
PERFORMANCE
1 60in 2 3 4 5 6 7 8

How to measure performance?

- Exact measurements are very difficult
  - What else is running on the machine?
  - Will you get different performance on different hardware?
- Estimate using number of Java instructions executed
  - Different implementations of the same algorithm are different.
- Estimate using “Order of” O(n) calculations
  - Gives a rough idea of how many instructions
  - Enables comparison of algorithms

Big O Concept

- Model the performance of your algorithm using parameters that control the performance
  - If you are sorting a list, how big is that list?
  - If you are analyzing a picture – how many pixels are there?

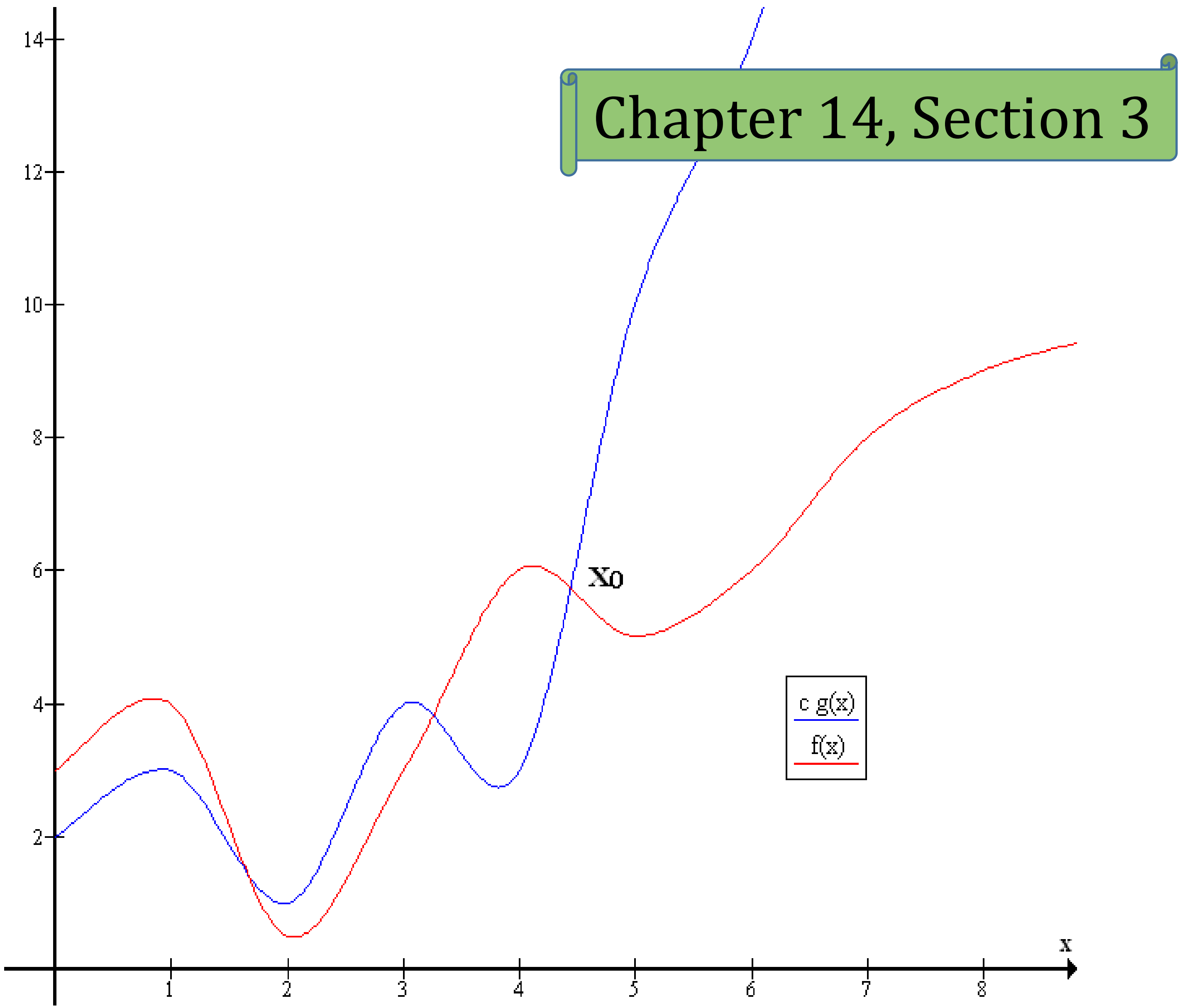
- Make the simplest possible model
  - Accuracy is not important
  - Looking for trends
  - Trying to describe the basic idea without worrying about too many details

Formal Definition of Big Oh

$$f(x) = O\big(g(x)\big)$$

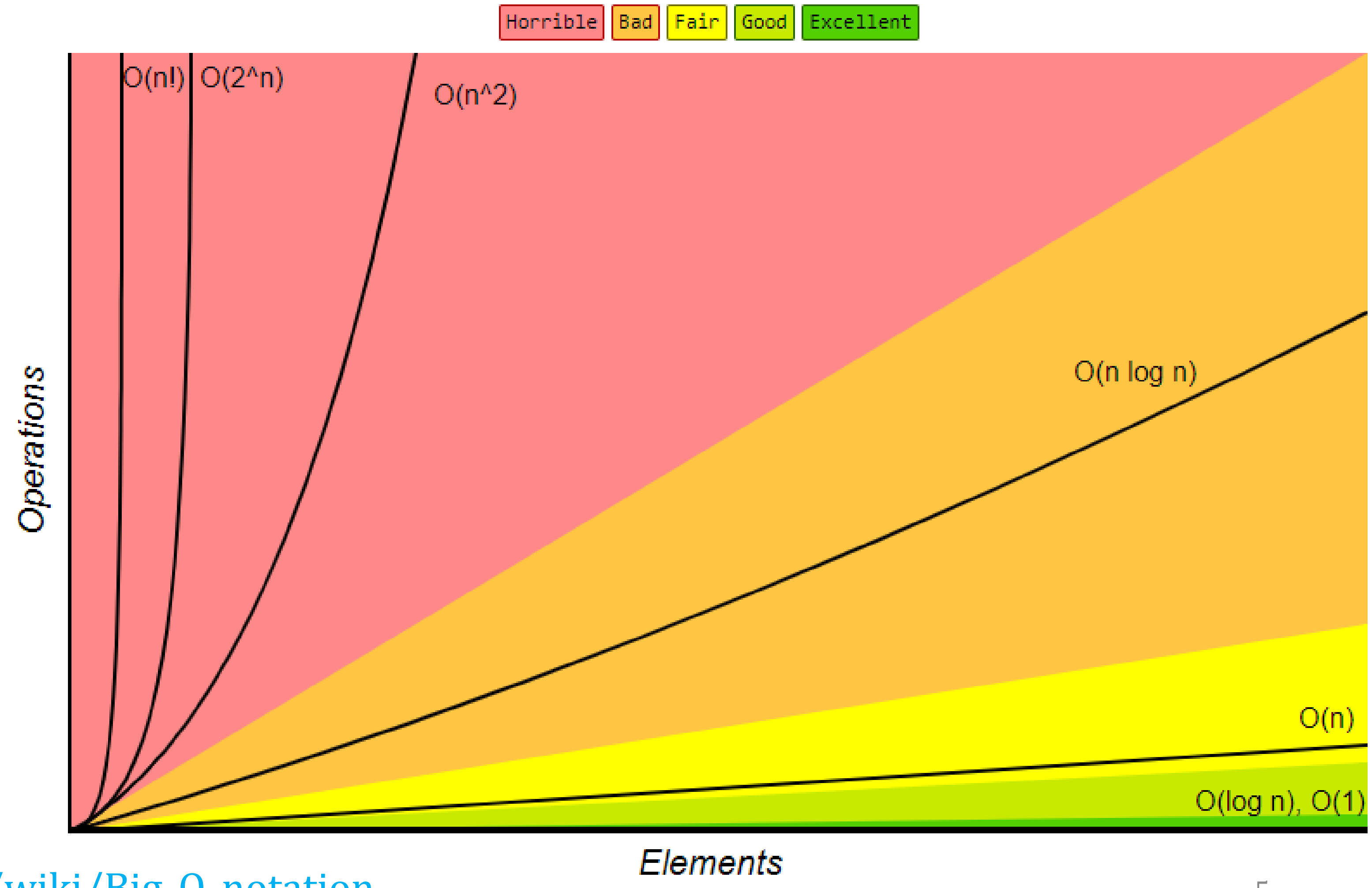
If and only if there exists some positive constant c, and starting point x_0 such that

$$f(x) \leq c \times g(x)$$
$$\forall\, x > x_o$$



How fast does it grow?

Order	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n \log n)$	Log Linear
$O(n^c)$	Polynomial
$O(c^n)$	Exponential
$O(n!)$	Factorial



https://en.wikipedia.org/wiki/Big_O_notation

Practical examples of Big Oh

Order	Name	Example
O(1)	Constant	Is a number even or odd?
O(log n)	Logarithmic	Binary search of n items
O(n)	Linear	Finding an item in an unsorted list of n items
O(n log n)	Log Linear	Sorting a list of n items
O(n^c)	Polynomial	Managing graphs with n nodes
O(c^n)	Exponential	Travelling salesman with n cities
O(n!)	Factorial	Brute Force solution of travelling salesman

https://en.wikipedia.org/wiki/Big_O_notation

Some Examples of Big-O notation

- Game of Life arrayUpdate: O(n) (n=number of cells)
  - arrayUpdate has many instructions in the update loop (constant)
  - arrayUpdate needs two loops through all the cells (constant)
- HashMap insertion O(n/m)
  - Where n is the number of elements in the hash map, and m is the number of bins.
  - Assuming n=c*m, this is O(c*m/m) = O(c) = O(1) or constant time.
- Naïve Sort : put least element first, move forward
  - To find the least element: O(n)
  - Number of times to find the least element O(n)
  - Total algorithm: O(n^2) or polynomial time

Performance and Scalability

- Algorithms with constant, linear, or n log(n) performance can be applied to very large data with little or no performance penalty. These are "scalable"
- Algorithms with polynomial, exponential, or factorial performance can ONLY be applied to very small problems! These are not scalable
- Academics often ignore scalability when innovating. Industry cannot!