θ

John Longley

© University of Edinburgh 2026

In Lecture 3, we made a start on the theory of asymptotics, introducing two members of our so-called *Gang of Five*, namely little o and little ω , which respectively captured the idea of one function being fundamentally slower growing or faster growing than another. In this lecture, we'll meet the rest of the gang: O , Ω , Θ , respectively pronounced as big- O , big- Ω , big- Θ .

Motivation: An absolute scale for growth rates

Let's start by giving an idea of where we're heading and what we're trying to achieve. So far, we've been considering two runtime functions, T_I and T_M , and we've found a way to articulate the idea that T_M slowly *relative to* T_I . But we might also ask: Is it possible to place the growth rates of these functions on some sort of an *absolute* scale, so that even if we're considering just one of these functions in isolation, we have a way of saying 'how fast' it grows from an asymptotic point of view? To use a simple analogy, we've found a way of saying something comparable to 'my little finger is shorter than my middle finger'—but can we also say something more like 'my little finger is 6cm long and my middle finger is 8cm'?

What might such a scale of growth rates look like? As a first stab, we might consider some scale or hierarchy of simple and well-known functions such as this:

$$\begin{array}{lll} f_0(n) & = & 1 \\ f_3(n) & = & n \\ f_6(n) & = & n^3 \end{array} \quad \begin{array}{lll} f_1(n) & = & \lg n \\ f_4(n) & = & n \lg n \\ f_7(n) & = & 2^n \end{array} \quad \begin{array}{lll} f_2(n) & = & \sqrt{n} \\ f_5(n) & = & n^2 \\ f_8(n) & = & 2^{2^n} \dots \end{array}$$

Here, for example, f_0 is the *constant* function that always takes the value 1 for any n —this function doesn't 'grow' at all as n increases. Next, we have the rather slow-growing function $\lg n$.¹ Then $\sqrt{n}$, which grows quite a bit faster than $\lg n$, and n , which grows faster still. Indeed, it should be clear, at least

¹In this course, $\lg$ will mean logarithm to base 2, which is the kind of logarithm we'll most often employ. For example, $\lg 8 = 3$, because $8 = 2^3$, while $\lg 100 = 6.643\dots$ because $100 = 2^{6.643\dots}$. You can see that $\lg n$ grows only very slowly with n —for instance, $\lg 1000000$ is still < 20 —but that nevertheless, $\lg n$ will tend to ∞ as n does.

intuitively, that each of our functions here is little- o of the one after it: $f_i = o(f_{i+1})$ for each i .

Because these are all very standard and familiar functions, we can imagine something like this might serve as a kind of scale against which we can measure other less familiar functions like T_I and T_M . In other words, we can ask: Which of the above functions do T_I and T_M most closely resemble in terms of their essential growth rate? Not that there's anything very special about the particular choice of simple functions that I've given here—these are just a few of the functions that tend to pop up quite commonly when we're thinking about algorithms and their runtimes. We could use any other functions as a 'yardstick' if they turned out to be useful. We're simply using the functions above to illustrate the basic idea of measuring growth rates in terms of simple and well understood functions. And that's what O , Ω and Θ will enable us to do.

Our end goal, in fact, will be to define a relation ' f is $\Theta(g)$ ', which we can read as a way of saying that f has the same 'essential growth rate' as g from the asymptotic point of view. We'll then be able to use Θ to measure or classify complicated functions in terms of simple ones. For instance, we'll see in Lecture 5 that our runtime function T_I for insert sort—which is in principle a rather complicated function—has the same essential growth rate as the simple function n^2 , while T_M has the same growth rate as the slower growing function $n \lg n$. So we'll be able to write

$$T_I = \Theta(n^2), \quad T_M = \Theta(n \lg n).$$

as a way of placing T_I and T_M on an absolute scale.

How are we going to define Θ ? The approach we'll take is first to define two other asymptotic relations:

- The relation ' f is $O(g)$ ': a way of saying f grows *no faster than* g .
- The relation ' f is $\Omega(g)$ ': a way of saying f grows *no slower than* g .

Once these are in place, we shall simply define f to be $\Theta(g)$ if f is *both* $O(g)$ and $\Omega(g)$: if f grows fundamentally no faster and no slower than g , it makes sense to say that f and g grow 'at the same rate' from the asymptotic point of view.

So actually, there's not too much to understand here. Most of the effort comes in understanding O : once we've grasped that, Ω is fairly easy because it's just the *dual* of big O , in the way that ω is the dual of o . Then once we have both O and Ω , the definition of Θ is just that both of these relations hold.

Introducing big O

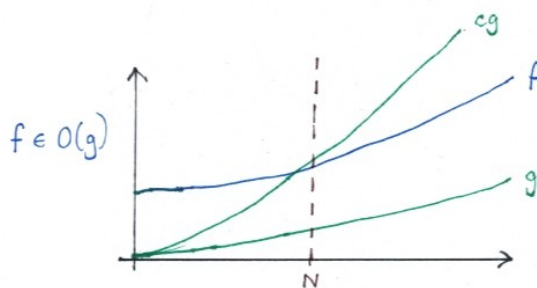
Let's recall two points about the general spirit of asymptotics that we're already familiar with from last time.

1. In asymptotics, we only care about what happens 'eventually' or 'in the limit'. We're allowed to ignore how f and g behave for 'small' values of n , which might not be typical of how they behave in the long run. For instance, many algorithms involve some initial 'setup cost' before the main work starts, which may make a significant contribution to the total runtime for small inputs, but whose impact dwindles into insignificance

as the inputs get larger. Asymptotics allows us to ignore any initial ‘settling-in period’.

2. The effect of *constant scaling factors* also gets ignored. We’re deliberately trying to make statements about f and g whose truth is unaffected by scaling f or g up or down.

We’re now looking for a way to say that a function f grows *asymptotically no faster* than g does. In accordance with the above ideas, it makes reasonable sense to say that this is true if there’s some *multiple* of g that eventually overtakes f . This is illustrated by the following graph:



Here we see that the g curve is, on the face of it, below the f curve. However, if we allow ourselves to scale g up by some constant factor C , then it will be above f —at least once we’ve passed a certain threshold N . So even though $g(n)$ might be less than $f(n)$ for every n , what we’re saying is that f isn’t growing fundamentally any faster than g in the asymptotic sense. We can pin this idea down as follows:

Definition: f is $O(g)$ means $\exists C > 0. \exists N. \forall n \geq N. f(n) \leq C \cdot g(n)$.

Or in words: we say f is $O(g)$, if there exist some scale-up factor C and some threshold value N , both sufficiently large that for all $n \geq N$ we have $f(n) \leq C \cdot g(n)$. When this is the case, we say that g is an *asymptotic upper bound* for f . As a typical use for this notation, we might say that the runtime of some algorithm is $O(n^2)$ if the way the runtime grows with n is ‘fundamentally no worse’ than n^2 —allowing for the possibility of constant factors which might mean the runtime is actually somewhere around $8n^2$ by some measure.

As a first example, let’s consider the functions

$$f(n) = 3n + \sqrt{n}, \quad g(n) = n.$$

Here it is indeed the case that f is $O(g)$ —we can also state this as ‘ $3n + \sqrt{n}$ is $O(n)$ ’.² How would we show this? Looking at the formal definition, we see that it will be enough to supply a specific example of a C and an N that make it true that $f(n) \leq C \cdot g(n)$ for all $n \geq N$. In this case, our task isn’t too hard: we know that $3n + \sqrt{n} \leq 3n + n = 4n$ for any $n \in \mathbb{N}$, so if we take our $C = 4$, we’ll have $f(n) \leq C \cdot g(n)$ for any n whatsoever. This means that it doesn’t

²See the comment in footnote 6 of Lecture Note 3 on the convention of using expressions involving n to denote mathematical functions.

really matter how we choose N : for instance, we could just take $N = 1$, then it will certainly be true that $f(n) \leq C \cdot g(n)$ for any $n \geq 1$. This is how we'd show formally that $3n + \sqrt{n} = O(n)$. Intuitively, we saying that fundamentally, $3n + \sqrt{n}$ grows no faster than n does, give or take a constant factor.

It's illuminating to compare the definition of big O with that of little o :

$$\begin{aligned} f \text{ is } o(g) & \text{ means } \forall c > 0. \exists N. \forall n \geq N. f(n) < c \cdot g(n), \\ f \text{ is } O(g) & \text{ means } \exists C > 0. \exists N. \forall n \geq N. f(n) \leq C \cdot g(n). \end{aligned}$$

The main difference is that with little o , we're saying that *any* multiple of g (however small) will eventually overtake f , while with big O , we're only saying that *some* multiple of g eventually overtakes f . In terms of the formula game we described in the last lecture, with little o , the demon gets to choose c and we have to be able to cope with whatever value it chooses, while with little O , we get to choose C , and it can be as large as we like. (Notice that C is an ' $\exists$ variable' in the definition of O .) So it's clear from this that if f is $o(g)$, then certainly f is $O(g)$. (Informally: if f grows slower than g , then certainly f grows no faster than g .) But the converse isn't true: for instance, for any function f at all, it will be the case that f is $O(f)$ (to show this, it's enough to take $C = 1$ and $N = 1$)—but it will *never* be true that f is $o(f)$.

Loosely speaking, then, we can think of o as a bit like a kind of ' $<$ ' relation between functions, and O as a kind of ' $\leq$ '. There are some limitations to this analogy, but on the whole it's a helpful way to think of it.

How is this notation used in practice? Just as with o , the official story is that $O(g)$ refers to a *set of functions*, namely the set of all f for which the above property ' f is $O(g)$ ' holds. Again, however, the common convention is to let ourselves write $O(g)$ to mean '*some* function (which we might not be bothering to specify) that is asymptotically bounded above by g '. This allows us to write things like $3n + \sqrt{n} = O(n)$, or $3n^2 + 5n + 17 = 3n^2 + O(n)$.

Examples of O

As with o , the way to get the feel of O is to look at some more examples (and non-examples). Regarding our first example above as 'Example 0', let's look at a few more. Again, we'll consider three levels of answer to each of these questions.

Example 1. $(5n + 4)(7n + 100) = O(n^2)$: **true or false?**

Even without doing the algebra, we can see here that the LHS is a product of two linear terms and hence a quadratic term, so its essential growth rate will be (no faster than) that of n^2 . So we can answer **YES** here.

For a slightly more detailed informal justification, we can see that the LHS expands to $35n^2 + 528n + 400$. Clearly, the dominant term here is $35n^2$, and the rest is just 'small change', i.e. stuff that will become negligible in relation to n^2 when n is large. (Indeed, it should be intuitively clear by now that $528n + 400 = o(n^2)$). So it's the n^2 here that basically determines the asymptotic growth rate of this function (remember that constant factors such as the 35 here can be ignored).

How would we give a rigorous justification from the formal definition? We want to come up with a C and an N such that $(5n + 4)(7n + 100) \leq Cn^2$ for

$n \geq N$. From the remarks above, it's clear that our C will need to be at least 35—in fact a bit more, to allow us to absorb the small change. In fact, there are many ways to arrive at a suitable C and N . Here's one way that requires relatively little work, based on looking at the two factors separately.

- We can note that $5n + 4 \leq 6n$ once $n \geq 4$.
- Likewise, $7n + 100 \leq 8n$ once $n \geq 100$.
- So we can say that once $n \geq 100$, we will have $(5n + 4)(7n + 100) \leq 6n \cdot 8n = 48n^2$.

This tells us that if we take $C = 48$ and $N = 100$, we will indeed have $(5n + 4)(7n + 100) \leq Cn^2$ for $n \geq N$.

It's worth noting that this isn't the only way to do it. There's quite a lot of freedom in how we pick our C and N here: for instance, we might pick a larger value of C , and that would let us get away with a smaller value for N . E.g. we could note that provided $n \geq 4$, we have $5n + 4 \leq 6n$ and $7n + 100 \leq 7n + 25n = 32n$, so taking $N = 4$ and $C = 6 \cdot 32 = 192$ would also work. Alternatively, we could take a smaller C and a larger N : e.g. we could note that $35n^2 + 528n + 400 < 36n^2$ once $n \geq 600$, so $C = 36$ and $N = 600$ would also work. My general advice here would be just to do whatever makes life easy for yourself: for the purpose of a big- O proof, it's fine to choose some C and N that are 'larger than necessary' if that makes the verification easier. You don't need to spend time working out some 'optimal' values for C and N based on exactly where the curves cross.

Example 2. $(5n + 4)(7n + 100) = O(n^3)$: true or false?

Here we have the same function f on the LHS, but now we're asking if n^3 (rather than n^2) is an asymptotic upper bound for our f .

The answer again is **YES**: if f is asymptotically bounded above by n^2 , it will certainly be bounded by n^3 which is even larger. Indeed, to show this formally, we can re-use the argument of the previous example. We've already shown that $f(n) \leq 48n^2$ once $n \geq 100$, so certainly $f(n) \leq 48n^3$ for all such n . So once again, the choice of $C = 48$ and $N = 100$ would work fine here, though many other choices are also possible.

What this example illustrates is that we can have an asymptotic upper bound for f that isn't a *tight* upper bound. In other words, if $f(n) = (5n+4)(7n+100)$, the statement $f = O(n^3)$ is certainly true, but it's not the strongest possible statement that we could make about f —not as strong as saying $f = O(n^2)$. As we'll see soon, this is a key difference between O and Θ , the latter expressing the idea of a tight bound: $f = \Theta(g)$ means f grows no slower and no faster than g . Thus, in the present case, $f = \Theta(n^2)$ would be correct but $f = \Theta(n^3)$ would not, whereas as both $f = O(n^2)$ and $f = O(n^3)$ are correct statements.

This is worth noting, because some writers, especially in older books and papers, tend to use big O rather loosely—sometimes meaning what we mean by O , sometimes meaning what we'd call Ω , and sometimes meaning what we'd call Θ . (There's a note about this at the end of Chapter 3 in the CLRS textbook.) But in this course, as in the CLRS book, we'll use these notations with the specific meanings that we're defining—because after all, they all express different

things we might reasonably want to say. But it's as well to be aware that not everyone in the world is quite as precise as we're encouraging you to be!

Example 3. $2^{2n} = O(2^n)$: true or false?

There may be a temptation at first to think that this one is true, because seemingly the only difference between 2^{2n} and 2^n is a 'constant factor' of 2, and we've emphasized that in asymptotics, constant factors don't count for anything. But where does this constant factor appear? It's not just a constant multiplier for the expression as a whole, but it's up in the exponent, and this makes a big difference.

Is it really true that 2^{2n} grows 'essentially no faster than 2^n '? If we look at the ratio $2^{2n}/2^n$, we see that this is 2^n , which itself grows rapidly without limit. (In fact, one can use this observation to show that $2^{2n} = \omega(2^n)$.)

How would we show rigorously from the formal definition that $2^{2n} = O(2^n)$ is false? Here we're wanting to show the negation of the relevant O statement:

$$\neg (\exists C > 0. \exists N. \forall n \geq N. 2^{2n} \leq C \cdot 2^n)$$

We can again apply the procedure of pushing the negation inwards and flipping the $\exists$ and $\forall$ symbols:

$$\forall C > 0. \forall N. \exists n \geq N. 2^{2n} > C \cdot 2^n .$$

So we're wanting to show that whatever values of C and N anyone proposes, they won't do for showing $2^{2n} = O(2^n)$. That is, we suppose we're given a C and N , and we have to pick an $n \geq N$ such that $2^{2n} > C \cdot 2^n$. What n should we pick? The property we're aiming for is equivalent to $2^n > C$, which in turn is equivalent to $n > \lg C$, and that tells us how large n needs to be. But we're also need to ensure that $n \geq N$, so taking any $n > \max(\lg C, N)$ will serve our purpose.

So the moral of this example is: a constant factor applied to an expression as a whole makes no difference asymptotically, but a constant factor somewhere else (e.g. up in an exponent) can make a very big difference.

Example 4. $\lg(n^7) = O(\lg n)$: true or false?

Again, this one may be deceptive at first. You may think that the answer is no, because n^7 certainly grows much faster than n does.

But what happens when we take logarithms? You will recall that $\lg(n^7) = 7 \lg n$ exactly (this is in fact true for logarithms to any base). And in the asymptotic sense, $7 \lg n$ and $\lg n$ grow at the same rate. We therefore *do* have $\lg(n^7) = O(\lg n)$, and indeed it suffices to take $C = 7$ and $N = 1$ in order to show this.

Big Ω : the dual of big O

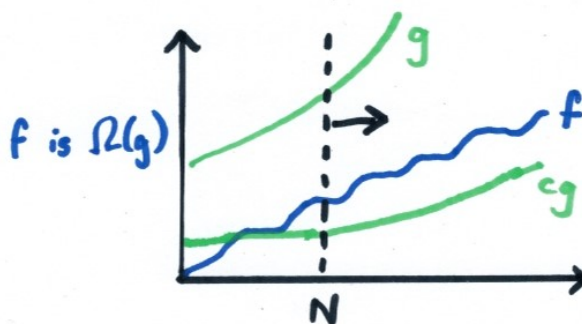
Once we've mastered big- O , big- Ω may not seem so hard, because it's simply the dual of big- O . Whereas we think of $f = O(g)$ as meaning ' f grows *no faster than* g ', we think of $f = \Omega(g)$ as ' f grows *no slower than* g ', which we can also express by saying that g is an *asymptotic lower bound* for f . A typical use we

might make of Ω is to say that even in the best case, the time taken to execute some algorithm on inputs of size n grows at least as fast as g .

How do we define the relation $f = \Omega(g)$? As you'd expect, the definition is dual to that of O : rather than saying that f is eventually bounded above by some sufficiently large multiple of g , we're now saying that f is eventually bounded below by some sufficiently small multiple of g .

Definition: f is $\Omega(g)$ means $\exists c > 0. \exists N. \forall n \geq N. f(n) \geq c \cdot g(n)$.

This is illustrated by the following picture:



Here, on the face of it, g seems to be larger than f . But what our $f = \Omega(g)$ is saying is that if we scale down g sufficiently by some small constant c , we can get something that's below f , at least once we've discounted some small values of n .

It probably won't surprise you now to learn that ' $f = \Omega(g)$ ' is precisely equivalent to ' $g = O(f)$ '. That's not hard to see from the definitions: if the numbers c and N suffice for showing $f = \Omega(g)$, then $C = 1/c$ and N will suffice for showing $g = O(f)$, and vice versa. However, it's definitely good to have both O and Ω , because the spirit of this notation is that we want to classify complicated functions in terms of simple ones. So if f was a complicated function like the runtime of some algorithm, and g was one of the standard simple functions mentioned at the start of the lecture, it would be more natural to say that $f = \Omega(g)$ rather than that $g = O(f)$.

Since everything we've said about O basically carries over to Ω , we'll content ourselves here with just one example of Ω .

Example 5. $n - \sqrt{n} = \Omega(n)$: true or false?

Even though $n - \sqrt{n} < n$ for any $n \geq 1$, it's clear that the growth rate of $n - \sqrt{n}$ is dominated by the term n , the $\sqrt{n}$ being 'small change' by comparison. So **YES**, it's true that $n - \sqrt{n}$ is asymptotically bounded below by n .

More rigorously: we want to show that there's a scale-down factor c such that cn is below $n - \sqrt{n}$ for large enough n . Obviously we'll need to take $c < 1$ here, and in fact *any* $c < 1$ will do. For instance, if we take $c = 0.5$, we'll have $n - \sqrt{n} \geq cn$ once n is large enough that $\sqrt{n} \geq 0.5n$, in other words $n \geq 4$. So taking $c = 0.5$ and $N = 4$, for all $n \geq N$ we have:

$$\sqrt{n} = 0.5 \cdot \sqrt{4} \cdot \sqrt{n} \leq 0.5 \cdot \sqrt{n} \cdot \sqrt{n} = cn.$$

And finally, big Θ

Now we're in a position to say what it means for f and g to have essentially the *same* growth rate.

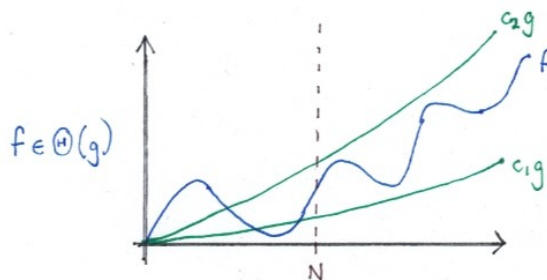
Definition: f is $\Theta(g)$ means f is both $O(g)$ and $\Omega(g)$.

So $f = \Theta(g)$ if g is both an asymptotic upper and lower bound for f . In this situation, we may say g is an asymptotic *tight* bound for f .

It's easy to see that this is equivalent to saying there exist both a scale-down factor and a scale-up factor, which I'll here call c_1 and c_2 respectively, such that $f(n)$ is sandwiched between $c_1 \cdot g(n)$ and $c_2 \cdot g(n)$ for sufficiently large n :

$$f = \Theta(g) \text{ says } \exists c_1 > 0, c_2 > 0, N. \forall n \geq N. c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n).$$

This is illustrated by the following picture:



Notice that we do allow f to wiggle around as much as it likes between the two bounds. Nonetheless, this does capture the idea that the rate at which f grows in the long run broadly matches that of g , discounting constant factors.

It's also not hard to see that $f = \Theta(g)$ is equivalent to $g = \Theta(f)$. This is immediate from our first definition above, if we've already bought into the idea that $f = O(g)$ means the same as $g = \Omega(f)$. But we can also see it like this. Suppose, in the above graph, we added a copy of the wiggly f curve scaled up by c_2/c_1 . For $n \geq N$, this would sit above the curve for c_2g , such as f sits above the curve for c_1g . So we now have a g -curve (specifically c_2g) sandwiched between two f -curves, and this shows that $g = \Theta(f)$.

Now for a few examples. This time, we'll be illustrating the typical use of Θ by giving a 'complicated' function f , and asking for some 'simple' function g such that $f = \Theta(g)$: this will be a way of placing the growth rate of f on some kind of absolute scale.

Example 6. What is the asymptotic growth rate of $f(n) = 3n^2 - 2n + 19$?

Here it's pretty clear that $3n^2$ is the dominant term, and the rest is just small change. So we might naturally hazard that $f(n) = \Theta(n^2)$. And indeed this is the case: because the $-2n + 19$ is small change, it's intuitively clear that $f(n)$ will be sandwiched between (say) $2n^2$ and $4n^2$ once n gets large enough. To make this formal, we need to say how large is 'large enough'? Here we can note that:

- once $n \geq 2$, we have $2n \leq n^2$, so $2n^2 \leq 3n^2 - 2n < f(n)$,
- once $n \geq 5$, we have $n^2 > 19$, so $4n^2 \geq 3n^2 + 19 > f(n)$.

So we can show $f(n) = \Theta(n^2)$ by taking $c_1 = 2$, $c_2 = 4$, and $N = 5$.

Thus, $\Theta(n^2)$ gives us the broad headlines of how fast f is growing—which may be all we need to know if we don't care about the finer details.

Example 7. What is the growth rate of $f(n) = 5 - 4/n$?

This may seem at first like a slightly odd example. (We'll quietly ignore the fact that $4/n$ is undefined when $n = 0$.) Clearly, the broad headlines are that when n is large, $f(n)$ is very close to 5. So we might say that $f(n)$ has the same essential 'growth rate' as the constant function $g(n) = 5$, which doesn't grow at all. But since 'constant factors don't matter', it's even simpler to say that

$$f(n) = \Theta(1),$$

where the '1' really means the constant function whose value is always 1. To show this rigorously, it's enough to note that if $n \geq 1$ then $n \leq 5 - 4/n \leq 5n$, so taking $c_1 = 1$, $c_2 = 5$, $N = 1$ will work.

This isn't just a frivolous example. Very often, we'll encounter quantities that are bounded both above and below by a constant, and calling them $\Theta(1)$ is a useful way to capture that. For example, any computational task for which there's an absolute upper (and lower) limit to how long it takes, regardless of the input size n , can be said to execute in time $\Theta(1)$, or *within constant time*. This doesn't imply that it always takes exactly the *same* time regardless of the input, only that there's a constant upper and lower bound to how long it takes.

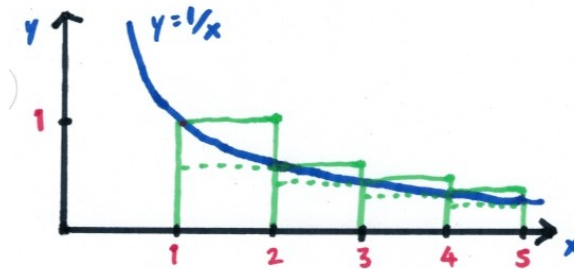
Example 8. What is the growth rate of $f(n) = \sum_{i=1}^n 1/i$?

We end with a more ambitious example. The point of this one is to illustrate that we can sometimes give Θ -estimates for complicated quantities whose exact value we don't have any good way of computing at all. This is perhaps where we see the real benefit of asymptotics—when we use it to reason about things that are simply too difficult to analyse or express exactly. Note, however, that this particular example is mathematically more demanding than anything we'd ask you in an exam or coursework.

The quantity $f(n)$ we have in mind is the sum $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$. (For example, $f(3) = 1\frac{5}{6}$.) In this case, we don't have a tidy formula that allows us to compute the exact value of $f(n)$ easily, other than by actually summing the relevant terms which may be impractical for large n . Nevertheless, the *growth rate* of this function can be expressed very neatly: it turns out that $f(n) = \Theta(\ln n)$, where $\ln n$ means the natural logarithm of n (to base e).³

Why is this true? We'll content ourselves here with an informal justification. From the picture below, we can see that the sum in question is something pretty close to the area under the curve $y = 1/x$ taken from 1 to n —that is, the integral $\int_1^n dx/x$, which is $[\ln x]_1^n = \ln n - 0 = \ln n$.

³Actually, as you'll see on the first tutorial sheet, $\Theta(\ln n)$ means exactly the same as $\Theta(\lg n)$, since $\ln$ and $\lg$ themselves have the same growth rate.



That's not the exact value, but you can see that the difference amounts to the areas of the small portions of the rectangles that lie above the $1/x$ curve, plus one extra rectangle at the right-hand end—a difference of less than 1 in total. And because in the limit, 1 is negligible compared with $\ln n$ (in asymptotic notation, $1 = o(\ln n)$), we can conclude that our $f(n)$ is $\Theta(\lg n)$. This isn't a fully rigorous argument, but with a bit of patience it can be turned into one.

Summing up

Let's now reconnect with a question we asked last time. We can imagine a whole bunch of implementations of MERGESORT (say), where each implementation J gives rise to its own runtime function T_J . Of course, these functions will all be slightly different from one another—but if we're after some essential property of MERGESORT itself, we'll want to ask: what is it that all these functions T_J can be expected to have in common?

We're now in a position to answer that question: we expect all these T_J to have the *same asymptotic growth rate*. That is, if J, J' are any two reasonable implementations of MERGESORT, we expect that $T_J = \Theta(T_{J'})$. What's more we can hope to identify what this growth rate is in terms of our scale of standard functions. In the next lecture, we'll do precisely this, and will argue that the runtime of (any implementation of) MERGESORT is in fact $\Theta(n \lg n)$. In this way, we're able to put our finger on an essential feature of the algorithm itself, abstracting away from the extraneous detail of some specific implementation.

This concludes our introduction to asymptotic analysis and the Gang of Five. To summarize:

1. $f = o(g)$ says f is *asymptotically smaller* than g (informally a kind of ' $<$ ' relation).
2. $f = O(g)$ says f is *asymptotically bounded above* by g (a kind of ' $\leq$ ').
3. $f = \Theta(g)$ is f and g have the same asymptotic growth rate (a kind of equivalence).
4. $f = \Omega(g)$ says f is *asymptotically bounded below* by g (a kind of ' $\geq$ ', dual to O).
5. $f = \omega(g)$ says f is *asymptotically larger* than g (a kind of ' $>$ ', dual to o).

The members of the Gang we'll meet most often in this course will be O and Θ . But to absorb the spirit of the theory, it's best to take the Gang as a whole, as we've done here. The theory of asymptotics is very general, and turns

out to be useful not just within Algorithms and Data Structures, but in many other areas of maths and in physics too. In this course, it will provide our basic framework that we use for expressing interesting properties of algorithms and quantifying how ‘good’ they are in various respects.

We’ve emphasized that there are three levels of mastery of these ideas that you might aspire to: yes/no answers, informal justification, and rigorous justification from the formal definitions. To keep things in perspective, it’s likely that you’ll be able to pass the course if you reach the second of these levels—but for a high mark, you will need to conquer the third as well. In any case, there are good reasons for putting in some effort to master the art of rigorous proofs:

- You’ll then end up with a crystal-clear understanding of what we do and don’t mean by e.g. $O(n \lg n)$ or $\Theta(n^2)$. Even if most of the time you resort to more informal reasoning, your intuition will then be very reliable.
- Getting your head round some definitions involving a mix of $\forall$ and $\exists$ variables is good preparation for various other complex ideas you might meet in later years, either in CS or Maths course. (For example, the *Pumping Lemma* for regular languages, covered in ITCS, also has this character.)

And finally: here, for convenience, is a list in increasing order of some of the growth rates that pop up quite frequently in connection with algorithms, along with the names they often go by.

- $\Theta(1)$: (within) constant time
- $\Theta(\lg n)$: logarithmic time
- $\Theta(n)$: linear time
- $\Theta(n \lg n)$: log-linear time
- $\Theta(n^2)$: quadratic time
- $\Theta(n^k)$ for some exponent k : polynomial time
- $\Theta(b^n)$ for some base $b > 1$: exponential time