We have seen several algorithms in the course, and we have loosely characterized their runtimes $t(n)$ in terms of the size n of the input. We say that the algorithm takes time $O(n)$ or $O(\log_2 n)$ or $O(n \log_2 n)$ or $O(n^2)$, etc, by considering how the runtime grows with n, ignoring constants. This level of understanding of $O(\ )$ is usually good enough for characterizing algorithms. However, we would like to be more formal about what this means, and in particular, what it means to ignore constants.

An analogy from Calculus: limits

A good analogy for informal versus formal definitions is one of the most fundamental ideas of Calculus: the limit. In your Calculus 1 course, you learned about various types of limits and you learned methods and rules for calculating limits e.g. squeeze rule, ratio test, l'Hopital's rule, etc.

You were also given the formal definition of a limit. This formal definition didn't play much role in your Calculus class. You were more concerned with using rules about limits than in fully understanding where these rules come from and why they work. If you go further ahead in your study of mathematics then you will find this formal definition comes up again[1]. And to your surprise, it also comes up in COMP 250.

Here is the formal definition of the limit of a sequence. *A sequence $t(n)$ of real numbers converges to (or has a limit of) a real number t_∞ if, for any $\epsilon > 0$, there exists an n_0 such that for all $n \geq n_0$, $\mid t(n) - t_\infty \mid < \epsilon$.*

This definition is subtle. There are two "for all " logical quantifiers and there is one "there exists" quantifier, and the three quantifiers have to be ordered in just the right way to express the idea, which is this: if you take any finite interval centered at the t_∞, namely $(t_\infty - \epsilon, t_\infty + \epsilon)$, then the values $t(n)$ of the sequence will all fall in that interval once n exceeds some finite value n_0. That finite value n_0 will depend on ϵ in many cases.

This is relevant to big O for two reasons. First, the formal definition of big O has a similar flavour to the formal definition of the limit of a sequence in Calculus. Second, we will see two lectures from now that there are rules that allow us to say that some function $t(n)$ is big O of some other function e.g. $t(n)$ is $O(\log_2 n)$, and these rules combine the formal definition of big O with the formal definition of a limit. Let's put limits aside for now, and return to them once we have big O under belts.

Big O

Let $t(n)$ be a well-defined sequence of integers. In the last several lectures, such a sequence represented the time or number of steps it takes an algorithm to run as a function of n which is the size of the input. However, today we put this interpretation aside and we just consider $t(n)$ to be a sequence of numbers, without any meaning. We will look at the behavior of this sequence $t(n)$ as n becomes large.

[1]in particular, starting in Real Analysis (e.g. MATH 242 at McGill)

Definition (preliminary)

Let $t(n)$ and $g(n)$ be two sequences of integers[2], where $n \geq 0$. We say that $t(n)$ *is asymptotically bounded above by* $g(n)$ if there exists a positive number n_0 such that,

$$\text{for all } \ n \geq n_0, \quad t(n) \ \leq g(n).$$

That is, $t(n)$ becomes less than or equal to $g(n)$ once n becomes sufficiently large.

Example

Consider the function $t(n) = 5n + 70$. It is never less than n, so for sure $t(n)$ is not asymptotically bounded above by n. It is also never less than $5n$, so it is not asymptotically bounded above by $5n$ either. But $t(n) = 5n + 70$ is less than $6n$ for sufficiently large n, namely $n \geq 12$, so $t(n)$ is asymptotically bounded above by $6n$. The constant 6 in $6n$ is one of infinitely many that works here. Any constant greater than 5 would do. For example, $t(n)$ is also asymptotically bounded above by $g(n) = 5.00001n$, although n needs to be quite large before $5n + 70 \leq 5.00001n$.

The formal definition of big O below is slightly different. It allows us to define an asymptotic upper bound on $t(n)$ in terms of a *simpler* function $g(n)$, e.g. :

$$1, \log n, \ n, \ n \log n, \ n^2, \ n^3, \ 2^n, \ldots$$

without having a constant coefficient. To do so, one puts the constant coefficient into the definition.

Definition (big O):

Let $t(n)$ and $g(n)$ be well-defined sequences of integers. We say $t(n)$ is $O(g(n))$ if there exist two positive numbers n_0 and c such that, for all $n \geq n_0$,

$$t(n) \ \leq \ c \ g(n).$$

We say "$t(n)$ is big O of $g(n)$". I emphasize: this definition allows us to keep the $g(n)$ simple by having a constant factor (c) that is separate from the simple $g(n)$.

A few notes about this definition: First, the definition still is valid if $g(n)$ is a complicated function, with lots of terms and constants. But the whole point of the definition is that we keep $g(n)$ simple. So that is what we will do for the rest of the course. Second, the condition the $n \geq n_0$ is also important. It allows us to ignore how $t(n)$ compares with $g(n)$ when n is small. This is why we are talking about an *asymptotic* upper bound.

[2] Usually we are thinking of positive integers, but the defition is general. The $t(n)$ could be real numbers, positive or negative

Example 1

The function $t(n) = 5n + 70$ is $O(n)$. Here are a few different proofs:

First,

$$\begin{aligned} t(n) &= 5n + 70 \\ &\leq 5n + 70n, \text{ when } n \geq 1 \\ &= 75n \end{aligned}$$

and so $n_0 = 1$ and $c = 75$ satisfies the definition.

Here is a second proof:

$$\begin{aligned} t(n) &= 5n + 70 \\ &\leq 5n + 6n, \text{ for } n \geq 12 \\ &= 11n \end{aligned}$$

and so $n_0 = 12$ and $c = 11$ also satisfies the definition.

Here is a third proof:

$$\begin{aligned} t(n) &= 5n + 70 \\ &\leq 5n + n, \text{ for } n \geq 70 \\ &= 6n \end{aligned}$$

and so $n_0 = 70$ and $c = 6$ also satisfies the definition.

A few points to note:

- If you can show $t(n)$ is $O(g(n))$ using constants c, n_0, then you can always increase c or n_0 or both, and these constants with satisfy the definition also. So, don't think of the c and n_0 as being uniquely defined.
- There is no "best" choice of c and n_0. The examples above show that if you make c bigger (less strict) then you can make n_0 smaller (more strict, since the inequality needs to hold for more values of n). The big O definition says nothing about "best" choice of c and n_0. It just says that there has to exist one such pair.
- There are inequalities in the definition, e.g. $n \geq n_0$ and $t(n) \leq cg(n)$. Does it matter if the inequalities are strict or not? No. If we were to change the definitions to be strict inequalities, then we just might have to increase the c or n slightly to make the definition work.

An example of an incorrect big O proof

Many of you are learning how to do proofs for the first time. It is important to distinguish a formal proof from a "back of the envelope" calculation. For a formal proof, you need to be clear on what you are trying to prove, what your assumptions are, and what are the logical steps that take you from your assumptions to your conclusions. Sometimes a proof requires a calculation, but there is more to the proof than calculating the "right answer".

For example, here is a typical example of an incorrect "proof" of the above: (in the lecture, I presented this a bit later)

$$\begin{array}{rcl} 5n + 70 & \leq & cn \\ 5n + 70n & \leq & cn, \quad n \geq 1 \\ 75n & \leq & cn \\ \text{Thus,} \quad c > 75, \quad n_0 = 1 & & \text{works.} \end{array}$$

This proof contains all the calculation elements of the first proof above. But the proof here is wrong, since it isn't clear which statement implies which. The first inequality may be true or false, possibly depending on n and c. The second inequality is different than the first. It also may be true or false, depending on c. And which implies which? The reader will assume (by default) that the second inequality *follows from* the first. But does it? Or does the second inequality imply the first? Who knows? Not me. Such proofs tend to get grades of 0.[3] This is not the big O that you want. Let's turn to another example.

ASIDE: another incorrect proof

When one is first learning to write proofs, it is common to leave out certain important information. Let's look at a few examples of how this happens.

Claim:

For all $n \geq 1$, $2n^2 \leq (n+1)^2$.

If you are like me, you probably can't just look at that claim and evaluate whether it is true or false. You need to carefully reason about it. Here is the sort of incorrect "proof" you might be tempted to write, given the sort of manipulations I've been doing in the course:

$$\begin{array}{rcl} 2n^2 & \leq & (n+1)^2 \\ & \leq & (n+n)^2, \text{ where } n \geq 1 \\ & \leq & 4n^2 \end{array}$$

which is true, i.e. $2n^2 \leq 4n^2$. Therefore, you might conclude that the claim you started with is true.

Unfortunately, the claim is false. Take $n = 3$ and note the inequality fails since $2 \cdot 3^2 > 4^2$. The proof is therefore wrong. What went wrong is that the first line of the proof *assumes what we are trying to prove.* This is a remarkably common mistake.

Let's now get back to big O and consider another example.

[3]As mentioned in class, with nearly 700 students, we just don't have the resources to ask you to write out these proofs on exams.

Example 2

Claim: The function $t(n) = 8n^2 - 17n + 46$ is $O(n^2)$.

Proof: We need to show there exists positive c and n_0 such that, for all $n \geq n_0$,

$$8n^2 - 17n + 46 \leq cn^2 \ .$$

$$\begin{aligned} t(n) &= 8n^2 - 17n + 46 \\ &\leq 8n^2 + 46n^2, \qquad \text{for } n \geq 1 \\ &= 54n^2 \end{aligned}$$

and so $n_0 = 1$ and $c = 54$ does the job.

Here is a second proof:

$$\begin{aligned} t(n) &= 8n^2 - 17n + 46 \\ &\leq 8n^2, \quad n \geq 3 \end{aligned}$$

and so $c = 8$ and $n_0 = 3$ does the job.

Miscellaneous notes

Here are a few points to be aware of:

- We sometimes say that a function $t(n)$ is $O(1)$. What does this mean? Applying the definition, it means that there exists constants c and n_0 such that, for all $n \geq n_0$, $t(n) \leq c$. That is, $t(n)$ is bounded above by a constant, namely $max\{t(0), t(1), \ldots, t(n_0), c\}$.
- You should never write that some function or algorithm is $O(3n)$ or $O(5log_2)$, etc. Instead, you should write $O(n)$ or $O(log_2)$, etc. Why? Because the point of big O notation is to avoid dealing with these constant factors. So, while it is still technically correct to write the above, in that it doesn't break the formal definition of big O, we just never do it.
- We generally want our $g(n)$ to be simple, and we also generally want it to be small. But the definition doesn't require this. For example, in the above example, $t(n) = 5n + 70$ is $O(n)$ but it is also $O(n \log n)$ and $O(n^2)$ and $O(n^3)$, etc. For this example, we say that $O(n)$ is a "tight bounds". We generally express $O(\ )$ using tight bounds. I will return to this point next lecture.

Big O and sets of functions

When we charaterize a function $t(n)$ as being $O(g(n))$ we usually use simple functions for $g(n)$ such as in the list of inequalities:

$$1 < \log_2 n < n < n \log_2 n < n^2 < n^3 < ... < 2^n < n! < n^n$$

Note that these inequalities don't hold for all n. Each of the inequalities holds only for sufficiently large n: For example, $2^n < n!$ holds only for $n \geq 4$, and $n^3 < 2^n$ only holds for $n \geq 10$. But for each of the inequalities, there is some n_0 such that the the inequality holds for all $n \geq n_0$. This is good enough, since big O deals with asymptoic behavior anyhow.

Generally when we talk about $O()$ of some function $t(n)$, we use the "tightest" (smallest) upper bound we can. For example, if we observe that a function $f(n)$ is $O(\log_2 n)$, then generally we would not say that that $f(n)$ is $O(n)$, even though technically $f(n)$ *would* be $O(n)$, and it would also be $O(n^2)$, etc.

A related observation is that, for a given simple $g(n)$ such as listed in the sequence of inequalities above, there are infinitely many functions $f(n)$ that are $O(g(n))$. So let's think about the set of functions that are $O(g(n))$. Up to now, we have been saying that some function $t(n)$ *is* $O(g(n))$. But sometimes we say that $t(n)$ *is a member of the set of functions that are* $O(g(n))$, or more simply $t(n)$ "belongs to" $O(g(n))$. In set notation, one writes "$t(n) \in O(g(n))$" where $\in$ is notation for set membership. With this notation in mind, and thinking of various $O(g(n))$ as sets of functions, the discussion in the paragraphs above implies that we have strict containment relations on sets of $t(n)$ functions:

$$O(1) \subset O(\log n) \subset O(n) \subset O(n \log n) \subset O(n^2) \ \cdots \subset O(2^n) \subset O(n!) \ldots$$

For example, any function $f(n)$ that is $O(1)$ must also be $O(\log_2 n)$, and any function $f(n)$ that is $O(\log_2 n)$ must also be $O(n)$,etc. It is common to use this set notation in the course and say "$t(n) \in O(g(n))$" instead of "$t(n)$ is $O(g(n))$".

Some Big O Rules

Last lecture, we showed some simple examples of how to show that some function $t(n)$ is $O(g(n))$ for some other function $g(n)$. In these examples, we manipulated an inequality and found a c and n_0. We don't want to have to do this every time we make a statement about big O, and so we would like to have some rules that allows us to avoid having to state a c and n_0. Thankfully and not surprisingly, there are such rules.

I emphasize: the point of the above rules that I am about to explain is that they allow us to make immediate statements about the $O(\ )$ behavior of some rather complicated functions. For example, we can just look at the following function

$$t(n) = 5 + 8 \log_2 n + 16n + \frac{n(n-1)}{25}$$

and observe it is $O(n^2)$ by noting that the largest term is n^2. The following rules *justify* this informal observation.

The "constant factors" rule says that if we multiply a function by a constant, then we don't change the $O()$ class of that function. The "sum rule" says that if we have a function that is a sum of two terms, then the $O()$ class of the function is the larger of the $O()$ classes of the two terms. The "product rule" says that if we have a function that is the product of two terms, then the $O()$ class of the function is given by the product of the (simple) functions that represent the $O()$ classes of the two terms. Here are the statements of the rules, and the proofs.

Constant Factors Rule

If $f(n)$ is $O(g(n))$ and $a > 0$ is a positive constant, then $af(n)$ is $O(g(n))$.

Proof: There exists a c such $f(n) \leq cg(n)$ for all $n \geq n_0$, and so $af(n) \leq acg(n)$ for all $n \geq n_0$. Thus we use ac and n_0 as constants to show $a\ f(n)$ is $O(g(n))$.

Sum Rule

If $f_1(n)$ is $O(g(n))$ and $f_2(n)$ is $O(h(n))$ and $g(n)$ is $O(h(n))$, then $f_1(n) + f_2(n)$ is $O(h(n))$.

Proof: From the three big O relationships given in the premise (i.e. the"if" part of the "if-then"), there exists constants $c_1, c_2, c_3, n_1, n_2, n_3$ such

$$f_1(n) \leq c_1 g(n) \text{ for all n} \geq \mathrm{n}_1$$

$$f_2(n) \leq c_2 h(n) \text{ for all n} \geq \mathrm{n}_2$$

$$g(n) \leq c_3 h(n) \text{ for all n} \geq \mathrm{n}_3$$

Thus,

$$f_1(n) + f_2(n) \leq c_1 g(n) + c_2 h(n) \text{ for all n } \geq \max(\mathrm{n}_1, \mathrm{n}_2)$$

Substituting for $g(n)$ using the third bound and making sure that n is large enough for all three bounds to hold:

$$f_1(n) + f_2(n) \leq c_1 c_3 h(n) + c_2 h(n) \text{ for all n } \geq \max(\mathrm{n}_1, \mathrm{n}_2, \mathrm{n}_3)$$

i.e.

$$f_1(n) + f_2(n) \leq ch(n) \text{ for all n } \geq \max(\mathrm{n}_1, \mathrm{n}_2, \mathrm{n}_3)$$

where $c = c_1 c_3 + c_2$, which proves the claim.

Product Rule

If $f_1(n)$ is $O(g_1(n))$ and $f_2(n)$ is $O(g_2(n)$, then $f_1(n) f_2(n)$ is $O(g_1(n)\ g_2(n))$.

Proof: Using similar constants as in the sum rule, we have: $f_1(n) * f_2(n) \leq c_1 g_1(n) * c_2 g_2(n)$ for all $n \geq max(n_1, n_2)$. So we can take $c_1 * c_2$ and $max(n_1, n_2)$ as our two constants.

Big Omega (asymptotic lower bound)

With big O, we defined an asymptotic *upper bound.* We said that one function grows at most as fast as another function. There is a similar definition for an asymptotic *lower* bound. Here we say that one function grows *at least* as fast as another function. The lower bound is called "big Omega".

Definition (big Omega): We say that $t(n)$ is $\Omega(g(n))$ if there exists positive constants n_0 and c such that, for all $n \geq n_0$,

$$t(n) \ \geq \ c\, g(n).$$

The idea is that $t(n)$ grows at least as fast as $g(n)$ times some constant, for sufficiently large n. Note that the only difference between the definition of $O()$ and $\Omega()$ is the $\leq$ vs. $\geq$ inequality.

Example

Claim: Let $t(n) = \frac{n(n-1)}{2}$. Then $t(n)$ is $\Omega(n^2)$.

To prove this claim, first note that $t(n)$ is less than $\frac{n^2}{2}$ for all n, so since we want a *lower* bound we need to choose a smaller c than $\frac{1}{2}$. Let's try something smaller, say $c = \frac{1}{4}$.

$$\begin{aligned} & \frac{n(n-1)}{2} \geq \frac{n^2}{4} \\ \iff & \ 2n(n-1) \geq n^2 \\ \iff & \ n^2 \geq 2n \\ \iff & \ n \geq 2 \end{aligned}$$

Note that the "if and only if" symbols $\iff$ are crucial here. For any n, the first inequality is either true or false. We don't know which, until we check. But putting the $\iff$ in there, we are say that the inequalities in the different lines have the same truth value.

The last lines says $n \geq 2$, this means that the first inequality is true if and only if $n \geq 2$. Thus, we can use $c = \frac{1}{4}$ and $n_0 = 2$.

Are these the only constants we can use? No. Let's try $c = \frac{1}{3}$.

$$\begin{aligned} & \frac{n(n-1)}{2} \geq \frac{n^2}{3} \\ \iff & \ \frac{3}{2}n(n-1) \geq n^2 \\ \iff & \ \frac{1}{2}n^2 \geq \frac{3}{2}n \\ \iff & \ n \geq 3 \end{aligned}$$

So, we can use $c = \frac{1}{3}$ and $n_0 = 3$.

Sets of $\Omega(\ )$ functions

We can say that if $f(n)$ *is* $\Omega(g(n))$ then $f(n)$ is a member of the set of functions that are $\Omega(g(n))$. The set relationship is different from what we saw with $O()$, namely the set membership symbols are in the opposite direction:

$$\Omega(1) \supset \Omega(\log n) \supset \Omega(n) \supset \Omega(n \log n) \supset \Omega(n^2) \ \cdots \supset \Omega(2^n) \supset \Omega(n!) \ldots$$

For example, any positive function that is increasing with n will automatically be $\Omega(1)$ since any of the values taken by that function will eventually be a lower bound (since the function is increasing). Thus, $\Omega(1)$ is a very weak condition and most functions are $\Omega(1)$. Indeed only functions that the only way a function would *not* be $\Omega(1)$ is if it had infinitely many n indices (a subsequence) that converged to 0. For example, $t(n) = \frac{1}{n}$ would be such a function.

Big Theta

For all functions $t(n)$ that we deal with in this course, there will be a simple function $g(n)$ such that $t(n)$ is both $O(g(n))$ and $\Omega(g(n))$. For example, $t(n) = \frac{n(n+1)}{2}$ is both $O(n^2)$ and $\Omega(n^2)$. In this case, we say that $t(n)$ is "big theta" of n, or $\Theta(g(n))$.

Definition (big theta): We say that $t(n)$ is $\Theta(g(n))$ if $t(n)$ is both $O(g(n))$ and $\Omega(g(n))$ for some $g(n)$. An equivalent definition is that there exists three positive constants n_0 and c_1 and c_2 such that, for all $n \geq n_0$,

$$c_1 \; g(n) \; \leq \; t(n) \; \leq \; c_2 \; g(n).$$

Obviously, $c_1 \leq c_2$. Note that here we have such one constant n_0. This is just the max of the n_1, n_2 constants of the $O(\ )$ and $\Omega(\ )$ definitions.

Sets of Big Theta functions

Since a function $t(n)$ is $\Theta(g(n))$ if it is both $O(g(n))$ and $\Omega(g(n))$, it means that the set of $\Theta(g(n))$ functions is the intersection of the set of $O(g(n))$ functions and the set of $\Omega(g(n))$ functions.

Thus, unlike different $O()$ and $\Omega()$ classes which form nested sets, as discussed earlier, the big Θ classes form disjoint sets. If $t(n)$ belongs to one big Θ class, then it doesn't belong to any other Θ class.

There are many possible functions $t(n)$ that do not belong to any $\Theta(g(n))$ class. For example, consider a function that has a constant value, say 5, when n is even and has value n when n is odd. Such a function $t(n)$ is $\Omega(1)$ and it is $O(n)$ but it is neither $\Theta(1)$ nor $\Theta(n)$. Obviously one can contrive many such functions. But these functions rarely come up in real computer science problems. For every $t(n)$ function that we have discussed in this course, it belongs in some $\Theta()$ class.

If the time complexity functions we care about in computer science are always characterized by some $\Theta(\ )$, then why do we bother talking about $O(\ )$ and $\Omega(\ )$? The answer is that often we are using these asympototic bounds because we want to express that something takes at most or at least a certain amount of time. When we want to say "at most", we are talking about an upper bound and so saying $O(\ )$ emphasizes this. When we say "at least", we are talking about a lower bound and so saying $\Omega(\ )$ that.

Best and worst case

The time it takes to run an algorithm depends on the size n of the input, but it also depends on the values of the input. For example, to remove an element from an arraylist takes constant time (fast) if one is removing the last element in the list, but it takes time proportion to the size of the list if one is removing the first element. Similarly, quicksort is very quick if one happens to choose good pivots in each call of the recursion but it is slow if one happens to choose poor pivots in each call of the recursion. So one cannot say in general that algorithm has a certain $t(n)$ time. There are different times, depending on the inputs. (Note for *ArrayList.remove(i)*, the variable `i` is a parameter, and here I am considering this to be part of the input to the algorithm. The other part of the input would be the list of size n.)

One often talks about *best case* and *worst case* for an algorithm. Here one is restricting the analysis of the algorithm to a particular set of inputs in which the algorithm takes a minimum number of steps or a maximum number of steps, respectively. With this restriction, one writes the best or worst case time as two functions $t_{best}(n)$ or $t_{worst}(n)$ respectively.

The $t_{best}(n)$ or $t_{worst}(n)$ functions usually *each* can be characterized by some $\Theta(\ g(n)\ )$. However, the $g(n)$ may be different for $t_{best}(n)$ and $t_{worst}(n)$. Here are some examples that you should be very familiar with.

List Algorithms	$t_{best}(n)$	$t_{worst}(n)$
add, remove element (array list)	$\Theta(1)$	$\Theta(n)$
add, remove an element (doubly linked list)	$\Theta(1)$	$\Theta(n)$
insertion sort	$\Theta(n)$	$\Theta(n^2)$
selection sort	$\Theta(n^2)$	$\Theta(n^2)$
binary search (sorted array)	$\Theta(1)$	$\Theta(\log n)$
mergesort	$\Theta(n \log n)$	$\Theta(n \log n)$
quick sort	$\Theta(n \log n)$	$\Theta(n^2)$

Note that selection sort and mergesort have the same best and worst case complexities, but for the others the best and worst cases differ.

For a given algorithm, one often characterizes $t_{best}(n)$ using a $\Omega(\ )$ bound and one often characterizes $t_{worst}(n)$ using a $O(\ )$ bound. The reason is that when discussing the best case one often wants to express how good (lower bound) the best case can be. Similarly when discussing the worst case one often wants to express how bad this worst case can be. I believe this is why `http://bigocheatsheet.com/` lists some best and worst case bounds using Ω and Θ respectively. However, it really depends what you want to say.

For example, it would still be mathematically correct to characterize $t_{best}(n)$ using a $O(\ )$ bound. For example, if we say that the best case of removing an element from an arraylist takes time $O(1)$, we literally mean that it is *not worse than* constant time. Similarly, if we say the worst case of quicksort is $\Omega(n^2)$, we would literally mean that the worst case takes *at least* that amount of time. So again, it really depends what you are trying to say. From what I've seen on various web posting and in textbooks, authors often are not clear about what they are trying to say when they use these terms, and in particular, people use $O()$ rather than Θ probably more than they should. Oh well...

Using "limits" for asymptotic complexity

The formal definitions of big O, Omega, and Theta require that we give specific constants n_0 and c. But in practice for a given $t(n)$ and $g(n)$ there many possible choices of constants and typically we don't care what the constants are. We only care that they exist. This is similar to the formal definition of a limit of a sequence from Calculus 1. There, you were asked whether a limit exists and what the limit is, but you were probably not asked for formal proofs. Instead you are given certain rules that you follow for determing the limits (l'Hopitals rule, etc).

Below I will state several rules that use limits for showing that one function is big O, big Omega, or big Theta of another function. In some cases, I will use the formal definition of a limit, so I will repeat that definition here:

A sequence $t(n)$ of real numbers converges to (or has a limit of) a real number t_∞ if, for any $\epsilon > 0$, there exists an n_0 such that, for all $n \geq n_0$, $\mid t(n) - t_\infty \mid \ < \epsilon$.

We would like to relate this definition to the definition of Big O, etc. Let's now write the definition of big O slightly differently than before, namely we consider the ratio of $t(n)$ to $g(n)$. The definition now is that $t(n)$ is $O(g(n))$ if there exist two positive numbers n_0 and c such that, for all $n \geq n_0$,

$$\frac{t(n)}{g(n)} \ \leq \ c \ .$$

Limit rule: case 1

Suppose that $t(n)$ and $g(n)$ are two sequences and that $lim_{n\to\infty} \frac{t(n)}{g(n)} = 0$. What can we say about their big relationship ? From the formal definition of a limit, we can say that for any $\epsilon > 0$, there exists an $n_0 > 0$ such that, for all $n \geq n_0$,

$$\mid \frac{t(n)}{g(n)} - \ 0 \mid < \epsilon$$

and in particular

$$\frac{t(n)}{g(n)} < \epsilon \ .$$

Since this is true for any $\epsilon > 0$, we take one such of these and call it c. This give immediately that $t(n)$ is $O(g(n))$. That is:

If $\lim_{n\to\infty} \frac{t(n)}{g(n)} = 0$, then $t(n)$ is $O(g(n))$.

Note that this rule is not equivalent to the definition of big O, since the implication (if then) does not work in the opposite direction, that is, if $t(n)$ is $O(g(n))$ then we cannot conclude that $\lim_{n\to\infty} \frac{t(n)}{g(n)} = 0$. A simple failure of this type is the case that $t(n) = g(n)$ since $t(n)$ would be $O(g(n))$ but $\frac{t(n)}{g(n)} = 1$ for all n and so the limit would not be 0.

Also note that the above rule for big O is weak in the sense we *only* conclude that $t(n)$ is $O(g(n))$. For example, the function $t(n) = 5n + 6$ is $O(n^2)$, but this is not so interesting since in fact $t(n)$ has a tighter big O bound, namely $O(n)$.

Here is a stronger statement than the above limit rule for the case that the limit is 0.

Limit rule: case 1 (continued)

If $\lim_{n\to\infty} \frac{t(n)}{g(n)} = 0$, then $t(n)$ is not $\Omega(g(n))$.

To prove this rule, we first note the definition for big Omega can be written equivalently as: there exist two positive numbers n_0 and $c > 0$ such that, for all $n \geq n_0$,

$$\frac{t(n)}{g(n)} \ \geq \ c \ .$$

Note that we do require $c > 0$. If we were to allow $c = 0$, then the definition would always hold for *any* positive sequences $t(n)$ and $g(n)$, which fail to express the lower bound.

Now for the proof: it should be obvious. The above inequality cannot hold when the limit $\frac{t(n)}{g(n)}$ is 0, since the limit of 0 means that the sequency would be less than c when n is sufficiently large. Thus, if the limit $\frac{t(n)}{g(n)}$ is 0, then it can't be that $t(n)$ is $\Omega(g(n))$.

Limit rule: case 2

If $\lim_{n\to\infty} \frac{t(n)}{g(n)} = \infty$, then $t(n)$ is $\Omega(g(n))$.

Saying that a limit of a sequence is infinity means (formally) that for any constant $\epsilon > 0$, the sequence will have values greater than that constant for n sufficiently large. i.e. No matter how big you make ϵ, you can always find some n_0 such that the sequence values will be greater than ϵ for all $n \geq n_0$. Note that the sequence in question here is $\frac{t(n)}{g(n)}$, and now we are thinking of ϵ as a big positive number.

The proof of this limit rule follows exactly the same as the case 1. We choose any ϵ (call it c) and we immediately satisfy the definition that $t(n)$ is $\Omega(g(n))$ since $\frac{t(n)}{g(n)} > 0$ for n sufficiently large.

Again note that the 'if-then' does not work in the opposite direction. The same counterexample above holds as before, namely if $t(n) = g(n)$. In this case, $t(n)$ is $\Omega(g(n))$ but $\frac{t(n)}{g(n)} = 1$ rather than infinity.

Limit rule: case 2 (cont.)

As in case 1, we can show here also that $t(n)$ and $g(n)$ are not in the same complexity class:

If $\lim_{n\to\infty} \frac{t(n)}{g(n)} = \infty$, then $t(n)$ is not $O(g(n))$.

We use the same argument idea as above: Suppose $\lim_{n\to\infty} \frac{t(n)}{g(n)} = \infty$. In order for $t(n)$ to be $O(g(n))$ we would need a $c > 0$ such that, for all n sufficiently large:

$$\frac{t(n)}{g(n)} \;\leq\; c \; .$$

But this contradicts the assumption that the limit is infinity. Thus, $t(n)$ is not $O(g(n))$.

Another possibility is that the limit exists but it neither 0 nor infinity. Let's look at this case next.

Limit rule: case 3

If $\lim_{n\to\infty} \frac{t(n)}{g(n)} = a$ and $0 < a < \infty$, then $t(n)$ is $\Theta(g(n))$.

To formally prove this rule, we need to use the definition of the limit: for any $\epsilon > 0$, there exists an n_0 such that, for all $n \geq n_0$,

$$\mid \frac{t(n)}{g(n)} - a \mid < \epsilon$$

or equivalently

$$-\epsilon < \; \frac{t(n)}{g(n)} - a \; < \epsilon$$

Take $\epsilon = \frac{a}{2}$. This gives the big O bound: for all $n \geq n_0$,

$$\frac{t(n)}{g(n)} < \frac{3a}{2}$$

and it gives the big Omega bound

$$0 < \frac{a}{2} < \frac{t(n)}{g(n)}.$$

Note this rule does not hold in the other direction. In particular, it can happen that $t(n)$ is $\Theta(g(n))$ but the limit does not exist and so the rule does not apply. For example, take the contrived example that

$$\frac{t(n)}{g(n)} = h(n)$$

where $h(n)$ is the sequence $2, 3, 2, 3, 2, 3 \ldots$. In this case, $lim_{n\rightarrow} \frac{t(n)}{g(n)}$ does not exist. Yet $t(n)$ is still $\Theta(g(n))$ in this case since the ratio $\frac{t(n)}{g(n)}$ is bounded both below (by 1) and above (by 4) for all n.

Ok, the math torture for over now, or perhaps you stopped reading a while ago). But if you like this sort of stuff, then I suggest you try the course MATH 242 Real Analysis. It is full of arguments like this.