Δομές Δεδομένων

Μέρος 2ο: Δυαδική Αναζήτηση, Ταξινόμηση

Εξάμηνο Σπουδών: 6ο

Κωδικός Μαθήματος: 681

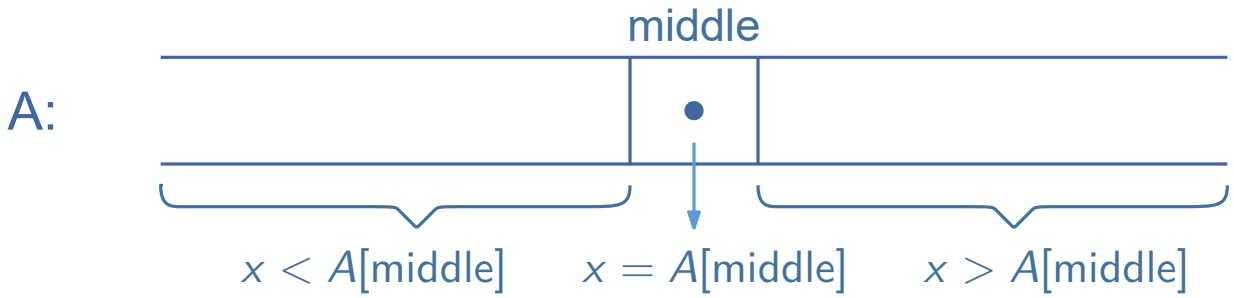
Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Δυαδική Αναζήτηση

Binary search

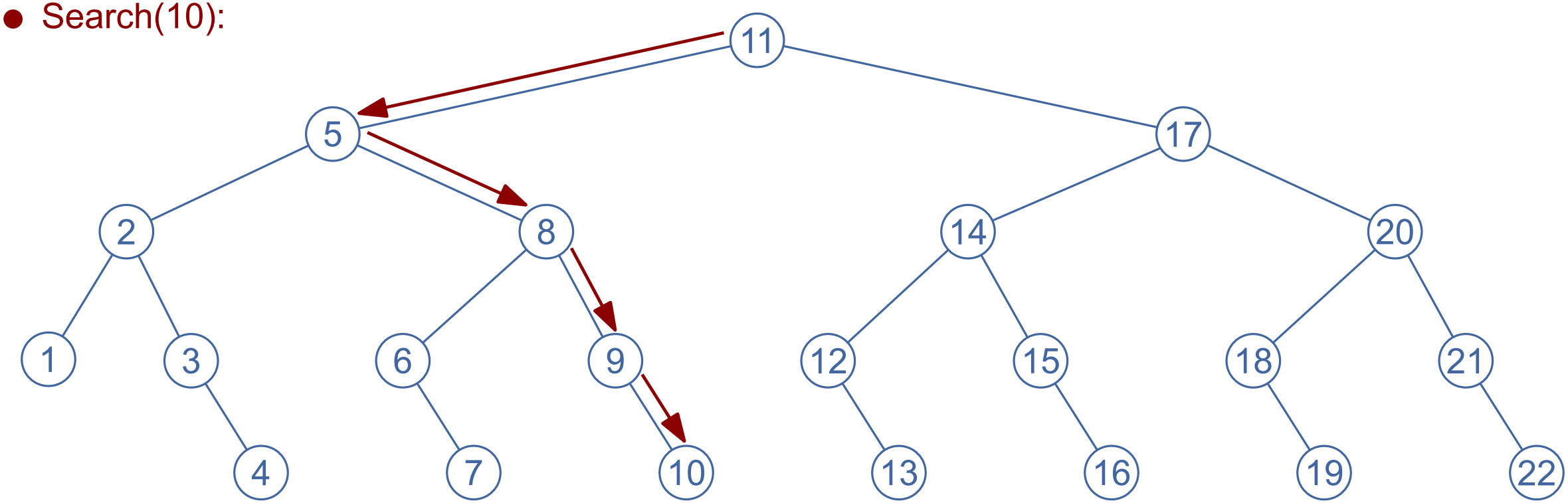
- Είσοδος: Ένας διατεταγμένος πίνακας n στοιχείων, και
  ένα στοιχείο x προς αναζήτηση
- Εξοδος: Η θέση του στοιχείου x στον πίνακα (-1, αν το στοιχείο δεν ανήκει στον πίνακα)
- Μέθοδος: Σύγκρινε το στοιχείο x με το μεσαίο στοιχείο του πίνακα
  - $=$: το στοιχείο βρέθηκε
  - $<$: αναζήτησε το στοιχείο x στο αριστερό τμήμα του πίνακα
  - $>$: αναζήτησε το στοιχείο x στο δεξί τμήμα του πίνακα



Δυαδική Αναζήτηση

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

- Search(10):



Δυαδική Αναζήτηση

```
/**
 * A C++ implementation of binary search
 */
template <typename T>
int binarySearch(T array[], int left, int right, T x)
{
    if (right >= left) {
        int middle = left + (right - left) / 2;

        if (array[middle] == x)
            return middle;

        if (array[middle] > x)
            return binarySearch(array, left, middle - 1, x);
        else
          return binarySearch(array, middle + 1, right, x);
    }
    return -1;
}
```

Ανάλυση Δυαδικής Αναζήτησης

- Θεώρημα: Η χρονική πολυπλοκότητα, σε πλήθος συγκρίσεων, της μεθόδου binarySearch δίνεται από την αναδρομική σχέση:

$$T[n] \leq T[\lfloor \tfrac{n}{2} \rfloor] + 1 \qquad T[1] = 1$$

η οποία ως έχει λύση:

$$T[n] \leq \lfloor \log n \rfloor + 1 = \mathcal{O}(\log n)$$

Απόδειξη:

Υποθέτουμε ότι το n είναι δύναμη του 2

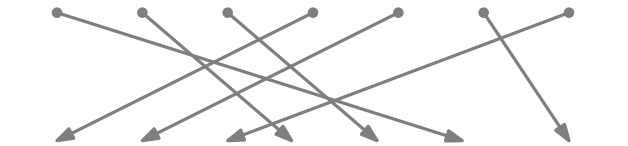
$\Rightarrow n = 2^k \iff k = \log n$

$$T[n] \leq T[\tfrac{n}{2}] + 1$$
$$T[\tfrac{n}{2}] \leq T[\tfrac{n}{4}] + 1$$
$$\cdots$$
$$T[\tfrac{n}{2^{k-1}}] \leq T[\tfrac{n}{2^k}] + 1$$

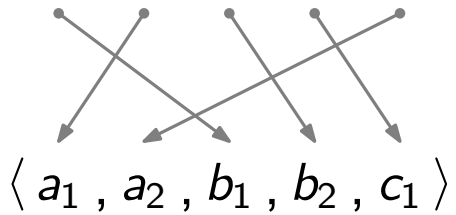
$$T[n] \leq T[\tfrac{n}{2^k}] + k = 1 + \log n = \mathcal{O}(\log n)$$

Ταξινόμηση
Sorting

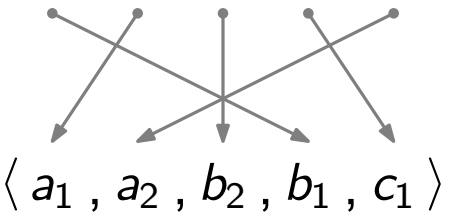
- Είσοδος: Ένας πίνακας n στοιχείων.
- Εξοδος: Μια ταξινόμηση των στοιχείων του πίνακα (από το μικρότερο προς το μεγαλύτερο)
- Παράδειγμα: $\langle 6, 4, 5, 1, 2, 7, 3 \rangle$



- Σταθερή ταξινόμηση: $\langle b_1, a_1, b_2, c_1, a_2 \rangle$
Stable



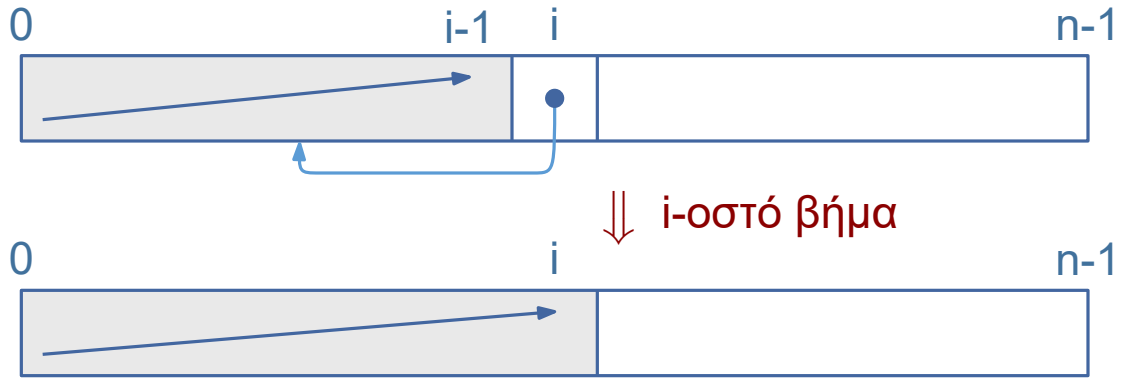
- Ασταθής ταξινόμηση: $\langle b_1, a_1, b_2, c_1, a_2 \rangle$
unstable



Ταξινόμηση εισαγωγής

Comparison based sorting algorithms

- Ταξινόμηση εισαγωγής
  Insertion sort




```
template <typename T>
void insertionSort(T array[], const int n) {
  for (int i=1; i<n; i++) {
    int key = array[i];
    int j = i;
    while (key < array[j-1] && j > 0) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = key;
  }
}
```

https://www.youtube.com/watch?v=OGzPmgsI-pQ

Ανάλυση Πολυπλοκότητας

- Χειρότερη περίπτωση: Το i-οστό στοιχείο συγκρίνεται με όλα τα $i-1$ στοιχεία που προηγούνται

  Συνολικά: $1+2+\ldots+(n-1)=\frac{n(n-1)}{2}=\mathcal{O}(n^2)$ συγκρίσεις

  Ο πίνακας είναι ταξινομημένος κατά φθίνουσα διάταξη

- Μέση περίπτωση: Το i-οστό στοιχείο συγκρίνεται με $\frac{i-1}{2}$ από τα στοιχεία που προηγούνται

  Συνολικά: $\frac{1}{2}+\frac{2}{2}+\ldots+\frac{n-1}{2}=\frac{n(n-1)}{4}=\mathcal{O}(n^2)$ συγκρίσεις

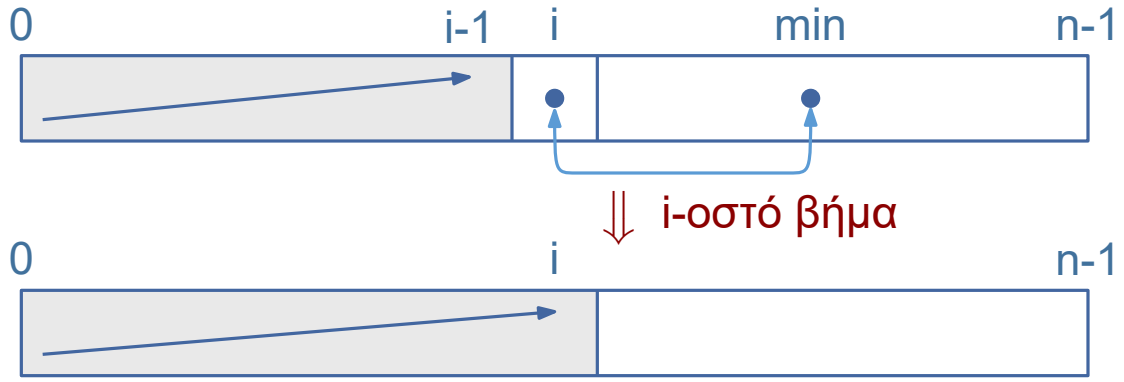
  Αναμένεται ότι το i-οστό στοιχείο είναι μεγαλύτερο από τα μισά στοιχεία του ήδη ταξινομημένου τμήματος

Η πολυπλοκότητα της ταξινόμησης εισαγωγής είναι $\Theta(n^2)$

Ταξινόμηση επιλογής

Comparison based sorting algorithms

- Ταξινόμηση επιλογής
  Selection sort




```
template <typename T>
void selectionSort(T array[], const int n) {
  for (int i = 0; i < n-1; i++) {
    // Find min in the unsorted part
    int min = i;
    for (int j = i+1; j < n; j++) {
      if (array[j] < array[min])
        min = j;
    }
    // Swap min with the i-th element
    std::swap(array[min], array[i]);
  }
}
```

https://www.youtube.com/watch?v=xWBP4lzkoyM

Ανάλυση Πολυπλοκότητας

- Το i-οστό στοιχείο συγκρίνεται πάντοτε με όλα τα $n - i$ στοιχεία που έπονται

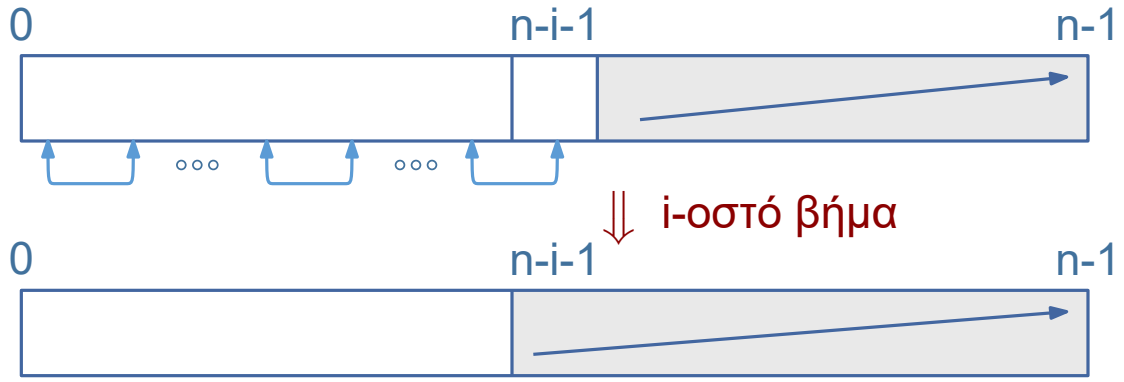
  Συνολικά: $(n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ συγκρίσεις

Η πολυπλοκότητα της ταξινόμησης επιλογής είναι $\Theta(n^2)$

Ταξινόμηση φυσαλίδας

Comparison based sorting algorithms

- Ταξινόμηση φυσαλίδας
  Bubble sort




```
template <typename T>
void bubbleSort(T array[], const int n) {
  for (int i=0; i<n-1; i++) {
    // Last i elements are already in place
    for (int j=0; j<n-i-1; j++) {
      if (array[j] > array[j+1])
        std::swap(array[j], array[j+1]);
    }
  }
}
```

https://www.youtube.com/watch?v=nmhjrI-aW5o

Ανάλυση Πολυπλοκότητας

- Η i-οστή επανάληψη απαιτεί πάντοτε $n - i$ συγκρίσεις (όμοια με την ταξινόμηση επιλογής)

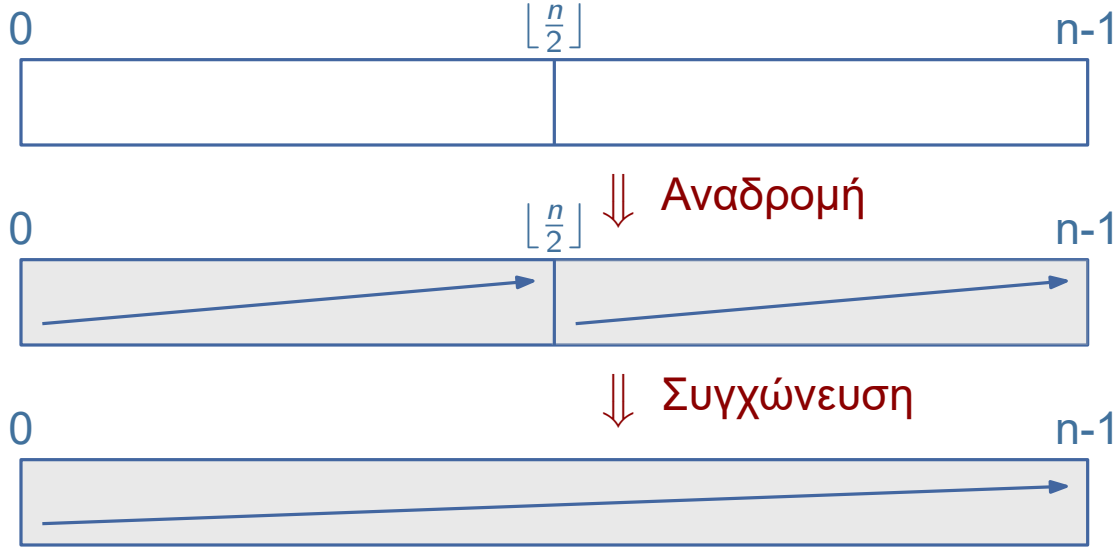
  Συνολικά: $(n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \mathcal{O}(n^2)$ συγκρίσεις

Η πολυπλοκότητα της ταξινόμησης φυσαλίδας είναι $\Theta(n^2)$

Ταξινόμηση μέσω συγχώνευσης

Comparison based sorting algorithms

- Ταξινόμηση μέσω συγχώνευσης
  Merge sort

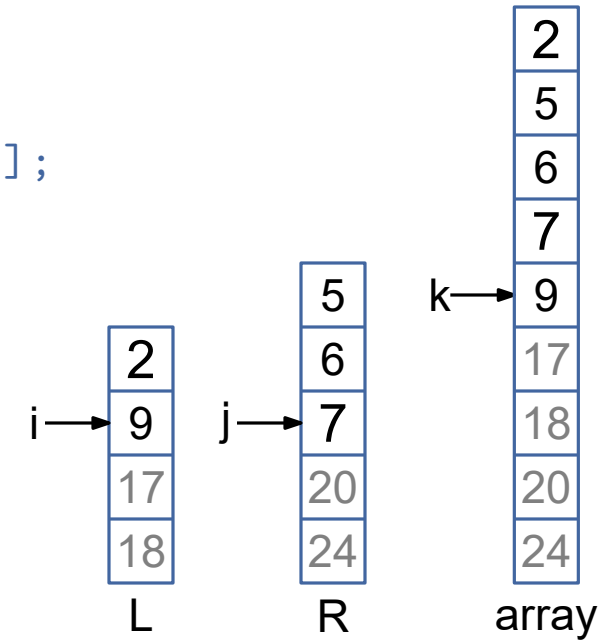



```
template <typename T>
void mergeSort(T array[], int left, int right){
    if(left >= right){
        return;//returns recursively
    }
    int middle = left + (right-left)/2;
    mergeSort(array, left, middle);
    mergeSort(array, middle+1, right);
    merge(array, left, middle, right);
}
```

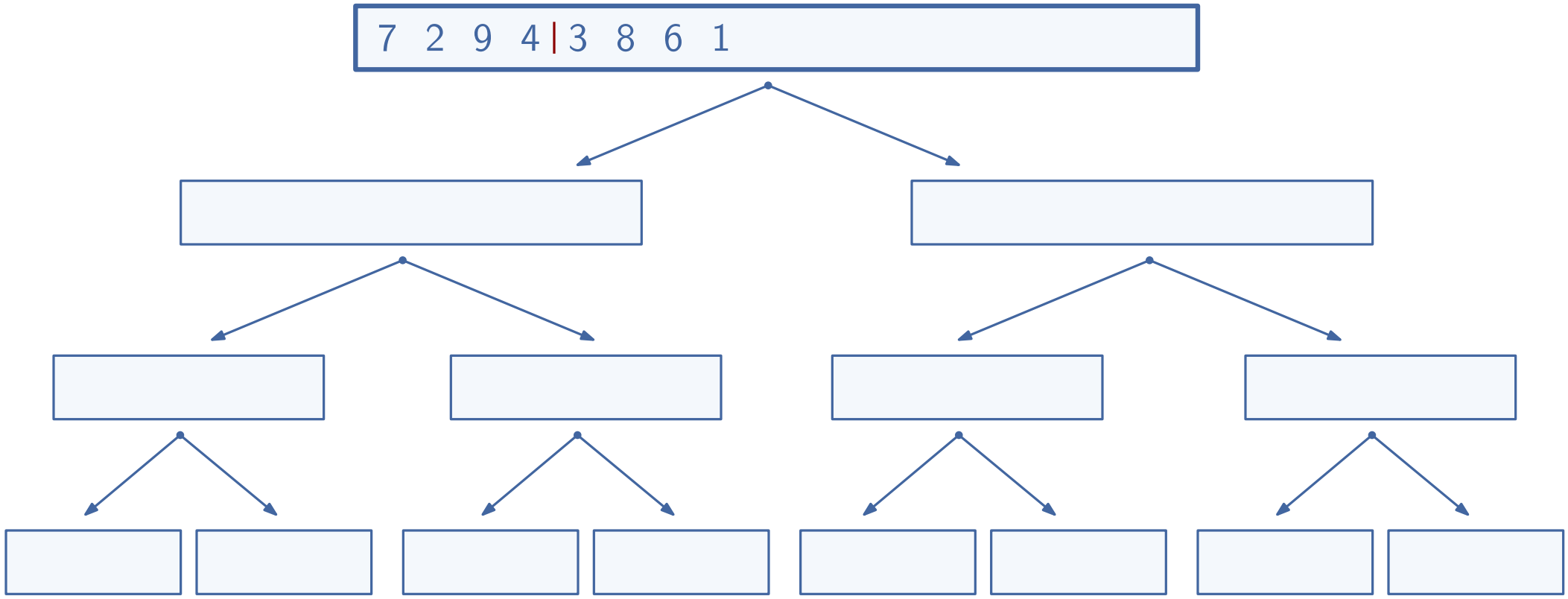
https://www.youtube.com/watch?v=JSceec-wEyw

Συγχώνευση

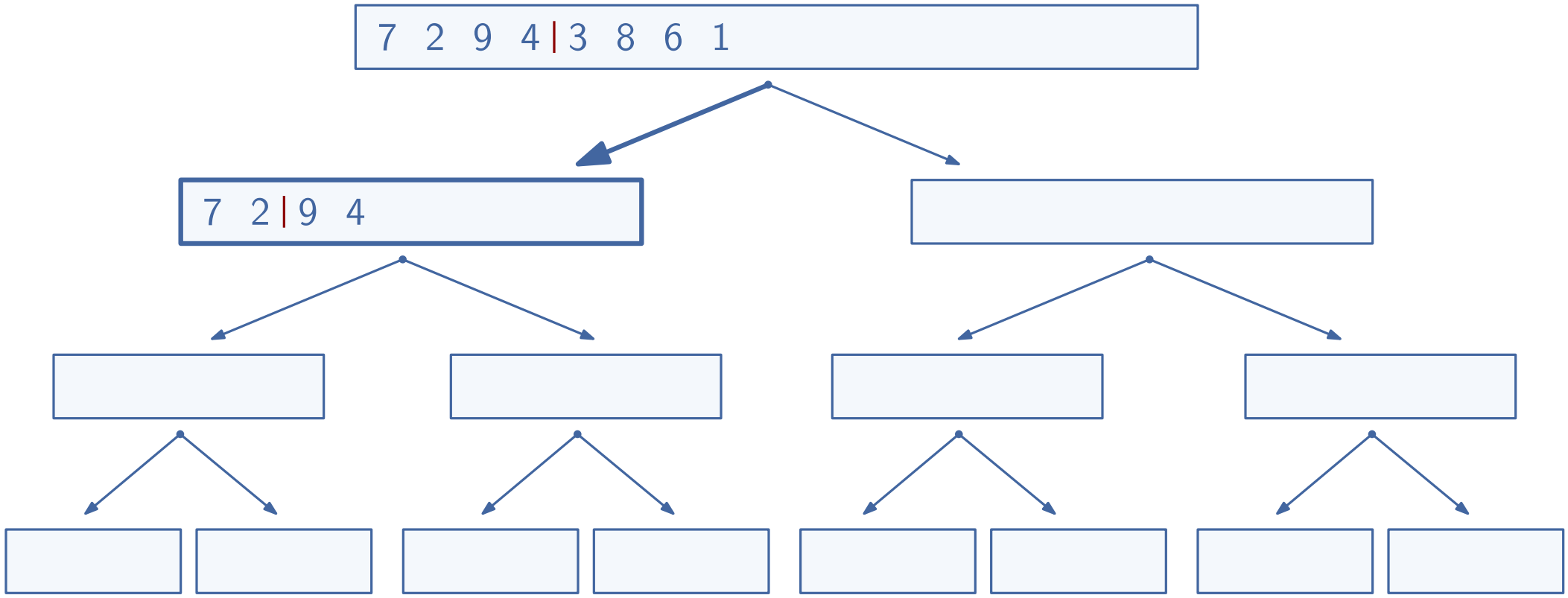
```
template <typename T>
void merge(T array[], int left, int middle, int right)
{
    // Create two temporary arrays
    int l = middle - left + 1, r = right - middle;
    T L[l], R[r];

    // Copy data to temporary arrays L and R
    for (int i = 0; i < l; i++) L[i] = array[left + i];
    for (int j = 0; j < r; j++) R[j] = array[middle + 1 + j];

    // Merge the L and R back into array[l...r]
    int i = 0, j = 0, k = left;
    while (i < l && j < r) {
        if (L[i] <= R[j]) { array[k] = L[i]; i++; k++;}
        else { array[k] = R[j]; j++; k++; }
    }

    // Copy the remaining elements of L and R, if any
    while (i < l) { array[k] = L[i]; i++; k++; }
    while (j < r) { array[k] = R[j]; j++; k++; }
}
```



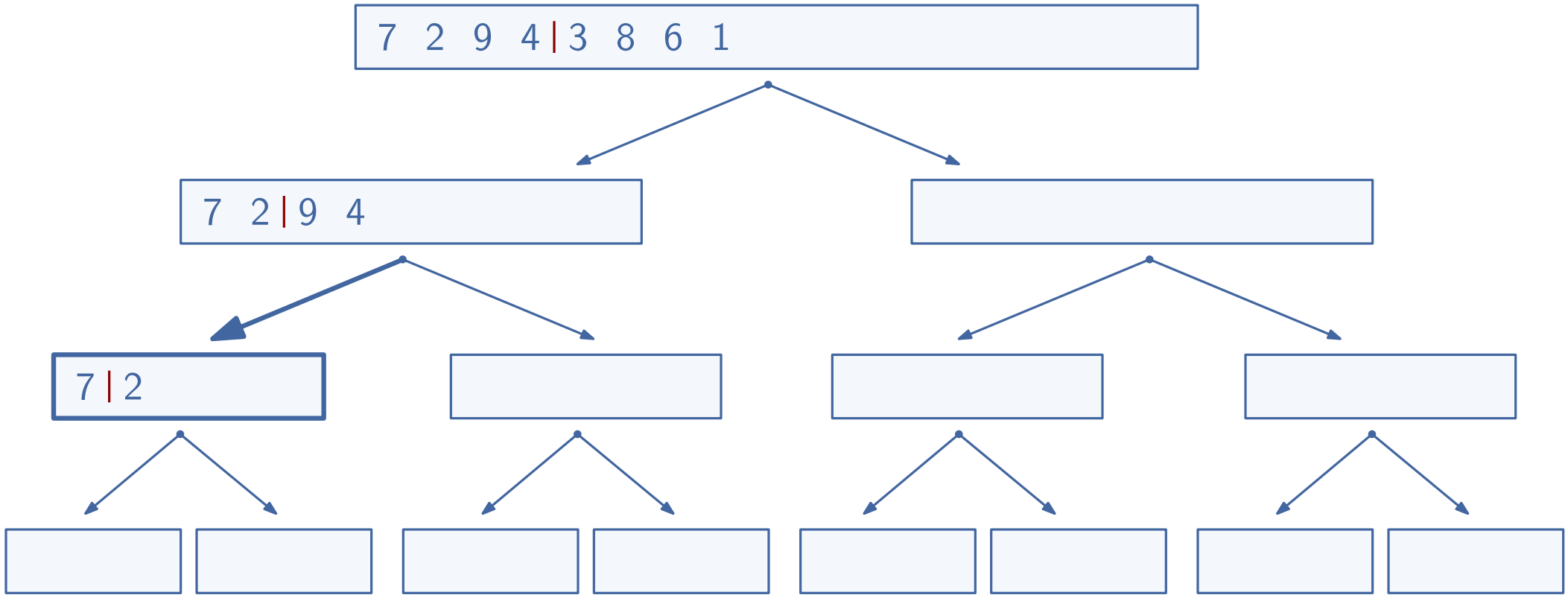
Παράδειγμα Εκτέλεσης



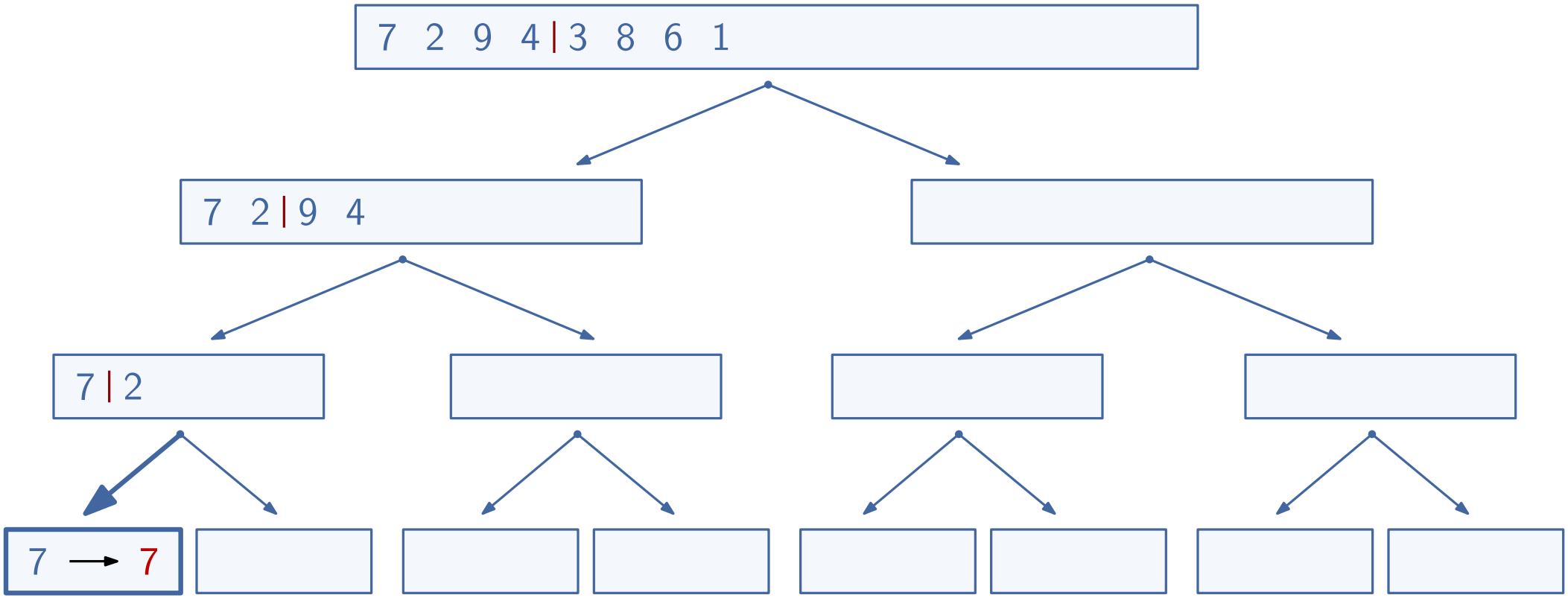
Παράδειγμα Εκτέλεσης



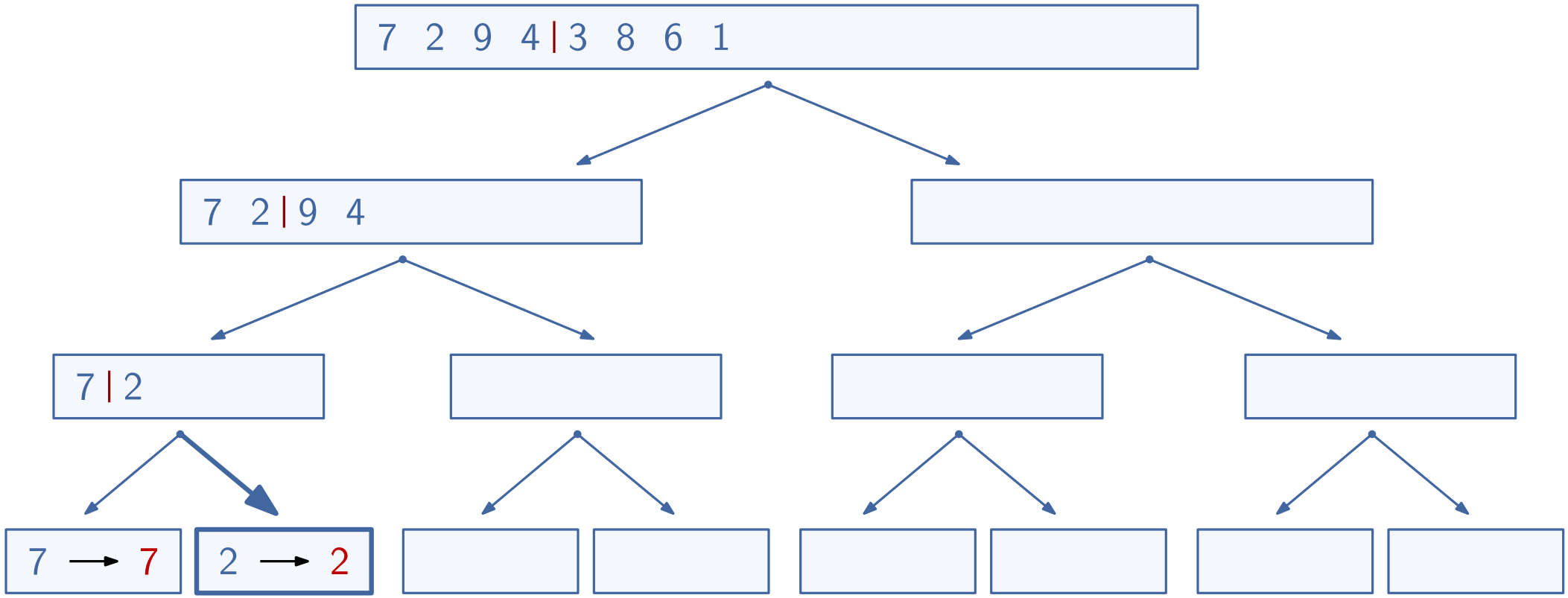
Παράδειγμα Εκτέλεσης



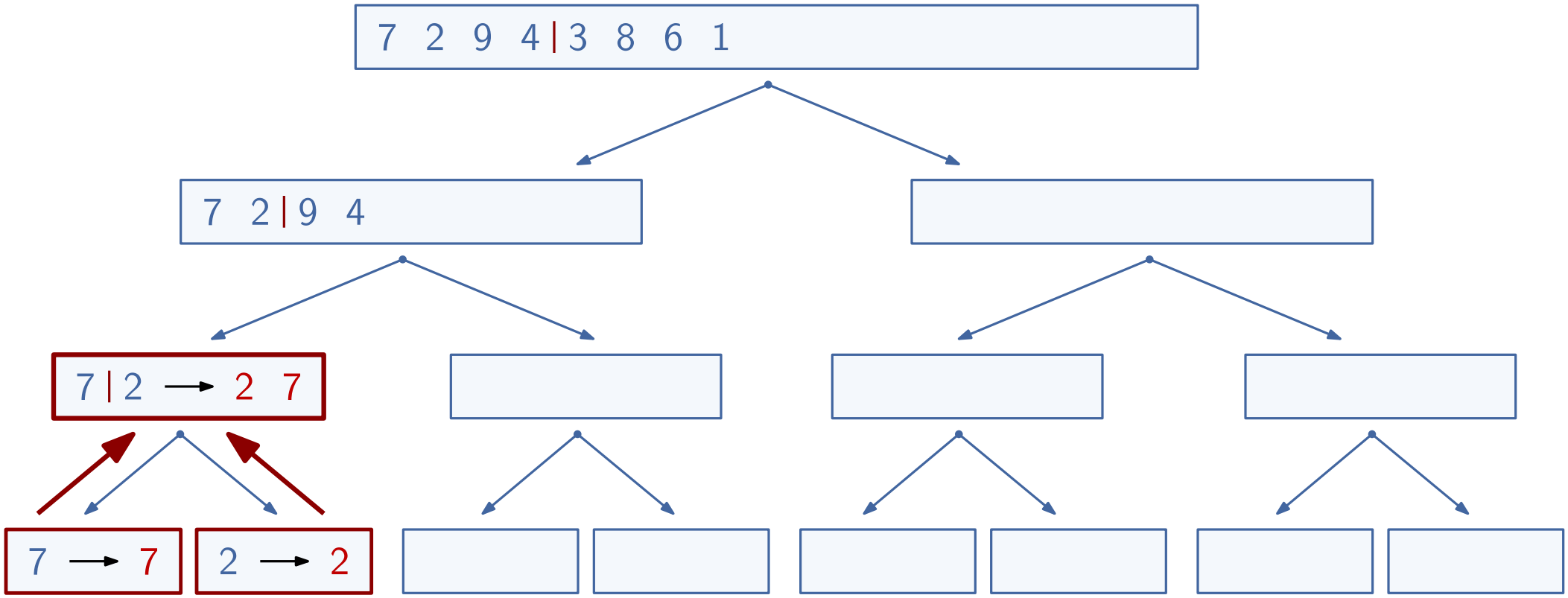
Παράδειγμα Εκτέλεσης



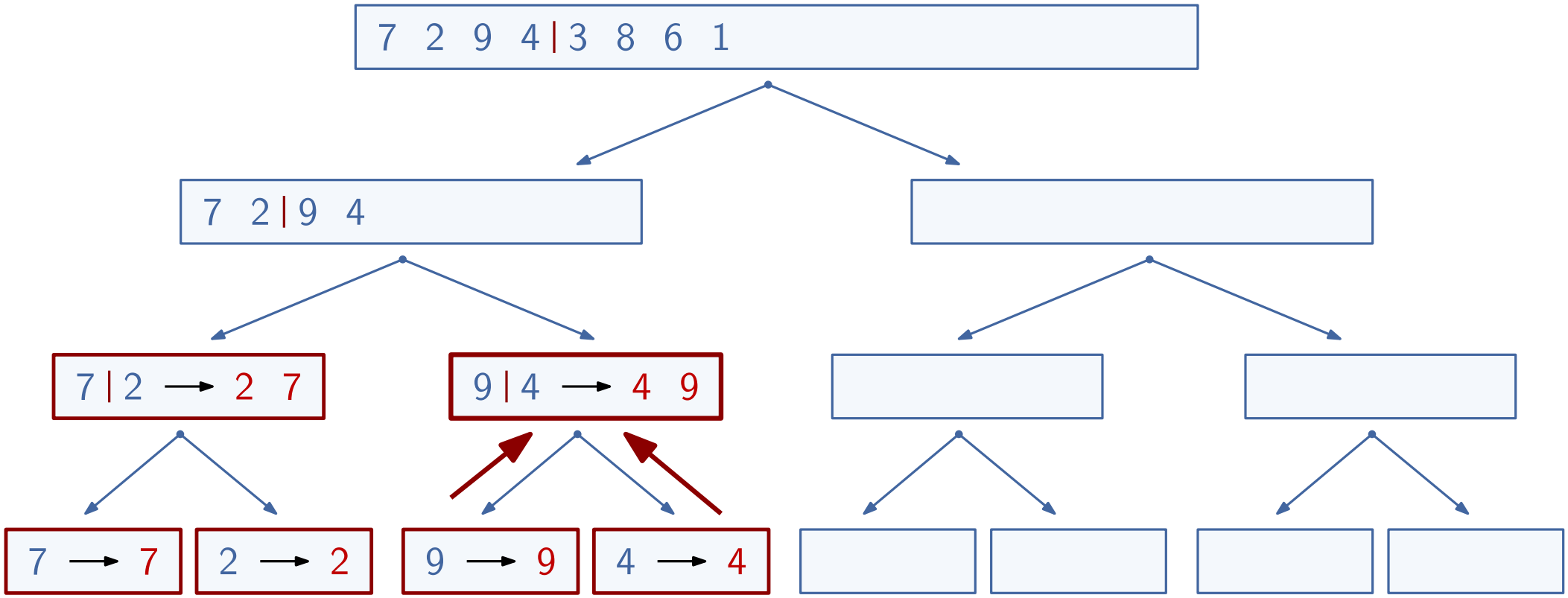
Παράδειγμα Εκτέλεσης



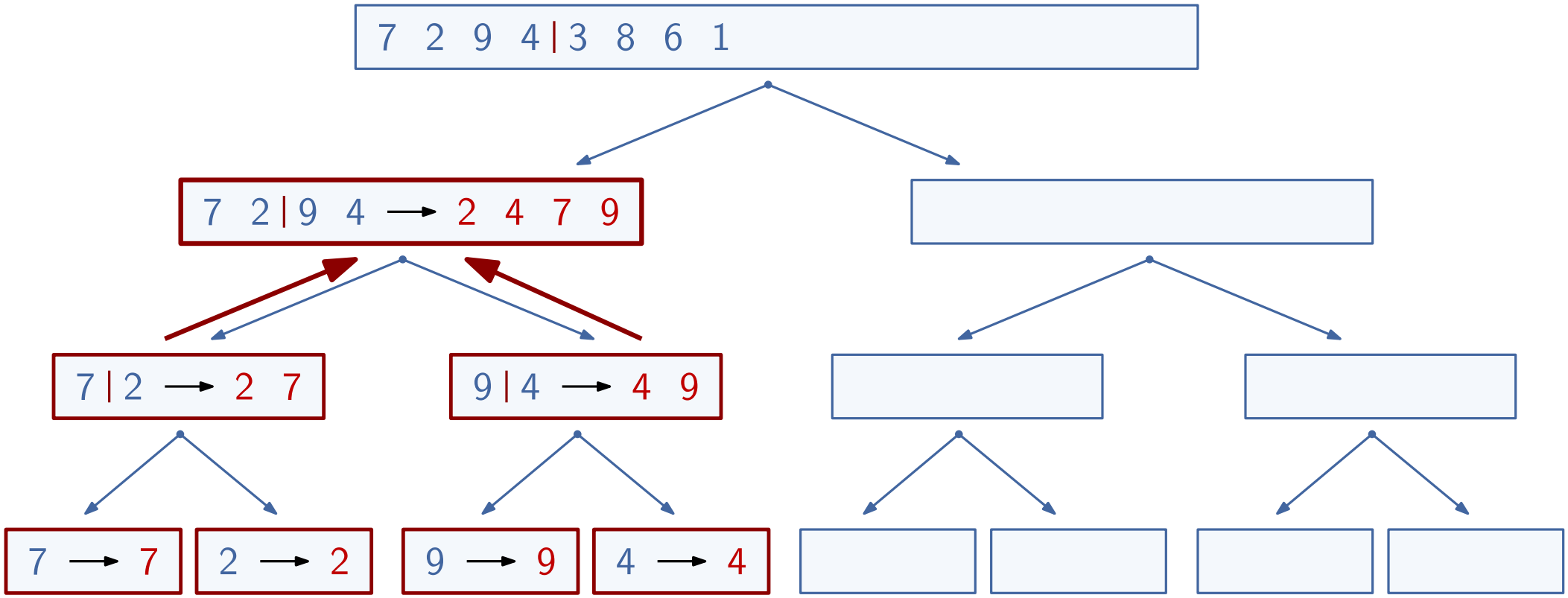
Παράδειγμα Εκτέλεσης



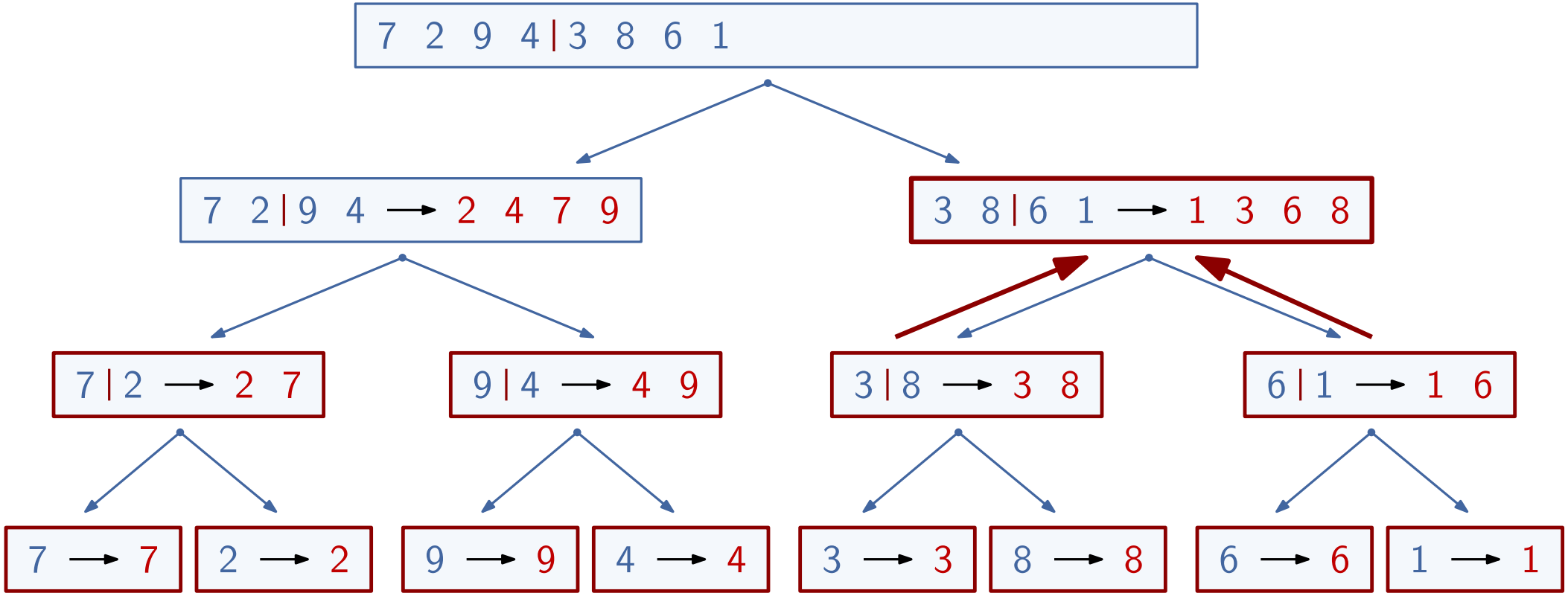
Παράδειγμα Εκτέλεσης



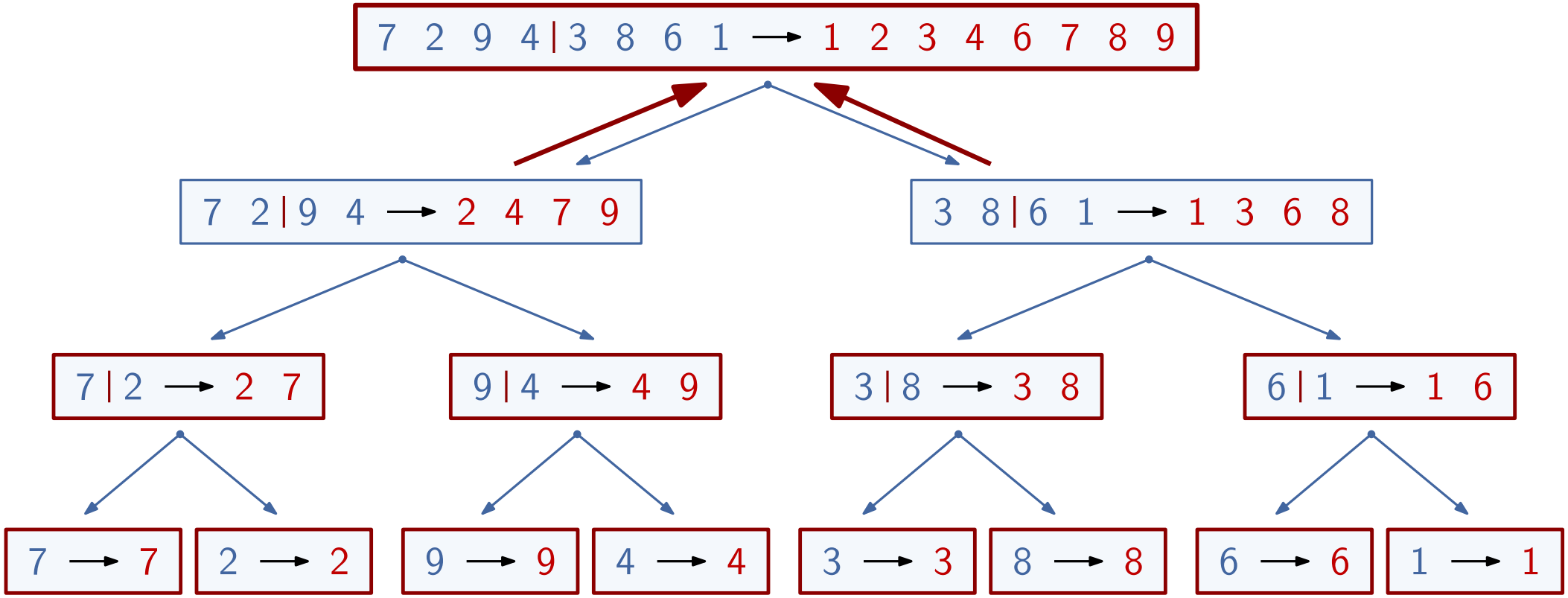
Παράδειγμα Εκτέλεσης



Παράδειγμα Εκτέλεσης



Παράδειγμα Εκτέλεσης



Ανάλυση Ταξινόμησης μέσω Συγχώνευσης

- Θεώρημα: Η χρονική πολυπλοκότητα, σε πλήθος συγκρίσεων, της ταξινόμησης μέσω συγχώνευσης δίνεται από την αναδρομική σχέση:

$$T[n] \leq T[\lfloor \tfrac{n}{2} \rfloor] + T[\lceil \tfrac{n}{2} \rceil] + n - 1 \qquad T[1] = 0$$

η οποία ως έχει λύση:

$$T[n] = n\lceil \log n \rceil - 2^{\lceil \log n \rceil} + 1 = \mathcal{O}(n \log n)$$

Απόδειξη:

Υποθέτουμε ότι το n είναι δύναμη του 2

$\Rightarrow n = 2^k \iff k = \log n$

$$\begin{array}{rcl}
T[n] & \leq & 2\ T[\frac{n}{2}] + \quad n - 1 \\
2T[\frac{n}{2}] & \leq & 2^2 T[\frac{n}{4}] + 2\ \frac{n}{2} - 2 \\
\cdots & & \\
2^{k-1} T[\frac{n}{2^{k-1}}] & \leq & 2^k T[\frac{n}{2^k}] + 2^{k-1} \frac{n}{2^{k-1}} - 2^{k-1} \\
\hline
T[n] & \leq & 2^k T[\frac{n}{2^k}] + nk - \sum_{j=0}^{k-1} 2^j \\
& = & n \log n - (2^k - 1) \\
& = & n \log n - n + 1 = \mathcal{O}(n \log n)
\end{array}$$

Σύνοψη

	worst case	average case	σημειώσεις
insertion sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	• αργές μέθοδοι • για μικρά σύνολα δεδομένων (<1K)
selection sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	
bubble sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	
merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	• γρήγορες μέθοδοι • για μεγάλα σύνολα δεδομένων (>1K)
heap sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	

- Ερώτημα: Μπορούμε να λύσουμε το πρόβλημα της ταξινόμησης σε χρόνο $o(n \log n)$?

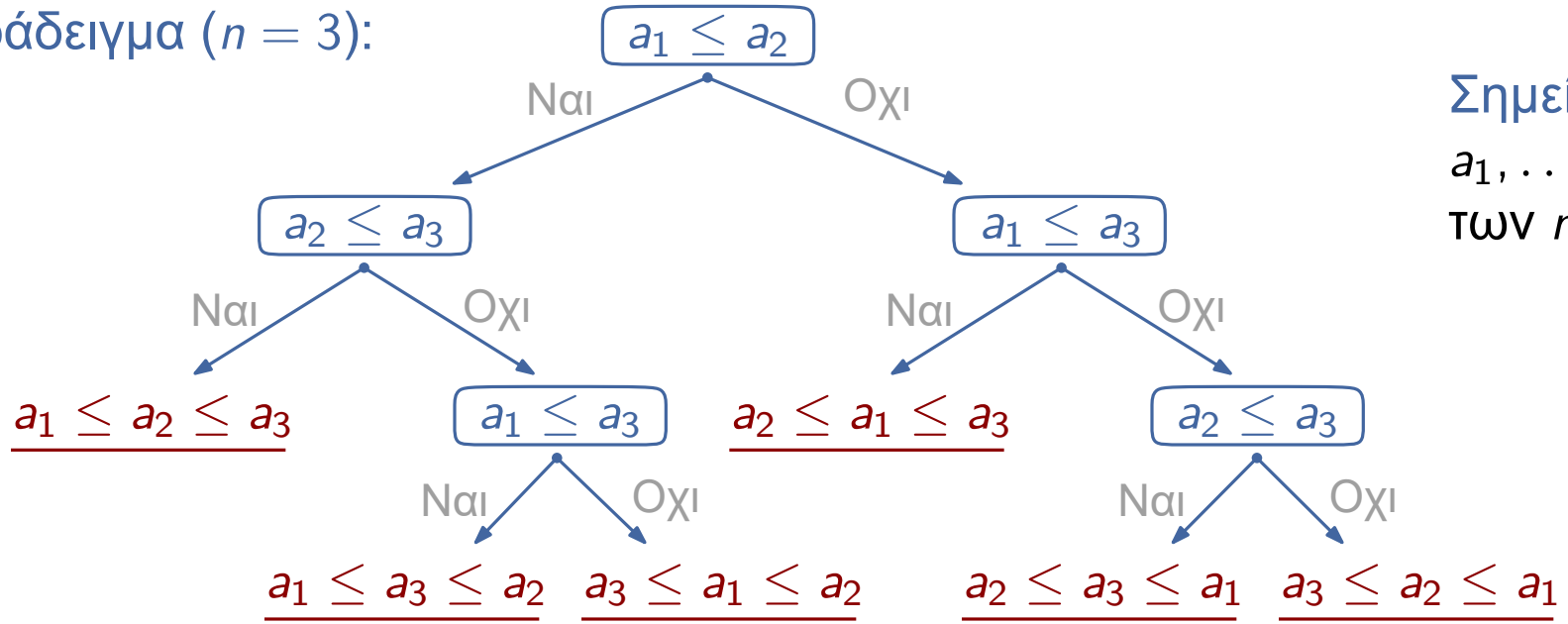
Κάτω Όριο για Συγκριτικούς Αλγορίθμους Ταξινόμησης

- Θεώρημα: Κάθε αλγόριθμος ταξινόμησης n στοιχείων, που βασίζεται σε συγκρίσης, απαιτεί $\Omega(n \log n)$ χρόνο

Απόδειξη:

Μπορούμε να αναπαραστήσουμε κάθε αλγόριθμο ταξινόμηση n στοιχείων ως δένδρο απόφασης.
Decision tree

Παράδειγμα ($n = 3$):



Σημείωση:

$a_1, \ldots, a_n$ αποτελεί μετάθεση των n στοιχείων
Permutation

Κάτω Όριο για Συγκριτικούς Αλγορίθμους Ταξινόμησης

- Θεώρημα: Κάθε αλγόριθμος ταξινόμησης n στοιχείων, που βασίζεται σε συγκρίσης, απαιτεί $\Omega(n \log n)$ χρόνο

  Απόδειξη:

  Μπορούμε να αναπαραστήσουμε κάθε αλγόριθμο ταξινόμηση n στοιχείων ως δένδρο απόφασης.
  Decision tree

  - Εκτέλεση του αλγορίθμου $\rightarrow$ μονοπάτι από τη ρίζα προς ένα φύλλο
  - Κάτω φράγμα για το πλήθος συγκρίσεων $\rightarrow$ μήκος κάθε τέτοιου μονοπατιού

    ύψος του δένδρου
  - Παρατήρηση: Το δένδρο έχει $n!$ φύλλα

    $\Rightarrow$ Το ύψος του δένδρου είναι $\geq \log n!$

    $\Rightarrow \log n!$ είναι ένα κάτω φράγμα
  - $\log n! \geq \log \left(\frac{n}{2}\right)^{\frac{n}{2}} = \frac{n}{2} \log \frac{n}{2} = \Theta(n \log n)$

- Ερώτημα: Υπάρχουν πιο αποδοτικοί αλγόριθμοι ταξινόμησης;

  - Ναι, αλλά απαιτούν επιπλέον υποθέσεις σχετικές με την είσοδο.
  - Δεν βασίζονται στο μοντέλο των συγκρίσεων

- Παραδείγματα: Bucket-sort

  Radix-sort

Ταξινόμηση Γραμμικού Χρόνου

Linear time sorting

- Είσοδος: Μια ακολουθία S αποτελούμενη από n στοιχεία, και ένας ακέραιος N, έτσι ώστε όλα τα στοιχεία της S να ανήκουν σε ένα σύνολο (σύμπαν) μεγέθους το πολύ N.

  Universe

- Εξοδος: Μια ταξινόμηση των στοιχείων της ακολουθίας S (από το μικρότερο προς το μεγαλύτερο)

- Παραδείγματα:
  - Ταξινόμηση n ακεραίων, καθένας από τους οποιούς ανήκει στο διάστημα $[0, N-1]$
  - Ταξινόμηση n ψηφίων, καθένα από τα οποία ανήκει στο αλφάβητο Σ = { 'Α', 'Β', . . ., 'Ω' }

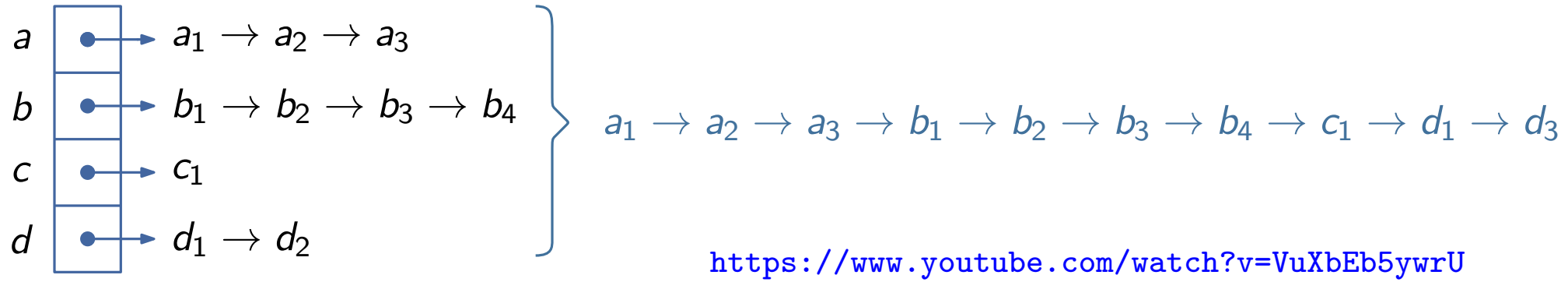
    ($N = 24$)

Ταξινόμηση Κάδου ή Δοχείου

Bucket or bin sort

```
1 B ← array of size N;
2 foreach entry e in S do
3 |   remove e from S and append it in the entry of B corresponding to it;
4 for i = 1 to N do
5 |   foreach entry e in B[i] do
6 |   |   remove e from B[i] and insert it at the end of S;
7 return S;
```

- Πολυπλοκότητα: $\mathcal{O}(n+N)$ (ευσταθής ταξινόμηση)
- Παράδειγμα: $S:\ b_1 \to d_1 \to a_1 \to b_2 \to b_3 \to c_1 \to a_2 \to b_4 \to d_2 \to a_3$

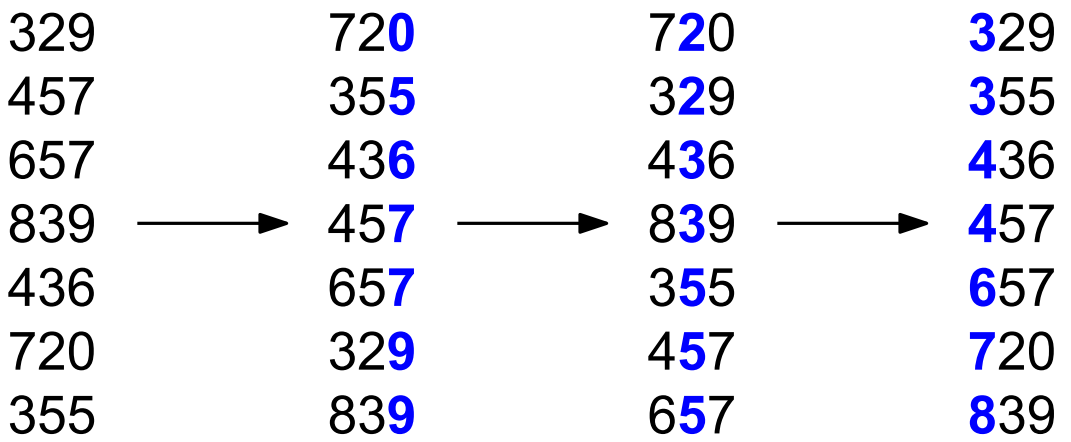


https://www.youtube.com/watch?v=VuXbEb5ywrU

Ταξινόμηση Γραμμικού Χρόνου

Linear time sorting

- Είσοδος: Μια ακολουθία S αποτελούμενη από n συμβολοσειρές d ψηφίων (ενός αλφαβήτου Σ μεγέθους N), έτσι ώστε το πρώτο (από τα αριστερά) ψηφίο καθεμίας συμβολοσειράς είναι το μικρότερης τάξης, ενώ το d-οστό είναι αυτό της μεγαλύτερης τάξης.
- Εξοδος: Μια ταξινόμηση των στοιχείων
- Ιδέα: Ταξινόμησε τα στοιχεία πρώτα με βάση το ελάχιστα σημαντικό στοιχείο.
- Παράδειγμα:



Ταξινόμηση Βάσης

Radix sort

```
1 for i = 1 to n do
2 |   use a stable sorting algorithm to sort S based on digit i
3 return S;
```

- Πολυπλοκότητα: $\mathcal{O}(d(n+N))$
- Απόδειξη ορθότητας (με επαγωγή στο πλήθος των θέσεων)
  - Υποθέτουμε ότι τα ψηφία χαμηλότερης τάξης είναι ταξινομημένα
  - Αποδεικνύουμε ότι η ταξινόμηση ως προς το επόμενο ψηφίο αφήνει την ακολουθία ορθά ταξινομημένη. Έστω δύο τυχόντα στοιχεία:
    - Εάν τα δύο ψηφία σε αυτή τη θέση είναι διαφορετικά, τότε ταξινομώντας τα σε αυτή τη θέση θα βρεθούν στη σωστή διάταξη.
    - Εάν τα δύο ψηφία σε αυτή τη θέση είναι ίδια, τότε τα στοιχεία αυτά βρίσκονται ήδη στη σωστή θέση, λόγω του ευσταθή αλγορίθμου.

https://www.youtube.com/watch?v=nu4gDuFabIM

Επιπλέον υλικό

- Ενότητες 8.1, 8.2, 9.1:
  Michael T. Goodrich, Roberto Tamassia, Αλγόριθμοι σχεδίαση και εφαρμογές
  ISBN: 9789605126971, εκδόσεις Γκιούρδα, 2016.