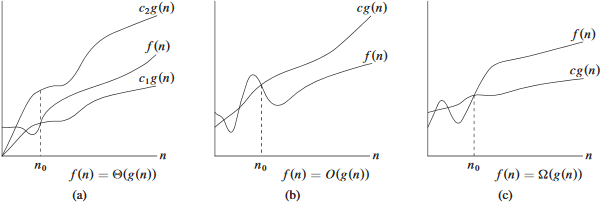
**Runtimes:**

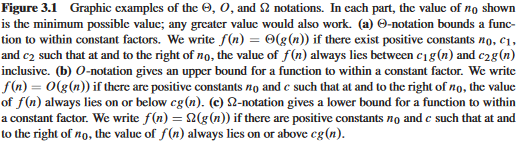
Remember big O and shit? Basically, we compare runtimes by comparing the number of steps taken. The steps are usually denoted with the letter ‘n’, which in big O is converted to some math formula like ‘’ or ‘’. With these, we usually see ‘n’ as being the “Worst case scenario” because let’s be honest, the worst-case scenario will always occur. ALWAYS.

**Difference between O, Omega, and Theta:**  
Big O provides us with the worst-case scenario, aka. What is the maximum time a program can run.

Big Omega on the other hand is the best-case scenario aka. What is the minimum time a program can take to run.

Big Theta basically refers to timeframes between ~~Omega and O~~. The small snitch glitch about Theta though is that it doesn’t really work for low inputs of n, ~~where the time can exceed O or be lower than Omega~~ (This presumably due to computation shenanigans, such as cashing and initialization of variables). Replace O and Omega with 2 instances of the same function, each instance having its own constant multiplied onto it. See below for somewhat clearer clarification. The short end of it is, that f(n) needs to at some point land between c1g(n) and c2g(n) without leaving that interval, regardless of how much n grows. Only then is Theta(n) applicable.





**Task1.1:** 

We know that has to fit the conditions . This means that we have to fit the task’s statement so that:

Here we have to keep in mind that for any n, max picks the resulting number from either f or g, based on whichever is bigger.

This means that for any n, f+g is always going to be bigger than max f,g. Therefore can be any number .

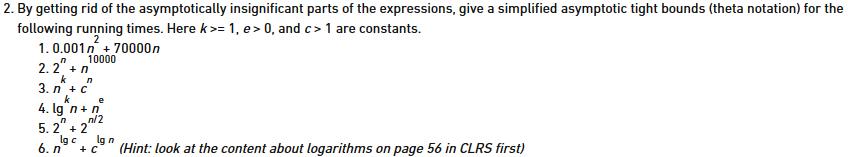
As for , we need it to always be to max. To accomplish this, we can just take the average of f and g for any n. This will allow max to always be to , and since we have 2 values added together, that means we just have to make . To summarize:

Bam, ezpz.

**Task1.2:** 

For we can rewrite it as . This means that all we have to do is pick any in order fulfill . The trick here is that could be written as multiplied by some constant, allowing us to set the bar for what c would have to be.

For the reason above, as cannot be written as , where c is a constant.

**Task 2:**

For reference, the above statements will be referred to as because I cannot be bothered to write them out.

Do this shit later, cannot be bothered to do it now.