Foundations of Computer Science

ECE NTUA

Asymptotic Notation

Slides: Stathis Zachos, Aris Pagourtzis

Algorithm efficiency

- We measure the algorithm cost as a function of the computational resources required, relative to the size of the input in the worst case:

$\text{cost}_A(n)$ = max {cost of algorithm A for input x}

among all

inputs x

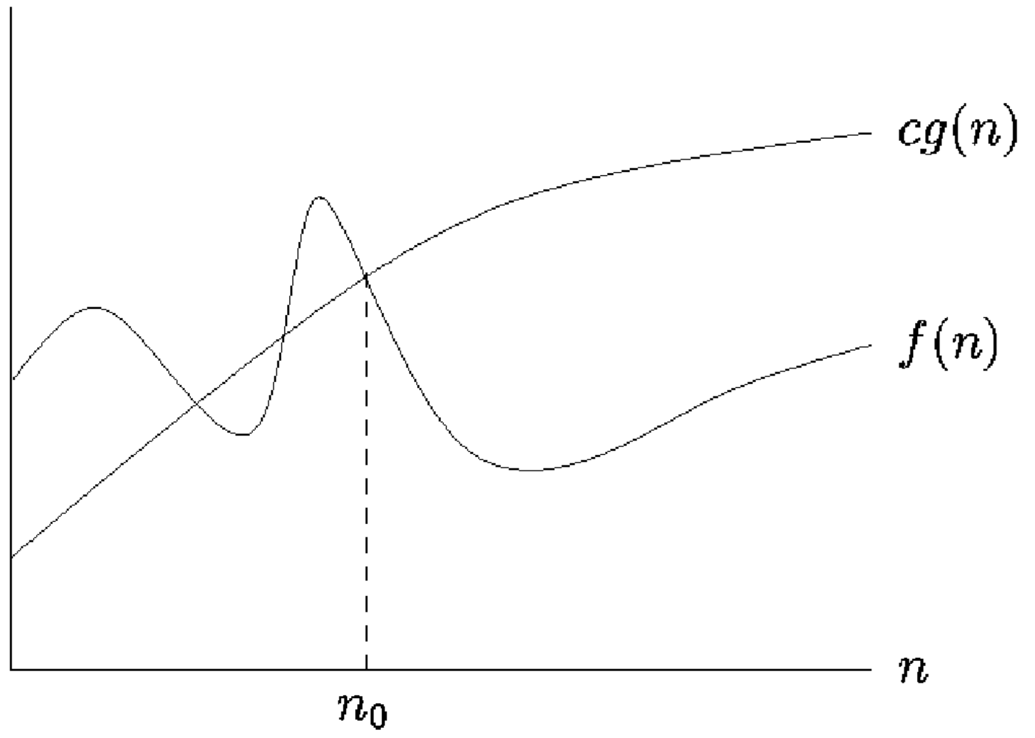
of size n

- <u>Example</u>: $\text{time-cost}_{MS}(n) <= c\ n\log n$
(MS = MergeSort, *c* a constant)

Algorithm efficiency

- We are usually interested in the time cost, or time complexity.
- Also of interest is the space cost, or space complexity.
- Example: $\text{space-cost}_{MS}(n) \leq c' n$
  (MS = MergeSort, *c'* a constant)

Asymptotic Notation (i)



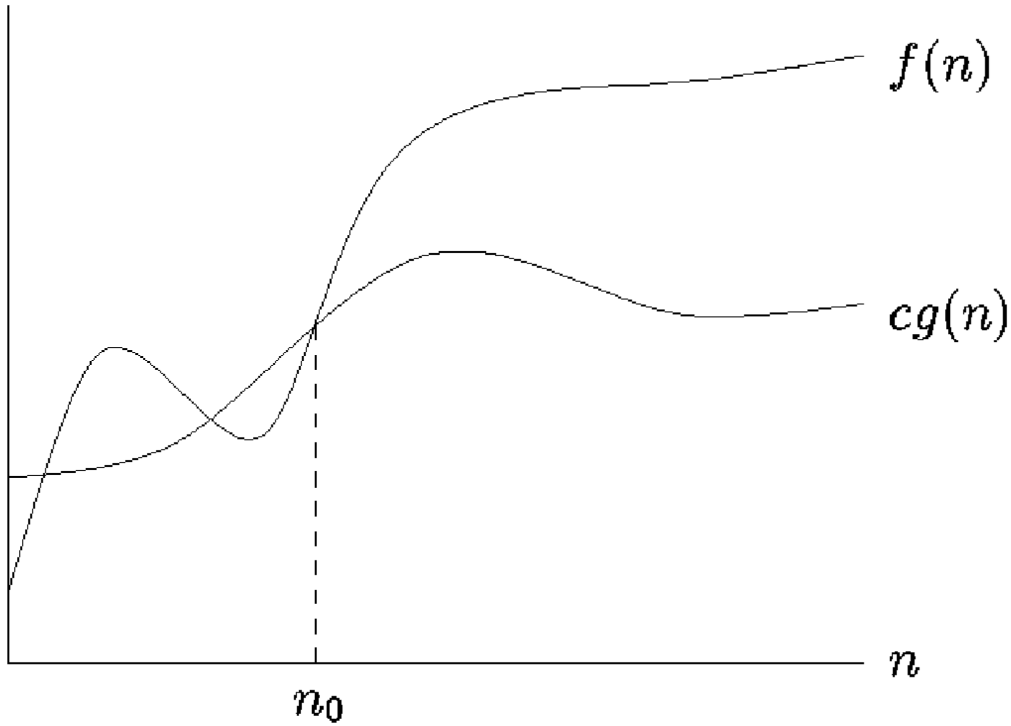
$$f = O(g)$$

$$O(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \ \ f(n) \leq cg(n)\}$$

O Notation : examples

- BubbleSort: $T_{BS}(n) = O(n^2)$
- InsertionSort: $T_{IS}(n) = O(n^2)$
- MergeSort: $T_{MS}(n) = O(n \log n)$
- Warning: worst-case complexity: the worst-case cost for MergeSort is at most $cn\log n$

Asymptotic Notation (ii)



$$f = \Omega(g)$$

$$\Omega(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0\ \ f(n) \geq cg(n)\}$$

Ω Notation : examples

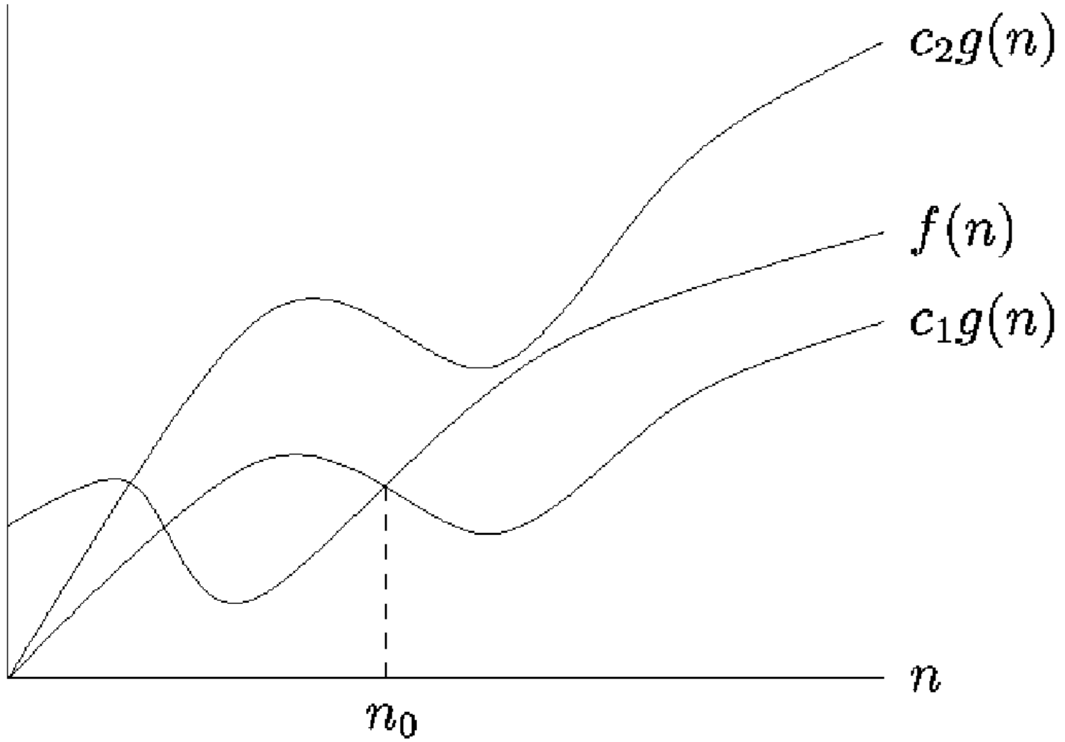
- BubbleSort: $T_{BS}(n) = \Omega(n^2)$

- InsertionSort: $T_{IS}(n) = \Omega(n^2)$

- MergeSort: $T_{MS}(n) = \Omega(n \log n)$

- Warning: worst-case complexity: the worst-case cost for MergeSort is at most $cn\log n$

Asymptotic Notation (iii)



$$f = \Theta(g)$$

$$\Theta(g) = \{f \mid \exists c_1 > 0,\ \exists c_2 > 0,\ \exists n_0 : \forall n > n_0\ \ c_1 \leq \frac{f(n)}{g(n)} \leq c_2\}$$

Θ Notation : examples

- BubbleSort: $T_{BS}(n) = \Theta(n^2)$

- InsertionSort: $T_{IS}(n) = \Theta(n^2)$

- MergeSort: $T_{MS}(n) = \Theta(n \log n)$

- Warning: worst-case complexity: the worst-case cost for MergeSort is at most $cn\log n$ and at least $c'n\log n$

Asymptotic notation: conventions and properties

- We write: $g(n) = O(f(n))$ instead of $g(n) \in O(f(n))$
- $\Theta(f) = O(f) \cap \Omega(f)$
- $p(n) = \Theta(n^k)$, for all polynomial p
- $O(poly) = \bigcup O(n^k)$ (for all $k \in N$)

Asymptotic notation: conventions and properties

$$O(1) < O(\alpha(n)) < O(\log^* n)$$

$$< O(\log(n)) < O(\sqrt{n}) < O(n)$$

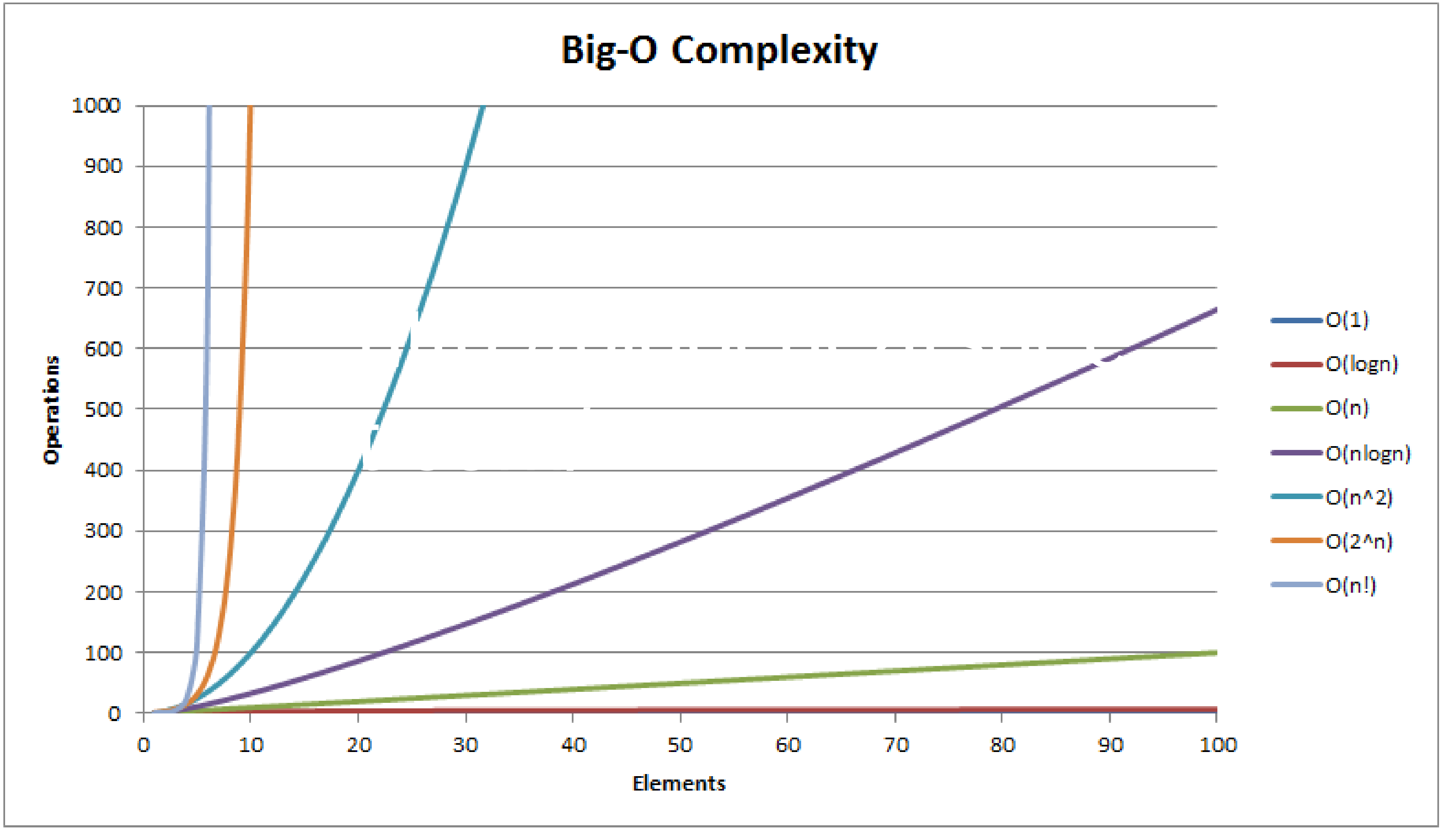
$$< O(n \log(n)) < O(n^2) < \ldots < O(poly)$$

$$< O(2^n) < O(n!) < O(n^n) < O(A(n))$$

log**n*: how many times we have to logarithmize n to get below 1 (inverse of hyperexponential)

A: Ackermann.

Why?



Source: bigocheatsheet.com/

Asymptotic notation: bounds proof

Theorem. $\log(n!) = \Theta(n \log n)$

Proof: asymptotically (for $n > n_0$) it holds that:

$(n/2)^{n/2} < n! < n^n$ =>

$(1/2)\, n\, (\log n - 1) < \log(n!) < n \log n$ =>

$(1/4)\, n \log n < \log(n!) < n \log n$

Algorithm complexity: conventions

- We often consider as **input size** the number of input elements only (ignoring their size in bits):
- A satisfactory estimate if the input numbers are «small» in relation to the rest of the input
- Or if they are «large» their value does not affect the number of elementary operations: e.g. sorting with comparisons (BubbleSort, MergeSort, InsertionSort), finding shortest paths (Dijkstra, Bellman-Ford), finding MST (Prim, Kruskal).

Algorithm complexity: conventions

- We also assume that each elementary arithmetic operation (addition, multiplication, comparison) has unit cost (1 step):

  that is called **arithmetic complexity** and is usually a good estimate (see also word RAM model)

- An estimation of **bit complexity** is necessary when numbers «grow» a lot during the execution: e.g. raise to power, *n*-th Fibonacci

Problem complexity

- Is the complexity of the optimal algorithm that solves the problem

$$\mathrm{cost}_{\Pi}(n) = \min \{\mathrm{cost}_{A}(n)\}$$

among all the algorithms
A that solve Π

- Example: $\text{time-cost}_{\mathrm{SORT}}(n) = O(n \log n)$
(SORT = sorting problem)

- To prove *algorithm optimality* we need *proof* of the *corresponding lower bound*: $\Omega(n \log n)$

Time complexity analysis of algorithms

Counting steps to be executed:

- either by direct summation of the number of the steps (iterative algorithms)
  - e.g.: $T_{BS}(n) <= c\ n^2 = O(n^2)$
    (BS = BubbleSort, *c* some constant)
- or by solving recursive relations (recursive algorithms)
  - e.g.: $T_{MS}(n) <= 2T_{MS}(n/2)+cn = \ldots = O(n \log n)$
    (MS = MergeSort, *c* some constant)

Time complexity

O(1)	a := b*c;	simple operations
O(log$n$)	if x<A[n/2] search(A[1,n/2])...	binary search
O(n)	for i:=1 to n do <O(1)>	simple loop
O(n log$n$)	mergesort(A[1, n/2]) mergesort(A[n/2+1, n]) merge(A[1, n/2], A[n/2+1, n])	sorting with merge
O(n^2)	for i:=1 to n do for j:=1 to n do <O(1)>	double loop
O(2^n)	for all S $\subseteq$ $\{0,1\}^n$ do <O(1)>	subsets
O(n!)	for all σ in S[n] do <O(1)>	permutations

Divide & Conquer algorithms

$O(\log n)$	if x<A[n/2] search(A[1,*n*/2])…	binary search
$O(\max(\mathrm{len}(a),\mathrm{len}(b))^3)$	$GCD(a,b) := GCD(b, a \bmod b)$	find GCD
$O(\mathrm{len}(n))$ *	$pow(a,n) := pow(a^2, n/2)$	raise to power
$O(\mathrm{len}(n))$ *	matrix fast doubling	*n*-th Fibonacci
$O(n^{\log 3})$	Gauss-Karatsuba algorithm	multiply *n*-digit numbers
$O(n^{\log 7})$	Strassen algorithm	multiply *n* x *n* *matrices*

* arithmetic complexity, len(x) = #digits of $x = \Theta(\log x)$