Informatics 1

Introduction to Computation

Lecture 10

Rates of Growth

Philip Wadler

University of Edinburgh

Associativity and Efficiency: Left vs. Right

Consider m lists, $xs_1, \ldots, xs_m$, each of length n.

Associated to the left, `foldl (++) []`

$$(((\mathrm{[]} +\!\!+ xs_1) +\!\!+ xs_2) +\!\!+ xs_3) \cdots +\!\!+ xs_m$$

computing takes

$$\underbrace{0 + n + 2n + 3n + ... + (m-1)n}_{m \text{ times}}$$

steps. If we have m lists of length n, it takes $O(m^2 n)$ steps.

Associated to the right, `foldr (++) []`

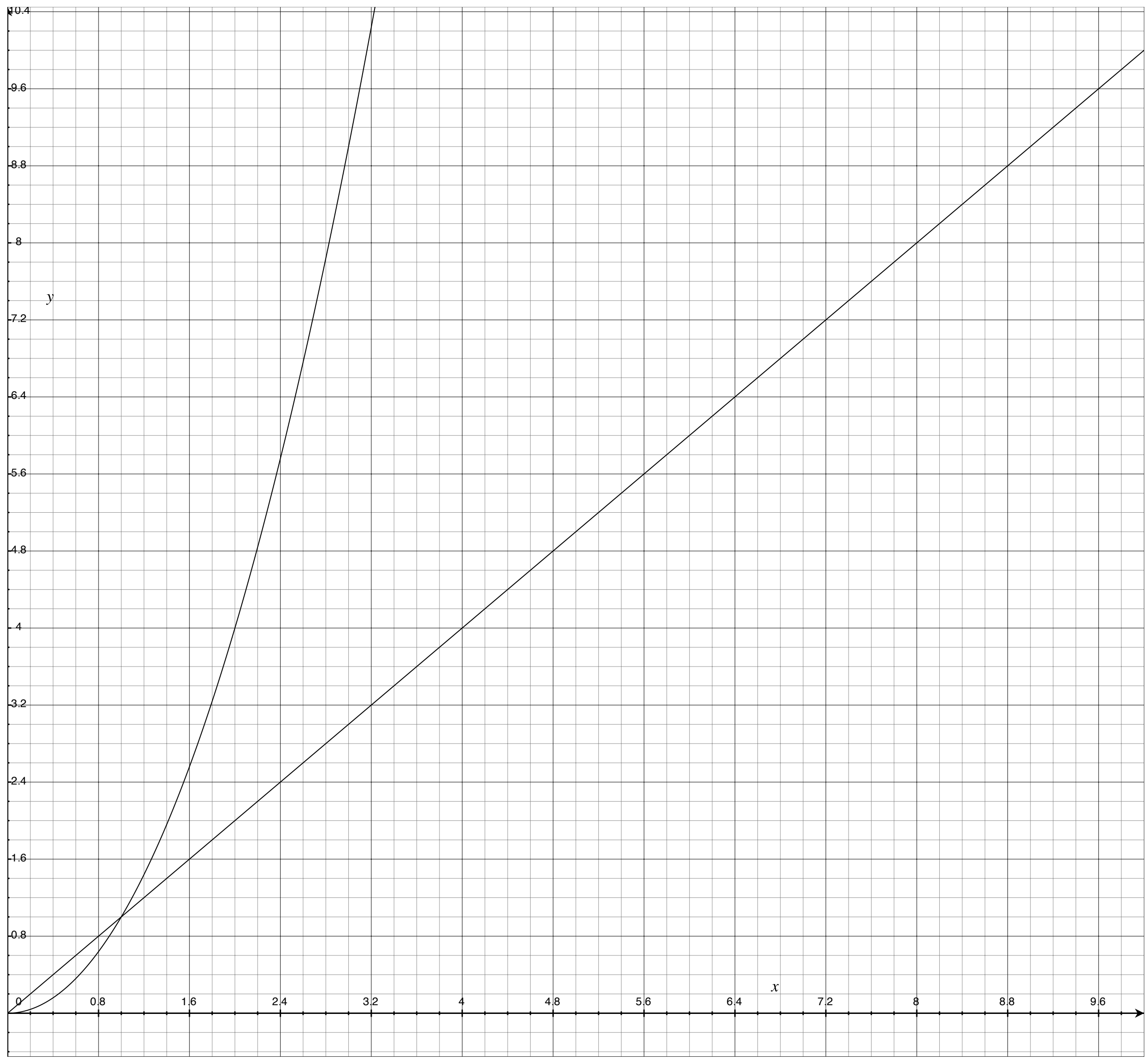
$$xs_1 +\!\!+ \cdots (xs_{m-2} +\!\!+ (xs_{m-1} +\!\!+ (xs_m +\!\!+ \mathrm{[]})))$$

computing takes

$$\underbrace{n + n + n + \cdots + n}_{m \text{ times}}$$

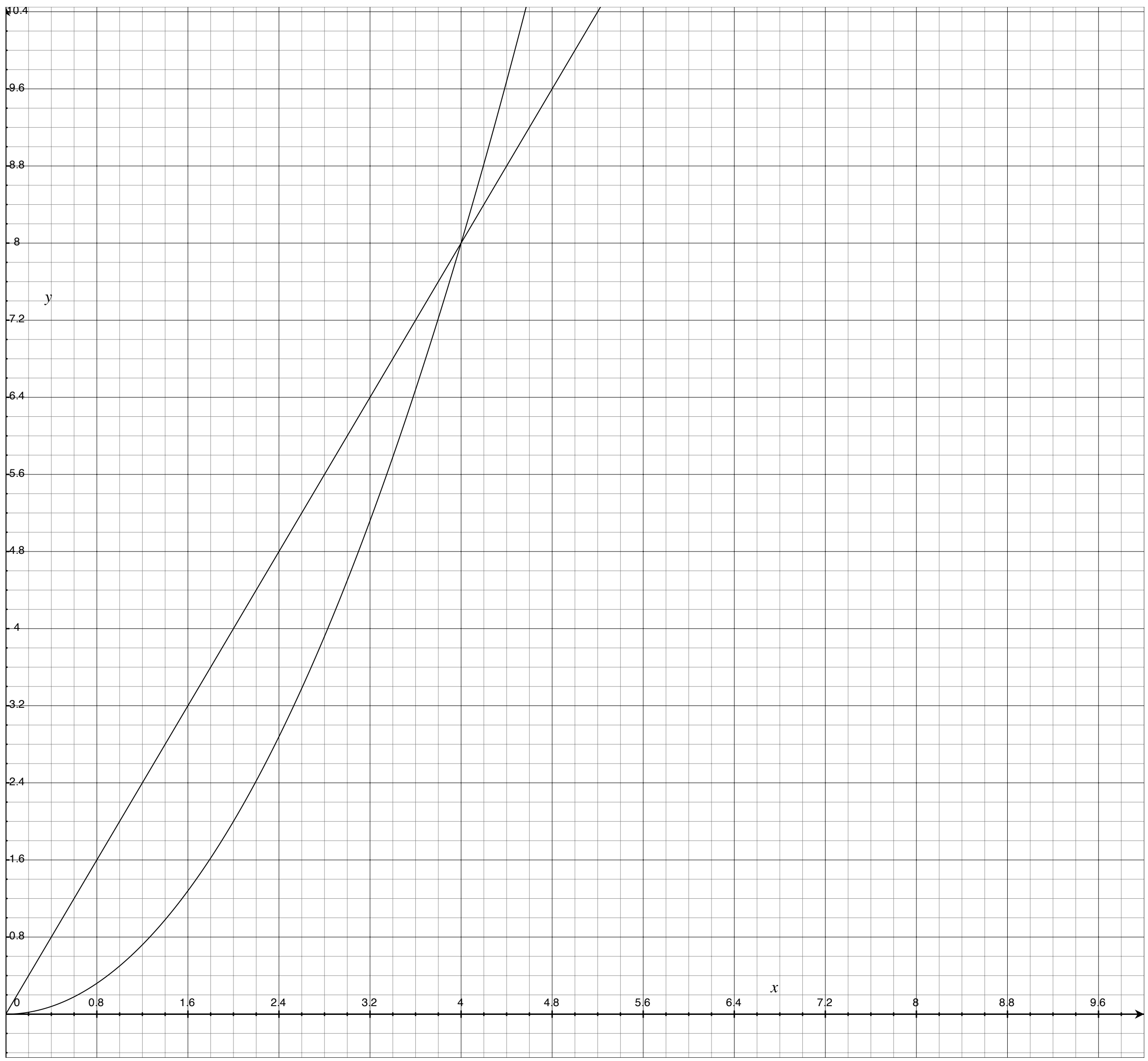
steps. If we have m lists of length n, it takes $O(mn)$ steps. When $m = 1000$, the first is a thousand times slower than the second!

$t = n$ vs $t = n^2$



10.4
9.6
8.8
8
7.2
6.4
5.6
4.8
4
3.2
2.4
1.6
0.8
0
y
x
0.8
1.6
2.4
3.2
4
4.8
5.6
6.4
7.2
8
8.8
9.6

$$t = 2n \text{ vs } t = 0.5n^2$$



Big-O notation

Definition *We say f is $O(g)$ when g is an upper bound for f, for big enough inputs. To be precise, f is $O(g)$ if there are constants c and m such that $f(n) \leq cg(n)$ for all $n \geq m$.*

For instance: $2n + 10$ is $O(n)$ because $2n + 10 \leq 4n$ for all $n \geq 5$.

Big-O notation

Definition *We say f is $O(g)$ when g is an upper bound for f, for big enough inputs. To be precise, f is $O(g)$ if there are constants c and m such that $f(n) \leq cg(n)$ for all $n \geq m$.*

For instance: $2n + 10$ is $O(n)$ because $2n + 10 \leq 4n$ for all $n \geq 5$.

Constant factors don't matter

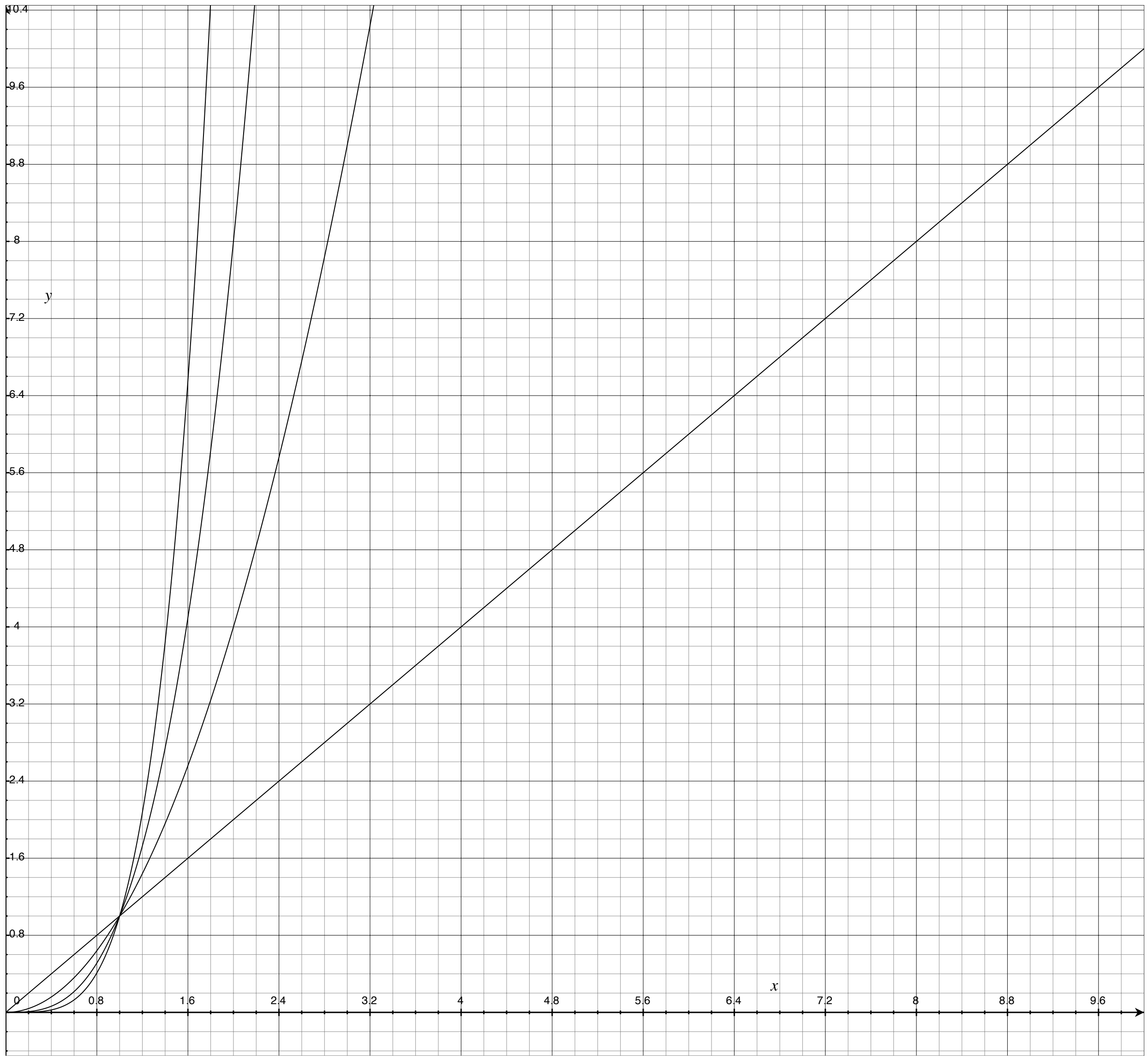
$O(n) = O(an + b)$, for any a and b

$O(n^2) = O(an^2 + bn + c)$, for any a, b, and c

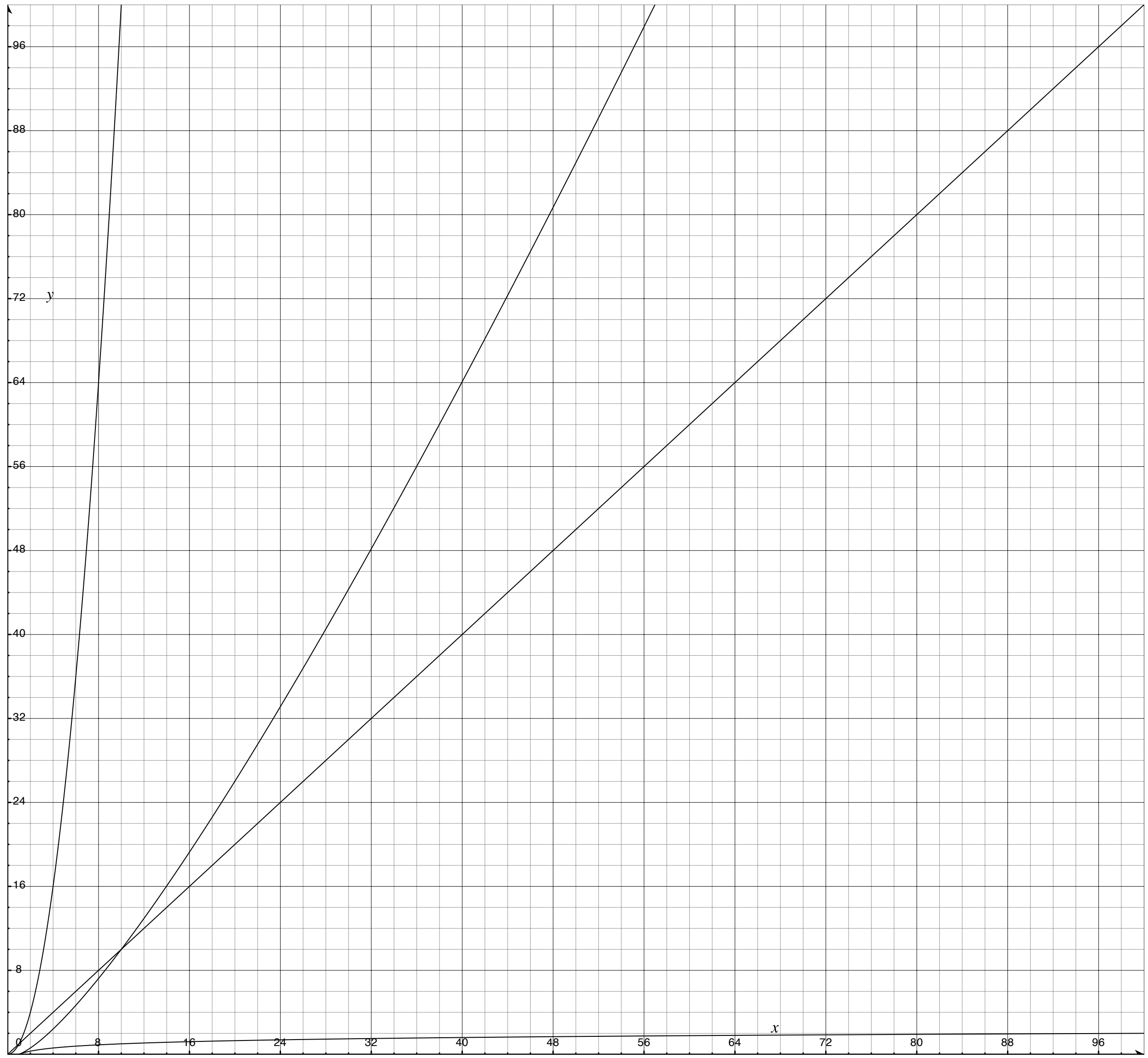
$O(n^3) = O(an^3 + bn^2 + cn + d)$, for any a, b, c, and d

$O(log_2(n)) = O(log_{10}(n))$

$$O(n), O(n^2), O(n^3), O(n^4)$$



$$O(\log n), O(n), O(n \log n), O(2^n)$$



Associativity and Efficiency: Sequential vs. Parallel

Sequential:

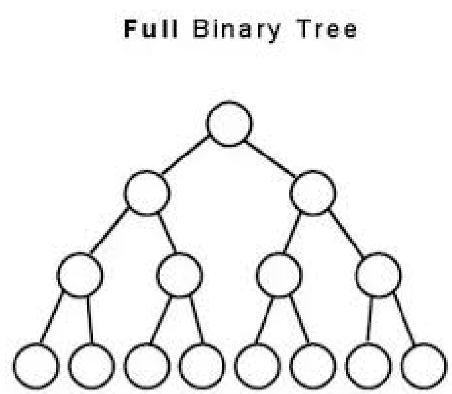
$$((((((x_1 + x_2) + x_3) + x_4) + x_5) + x_6) + x_7) + x_8$$

Summing 8 numbers takes 7 steps. If we have m numbers it takes $O(m)$ steps.

Parallel:

$$((x_1 + x_2) + (x_3 + x_4)) + ((x_5 + x_6) + (x_7 + x_8))$$

Summing 8 numbers takes 3 steps.



If we have m numbers it takes $O(\log(m))$ steps. When $m = 1000$, the first is a hundred times slower than the second!

$O(\log n), O(n \log n), O(2^n)$

$O(\log n)$ "logarithmic": parallel sum, divide and conquer search algorithms

$O(n)$ "linear": ordinary sum

$O(n \log n)$: sorting algorithms

$O(2^n)$ "exponential": tautology checking