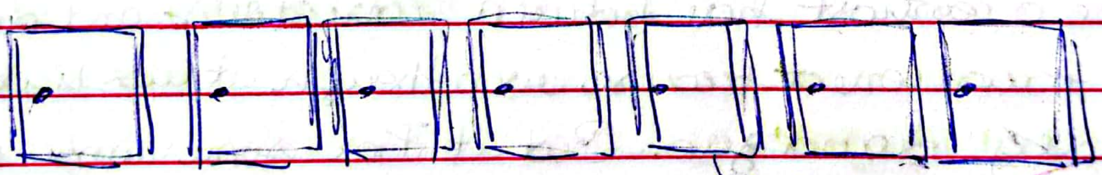
as the lower bound.

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

So here we have 7 lockers on an array of memory.

And behind of these doors is a number. And the goal, quite simply is, given this array of memory as input, to return, true or false, is the number I care about actually there?

So suppose I care about the number 0, what would be the simplest, most correct algorithm you could apply in order to find us the number 0?

→ Ok, so I kind of set you up for a fairly slow algorithm, but let me just ask you to describe what is it you did by following the steps I gave you...

" I just went one by one to each character..." "It's not the most efficient way to do it..."

* Some see a contrast here between correctness and design. I do think it was correct because even though it was slow, you eventually found zero. But it took some number of steps. So in fact, this would be an algorithm and has a name, called

Linear Search

" As you did, you did, you kind of walked along a line going from left to right. If you had gone from right to left, would the algorithm have been ~~[illegible]~~ fundamentally better? "

" Yes. "

" Ok, and why?"

" Because the zero is here in the 1º scenario, but if the zero is in the middle, it wouldn't have been."

" So in the general case, going left to right or right to left, is probably as correct as you can get because if you know nothing about the ORDER of these numbers - and indeed, they seem to be fairly random."

Linear Search is about as good as you can do when you don't know anything a priori about the numbers.

— PSEUDOCODE —

We just need a terse English or any language, syntax to describe what we did.

For each door, from left to right, if the number is behind the door, return true.

```
For each door, from left to right          }
    If the number is behind the door        } Pseudocode.
        Return true                         }
Return false
```

→ Else, at the very end of the program, you would return false by default.

Let's start to translate it from sort of higher level pseudo code to something a little LOWER LEVEL.

We've been writing code using (n) and loops and the like, get a middle ground between English and C:

```
For i from 0 to n-1
        If number behind doors [i]
                Return true
Return false
```

And notice this pattern here

name of this array

Now I'm kind of mixing English and C here, but that's reasonable if the reader is familiar with C or some similar language.

This is a way of just saying in pseudocode, give myself a variable called i, start at 0 and then just count up to n-1, and recall n minus 1 is not one shy of the end of the array.

n-1 is the end of the array because again, we started counting at 0.

So this is a very common way of expressing this kind of loop from the left all the way to the right of an array.

doors [i] means that when i is 0, it's this location [0], when it is 1, it's [1] and,,, when i = 6, $\boxed{n-1}$, the last one location. So it's the same idea but a translation of it.

So now let's consider what the running time of this algorithm is. If we have this menu of possible answers to this question, how efficient or inefficient is this algorithm?

Let's take a look in the context of this pseudocode. We don't even have to bother going all the way to C. How do we go about analyzing each of these steps?

1º For i from 0 to n-1

That line of code is going to execute how many times?

n times → Because it's from 0 to n-1

this is essentially the same mathematically as from 1 to n.

So this loop is going to operate n times!

2º If number behind doors [i]

How many steps or seconds does it take to ask a question?

→ It's a constant number of steps... Let's just call it one for simplicity. How many steps or seconds does it take to return true?

I don't know exactly in the computer's memory, but that feels like a single step, just return true.

It looks like you're doing a constant number of things n times or maybe you're doing one additional step.

So in short, the only thing that really matters here in terms of the efficiency or inefficiency of the algorithm is what are you doing again and again and again, because that's obviously the thing that's going to add up. *

Doing 1 thing or 2 things a constant number of times? Not a big deal!

But looping, that's going to add up over time because the more doors there are, the bigger n is going to be and the more steps that's going to take, which is all to say if you were to describe roughly how many steps does this algorithm take in big O notation, what might your instincts say?

" How many steps is this algorithm on the order of given n doors or n integers? "

$O(n^2)$	$\boxed{O(n)}$	$O(1)$
$O(n \log n)$	$O(\log n)$	

You're doing n things as an upper bound on running time. And that's, in fact, what exactly what happened. She had to look at all n lockers before finally getting to the right answer.

But what if she got lucky and the number we were looking for was NOT at the end of the array but was at the beginning of the array?

↳ OMEGA NOTATION → lower bound. ✓

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

} So given this menu of possible running times for lower bounds on an algorithm, what might the omega notation be for Namira's linear search?

↳ Because if just by ~~change~~ chance she gets lucky and the number she's looking for is right there where she begins the algorithm, that's it. It's ONE STEP. Maybe it's 2 steps if you have to unlock the door and open it, but it's a constant number of steps.

↳ the way we describe constant number of steps is just with a single number like $\textcircled{1}$. So the omega notation for linear search might be omega of 1 because in the best case, she might just get the number right from the get go.

But in the worst case, we need to talk about the UPPER bound, which might indeed be big O of n.

So again there's this way now of talking symbolically about best cases and worst cases or lower bounds and upper bounds.

Θ notation, just as a little trivia now, is it applicable based on the definition I gave earlier? No, because you only take out the theta notation when those 2 bounds, upper and lower, happen to be the same for short-~~[illegible]~~ hand notation, if you will.