We have seen several algorithms in the course, and loosely characterized the time it takes to run them 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 only the largest term that depends on n. This level of understanding of $O(\ )$ is usually good enough for distinguishing algorithms, but we would like to be a bit more formal about what this means.

Big O

Let $t(n)$ be a well-defined sequence of integers. This sequence often $t(n)$ represents the "time" or number of steps it takes an algorithm to run as a function of some variable n which often represents the size of the input, but this common interpretation is not what we'll be concerned with today.

Let $t(n)$ and $g(n)$ be two sequence of positive integers, where $n \geq 0$. We commonly consider $g(n)$ to be a relatively simple function such as:

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

although this is not necessary for the definitions we'll see today and next lecture.

Definition

We say that $t(n)$ *is asymptotically bounded above by* $g(n)$ if there exists a positive number n_0 such that, for all $n \geq n_0$,

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

That is, $t(n)$ becomes less than or equal to $g(n)$ when 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, namelly $n \geq 12$, so $t(n)$ is asymptotically bounded above by $6n$. Notice that the constant 6 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 $g(n) = 5.00001n$, althouth n needs to be quite large before $5n + 70 \leq 5.00001n$.

It is much more common to talk about a asymptotic upper bounds on $t(n)$ in terms of a simpler function $g(n)$, namely where we don't have constants in the $g(n)$. To do this, one needs changes the above definition slightly.

Definition (big O):

The sequence $t(n)$ is $O(g(n))$ if there exists 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 $g(n)$". What is nice about this definition is that we can keep $g(n)$ simple, by choosing our constant c appropriately.

Note also the we keep the condition $n \geq n_0$, for some n_0. This allows us to ignore how $t(n)$ compares with $g(n)$ when n is small. In this sense, it describes an *asymptotic* upper bound.

Example 1

The function $t(n) = 5n + 70$ is $O(n)$. To prove this, we write:

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

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 and/or n_0 and these constants with satisfy the definition also. So, don't think of the c and n_0 as being uniquely defined.
- 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? Not really. 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.
- We generally want our $g(n)$ to be simple. 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 $O(n^2)$ and it is also $O(n \log n)$ and $O(n^3)$, etc. I will return to this point next lecture.

Example 2

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

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

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

$$\begin{aligned} t(n) &= 17 - 46n + 8n^2 \\ &\leq 17 + 8n^2, \quad n > 0 \\ &\leq 17n^2 + 8n^2, \text{ if } n \geq 1 \\ &= 25n^2 \end{aligned}$$

and so $n_0 = 1$ and $c = 25$ does the job.

Here is a second proof:

$$\begin{aligned} t(n) &= 17 - 46n + 8n^2 \\ &\leq 17 + 8n^2, \quad n > 0 \\ &\leq n^2 + 8n^2, \textit{ if } n \geq 5 \\ &= 9n^2 \end{aligned}$$

and so $c = 9$ and $n_0 = 5$ does the job.

Here is a third proof:

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

and so $n_0 = 1$ and $c = 8$ does the job.

What does O(1) mean?

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. You will see an example in Assignment 2, Question 2d.

On writing proofs

Many of you are writing proofs for the first time and it is common that I read proofs that correct or incomplete. For example, consider the following.

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 not. Here is the sort of incorrect "proof" I sometimes see:

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

which is clearly true, i.e. $2n^2 \leq 4n^2$. So, does this prove the claim?

No, it doesn't. The claim is not true. Take $n = 3$ and you see that the inequality fails since $2 \cdot 3^2 > 4^2$. What went wrong is that the first line of the proof *assumes what you are trying to prove.*

Here is another example, which we had on the midterm test.

Claim: For all $n \geq 4, 2^n \leq n!$.

Some students gave "proofs" of the induction step as follows:

$$\begin{aligned} 2^{k+1} &\leq (k+1)! \\ 2 \cdot 2^k &\leq (k+1) \cdot k! \\ 2 \cdot 2^k &\leq (k+1) \cdot 2^k \quad \text{[ML : presumably by induction hypothesis]} \\ 2 &\leq k+1 \qquad \text{true if } n \geq 4 \text{ (base case)} \end{aligned}$$

I wouldn't say this proof is *incorrect*, but rather I would say that it is *incomplete*. The problem with this "proof" is that it isn't clear which statement implies which, or which statements are equivalent? I need to figure that out, and keep track. This is a burden, and it can more easily lead to errors.

In fact, the first and second statements have equivalent truth values. For any k, they are both true or they are both false. What about the second and third statement? Which implies which? Applying the induction hypothesis $2^k \leq k!$ would mean that the third statement implies the second, not vice-versa. What about the third and fourth statements? They have equivalent truth values: they are either both true or both false. One can see easily that the fourth statment is true (for $k \geq 4$), and thus the third statement is true, and thus the second is true, and thus the first is true, which is what we wanted to show.

A common type of incomplete big O proof

Consider again the claim that $5n + 70$ is $O(n)$. Here is a typical "proof" which we see on exams:

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

Like the previous example, this proof is incomplete because it isn't clear which statement implies which. The first statement may be true or false, possibly depending on n and c. The second statement is different than the first. It also may be true or false, depending on c. Because it is written second, this suggests that it *follows* from the first. Does it? Or does the second statement imply the first? Who knows? Such proofs may get grades of 0. This is not the big O that you want.