Διακριτά Μαθηματικά

Ενότητα 1: Ανάλυση Αλγορίθμων

Αν. Καθηγητής Κ. Στεργίου
e-mail: kstergiou@uowm.gr

Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Περιεχόμενα

- **Πολυπλοκότητα αλγορίθμων:**
  - χρονική πολυπλοκότητας χειρότερης περίπτωσης,
  - μελέτη παραδειγμάτων:
    - ✓αναζήτηση,
    - ✓ταξινόμηση,
    - ✓αλγόριθμος Dijkstra.
- **Ασυμπτωτική ανάλυση:**
  - ασυμπτωτική συμπεριφορά αριθμητικών συναρτήσεων,
  - ο συμβολισμός Ο,
  - οι συμβολισμοί Θ και Ω.

Στόχοι

- Εισαγωγή στην έννοια της πολυπλοκότητας ενός αλγόριθμου.
- Μελέτη της χρονικής πολυπλοκότητας χειρότερης περίπτωσης συγκεκριμένων απλών αλγορίθμων.
  - Εξοικείωση με τον τρόπο ανάλυσης της πολυπλοκότητας για σχετικά απλούς αλγορίθμους.
- Εισαγωγή στην έννοια της ασυμπτωτικής ανάλυσης και των συμβολισμών Ο, Θ, Ω.
- Κατανόηση των διαφορετικών τάξεων πολυπλοκότητας και της πρακτικής τους σημασίας.

Αλγόριθμος

- Ένας ***αλγόριθμος*** είναι μια βήμα-προς-βήμα περιγραφή του τρόπου εκτέλεσης μιας συγκεκριμένης διεργασίας.
  - Ο *αλγόριθμος γραμμικής αναζήτησης* είναι μια βηματική περιγραφή της διεργασίας αναζήτησης ενός συγκεκριμένου αριθμού σε μια λίστα αριθμών.
  - Ο *αλγόριθμος του Prim* είναι μια βηματική περιγραφή της διεργασίας υπολογισμού ενός ελάχιστου επικαλύπτοντος δέντρου σε ένα βεβαρημένο γράφημα.
- Στην ενότητα αυτή του μαθήματος θα μελετήσουμε την ανάλυση πολυπλοκότητας αλγορίθμων.

Ανάλυση Αλγορίθμων

- Πως μπορούμε να εκτιμήσουμε έναν αλγόριθμο που προτάθηκε για τη λύση ενός συγκεκριμένου προβλήματος;
- **Είναι ορθός**; Εκτελεί τη διεργασία που πρέπει για όλα τα στιγμιότυπα του προβλήματος;
  - π.χ. Ο αλγόριθμος του Prim υπολογίζει ένα ελάχιστο επικαλύπτων δέντρο για οποιονδήποτε εμβαρημένο γράφημα;
- **Πόσο καλά αποτελέσματα παράγει**;
  - παράδειγμα;
- **Αποδοτικότητα**; Ποιο είναι το κόστος εκτέλεσης του;
  - **Χρόνος εκτέλεσης**;
    - ποια είναι η χρονική του πολυπλοκότητα;
  - **Χώρος που χρησιμοποιείται**;
    - ποια είναι η χωρική του πολυπλοκότητα;

Πολυπλοκότητα Αλγορίθμων

- Η *αλγοριθμική πολυπλοκότητα* ενός υπολογισμού είναι γενικά ένα μέτρο του πόσο *δύσκολο* είναι να πραγματοποιηθεί ο υπολογισμός.
- Δηλαδή, μετράει το ***κόστος*** του υπολογισμού
  - Το πλήθος των πόρων που απαιτούνται για τον υπολογισμό.
- Οι πιο κοινές μετρικές πολυπλοκότητας είναι:
  - "Χρονική" πολυπλοκότητα: # λειτουργιών ή βημάτων που απαιτούνται,
  - "Χωρική" πολυπλοκότητα: # θέσεων μνήμης (π.χ. bits) που απαιτούνται.

Χρονική Πολυπλοκότητα Αλγορίθμων (1/2)

- Ας υποθέσουμε ότι θέλουμε να βρούμε το μέγιστο σε μια ακολουθία n αριθμών.
- Πόσο χρόνο πρέπει να ξοδέψουμε;
- Ας υποθέσουμε ότι απαιτείται 1 μονάδα χρόνου για να γίνει μια σύγκριση μεταξύ δύο αριθμών.
- Δεν έχουμε ιδέα για το πόσο χρόνο θα χρειαστεί το πρόγραμμα για να τρέξει, αλλά γνωρίζουμε ότι αυτό εξαρτάται γραμμικά από το n.

Σε αυτή την περίπτωση χρειαζόμαστε περίπου n μονάδες χρόνου

Λέμε ότι ο αλγόριθμος έχει χρόνο εκτέλεσης "τάξης n"

Χρονική Πολυπλοκότητα Αλγορίθμων (2/2)

- Έχω ένα νούμερο μεταξύ 0 και 63. Κάνετε μια ερώτηση κι εγώ απαντάω με ναι ή όχι.
- Πόσο χρόνο θα χρειαστείτε για να βρείτε τον αριθμό στη χειρότερη περίπτωση;
- Ας υποθέσουμε ότι απαιτείται 1 μονάδα χρόνου για να τεθεί και να απαντηθεί μια ερώτηση.
- Δεν έχουμε ιδέα για το πόσο χρόνο θα χρειαστεί το πρόγραμμα για να τρέξει, αλλά γνωρίζουμε ότι αυτό εξαρτάται λογαριθμικά από το n.

Τότε χρειαζόμαστε περίπου log n μονάδες χρόνου

Λέμε ότι ο αλγόριθμος έχει χρόνο εκτέλεσης "τάξης log n"

Παράδειγμα αλγόριθμου (1/5)

- **Πρόβλημα:** Δοθείσης μιας ακολουθίας $\{a_i\}=a_1,\ldots,a_n$, $a_i \in \mathbb{N}$, βρες το μεγαλύτερο της στοιχείο.
- Αλγόριθμος **MAX1**:
  - Θέσε την τιμή μιας *προσωρινής μεταβλητής v* (που θα αποθηκεύει το μεγαλύτερο στοιχείο που έχουμε δει μέχρι τώρα) στην τιμή a_1.
  - Για $i=2,3,\ldots,n$ επανέλαβε τα εξής:
    - Πάρε το επόμενο στοιχείο a_i στην ακολουθία. Αν $a_i>v$, τότε ανάθεσε στην v τον αριθμό a_i. Αλλιώς μην κάνεις τίποτα.
  - Επέστρεψε την τιμή της μεταβλητής v.

Παράδειγμα αλγόριθμου (2/5)

- **Απόδειξη Ορθότητας**: Με επαγωγή στον αριθμό των βημάτων (επαναλήψεων) του αλγόριθμου.
  - ***Βάση***: Για i=2, η μεταβλητή v έχει την μεγαλύτερη τιμή ανάμεσα στους a_1 και a_2.
  - ***Υπόθεση***: Για i=k, η μεταβλητή v έχει την μεγαλύτερη τιμή ανάμεσα στους a_1 ,…, a_k .
  - ***Επαγωγικό Βήμα***: Η εκτέλεση του βήματος k+1 συγκρίνει τον αριθμό a_{k+1} με τον μέγιστο ανάμεσα στους αριθμούς a_1 ,…, a_k και αναθέτει τον μεγαλύτερο από τους δύο στη μεταβλητή v. Άρα μετά την εκτέλεση του βήματος, η μεταβλητή v θα έχει αποθηκευμένο τον μέγιστο ανάμεσα στους αριθμούς a_1 ,.., a_{k+1} .

Παράδειγμα αλγόριθμου (3/5)

- **Χρονική Πολυπλοκότητα:**
  - Καθένας από τους αριθμούς a_1 ,…, a_n συγκρίνεται με τη μεταβλητή *v* μια φορά. Αν *c* είναι ο χρόνος που απαιτείται για να συγκριθούν δυο αριθμοί, τότε ο συνολικός χρόνος που απαιτείται για τις συγκρίσεις είναι *c*(*n*-1).
    - αν λάβουμε υπ'όψην μας και τον χρόνο που απαιτείται για την αντιγραφή ενός αριθμού στη μεταβλητή *v* τότε η ανάλυση γίνεται πολύ πιο περίπλοκη αφού πρέπει να κάνουμε υποθέσεις για την κατανομή των αριθμών στην ακολουθία. Απλά υποθέτουμε ότι ο χρόνος αυτός συμπεριλαμβάνεται στην ποσότητα *c*.
    - Το *c* εξαρτάται από τον συγκεκριμένο Η/Υ που χρησιμοποιούμε. Για αυτό λέμε ότι η χρονική πολυπλοκότητα του αλγόριθμου είναι ανάλογη προς το *n*-1.

Παράδειγμα αλγόριθμου (4/5)

- **Πρόβλημα:** Δοθείσης μιας ακολουθίας $\{a_i\}=a_1,\ldots,a_n$, $a_i \in \mathbb{N}$, βρες το μεγαλύτερο της στοιχείο.
- Αλγόριθμος **MAX2**:
  - Για i=1,2,…,n-1 επανέλαβε τα εξής:
    - Σύγκρινε τα στοιχεία a_i και a_{i+1} της ακολουθίας. Αν $a_i > a_{i+1}$, τότε άλλαξε θέση στους δύο αριθμούς. Αλλιώς μην κάνεις τίποτα.
  - Επέστρεψε την τιμή του στοιχείου a_n.

Παράδειγμα αλγόριθμου (5/5)

- **Απόδειξη Ορθότητας:**
  - Με επαγωγή.
- **Χρονική Πολυπλοκότητα:**
  - Ο αλγόριθμος MAX2 εκτελεί *n*-1 συγκρίσεις αριθμών. Άρα η χρονική του πολυπλοκότητα είναι ανάλογη προς το *n*-1.
- **Τι διαφορά έχει με τον MAX1;**
  - Μεταβάλει τα δεδομένα στη μνήμη αλλάζοντας θέση στους αριθμούς, ενώ ο MAX1 απλά γράφει το αποτέλεσμα σε μια άλλη θέση μνήμης.
- **Γενικά ένα πρόβλημα είναι πιθανό να λύνεται με πλήθος διαφορετικών αλγορίθμων,**
  - που μπορεί να έχουν ίδιες ή διαφορετικές πολυπλοκότητες.

Η πολυπλοκότητα εξαρτάται από το input

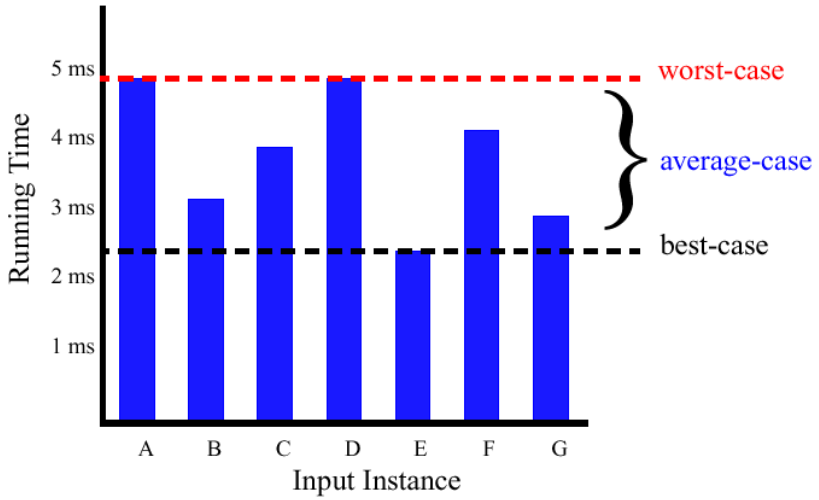
- Οι περισσότεροι αλγόριθμοι έχουν διαφορετικούς χρόνους εκτέλεσης για inputs διαφορετικού μεγέθους.
  - *Π.χ.* Η αναζήτηση σε μια μεγάλη λίστα συνήθως παίρνει περισσότερο χρόνο από την αναζήτηση σε μια μικρή λίστα.
- Για αυτό τον λόγο, η χρονική πολυπλοκότητα συνήθως εκφράζεται ως *συνάρτηση* του μεγέθους του input.
  - Αυτή η συνάρτηση συνήθως δίνει την πολυπλοκότητα για την ***χειρότερη περίπτωση*** input κάθε συγκεκριμένου μεγέθους.
  - Πιο σπάνια χρησιμοποιούνται και συναρτήσεις που εκφράζουν τις πολυπλοκότητες ***καλύτερης*** και ***μέσης περίπτωσης.***

Μέγεθος input

- Με ποια παράμετρο μπορούμε να αναπαραστήσουμε το μέγεθος του input ενός αλγορίθμου;
- **Παραδείγματα:**
  - Διάταξη λίστας ακέραιων αριθμών:
    - **Το μέγεθος της λίστας.**
  - Εύρεση του αν ένα γράφημα έχει κύκλους:
    - **Το πλήθος των κόμβων και ακμών του γραφήματος.**
  - Ανάθεση μαθημάτων σε διδάσκοντες:
    - **Το πλήθος των μαθημάτων και των διδασκόντων.**

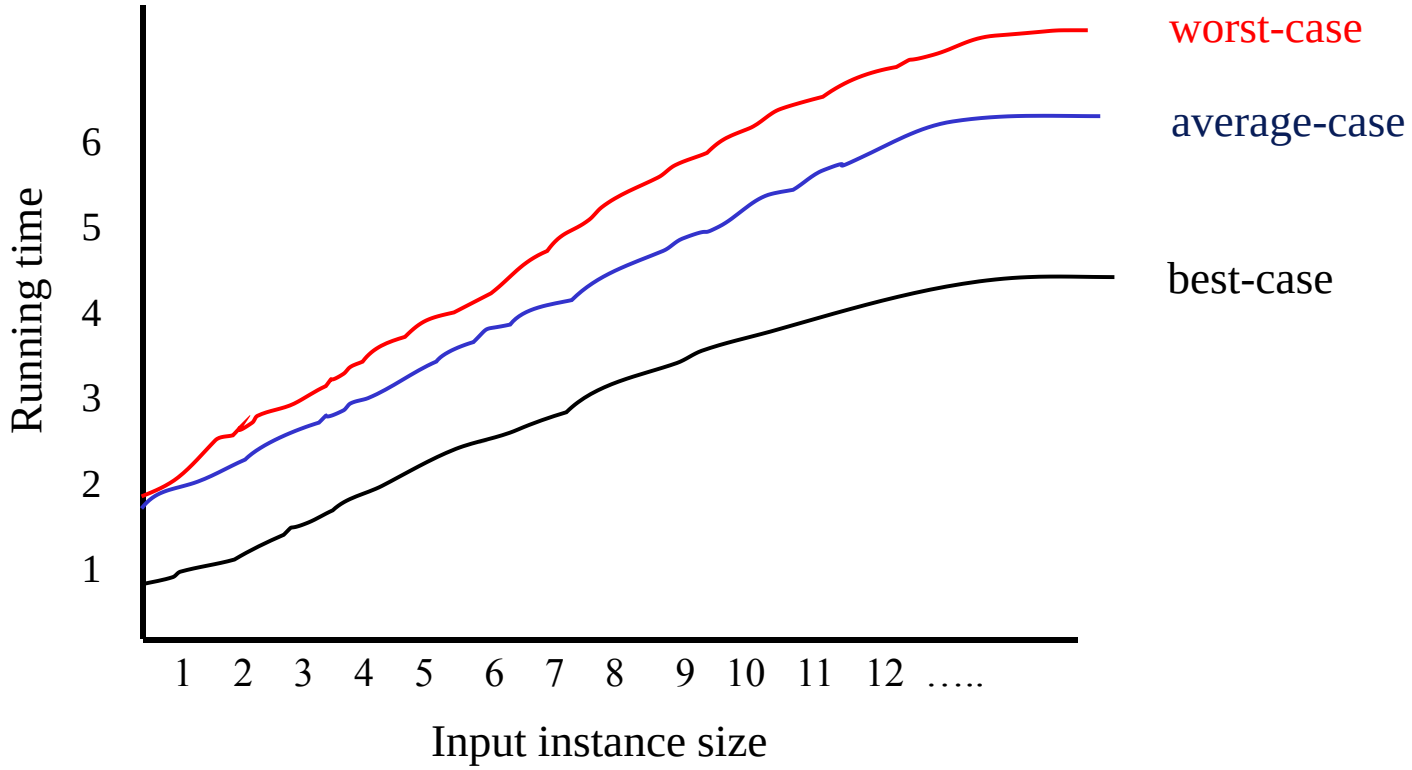
Καλύτερη/ Χειρότερη / Μέση Περίπτωση (1/3)

- Για ένα συγκεκριμένο μέγεθος input *n*, διερευνούμε τους χρόνους εκτέλεσης για διαφορετικά στιγμιότυπα του input:



Καλύτερη/ Χειρότερη / Μέση Περίπτωση (2/3)

- Για inputs όλων των μεγεθών:



Καλύτερη/ Χειρότερη / Μέση Περίπτωση (3/3)

- Η **Χειρότερη Περίπτωση** (Worst case) χρησιμοποιείται συνήθως γιατί:
  - Είναι ένα πάνω όριο και σε συγκεκριμένες εφαρμογές (e.g., έλεγχος εναέριας κυκλοφορίας, εγχειρήσεις) το να ξέρουμε τη χρονική **πολυπλοκότητα χειρότερης περίπτωσης** έχει πολύ μεγάλη σημασία.
  - Για μερικούς αλγόριθμους η **χειρότερη περίπτωση** συμβαίνει σχετικά συχνά.
  - Η **μέση περίπτωση** είναι συχνά όσο κακή είναι και η **χειρότερη.**
  - Η εύρεση της **μέσης περίπτωσης** είναι συχνά πολύ δύσκολη.

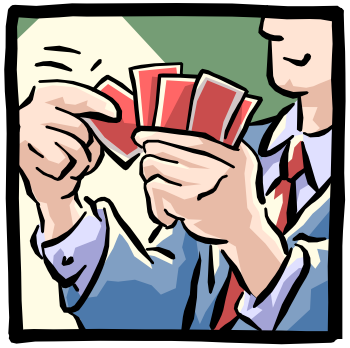
Παράδειγμα: Διάταξη Αριθμών (1/12)

INPUT

Μια σειρά αριθμών

$a_1, a_2, a_3, \ldots, a_n$

2 5 4 10 7



OUTPUT

Η σειρά των αριθμών διατεταγμένη

$b_1, b_2, b_3, \ldots, b_n$

2 4 5 7 10

Ορθότητα

Για οποιοδήποτε input ο αλγόριθμος σταματάει με το output:

- $b_1 < b_2 < b_3 < \ldots < b_n$
- $b_1, b_2, b_3, \ldots, b_n$ είναι μια αναδιάταξη των $a_1, a_2, a_3, \ldots, a_n$

Χρόνος εκτέλεσης

Εξαρτάται από

- πλήθος αριθμών (n)
- πόσο (μερικώς) διατεταγμένοι είναι
- τον αλγόριθμο

Παράδειγμα: Διάταξη Αριθμών (2/12)

- Λέγοντας ότι *διατάσουμε n* αριθμούς $x_1 \ldots x_n$, εννοούμε ότι τους μεταθέτουμε έτσι ώστε τελικά να είναι διατεταγμένοι σε αύξουσα (ή φθίνουσα) διάταξη.
- Αλγόριθμος **Bubble Sort**
  - Κάνε τα εξής για $i=n,n-1,n-2,\ldots,4,3,2$: Βρες και βάλε στη θέση x_i τον μεγαλύτερο από τους i αριθμούς $x_1 \ldots x_i$.
  - Τελικά οι αριθμοί $x_1 \ldots x_n$ είναι σε αύξουσα διάταξη.

```
for i=n to 2 do
    for j = 1 to i-1 do
    if x_j > x_{j+1} then
            swap (x_j, x_{j+1})
```

MAX2

Παράδειγμα: Διάταξη Αριθμών (3/12)

8	6	34	2	51	32	21	**αρχική ακολουθία**
6	**8**	34	2	51	32	21	**πέρασμα 1**
6	**8**	**34**	2	51	32	21	
6	8	**2**	**34**	51	32	21	
6	8	2	**34**	**51**	32	21	
6	8	2	34	**32**	**51**	21	
6	8	2	34	32	**21**	**51**	

Παράδειγμα: Διάταξη Αριθμών (4/12)

6	8	2	34	32	21	**51**	
6	**8**	2	34	32	21	51	**πέρασμα 2**
6	**2**	**8**	34	32	21	51	
6	2	**8**	**34**	32	21	51	
6	2	8	**32**	**34**	21	51	
6	2	8	32	**21**	**34**	51	

Επανέλαβε τη διαδικασία μέχρι η λίστα να έχει ταξινομηθεί

Παράδειγμα: Διάταξη Αριθμών (5/12)

- Για να εξακριβώσουμε ότι ο Bubble Sort είναι ορθός, παρατηρούμε ότι:
  - για $i=n$ η εκτέλεση του βήματος 1 τοποθετεί τον μεγαλύτερο από τους n αριθμούς στη θέση n.
  - για $i=n$-1 η εκτέλεση του βήματος 1 τοποθετεί τον δεύτερο μεγαλύτερο από τους n αριθμούς στη θέση n-1.
  - τελικά για i=2 η εκτέλεση του βήματος 1 τοποθετεί τον (n-1)-μεγαλύτερο (δηλ. τον δεύτερο μικρότερο) από τους n αριθμούς στη θέση n-1.
  - άρα ο αριθμός που απομένει στη θέση 1 είναι ο μικρότερος από τους n αριθμούς.
- Οπότε ο αλγόριθμος πραγματικά διατάσσει τους αριθμούς σε αύξουσα διάταξη.

Παράδειγμα: Διάταξη Αριθμών (6/12)

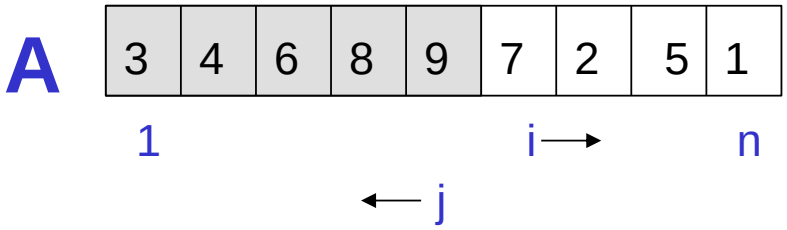
- **Χρονική Πολυπλοκότητα:**
  - Κατά την πρώτη εκτέλεση του βήματος 1 (δηλ. όταν $i=n$) εκτελούνται n-1 συγκρίσεις, κατά τη δεύτερη εκτελούνται n-2 συγκρίσεις, κ.ο.κ. Κατά την n-1 εκτέλεση του βήματος 1 εκτελείται μια σύγκριση. Άρα ο συνολικός αριθμός συγκρίσεων που εκτελούνται είναι:

$$(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$$

  - Επομένως η χρονική πολυπλοκότητα του αλγόριθμου είναι ανάλογη προς το $n(n-1)/2$.

Παράδειγμα: Διάταξη Αριθμών (7/12)

- Αλγόριθμος **Insertion Sort**



Στρατηγική

- Ξεκίνα "με άδεια χέρια"
- Βάλε έναν αριθμό στην κατάλληλη θέση του χεριού όπου κάνεις τη διάταξη
- Συνέχισε ώσπου όλοι οι αριθμοί έχουν διαταχθεί

```
for i=2 to n
    key=x_i
    "insert x_i into the
    sorted sequence x_1…x_i-1"
      j=i-1
      while j>0 and x_j>key
         do x_j+1=x_j
            j--
      x_j+1=key
```

Παράδειγμα: Διάταξη Αριθμών (8/12)

8	6	34	2	51	32	21	**αρχική ακολουθία**
6	**8**	34	2	51	32	21	**πέρασμα 1**
6	**8**	**34**	2	51	32	21	**πέρασμα 2**
2	**6**	**8**	**34**	51	32	21	**πέρασμα 3**
2	**6**	**8**	**34**	**51**	32	21	**πέρασμα 4**
2	**6**	**8**	**32**	**34**	**51**	21	**πέρασμα 5**
2	**6**	**8**	**21**	**32**	**34**	**51**	**πέρασμα 6**

Παράδειγμα: Διάταξη Αριθμών (9/12)

- Απόδειξη ορθότητας για τον αλγόριθμο Insertion Sort;
  - παρόμοια με τον bubble sort.
- **Χρονική Πολυπλοκότητα:**
  - Κατά την πρώτη εκτέλεση του βήματος 1 (δηλ. όταν *i=2*) εκτελείται 1 σύγκριση, κατά τη δεύτερη εκτελούνται 2 συγκρίσεις, κ.ο.κ. Κατά την *n*-1 εκτέλεση του βήματος 1 εκτελούνται n-1 συγκρίσεις. Άρα ο συνολικός αριθμός συγκρίσεων που εκτελούνται είναι:

    $$1 + 2 + \ldots + (n-2) + (n-1) = n(n-1)/2$$

  - Επομένως η χρονική πολυπλοκότητα του αλγόριθμου είναι ανάλογη προς το $n(n-1)/2$.

Παράδειγμα: Διάταξη Αριθμών (10/12)

- Αλγόριθμος **Selection Sort**

Στρατηγική

• Ψάξε όλη τη λίστα και επέλεξε το μικρότερο στοιχείο. Αντάλλαξε το με το πρώτο στοιχείο της λίστας
• Συνέχισε με τον ίδιο τρόπο μέχρι να γίνουν n-1 τέτοια περάσματα στη λίστα

```
for i=1 to n-1 do
  mikr=x_i
  for j = i to n do
    if (x_j<x_mikr)
      mikr=j
  swap(x_i,x_mikr)
```

Παράδειγμα: Διάταξη Αριθμών (11/12)

8	6	34	2	51	32	21	**αρχική ακολουθία**
2	6	34	8	51	32	21	**πέρασμα 1**
2	**6**	34	8	51	32	21	**πέρασμα 2**
2	**6**	**8**	34	51	32	21	**πέρασμα 3**
2	**6**	**8**	**21**	51	32	34	**πέρασμα 4**
2	**6**	**8**	**21**	**32**	51	34	**πέρασμα 5**
2	**6**	**8**	**21**	**32**	**34**	51	**πέρασμα 6**

Παράδειγμα: Διάταξη Αριθμών (12/12)

- Απόδειξη ορθότητας για τον αλγόριθμο Selection Sort;
  - παρόμοια με τον bubble και τον insertion sort.
- **Χρονική Πολυπλοκότητα:**
  - Κατά την πρώτη εκτέλεση του βήματος 1 (δηλ. όταν i=1) εκτελούνται n-1 συγκρίσεις, κατά τη δεύτερη εκτελούνται n-2 συγκρίσεις, κ.ο.κ. Κατά την n-1 εκτέλεση του βήματος 1 εκτελείται μια σύγκριση. Άρα ο συνολικός αριθμός συγκρίσεων που εκτελούνται είναι:

$$(n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2$$

  - Επομένως η χρονική πολυπλοκότητα του αλγόριθμου είναι ανάλογη προς το $n(n-1)/2$.

Παράδειγμα: Αλγόριθμος του Dijkstra (1/4)

- Θέλουμε να βρούμε το ελάχιστο μονοπάτι από μια κορυφή *s* προς όλες τις άλλες σε ένα εμβαρημένο γράφημα *G* με *V* κορυφές.

```
DIJKSTRA(G, w, s)
 1 for each v ∈ V
 2       do d[v] ← ∞
 3 d[s] ← 0
 4 S ← ∅   ▷ Set of discovered node
 5 Q ← V
 6 while Q ≠ ∅
 7       do u ← EXTRACT-MIN(Q)
 8          S ← S ∪ {u}
 9          for each v ∈ Adj[u]
10                do if d[v] > d[u] + w(u, v)
11                      then d[v] ← d[u] + w(u, v)
```

w: συνάρτηση βάρους ακμής

d: μικρότερη απόσταση (δείκτης) κάθε κορυφής

S: σύνολο κορυφών για τις οποίες έχουμε βρει το ελάχιστο μονοπάτι

Παράδειγμα: Αλγόριθμος του Dijkstra (2/4)

- **Χρονική Πολυπλοκότητα:**
  - Ο Αλγόριθμος ξεκινάει με *S*={}, διευρύνει το σύνολο *S* κατά μια κορυφή σε κάθε επανάληψη, και τερματίζει όταν όλες οι κορυφές είναι στο *S*. Αν *n* είναι ο αριθμός των κορυφών, θα εκτελεστούν *n* επαναλήψεις. Σε κάθε επανάληψη επιλέγεται η κορυφή *u* με τον μικρότερο δείκτη ως προς το *S* και υπολογίζονται ξανά οι δείκτες για όλες τις γειτονικές κορυφές της *u*. Κατά την πρώτη επανάληψη, επιλέγουμε την κορυφή *s* και υπολογίζουμε τους δείκτες *n*-1 κορυφών στη χειρότερη περίπτωση. Κατά τη δεύτερη επανάληψη επιλέγουμε μια από *n*-1 κορυφές και υπολογίζουμε ξανά τους δείκτες *n*-2 κορυφών. Κατά την *n*-2 επανάληψη επιλέγουμε την προτελευταία κορυφή και υπολογίζουμε τους δείκτες της τελευταίας κορυφής.

Παράδειγμα: Αλγόριθμος του Dijkstra (3/4)

- **Χρονική Πολυπλοκότητα:**
  - Ποια είναι η χρονική πολυπλοκότητα επιλογής μιας από *n* κορυφές;
    - Ανάλογη του *n*-1 μια και το πρόβλημα είναι ουσιαστικά το πρόβλημα εύρεσης του μικρότερου ανάμεσα σε *n* αριθμούς.
  - Ποια είναι η χρονική πολυπλοκότητα υπολογισμού *n* δεικτών; Πόσος χρόνος απαιτείται για τον υπολογισμό ενός δείκτη;
    - Σταθερός χρόνος. Άρα για *n* δείκτες η πολυπλοκότητα είναι ανάλογη του *n.*
  - Συνεπώς η πολυπλοκότητα της πρώτης επανάληψης είναι:

$$(n\text{-}1) + (n\text{-}1)$$

    - η πολυπλοκότητα της δεύτερης επανάληψης είναι:

$$(n\text{-}2) + (n\text{-}2)$$

    - η πολυπλοκότητα της *n*-2 επανάληψης είναι:

$$1 + 1$$

Παράδειγμα: Αλγόριθμος του Dijkstra (4/4)

- **Χρονική Πολυπλοκότητα:**
  - Άρα για όλες τις επαναλήψεις η πολυπλοκότητα είναι:

    2 ((n-1) + (n-2) + … + 2 + 1) = 2 (n(n-1)/2) = n(n-1)
  - Άρα η πολυπλοκότητα είναι ανάλογη του n^2. Γιατί;
    - Διότι ο παράγοντας 1 στο n-1 είναι αμελητέος. Φανταστείτε n=1.000.000. Τότε το $n(n-1)$ και n^2 έχουν ελάχιστη διαφορά.

Πολυπλοκότητα Προβλημάτων (1/2)

- Πως μπορούμε να προσδιορίσουμε την πολυπλοκότητα ενός προβλήματος;
  - δοθέντος ενός προβλήματος ποια είναι η μικρότερη δυνατή πολυπλοκότητα ενός αλγόριθμου επίλυσης του;
  - μπορούμε να εκτελέσουμε αναζήτηση του μεγαλύτερου με πολυπλοκότητα μικρότερη από *n*-1;
- Αν γνωρίζουμε τη χρονική πολυπλοκότητα ενός προβλήματος τότε ξέρουμε ότι υπάρχει ένας αλγόριθμος με αυτή την πολυπλοκότητα που λύνει το πρόβλημα και δεν υπάρχει αλγόριθμος με καλύτερη πολυπλοκότητα.
  - ο προσδιορισμός της χρονικής πολυπλοκότητας ενός προβλήματος είναι γενικά μια δύσκολη διεργασία.

Πολυπλοκότητα Προβλημάτων (2/2)

- Συχνά μπορούμε μόνο να προσδιορίσουμε άνω και κάτω φράγμα στην πολυπλοκότητα ενός προβλήματος.
  - ***Άνω φράγμα***: ο καλύτερος δυνατός αλγόριθμος επίλυσης του προβλήματος έχει πολυπλοκότητα μικρότερη ή ίση με το φράγμα.
    - Ο προσδιορισμός ενός άνω φράγματος είναι εύκολος. Αρκεί να βρούμε έναν αλγόριθμο που λύνει το πρόβλημα.
  - ***Κάτω φράγμα***: ο καλύτερος δυνατός αλγόριθμος επίλυσης του προβλήματος έχει πολυπλοκότητα μεγαλύτερη ή ίση με το φράγμα.
    - Ο προσδιορισμός ενός κάτω φράγματος είναι πιο δύσκολος. Πρέπει να αποδείξουμε ότι δεν υπάρχει καλύτερος αλγόριθμος.
  - Αν ένα άνω φράγμα συμπέσει με ένα κάτω φράγμα τότε έχουμε προσδιορίσει την πολυπλοκότητα του προβλήματος.

Πολυπλοκότητα Προβλημάτων – Παράδειγμα

- Για το πρόβλημα εύρεσης του μεγαλύτερου μεταξύ *n* αριθμών έχουμε ένα άνω φράγμα: *n*-1 βήματα σύγκρισης.
- Πως θα δείξουμε ότι το *n*-1 είναι και κάτω φράγμα;
  - **Απόδειξη:** Υποθέστε ότι βρίσκουμε έναν αλγόριθμο που λύνει το πρόβλημα με λιγότερα από *n*-1 βήματα. Ας κατασκευάσουμε ένα γράφημα με *n* κορυφές που αντιστοιχούν στους *n* αριθμούς. Αν ο αλγόριθμος συγκρίνει δύο αριθμούς που αντιστοιχούν στις κορυφές *v* και *u* τότε υπάρχει ακμή μεταξύ των *v* και *u*. Μια και ο αλγόριθμος κάνει λιγότερα από *n*-1 βήματα, υπάρχουν λιγότερες από *n*-1 ακμές. Άρα το γράφημα δεν είναι συνεκτικό. Οπότε οι αριθμοί χωρίζονται σε (τουλάχιστον) δύο διαζευγμένα υποσύνολα. Ο μεγαλύτερος αριθμός μέσα σε κάθε υποσύνολο μπορεί να προσδιοριστεί, αλλά δε μπορεί να προσδιοριστεί ο μεγαλύτερος ανάμεσα σε ξεχωριστά υποσύνολα. Οπότε ο αλγόριθμος δεν είναι σωστός και ως αποτέλεσμα δε μπορεί να υπάρξει αλγόριθμος που λύνει το πρόβλημα με λιγότερα από *n*-1 βήματα.

Τεστ

- Ποια είναι η χρονική πολυπλοκότητα των παρακάτω κομματιών ψευδοκώδικα σε σχέση με το input;

- F1(n)
  - i ← 0
- F2(n)
  - **while** n > 1
  - **do** n ← n-1
- F3(n)
  - **while** n > 1
  - **do** n ← ⌊n/2⌋

- F4(n)
- **while** n > 1
  - **do for** i = 1 **to** n
    - **do** j ← i
  - n ← n-1

Ασυμπτωτική Ανάλυση Αλγορίθμων

Αριθμητικές Συναρτήσεις

- Μια συνάρτηση είναι μια διμελής σχέση που σε κάθε στοιχείο του πεδίου ορισμού αντιστοιχίζει ένα στοιχείο του πεδίου τιμών.
- Μια (***διακριτή***) ***αριθμητική συνάρτηση*** έχει πεδίο ορισμού το σύνολο των φυσικών αριθμών και πεδίο τιμών το σύνολο των πραγματικών αριθμών.
  - $a_r = 7r^3 + 1 \quad r \geq 0$
  - $b_r = \begin{cases} 2r & 0 \leq r \leq 11 \\ 3^r - 1 & r > 11 \end{cases}$
- οι αριθμητικές συναρτήσεις εμφανίζονται πολύ συχνά στον ψηφιακό υπολογισμό.

Ασυμπτωτική Συμπεριφορά Αριθμητικών Συναρτήσεων (1/3)

- Για τις αριθμητικές συναρτήσεις, συχνά χρειάζεται να γνωρίζουμε ένα μέτρο του ***πόσο γρήγορα αυξάνεται η συνάρτηση***.
- Αν η $f(x)$ ***αυξάνεται γρηγορότερα*** της $g(x)$, τότε η $f(x)$ πάντα κάποια στιγμή γίνεται μεγαλύτερη της $g(x)$ ***οριακά*** (δηλαδή για **αρκετά μεγάλες** τιμές του x).
- Αυτό είναι πολύ χρήσιμο στους επιστήμονες υπολογιστών και στους μηχανικούς για να δείχνουμε ότι ένας αλγόριθμος ή μια κατασκευή *κλιμακώνεται (scales)* καλύτερα ή χειρότερα από μια άλλη.
  - δηλ. έχει καλύτερη (πιο αποδοτική) συμπεριφορά καθώς το μέγεθος του προβλήματος που επιλύει αυξάνεται.

Ασυμπτωτική Συμπεριφορά Αριθμητικών Συναρτήσεων (2/3)

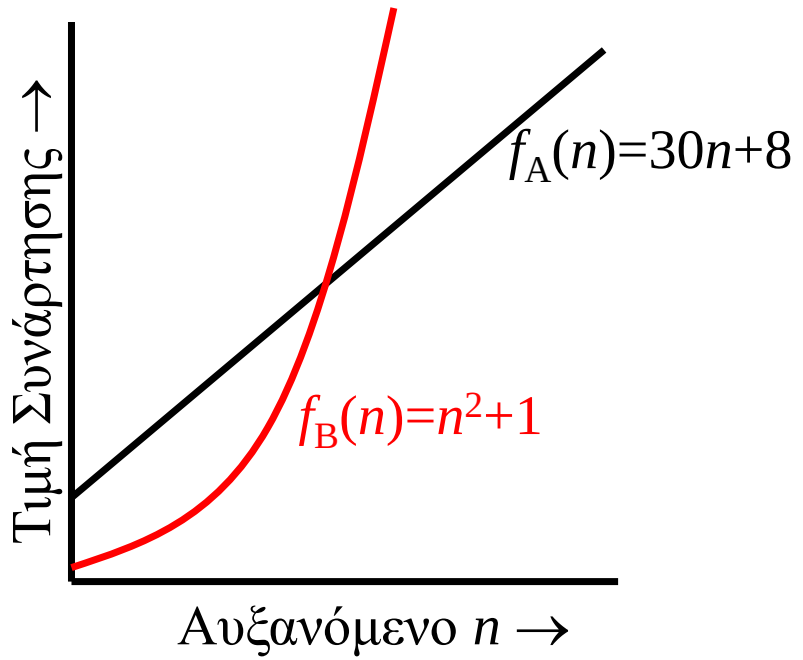
- Με τον όρο ***ασυμπτωτική συμπεριφορά*** μιας αριθμητικής συνάρτησης $f(n)$ εννοούμε το πως μεταβάλλεται η τιμή της συνάρτησης για μεγάλο n.
- Αν έχουμε έναν αλγόριθμο για ταξινόμηση ονομάτων και ο χρόνος που απαιτεί για n αριθμούς δίνεται από τη συνάρτηση $f(n)$ τότε είναι πολύ σημαντικό το πως αυξάνεται το $f(n)$ όταν χρησιμοποιούμε τον αλγόριθμο για να ταξινομήσουμε δεκάδες χιλιάδες ονομάτων.
  - ενώ δεν είναι ιδιαίτερα σημαντικό αν χρησιμοποιούμε τον αλγόριθμο μόνο για ταξινόμηση μερικών εκατοντάδων ονομάτων το πολύ.

Ασυμπτωτική Συμπεριφορά Αριθμητικών Συναρτήσεων (3/3)

- Για παράδειγμα υποθέστε ότι σχεδιάζετε ένα web site για την επεξεργασία δεδομένων χρηστών (*π.χ.*, αρχεία εσόδων-εξόδων).
- Υποθέστε ότι το πρόγραμμα Α για ανάκτηση από βάση δεδομένων χρειάζεται $f_A(n)=30n+8$ microseconds για την επεξεργασία n εγγραφών, ενώ το πρόγραμμα Β χρειάζεται $f_B(n)=n^2+1$ microseconds για την επεξεργασία των n εγγραφών.
- Ποιο πρόγραμμα θα διαλέγατε αν ξέρατε ότι πρέπει να υποστηρίξετε εκατομμύρια χρηστών;

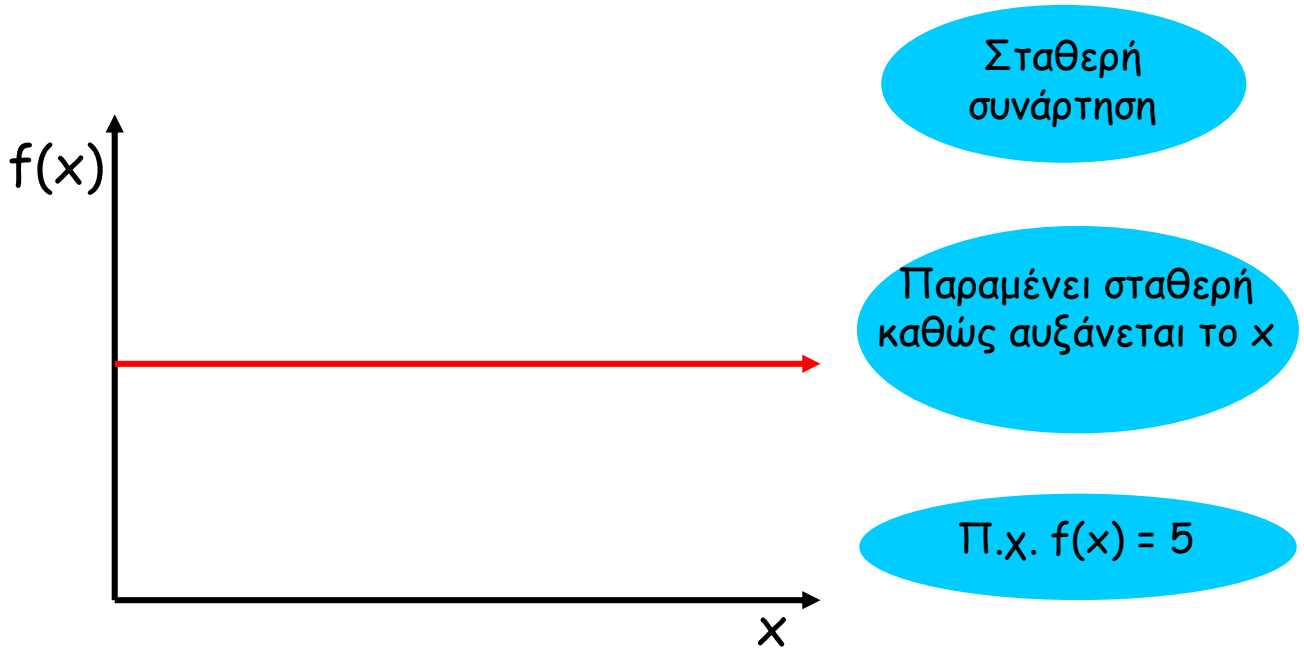
Τάξη Μεγέθους Αύξησης Συναρτήσεων (1/8)

- Στη γραφική αναπαράσταση των συναρτήσεων, καθώς προχωράμε προς τα δεξιά, η συνάρτηση που αυξάνεται γρηγορότερα πάντα κάποια στιγμή γίνεται η μεγαλύτερη...



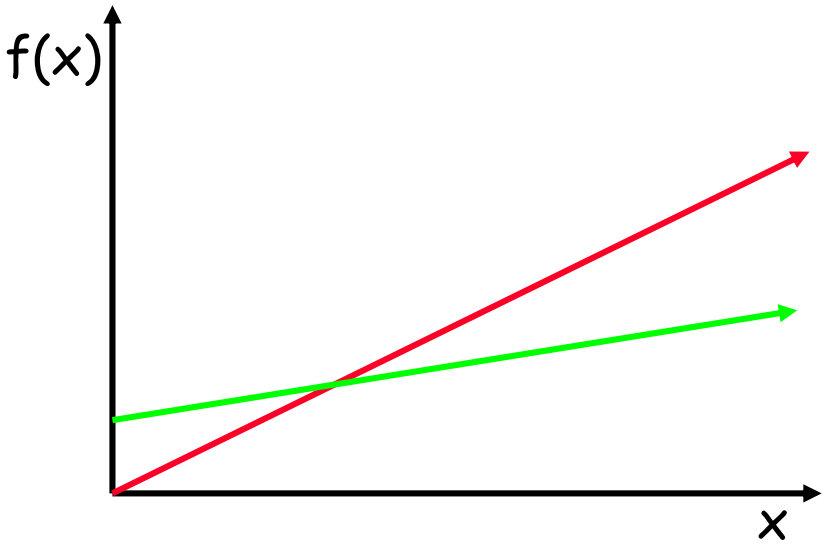
Τάξη Μεγέθους Αύξησης Συναρτήσεων (2/8)

- Περιγράψτε αυτή τη συνάρτηση:



Τάξη Μεγέθους Αύξησης Συναρτήσεων (3/8)

- Περιγράψτε αυτές τις συναρτήσεις:



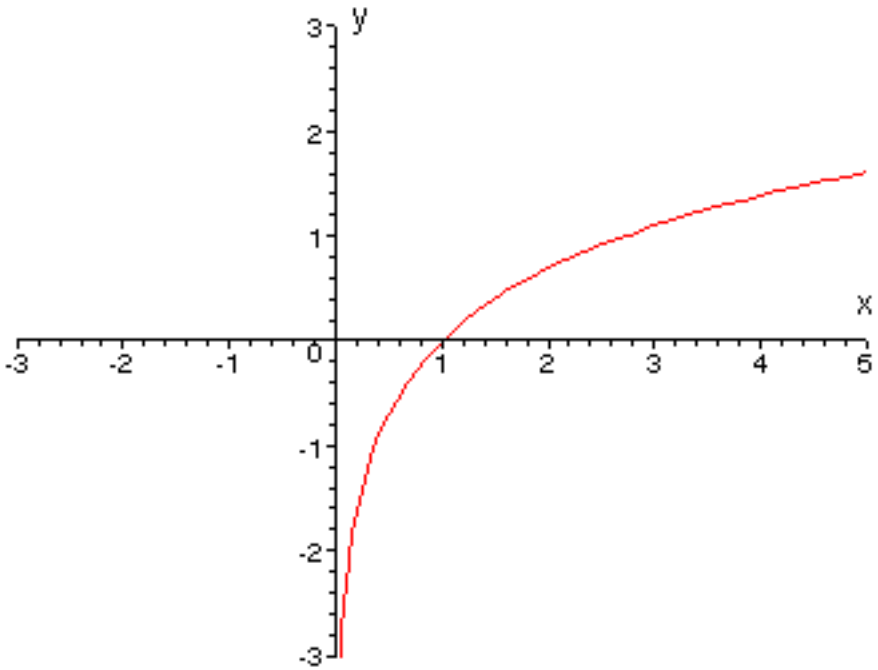
Γραμμικές συναρτήσεις

Κάθε μοναδιαία αύξηση στο x έχει ως αποτέλεσμα παρόμοια αύξηση στο f(x).

Π.χ. f(x)=5x

Τάξη Μεγέθους Αύξησης Συναρτήσεων (4/8)

- Περιγράψτε αυτή τη συνάρτηση:



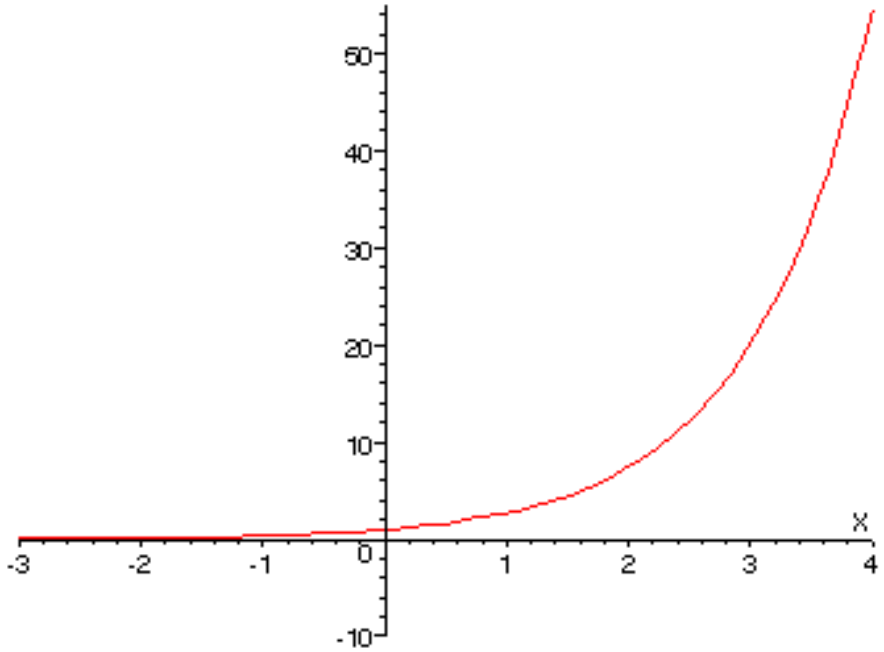
Λογαριθμική συνάρτηση

Αυξάνεται πολύ αργά

Π.χ. f(x)=5logx

Τάξη Μεγέθους Αύξησης Συναρτήσεων (5/8)

- Περιγράψτε αυτή τη συνάρτηση:



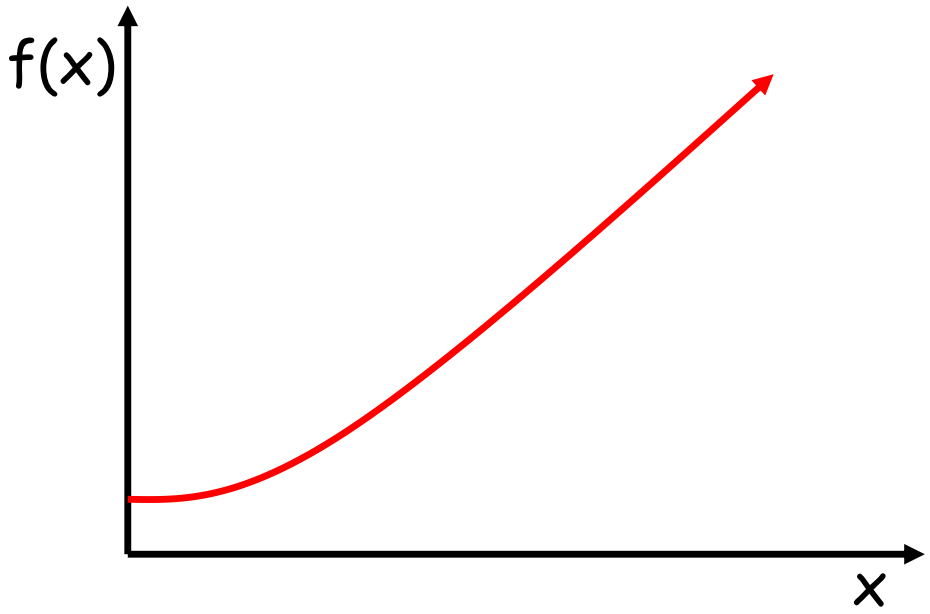
Εκθετική συνάρτηση

Αυξάνεται πολύ γρήγορα

Π.χ. $f(x)=5^x$

Τάξη Μεγέθους Αύξησης Συναρτήσεων (6/8)

- Περιγράψτε αυτή τη συνάρτηση:

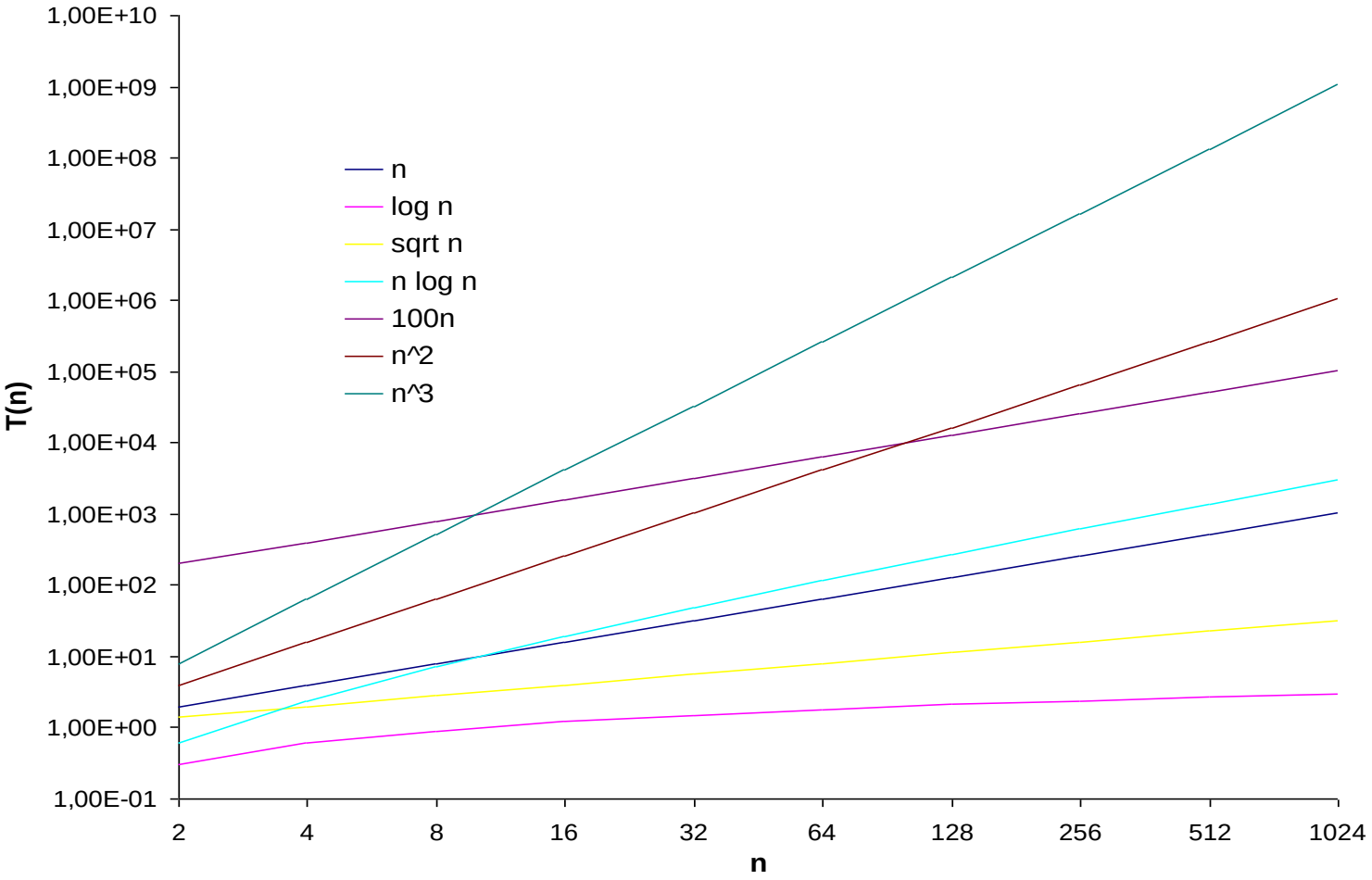


Δυωνυμική συνάρτηση

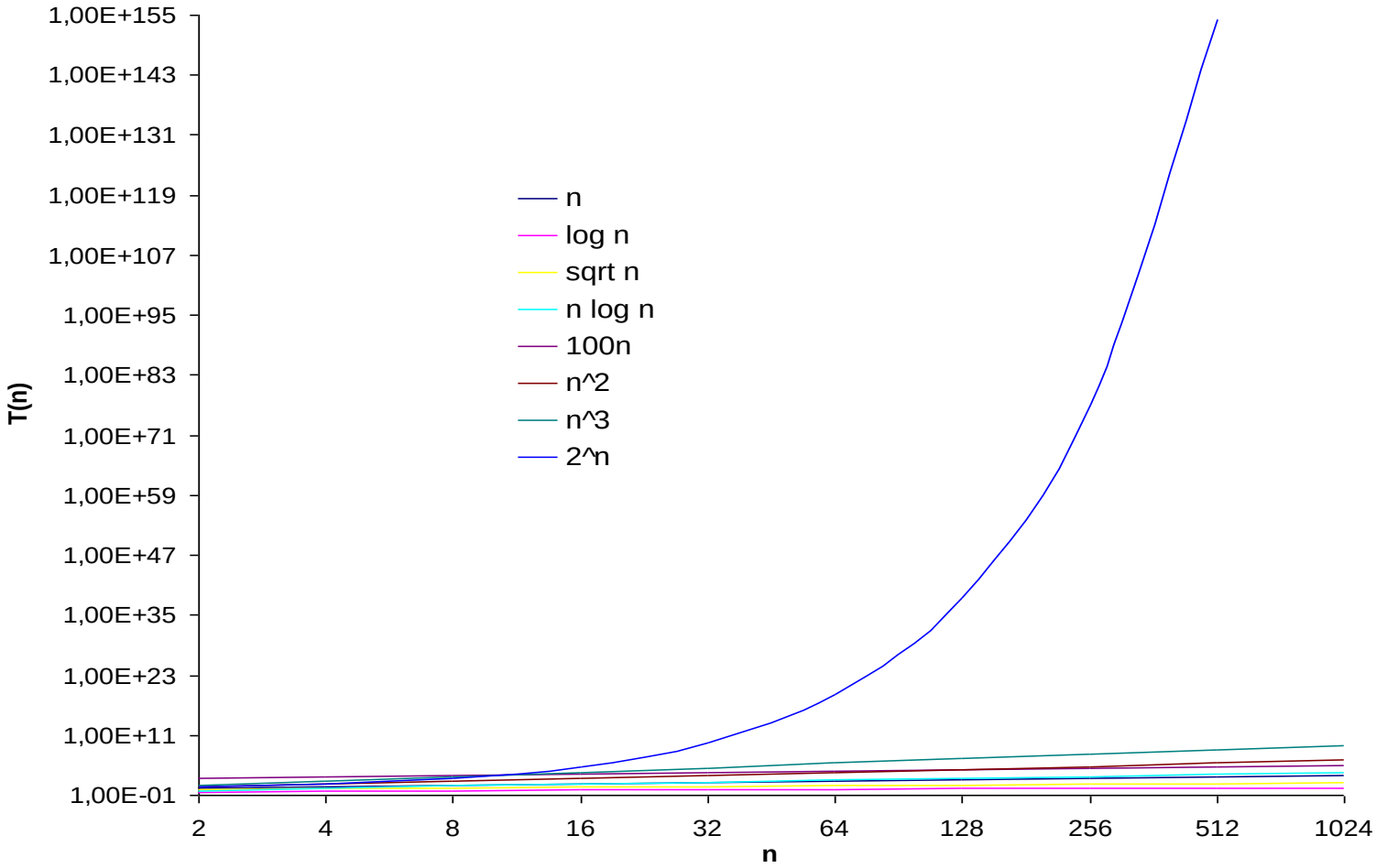
Πολυωνυμική

Π.χ. $f(x)=5x^2$

Τάξη Μεγέθους Αύξησης Συναρτήσεων (7/8)



Τάξη Μεγέθους Αύξησης Συναρτήσεων (8/8)



Ασυμπτωτική συμπεριφορά Αριθμητικών συναρτήσεων (1/5)

- Για να συγκρίνουμε την ασυμπτωτική συμπεριφορά δύο αριθμητικών συναρτήσεων εισάγουμε την έννοια της ***ασυμπτωτικής επικράτησης***.
- Έστω δυο αριθμητικές συναρτήσεις *a* και *b*. Λέμε ότι η *a* επικρατεί ασυμπτωτικά της *b* αν υπάρχουν θετικές σταθερές *k* και *m* τέτοιες ώστε:

$$|b(x)| \leq ma(x) \quad \text{για } x \geq k$$

  - π.χ. έστω $a(x) = x + 1 \quad x \geq 0$

    και $b(x) = 1/x + 7 \quad x \geq 0$
  - Η *a* επικρατεί ασυμπτωτικά της *b* γιατί για $k=7$ και $m=1$

$$|b(x)| \leq a(x) \quad \text{για } x \geq 7$$

Ασυμπτωτική συμπεριφορά Αριθμητικών συναρτήσεων (2/5)

- Έστω $a(x) = (1/100)x^2 - 1000$

και $b(x) = \begin{cases} -1.000.000 & 0 \le x \le 10 \\ -100.000x & x > 10 \end{cases}$

Η a επικρατεί ασυμπτωτικά της b γιατί για $k=10^6+1$ και $m=10$.

$$|b(x)| \le 10a(x) \qquad \text{για } x \ge 10^6+1$$

Το ότι η a επικρατεί ασυμπτωτικά της b σημαίνει ότι η a μεγαλώνει γρηγορότερα από ότι η b

Ασυμπτωτική συμπεριφορά Αριθμητικών συναρτήσεων (3/5)

- Ισχύουν τα εξής:
  - Για οποιαδήποτε αριθμητική συνάρτηση ***a***, η συνάρτηση $|a|$ επικρατεί ασυμπτωτικά της ***a***.
  - Αν η ***a*** επικρατεί ασυμπτωτικά της ***b***, τότε επικρατεί ασυμπτωτικά και της sb, όπου s είναι μια οποιαδήποτε σταθερά.
  - Αν η ***a*** επικρατεί ασυμπτωτικά της ***b*** και η ***b*** επικρατεί ασυμπτωτικά της ***c***, τότε και η ***a*** επικρατεί ασυμπτωτικά της ***c***.
  - Αν η ***a*** επικρατεί ασυμπτωτικά των ***b*** και ***c***, τότε επικρατεί ασυμπτωτικά και της $sb+tc$, όπου s και t είναι δύο οποιεσδήποτε σταθερές.

Ασυμπτωτική συμπεριφορά Αριθμητικών συναρτήσεων (4/5)

- Για κάποια δεδομένη αριθμητική συνάρτηση ***f***, συμβολίζουμε με **O(*f*)** το σύνολο όλων των αριθμητικών συναρτήσεων των οποίων η ***f*** επικρατεί ασυμπτωτικά.
  - Το **O(*f*)** διαβάζεται "*τάξη **f***" ή "*κεφαλαίο όμικρον της **f***"
- Λέμε ότι $f(n)=30n+8$ *είναι (το πολύ) τάξης n*, ή $O(n)$.
  - Είναι, το πολύ, περίπου *ανάλογη* του *n*.
- Η $f(n)=n^2+1$ είναι *τάξης* n^2, ή $O(n^2)$.
  - Είναι (το πολύ) περίπου *ανάλογη* του n^2.

Ασυμπτωτική Αύξηση Συναρτήσεων (1/2)

Σημαντικός Ορισμός:

Για τις συναρτήσεις f και g γράφουμε f(x) = O(g(x))

για να αναπαραστήσουμε ότι

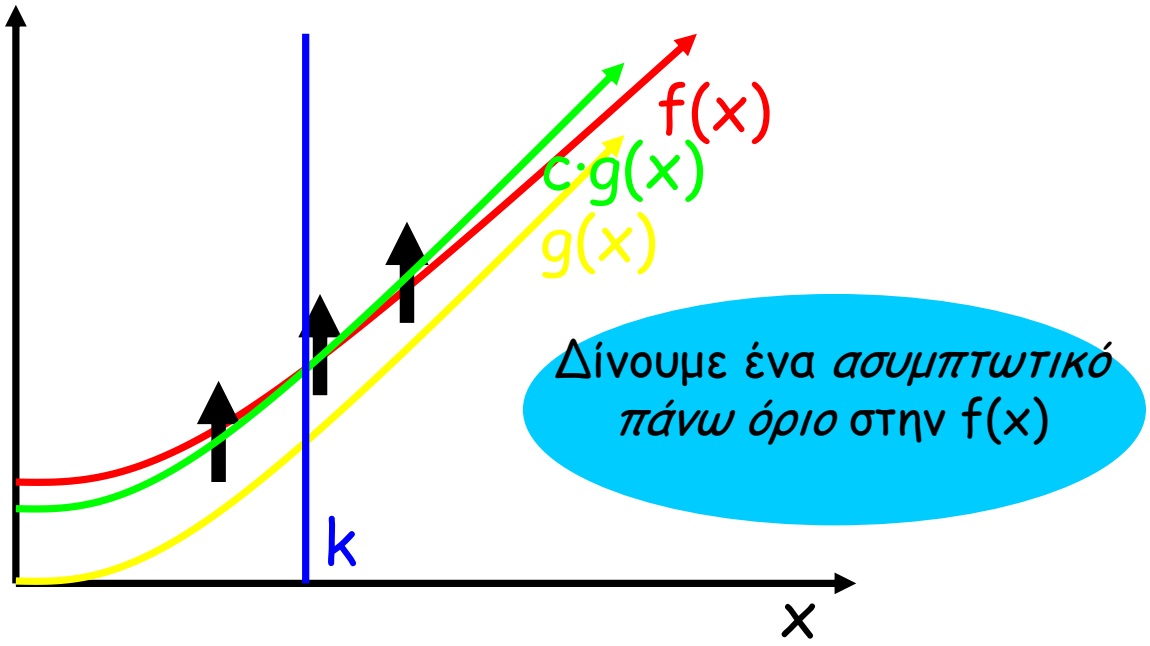
$$\exists\ c,k \text{ έτσι ώστε } \forall\ x>k,\ f(x) \leq c \cdot g(x)$$

Λέμε ότι "f(x) είναι μεγάλο O του g(x)"

Για να αποδείξουμε f(x) = O(g(x)): πρέπει να βρούμε c και k έτσι ώστε να ισχύει η ανισότητα

Ασυμπτωτική Αύξηση Συναρτήσεων (2/2)

f(x) = O(g(x))
Αν και μόνο αν
$\exists$ c,k έτσι ώστε $\forall$ x>k, f(x) $\leq$ c·g(x)



Σχετικά με τον Ορισμό

- Προσέξτε ότι η *f* είναι O(*g*) αν υπάρχουν *οποιεσδήποτε* τιμές των *c* και *k* που ικανοποιούν τον ορισμό.
- Αλλά: Οι συγκεκριμένες τιμές των *c*, *k*, που ικανοποιούν τον ορισμό *δεν* είναι μοναδικές: **Οποιαδήποτε μεγαλύτερη τιμή του *c* και/ή του *k* μας κάνει επίσης.**
- **Δεν** είναι απαραίτητο να βρείτε τις μικρότερες τιμές των *c* και *k* που δουλεύουν. (Αν και σε κάποιες περιπτώσεις μπορεί να είναι οι μικρότερες!)
- Αλλά, πρέπει να **αποδείξετε** ότι οι τιμές που επιλέξατε λειτουργούν.

Ασυμπτωτική συμπεριφορά Αριθμητικών συναρτήσεων (5/5)

- Ισχύουν τα εξής:
  - Για οποιαδήποτε αριθμητική συνάρτηση ***a***, η συνάρτηση ***a*** είναι O(|***a***|).
  - Αν η ***b*** είναι O(***a***), τότε για οποιαδήποτε σταθερά *s* η *s****b***, είναι επίσης O(***a***).
  - Αν η ***c*** είναι O(***b***) και η ***b*** είναι O(***a***) τότε και ***c*** είναι O(***a***).
  - Είναι δυνατόν η ***a*** να είναι O(***b***) και η ***b*** να είναι επίσης O(***a***).
  - Αν οι ***b*** και ***c*** είναι O(***a***), τότε για οποιεσδήποτε σταθερές *s* και *t*, η *s****c***+*t****b*** είναι επίσης O(***a***).
    - θα δούμε την απόδειξη αργότερα.

Παραδείγματα “Μεγάλου-Ο” (1/7)

f(x) = O(g(x))

Αν και μόνο αν

$\exists$ c,k έτσι ώστε $\forall$ x>k, $f(x) \leq c \cdot g(x)$

$3n = O(15n)$ μια και $\forall$ $n>0$, $3n \leq 1 \cdot 15n$

υπάρχει k

υπάρχει c

Παραδείγματα “Μεγάλου-Ο” (2/7)

f(x) = O(g(x))

Αν και μόνο αν

$\exists$ c,k έτσι ώστε $\forall$ x>k, $f(x) \leq c \cdot g(x)$

$15n = O(3n)$ μια και $\forall$ $n > \_0\_$, $15n \leq \_5\_ \cdot 3n$

Παραδείγματα “Μεγάλου-Ο” (3/7)

f(x) = O(g(x))

Αν και μόνο αν

$\exists$ c,k έτσι ώστε $\forall$ x>k, $f(x) \leq c \cdot g(x)$

$x^2 = O(x^3)$ μια και $\forall$ x> 1, $x^2 \leq x^3$

Παραδείγματα “Μεγάλου-Ο” (4/7)

f(x) = O(g(x))

Αν και μόνο αν

$\exists$ c,k έτσι ώστε $\forall$ x>k, $f(x) \leq c \cdot g(x)$

$1000x^2 = O(x^2)$ μια και $\forall$ x> 0, $1000x^2 \leq 1000 \cdot x^2$

Παραδείγματα “Μεγάλου-Ο” (5/7)

- Δείξτε ότι το $30n+8$ είναι $O(n)$.
  - Δείξτε $\exists c,k: \forall n>k: 30n+8 \leq cn$.
    - Ας θέσουμε $c=31$, $k=8$. Υποθέτουμε ότι $n>k=8$. Τότε $cn = 31n = 30n + n > 30n+8$, οπότε $30n+8 < cn$.

- Δείξτε ότι το n^2+1 είναι $O(n^2)$.
  - Δείξτε $\exists c,k: \forall n>k: n^2+1 \leq cn^2$.
    - Ας θέσουμε $c=2$, $k=1$. Υποθέτουμε ότι $n>1$. Τότε $cn^2 = 2n^2 = n^2+n^2 > n^2+1$, ή $n^2+1< cn^2$.

Παραδείγματα “Μεγάλου-Ο” (6/7)

f(x) = O(g(x))

Αν και μόνο αν

$\exists$ c,k έτσι ώστε $\forall$ x>k, $f(x) \leq c \cdot g(x)$

Αποδείξτε ότι $x^2 + 100x + 100 = O((1/100)x^2)$

$x^2 + 100x + 100 \leq 201x^2$ όταν $x > 1$

$\leq 20100 \cdot (1/100)x^2$

$100x \leq 100x^2$

$100 \leq 100x^2$

k = 1,
c = 20100

Παραδείγματα "Μεγάλου-Ο" (7/7)

Αποδείξτε ότι $5x + 100 = O(x/2)$

Πρέπει $\forall$ x> ____, $5x + 100 \leq$ ____ $\cdot x/2$

Δοκιμάστε c = 10

$\forall$ x> ____, $5x + 100 \leq 10 \cdot x/2$

Δε γίνεται τίποτα με το k

Δοκιμάστε c = 11

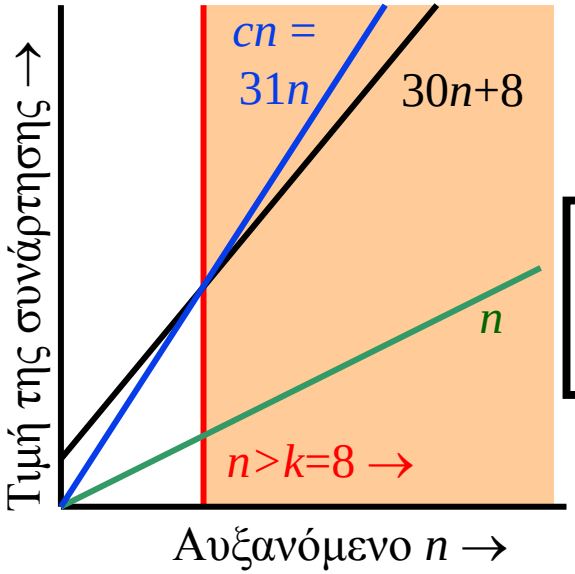
$\forall$ x> ____, $5x + 100 \leq 5x + x/2$

$\forall$ x> 200, $100 \leq x/2$

k = 200, c = 11

Παράδειγμα “Μεγάλου-Ο”, γραφικά

- Παρατηρήστε ότι το 30*n*+8 δεν είναι μικρότερο του *n πουθενά* (*n*>0).
- Δεν είναι καν μικρότερο του 31*n παντού*.
- Αλλά *είναι* μικρότερο του 31*n* **παντού στα δεξιά του *n*=8.**



$30n+8 \in O(n)$

Χρήσιμα Στοιχεία σχετικά με το“Μεγάλο-Ο”

- $\forall c>0$, $O(cf)=O(f+c)=O(f-c)=O(f)$
- $f_1 \in O(g_1) \wedge f_2 \in O(g_2) \rightarrow$
  - $f_1 f_2 \in O(g_1 g_2)$
  - $f_1+f_2 \in O(g_1+g_2)$

    $= O(\max(g_1,g_2))$

    $= O(g_1)$ if $g_2 \in O(g_1)$ (Πολύ χρήσιμο!)

Ασυμπτωτική αύξηση συναρτήσεων (1/3)

- Αν f1(x) = O(g1(x)) και f2(x)=O(g2(x)), τότε f1(x) + f2(x) = O(max{g1(x),g2(x)})

- **Απόδειξη:** Έστω h(x) = max{$g_1(x)$,$g_2(x)$})…
  - Πρέπει να βρούμε σταθερές c και k έτσι ώστε $\forall x>k$, $f_1(x) + f_2(x) \leq c \bullet h(x)$.
  - Ξέρουμε ότι $f_1(x) \leq c_1 \bullet g_1(x)$ και $f_2(x) \leq c_2 \bullet g_2(x)$
  - Έτσι, $f_1(x) + f_2(x) \leq c_1 \bullet g_1(x) + c_2 \bullet g_2(x) \leq c_1 \bullet h(x) + c_2 \bullet h(x)$
    $= (c_1+c_2) \bullet h(x)$

$c = c_1+c_2$,
$k = \max\{k_1,k_2\}$

Ασυμπτωτική αύξηση συναρτήσεων (2/3)

- Ξέρουμε ότι: Αν f1(x) = O(g1(x)) και f2(x)=O(g2(x)), τότε f1(x) + f2(x) = O(max{g1(x),g2(x)}).
- Συνεπώς: Αν $f_1(x) = O(g(x))$ και $f_2(x)=O(g(x))$, τότε $f_1(x) + f_2(x) = O(g(x))$.

Απλά η περίπτωση όπου $g_1(x) = g_2(x) = g(x)$

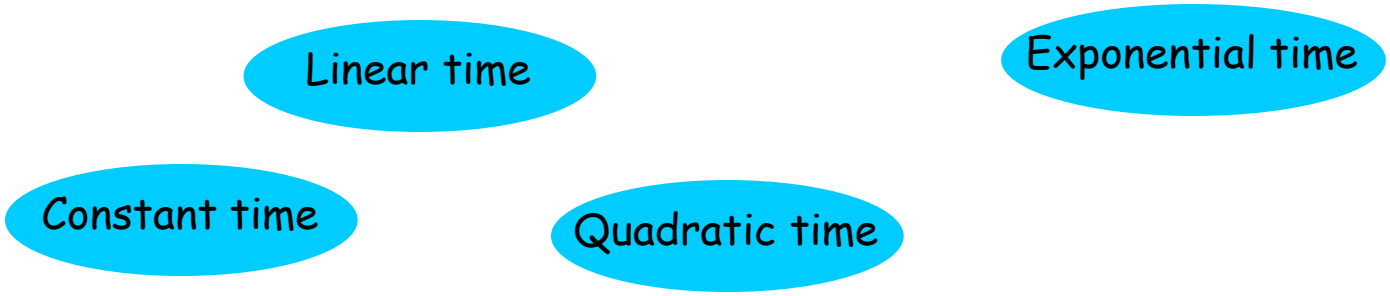
Εκφράσεις με “Μεγάλο-Ο”

- Όταν το “$O(f)$” χρησιμοποιείται ως όρος σε μια αριθμητική έκφραση, σημαίνει: “κάποια συνάρτηση f τέτοια ώστε $f \in O(f)$”.
- Π.χ.: “$x^2+O(x)$” σημαίνει “x^2 συν κάποια συνάρτηση που είναι $O(x)$”.
- Κάθε τέτοια έκφραση ουσιαστικά αναπαριστά ένα **σύνολο** από συναρτήσεις:

$$x^2+O(x) :\equiv \{g \mid \exists f \in O(x): g(x) = x^2+f(x)\}$$

Ασυμπτωτική αύξηση συναρτήσεων (3/3)

- Κανόνες υπολογισμού του O:
- Γενικά, μόνο ο μεγαλύτερος όρος μετράει σε ένα άθροισμα.
  - $a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x^1 + a_nx^0 = O(x^n)$
- Το n είναι ισχυρότερο του log n.
  - $n^5 \log n = O(n^6)$
- Κοινές συναρτήσεις σε αυξανόμενη τάξη του O():
  - $1 \quad n \quad (n \lg n) \quad n^2 \quad n^3 \quad \ldots \quad 2^n \quad n!$



Παραδείγματα Ασυμπτωτικής Αύξησης Συναρτήσεων (1/2)

$$n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 \leq n \cdot n \cdot \ldots \cdot n \cdot n \cdot n = n^n$$

$$n! = O(n^n)$$

Ορισμός: $\Theta(g)$, *ακριβώς τάξης g*

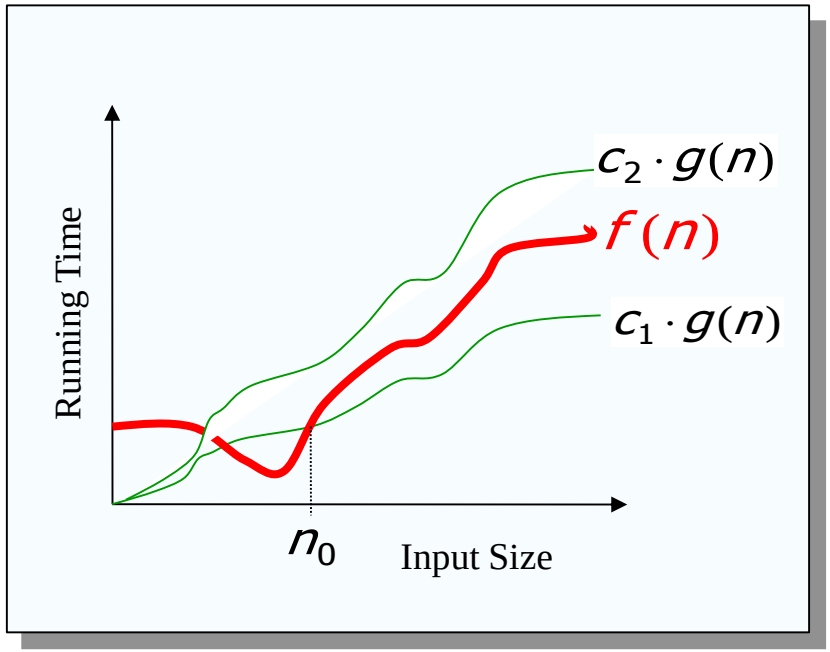
- Αν $f \in O(g)$ και $g \in O(f)$, τότε λέμε πως "οι *g και f είναι της ίδιας τάξης*" ή "η *f είναι (ακριβώς) τάξης g*" και γράφουμε $f \in \Theta(g)$.
- Άλλος, ισοδύναμος ορισμός:

$$\Theta(g) \equiv \{f: \mathbf{R} \to \mathbf{R} \mid \exists c_1 c_2 k > 0 \; \forall x > k: |c_1 g(x)| \leq |f(x)| \leq |c_2 g(x)| \}$$

  - "Παντού πέρα από ένα σημείο *k*, η *f*(*x*) βρίσκεται μεταξύ δύο πολλαπλασίων της *g*(*x*)."

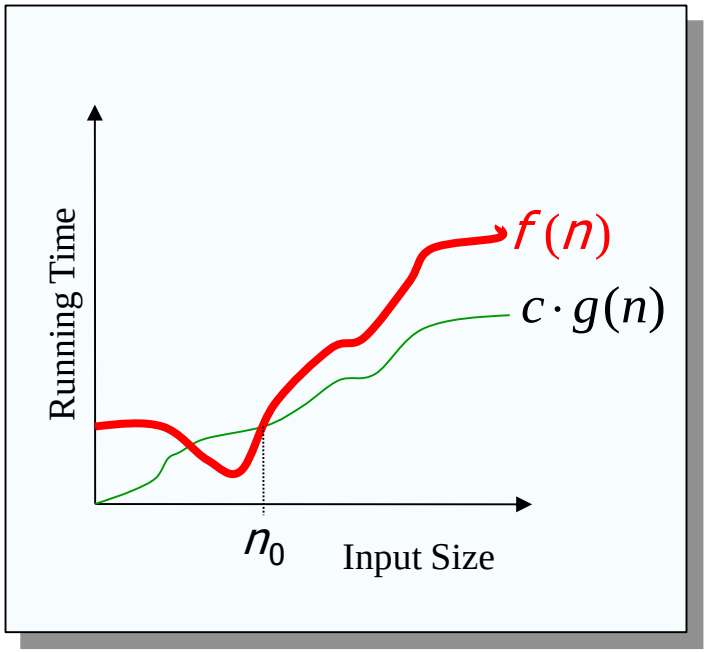
Συμβολισμός Θ

- Ο συμβολισμός Θ (μεγάλο θήτα) δίνει ένα αυστηρό ασυμπτωτικό όριο:
  - $f(n) = \Theta(g(n))$ αν υπάρχουν σταθερές c_1, c_2, και n_0, *ε.ω.* $\mathbf{c_1\, g(n) \leq f(n) \leq c_2\, g(n)}$ for $n \geq n_0$.



Συμβολισμός Ω

- Ο συμβολισμός Ω (μεγάλο ωμέγα) δίνει ένα ασυμπτωτικό κάτω όριο
  - *f(n) = Ω(g(n))* αν υπάρχουν σταθερές *c* and n_0, *ε.ω.* **c g(n) ≤ f(n)** for $n \geq n_0$.
- Χρησιμοποιείται για περιγραφή χρόνων *καλύτερης περίπτωσης* (*best-case*).



Κανόνες για το Θ

- Παρόμοιοι με τους κανόνες για το O( ), εκτός από:
- $\forall$ $f,g>0$ & σταθερές $a,b \in \mathbf{R}$, με $b>0$,

$af \in \Theta(f)$, αλλά ← Όπως με το O.

$f \notin \Theta(fg)$ εκτός αν $g=\Theta(1)$ ← Σε αντίθεση με το O.

$|f|^{1-b} \notin \Theta(f)$, και ← Σε αντίθεση με το O.

$(\log_b |f|)^c \notin \Theta(f)$. ← Σε αντίθεση με το O.

- Οι συναρτήσεις στις δύο τελευταίες περιπτώσεις είναι *αυστηρά χαμηλότερης τάξης* από ότι η $\Theta(f)$.

Παράδειγμα Θ

- Προσδιορίστε αν: $\left(\sum_{i=1}^{n} i\right) \stackrel{?}{\in} \Theta(n^2)$
- Γρήγορη λύση:

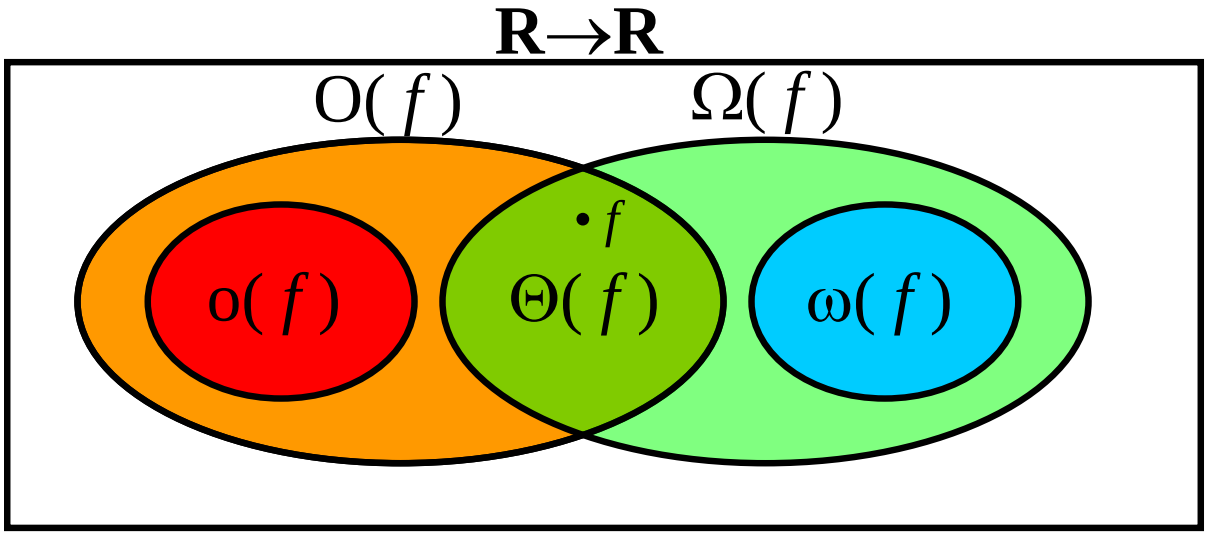
$$\left(\sum_{i=1}^{n} i\right) = n(n-1)/2$$
$$= n\,\Theta(n)/2$$
$$= n\,\Theta(n)$$
$$= \Theta(n^2)$$

Συσχετίσεις Αύξησης Συναρτήσεων

- $\Omega(g) = \{f \mid g \in O(f)\}$
  "Οι συναρτήσεις που είναι *τουλάχιστον τάξης g*."
- $o(g) = \{f \mid \forall c>0\ \exists k\ \forall x>k : |f(x)| < |cg(x)|\}$
  "Οι συναρτήσεις που είναι *αυστηρά χαμηλότερης τάξης από τη g*." $o(g) \subset O(g) - \Theta(g)$.
- $\omega(g) = \{f \mid \forall c>0\ \exists k\ \forall x>k : |cg(x)| < |f(x)|\}$
  "Οι συναρτήσεις που είναι *αυστηρά υψηλότερης τάξης από τη g*." $\omega(g) \subset \Omega(g) - \Theta(g)$.

Συσχετίσεις

- Σχέσεις υποσυνόλων ανάμεσα στα σύνολα τάξεων αύξησης συναρτήσεων.



Παράδειγμα 1: Ο αλγόριθμος Max (1/3)

procedure *max*(a_1, a_2, …, a_n: integers)

$v := a_1$	t_1
for $i := 2$ **to** n	t_2
if $a_i > v$ **then** $v := a_i$	t_3
return v	t_4

Φορές εκτέλεσης *κάθε γραμμής*

Ας υπολογίζουμε την πολυπλοκότητα

Παράδειγμα 1: Ο αλγόριθμος Max (2/3)

procedure $max(a_1, a_2, \ldots, a_n$: integers)

$v := a_1$	t_1
for $i := 2$ **to** n	t_2
if $a_i > v$ **then** $v := a_i$	t_3
return v	t_4

Φορές εκτέλεσης *κάθε γραμμής*

$$t(n) = t_1 + \left(\sum_{i=2}^{n} (t_2 + t_3) \right) + t_4$$

Παράδειγμα 1: Ο αλγόριθμος Max (3/3)

$$t(n) = t_1 + \left(\sum_{i=2}^{n} (t_2 + t_3) \right) + t_4$$

$$= \Theta(1) + \left(\sum_{i=2}^{n} \Theta(1) \right) + \Theta(1) = \Theta(1) + (n-1)\Theta(1)$$

$$= \Theta(1) + \Theta(n)\Theta(1) = \Theta(1) + \Theta(n) = \Theta(n)$$

Παράδειγμα 2: Γραμμική Αναζήτηση (1/2)

INPUT

- μια ακολουθία αριθμών (database)
- ένας συγκεκριμένος αριθμός (query)

$a_1, a_2, a_3, \ldots, a_n;\ q$

2 5 4 10 7; 5

2 5 4 10 7; 9

OUTPUT

- η θέση του αριθμού στην ακολουθία ή *NULL*

j

2

NULL

Παράδειγμα 2: Γραμμική Αναζήτηση (2/2)

- **procedure** *linear search* (x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)

$i := 1$	t_1
while $(i \leq n \wedge x \neq a_i)$	t_2
$\quad i := i + 1$	t_3
if $i \leq n$ **then** *location* $:= i$	t_4
else *location* $:= 0$	t_5
return *location*	t_6

Ανάλυση Γραμμικής Αναζήτησης

- Πολυπλοκότητα χειρότερης περίπτωσης:

$$t(n) = t_1 + \left(\sum_{i=1}^{n} (t_2 + t_3) \right) + t_4 + t_5 + t_6 = \Theta(n)$$

- Καλύτερη περίπτωση:

$$t(n) = t_1 + t_2 + t_4 + t_6 = \Theta(1)$$

- Μέση περίπτωση, αν το αντικείμενο υπάρχει στη λίστα: $t(n) = t_1 + \left(\sum_{i=1}^{n/2} (t_2 + t_3) \right) + t_4 + t_5 + t_6 = \Theta(n)$

Παράδειγμα 3: Δυαδική Αναζήτηση (1/3)

INPUT

• μια ταξινομημένη ακολουθία αριθμών σε αύξουσα σειρά
• ένας συγκεκριμένος αριθμός

$a_1, a_2, a_3, \ldots, a_n;\ q$

2 4 5 7 10; 5

2 4 5 7 10; 9

OUTPUT

• η θέση του αριθμού στην ακολουθία ή NULL

j

2

NULL

Παράδειγμα 3: Δυαδική Αναζήτηση (2/3)

- **procedure** *binary search* (x: integer, $a_1, a_2, \ldots, a_n$: distinct integers, sorted smallest to largest)

  $i := 1$

  $j := n$ } $\Theta(1)$

  while $i<j$ **begin**

  $m := \lfloor (i+j)/2 \rfloor$

  if $x > a_m$ **then** $i := m+1$ **else** $j := m$ } $\Theta(1)$

  end

  if $x = a_i$ **then** *location* $:= i$ **else** *location* $:= 0$ } $\Theta(1)$

  return *location*

Βασική Ερώτηση:

Πόσες επαναλήψεις του loop?

Παράδειγμα 3: Δυαδική Αναζήτηση (3/3)

Υποθέστε ότι το *n* είναι δύναμη του 2, *π.χ*, $\exists k: n=2^k$.
Στο διάστημα από $i=1$ ως $j=n$ περιέχονται *n* αντικείμενα.
Σε κάθε επανάληψη: Το μέγεθος $j-i+1$ του διαστήματος κόβεται περίπου στο μισό.
Ο βρόγχος σταματάει όταν $1=2^0$ ($i=j$).
Οπότε το πλήθος των επαναλήψεων είναι:
$k = \log_2 n = \Theta(\log_2 n) = \Theta(\log n)$

Ακόμα και αν $n \neq 2^k$ (δεν είναι ακριβώς δύναμη του 2), η χρονική πολυπλοκότητα παραμένει $\Theta(\log_2 n) = \Theta(\log n)$.

Χωρική Πολυπλοκότητα

- Η χωρική πολυπλοκότητα προσδιορίζεται επίσης ως συνάρτηση του μεγέθους των δεδομένων (input).
- **Ορισμός:** Η **χωρική πολυπλοκότητα χειρότερης περίπτωσης** (worst-case space complexity) ενός αλγόριθμου είναι η συνάρτηση S(*n*), η οποία είναι η μέγιστη, για όλα τα inputs μεγέθους *n*, των αθροισμάτων χώρου μνήμης από κάθε **βασική πράξη** (primitive operation).
- Αν είναι εκθετική υπάρχει σοβαρό πρόβλημα!

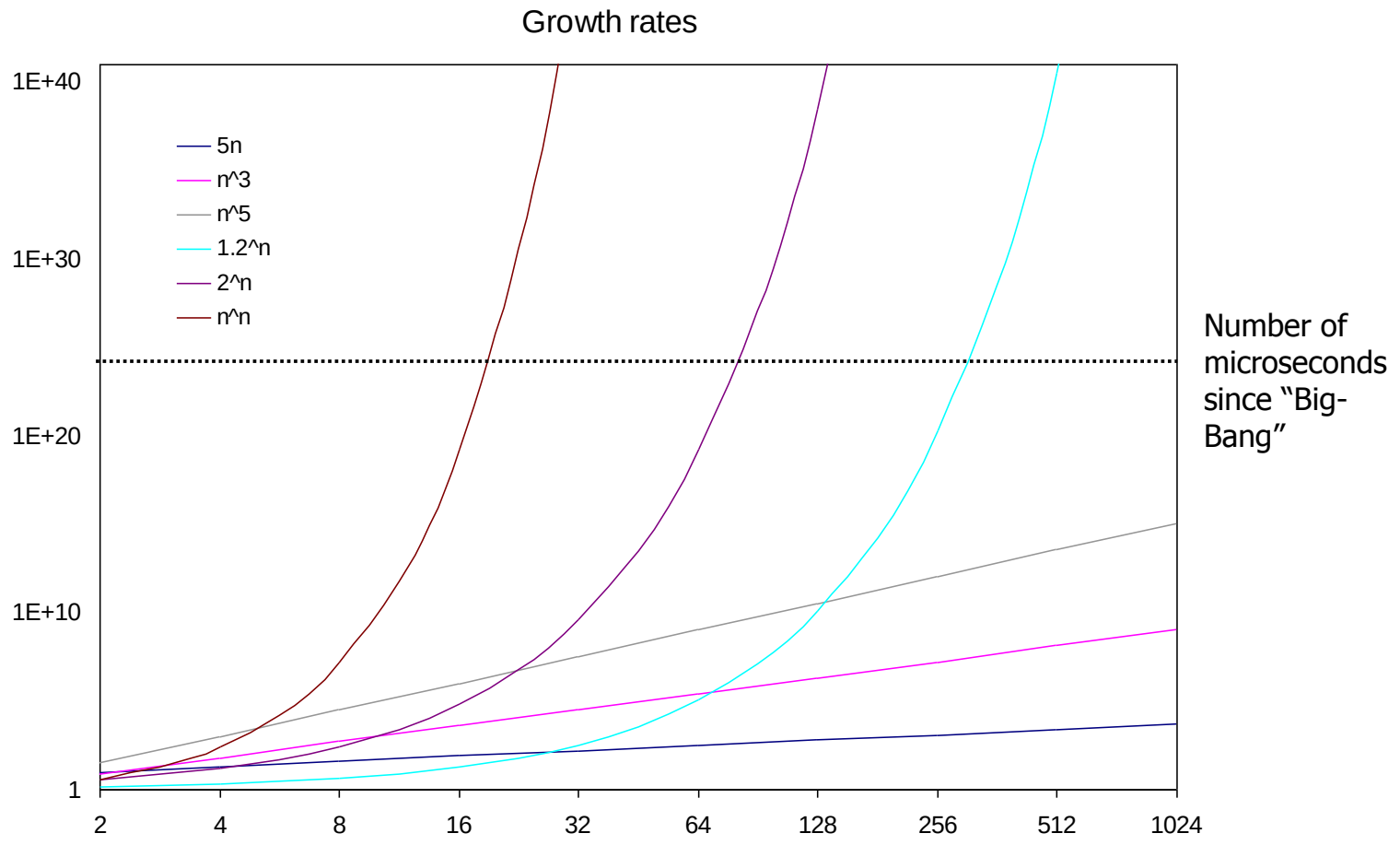
Πολυπλοκότητα προβλημάτων

- Η πολυπλοκότητα ενός *υπολογιστικού προβλήματος* είναι (η τάξη αύξησης) της πολυπλοκότητας του αλγόριθμου με την χαμηλότερη τάξη αύξησης πολυπλοκότητας για την επίλυση του προβλήματος.
  - ΠΡΟΣΟΧΗ: Δεν αναφερόμαστε στον “καλύτερο” γνωστό αλγόριθμο, αλλά στον “καλύτερο” που μπορεί να υπάρξει!
- *Π.χ.* Το πρόβλημα αναζήτησης σε μια διατεταγμένη λίστα έχει *το πολύ λογαριθμική* χρονική πολυπλοκότητα. (**O(log *n*)**.)

Πρακτικώς επιλύσιμα και δυσεπίλυτα προβλήματα

- Ένα πρόβλημα με το πολύ πολυωνυμική χρονική πολυπλοκότητα θεωρείται *πρακτικά επιλύσιμο*. Το **P** είναι το σύνολο όλων των πρακτικά επιλύσιμων προβλημάτων.
- Ένα πρόβλημα με πολυπλοκότητα μεγαλύτερη της πολυωνυμικής θεωρείται *δυσεπίλυτο.*
- Προσέξτε ότι $n^{1,000,000}$ είναι *τεχνικά* επιλύσιμο, αλλά στην πραγματικότητα πολύ δύσκολο. $n^{\log \log \log n}$ είναι *τεχνικά* δυσεπίλυτο, αλλά εύκολο.
  - Τέτοιες περιπτώσεις είναι σπάνιες πάντως.

Συναρτήσεις αύξησης κόστους



Μη επιλύσιμα προβλήματα (intractable problems)

	function/ n	10	20	50	100	300
Polynomial	n^2	1/10,000 second	1/2,500 second	1/400 second	1/100 second	9/100 second
Polynomial	n^5	1/10 second	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
Exponential	2^n	1/1000 second	1 second	35.7 years	400 trillion centuries	a 75 digit-number of centuries
Exponential	n^n	2.8 hours	3.3 trillion years	a 70 digit-number of centuries	a 185 digit-number of centuries	a 728 digit-number of centuries

Παραδείγματα Χρόνων Εκτέλεσης

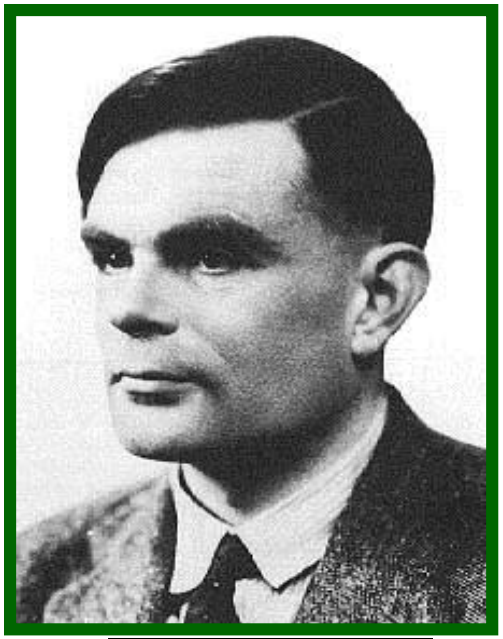
	(1.25 bytes)	(125 kB)
#ops(n)	***n=10***	***n=10^6***
$\log_2 n$	3.3 ns	19.9 ns
n	10 ns	1 ms
$n \log_2 n$	33 ns	19.9 ms
n^2	100 ns	16 m 40 s
2^n	1.024 μs	$10^{301,004.5}$ Χιλιετίες
$n!$	3.63 ms	Ωχ!

- Υποθέστε ότι απαιτείται 1 ns (10^{-9} δευτερόλεπτα) για κάθε βασική διεργασία (#op), το πρόβλημα έχει μέγεθος *n* bits, και το πλήθος #ops είναι συνάρτηση του *n*, όπως φαίνεται.

Μη-επιλύσιμα Προβλήματα

- Ο Turing ανακάλυψε στις αρχές της δεκαετία του 1930 ότι υπάρχουν προβλήματα που δεν μπορούν να επιλυθούν από *οποιονδήποτε* αλγόριθμο.
  - Ή ισοδύναμα, υπάρχουν ερωτήσεις ναι/όχι που δε μπορούν να απαντηθούν, και συναρτήσεις που δε μπορούν να υπολογιστούν.
- Κλασσικό Παράδειγμα: το *πρόβλημα τερματισμού (halting problem)*.
  - Δοθέντος ενός οποιουδήποτε αλγόριθμου και κάποιων δεδομένων εισόδου, θα τερματίσει κάποτε ο αλγόριθμος, ή θα συνεχίσει να τρέχει για πάντα σε ένα *"ατέρμονο loop;"*

[Πηγή Εικόνας:
http: //www.singularityweblog.com/17-definitions-of-the-technological-singularity/]



Alan Turing
1912-1954

P και NP

- Το **NP** είναι το σύνολο των προβλημάτων για τα οποία υπάρχει ένας πρακτικά υπολογίσιμος αλγόριθμος που μπορεί *να ελέγξει αν μια προτεινόμενη λύση* είναι σωστή.
- Ξέρουμε ότι **P**$\subseteq$**NP**, αλλά η πιο διάσημη υπόθεση που δεν έχει αποδειχθεί στην επιστήμη υπολογιστών είναι ότι η σχέση αυτή είναι *κανονική*.
  - *δηλ.*, ότι **P**$\subset$**NP** κι όχι **P**=**NP**.
- Όποιος το αποδείξει αυτό θα γίνει διάσημος!

(ή αποδείξει ότι δεν ισχύει!)

Και τι έγινε;

- Οι Η/Υ γίνονται γρηγορότεροι μέρα με τη μέρα:
  - αυτό είναι ασήμαντο (μια σταθερή ποσότητα) σε σύγκριση με τον εκθετικό χρόνο εκτέλεσης.
- Υπάρχει μια μεγάλη κατηγορία προβλημάτων που ονομάζονται (NP- complete) προβλήματα (περίπου 1000 είναι).
- Για κανένα δεν υπάρχει πολυωνυμικός αλγόριθμος.
- Δεν έχει αποδειχθεί όμως ότι δε μπορεί να υπάρξει πολυωνυμικός αλγόριθμος…
  - Περισσότερα στο επόμενο 6μηνο...

Βασικά πράγματα που πρέπει να ξέρετε

- Ορισμοί αλγοριθμικής πολυπλοκότητας, χρονικής πολυπλοκότητας, χρονικής πολυπλοκότητας χειρότερης περίπτωσης.
- Τάξεις αύξησης πολυπλοκότητας.
- Πως να αναλύετε τη χειρότερη περίπτωση τάξης αύξησης της χρονικής πολυπλοκότητας για απλούς αλγόριθμους.

Τέλος Ενότητας

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημείωμα Αναφοράς

- Copyright Πανεπιστήμιο Δυτικής Μακεδονίας, Τμήμα Μηχανικών Πληροφορικής και Τηλεπικοινωνιών, Στεργίου Κωνσταντίνος. «Διακριτά Μαθηματικά». Έκδοση: 1.0. Κοζάνη 2015. Διαθέσιμο από τη δικτυακή διεύθυνση:

  https: //eclass.uowm.gr/courses/ICTE257/

Σημείωμα Αδειοδότησης

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Εικόνες/Σχήματα/Διαγράμματα/Φωτογραφίες

- *http: //www.singularityweblog.com/17-definitions-of-the-technological-singularity/*