Big O Notation of your app

Vadim Drobinin | @valzevul

tl;dr

— Big O Notation
— Why bother?
— Measure and manage

Handouts

Slides and links to follow along:

drobinin.com/nsspain20

A mandatory disclaimer

What's Big O Notation?

Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity. It is a member of a family of notations invented by Paul Bachmann,[1] Edmund Landau,[2] and others, collectively called Bachmann–Landau notation or asymptotic notation.

In computer science, big O notation is used to classify algorithms according to how their running time or space requirements grow as the input size grows.[3] In analytic number theory, big O notation is often used to express a bound on the difference between an arithmetical function and a better understood approximation; a famous example of such a difference is the remainder term in the prime number theorem.

Big O notation characterizes functions according to their growth rates: different functions with the same growth rate may be represented using the same O notation.

The letter O is used because the growth rate of a function is also referred to as the order of the function. A description of a function in terms of big O notation usually only provides an upper bound on the growth rate of the function. Associated with big O notation are several related notations, using the symbols o, Ω, ω, and Θ, to describe other kinds of bounds on asymptotic growth rates.

Big O notation is also used in many other fields to provide similar estimates.

In computer science, big O notation is used to classify algorithms according to how their running time or space requirements grow as the input size grows.[3] In analytic number theory, big O notation is often used to express a bound on the difference between an arithmetical function and a better understood approximation; a famous example of such a difference is the remainder term in the prime number theorem.

Big O notation characterizes functions according to their growth rates: different functions with the same growth rate may be represented using the same O notation.

The letter O is used because the growth rate of a function is also referred to as the order of the function. A description of a function in terms of big O notation usually only provides an upper bound on the growth rate of the function. Associated with big O notation are several related notations, using the symbols o, Ω, ω, and Θ, to describe other kinds of bounds on asymptotic growth rates.

Big O notation is also used in many other fields to provide similar estimates.

- What's Big O Notation?

— A way of describing the efficiency.

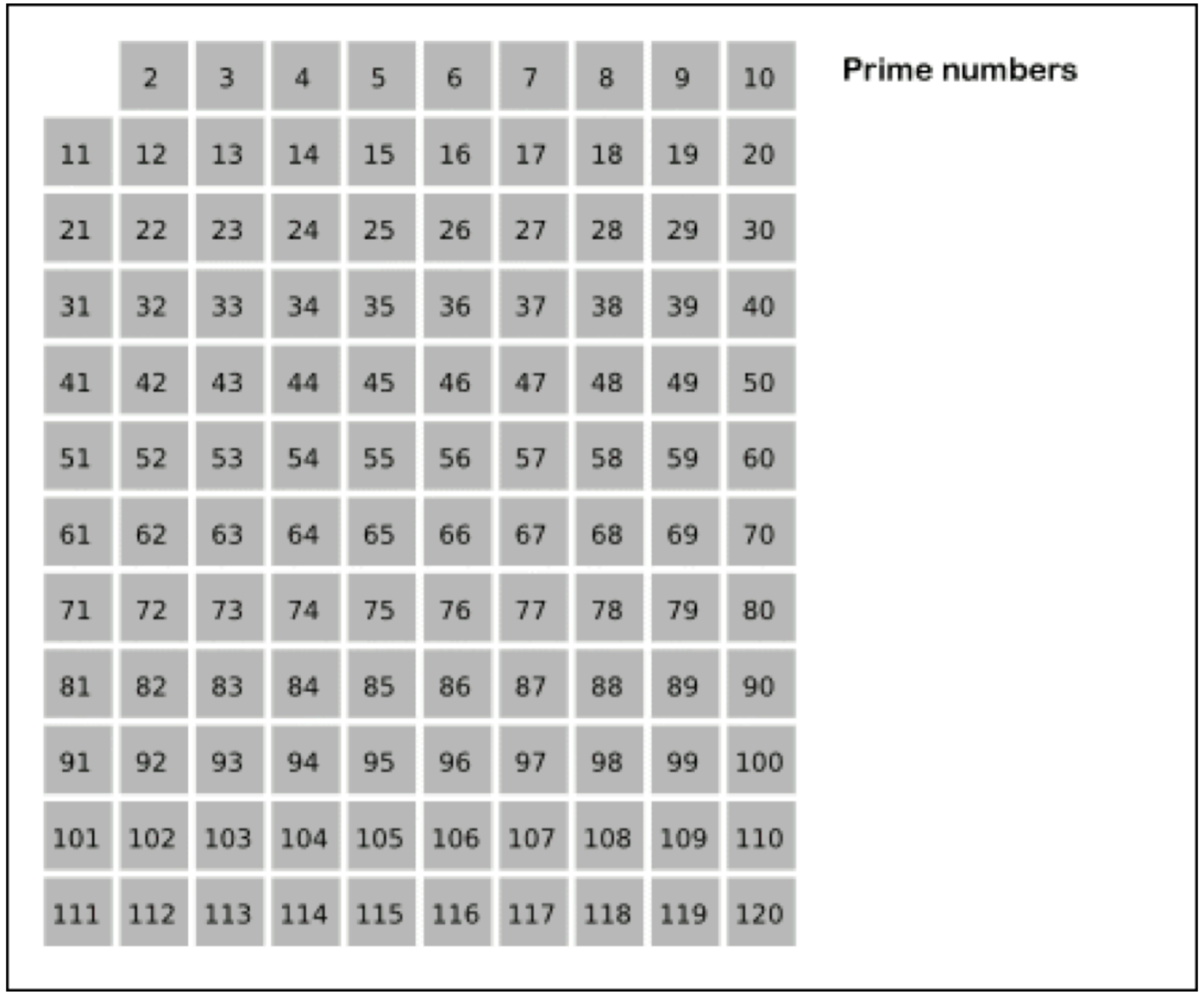
Big θ ⩽ Big O

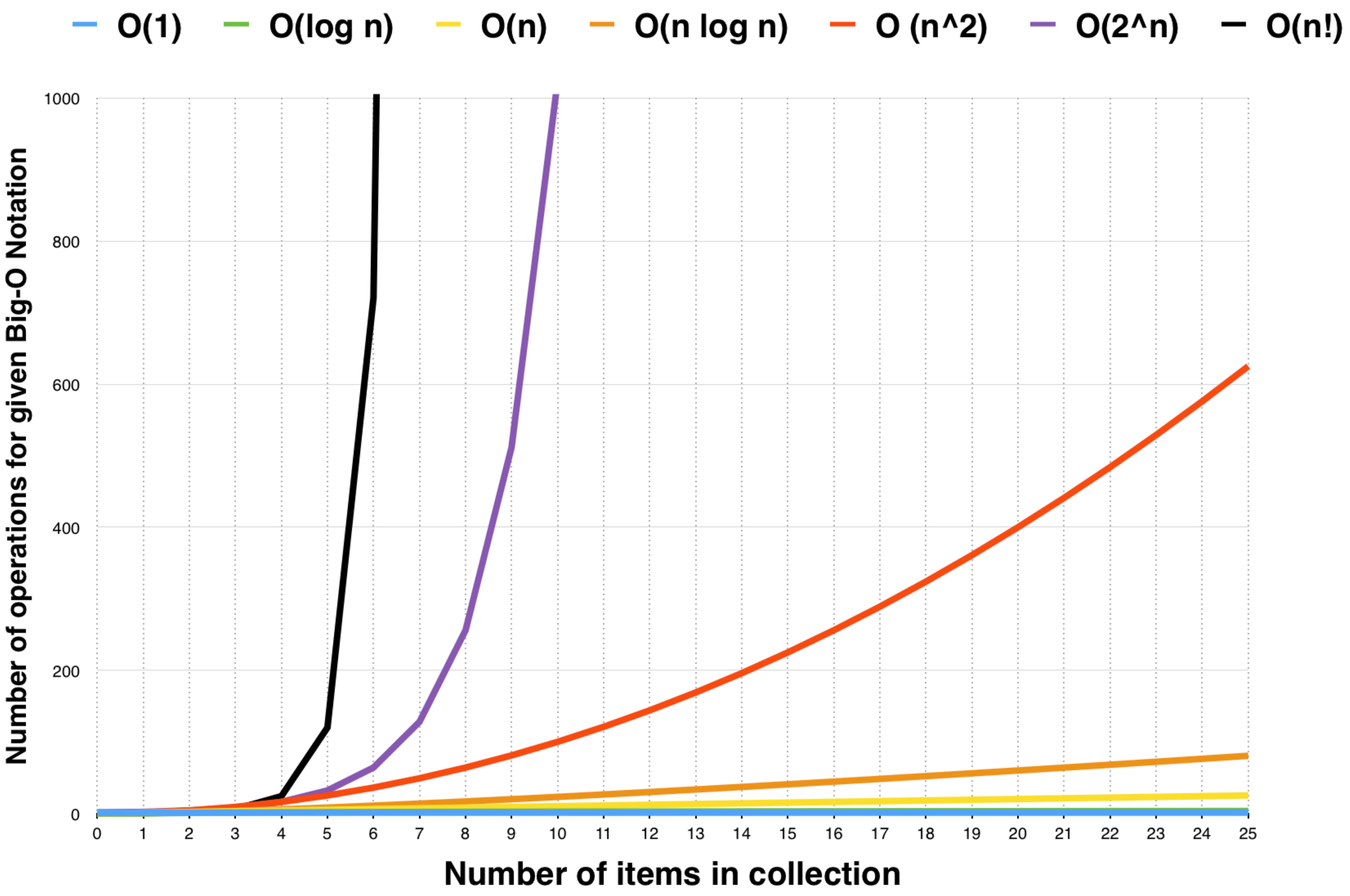
- Big O is an upper bound.
- Big θ is a tight bound (upper ***and*** lower).

The Sieve of Eratosthenes

An ancient algorithm for finding all prime numbers up to any given limit.

— ✅ $\mathcal{O}(n \log n)$
— ❌ $\Theta(n \log n)$
— ✅ $\Theta(n \log logn)$





O(1)
O(log n)
O(n)
O(n log n)
O (n^2)
O(2^n)
O(n!)
Number of operations for given Big-O Notation
1000
800
600
400
200
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Number of items in collection

What's Big O Notation?

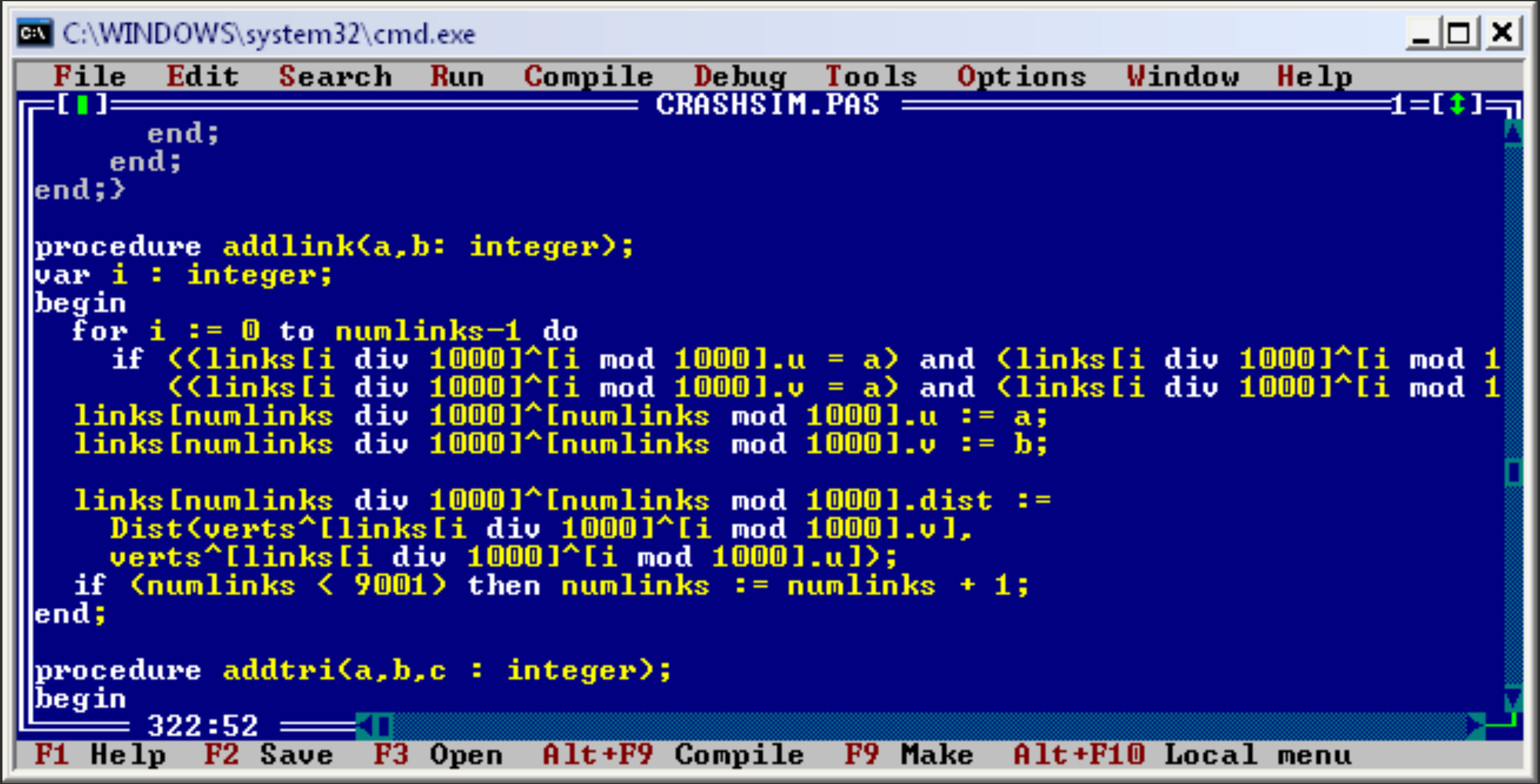
- $\mathcal{O}(1) \rightarrow$ the best
- $\mathcal{O}(\log n) \rightarrow$ pretty great
- $\mathcal{O}(n) \rightarrow$ good performance
- $\mathcal{O}(n \log n) \rightarrow$ decent performance
- $\mathcal{O}(n^2) \rightarrow$ kinda slow
- $\mathcal{O}(n^3) \rightarrow$ poor performance
- $\mathcal{O}(2^n) \rightarrow$ very poor performance
- $\mathcal{O}(n!) \rightarrow$ intolerably slow

Why bother?

Maybe...

Algorithms are essential to whiteboard interviews? *

* they're not; and if you interview people, please don't ask them to code on a whiteboard


C:\WINDOWS\system32\cmd.exe
File Edit Search Run Compile Debug Tools Options Window Help
CRASHSIM.PAS
end;
end;
end;}
procedure addlink(a,b: integer);
var i : integer;
begin
for i := 0 to numlinks-1 do
if ((links[i div 1000]^[i mod 1000].u = a) and (links[i div 1000]^[i mod 1
((links[i div 1000]^[i mod 1000].v = a) and (links[i div 1000]^[i mod 1
links[numlinks div 1000]^[numlinks mod 1000].u := a;
links[numlinks div 1000]^[numlinks mod 1000].v := b;
links[numlinks div 1000]^[numlinks mod 1000].dist :=
Dist(verts^[links[i div 1000]^[i mod 1000].v],
verts^[links[i div 1000]^[i mod 1000].u]);
if (numlinks < 9001) then numlinks := numlinks + 1;
end;
procedure addtri(a,b,c : integer);
begin
322:52
F1 Help F2 Save F3 Open Alt+F9 Compile F9 Make Alt+F10 Local menu

All really useful algorithms are either on Github and StackOverflow, or already in system frameworks

— Some developers

Binary Search from StackOverflow

```
public func binarySearch<T: Comparable>(_ a: [T], key: T) -> Int? {
    var lowerBound = 0
    var upperBound = a.count
    while lowerBound < upperBound {
        let midIndex = (lowerBound + upperBound) / 2
        if a[midIndex] == key {
            return midIndex
        } else if a[midIndex] < key {
            lowerBound = midIndex + 1
        } else {
            upperBound = midIndex
        }
    }
    return nil
}
```

Binary Search from StackOverflow

```
public func binarySearch<T: Comparable>(_ a: [T], key: T) -> Int? {
    var lowerBound = 0
    var upperBound = a.count
    while lowerBound < upperBound {
        let midIndex = lowerBound + (upperBound - lowerBound) / 2
        if a[midIndex] == key {
            return midIndex
        } else if a[midIndex] < key {
            lowerBound = midIndex + 1
        } else {
            upperBound = midIndex
        }
    }
    return nil
}
```

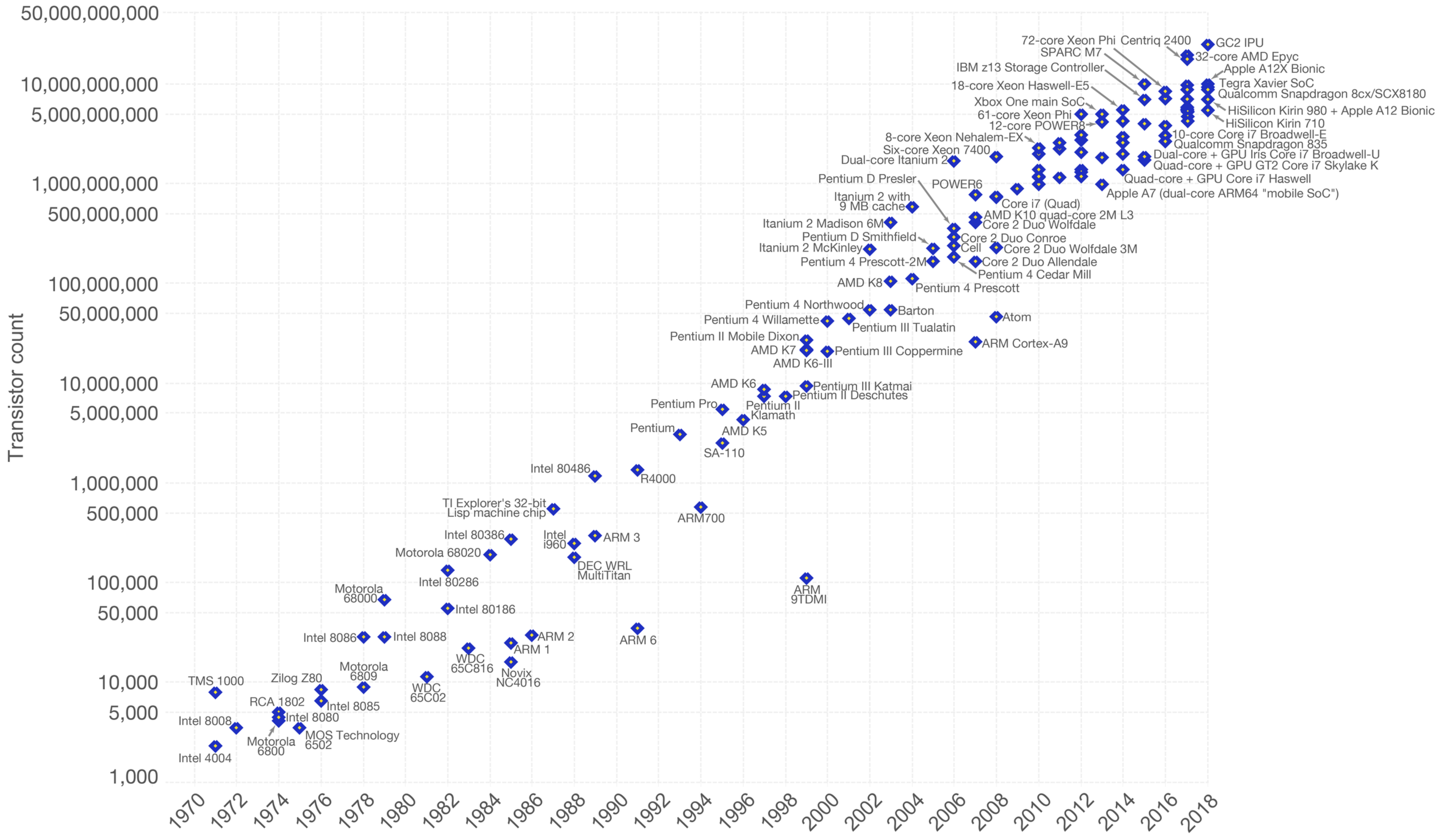
Modern devices are way too powerful for users to notice a difference between ***kinda slow*** and ***decent performance*** algorithms

— Some developers

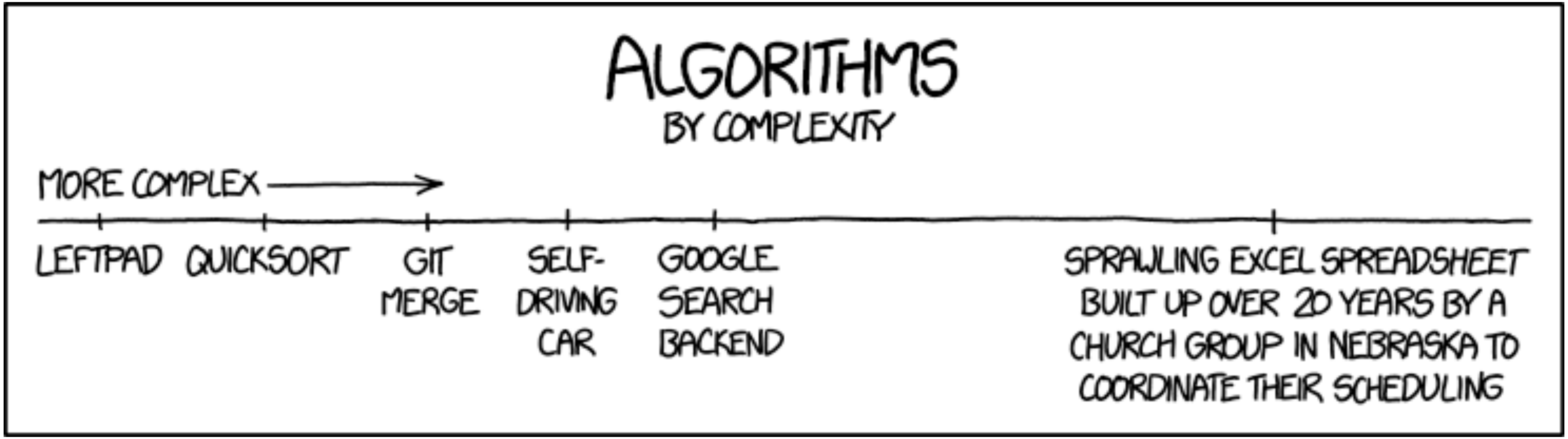
Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

Our World in Data



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.


ALGORITHMS
BY COMPLEXITY
MORE COMPLEX
LEFTPAD
QUICKSORT
GIT MERGE
SELF-DRIVING CAR
GOOGLE SEARCH BACKEND
SPRAWLING EXCEL SPREADSHEET BUILT UP OVER 20 YEARS BY A CHURCH GROUP IN NEBRASKA TO COORDINATE THEIR SCHEDULING

Real-world examples?

Show only unique messages in a chat history

Remove duplicates from an array.

```
// Source: https://stackoverflow.com/a/35014912

func removeDuplicates() -> [Element] {
    var result = [Element]() // O(1)
    for value in self { // O(n)
        if result.contains(value) == false { // O(n) + O(1)
            result.append(value) // O(1)
        }
    } // O(n * [n + 1 + 1]) = O(n^2)
    return result
}
```

Show only unique messages in a chat history

Remove duplicates from an array.

```
// Source: https://stackoverflow.com/a/46354989

func removeDuplicates() -> [Element] {
    let seen = Set<Element>() // O(1)
    self.filter { // O(n)
        seen.insert($0).inserted  // O(1)
    } // O(1 + n + 1) = O(n)
}
```

And many more

— Reduce storage in a navigation app
— Draw a route in a mobile game
— Snap video preview to a screen edge
— Hashing / Cryptography
— Low-level performance issues (i.e dropping frames)

Measure & Manage

Big O Notation of your app

1. "A way of describing the efficiency" → "Performance"
2. The amount of useful work accomplished estimated in terms of time needed, resources used, etc (Wictionary)
3. Reverse estimation: fix the work and optimise for time
4. Both software and user experience

Big O Notation of your app

— Number of taps

— Seconds of loading

— Actual code execution metrics

— Both memory consumption and time efficiency

Big O Notation of your app

- Kinda slow:
  $\mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2)$
- Poor performance:
  $\mathcal{O}(n^2) * \mathcal{O}(n) = \mathcal{O}(n^3)$

To Sum up

— Remember the difference between Big O and Big Θ

— Be careful copying code from the Internet

— Optimizing algorithms is not the only way to care about your users

— Evaluate your app's efficiency using Big O Notation as an inspiration

Further Reading

1. Green Development: Is it a thing [1]
2. Measure the performance of code in Swift [2]
3. Practical Approaches to Great App Performance [3]
4. Ukkonen's suffix tree algorithm in plain English [4]
5. codeforces.com

---

[1] https://aleksandra.tech/talks/2019/pragmaconf-green-development-is-it-a-thing-v2/

[2] https://www.avanderlee.com/optimization/measure-performance-code/

[3] https://developer.apple.com/videos/play/wwdc2018/407/

[4] https://stackoverflow.com/a/9513423

Learn about algorithms because it is fun, not because of interviews

Let's stay in touch 👇

@valzevul
drobinin.com

👇 Also we're actively hiring at Epsy Health 💜