Θεμελιώδη Θέματα Επιστήμης Υπολογιστών

ΣΗΜΜΥ – ΣΕΜΦΕ ΕΜΠ

Ασυμπτωτικός συμβολισμός

Επιμέλεια διαφανειών: Στάθης Ζάχος, Άρης Παγουρτζής

Αποδοτικότητα αλγορίθμου

- Μετράμε το κόστος αλγορίθμου σαν συνάρτηση των υπολογιστικών πόρων που απαιτούνται σε σχέση με το μέγεθος της (αναπαράστασης της) εισόδου στην *χειρότερη περίπτωση*:

  $cost_A(n)$ = max {κόστος αλγορίθμου A για είσοδο x}

  μεταξύ όλων
  των εισόδων
  x μήκους *n*

- Παράδειγμα: $time\text{-}cost_{MS}(n) <= c\ n \log n$
  (MS = MergeSort, *c* μία σταθερά)

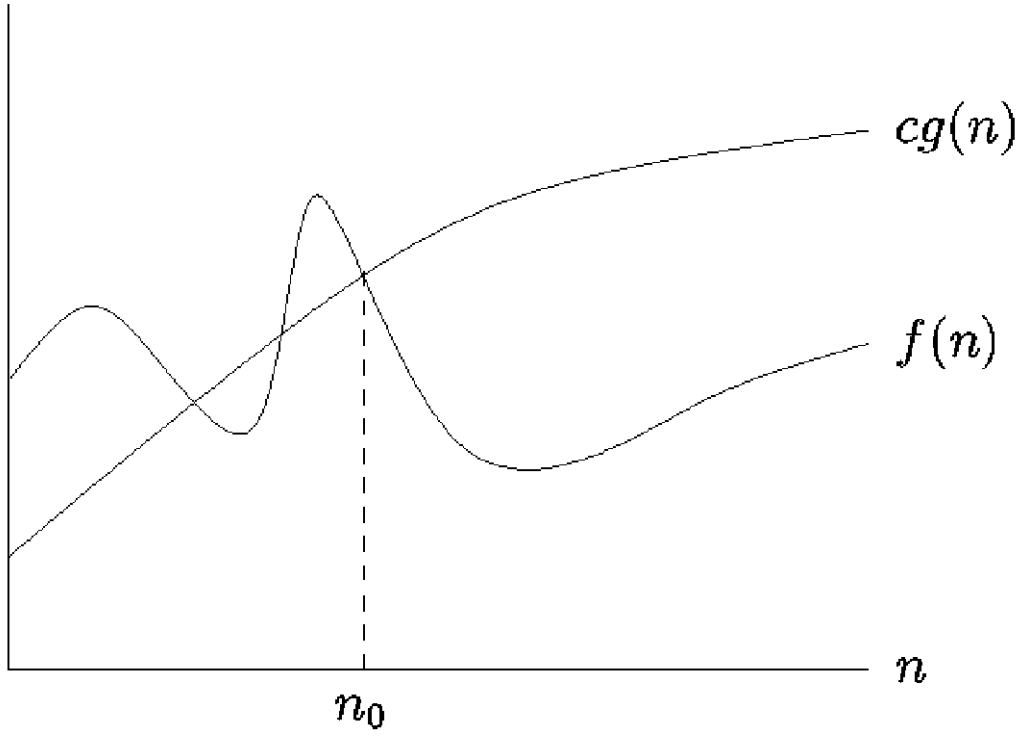
Αποδοτικότητα αλγορίθμου

- Συνήθως μας ενδιαφέρει το κόστος σε χρόνο, ή αλλιώς η χρονική πολυπλοκότητα.

- Ενδιαφέρον παρουσιάζει και το κόστος σε χώρο, ή αλλιώς χωρική πολυπλοκότητα.

- Παράδειγμα: $\text{space-cost}_{MS}(n) <= c' n$
  (MS = MergeSort, *c'* κάποια σταθερά)

Ασυμπτωτικός συμβολισμός (i)



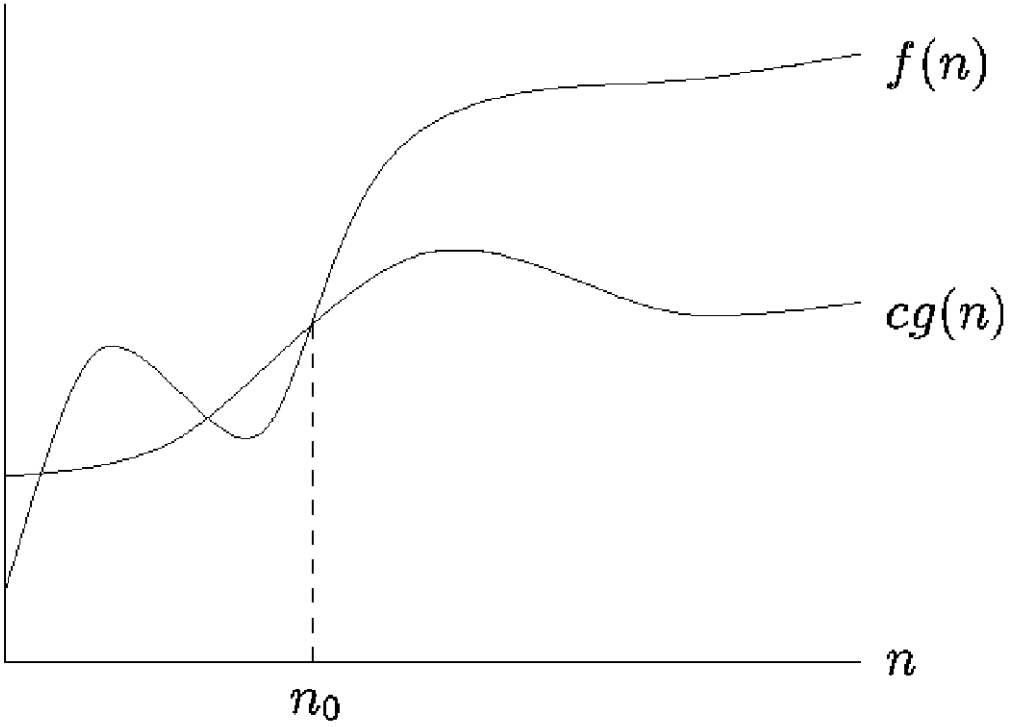
$$f = O(g)$$

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

Συμβολισμός O : παραδείγματα

- BubbleSort: $T_{BS}(n) = O(n^2)$
- InsertionSort: $T_{IS}(n) = O(n^2)$
- MergeSort: $T_{MS}(n) = O(n \log n)$
- Προσοχή: πολυπλοκότητα χειρότερης περίπτωσης: το κόστος της χειρότερης περίπτωσης για την MergeSort είναι το πολύ $cn\log n$

Ασυμπτωτικός συμβολισμός (ii)



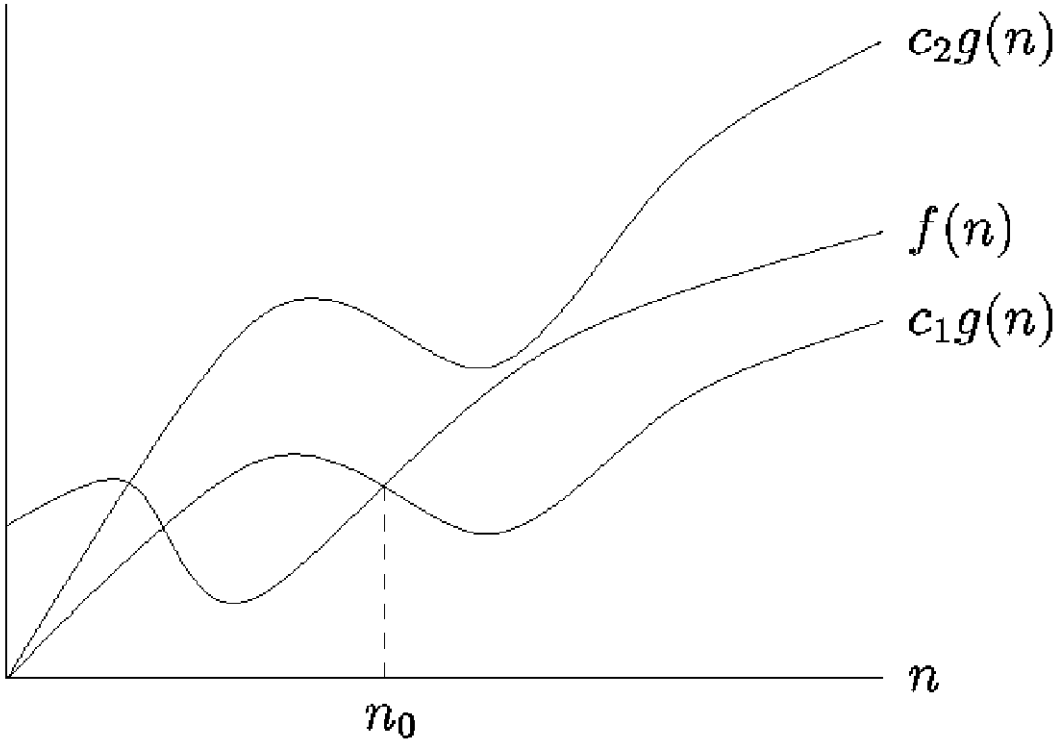
$$f = \Omega(g)$$

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

Συμβολισμός Ω : παραδείγματα

- BubbleSort: $T_{BS}(n) = \Omega(n^2)$
- InsertionSort: $T_{IS}(n) = \Omega(n^2)$
- MergeSort: $T_{MS}(n) = \Omega(n \log n)$
- Προσοχή: πολυπλοκότητα χειρότερης περίπτωσης: το κόστος της χειρότερης περίπτωσης για την MergeSort είναι τουλάχιστον $cn\log n$

Ασυμπτωτικός συμβολισμός (iii)



$$f = \Theta(g)$$

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

Συμβολισμός Θ : παραδείγματα

- BubbleSort: $T_{BS}(n) = \Theta(n^2)$
- InsertionSort: $T_{IS}(n) = \Theta(n^2)$
- MergeSort: $T_{MS}(n) = \Theta(n \log n)$
- Προσοχή: πολυπλοκότητα χειρότερης περίπτωσης: το κόστος της χειρότερης περίπτωσης για την MergeSort είναι το πολύ $cn\log n$ και τουλάχιστον $c'n\log n$

Ασυμπτωτικός συμβολισμός : συμβάσεις και ιδιότητες

- Γράφουμε: $g(n) = O(f(n))$ αντί για $g(n) \in O(f(n))$
- $\Theta(f) = O(f) \cap \Omega(f)$
- $p(n) = \Theta(n^k)$, για κάθε πολυώνυμο p
- $O(poly) = \bigcup O(n^k)$ (για όλα τα $k \in \mathbb{N}$)

Ασυμπτωτικός συμβολισμός : συμβάσεις και ιδιότητες

$$O(1) < O(\alpha(n)) < O(\log^* n)$$
$$< O(\log(n)) < O(\sqrt{n}) < O(n)$$
$$< O(n\log(n)) < O(n^2) < \ldots < O(poly)$$
$$< O(2^n) < O(n!) < O(n^n) < O(A(n))$$

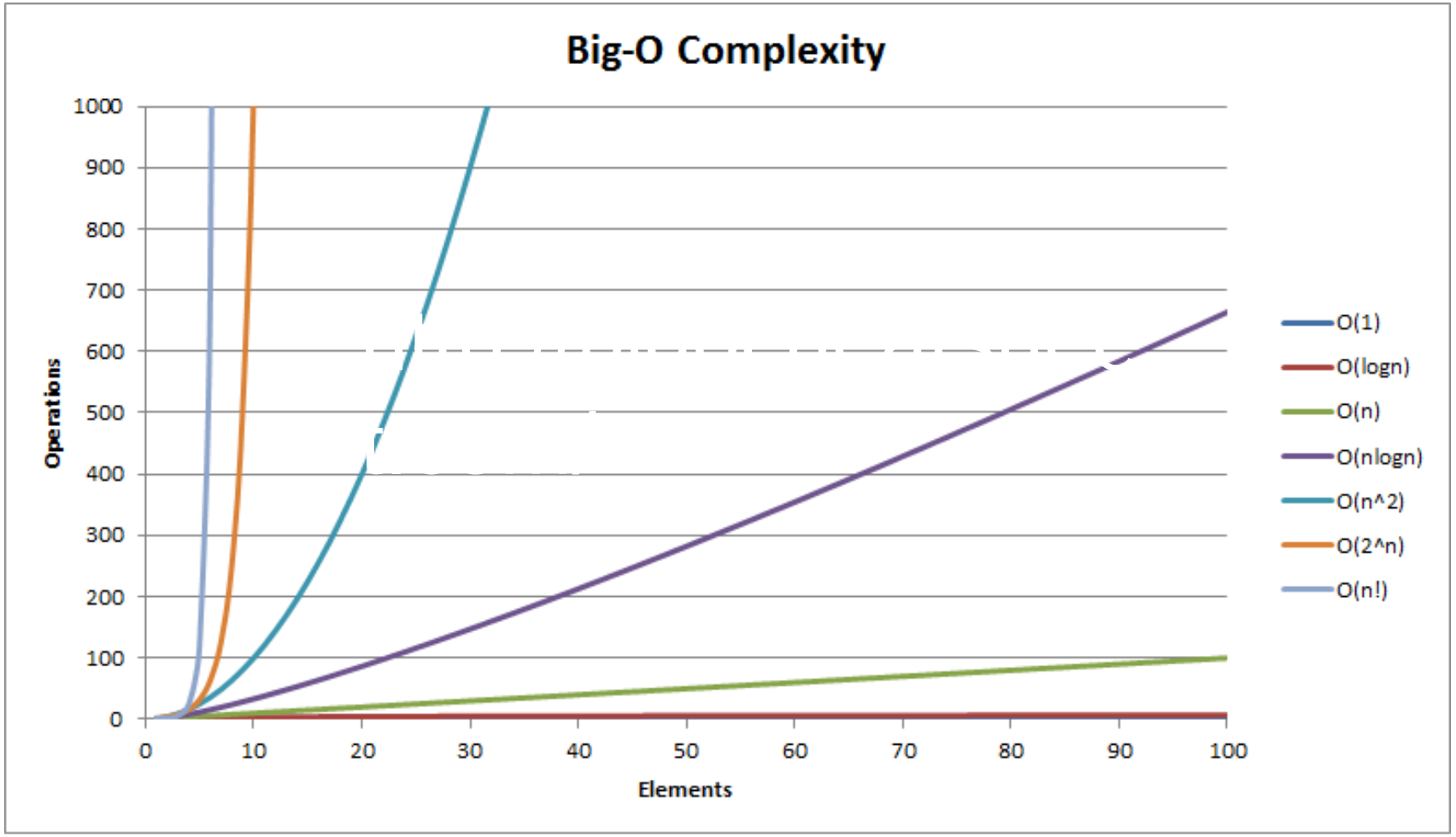
Σημείωση: γράφουμε "$<$" αντί "$\subset$".

log*n: πόσες φορές πρέπει να λογαριθμήσουμε το n για να φτάσουμε κάτω από το 1 (αντίστροφη υπερεκθετικής)

A: Ackermann.

α: αντίστροφη της A.

Γιατί ασυμπτωτικός συμβολισμός;



Source: bigocheatsheet.com/

Ασυμπτωτικός συμβολισμός: απόδειξη φραγμάτων

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

Απόδειξη: ασυμπτωτικά (για $n > n_0$) ισχύει:

$(n/2)^{n/2} < n! < n^n \Rightarrow$

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

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

Προσοχή: μπορείτε να χρησιμοποιήσετε κανόνα de l'Hospital, αλλά συνήθως *γίνεται και πιο απλά, όπως παραπάνω*!

Πολυπλοκότητα αλγορίθμων: απλοποιήσεις

- Συχνά θεωρούμε ως **μέγεθος της εισόδου** το πλήθος των στοιχείων εισόδου μόνο (αγνοώντας το μέγεθός τους σε bits):
- ικανοποιητική εκτίμηση, αν οι τυχόν αριθμοί εισόδου είναι «μικροί» σε σχέση με την υπόλοιπη είσοδο
- ή αν είναι «μεγάλοι» αλλά η τιμή τους δεν επηρεάζει το πλήθος των στοιχειωδών πράξεων: π.χ. ταξινόμηση με συγκρίσεις (BubbleSort, MergeSort, InsertionSort), εύρεση συντομότερων διαδρομών (Dijkstra, Bellman-Ford), εύρεση MST (Prim, Kruskal).

Πολυπλοκότητα αλγορίθμων: απλοποιήσεις

- Θεωρούμε ακόμη ότι κάθε στοιχειώδης αριθμητική πράξη (πρόσθεση, πολ/σμός, σύγκριση) έχει μοναδιαίο κόστος (1 βήμα):

  αυτό λέγεται **αριθμητική πολυπλοκότητα** (arithmetic complexity) και είναι συνήθως ικανοποιητική εκτίμηση (δείτε και word RAM model)

- η εκτίμηση της **πολυπλοκότητας ψηφιοπράξεων** (bit complexity) είναι απαραίτητη όταν οι αριθμοί «μεγαλώνουν» πολύ κατά τη διάρκεια εκτέλεσης: π.χ. ύψωση σε δύναμη, *n*-οστός Fibonacci

Πολυπλοκότητα προβλήματος

- Είναι η πολυπλοκότητα του βέλτιστου αλγορίθμου που λύνει το πρόβλημα.

  $cost_{\Pi}(n) = \min \{cost_{A}(n)\}$

  μεταξύ όλων των αλγορίθμων
  Α που επιλύουν το Π

- Παράδειγμα: $time\text{-}cost_{SORT}(n) = O(n \log n)$
  (SORT = πρόβλημα ταξινόμησης)

- Για να δείξουμε *βελτιστότητα αλγορίθμου* χρειάζεται και *απόδειξη αντίστοιχου κάτω φράγματος*: $\Omega(n \log n)$

Ανάλυση χρονικής πολυπλοκότητας αλγορίθμων

Μέτρηση βημάτων που θα εκτελεστούν:

- είτε με απευθείας άθροιση πλήθους βημάτων (επαναληπτικοί αλγόριθμοι)
  - Π.χ.: $T_{BS}(n) <= c\ n^2 = O(n^2)$
    (BS = BubbleSort, *c* κάποια σταθερά)
- είτε με επίλυση αναδρομικών σχέσεων (αναδρομικοί αλγόριθμοι)
  - Π.χ.: $T_{MS}(n) <= 2T_{MS}(n/2)+cn = \ldots = O(n \log n)$
    (MS = MergeSort, *c* κάποια σταθερά)

Πίνακας χρονικής πολυπλ/τας

$O(1)$	a := b*c;	απλές εντολές
$O(\log n)$	if x<A[n/2] search(A[1,n/2])…	δυαδική αναζήτηση
$O(n)$	for i:=1 to n do <O(1)>	απλός βρόχος
$O(n \log n)$	mergesort(A[1, n/2]) mergesort(A[n/2+1, n]) merge(A[1, n/2], A[n/2+1, n])	ταξινόμηση με συγχώνευση
$O(n^2)$	for i:=1 to n do for j:=1 to n do <O(1)>	διπλός βρόχος
$O(2^n)$	for all S $\subseteq \{0,1\}^n$ do <O(1)>	υποσύνολα
$O(n!)$	for all σ in S[n] do <O(1)>	μεταθέσεις

Αλγόριθμοι divide & conquer

$O(\log n)$	if x<A[n/2] search(A[1,n/2])…	δυαδική αναζήτηση
$O(\max(\text{len}(a),\text{len}(b))^3)$	GCD(a,b) := GCD(b,a **mod** b)	εύρεση ΜΚΔ
$O(\text{len}(n))$ *	pow(a,n) := pow(a^2,n/2)	ύψωση σε δύναμη
$O(\text{len}(n))$ *	αλγόριθμος πίνακα fast doubling	υπολογισμός n-οστού αριθμού Fibonacci
$O(n \log n)$	mergesort(A[1, n/2]) mergesort(A[n/2+1, n]) merge(A[1, n/2], A[n/2+1, n])	ταξινόμηση με συγχώνευση
$O(n^{\log 3})$	αλγόριθμος Gauss-Karatsuba	πολλαπλασιασμός n-ψήφιων αριθμών
$O(n^{\log 7})$	αλγόριθμος Strassen	πολλαπλασιασμός πινάκων n x n

* αριθμητική πολυπλοκότητα, len(x) = #ψηφίων του x = $\Theta(\log x)$