Sorting algorithms

Things to consider

Theory *vs* Practice — Algorithms *vs* Implementations

Theoretical best-case performance on worst-case input: *n log n*

Is the algorithm **in-place**?

Does it use space efficiently?

Is the algorithm **adaptive**?

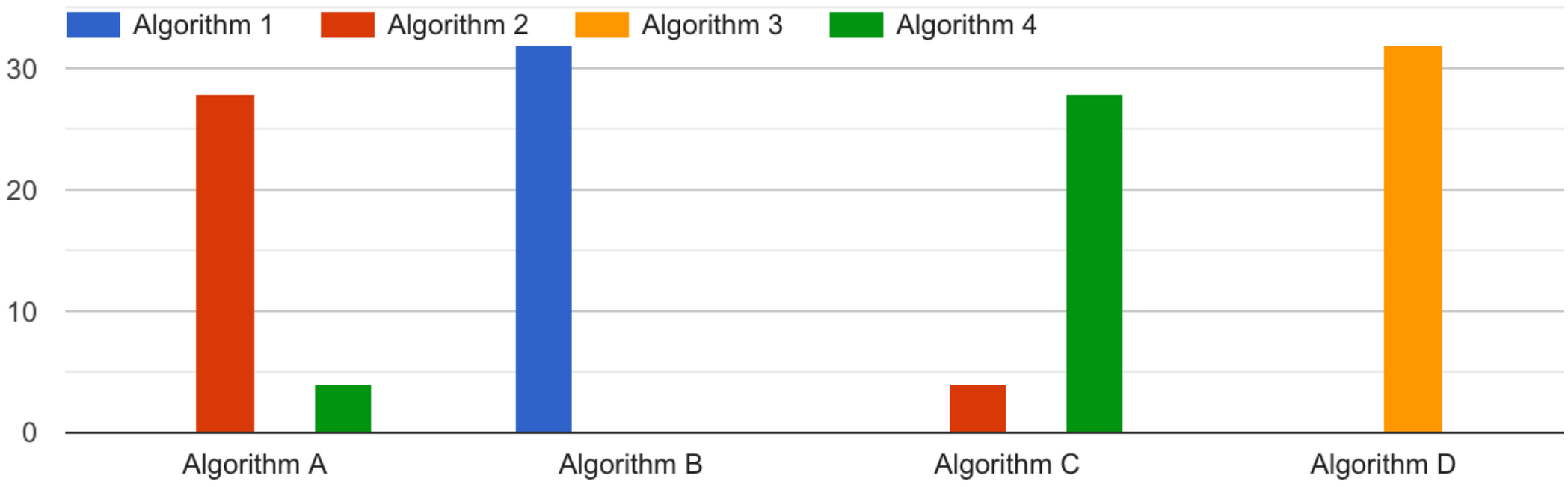
Does it perform well when the data is already sorted?

What are we measuring / modeling / optimizing for?

comparisons *vs* swaps • time *vs* space *vs* energy *vs* codability

Results

vote here: tinyurl.com/cs42sortdetective



alg.	input	math	closed form	asymptotic
A	sorted	$\sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} 1$	$\frac{N(N-1)}{2}$	$O(N^2)$
	antisorted			
B	sorted	$T(1) = 0$ $T(N) = N + 2T(\frac{N}{2})$	$N \log_2(N)$	$O(N \log N)$
	antisorted			
C	sorted	$\sum_{i=1}^{N} \sum_{j=0}^{N-2} 1$	$N(N-1)$	$O(N^2)$
	antisorted			
D	sorted	$\sum_{i=1}^{N-1} 1$	$N-1$	$O(N)$
	antisorted	$\sum_{i=1}^{N-1} \sum_{j=1}^{i} 1$	$\frac{N(N-1)}{2}$	$O(N^2)$

selection sort (2)

merge sort (1)

bubble sort (4)

insertion sort (3)

More fun with sorting

More ways to learn about sorting algorithms:

- On Wikipedia
- Using visualizations
- Using sonifications
- Using folk-dancification