Part 2

The Ingredients: Data Structures

Data structures are just containers. Each is fast at some things and slow at others. That's the whole game.

WHAT GOES INSIDE THE CONTAINERS

Containers hold values. In JavaScript, those values can be numbers, strings, booleans, objects, or even other containers. For this chapter, the important question is not the value's type. It is which container makes the operation you need easy.

Three containers do almost all the work in a JavaScript app: Array, Object/Map, and Set. Stack and Queue are patterns worth recognizing. Specialized structures live in Going Deeper.

Array: Numbered Parking Spots

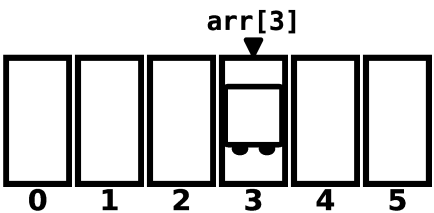
Speed: the workhorse

IN PLAIN ENGLISH

An ordered row of slots, each with a number starting at 0. If you know the slot number, access is O(1).

THE ANALOGY

A parking lot with numbered spots. "Get spot 3" goes straight there. "Find the red car" means walking the lot. Insert a new spot near the front and every later car must shuffle over.




```
const cars = ["kia", "bmw", "jeep", "fiat"];
cars[2]          // "jeep": O(1)
cars.push("vw")  // add at end: usually O(1)
cars.unshift("gmc") // add at front: O(n)
```

Adding at the end is usually O(1). Occasionally JavaScript must make a larger array and copy the existing items. Adding at the front is O(n) because every existing item must move one position.

RULE OF POCKET

Adding or removing an item at the end of an array is usually fast. Doing the same thing at the front makes the remaining items move.

SPOT IT IN THE WILD

The default container for lists of users, messages, and products. Reading an item by its position and adding or removing items at the end are usually fast. Front operations move the remaining items, and searching for a particular item may require checking the entire array.

Object/Map: The Coat Check

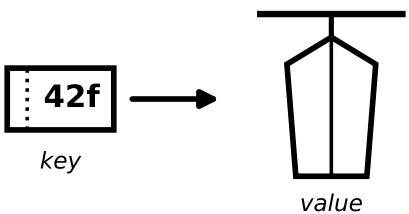
Speed: the superpower

IN PLAIN ENGLISH

Pairs of key and value. Hand over the key and get the value without a linear scan. Map access is expected near-constant time in ordinary use.

THE ANALOGY

A coat check. You do not count coats from the left. You hand over ticket 42f and the attendant uses the ticket to choose where to look.




```
const ages = { ada: 36, bob: 29 };
ages["ada"]      // 36
ages.cleo = 41;

const m = new Map();
m.set("ada", 36);
m.get("ada");    // 36
```

SPOT IT IN THE WILD

One of the most useful ideas in this book. When you repeatedly match by ID or name, a lookup structure replaces repeated scans with expected near-constant-time grabs. Part 5's first fix is built on this.

Set: The Guest List

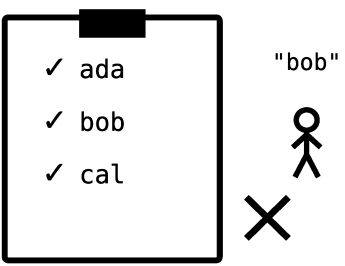
Speed: one job, done well

IN PLAIN ENGLISH

A collection of unique values. In ordinary use, membership checks are expected near-constant time.

THE ANALOGY

The bouncer's guest list. The only question is "are you on the list?" Adding "bob" twice still leaves one bob.




```
const seen = new Set(["ada", "bob"]);
seen.has("bob")  // true
seen.add("bob"); // still one "bob"

const uniq = [...new Set(list)];
```

SPOT IT IN THE WILD

"Have I seen this before?" checks: visited pages, used coupon codes, processed IDs. Also the cleanest built-in way to remove duplicate numbers or strings from an array.

Stack: The Plate Stack

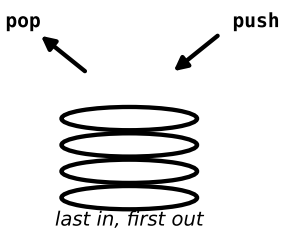
Speed: *a concept, not a class*

IN PLAIN ENGLISH

Add to the top and remove from the top. Last in, first out (LIFO). In JavaScript, an array used with `push`/`pop` is a stack.

THE ANALOGY

Like a stack of plates, the last item placed on top is the first one removed. Your browser's back button follows the same shape: each page goes on top, and Back takes the newest one off.



SPOT IT IN THE WILD

Undo/redo, back buttons, parsing nested brackets, and call stacks. You usually recognize the pattern and reach for `push`/`pop` rather than building a Stack class.

Queue: The Deli Line

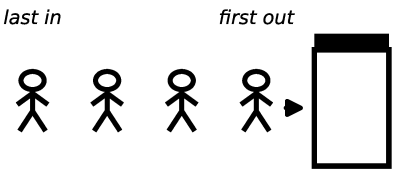
Speed: FIFO; O(1) with a proper queue

IN PLAIN ENGLISH

Add at the back, take from the front. First in, first out (FIFO).

THE ANALOGY

New customers join the back of the deli line; the counter serves whoever has waited longest.



SPOT IT IN THE WILD

A proper queue can add at the back and remove from the front in O(1). JavaScript's `shift()` moves every remaining array item one position, so it is O(n). For a large queue, keep a pointer to the front instead of moving every item.

Containers cheat sheet

Choose a container based on the job you need it to do most often. "Usually O(1)" means the work normally does not grow as the container gets larger.

container	what you are doing	cost
Array	Get an item by its position	O(1)
Array	Search for a particular value	O(n)
Array	Add or remove at the end	usually O(1)
Array	Add or remove at the front	O(n)
Object/Map	Get, add, or remove by key	usually O(1)
Set	Check, add, or remove a value	usually O(1)
Stack	Add or remove the top item	usually O(1)
Proper queue	Add at the back or remove from the front	O(1)

A proper queue removes from the front without moving all the remaining items.

See Problem 4 on page 79.

WHICH CONTAINER DO I WANT?

Ordered list? Array. Look up by name or ID? Object/Map. Seen-before check? Set. Undo? Stack. Process in arrival order? Queue. Hierarchy or relationships? Trees and graphs, in Going Deeper.

Keep Reading

Continue with The Self-Taught Developer's Guide to Big O Notation.

Buy the paperback on Amazon

amazon.com/dp/B0H7Z13T1D

Visit the book's website

thebigobook.com

Robert M. Cavezza