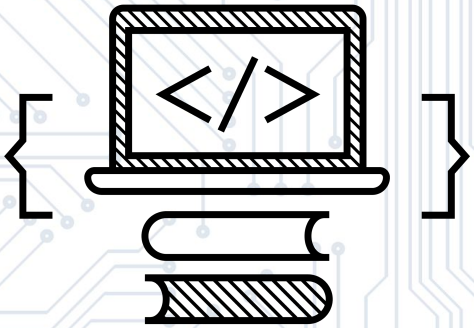


Big O Notation

Web Development Boot Camp

Lesson 11.5

Welcome to…
Computer Science Fundamentals!

Remember...

Computer science fundamentals

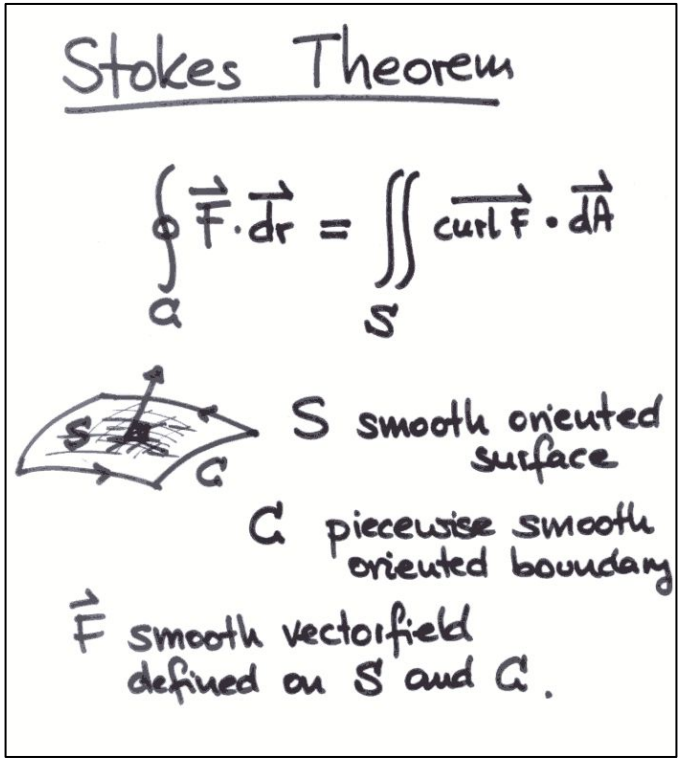
Fundamentals aren't the "easy" computer science stuff.

Rather, they are the fundamental concepts that underlie all of the work we've been doing to date.

The biggest takeaway is to understand that there are different tools to increase computational efficiency.

Fundamentals

Remember this stuff? Yeah, me neither.



Be Wary of Imposter Syndrome

Don’t let the hard stuff scare you.

Why Cover This?

01 These concepts sometimes appear in coding interviews.

02 When inheriting large codebases, you might be tasked to optimize code efficiency.

03 The computational challenges in this lesson force you to deepen your understanding.

Bottom Line

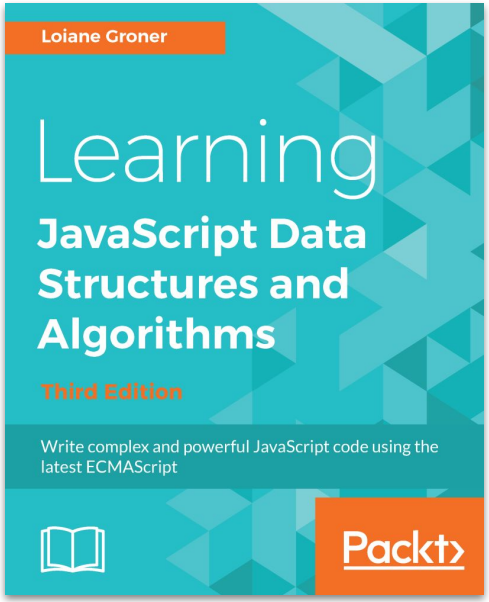
My goal is to give you the terminology and the concepts.

I want to give you enough insight so you can understand the context of interview questions that come your way.

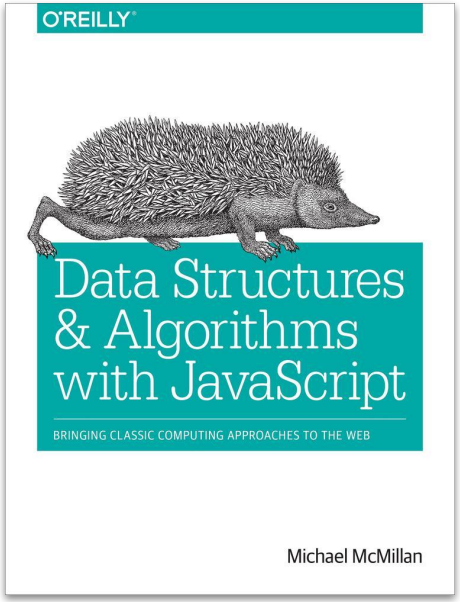
And to encourage those of you who are into math to take a second look!

Going Deep

For those who dare dive deeper



Learning JavaScript Data Structures and Algorithms—Third Edition
by Loiane Groner
Publisher: Packt Publishing
Release Date: April 2018



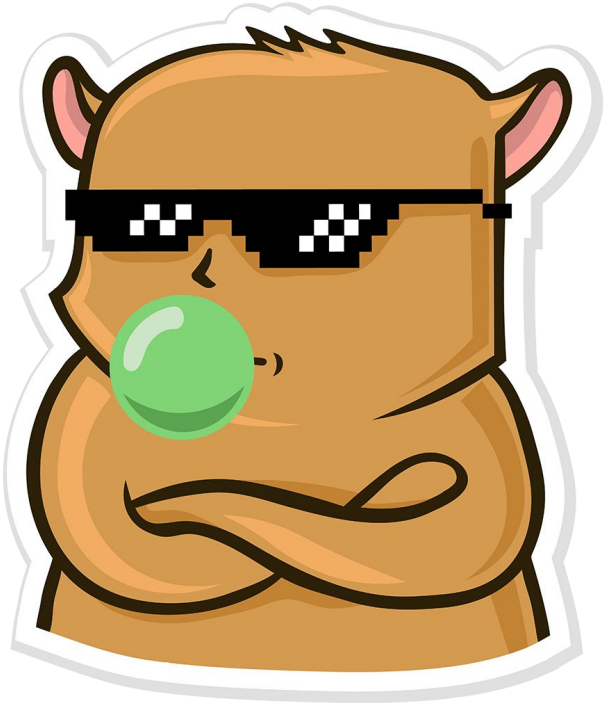
Data Structures and Algorithms with JavaScript
by Michael McMillan
Publisher: O'Reilly Media, Inc.
Release Date: March 2014

Efficiency

Activities 1 & 2

What does “Efficient” Mean?

We talk a lot about efficiency. But what exactly does “efficient” mean?

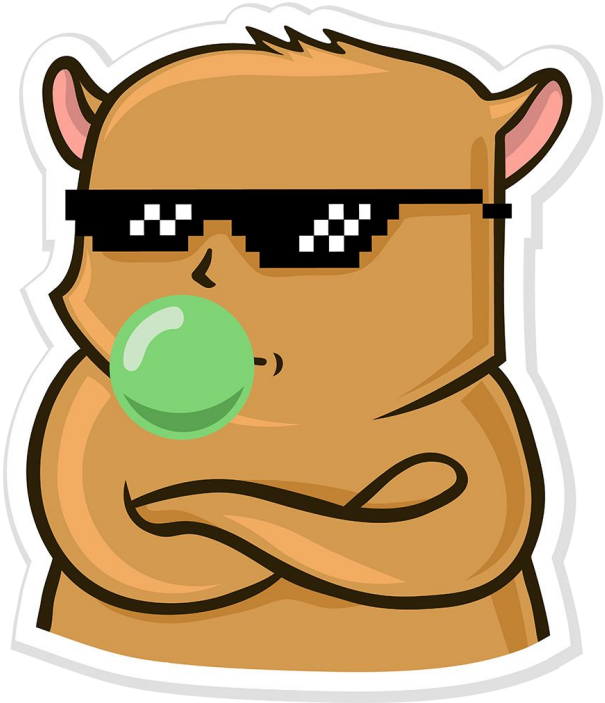


What's a Step?

- A step is an instruction to the computer.
- All computations boil down to a handful of basic steps.
  - Arithmetic (+, *, etc.)
  - Assignment (`var x = 42;`)
  - Boolean tests (`x === 42`)
  - Reading from memory
  - Writing from memory

What’s a Step?

Each of these counts as a step.



Some Common Sorting Algorithms

Activity 3

Time Complexity

Time complexity =
the rate at which algorithm **slows** as input **grows**.

Big O Notation

Big O

- Big O Notation lets us describe how running time scales when we increase the input size (n).
- It is denoted with a big O and the growth factor in parentheses.

Examples:

- `O(1)` Grows like “1” (i.e., running time never grows)
- `O(n)` Grows like “n” (i.e., gets bigger as n gets bigger)
- `O(n^2)` Grows like “n^2” (i.e., grows as square n as n gets bigger)
- `O(lg(n))` Grows like “log(n)” (i.e., grows as log2(n) as n gets bigger)

2 nested for loops ~ $O(n^2)$

NOT COINCIDENCE!

3 nested for loops ~ $O(n^3)$

Etc.

Big O:
One More...

Activity:

Binary search this array by hand, for 3, then 9.
Count the steps.

```
// Ready for binary search!
var sorted = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

Activity:

Binary search this array by hand, for 3, then 9.
Count the steps.

```
// Ready for binary search!
var sorted = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

Answer: 3 Steps

Add the digits 11–20. Repeat.

Add the digits 11–20. Repeat.

Answer: 4 Steps (!)

Much faster than linear.

Big O

$(\text{input size})^2 \sim 2\text{x running time}$
$(\text{input size})^3 \sim 3\text{x running time}$

Etc.

This is called O(lg n).

lg n = how many times do I divide n by 2 to get to 1?

Logarithm Example

What is lg 8?

Logarithm Example

$$8 / 2 = 4 \ (1)$$
$$4 / 2 = 2 \ (2)$$
$$2 / 2 = 1 \ (\mathbf{3})$$

Logarithm Example

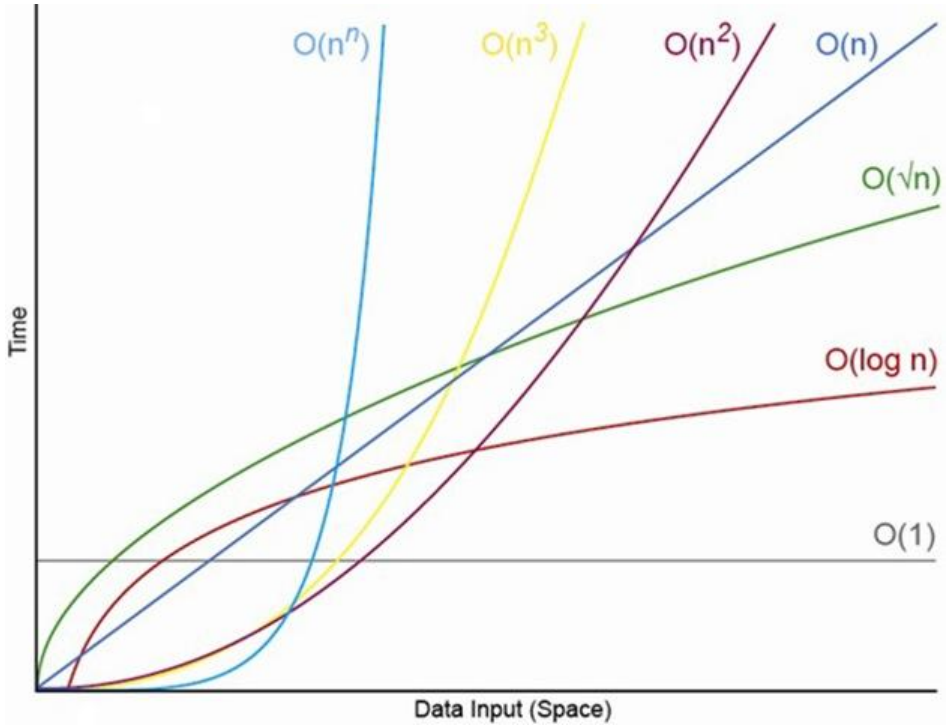
$$\lg 8 = 3$$

But if this is confusing…

Logarithm Example

Don’t worry about it.

Big O Comparisons



Asymptotic Analysis

`Examine the Code Base`	The function should exist and be observable.
`Count the Steps`	Remember what counts as a step? (Slide 11)
`Identify the maximum growth term`	Which step grew the fastest as input size changes?
`Simplify`	Remove any constants or coefficients.

Activity 4

Big O Review

`Calculation1.js` ~ O(1)	Grows like “1” (i.e., 2x input size -> 1x running time)
`Calculation2.js` ~ O(n)	Grows like “n” (i.e., 2x input size -> 2x running time)
`Calculation3.js` ~ $O(n^2)$	Grows like “n^{2}” (i.e., 2x input size -> 4x running time)
`Calculation4.js` ~ O(lg n)	Grows like “lg n” (i.e, $(\text{input size})^2$ -> 2x running time)

Questions?

Activity 5

Back to Projects!