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 Cal courses, 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, which was more concerned with using rules about limits than understanding where these rules come from. But if you look back at your Calculus textbook then you’ll see the formal definitions were there. Moreover if you go further ahead in your study of mathematics then you will find this formal definition comes up again.

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 \( t_\infty \) means that the following holds: for any \( \epsilon > 0 \), there exists an \( n_0 \) such that for all \( n \geq n_0 \), \( |t(n) - t_\infty| < \epsilon \).

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. The statement is saying that if you take any finite interval centered at the \( t_\infty \), 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 \( n_0 \).

Here is the analogy to big O. We will have a set of rules that we can use to say that some function \( t(n) \) is big O of some other function e.g. \( t(n) \) is \( O(\log_2 n) \). These rules come from a formal definition of big O. We’ll look at that formal definition next. This formal definition has a similar flavour to the formal definition of the limit of a sequence in Calculus. But the formal definition of big O says something quite different. Next lecture, I’ll discuss the connection to limits again.

**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 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 $n \geq n_0$,

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

We say “$t(n)$ is big O of $g(n)$”. I emphasize: this definition allows us to keep the $g(n)$ simple by not having a constant factor as part of the $g(n)$.

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.

---

2Usually 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,

\[
\begin{align*}
t(n) &= 5n + 70 \\
&\leq 5n + 70n, \text{ when } n \geq 1 \\
&= 75n
\end{align*}
\]

and so \( n_0 = 1 \) and \( c = 75 \) satisfies the definition.

Here is a second proof:

\[
\begin{align*}
t(n) &= 5n + 70 \\
&\leq 5n + 6n, \text{ for } n \geq 12 \\
&= 11n
\end{align*}
\]

and so \( n_0 = 12 \) and \( c = 11 \) also satisfies the definition.

Here is a third proof:

\[
\begin{align*}
t(n) &= 5n + 70 \\
&\leq 5n + n, \text{ for } n \geq 70 \\
&= 6n
\end{align*}
\]

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. Don’t even think of them as the “best” \( c \) and \( n_0 \). The big O definition says nothing about “best”.

- 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.

- 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 \( O(n) \) but it is also \( O(n \log n) \) and \( O(n^2) \) and \( O(n^3) \), etc. I will return to this in the next few lectures.

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”.

last updated: 17th Oct, 2017 at 10:42
For example, here is a typical example of an incorrect “proof” of the above:

\[
\begin{align*}
5n + 70 & \leq cn \\
5n + 70n & \leq cn, \quad n \geq 1 \\
75n & \leq cn \\
\end{align*}
\]

Thus, \( c > 75, \ n_0 = 1 \) works.

This proof contains all the calculation elements of the previous proof. But this proof 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. This is not the big O that you want. Let’s turn to another example.

**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 \( n \geq n_0 \),

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

\[
t(n) = 8n^2 - 17n + 46 \\
\leq 8n^2 + 46n^2, \quad \text{for } n \geq 1 \\
= 54n^2
\]

and so \( n_0 = 1 \) and \( c = 54 \) does the job.

Here is a second proof:

\[
t(n) = 8n^2 - 17n + 46 \\
\leq 8n^2, \quad n \geq 3
\]

and so \( c = 8 \) and \( n_0 = 3 \) 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.