Analysing Costs (in general)

How can we determine the costs of a program?

- **Time:**
  - Run the **program** and count the milliseconds/minutes/days.
  - Count number of steps/operations the **algorithm** will take.
- **Space:**
  - Measure the amount of memory the **program** occupies.
  - Count the number of elementary data items the **algorithm** stores.

- Applies to Programs or Algorithms? *Both.*
  - programs → "benchmarking"
  - algorithms → "analysis"

What is a good algorithm?

Obviously needs to do what is expected consistently. However most problems can be solved in many ways. What is most important?

- Clarity - easy to read/implement
- Efficiency - the cost of running it

Clarity is relatively simple to measure. Find somebody else to read you code.

But how do we measure efficiency of an algorithm?

Benchmarking: program cost

Measure:

- actual programs, on real machines, with specific input
- measure elapsed time
  - System.currentTimeMillis ()
    → time from the system clock in milliseconds
- measure real memory usage

Problems:

- what input? ⇒ use large data sets
  don't include user input
- other users/processes? ⇒ minimise
  average over many runs
- which computer? ⇒ specify details

- how to compare cross-platform? ⇒ measure cost at an abstract level

Analysis: Algorithm "complexity"

- Abstract away from the details of
  - the hardware, the operating system
  - the programming language, the compiler
  - the specific input
- Measure number of "steps" as a function of the data size
  - best case (easy, but not interesting)
  - worst case (usually easy)
  - average case (harder)
- The precise number of steps is not required
  - $3.47\ n^2 - 67n + 53$ steps
  - $3n \log(n) + 5n - 3$ steps
- Rather, we are interested in how the cost grows with data size on large data

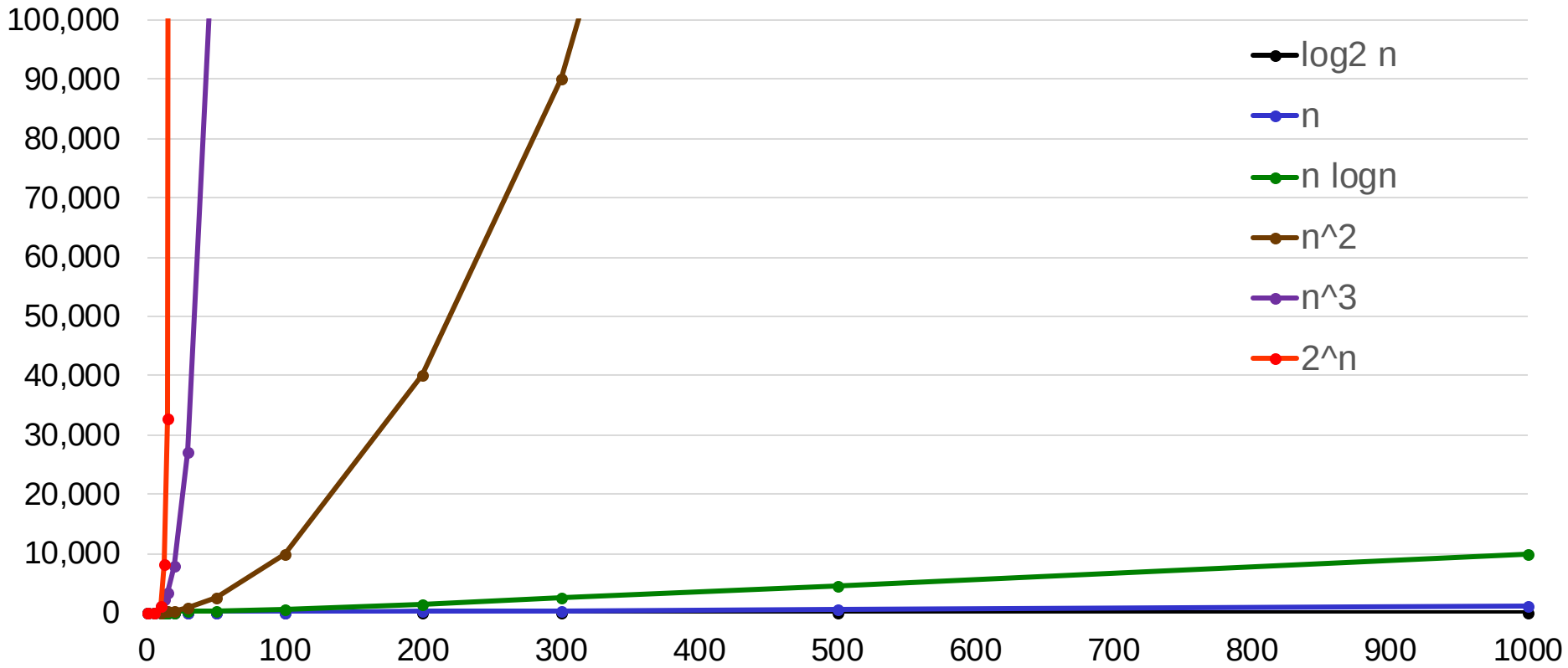
Big-O Notation

- "Asymptotic cost", or "big-O" cost describes how cost grows with large input size
- Only care about large input sets
  - Lower-order terms become insignificant for large n
- We care about how cost grows with input size
  - Don't care about constant factors
  - Multiplication factors (3, 102, 3 and 12 below) don't tell us how things "scale up"

$3.47\ n^2 + 102n + 10064$ steps → $O(n^2)$

$3n \log n + 12n$ steps → $O(n \log n)$

How the different costs grow



n: size of input