Suppose we think of 4-5 algorithms & we've argued they always yield the correct answer.

How can we determine which will be most efficient, especially as n gets large?

Why not implement all of them & run them?

Problems

- Takes a lot of time to implement & test
- Time complexity often depends on input inself
- What data size should you use?

What does time complexity depend on?

- input itself
- input size (n = 100 vs n = 1,000,000)
- hardware (computer)
- compiler

⋮

machine dependent

Focus on a machine-independent analysis.

Asymptotic Time Complexity

machine independent, rough
measure of time complexity
as a function of input size (n)

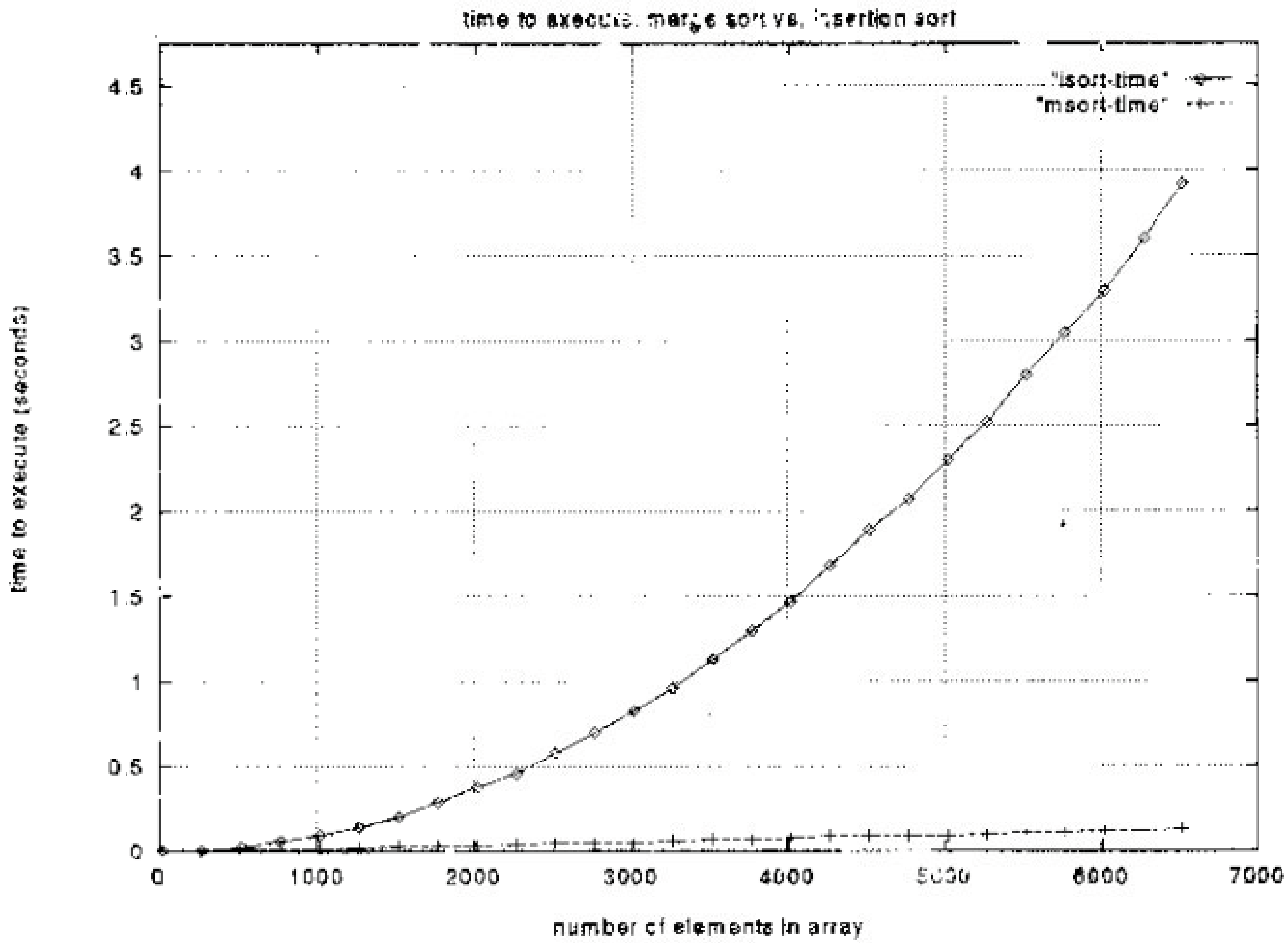
"Back of the envelope" calculation

What are we going to measure?

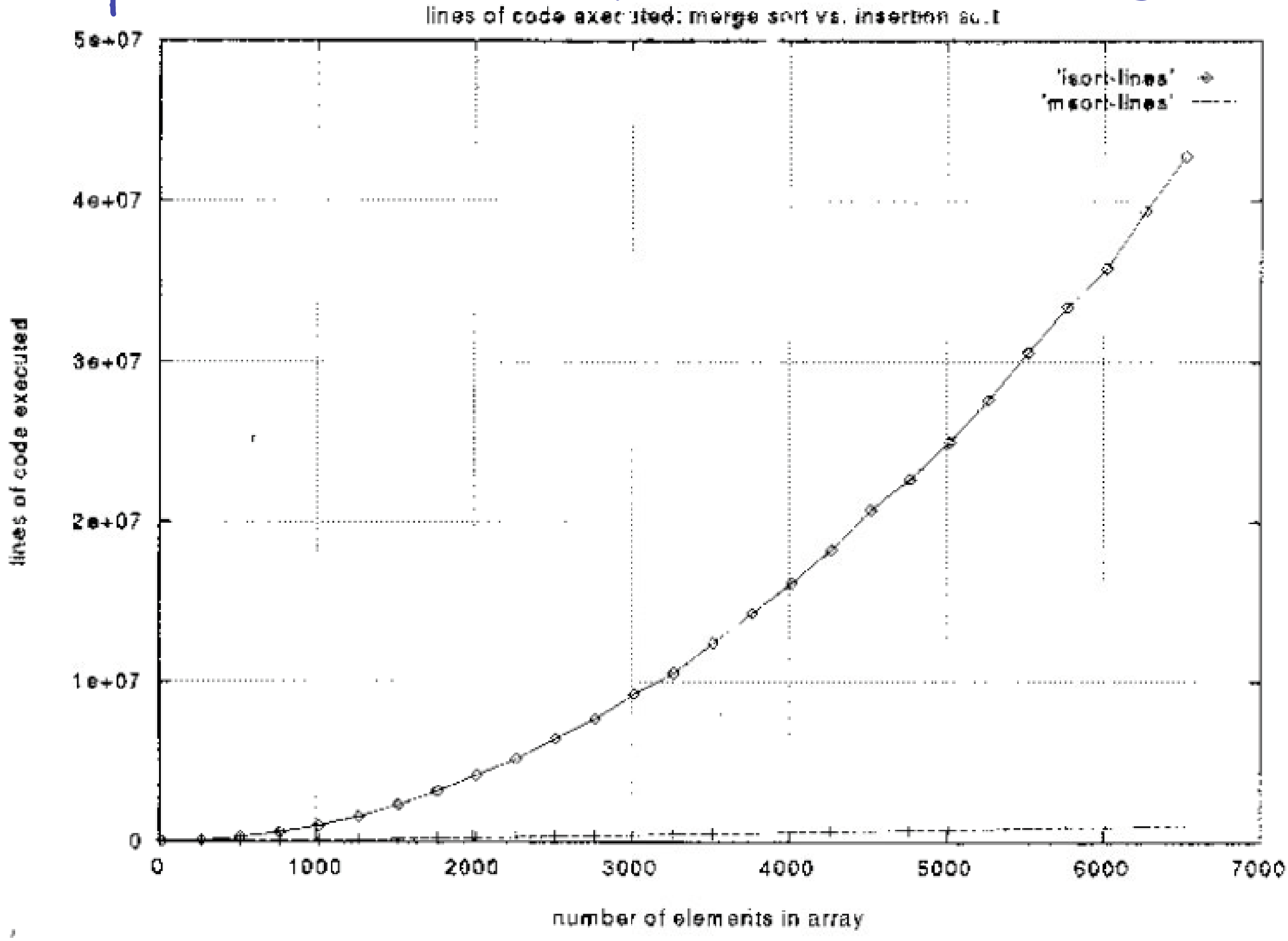
# of statements (lines of code) executed

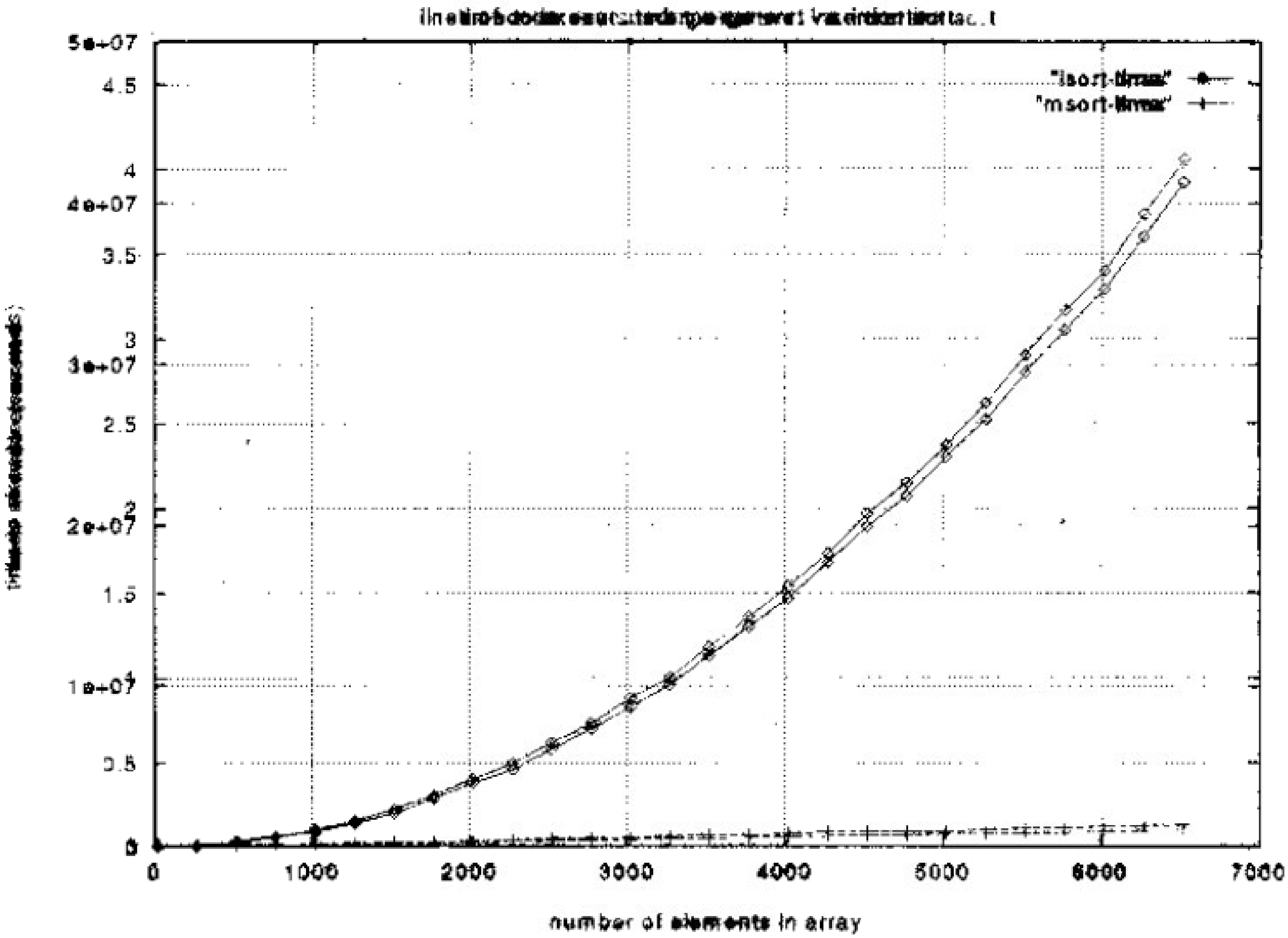
need to be careful that all statements in your high-level language take "roughly" the same time.

Input size vs Execution time



Input Size vs lines of Code




"isort-times"
"msort-times"
5e+07
4.5
4
4e+07
3.5
3
3e+07
2.5
2
2e+07
1.5
1e+07
0.5
0
1000
2000
3000
4000
5000
6000
7000
number of elements in array

How do we account for dependencies in the input?

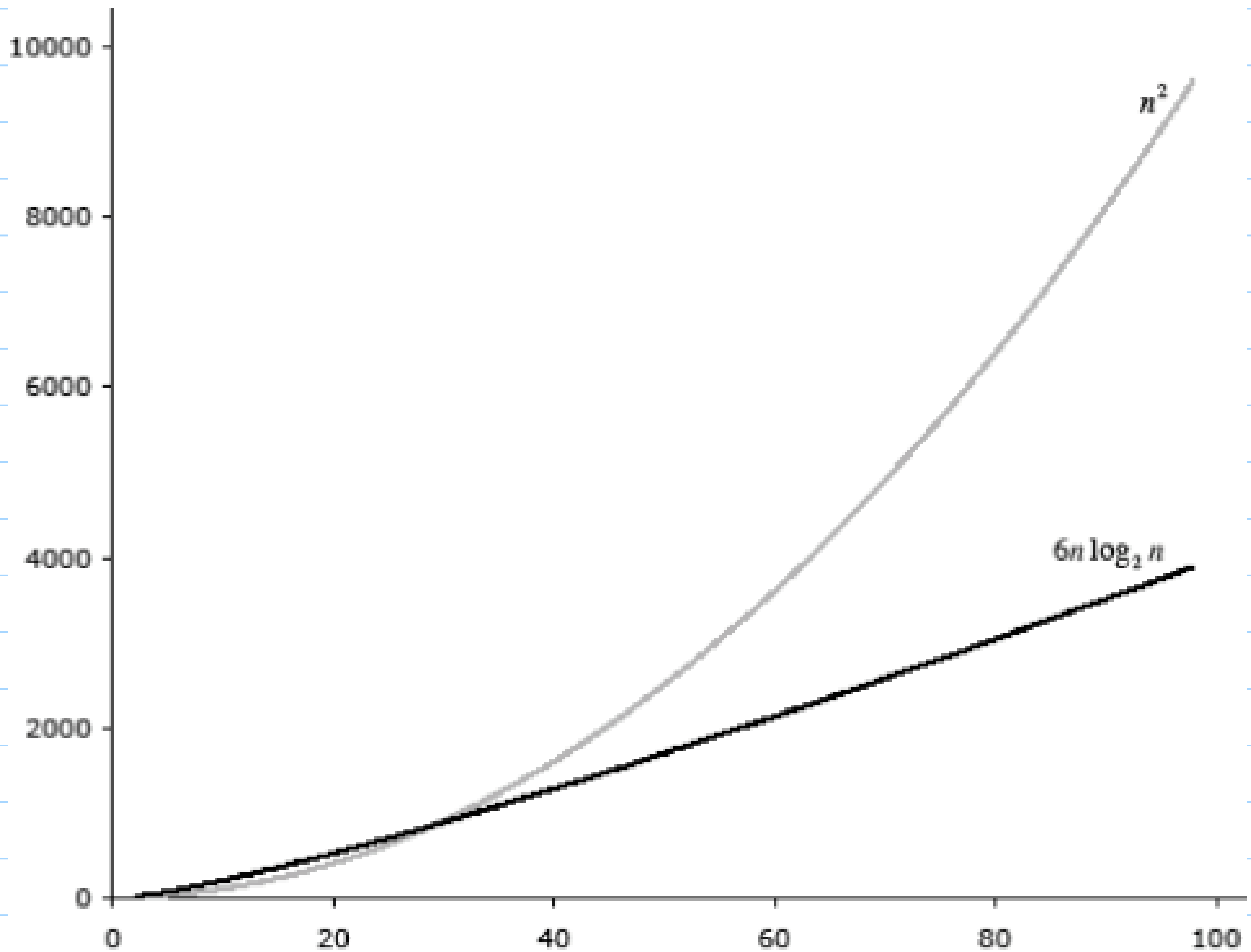
Worst-case analysis — consider the input of the given size that is slowest

Expected-case (average-case) analysis

— assume some distribution over the data

n^2 vs $n \log n$

Note Title 8/30/2007



Asymptotic Growth Rates

statements

c	constant
$c \cdot \log n$	logarithmic
$c \cdot n$	linear
$n \cdot \log n$	
$c \cdot n^2$	quadratic
$c \cdot n^3$	cubic

we'll give alg for closest pairs w/ this complexity