S O T N I R G

Algorithms & Analysis

But first: how many pebbles?

Example Heuristic Approach

- Take a sample for a small area
- Multiply sample by total area / small area
- Probably not correct, but also probably close enough for us to do useful stuff


HEURISTIC

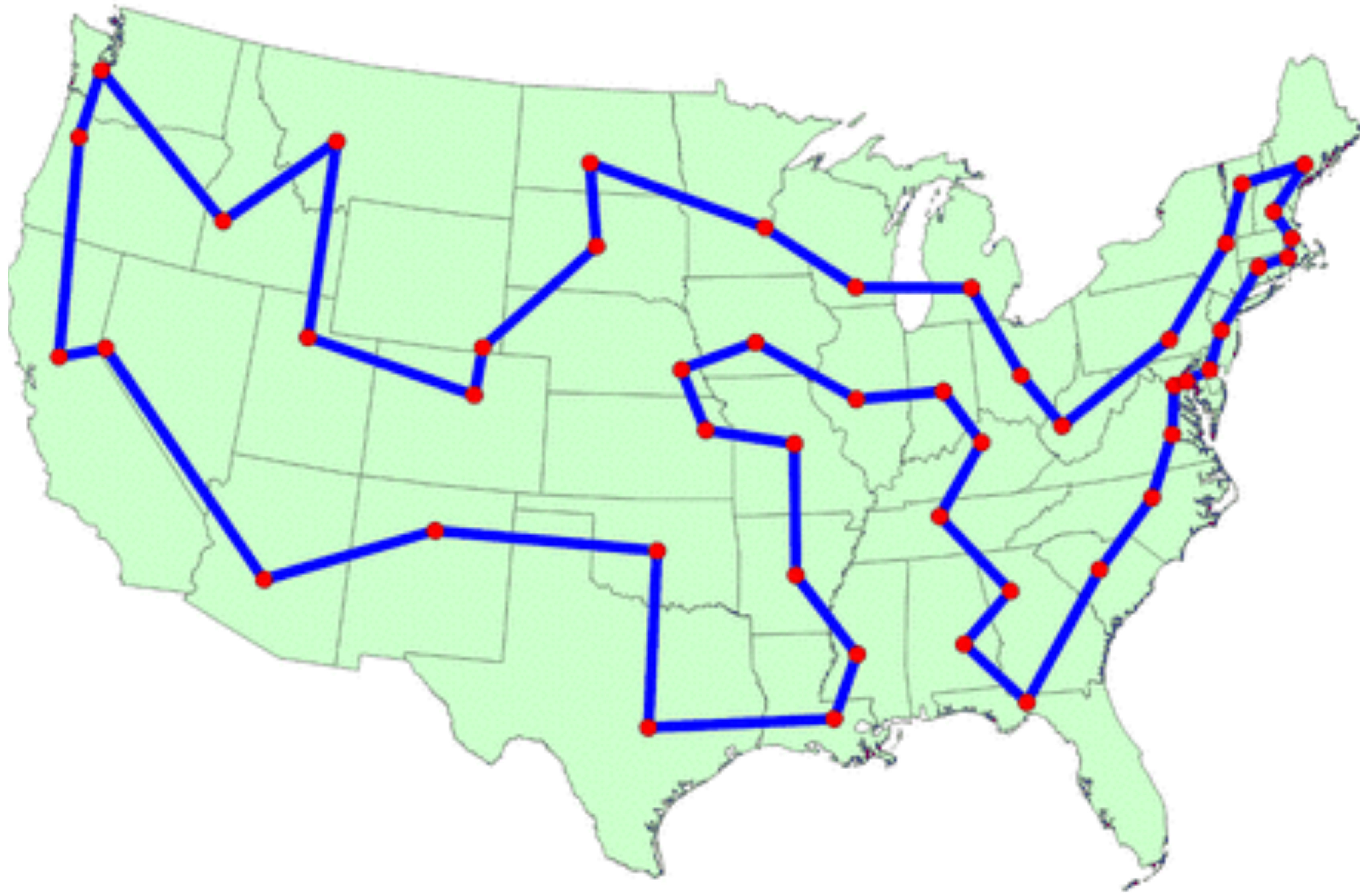
Example Algorithmic Approach

- Count all the pebbles.


ALGORITHM
1
2
3
4
5
...etc.

Heuristics

- Usually not *correct* (only gets you a "*good enough*" answer)
- Advantage: *fast* (often way faster than an algorithm)
- Famous example: the Traveling Salesman Problem



Algorithms

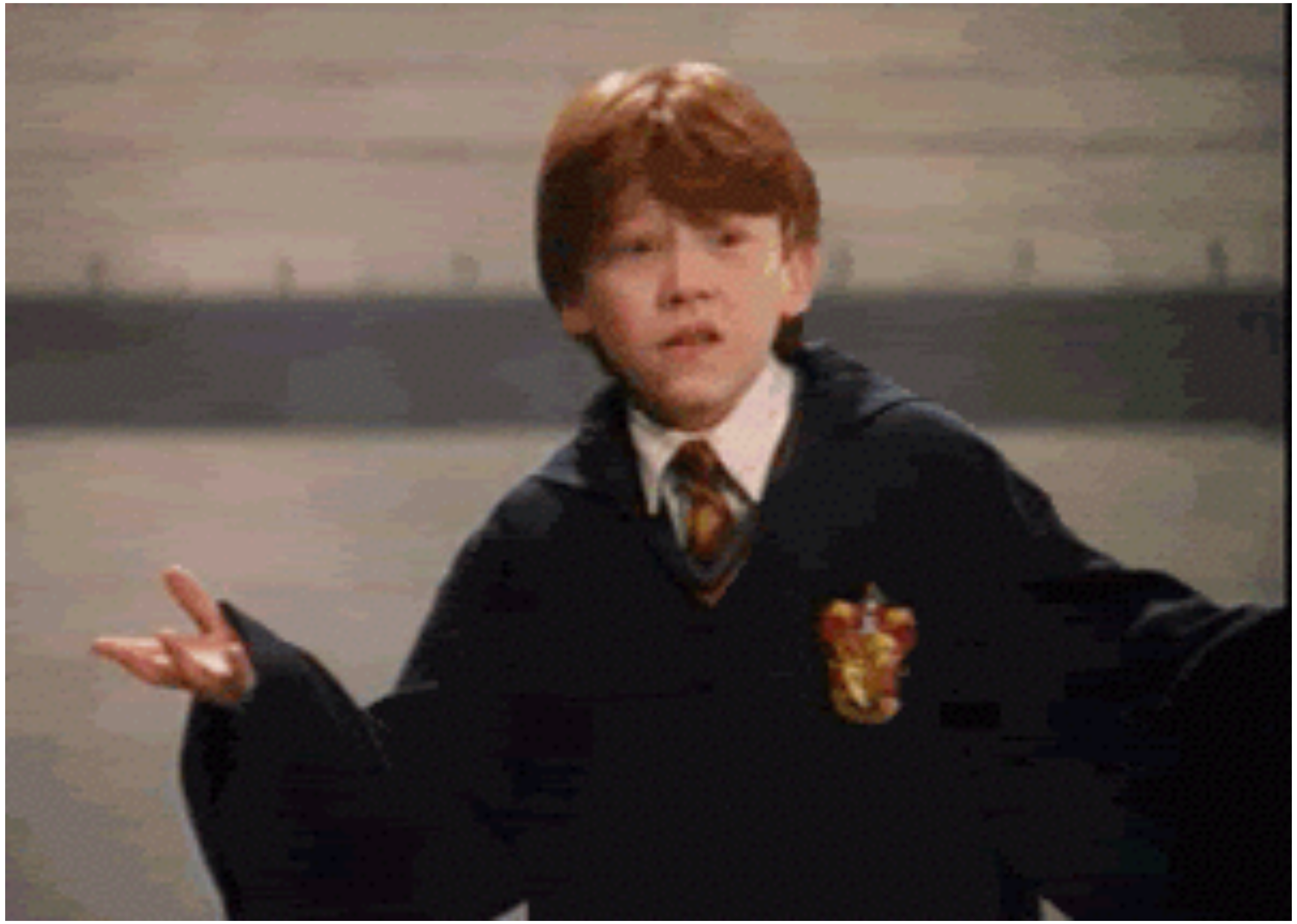
- Step-by-step instructions (deterministic)
- Complete (gets you an answer)
- Efficient (doesn't waste time getting you the correct answer)
- Correct (the answer isn't just close, it is true)
- Often we loosely call functions algorithms & vice-versa
- Downside: some problems are very hard / slow

How can we compare algorithms?


THE
BIG O

Algorithm Analysis: Big O Notation

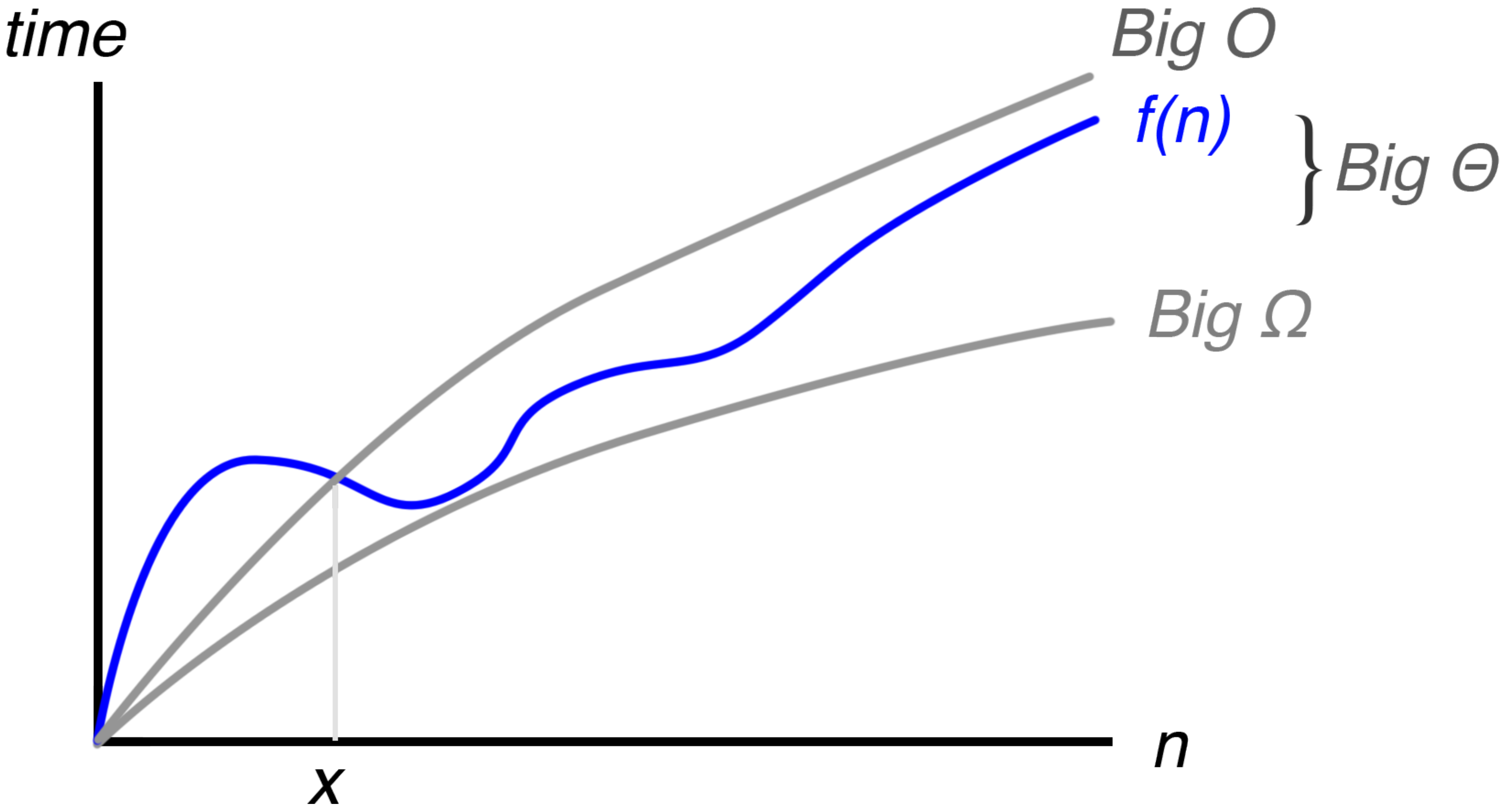
- A *comparative* way to classify different algorithms
- Based on *shape* of *time* vs *input size(s)*, e.g. "*n*", "*n* + *m*," "*nk*"
- For *big enough* inputs
  - "Who cares when *n* is small? Computers are fast."
- Establishing an *upper bound* on the time
  - "Not worse than this. Might be better, but it ain't worse!"
- Including just the *highest order* term
  - In $f(n) = n^3 + 5n + 3$, only n^3 matters
- *Ignores constants* (irrelevant! $0.005 \cdot n^2$ will overtake $50000 \cdot n$)



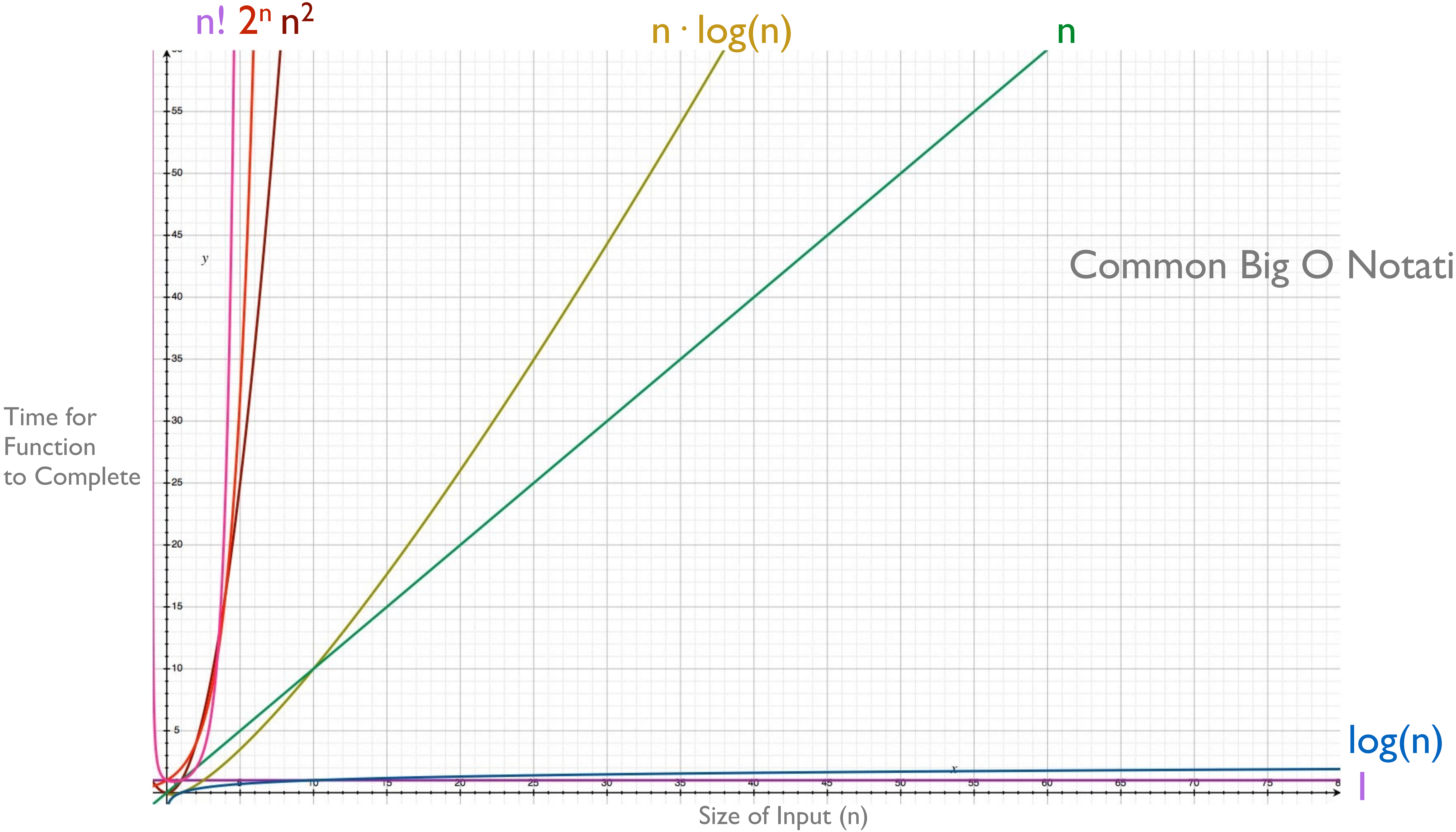
What?

Big O: comparative

- A very coarse, broad tool — big simplification
- Only useful when algorithms have *different* Big O notations
  - O(n) will always beat O(n^2), *for big enough n*
- If two algorithms have the same Big O, we don't know much. One might actually be quite slower than the other.


time
Big O
f(n)
Big Θ
Big Ω
n
x

Common Big O Notations



*Source: Skiena, The Algorithm Design Manual

Time Complexities (if 1 op = 1 ns)

input for f(n)	log n	n	n·log n	n^2	2^n	n!
10	0.003 µs	0.01 µs	0.033 µs	0.1 µs	1 µs	3.63 ms
20	0.004 µs	0.02 µs	0.086 µs	0.4 µs	1 ms	77.1 years
30	0.005 µs	0.03 µs	0.147 µs	0.9 µs	1 sec	8.4×10^{15} yrs
40	0.005 µs	0.04 µs	0.213 µs	1.6 µs	18.3 min	
50	0.006 µs	0.05 µs	0.282 µs	2.5 µs	13 days	
100	0.007 µs	0.1 µs	0.644 µs	10 µs	4×10^{13} yrs	
1 000	0.010 µs	1.00 µs	9.966 µs	1 ms		
10 000	0.013 µs	10 µs	130 µs	100 ms		
100 000	0.017 µs	0.10 ms	1.67 ms	10 sec		
1 000 000	0.020 µs	1 ms	19.93 ms	16.7 min		
10 000 000	0.023 µs	0.01 sec	0.23 sec	1.16 days		
100 000 000	0.027 µs	0.10 sec	2.66 sec	115.7 days		
1 000 000 000	0.030 µs	1 sec	29.90 sec	31.7 years		

Time Complexities

Big O	Name	Think	Example
$O(1)$	*Constant*	Doesn't depend on input	get array value by index
$O(\log n)$	*Logarithmic*	Using a tree	find min element of BST
$O(n)$	*Linear*	Checking (up to) all elements	search through linked list
$O(n \cdot \log n)$	*Loglinear*	A tree for each element	merge sort average & worst case
$O(n^2)$	*Quadratic*	Checking pairs of elements	bubble sort average & worst case
$O(2^n)$	*Exponential*	Generate all combinations	brute-force n-long binary number
$O(n!)$	*Factorial*	Generating all permutations	the Traveling Salesman

"By understanding sorting, we obtain an amazing amount of power to solve other problems."

– STEVEN SKIENA, THE ALGORITHM DESIGN MANUAL

Classic Sorting Algorithms

- Selection
- Insertion
- Bubble
- Merge
- Quick
- Heap
- Bogo?