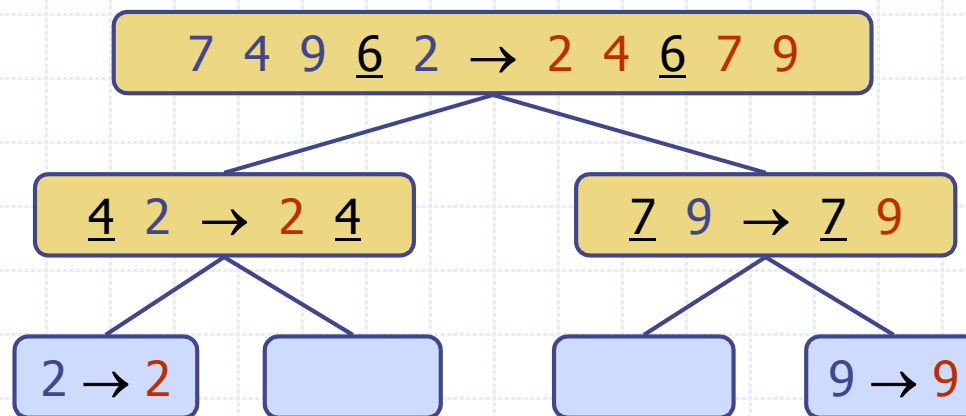


Ο αλγόριθμος Quick-Sort

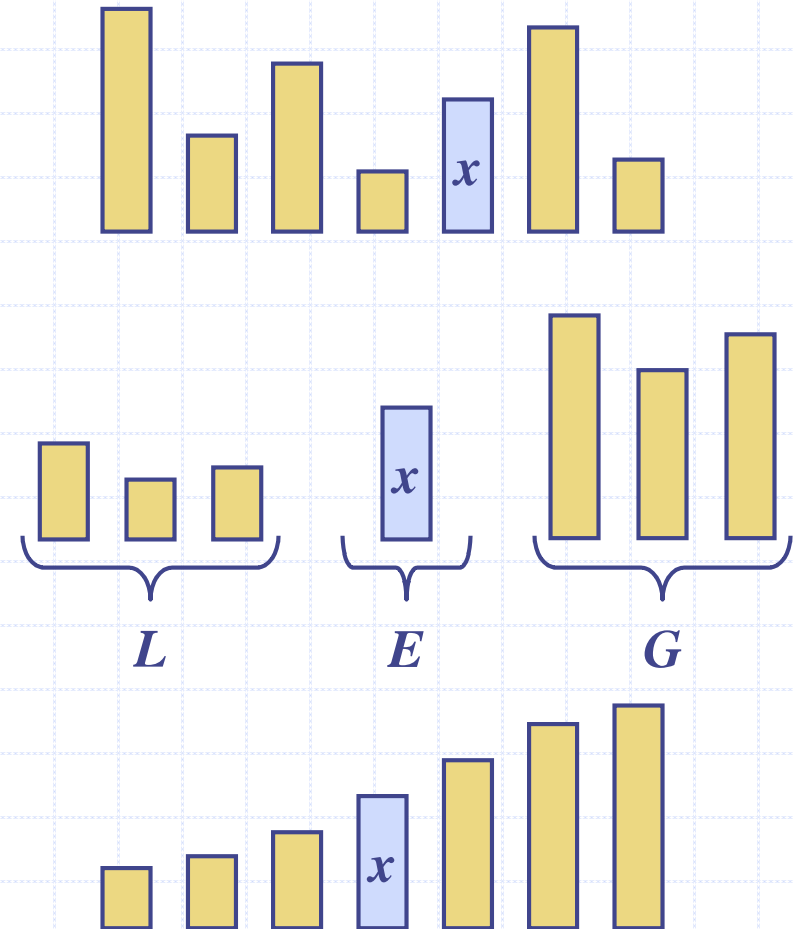


Κύρια σημεία για μελέτη

- ◆ Quick-sort (§4.3)
 - Αλγόριθμος
 - Partition step
 - Δέντρο Quick-sort
 - Παράδειγμα εκτέλεσης
- ◆ Ανάλυση του quick-sort (4.3.1)
- ◆ In-place quick-sort (§4.8)
- ◆ Συνόψιση αλγόριθμων ταξινόμησης

Quick-Sort

- ◆ Ο Quick-sort είναι ένας αλγόριθμος ταξινόμησης που χρησιμοποιεί τυχαιότητα και βασίζεται στην τεχνική «διαίρει και βασίλευε».
 - **Divide**: διαλέγεται ένα τυχαίο στοιχείο X (που ονομάζεται **pivot**) και ένα τμήμα S μέσα
 - ◆ L τα στοιχεία μικρότερα από x
 - ◆ E τα στοιχεία ίσα με x
 - ◆ G τα στοιχεία μεγαλύτερα από x
 - **Recur**: ταξινομούνται τα σύνολα L και G
 - **Conquer**: ενώνονται τα στοιχεία L , E και G



Τμηματοποίηση (partition)

- ◆ Τμηματοποιούμε μια ακολουθία εισόδου ως εξής:
 - Απομακρύνουμε διαδοχικά, κάθε στοιχείο y από την ακολουθία S και
 - Εισάγουμε το y στο σύνολο L , E ή στο G , ανάλογα με το αποτέλεσμα της σύγκρισης του με το x .
- ◆ Κάθε εισαγωγή ή απομάκρυνση γίνεται στην αρχή ή στο τέλος της ακολουθίας, και έτσι παίρνει χρόνο $O(1)$.
- ◆ Έτσι, το τμήμα του quick-sort που αφορά το partition παίρνει χρόνο $O(n)$.

Algorithm *partition*(S, p)

Input sequence S , position p of pivot

Output subsequences L, E, G of the elements of S less than, equal to, or greater than the pivot, resp.

$L, E, G \leftarrow$ empty sequences

$x \leftarrow S.remove(p)$

while $\neg S.isEmpty()$

$y \leftarrow S.remove(S.first())$

if $y < x$

$L.insertLast(y)$

else if $y = x$

$E.insertLast(y)$

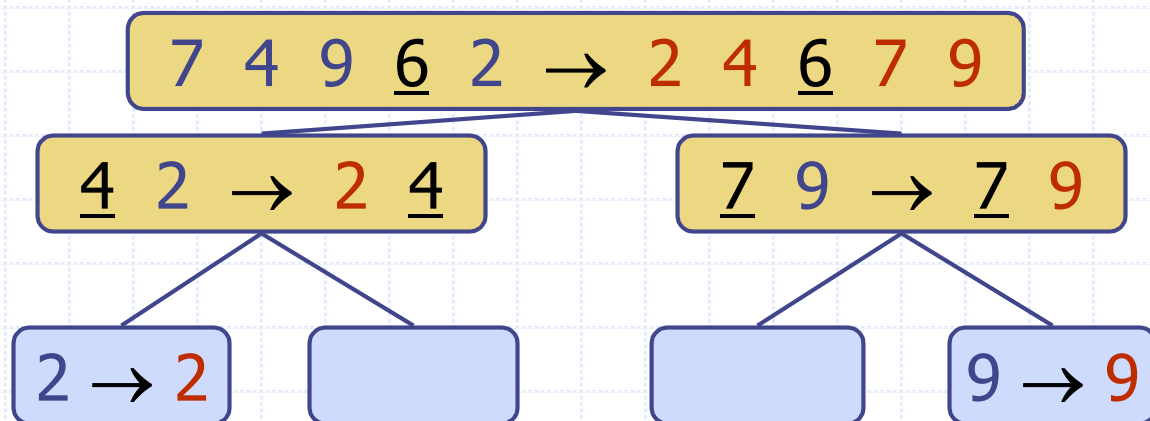
else $\{ y > x \}$

$G.insertLast(y)$

return L, E, G

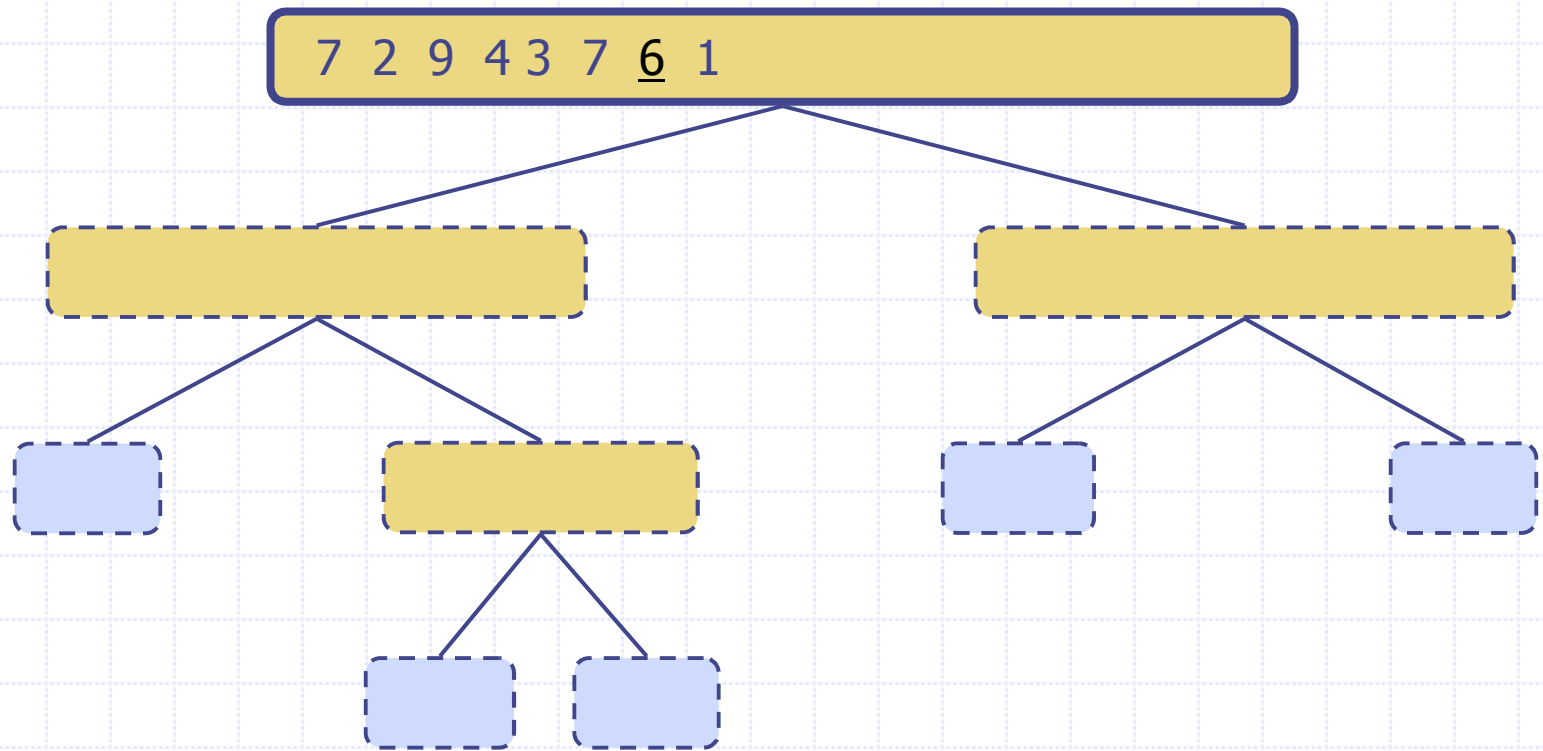
Το δέντρο Quick-Sort

- ◆ Μια εκτέλεση του quick-sort αναπαρίσταται με ένα δυαδικό δέντρο.
 - Κάθε κόμβος αναπαριστά μια αναδρομική κλήση του αλγόριθμου quick-sort και αποθηκεύει
 - ◆ Την αταξινόμητη ακολουθία πριν από την εκτέλεση και το pivot
 - ◆ Την ταξινομημένη ακολουθία στο τέλος της εκτέλεσης.
 - Η ρίζα είναι η αρχική κλήση.
 - Τα φύλλα είναι κλήσεις σε υποακολουθίες 0 ή 1



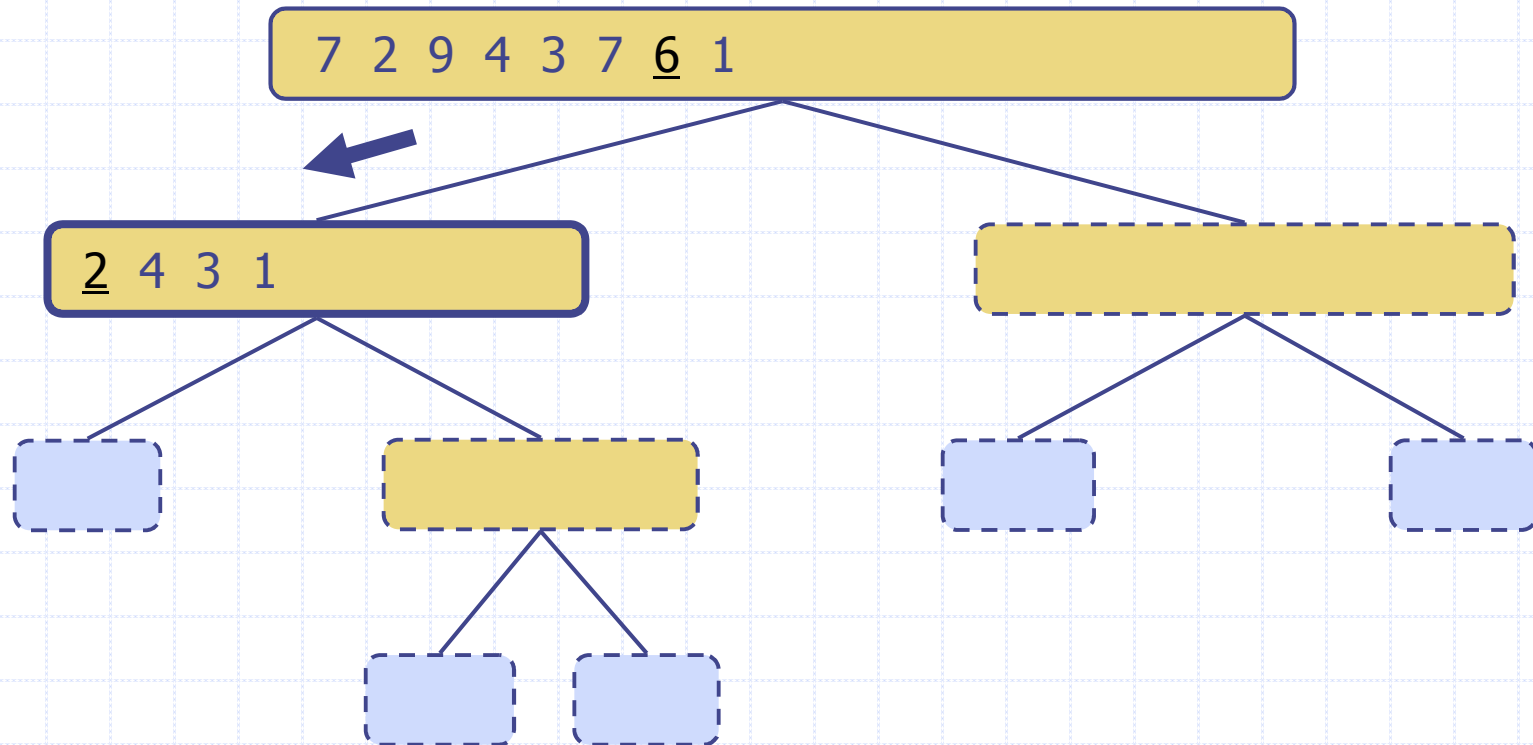
Παράδειγμα εκτέλεσης

◆ Η επιλογή του Pivot



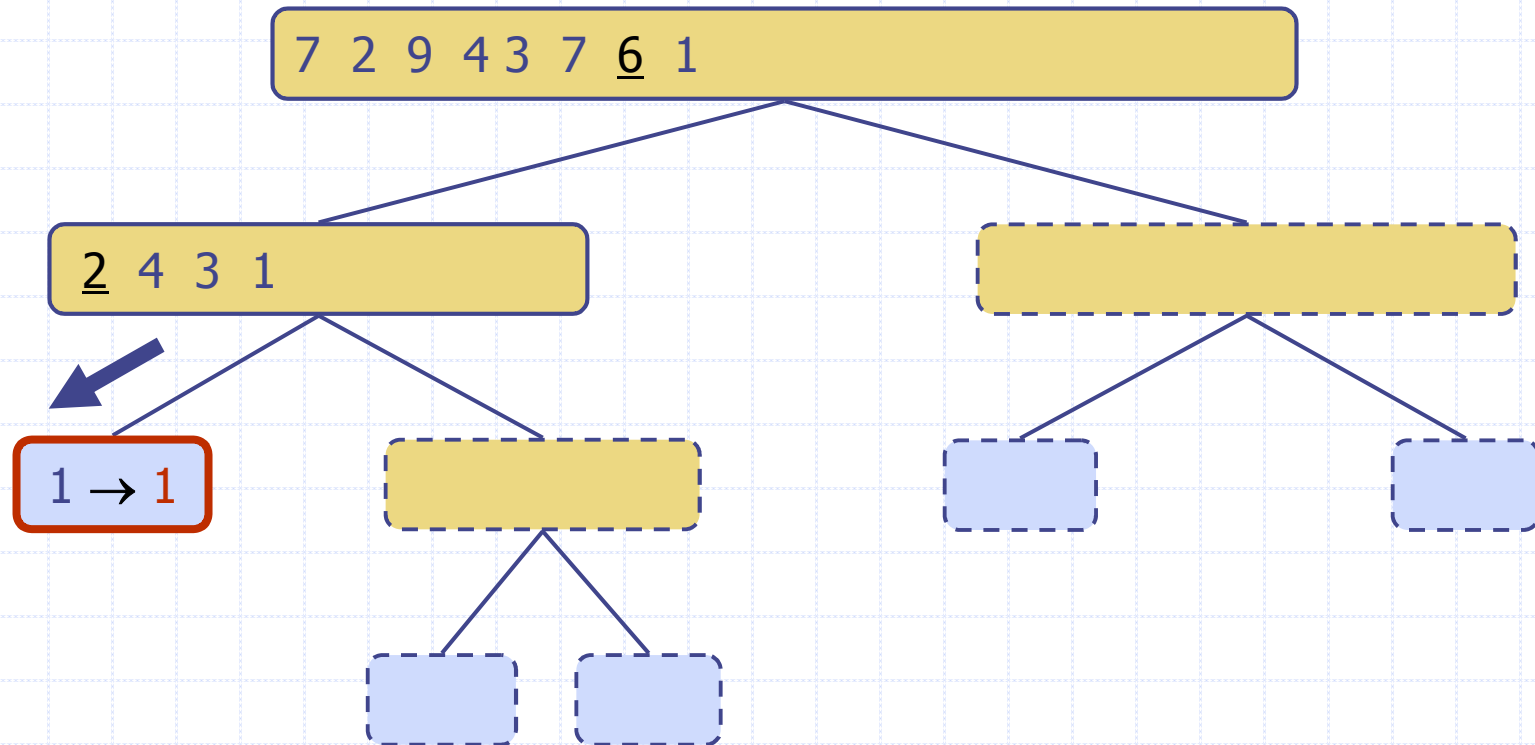
Παράδειγμα εκτέλεσης (συν.)

- ◆ Τμηματοποίηση ,αναδρομική κλήση, επιλογή του ρινοτ.



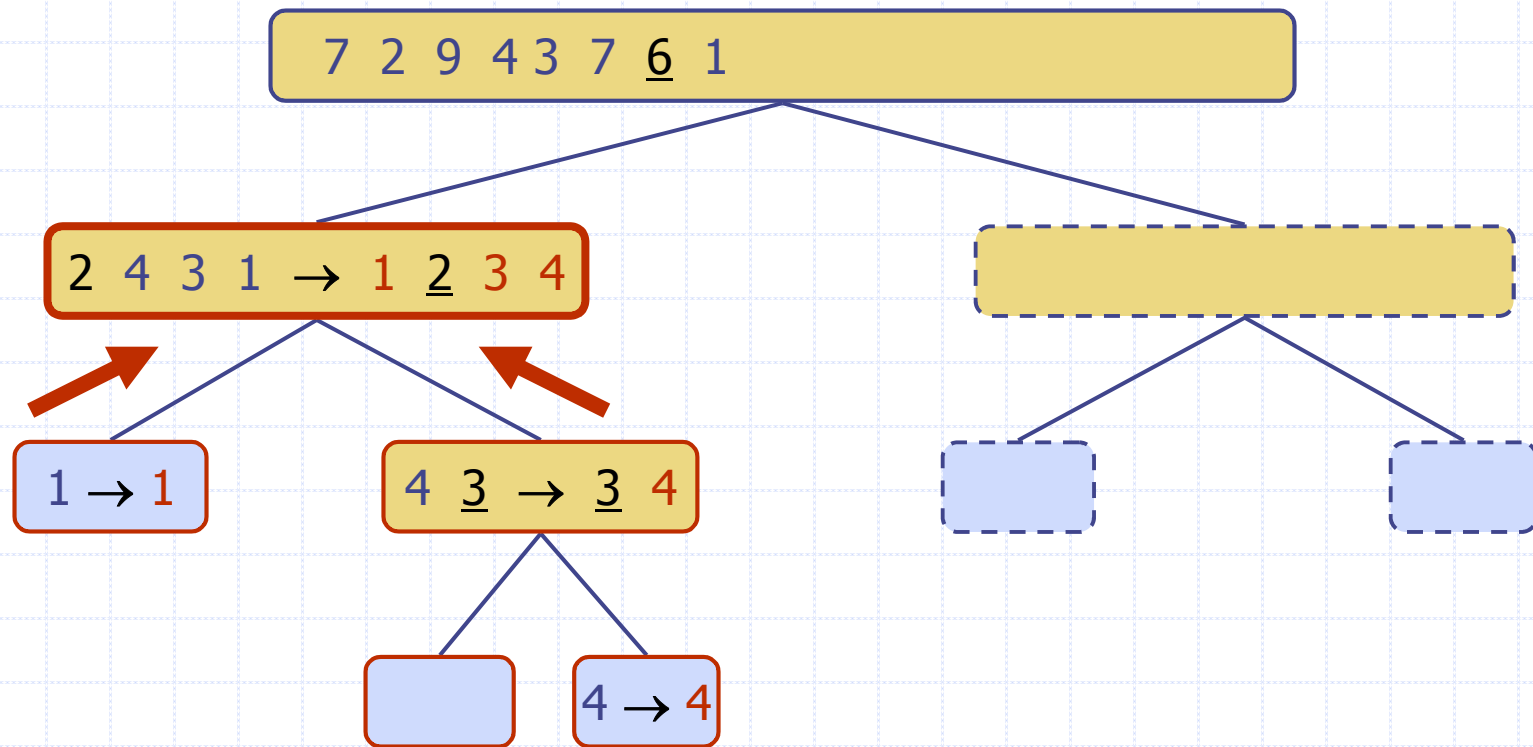
Παράδειγμα εκτέλεσης (συν.)

- ◆ Τμηματοποίηση , αναδρομική κλήση, base case



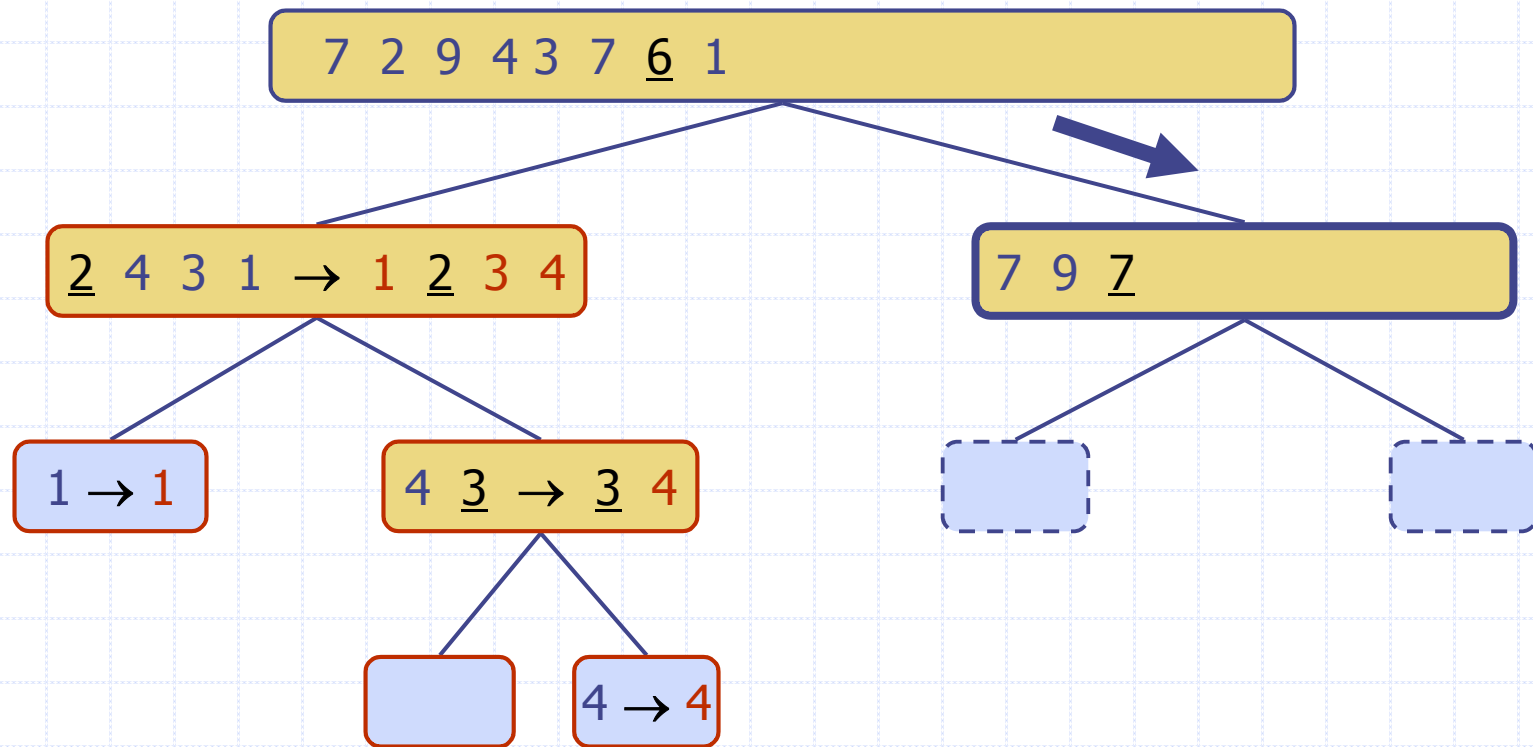
Παράδειγμα εκτέλεσης (συν.)

- ◆ Αναδρομική κλήση, ..., base case, συνένωση



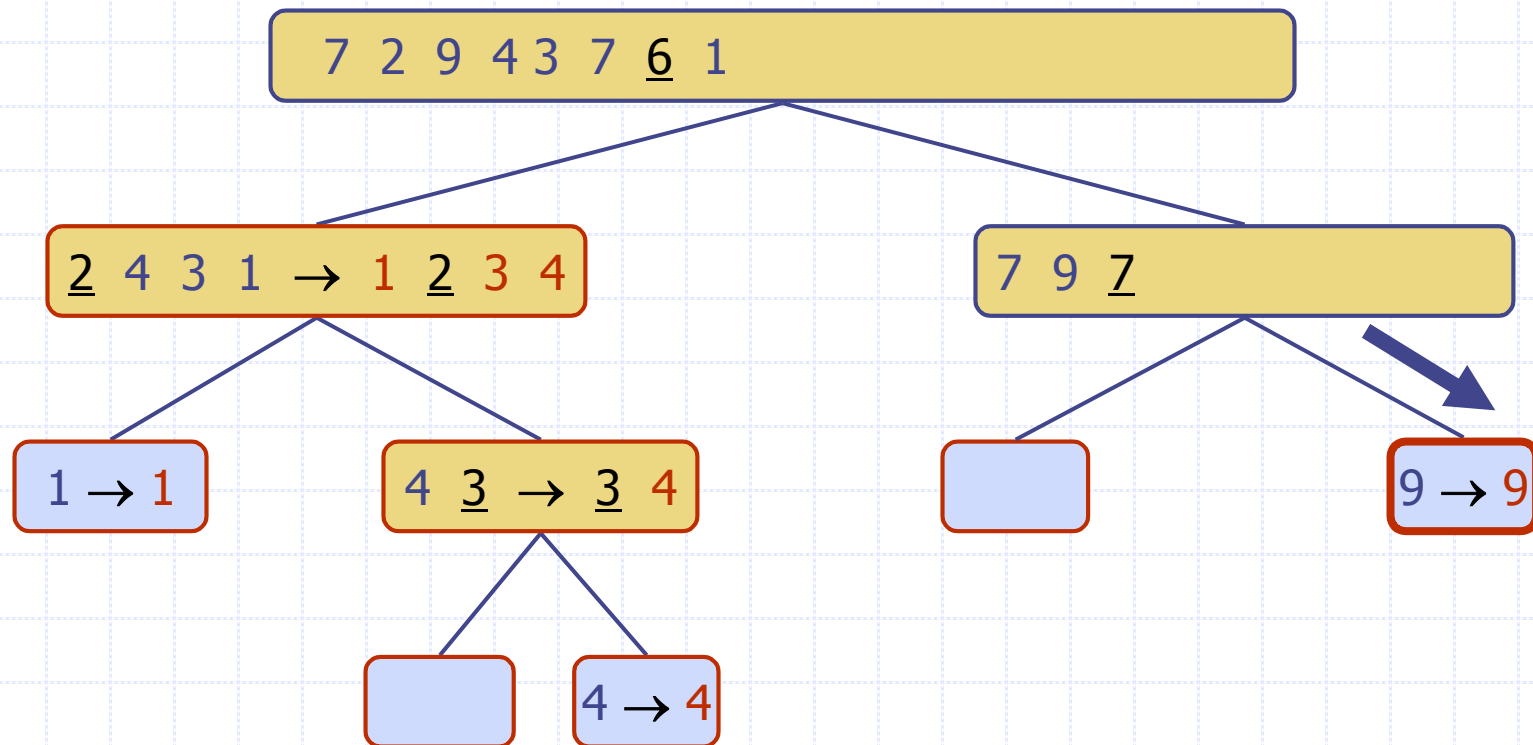
Παράδειγμα εκτέλεσης (συν.)

- ◆ Αναδρομική κλήση, επιλογή του pivot



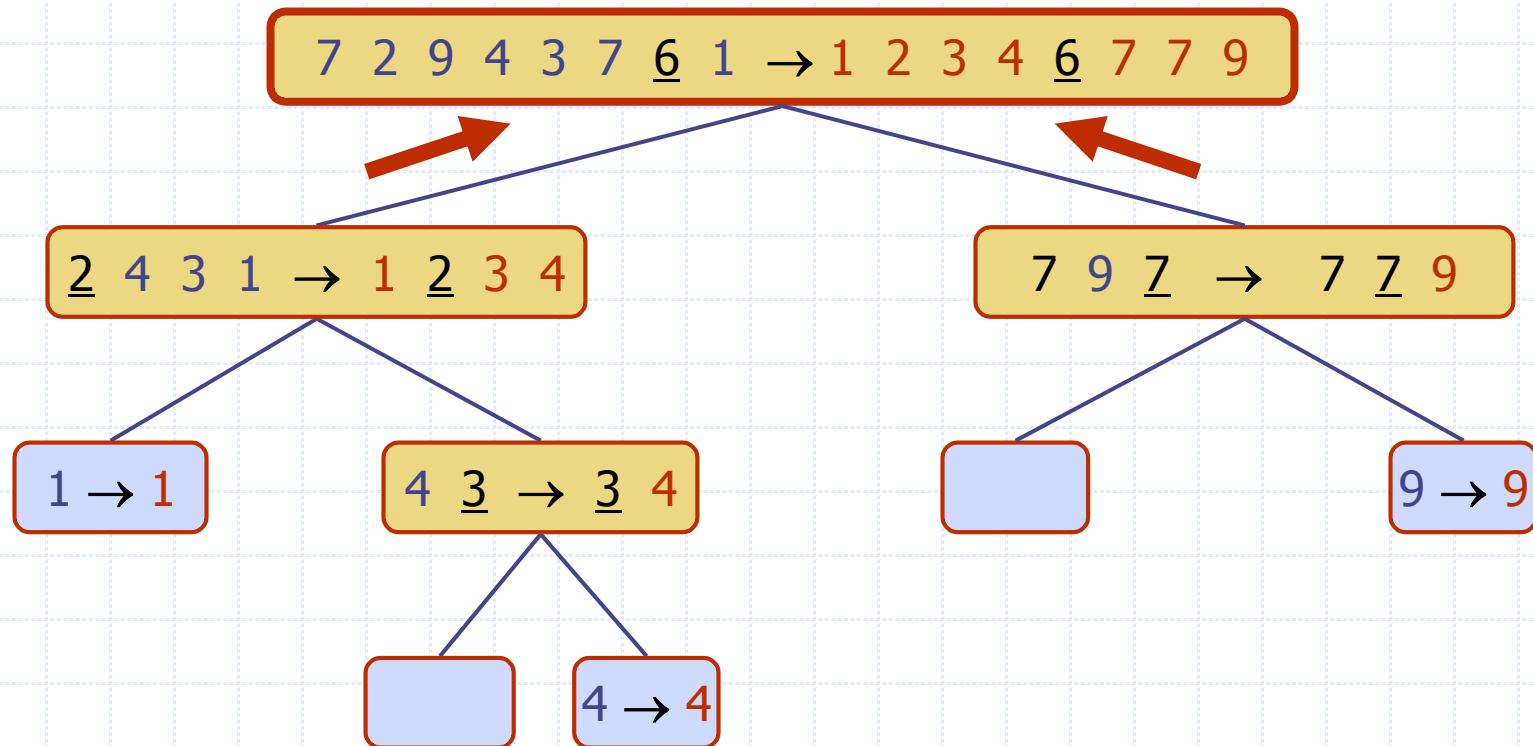
Παράδειγμα εκτέλεσης (συν.)

- ◆ Τμηματοποίηση,.. ,αναδρομική κλήση, base case



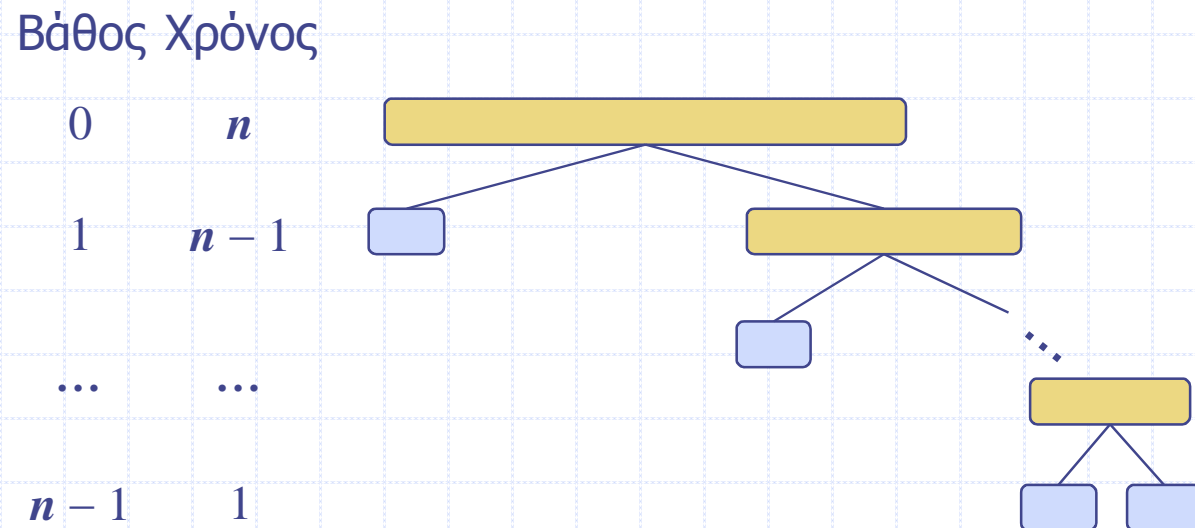
Παράδειγμα εκτέλεσης (συν.)

◆ Ένωση, ένωση



Πολυπλοκότητα χειρότερης περίπτωσης

- ◆ Η πολυπλοκότητα χειρότερης περίπτωσης για τον quick-sort συμβαίνει όταν το ρινοτ είναι το μοναδικό minimum ή maximum στοιχείο.
- ◆ Το ένα σύνολο από τα L και G έχει μέγεθος $n - 1$ και το άλλο έχει μέγεθος 0
- ◆ Ο χρόνος εκτέλεσης είναι ποσοστό του συνολικού χρόνου εκτέλεσης
$$n + (n - 1) + \dots + 2 + 1$$
- ◆ Έτσι, ο χρόνος εκτέλεσης της χειρότερης περίπτωσης είναι $O(n^2)$



Αναμενόμενος χρόνος εκτέλεσης

- ◆ Θεωρείστε μια αναδρομική κλήση του quick-sort σε μια ακολουθία μεγέθους s
 - Καλή κλήση: τα μεγέθη των L και G είναι το καθένα μικρότερο από $3s/4$
 - Άσχημη κλήση: ένα από τα L και G έχει μέγεθος μεγαλύτερο από $3s/4$
- ◆ Μια κλήση είναι καλή με πιθανότητα $1/2$
- ◆ **Probabilistic Fact:** Ο αναμενόμενος αριθμός ρίψης κερμάτων για να φέρουμε k κορώνες είναι $2k$
- ◆ Έτσι, για έναν κόμβο βάθους i , αναμένουμε ότι:
 - $i/2$ κόμβοι πατέρες σχετίζονται με καλές κλήσεις
 - Το μέγεθος της ακολουθίας εισόδου για την παρούσα κλήση είναι το πολύ $(3/4)^{i/2}n$
- ◆ Έτσι, έχουμε
 - Για έναν κόμβο βάθους $2\log_{4/3}n$, το αναμενόμενο μέγεθος της ακολουθίας εισόδου είναι ένα
 - Το αναμενόμενο ύψος του δέντρου quick-sort είναι $O(\log n)$
- ◆ Η συνολική ποσότητα δουλειάς που γίνεται στους κόμβους του ίδιου βάθους του δέντρου quick-sort είναι $O(n)$
- ◆ Έτσι ο αναμενόμενος χρόνος εκτέλεσης του quick-sort είναι $O(n \log n)$

In-Place Quick-Sort

- ◆ Ο Quick-sort μπορεί να υλοποιηθεί για να τρέχει in-place
- ◆ Στο βήμα τμηματοποίησης, εκτελούμε λειτουργίες αντικατάστασης για να επανατακτοποιήσουμε τα στοιχεία της ακολουθίας εισόδου έτσι ώστε
 - Τα στοιχεία που είναι μικρότερα από το ρινοτ έχουν τάξη μικρότερη από h
 - Τα στοιχεία που είναι ίσα με το ρινοτ έχουν τάξη ανάμεσα στο h και στο k
 - Τα στοιχεία που είναι μεγαλύτερα από το ρινοτ έχουν τάξη μεγαλύτερη από k
- ◆ Οι αναδρομικές κλήσεις θεωρούν ότι
 - Στοιχεία με τάξη μικρότερη από h
 - Στοιχεία με τάξη μεγαλύτερη από k

```
Algorithm inPlaceQuickSort( $S, l, r$ )  
Input sequence  $S$ , ranks  $l$  and  $r$   
Output sequence  $S$  with the  
elements of rank between  $l$  and  $r$   
rearranged in increasing order  
if  $l \geq r$   
    return  
 $i \leftarrow$  a random integer between  $l$  and  $r$   
 $x \leftarrow S.elemAtRank(i)$   
 $(h, k) \leftarrow inPlacePartition(x)$   
inPlaceQuickSort( $S, l, h - 1$ )  
inPlaceQuickSort( $S, k + 1, r$ )
```

Συνόψιση αλγόριθμων ταξινόμησης

Αλγόριθμος	Πολυπλοκότητα	Σημειώσεις
selection-sort	$O(n^2)$	<ul style="list-style-type: none">◆ in-place◆ αργός (καλός για μικρά δεδομένα)
insertion-sort	$O(n^2)$	<ul style="list-style-type: none">◆ in-place◆ αργός (καλός για μικρά δεδομένα)
quick-sort	$O(n \log n)$ expected	<ul style="list-style-type: none">◆ in-place, randomized◆ γρηγορότερος (καλός για μεγάλα δεδομένα)
heap-sort	$O(n \log n)$	<ul style="list-style-type: none">◆ in-place◆ γρήγορος (καλός για μεγάλα δεδομένα)
merge-sort	$O(n \log n)$	<ul style="list-style-type: none">◆ ακολουθιακή πρόσβαση δεδομένων◆ Πολύ γρήγορος (καλός για τεράστια δεδομένα)