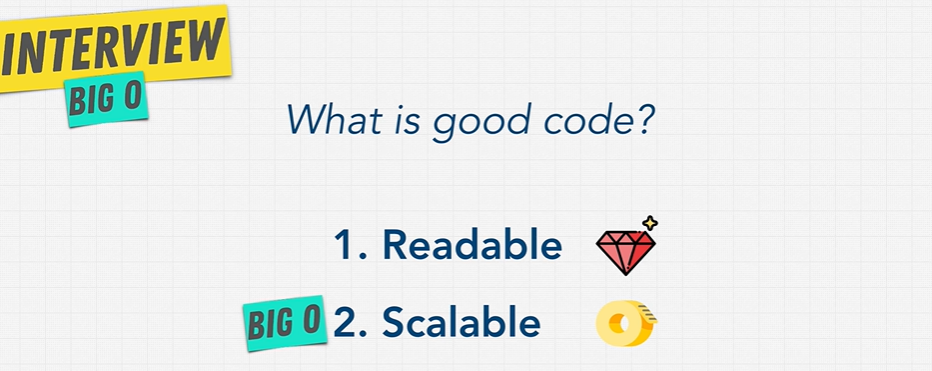
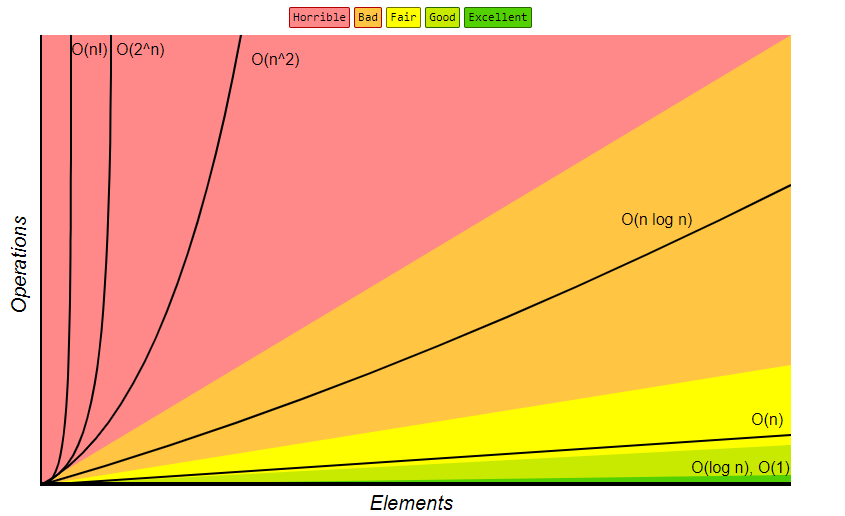
Big-O and Scalability -



Big notation is the language we use for talking about how long an algorithm takes to run. We can compare it to different algorithms or in this case, functions using big O and say which one

is better than the other when it comes to scale. Regardless of our computer differences.

We can measure Big-O like this:



When we talk about Big O and scalability of code, we simply mean **when we grow bigger and bigger with our input, How much does the algorithm or function Slow down.**

If the list of characters, let's say, elements here, so **as the array of elements increases,**

**How many more operations do we have to do?**

This is what we call algorithmic efficiency.

Big O allows us to explain this concept.

Different functions have different big-o complexities.

That is, these number of operations can increase really, really fast like this one, which is not

good.

You can see here that it's horrible and things that are quite good actually, and don't increase as

Much.

Just remember at this point, when we talk about big O and scalability of code, we simply mean **when we grow bigger and bigger with our input, how much does the algorithm slow down?**

The less it slows down or the slower it slows down, the better it is.

So instead of using performance now and using time to measure the efficiency of our function, we can just **calculate how many operations a computer has to perform** because each operation takes time on a computer.

So big O allows us and concerns us with **how many steps it takes in a function**.