ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Δομές Δεδομένων

Ιωάννης Γ. Τόλλης
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» 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.

Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά, Μη Εμπορική Χρήση, Όχι Παράγωγο Έργο 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

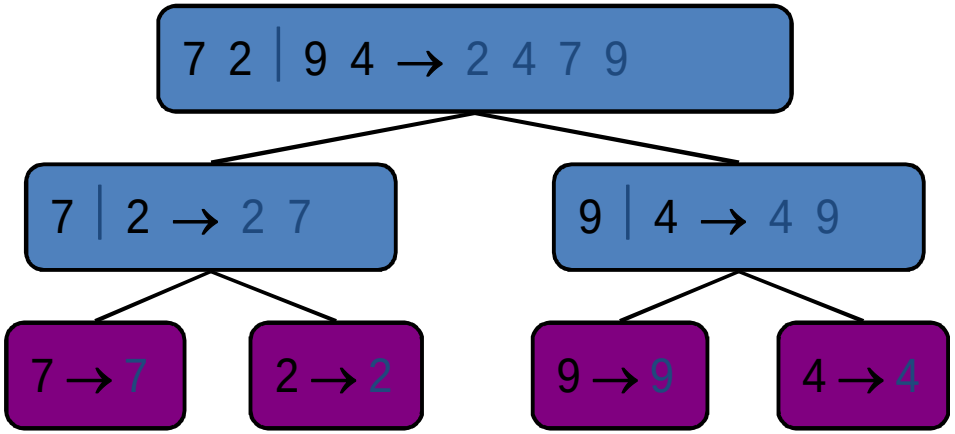
[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

- Ως **Μη Εμπορική** ορίζεται η χρήση:
  - που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
  - που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
  - που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

- Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

.

Merge Sort (Ταξινόμηση με συγχώνευση)



Κύρια σημεία για μελέτη

- Το παράδειγμα του «διαίρει και βασίλευε» (§4.1.1)
- Merge-sort (§4.1.1)
  - Ο αλγόριθμος
  - Συγχωνεύοντας δύο ταξινομημένες ακολουθίες
  - Το δέντρο Merge-sort
  - Παράδειγμα εκτέλεσης
  - Ανάλυση
- Γενική συγχώνευση και λειτουργίες συνόλων (§4.2.1)
- Συνόψιση αλγόριθμων ταξινόμησης (§4.2.1)

«Διαίρει και Βασίλευε»

- Η τεχνική divide and conquer είναι μια γενική αλγοριθμική τεχνική σχεδιασμού:
  - Divide: Το σύνολο εισόδου S, χωρίζεται σε δύο υποσύνολα S_1 και S_2
  - Recur: Επίλυση των επιμέρους προβλημάτων που σχετίζονται με τα S_1 και S_2
  - Conquer: Συνδυασμός των λύσεων S_1 και S_2 σε μια λύση για το S
- Η base case για την αναδρομή είναι επιμέρους προβλήματα μεγέθους 0 ή 1
- Ο Merge-sort είναι ένας αλγόριθμος ταξινόμησης που βασίζεται στην τεχνική «διαίρει και βασίλευε»
- Όπως και ο heap-sort
  - Χρησιμοποιεί έναν συγκριτή
  - Έχει πολυπλοκότητα $O(n \log n)$
- Σε αντίθεση με τον heap-sort
  - Δεν χρησιμοποιεί βοηθητική ουρά προτεραιότητας
  - Η πρόσβαση στα δεδομένα γίνεται με τρόπο ακολουθιακό (κατάλληλος για αποθήκευση δεδομένων στον δίσκο)

Merge-Sort

- Ο Merge-sort με μια ακολουθία εισόδου S με n στοιχεία αποτελείται από τρία βήματα:
  - Divide: Το σύνολο εισόδου S, χωρίζεται σε δύο υποσύνολα S_1 και S_2 με $n/2$ στοιχεία το καθένα
  - Recur: Τα S_1 και S_2 ταξινομούνται αναδρομικά
  - Conquer: Συγχώνευση των S_1 και S_2 σε μια ταξινομημένη ακολουθία S

Algorithm *mergeSort*(S, C)
  Input sequence S with n elements, comparator C
  Output sequence S sorted according to C
  if $S.size() > 1$
    (S_1, S_2) ← *partition*(S, $n/2$)
    mergeSort(S_1, C)
    mergeSort(S_2, C)
    S ← *merge*(S_1, S_2)

Συγχωνεύοντας δύο ταξινομημένες ακολουθίες

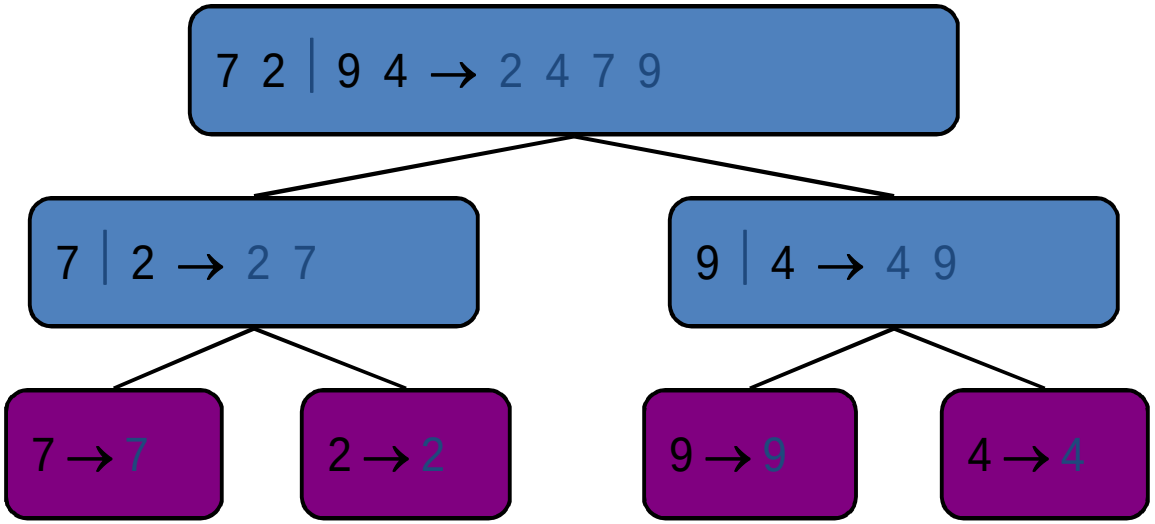
- Το βήμα conquer του merge-sort αποτελείται από την συγχώνευση δύο ακολουθιών ***A και B*** σε μια ταξινομημένη ακολουθία ***S*** που είναι η ένωση των ***A και B***
- Η Συγχώνευση δύο ταξινομημένων ακολουθιών η καθεμία με $n/2$ στοιχεία και η υλοποίηση τους με διπλά συνδεδεμένη λίστα, παίρνει χρόνο $O(n)$.

```
Algorithm merge(A, B)
    Input sequences A and B with
        n/2 elements each
    Output sorted sequence of A ∪ B

    S ← empty sequence
    while ¬A.isEmpty() ∧ ¬B.isEmpty()
        if A.first().element() < B.first().element()
            S.insertLast(A.remove(A.first()))
        else
            S.insertLast(B.remove(B.first()))
    while ¬A.isEmpty()
        S.insertLast(A.remove(A.first()))
    while ¬B.isEmpty()
        S.insertLast(B.remove(B.first()))
    return S
```

Το δέντρο Merge-Sort

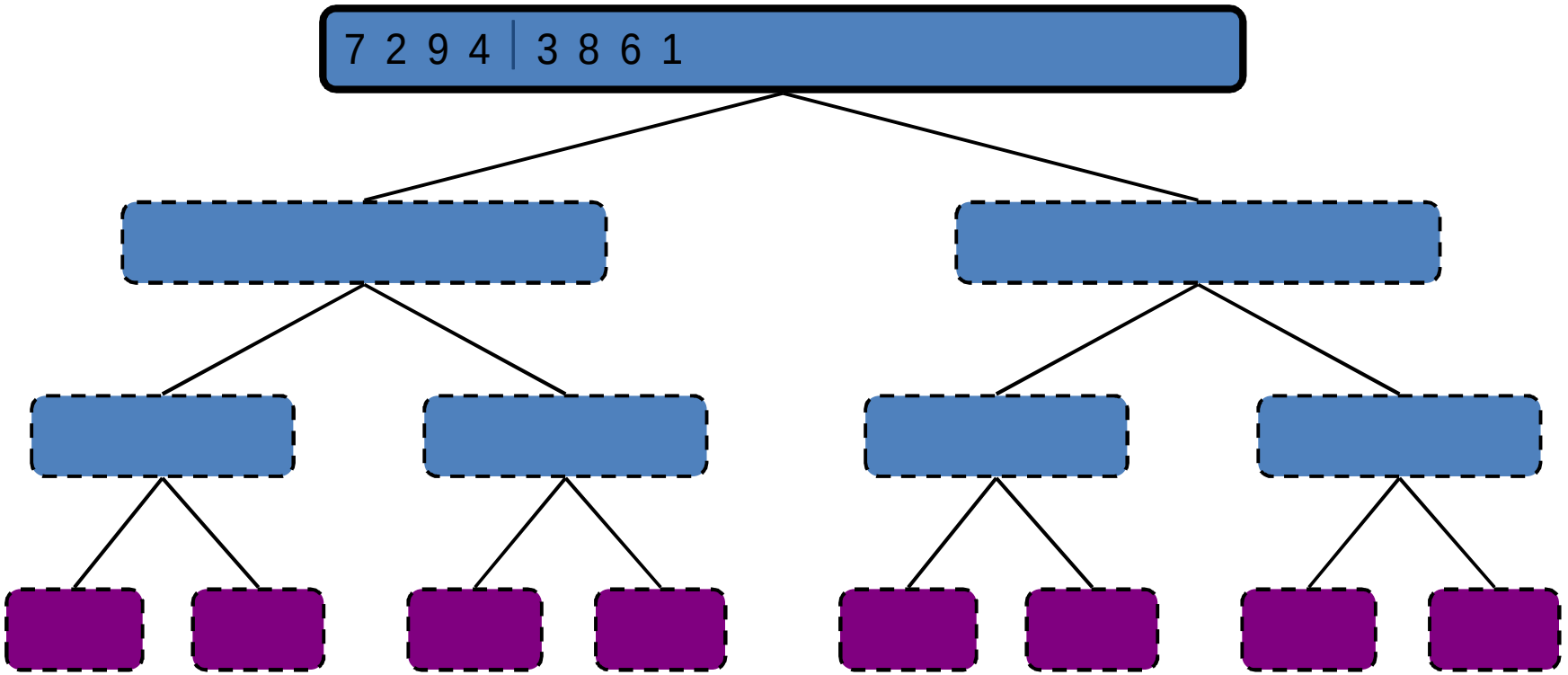
- Η εκτέλεση του merge-sort απεικονίζεται με ένα δυαδικό δέντρο
  - Κάθε κόμβος αναπαριστά μια αναδρομική κλήση του merge-sort και αποθηκεύει
    - Την αταξινόμητη ακολουθία και το partition της
    - Την ταξινομημένη ακολουθία στο τέλος της εκτέλεσης
  - Η ρίζα είναι η αρχική κλήση
  - Τα φύλλα είναι κλήσεις σε υποακολουθίες μεγέθους 0 ή 1



Merge Sort

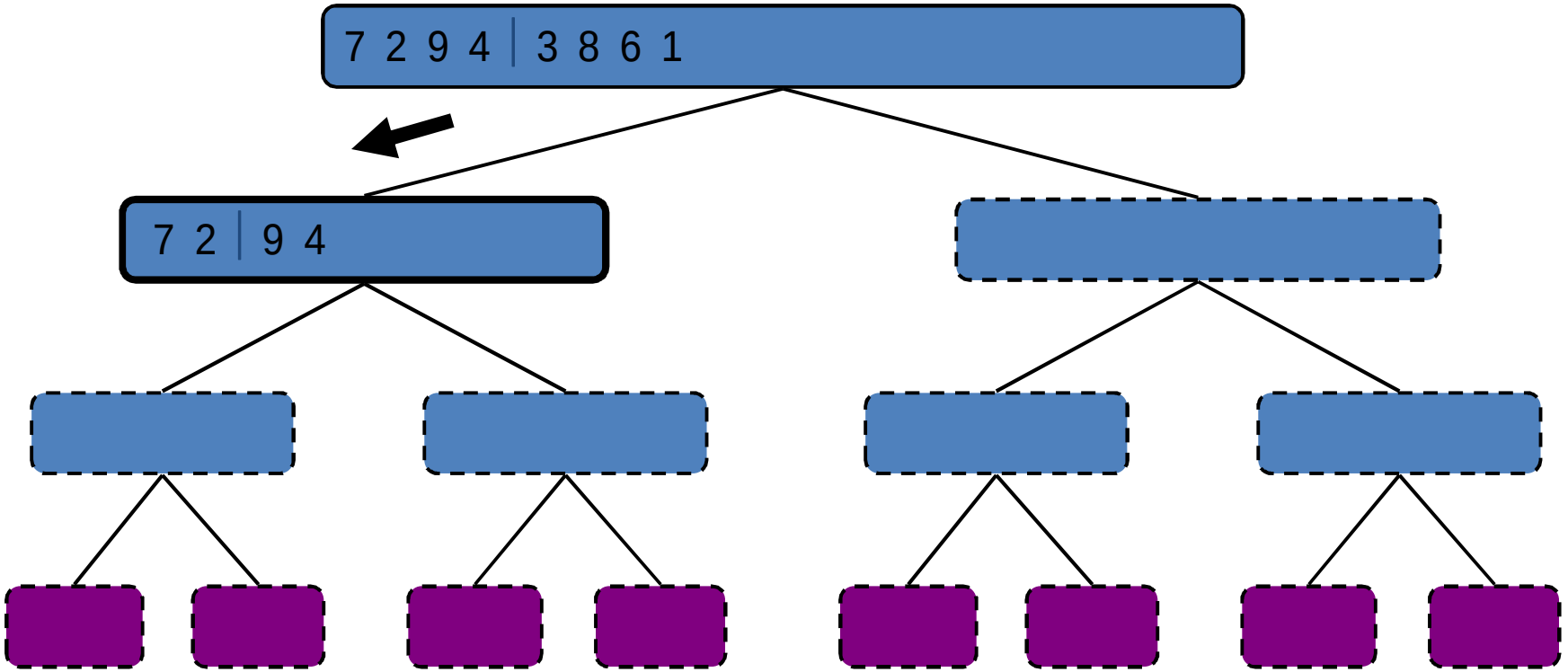
Παράδειγμα εκτέλεσης

- Τμήμα (Partition)



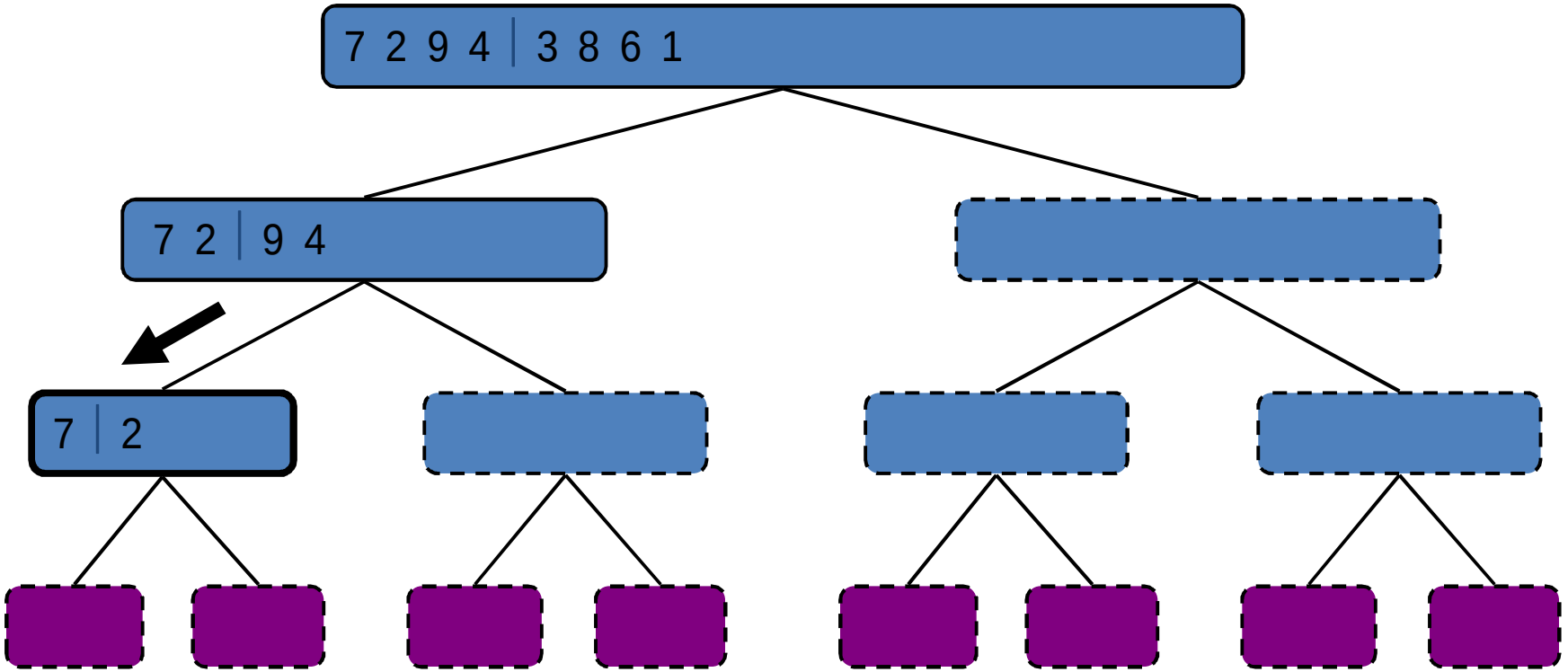
Παράδειγμα εκτέλεσης (συν.)

- Αναδρομική κλήση, partition



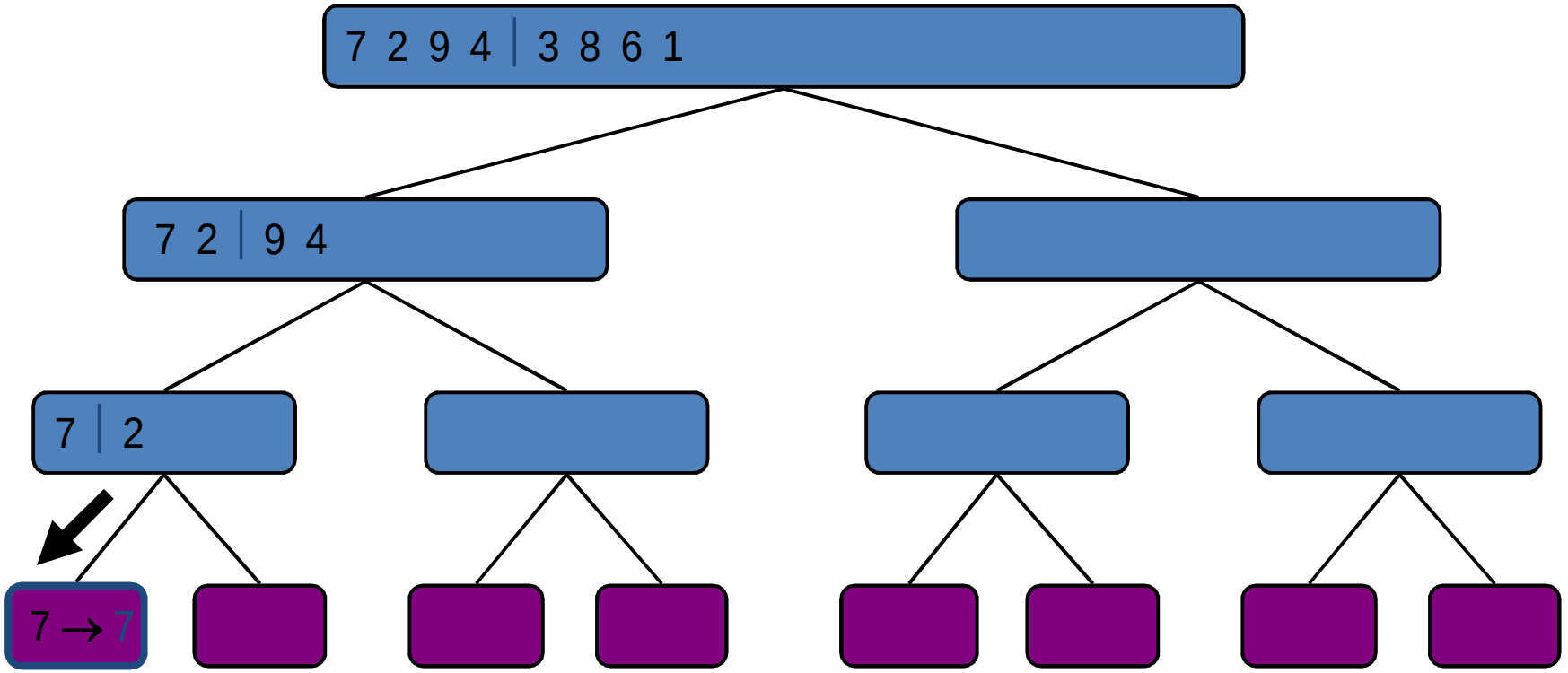
Παράδειγμα εκτέλεσης (συν.)

- Αναδρομική κλήση, partition



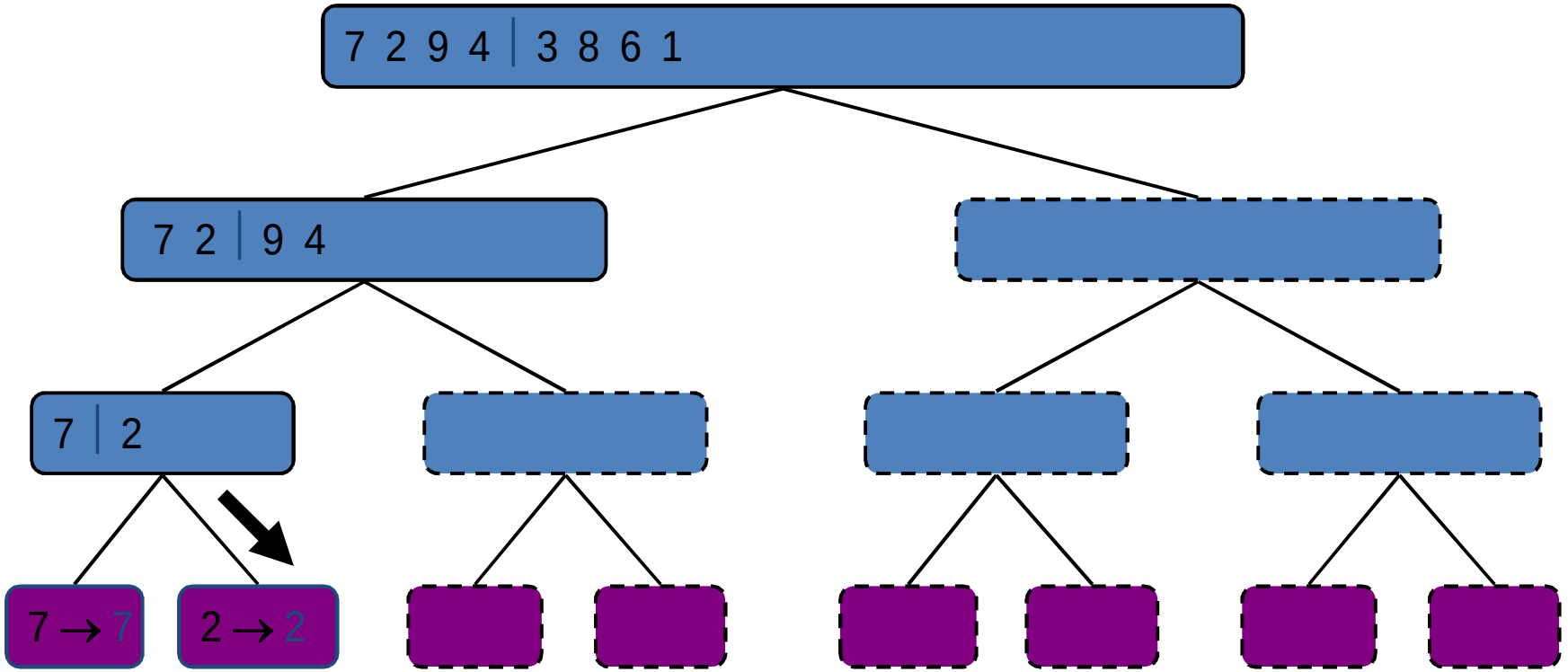
Παράδειγμα εκτέλεσης (συν.)

- Αναδρομική κλήση, base case



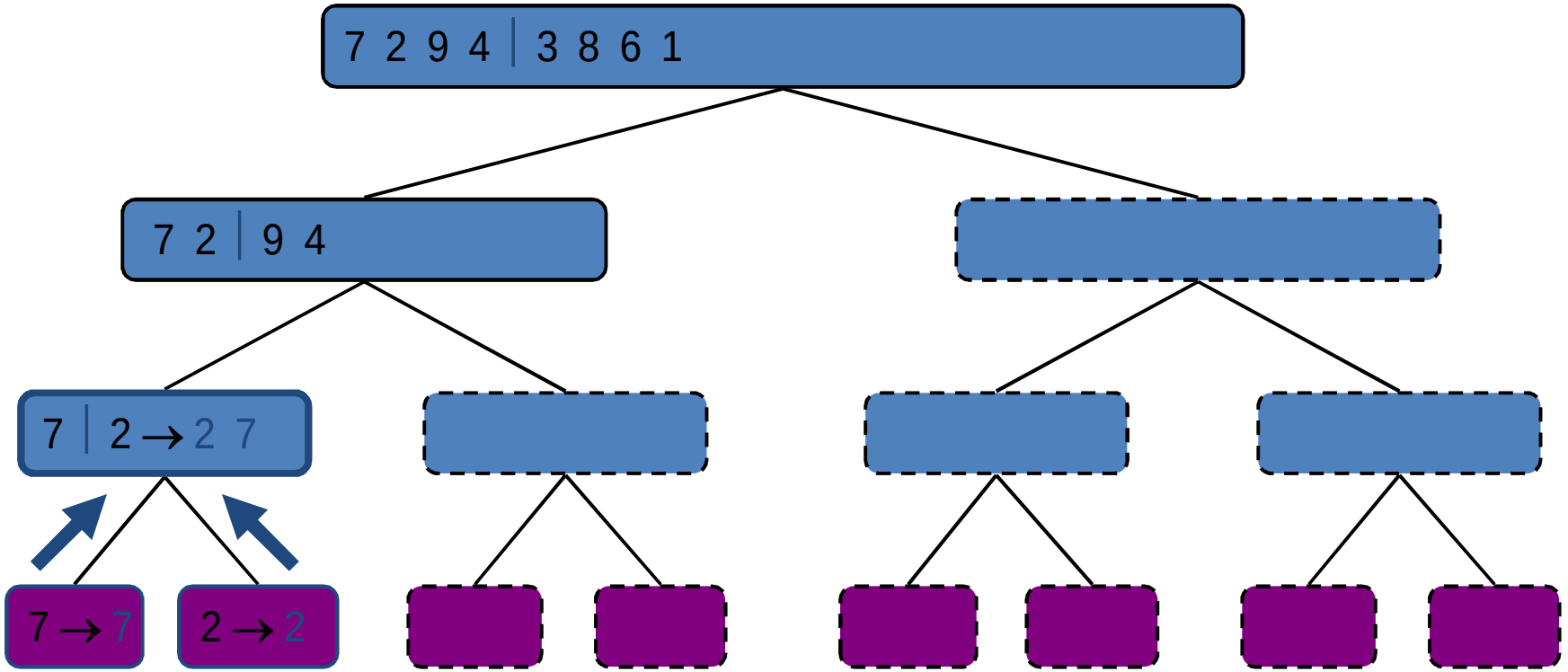
Παράδειγμα εκτέλεσης (συν.)

- Αναδρομική κλήση, base case



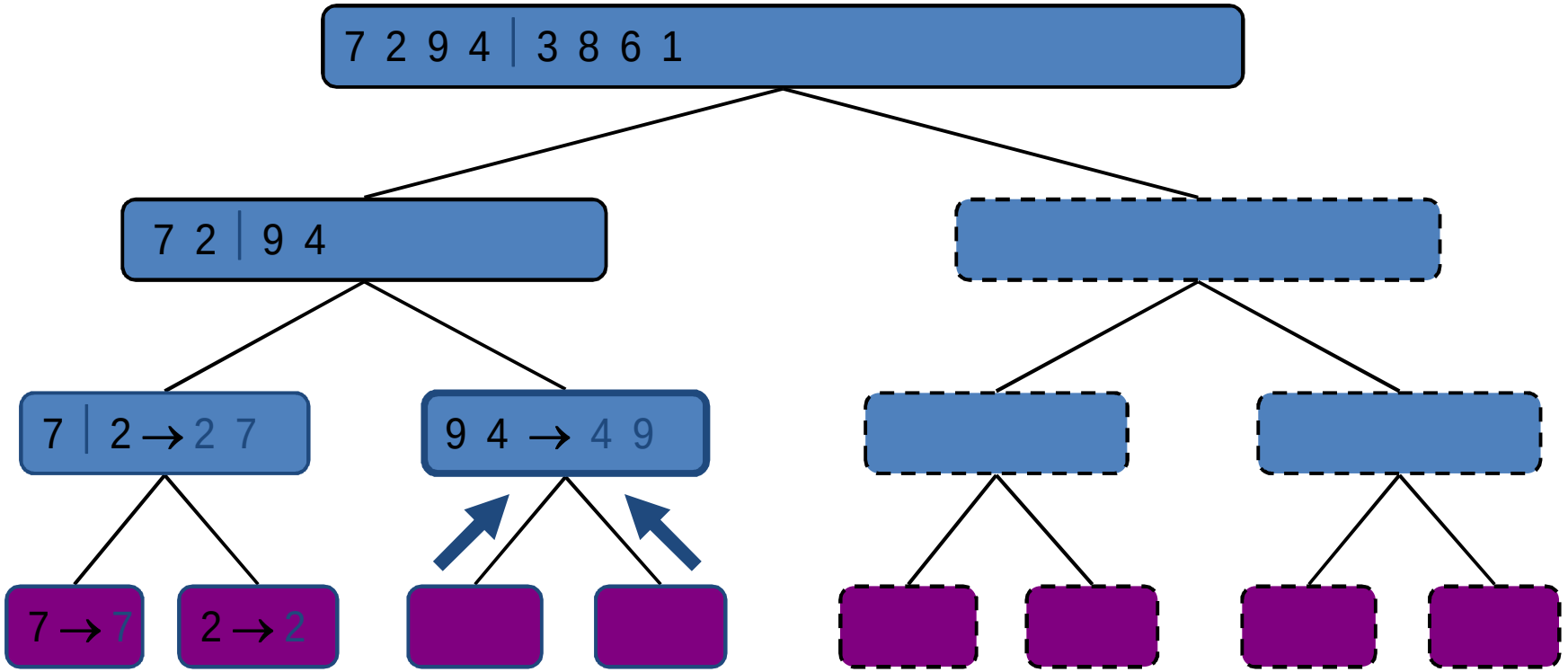
Παράδειγμα εκτέλεσης (συν.)

- Συγχώνευση



Παράδειγμα εκτέλεσης (συν.)

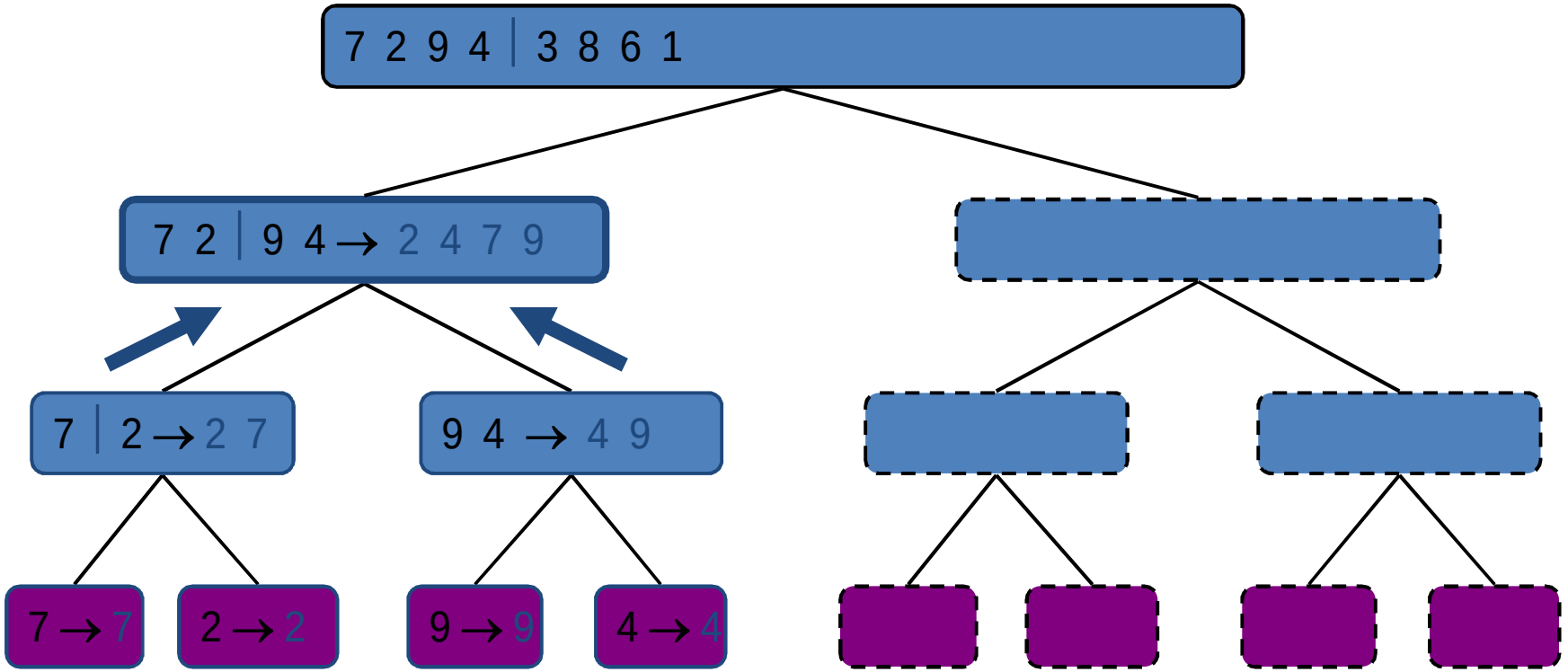
- Αναδρομική κλήση, …, base case, συγχώνευση



Merge Sort

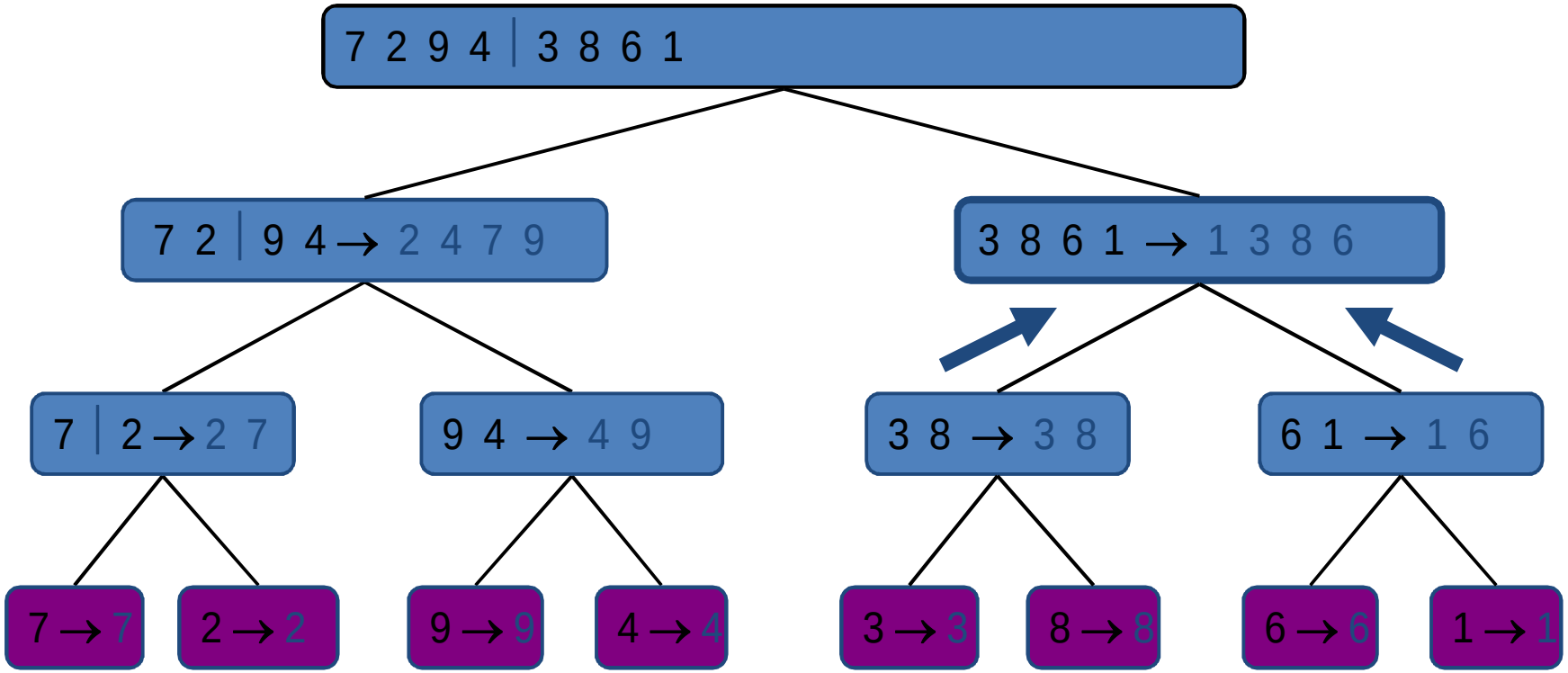
Παράδειγμα εκτέλεσης (συν.)

- Συγχώνευση



Παράδειγμα εκτέλεσης (συν.)

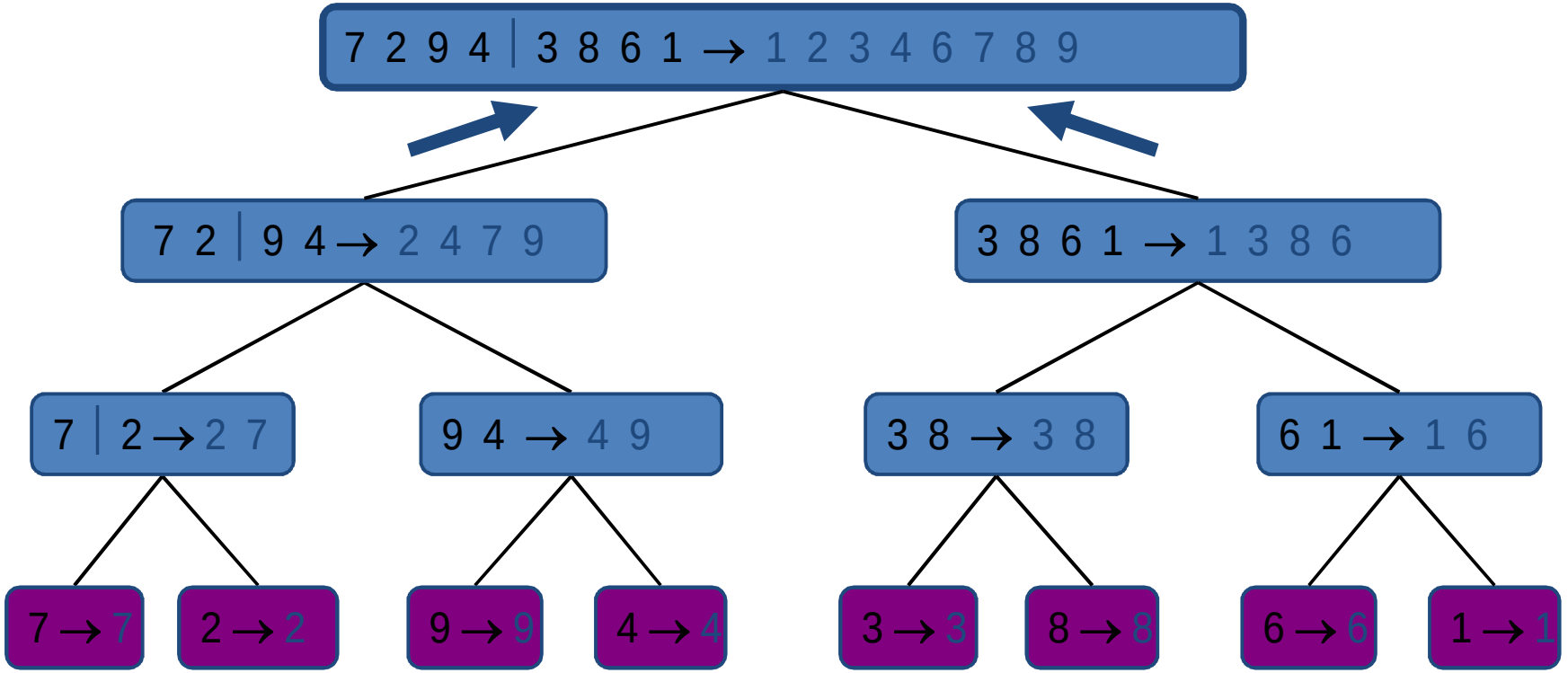
- Αναδρομική κλήση, …,συγχώνευση,συγχώνευση



Merge Sort

Παράδειγμα εκτέλεσης (συν.)

- Συγχώνευση

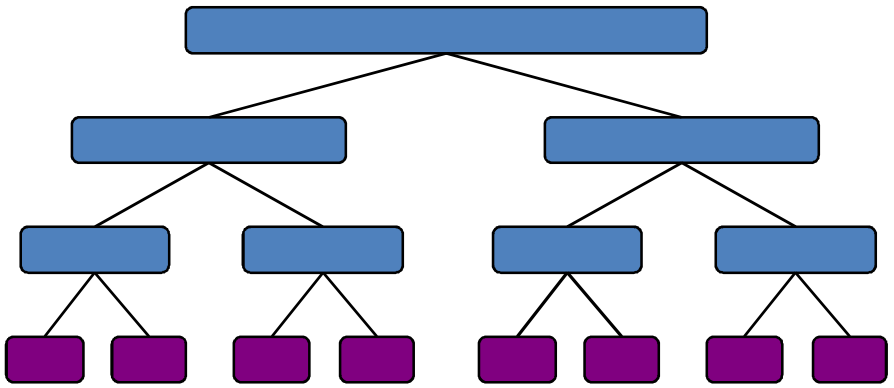


Merge Sort

Ανάλυση του Merge-Sort

- Το ύψος h του merge-sort δέντρου είναι $O(\log n)$
  - Σε κάθε αναδρομική κλήση χωρίζουμε την ακολουθία στο μισό,
- Η συνολική δουλειά που γίνεται στους κόμβους βάθους i είναι $O(n)$
  - Τμηματοποιούμε και συγχωνεύουμε 2^i ακολουθίες μεγέθους $n/2^i$
  - Γίνονται 2^{i+1} αναδρομικές κλήσεις
- Έτσι , ο συνολικός χρόνος εκτέλεσης του merge-sort είναι $O(n \log n)$

βάθος	#ακολ.	μέγεθος
0	1	n
1	2	$n/2$
i	2^i	$n/2^i$
…	…	…



Merge Sort

Γενική Συγχώνευση

- Γενική συγχώνευση δύο ταξινομημένων ακολουθιών ***A και B***
- Πρότυπη μέθοδος genericMerge
- Βοηθητικοί μέθοδοι
  - aIsLess
  - bIsLess
  - bothAreEqual
- Τρέχει σε χρόνο $O(n_A + n_B)$ δεδομένου ότι οι βοηθητικές μέθοδοι τρέχουν σε χρόνο $O(1)$

```
Algorithm genericMerge(A, B)
    S ← empty sequence
    while ¬A.isEmpty() ∧ ¬B.isEmpty()
        a ← A.first().element(); b ← B.first().element()
        if a < b
            aIsLess(a, S); A.remove(A.first())
        else if b < a
            bIsLess(b, S); B.remove(B.first())
        else { b = a }
            bothAreEqual(a, b, S)
            A.remove(A.first()); B.remove(B.first())
    while ¬A.isEmpty()
        aIsLess(a, S); A.remove(A.first())
    while ¬B.isEmpty()
        bIsLess(b, S); B.remove(B.first())
    return S
```

Πράξεις συνόλων

- Αναπαριστούμε ένα σύνολο με την ταξινομημένη ακολουθία των στοιχείων του
- Με την εξειδίκευση των βοηθητικών μεθόδων ο generic merge αλγόριθμος μπορεί να χρησιμοποιηθεί για βασικές πράξεις συνόλων:
  - Ένωση
  - Τομή
  - Διαφορά
- Ό χρόνος εκτέλεσης για μια πράξη είναι $O(n_A + n_B)$

- Ένωση:
  - ***aIsLess(a, S)***
    S.insertFirst(a)
  - ***bIsLess(b, S)***
    S.insertLast(b)
  - ***bothAreEqual(a, b, S)***
    S. insertLast(a)
- Τομή:
  - ***aIsLess(a, S)***
    { ***do nothing*** }
  - ***bIsLess(b, S)***
    { ***do nothing*** }
  - ***bothAreEqual(a, b, S)***
    S. insertLast(a)

Συνόψιση αλγόριθμων ταξινόμησης

Αλγόριθμος	Χρόνος	Σημειώσεις
selection-sort	$O(n^2)$	◆ Αργός ◆ in-place ◆ για μικρά δεδομένα (< 1K)
insertion-sort	$O(n^2)$	◆ Αργός ◆ in-place ◆ για μικρά δεδομένα (< 1K)
heap-sort	$O(n \log n)$	◆ Γρήγορος ◆ in-place ◆ για μεγάλα δεδομένα(1K — 1M)
merge-sort	$O(n \log n)$	◆ Γρήγορος ◆ sequential data access ◆ για πολύ μεγάλα δεδομένα (> 1M)

Τέλος Ενότητας