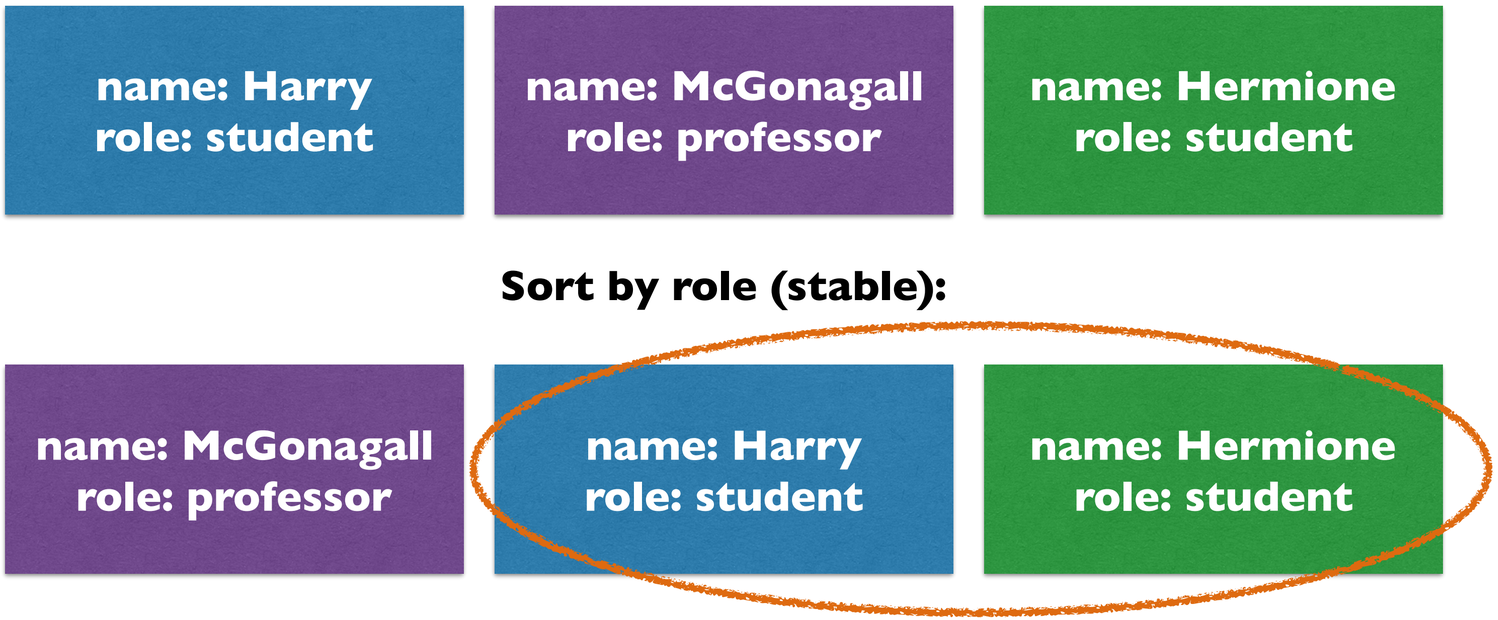
More Sorting

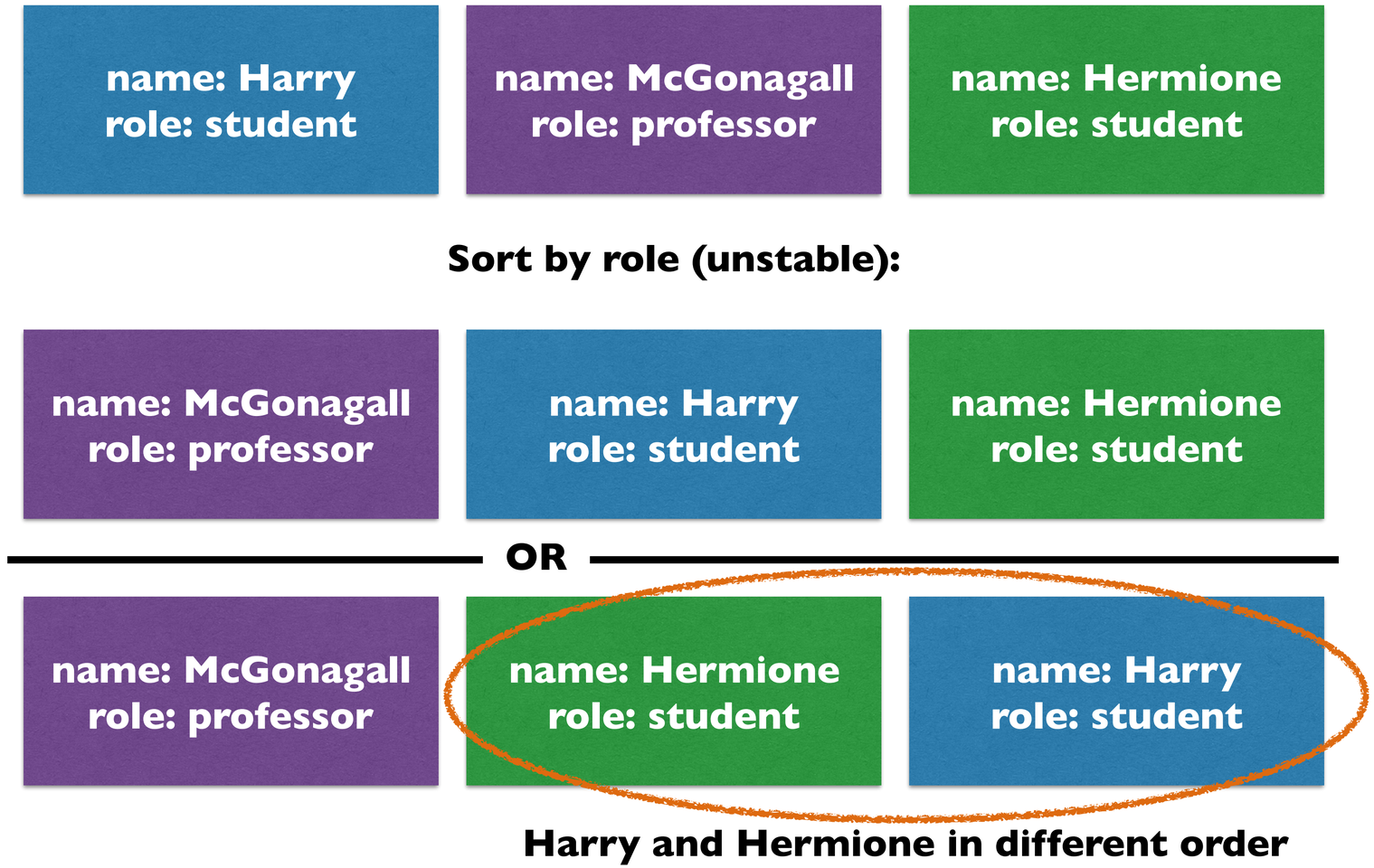
Terminology &c.

Stable vs. Unstable ...vs. anti-stable.

Stable Preserves Order of "Equal" Els



Unstable *Might Not* Preserve Order of "Equal" Els



Anti-Stable *Always Switches* Order of "Equal" Els

Sort by role (anti-stable):

Harry and Hermione in different order

Sorting Stability: (Some) Examples

Stable	Unstable*
Bubble	Quick†
Merge	Heap
Insertion	Selection
Bucket	Shell

* Any sort can be made stable with O(n) extra space

† If implemented in a standard way

WHAT ABOUT JS?

ES `.sort` is *not required* to be stable.

V8 (< v.70) `.sort` is ***un***stable.

V8 (≥ v.70) `.sort` is stable.

SpiderMonkey is stable.

In-Place

In-Place & In-Place Sorting

- An in-place algorithm uses only a ***small***, ***constant*** amount of extra space (*O(1) space complexity*) to achieve its goal

```
function sumArray (arr) {
  return arr.reduce(function (sum, el) { return sum + el; });
}
```

- As a **consequence** (but not summary!) of this definition, in-place ***sorting*** algorithms **mutate the input array**
  - This is intuitive; any sort that doesn't mutate the array must copy it, and if it copies the array then it has minimum O(n) space complexity.

Sorting Memory: (Some) Examples

In-Place (O(1))	Not In-Place
Bubble	**Merge: O(n)**
Heap	Quick: O(log(n) | n)
Insertion	Tim: O(n)
Shell	Cube: O(n)

WHAT ABOUT JS?

ES *doesn't require* .sort to be in-place.
But it ***does require*** it to mutate the array.

V8 .sort is ***not*** in-place.
But it ***does*** mutate the array.

(Note: many programmers misuse "in-place" to mean "mutates the array")

JavaScript Native Sort Summary

- **ECMAScript**
  - Must mutate input array
  - *Not required* to be stable (though it is allowed)
  - *Not required* to be in-place (though it is allowed)
  - Takes an optional comparator function which returns negative, 0, or positive num
- **V8 (Node, Chrome — but not other browsers)**
  - Hybrid approach — source code here
    - Insertion sort for small arrays (< 11)
    - Quicksort (<v.70) or Timsort (≥v.70) for larger arrays
  - Unstable (<v.70) or Stable (≥v.70)
  - Not in-place (but does mutate array!)

Bubble vs. Merge Sort, One More Time

	Bubble	Merge
Time Complexity	$O(n^2)$	$O(n \cdot \log(n))$
Space Complexity / In-Place	$O(1)$ → Yes	$O(n)$ → no
Stable	Yes	Yes

Other Sorting Considerations

- **Some sorts are far better or far worse when data is:**
  - Random
  - Nearly / already sorted
  - Backwards
  - Duplicated
- **Some sorts are significantly faster in the average case**
  - Quicksort is O(n^2) *worst*-case, yet is often preferred over merge sort (O(n · log(n)) because it can be implemented with less memory and faster *average* (i.e. typical) time!
- Click here for animations

Final Considerations

“The most important function of computer code is to communicate the programmer's intent to a human reader.”

ANY CODING PRACTICE THAT MAKES YOUR CODE HARDER TO UNDERSTAND IN THE NAME OF PERFORMANCE IS A PREMATURE OPTIMIZATION.

“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.”

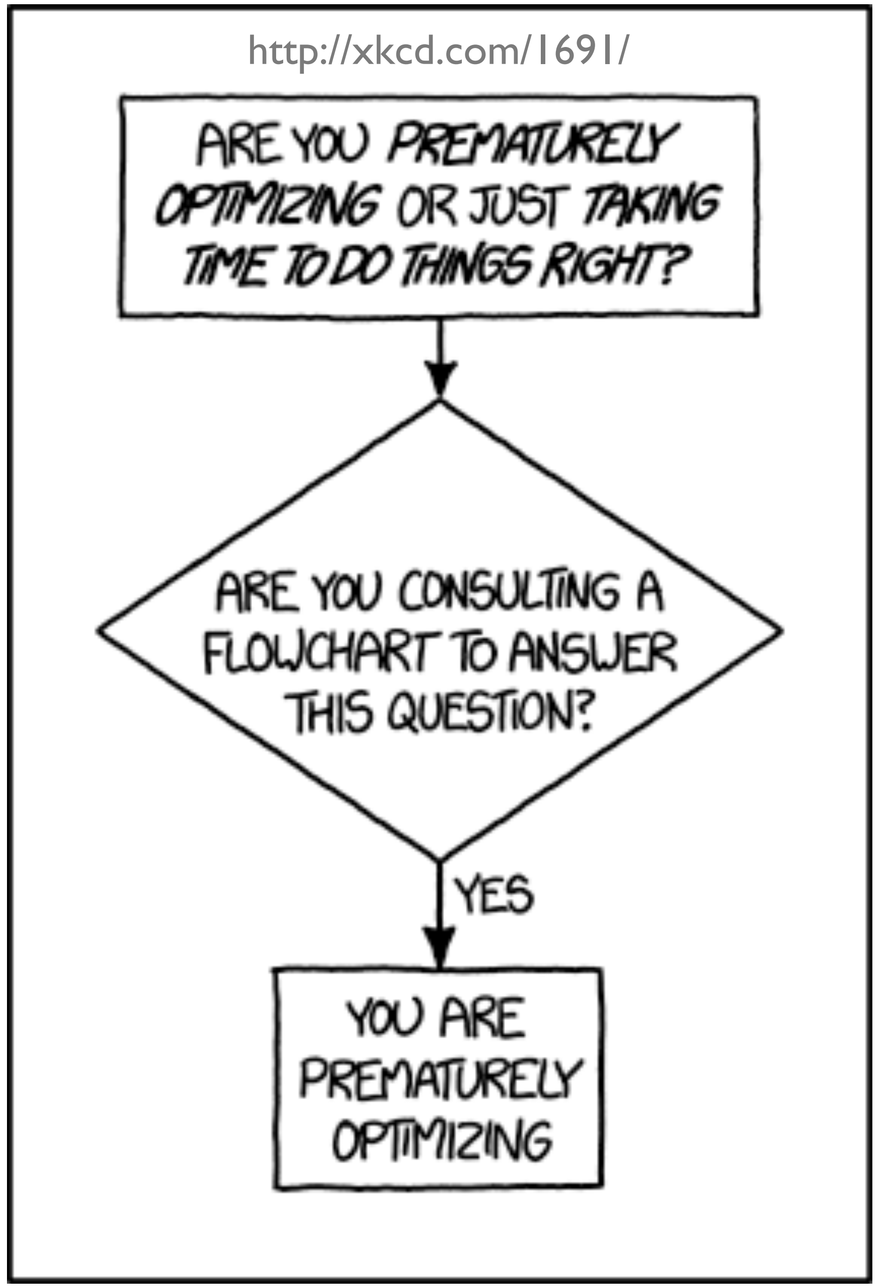
DO NOT SACRIFICE CODE CLARITY BY OPTIMIZING BEFORE YOU KNOW THAT YOU NEED TO.

Rob Pike's 5 Rules of Programming

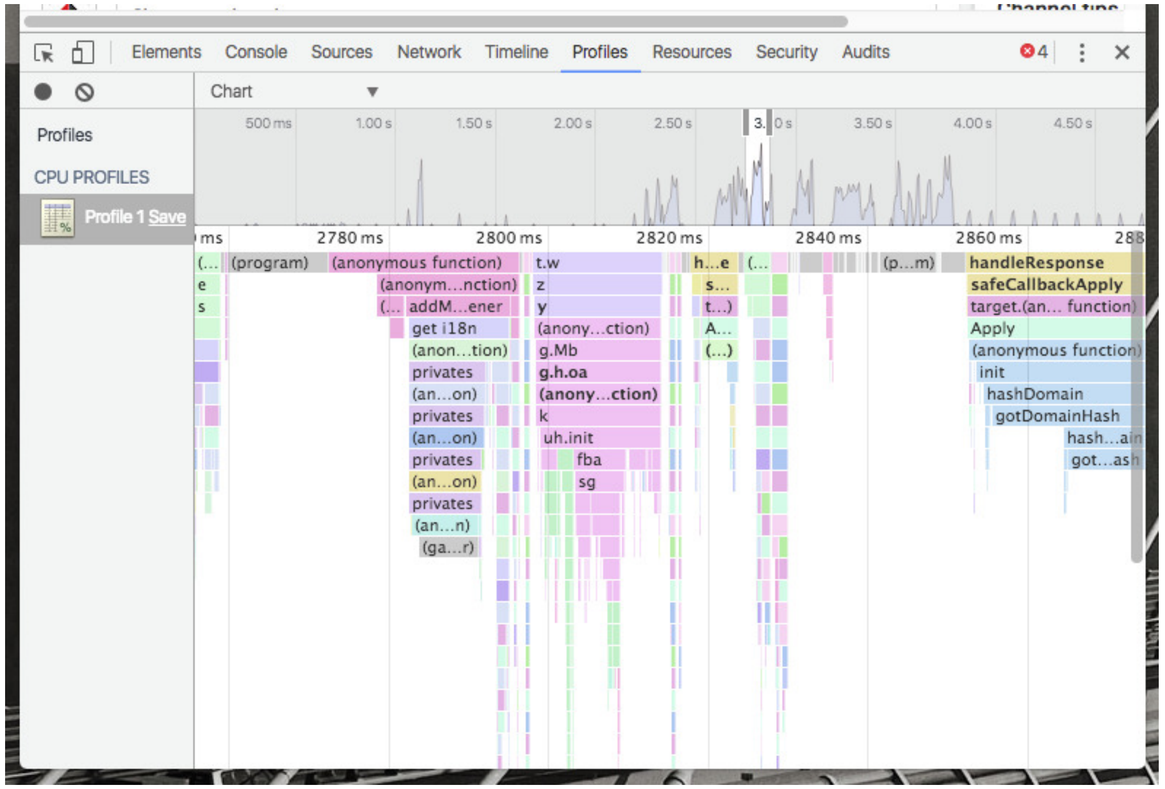
Bell Labs
Unix Team
UTF-8
Go Language
...and a lot more

1

- You can't tell where a program is going to spend its time. Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.



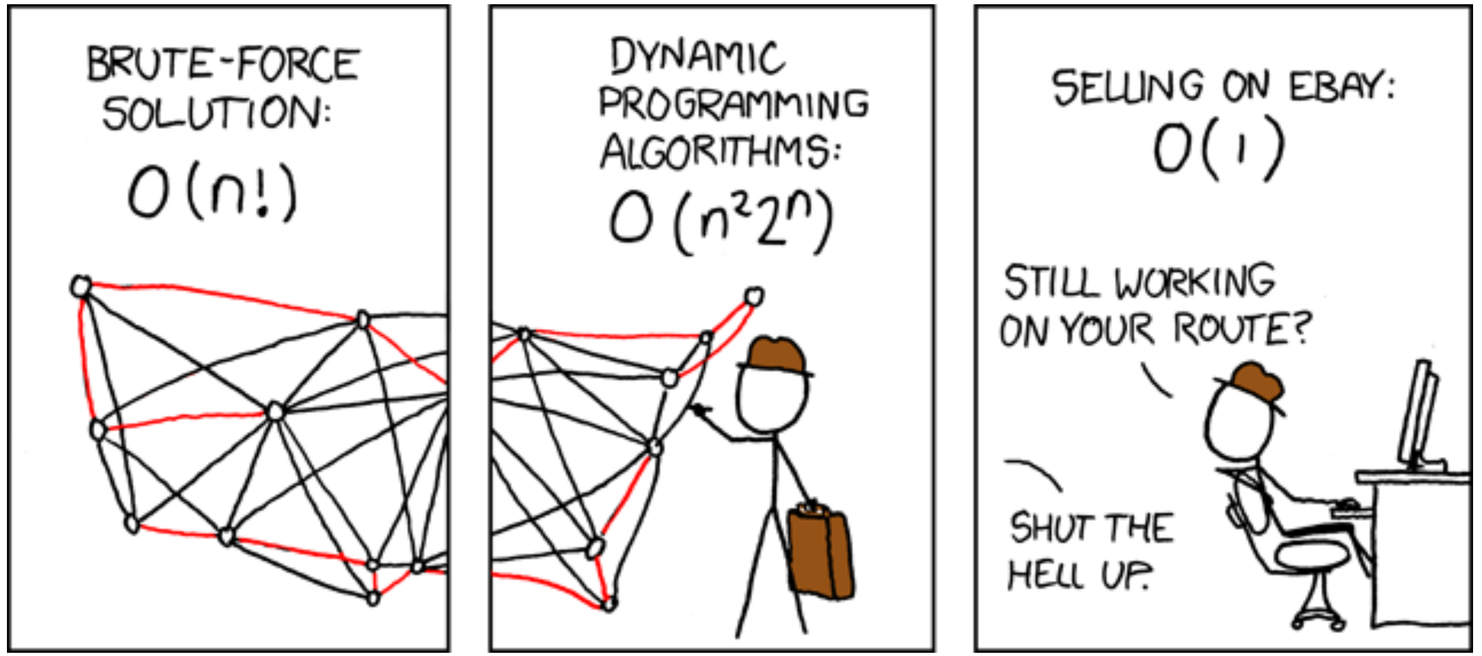
2



- Measure. Don't tune for speed until you've measured, and even then don't unless one part of the code overwhelms the rest.

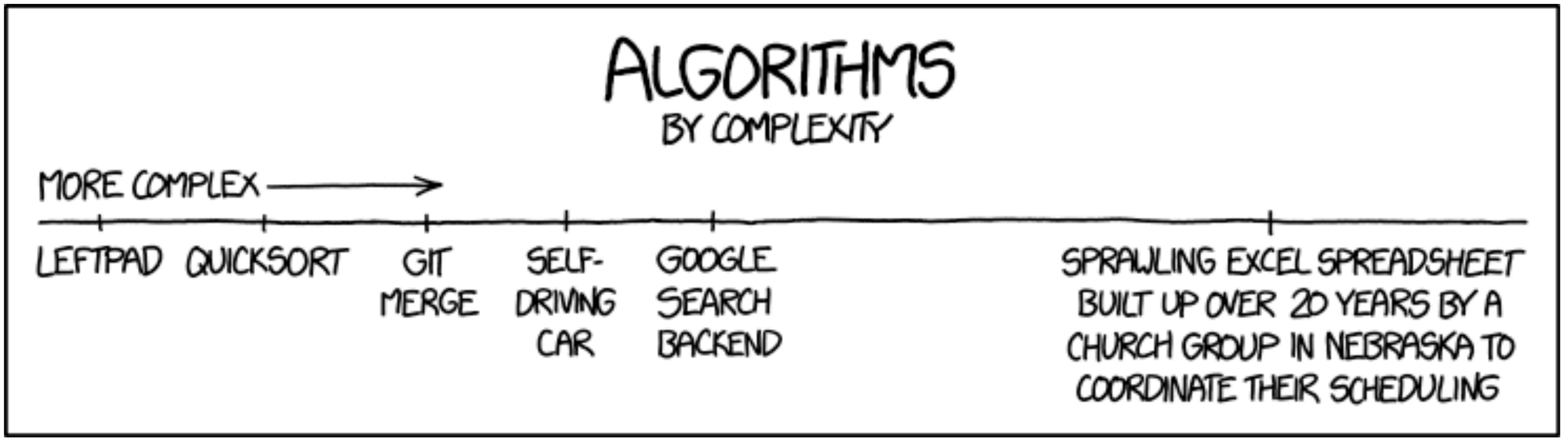
3

- Fancy algorithms are slow when n is small, and n is usually small. Fancy algorithms have big constants. Until you know that n is frequently going to be big, don't get fancy.



4

- Fancy algorithms are buggier than simple ones, and they're much harder to implement. Use simple algorithms as well as simple data structures.



5

- Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.