Why I left Big-O in the dustbin when I left school

Yao Yue

Why was Big-O introduced into computer science?

“To approximate and classify algorithms according to how their run time or space requirements grow as the input size grows”

-wikipedia

Is it true?

Binary search is *faster* than linear search: $O(\log n) < O(n)$

Hashmap insertion is *faster* than skip list: $O(1) < O(\log n)$

Coppersmith-Winograd is *faster* than naive matrix multiplication: $O(n^{2.37}) < O(n^3)$

As a member of the Bachmann–Landau notations, $O()$ predates computers.

The big lie:

scaling is smooth

Why Big-O fails to predict reality

Big-O: asymptotic convergence as input grows toward **infinity**

Reality: finite, often of quite small a scale

Big-O: assumes uniform computation cost

Reality: varied, with interferences, steps, and saturation points

Sure, but what’s the harm?

- Gravitating toward algorithms that are theoretically good but practically bad

- Thinking there is “one true winner” for all scenarios

What to do (instead)?

Decide the scale range of the problem: S: $[s_{min}, s_{max}]$

Identify dominant operation(s): p, and their cost $C(p, s)$

- Cost is a function of *both* operation p and scale s

Total cost model

- Often a piecewise function
- Dependent on multiple operations

What you should actually do

Start with just about any seemingly reasonable choice

Calculate cost leader(s) by tying resource utilization telemetry to “unit of work”, e.g. *rps*

Identify nature of bottleneck(s)—lower algorithm complexity matters only if compute is the bottleneck!