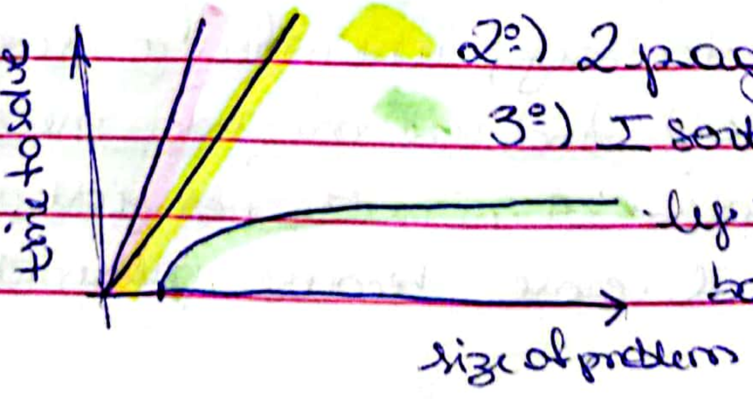
# Big O notation

capitalized O

this generally means that
the running time of some algorithm is on the order of such and such, where such and such, we'll see, is just going to be a very simple mathematical formula.

Way of waving your hands mathematically to convey the idea of just how fast or how slow some algorithm or code is, without getting into the weeds of like, it took this many miliseconds or this many specific number of steps.



1º algorithm (one page at a time)

2º) 2 pages at a time

3º) I sort of unnecessarily tore the phone book in half and them in half and .... which as dramatic as that was unnecessarily, it actually took significant bigger bites out of the problem.

So if we have (n) pages in that phone book, n just representing a generic number, the 1º algorithm here we might describe as taking n steps; 2º algorithm we might describe as taking by n/2 steps and then the 3º algorithm, if you remember about logarithms, was sort of a fundamentally different formula — log base 2 of n or just of n for short.

↳ So this is of a fundamentally ≠ formula.

If I start to zoom out and if I increase my y-axis and x-axis, these 1º two — the 1º and 2º algoritm — start to look awfully similar to one another.

These 1º two start to look awfully similar to one another. And if we keep zooming out and zooming out ... as n gets really large that is, the x-axis gets really long, these 1º two algorithms start to become essentially the SAME.

And so this is where computer scientists use big O notation, instead of saying specifically, this algorithm takes any steps. And this one (n) divided by 2, a computer scientist would say, each of those algorithms takes on the order n steps or on the order of n over 2.

But on the order of n over 2 is pretty much the same when n gets really large as being equivalent to big O of n itself.

So yes, in practice, it's obviously fewer steps to move twice as fast. But in the big picture, when n becomes a million, a billion, the numbers are already so darn big at that point that these are or, the shapes of these curves imply, pretty much functionally equivalent.

$O(\log_2 n)$

But this one still looks better and better as n gets large, because it's rising so much less quickly. And so here, a computer scientist would say that 3: algorithm was on the order of — that is, big O of — log n. And they don't have to bother with the base because it's a smaller mathematical detail that is also just in some sense a constant, multiplicative factor.

So in short, what are the takeaways here? this is just a new vocabulary that we'll start to use when we just want to describe the running time of an algorithm.

So make this more real, if any of you have implemented a FOR loop at this point in any of your code and that for loop iterated n times where maybe n was the height of your pyramid or maybe n was something else that you wanted to do n times, you wrote code or you implemented an algorithm that operated in big O of n time, if you will.

So this is just a way now to retroactively start describing with somewhat mathematical notation what we've been doing in practice for a while now.

. List of commonly seen running times in the real world:

$O(n^2)$
$O(n \log n)$
$O(n)$
$O(\log n)$
$O(1)$

} this is not a thorough list because you could come up with an ~~infinitude~~ infinite number of mathematical formulas, certainly.

But the common ones we'll discuss and you will see in your own code probably reduce to this list here. And if you were to study more computer science theory, this list would get longer and longer... But for now, these are sort of the most familiar ones that we'll soon see.

Two other pieces of vocabulary, if you will, before we start to use this stuff — so this, a big omega, capital

Digitalizada com CamScanner

omega-symbol, is used now to describe a lower bound on the running time of an algorithm.

## Asymptotic Notation ↓ → Complexity

So to be clear, big O is on the order of — that is, an upper bound — on how many steps an algorithm might take, on the order of so many steps.

How FEW steps my algorithm take?

Maybe in the so-called best case, it'd be nice if we had a notation to just describe what a lower bound is because some algorithms might be super fast in these so-called best cases.

So the symbology is almost the same, but we replace the big O ① with the big omega ②.

↓ describes an upper bound → limite superior

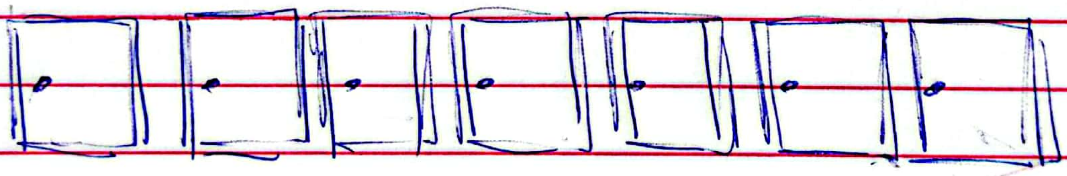
↓ describes a lower bound → limite inferior

Θ And then lastly, big theta ③, is used by a computer scientist when you have a case where both the upper bound on an algorithm's running time is the same as the lower bound.

You can then describe it in one breath as being in theta of such and such instead of saying it's in big O and in omega of something else. *

→ What is the input to each of these functions?

It is an expression of how many steps an algorithm takes. So in fact, let make this more concrete with an actual example here if we could.
So we have 7 lockers which represent an array of memory and this array of memory is maybe storing 7 integers that we might actually want to search for.



→ And if we want to search for these values, how might we go about doing this? Why don't we make things interesting? :)

↳ SEARCHING LOCKERS