Οι διαφάνειες βασίζονται σε αυτές του ακόλουθου μαθήματος:

Introduction to Algorithms (6-046J), MIT

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/

# *Αλγόριθμοι*
# *4ο Εξάμηνο*

# Ανάλυση Αλγορίθμων

*Η Θεωρητική μελέτη της απόδοσης των προγραμμάτων υπολογιστή και της χρήσης πόρων*

# Ασυμπτωτική Απόδοση

Ενδιαφερόμαστε για την ασυμπτωτική απόδοση των προγραμμάτων

Πως, ο αλγόριθμος συμπεριφέρεται καθώς το μέγεθος του προβλήματος γίνεται πολύ μεγάλο;

- Ο χρόνος εκτέλεσης ,
- Οι απαιτήσεις σε χώρο μνήμης ,
- Απαιτήσεις σε εύρος ζώνης, κατανάλωση ενέργειας, πλήθος λογικών πυλών ,

# Ασυμπτωτικός Συμβολισμός

Ασυμπτωτικός συμβολισμός (big-O):

- Τι σημαίνει χρόνος εκτέλεσης *O(n)*, *O($n^2$)*, *O(n lg n);*
- Πως ο ασυμπτωτικός χρόνος εκτέλεσης σχετίζεται με τη ασυμπτωτική χρήση μνήμης;

# Ανάλυση Αλγορίθμων

- Η ανάλυση γίνεται με βάση ένα υπολογιστικό μοντέλο.
- Θα χρησιμοποιήσουμε ένα μονό-επεξεργαστή γενικού σκοπού τυχαίας πρόσβασης (RAM=Random Access Machine)
  - Όλες οι προσβάσεις στη μνήμη κοστίζουν το ίδιο.
  - Δεν εκτελούνται παράλληλα λειτουργίες
  - Όλες οι συνήθεις εντολές απαιτούν σταθερό χρόνο εκτέλεσης με εξαίρεση τις κλήσεις συναρτήσεων
  - Σταθερό μήκος λέξης εκτός αν χειριζόμαστε κατευθείαν δυαδικά ψηφία

# Μέγεθος Εισόδου

Πολυπλοκότητα Χρόνου και Χώρου:

- Γενικά, συνάρτηση του μεγέθους εισόδου, π.χ., ταξινόμηση, πολλαπλασιασμός
- Το μέγεθος της εισόδου εξαρτάται από το συγκεκριμένο υπολογιστικό πρόβλημα:
  - Ταξινόμηση: πλήθος των στοιχείων εισόδου
  - Πολλαπλασιασμός: συνολικό πλήθος των δυαδικών ψηφίων
  - Αλγόριθμοι γραφημάτων: πλήθος των κόμβων και ακμών
  - κτλ.

# Χρόνος Εκτέλεσης

- Πλήθος των βασικών βημάτων που εκτελούνται
- Με εξαίρεση την εκτέλεση συναρτήσεων, οι περισσότερες εντολές απαιτούν περίπου τον ίδιο χρόνο:

  y = m * x + b
  c = 5 / 9 * (t - 32 )
  z = f(x) + g(y)

- Η ανάλυση μπορεί να γίνει πιο ακριβής αν χρειαστεί.

# Το πρόβλημα της ταξινόμησης

***Είσοδος:*** ακολουθία $\langle a_1, a_2, \ldots, a_n \rangle$ αριθμών.

***Έξοδος:*** μετάθεση $\langle a'_1, a'_2, \ldots, a'_n \rangle$ τέτοια ώστε $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

**Παράδειγμα:**

***Είσοδος:*** 8 2 4 9 3 6

***Έξοδος:*** 2 3 4 6 8 9

# Ταξινόμηση Ένθεσης

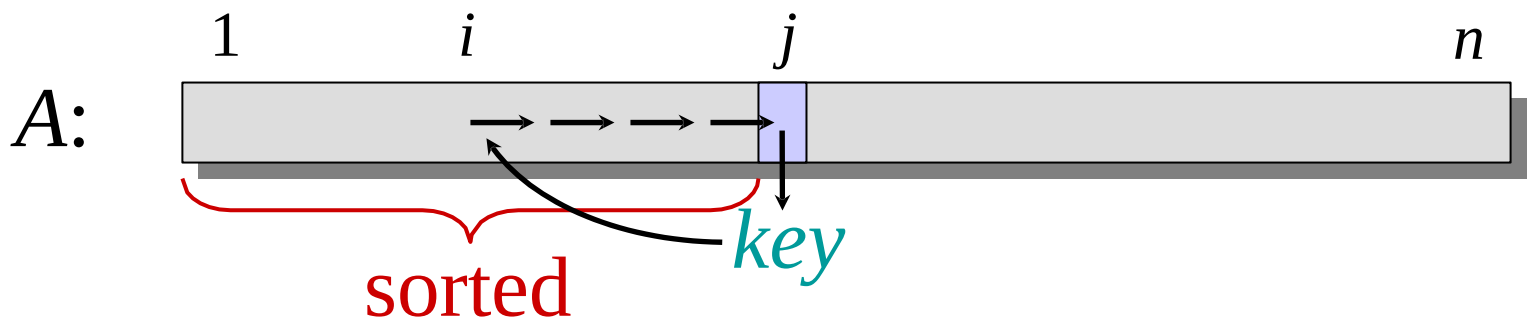
"Ψευδοκώδικας"

```
INSERTION-SORT (A, n)        ▷ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```

# Ταξινόμηση Ένθεσης

"Ψευδοκώδικας"

```
INSERTION-SORT (A, n)        ▷ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```

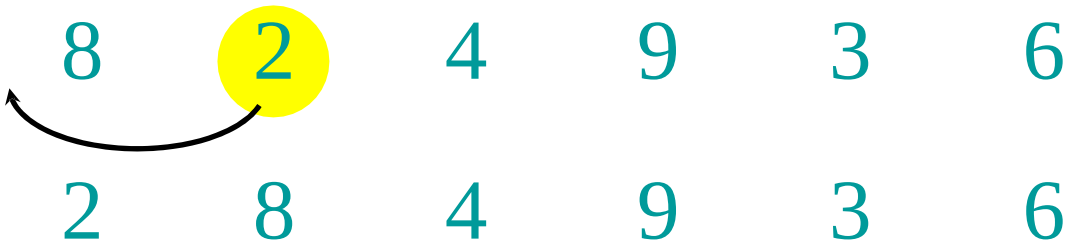


# Παράδειγμα της ταξινόμησης ένθεσης

8 2 4 9 3 6

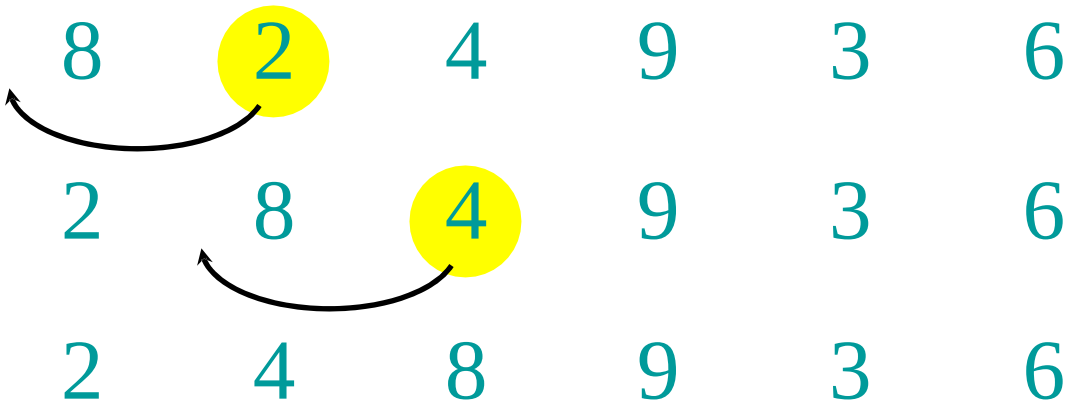
# Παράδειγμα της ταξινόμησης ένθεσης

# Παράδειγμα της ταξινόμησης ένθεσης

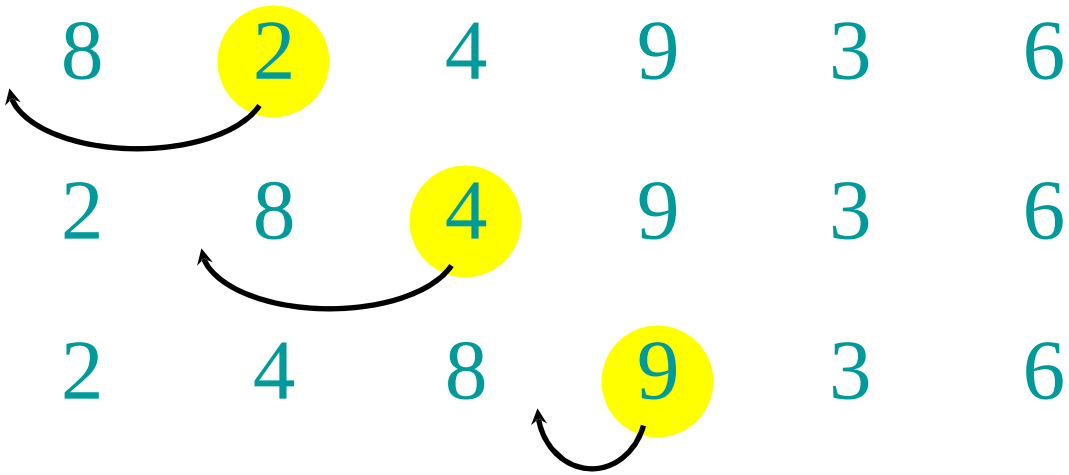


# Παράδειγμα της ταξινόμησης ένθεσης

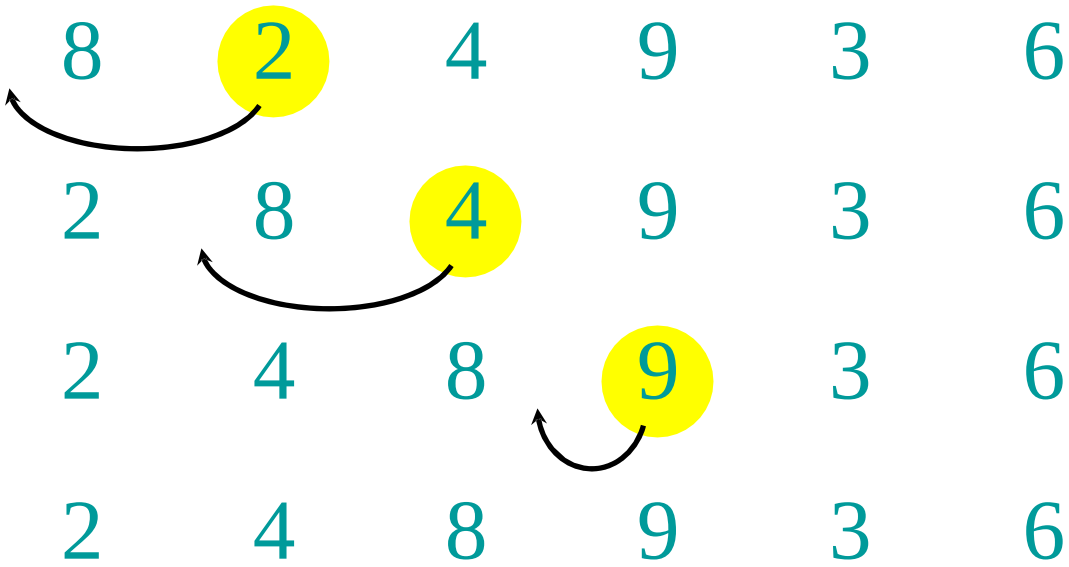
# Παράδειγμα της ταξινόμησης ένθεσης



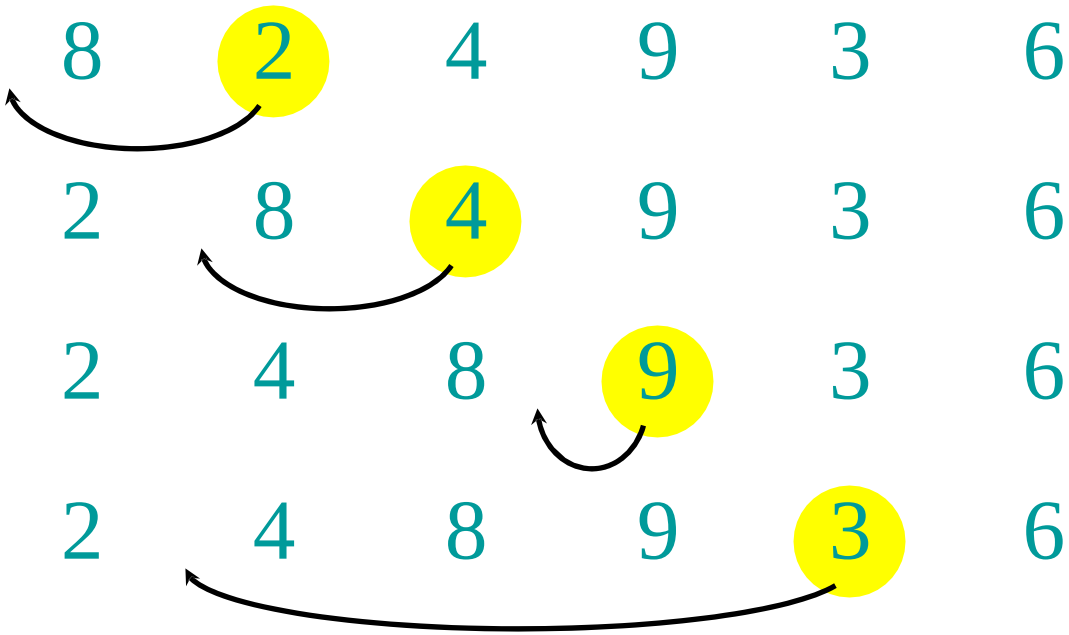
# Παράδειγμα της ταξινόμησης ένθεσης



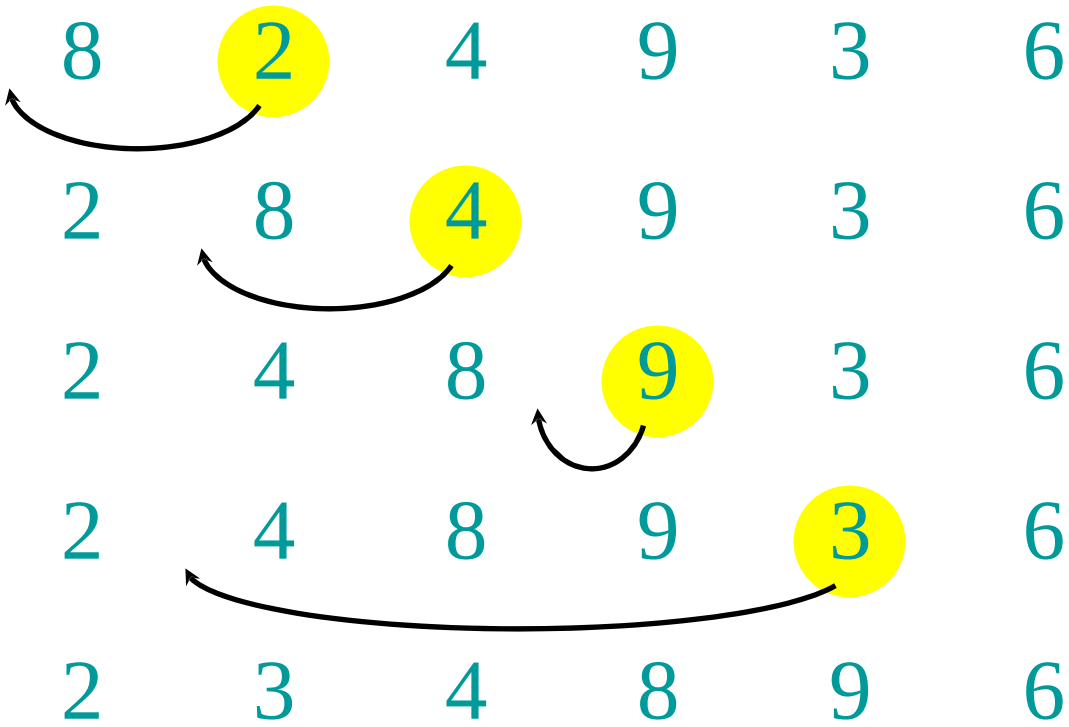
# Παράδειγμα της ταξινόμησης ένθεσης



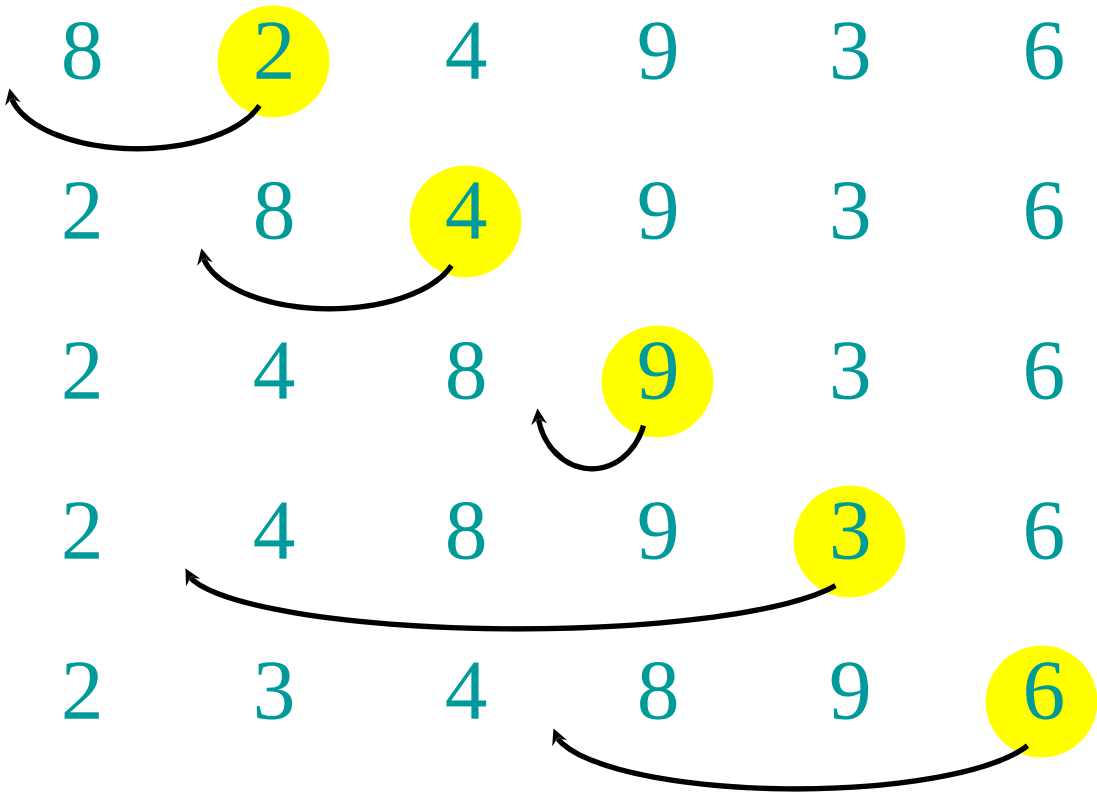
# Παράδειγμα της ταξινόμησης ένθεσης



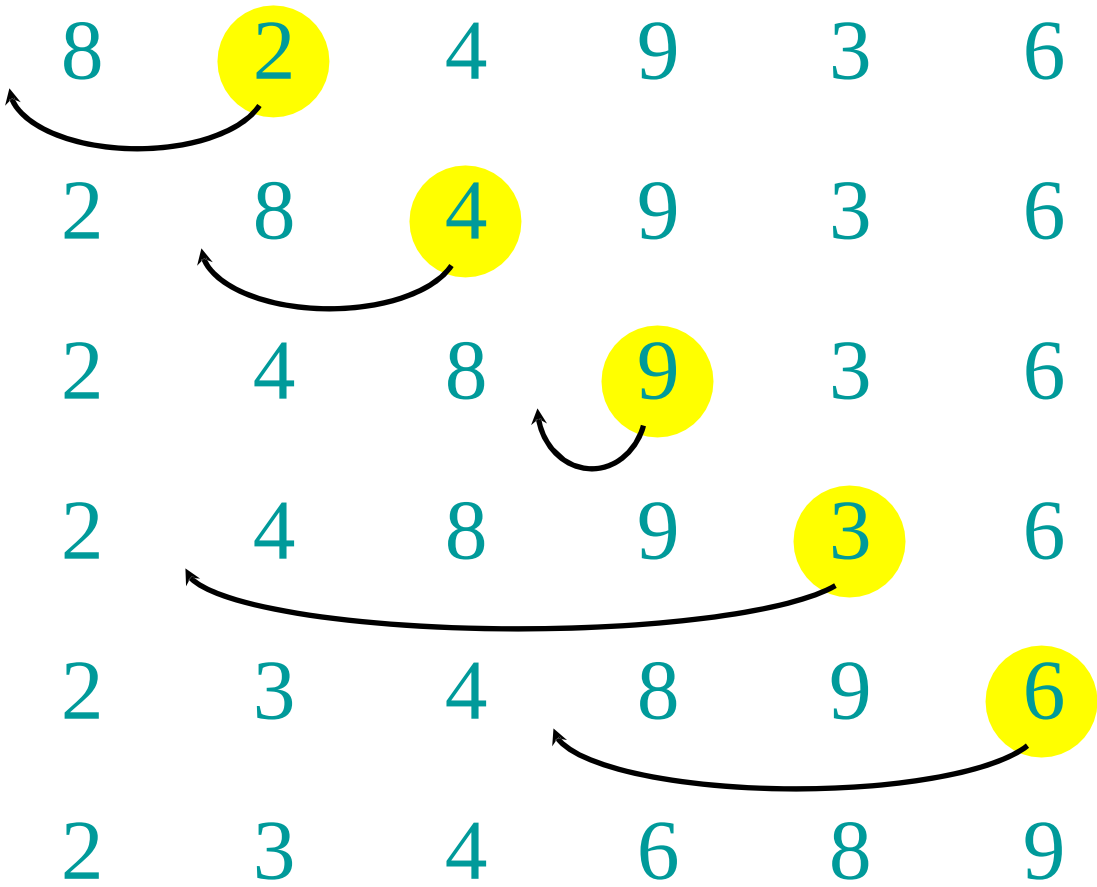
# Παράδειγμα της ταξινόμησης ένθεσης



# Παράδειγμα της ταξινόμησης ένθεσης



# Παράδειγμα της ταξινόμησης ένθεσης



# Ο χρόνος εκτέλεσης

- Ο χρόνος εκτέλεσης εξαρτάται από την είσοδο: είναι πιο εύκολο να ταξινομήσεις μία ήδη ταξινομημένη λίστα.
- Εκφράζουμε το χρόνο εκτέλεσης συναρτήσει του μεγέθους εισόδου, αφού είναι πιο εύκολο να ταξινομήσουμε μικρές ακολουθίες σε σχέση με μεγάλες.
- Γενικά, αναζητούμε πάνω όρια στο χρόνο εκτέλεσης, διότι η ύπαρξη εγγυήσεων στην ταχύτητα εκτέλεσης αλγορίθμων είναι επιθυμητή.

| INSERTION-SORT $(A)$ | *cost* | *times* |
|---|---|---|
| 1 for $j = 2$ to $A.length$ | $c_1$ | $n$ |
| 2 $key = A[j]$ | $c_2$ | $n-1$ |
| 3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$. | 0 | $n-1$ |
| 4 $i = j-1$ | $c_4$ | $n-1$ |
| 5 while $i > 0$ and $A[i] > key$ | $c_5$ | $\sum_{j=2}^{n} t_j$ |
| 6 $A[i+1] = A[i]$ | $c_6$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| 7 $i = i-1$ | $c_7$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| 8 $A[i+1] = key$ | $c_8$ | $n-1$ |

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{i=2}^{n} (t_j - 1) + c_8(n-1) .$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

$t_j = 1$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

$t_j = j$

# Είδη Ανάλυσης

**Χειρότερη περίπτωση:** (συνήθως)

- $T(n)$ = ο μέγιστος χρόνος εκτέλεσης του αλγορίθμου σε οποιαδήποτε είσοδο μεγέθους $n$.

**Μέση περίπτωση:** (μερικές φορές)

- $T(n)$ = αναμενόμενος χρόνος του αλγορίθμου για όλες τις εισόδους μεγέθους $n$.
- Πρέπει να είναι γνωστή η στατιστική κατανομή των εισόδων.

**Καλύτερη περίπτωση:** (πιθανή παραπλάνηση)

- Τρέχουμε ένα αργό αλγόριθμο ο οποίος τρέχει γρήγορα σε κάποιες εισόδους.

# Χρόνος ανεξάρτητος της αρχιτεκτονικής υπολογιστή

*Ποιος είναι ο χρόνος χειρότερης περίπτωσης της ταξινόμησης ένθεσης;*

- Εξαρτάται από την ταχύτητα του υπολογιστή μας:
  - Σχετική ταχύτητα(στον ίδιο υπολογιστή),
  - Απόλυτη ταχύτητα (σε διαφορετικές μηχανές).

**Η βασική ιδέα:**

- Αγνόησε τις σταθερές που εξαρτώνται από την αρχιτεκτονική.
- Κοιτάμε στην αύξηση της $T(n)$ καθώς $n \to \infty$ .

**“Ασυμπτωτική Ανάλυση”**

# Συμβολισμός Θ

***Ορισμός:***

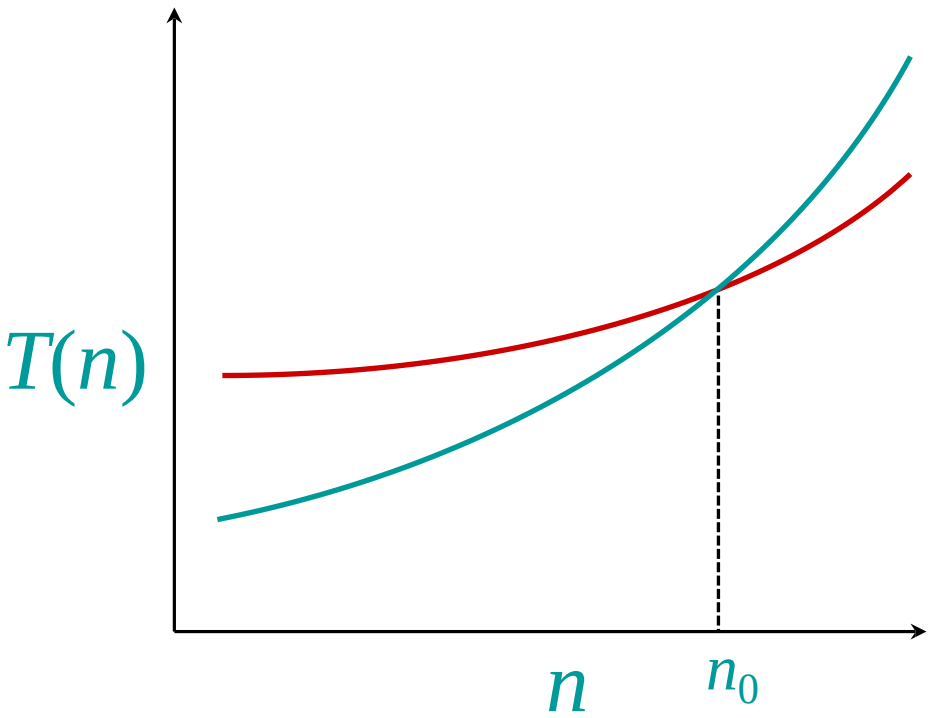
$\Theta(g(n)) = \{\ f(n)$ : υπάρχουν θετικές σταθερές , $c_1$, $c_2$, και $n_0$ τέτοιες ώστε $0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ για όλα τα $n \geq n_0\}$

***Ουσιαστικά:***

- Αγνόησε τους όρους χαμηλότερης τάξης
- Αγνόησε τους συντελεστές στους όρους υψηλότερης τάξης.
- Παράδειγμα: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

# Ασυμπτωτική απόδοση

Όταν το $n$ γίνει αρκετά μεγάλο, ένας αλγόριθμος πολυπλοκότητας $\Theta(n^2)$ *πάντα* είναι καλύτερος από ένα αλγόριθμο πολυπλοκότητας $\Theta(n^3)$.




- Δεν θα πρέπει να αγνοούμε τους ασυμπτωτικά αργότερους αλγορίθμους
- Ο σχεδιασμός στην πράξη συχνά απαιτεί προσεκτική εξισορρόπηση διαφορετικών συχνά αντικρουόμενων στόχων

# Ανάλυση της Ενθετικής Ταξινόμησης

***Χειρότερη περίπτωση:*** Είσοδος ταξινομημένη αντίστροφα.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$ [αριθμητική πρόοδος]

***Μέση περίπτωση:*** Όλες οι μεταθέσεις είναι εξίσου πιθανές

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

*Είναι η ενθετική ταξινόμηση ένας γρήγορος αλγόριθμος ταξινόμησης;*

- Ισχύει μερικώς, για μικρά $n$.
- Καθόλου, για μεγάλα $n$.

# Συγχωνευτική ταξινόμηση

**MERGE-SORT** $A[1 \ldots n]$

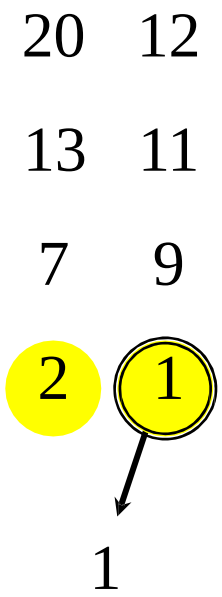
1. If $n = 1$, done.
2. Recursively sort $A[\,1 \ldots \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \ldots n\,]$.
3. "***Merge***" the 2 sorted lists.

***Βασική υπορουτίνα:*** **MERGE**

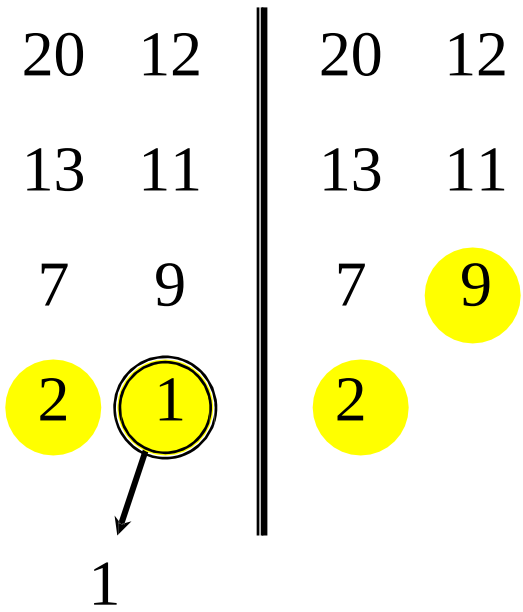
# Ταξινόμηση δύο ταξινομημένων συστοιχιών

| | |
|---|---|
| 20 | 12 |
| 13 | 11 |
| 7 | 9 |
| 2 | 1 |

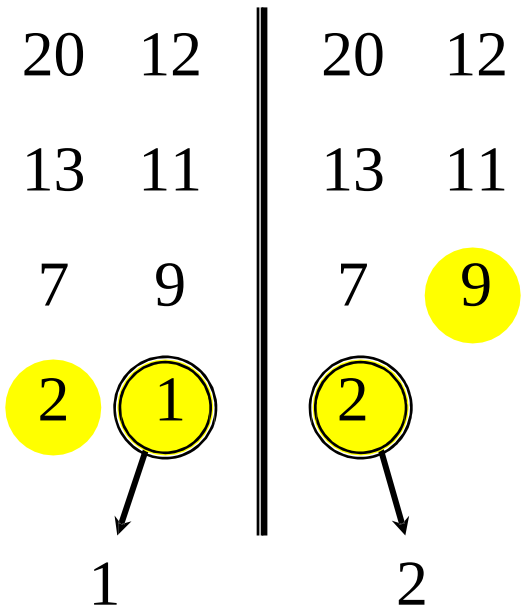
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



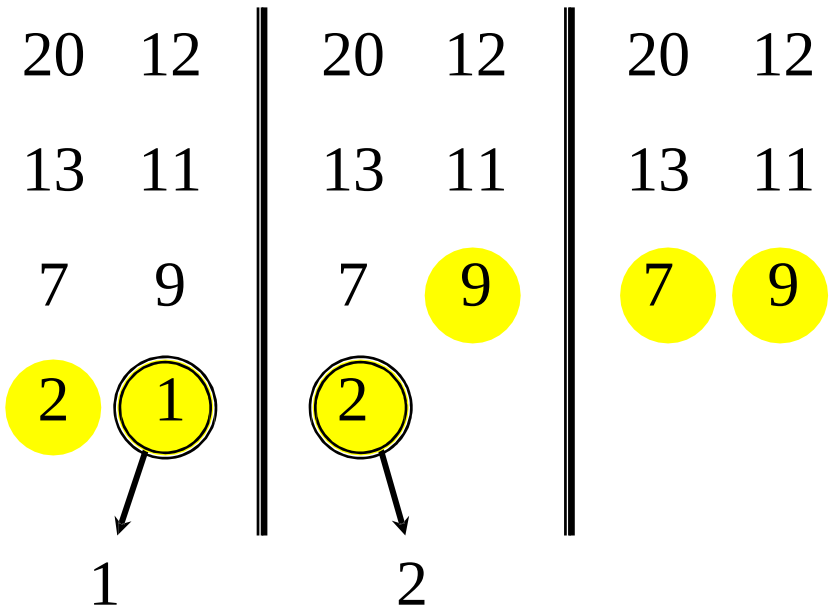
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



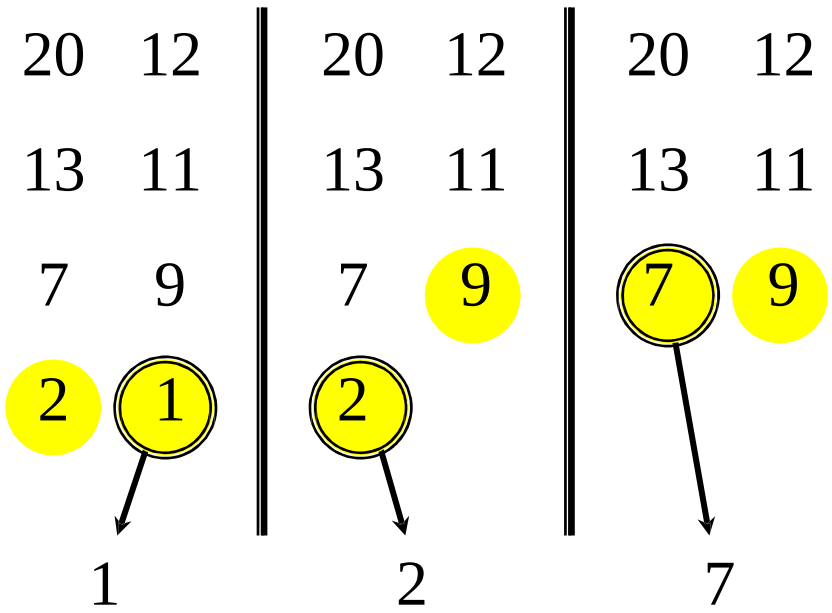
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



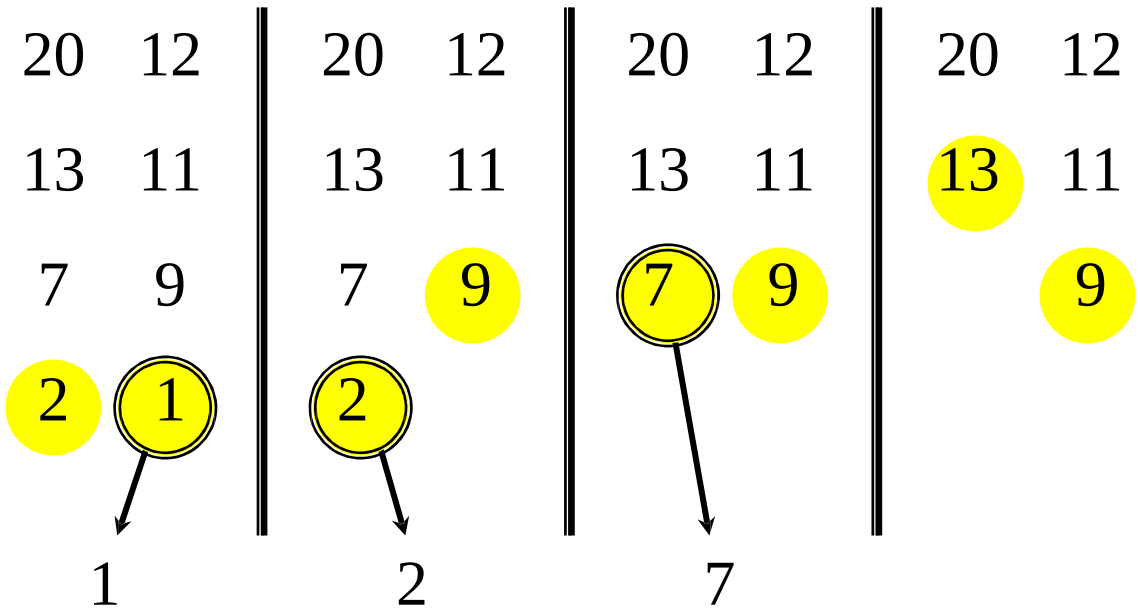
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



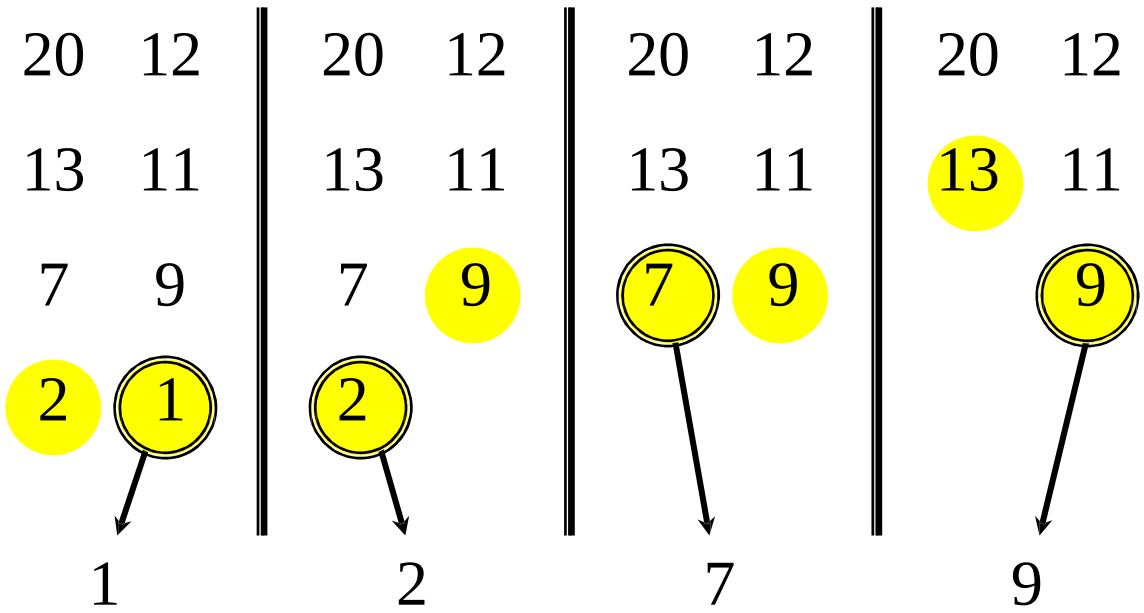
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



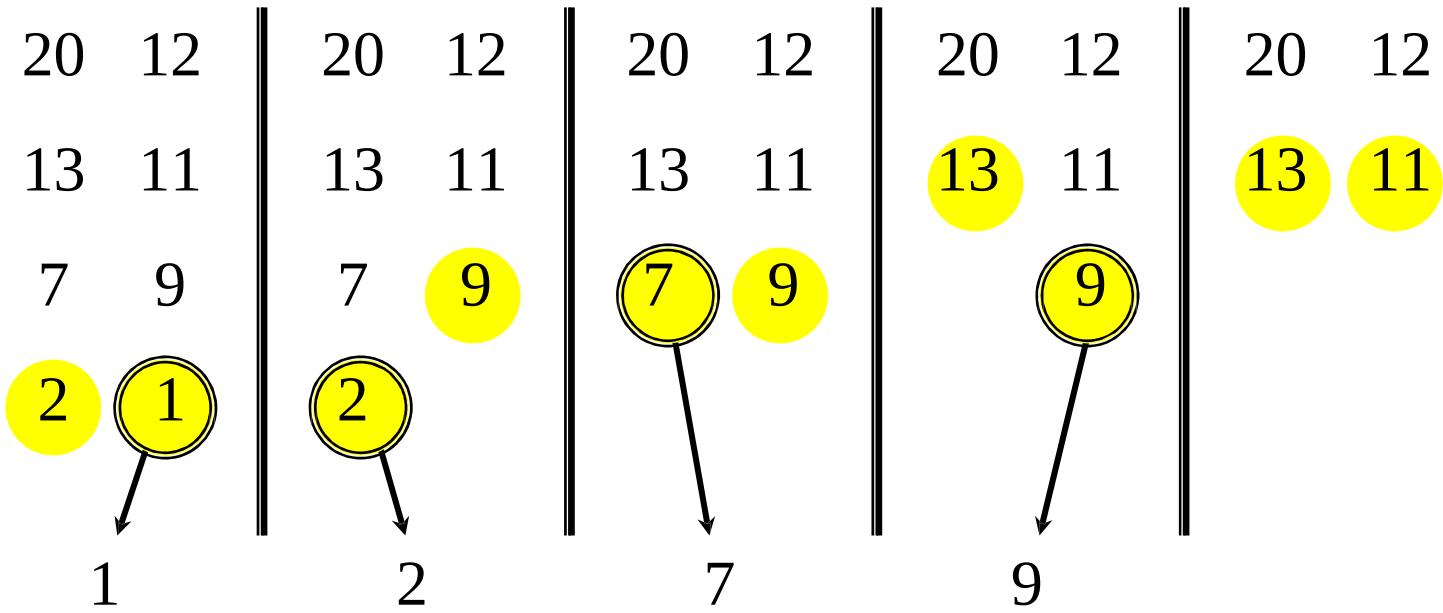
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



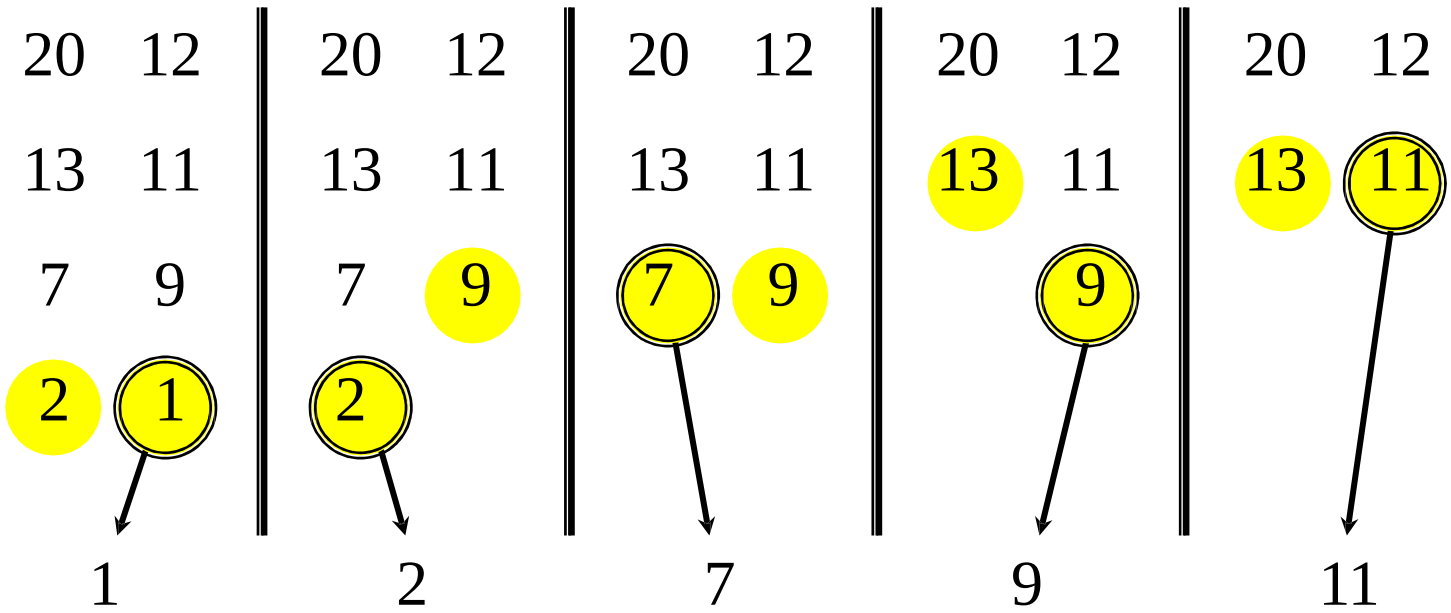
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



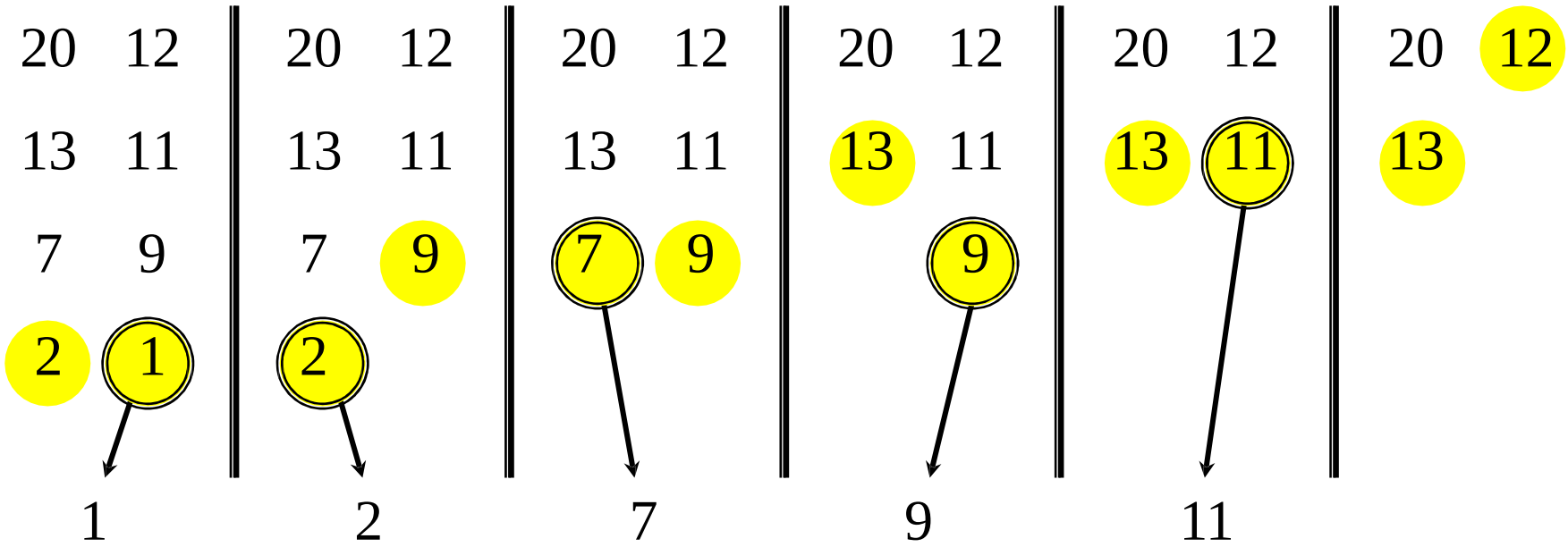
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



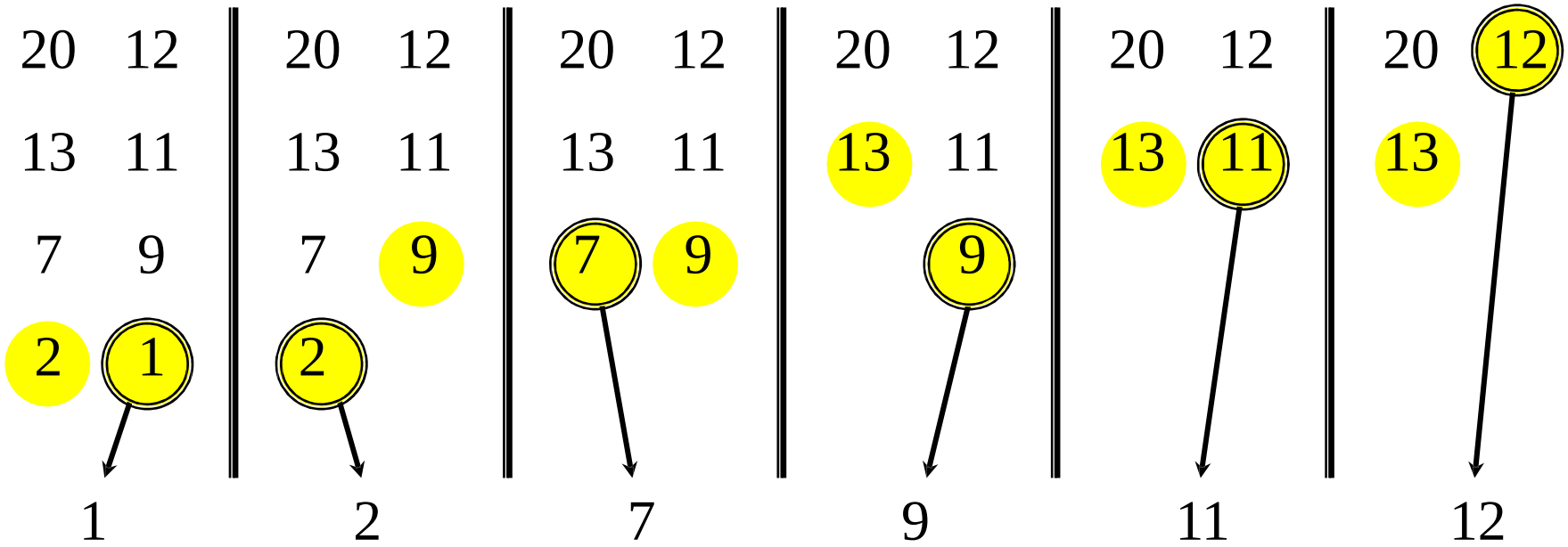
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



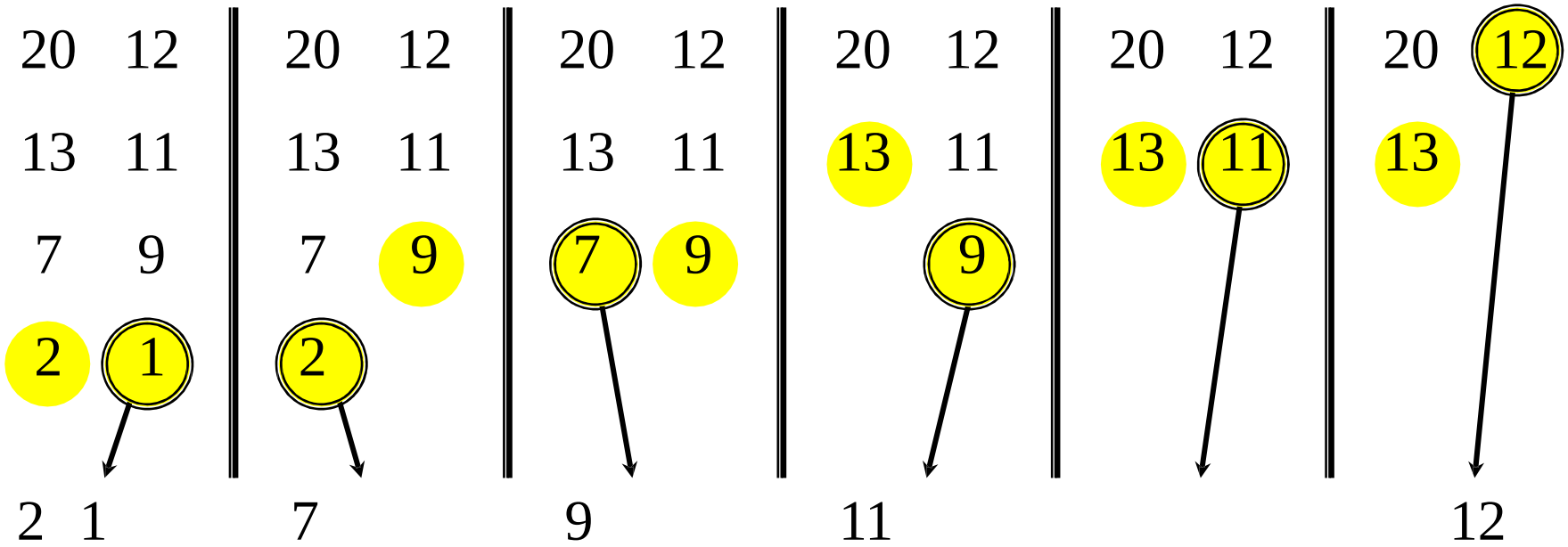
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



# Ταξινόμηση δύο ταξινομημένων συστοιχιών



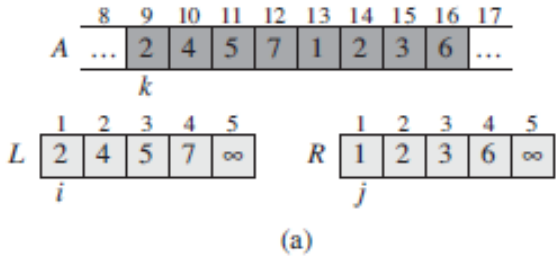
# Ταξινόμηση δύο ταξινομημένων συστοιχιών



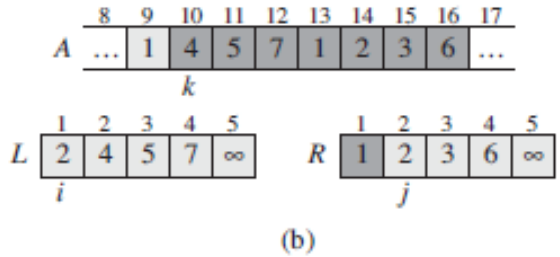

Χρόνος = $\Theta(n)$ για τη συγχώνευση συνολικά $n$ στοιχείων (γραμμικός χρόνος).

```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1 .. n_1 + 1] and R[1 .. n_2 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n_2
 7      R[j] = A[q + j]
 8  L[n_1 + 1] = ∞
 9  R[n_2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

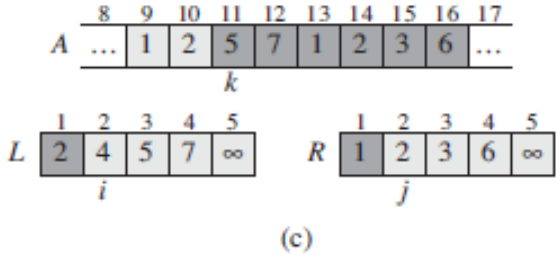
Παράδειγμα:
MERGE(A,9,12,16)



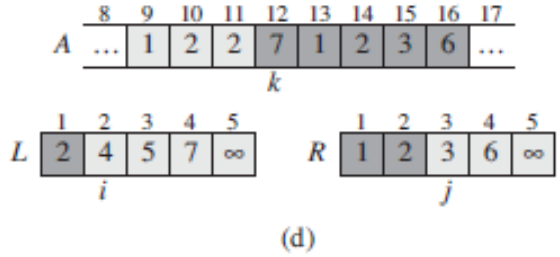

(a)



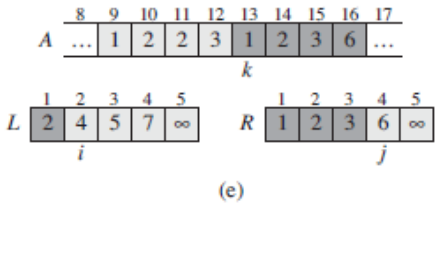

(b)



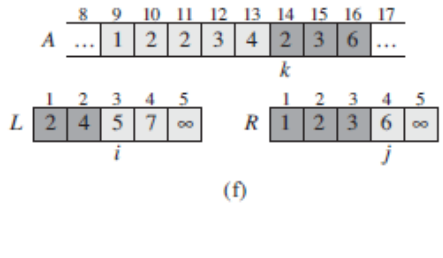

(c)



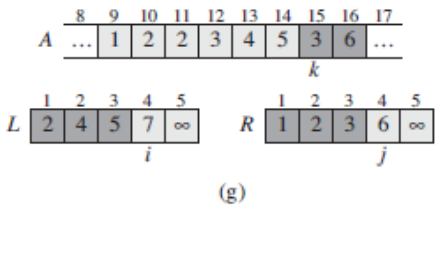

(d)



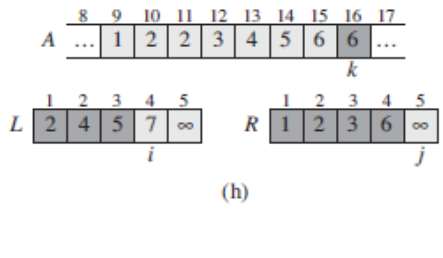

(e)



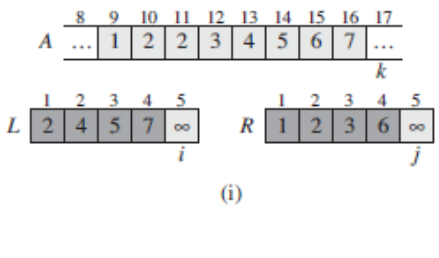

(f)



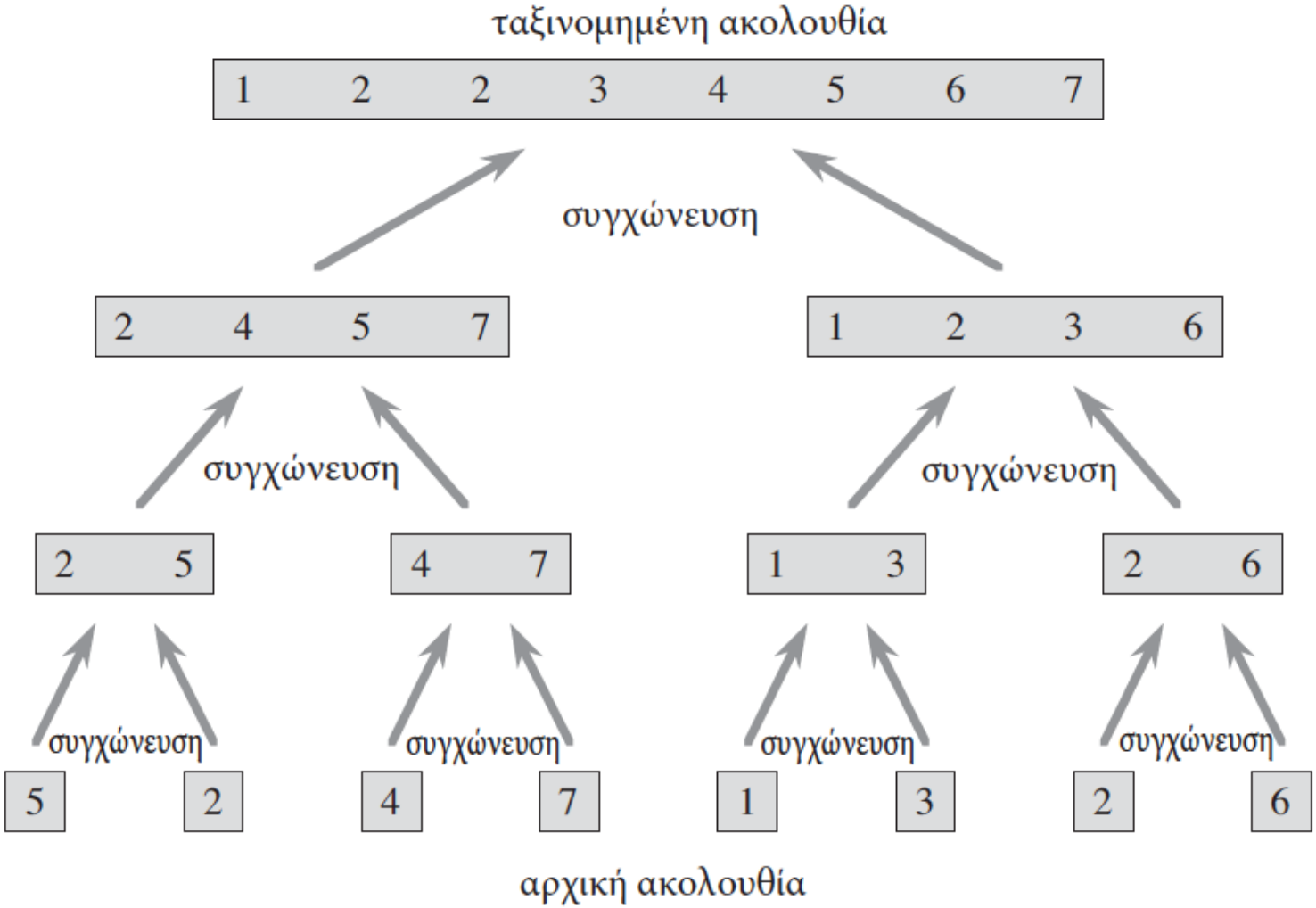

(g)




(h)




(i)




**Σχήμα 2.4** Η λειτουργία της συγχωνευτικής ταξινόμησης στη συστοιχία $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. Τα μήκη των συγχωνευόμενων ταξινομημένων ακολουθιών αυξάνονται διαδοχικά καθώς ο αλγόριθμος προχωρά σταδιακά από τη βάση προς την κορυφή.

# Ανάλυση Συγχωνευτικής Ταξινόμησης

| $T(n)$ | **MERGE-SORT** $A[1 \,..\, n]$ |
|---|---|
| $\Theta(1)$ | 1. If $n = 1$, done. |
| $2T(n/2)$ | 2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$. |
| $\Theta(n)$ | 3. "***Merge***" the 2 sorted lists. |

Κανονικά, $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$

αλλά τελικά προκύπτει ότι δεν έχει σημασία για τον προσδιορισμό της ασυμπτωτικής πολυπλοκότητας.

# Αναδρομική σχέση για τη συγχωνευτική ταξινόμηση

$$T(n) = \begin{cases} \Theta(1) \text{ αν } n = 1; \\ 2T(n/2) + \Theta(n) \text{ αν } n > 1. \end{cases}$$

- Συνήθως, θα παραλείπουμε να δηλώνουμε την περίπτωση βάσης όταν $T(n) = \Theta(1)$ για επαρκώς μικρό $n$, αλλά μόνο όταν αυτό δεν έχει επιπτώσεις στην ασυμπτωτική λύση της αναδρομής.
- Αργότερα, θα δούμε αρκετούς τρόπους για να βρούμε ένα καλό πάνω όριο για το χρόνο $T(n)$.

# Δένδρο Αναδρομής

Επίλυση της $T(n) = 2T(n/2) + cn$, όταν $c > 0$ είναι σταθερά.

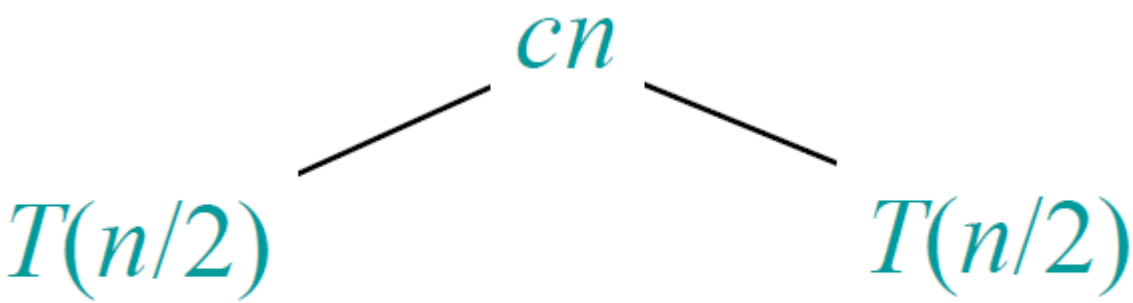
# Δένδρο Αναδρομής

Επίλυση της $T(n) = 2T(n/2) + cn$, όταν $c > 0$ είναι σταθερά.

$T(n)$

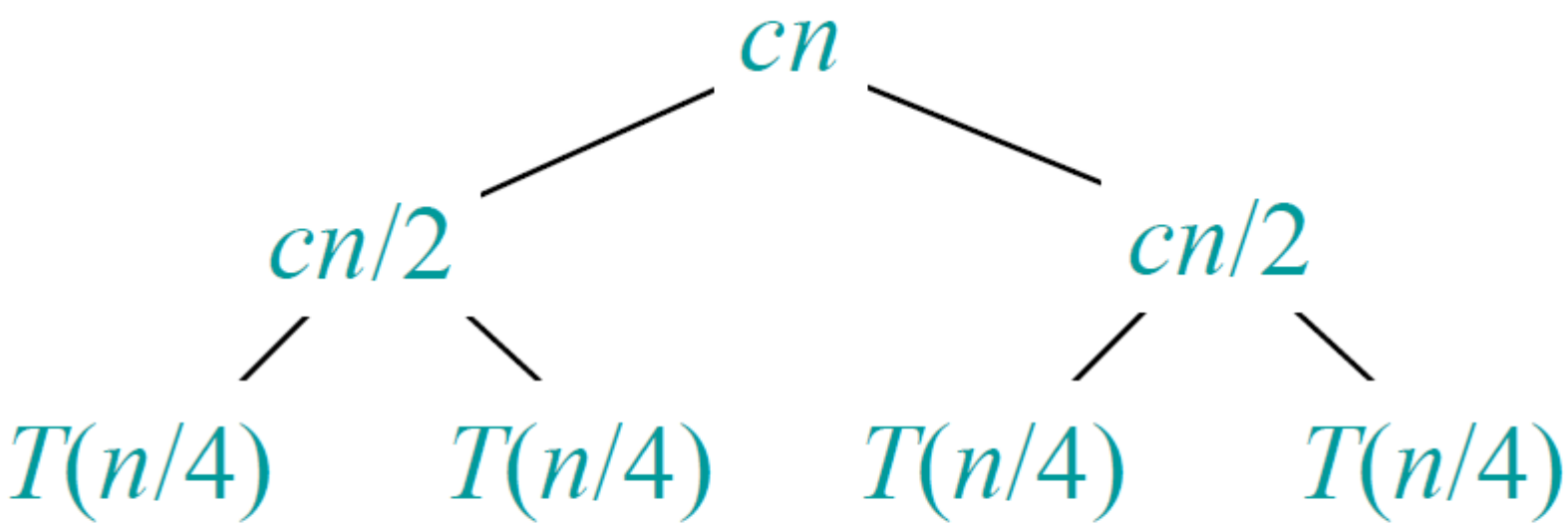
# Δένδρο Αναδρομής

Επίλυση της $T(n) = 2T(n/2) + cn$, όταν $c > 0$ είναι σταθερά.



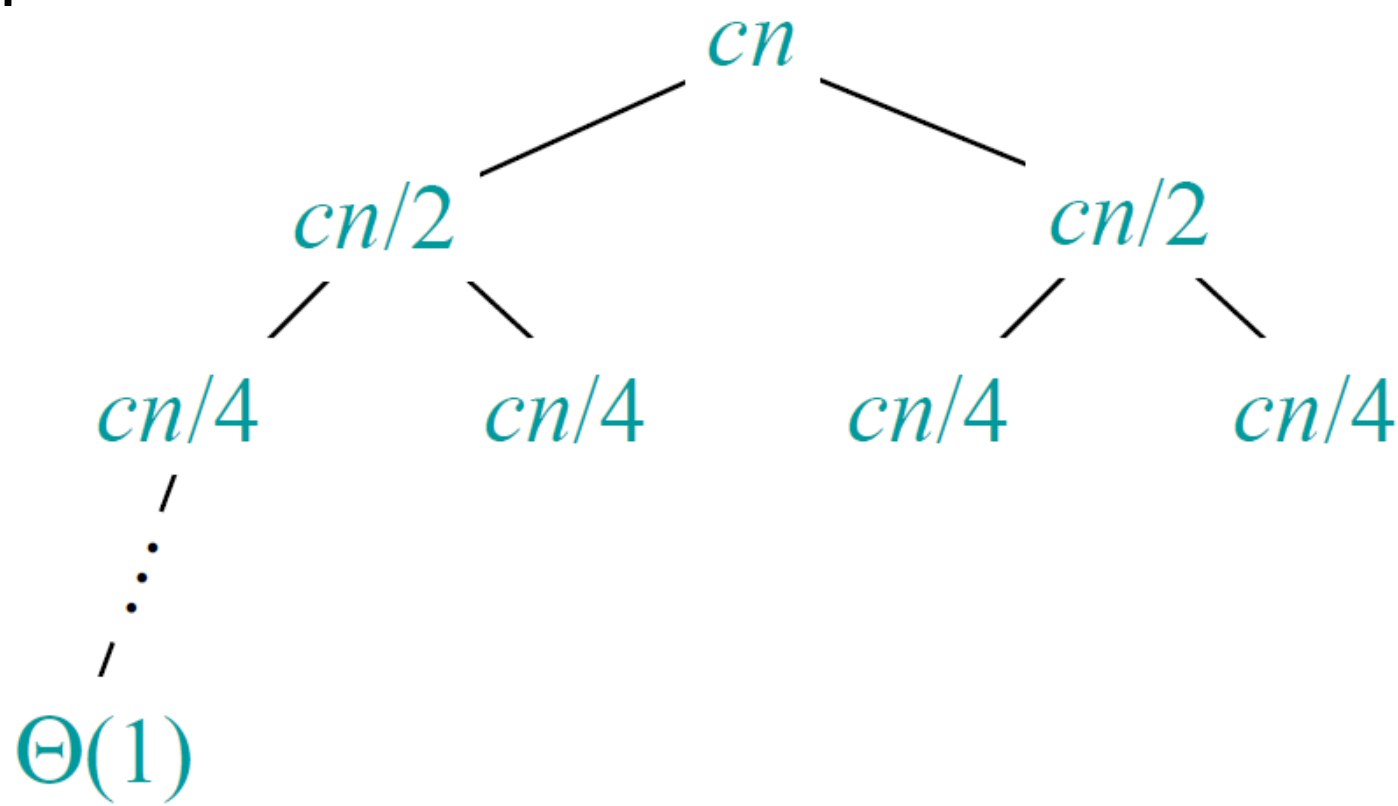
# Δένδρο Αναδρομής

Επίλυση της $T(n) = 2T(n/2) + cn$, όταν $c > 0$ είναι σταθερά.



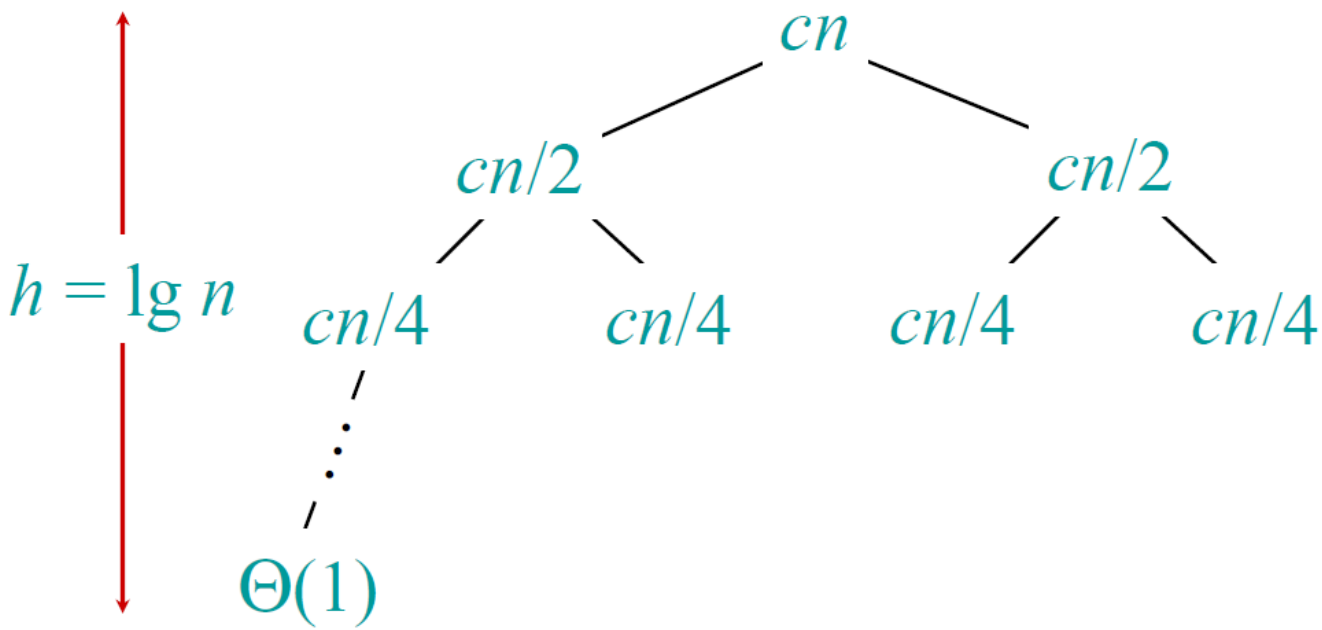
# Δένδρο Αναδρομής

Επίλυση της $T(n) = 2T(n/2) + cn$, όταν $c > 0$ είναι σταθερά.

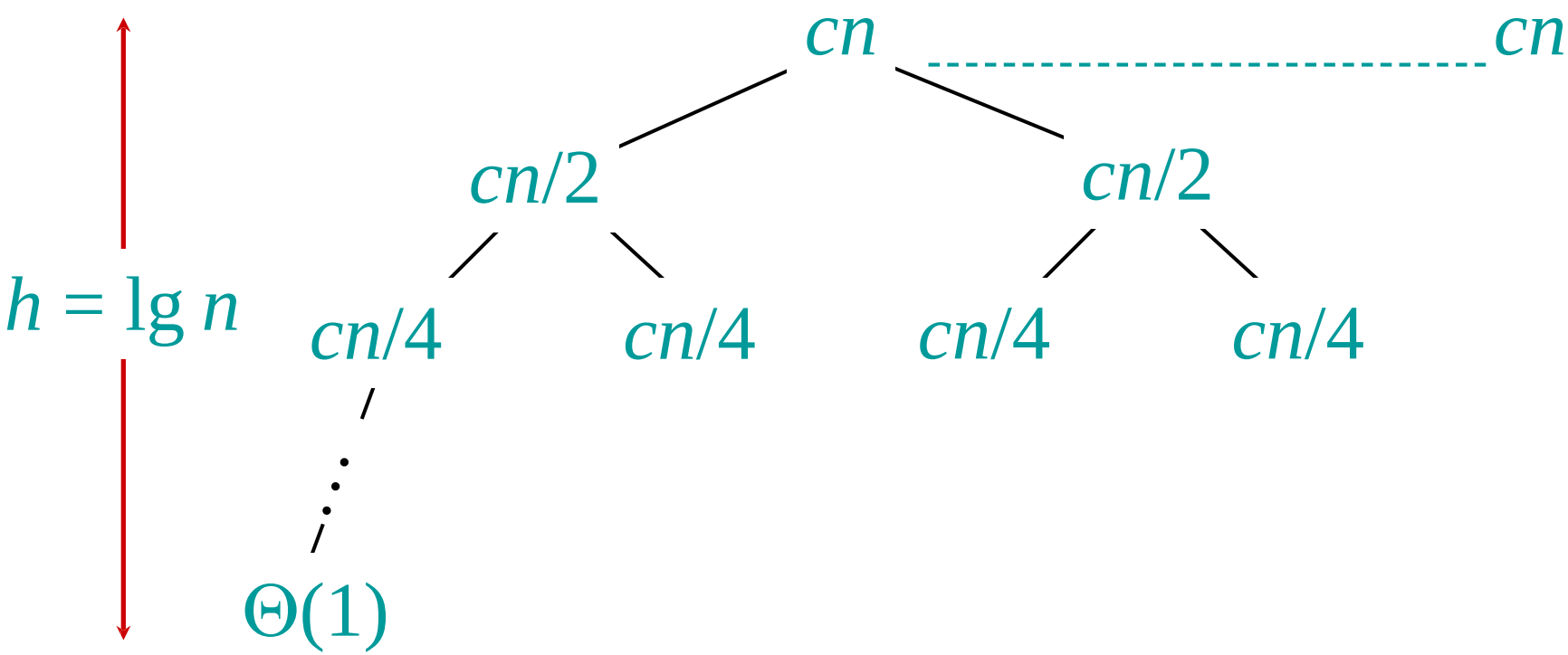


# Δένδρο Αναδρομής

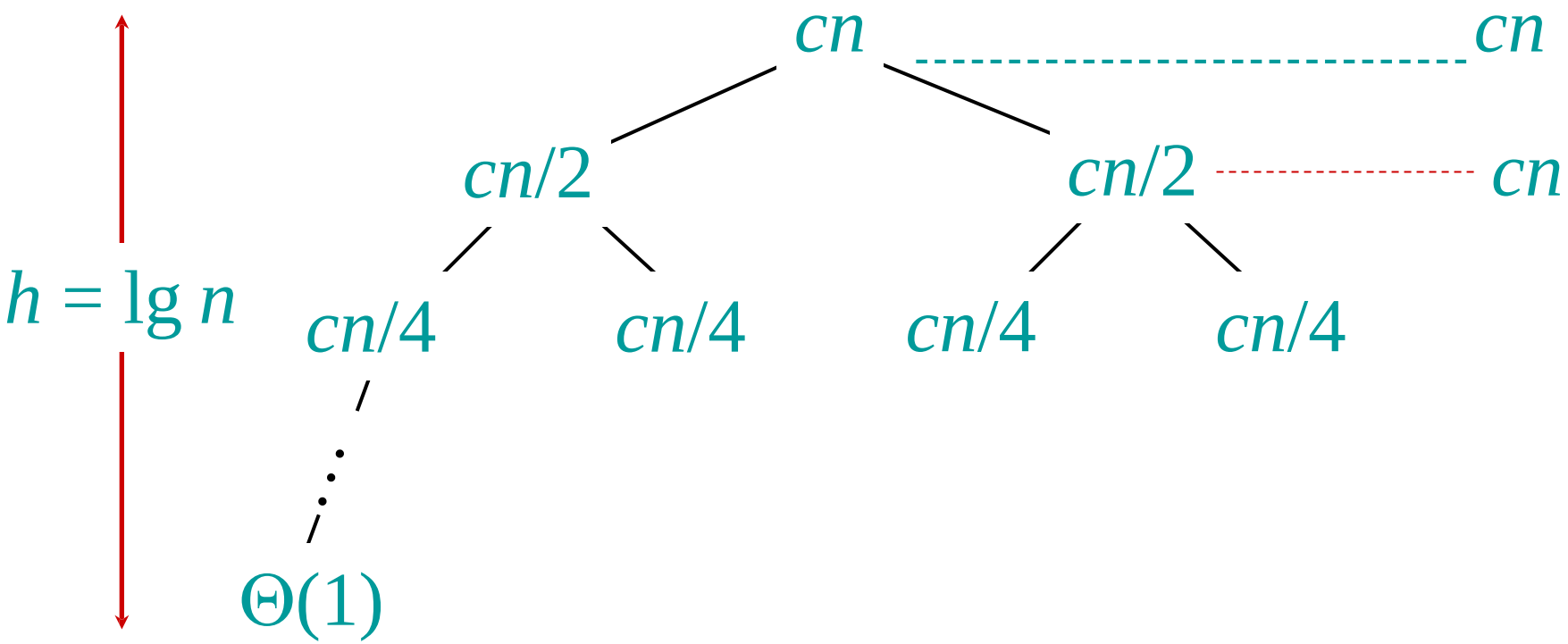
Επίλυση της $T(n) = 2T(n/2) + cn$, όταν $c > 0$ είναι σταθερά.



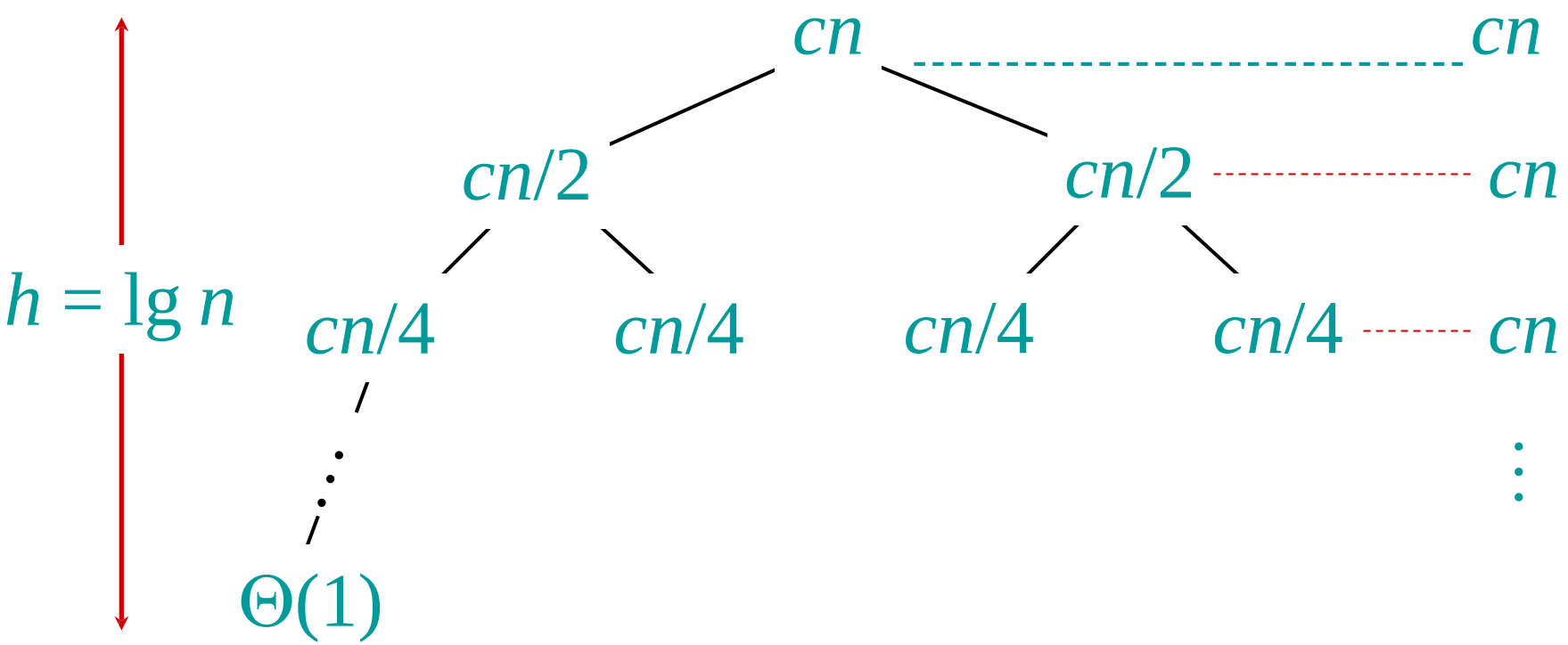
# Δένδρο Αναδρομής



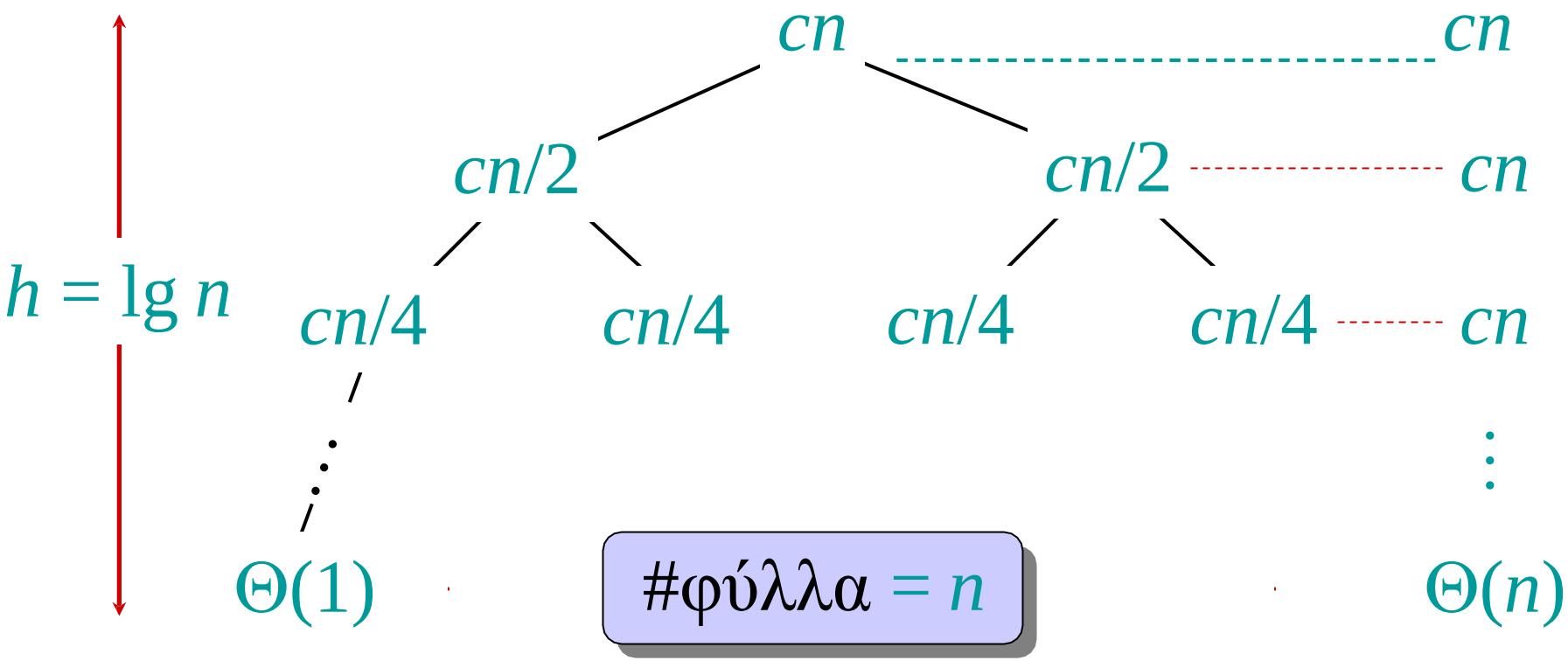
# Δένδρο Αναδρομής



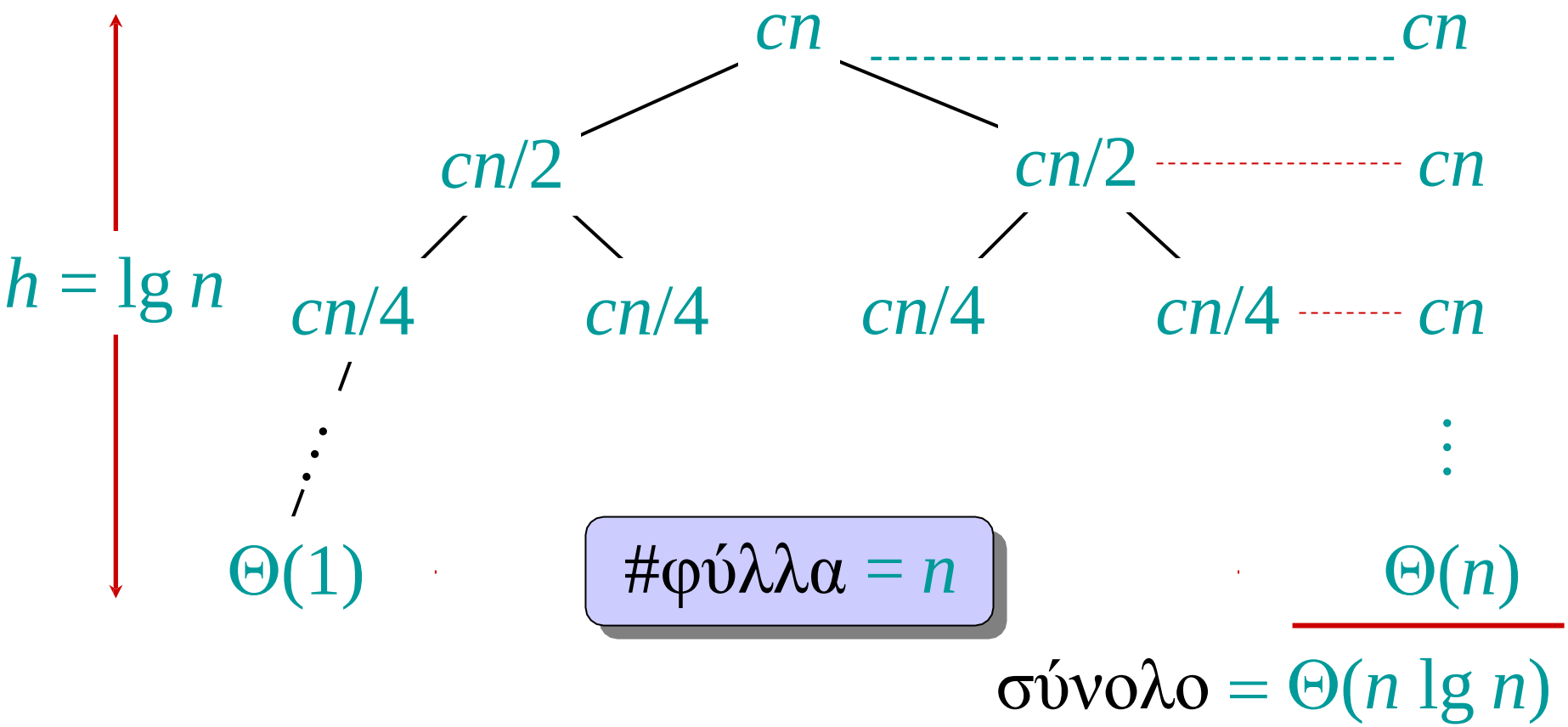
# Δένδρο Αναδρομής



# Δένδρο Αναδρομής



# Δένδρο Αναδρομής



# Συμπεράσματα

- $\Theta(n \lg n)$ αυξάνει πιο αργά από το $\Theta(n^2)$.
- Επομένως, η συγχωνευτική ταξινόμηση είναι ασυμπτωτικά καλύτερος από την ενθετική ταξινόμηση στη χειρότερη περίπτωση.
- Στην πράξη, η συγχωνευτική ταξινόμηση είναι καλύτερη από την ενθετική ταξινόμηση για $n > 30$.

| | $n$ | $n \log_2 n$ | $n^2$ | $n^3$ | $1.5^n$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|---|
| $n = 10$ | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 4 sec |
| $n = 30$ | < 1 sec | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 18 min | $10^{25}$ years |
| $n = 50$ | < 1 sec | < 1 sec | < 1 sec | < 1 sec | 11 min | 36 years | very long |
| $n = 100$ | < 1 sec | < 1 sec | < 1 sec | 1 sec | 12,892 years | $10^{17}$ years | very long |
| $n = 1{,}000$ | < 1 sec | < 1 sec | 1 sec | 18 min | very long | very long | very long |
| $n = 10{,}000$ | < 1 sec | < 1 sec | 2 min | 12 days | very long | very long | very long |
| $n = 100{,}000$ | < 1 sec | 2 sec | 3 hours | 32 years | very long | very long | very long |
| $n = 1{,}000{,}000$ | 1 sec | 20 sec | 12 days | 31,710 years | very long | very long | very long |

- Χρόνοι εκτέλεσης αλγορίθμων σε επεξεργαστή που εκτελεί 1.000.000 εντολές υψηλού επιπέδου
- $>10^{25}$ χρόνια = very long time