COURSE NOTEBOOK

**SUMÁRIO**

[1 CAPÍTULO 3](#_Toc46487896)

# BIG O TIME COMPLEXITY

The very first thing we tackle Time and Space. Big O as they call it.

## What’s Big O

Big O is a way to categorize your algorithms time or memory requirements based io input. It’s not mean to be an exact measurement. It won’t tell you how many CPU cycles it takes, instead it’s meant to generalize the growth of your algorithm.

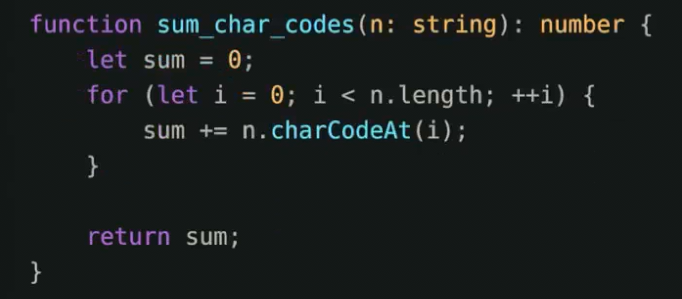
Example:

So, when someone says Oh of N, they mean your algorithm will grow linearly based on input.

## Why do we use it?

Often, it’ll help us make decisions about what data structures and algorithms to use. Knowing how they’ll perform can greatly help create the best possible program out there.

So, let’s do a small example, first, let’s consider the following code:



For those that know big O, this is easy but for those who have never even classified a function, this may be a complete mystery. That’s fine.

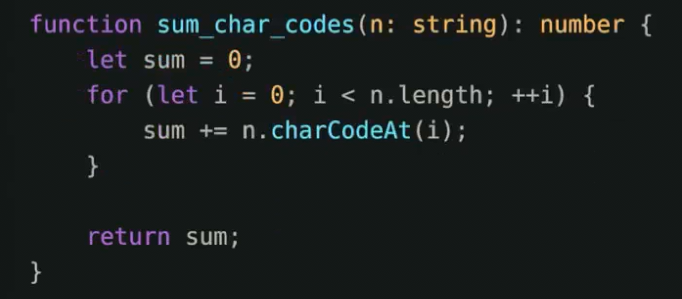
Big O, said differently as your input grown, how fast does computation or memory grow?

Important concepts

1. Growth is with respect to the input

In the real world obviously memory growing isn’t computationally free, but in the matter of thinking about algorithms, we don’t necessarily think about that. In languages like Go or JavaScript you pay ever heavier penalties because the memory can be kept around, grows faster and causes complete halts in your program for cleanup.

Let’s go back to our example, let’s look at input. How does our program’s execution time grow with respect to input?

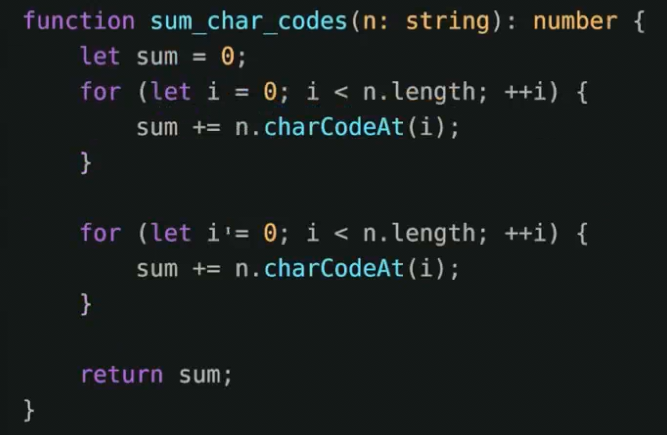


Me > It may or may not be obvious, but we have an N relationship. O(N) time complexity

You > How can you tell?

Simplest trick for complexity look for loops.

What’s the running time if the previous was O(N), what’s this?



Important concepts

1. Growth is with respect to the input
2. Constants are dropped

0(2N) -> 0(N) and this makes sense. That is because Big O is meant to describe the upper bound of the algorithm (the growth of the algorithm). The constant eventually becomes irrelevant.

Takes the following:

N = 1, O(10N) = 10, O(N^2) = 1

N = 5, O(10N) = 50, O(N^2) = 25

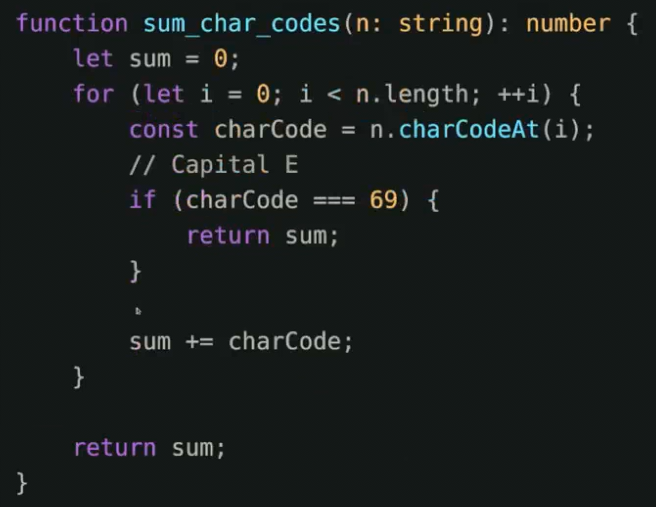
N = 100, O(10N) = 1,000, O(N^2) = 10,000 // 10x bigger

N = 10000, O(10N) = 100,000, O(N^2) = 100,000,000 // 1000x bigger

There are practical vs theoretical differences just because N is faster than N^2, doesn’t mean practically its always faster for smaller input.

Remember, we drop constants. Therefore O(100N) is faster than O(N^2) but practically speaking, you would probably win for some small set of input.

Let’s do another example:



In BigO we often consider the worst case especially in interviews (I have never been asked for best, average and worst case, just worst case).

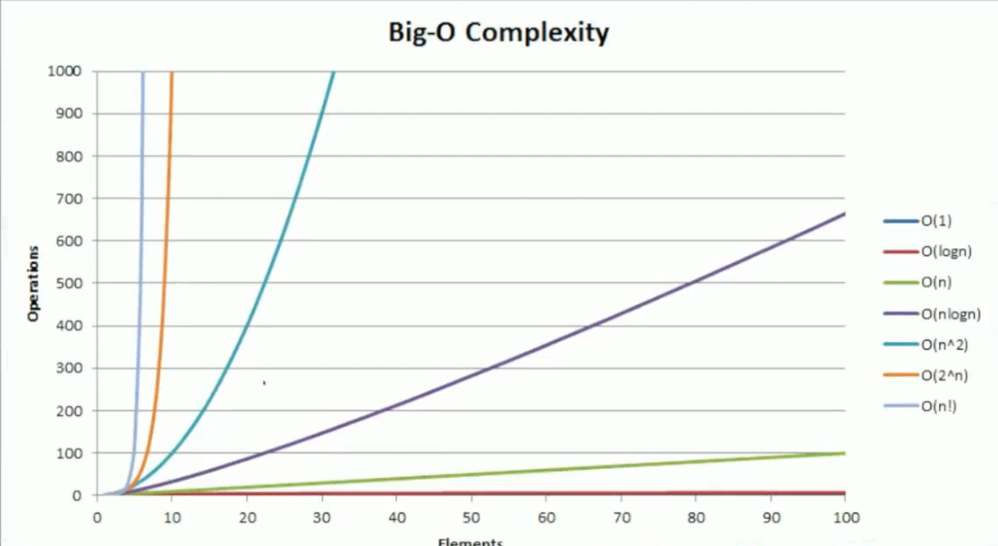
E = 69

Therefore, any string with E in it will terminate early (unless E is the last item in the list).

ITS STILL 0(N)

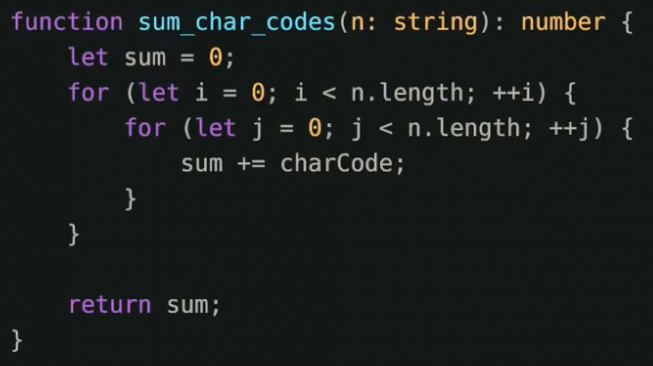
Important concepts

1. Growth is with respect to input
2. Constants are dropped
3. Worst case is usually the way we measure

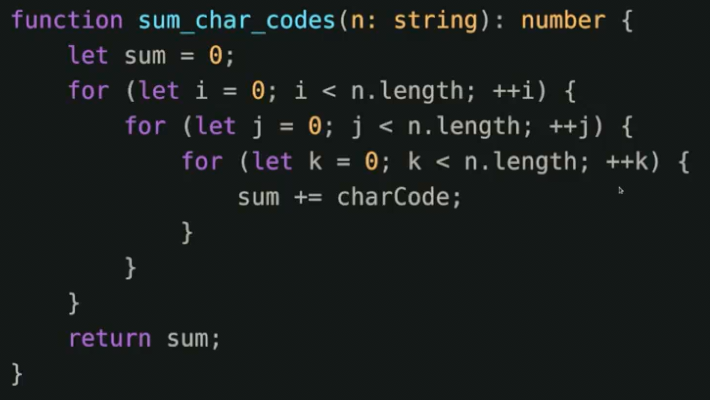


Some examples

O(N^2)



O(N^3)



O(n log n)

Quicksort (we will implement and explain)

O(log n)

Binary Search trees

There is one time that we’ll see today, that hasn’t been mentioned O(sqrt(n))

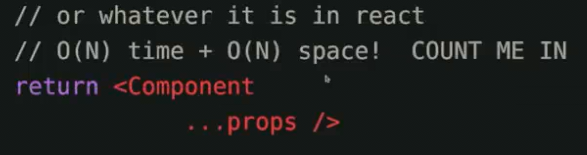
## Why so obviated?

There are other resources out there to dive deep into big O notation. And I just don’t think we need yet another Big O explanation. Instead, I am going to focus on actively looking at running times and we’ll determine things together.

Is there anything else besides Big O? Well, there is technically a bunch of different ways to measure the complexity of algorithms, but in generally the easiest one is the “Upper Bound”

Space the final frontier

We pretty much won’t be going over space in this course. Just not something we will be discussing. I find this less consequential in daily life considering I see things like this.



# ARRAYS DATA STRUCTURE

This is shockingly going to be a data structure you know, and it is one you often refer to, but likely incorrectly. Are you excited? Array

If const a = [] isn’t an array, what is it? Well array is a structure that store data in memory allocation, they are fixed sizes, contiguous memory chunks, that means that you can’t grow it, there is no “insertAt” or push or pop. Doesn’t mean you can’t write those though

Algorithms The A in DSA

When applicable

I’ll pretend that JavaScript actually has array

I only have access to the .lenght property of an array

No I wont go full C neckbeard and require length to be passed

# LINEAR SEARCH & KATA SETUP

Is a very and good practice to get into is being able to visualize the problem, discuss it with boxes and arrows, and then program it. Its definitively a core competency that will follow you for life.

Our first search! Possibly the simplest algo, Linear Search

Let’s implement it

To follow along I recommend using my kata typescript library

git clone [git@github.com:ThePrimeagen/kata-machine.git](mailto:git@github.com:ThePrimeagen/kata-machine.git)

<https://github.com/ThePrimeagen/kata-machine>

# BINARY SEARCH ALGORITHM

This second algorithm is a bit of a doosy, but its definitely a basis for other algorithms we will encounter, an important question we should all ask yourself is:

Is it ordered? Let’s say it is this time, So, how can we search this array?

## The two crystal ball problem

Given two crystal balls that will break if dropped from high enough distance, determine the exact spot in which it will break in the most optimized way.

# BUBBLE SORT

Yes! Sorting! Yes it wouldn’t be an algorithm course without at least a couple sorts! I know… we’ll implement a sorting algorithm (actually 2 of them).