Σχεδίαση και Ανάλυση Αλγορίθμων

Ενότητα 5.0

Ταξινόμηση

Selection-Sort

Bubble-Sort και Insertion-Sort

Quick-Sort, Merge-Sort και Heap-Sort

Counting-Sort και Radix-Sort

Shell-Sort

Σταύρος Δ. Νικολόπουλος | 2016-17
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Ιωαννίνων
Webpage: www.cs.uoi.gr/~stavros

Αλγόριθμοι Ταξινόμησης

Selection-Sort

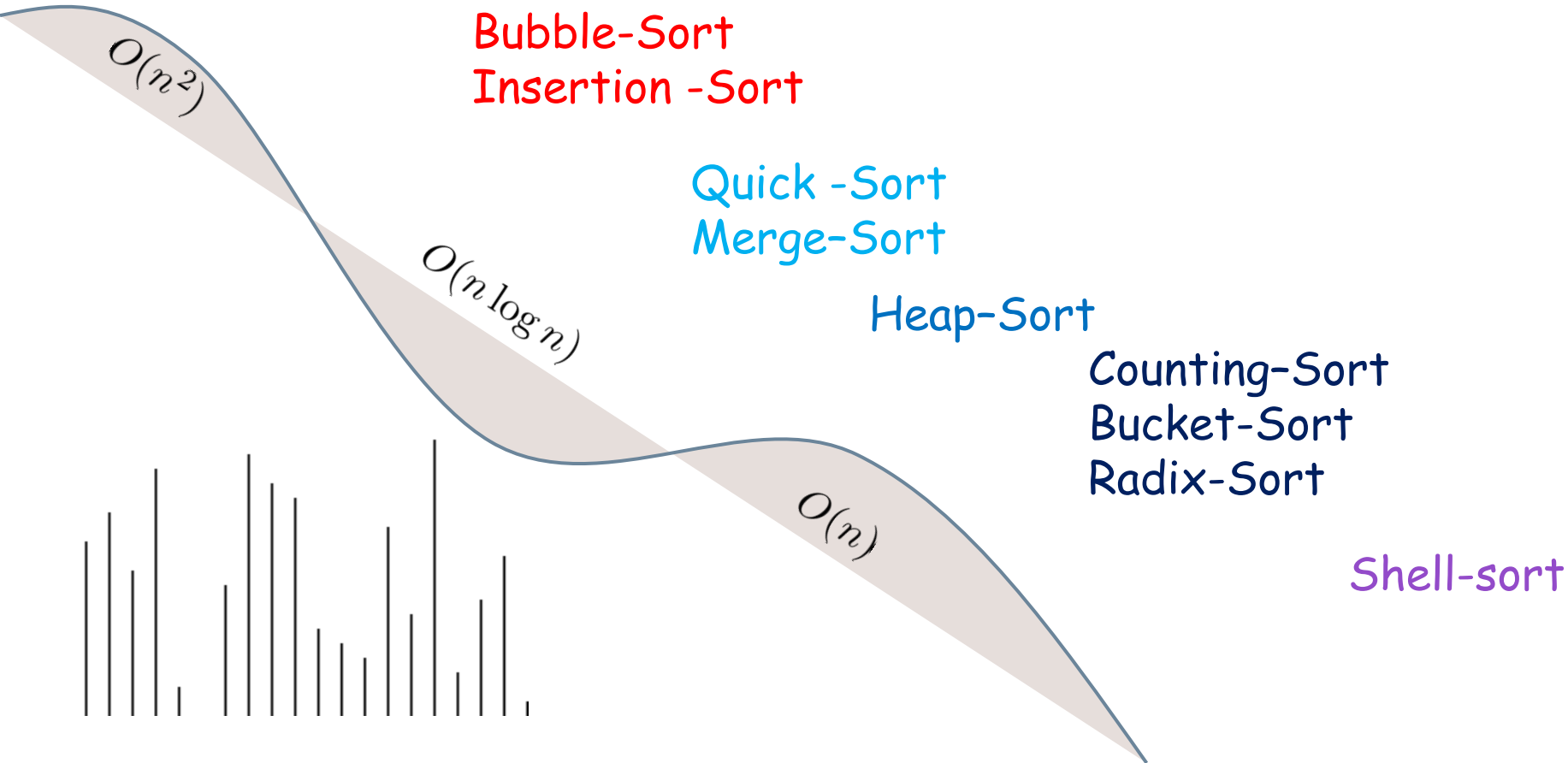
Bubble-Sort
Insertion -Sort

Quick -Sort
Merge-Sort

Heap-Sort

Counting-Sort
Bucket-Sort
Radix-Sort

Shell-sort



Αλγόριθμοι Ταξινόμησης

Selection-Sort

$O(n^2)$

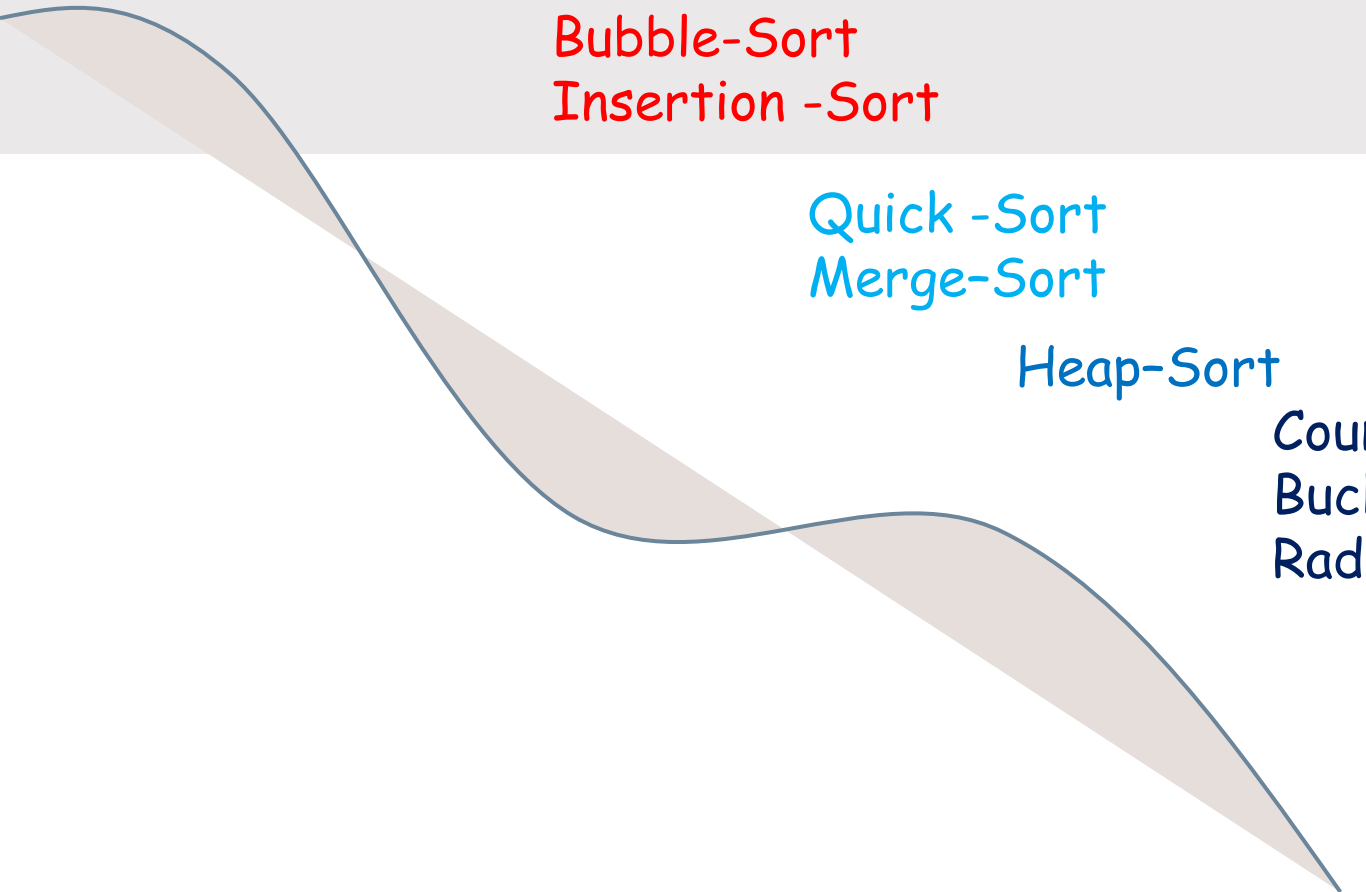
Bubble-Sort
Insertion -Sort

Quick -Sort
Merge-Sort

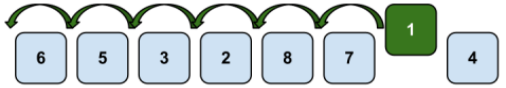
Heap-Sort

Counting-Sort
Bucket-Sort
Radix-Sort

Shell-sort



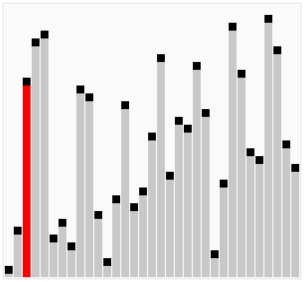
Το Πρόβλημα της Ταξινόμησης



Ορισμός Προβλήματος

Είσοδος : ακολουθία n στοιχείων $S = (\alpha_1, \alpha_2, \ldots, \alpha_n)$.

Έξοδος : αναδιάταξη $(\alpha'_1, \alpha'_2, \ldots, \alpha'_n)$ των στοιχείων της S σε αύξουσα τάξη ($\forall i, 1 \leq i \leq n-1, \alpha'_i \leq \alpha'_{i+1}$).



Θεμελιώδες αλγοριθμικό πρόβλημα

Πολλές εφαρμογές (~20-25% του υπολογιστικού χρόνου).

Ταχύτατη αναζήτηση σε ταξινομημένα δεδομένα.

Σημαντικές αλγοριθμικές ιδέες και τεχνικές !!!

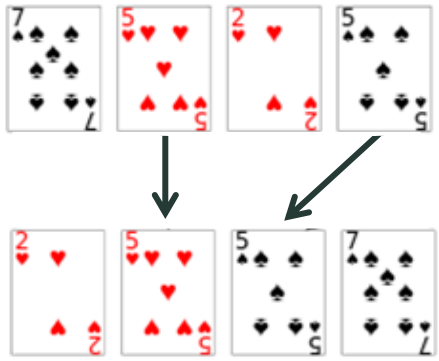
Το Πρόβλημα της Ταξινόμησης

Ευσταθής Αλγόριθμος Ταξινόμησης

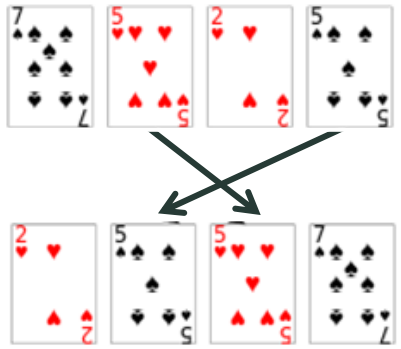
Ο αλγόριθμος ταξινόμησης που *διατηρεί η σχετική σειρά των στοιχείων με ίσες τιμές*.

Δηλαδή, ένας αλγόριθμος ταξινόμησης είναι ευσταθής εάν για οποιαδήποτε δύο στοιχεία x και y με την *ίδια τιμή* ισχύει ότι το x *βρίσκεται πριν το* y στην αρχική ακολουθία S, να ισχύει ότι το x *βρίσκεται πριν το* y στην ταξινομημένη ακολουθία S' !!!

Ευσταθής Ταξινόμηση

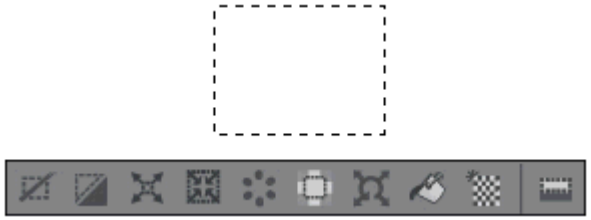


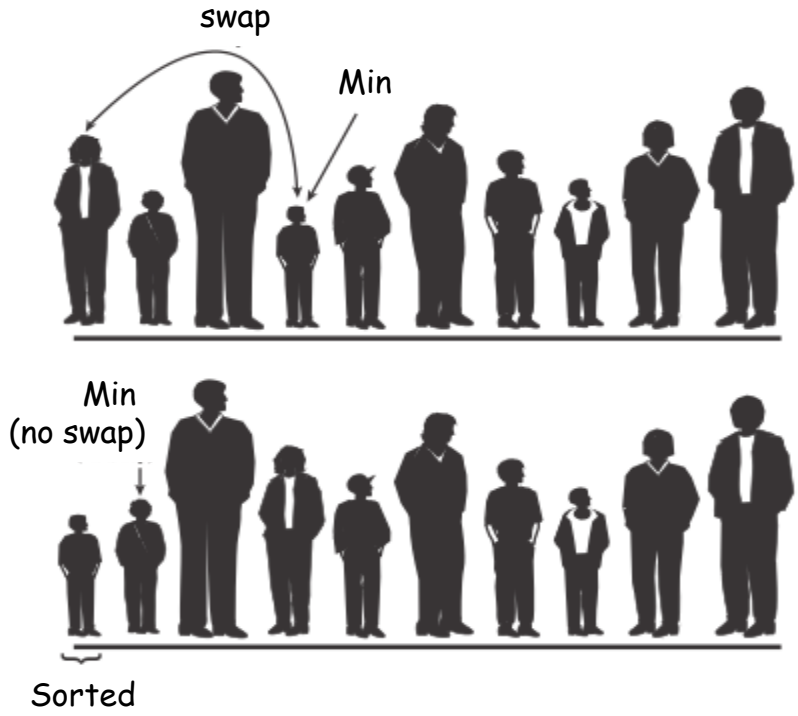
Μη-ευσταθής Ταξινόμηση



Αλγόριθμος Selection-sort

Επιλογής





Αλγόριθμος Selection-sort (Επιλογής)

Ιδέα Αλγόριθμου !!!

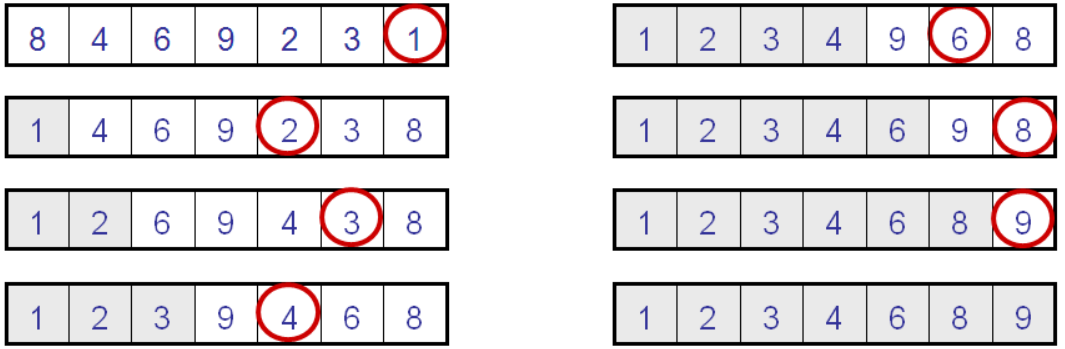
Quick-Select(S[1,n], i)

Το **i-th** τάξης στοιχείο της ακολουθίας **S[1,n]** στη θέση **i, 1 ≤ i ≤ n.**

or

Quick-Select(S[i,n], 1)

Το **min** της ακολουθίας **S[i,n]** στη θέση **i** (και το **S[i]** στη θέση του **min**).



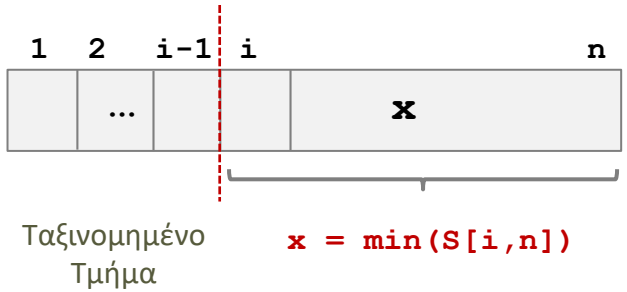
Ονομάζεται και Min-Sort (ή Max-Sort)

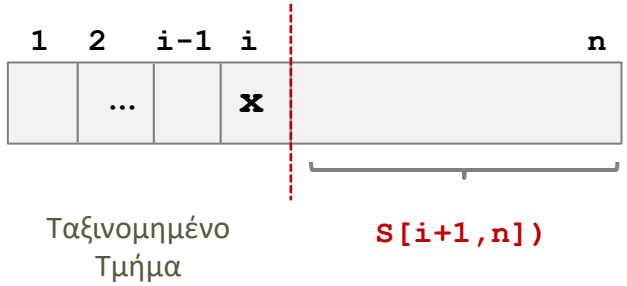
Αλγόριθμος Selection-Sort

Αλγόριθμος Selection-sort (Επιλογής)

```
Algorithm Selection-sort
begin
        Quick-Select(S[1,n], i)
                or
        x = Min(S[i,n])
        και αντάλλαξέ το με το στοιχείο
        S[i] ⟷ x
    end
    return S
end
```

Ονομάζεται και Min-Sort (ή Max-Sort)





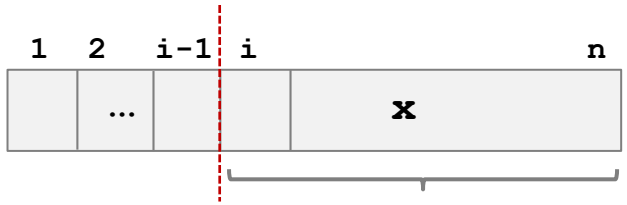
Το **min** της ακολουθίας **S[i,n]** στη θέση **i**

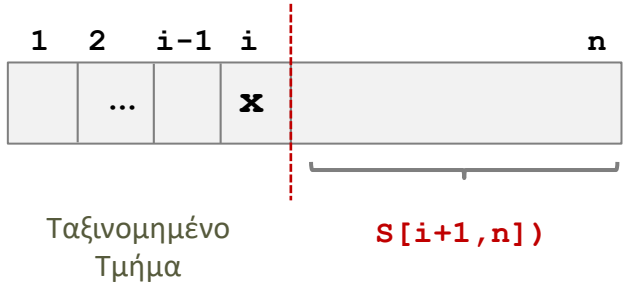
Αλγόριθμος Selection-Sort

Αλγόριθμος Selection-sort (Επιλογής)

```
Algorithm Selection-sort
begin
        Βρες το x = min στοιχείο στην
              υποακολουθία
                  S[i,n]
        και αντάλλαξέ το με το στοιχείο
                  S[i] ⟷ x
    end
    return S
end
```

Ονομάζεται και Min-Sort (ή Max-Sort)





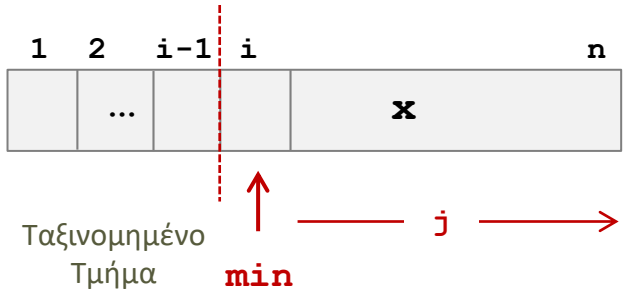
Το **min** της ακολουθίας **S[i,n]** στη θέση **i**

Αλγόριθμος Selection-Sort

Αλγόριθμος Selection-sort (Επιλογής)

```
Algorithm Selection-sort
begin
    for i ← 1 to n-1 do
        min ← S[i]; pos ← i
        for j ← i+1 to n do
            if S[j] < min then
                min ← S[j]; pos ← j
            end
        end
        Swap(S[i], S[pos])
    end
    return S
end
```

Ονομάζεται και Min-Sort (ή Max-Sort)



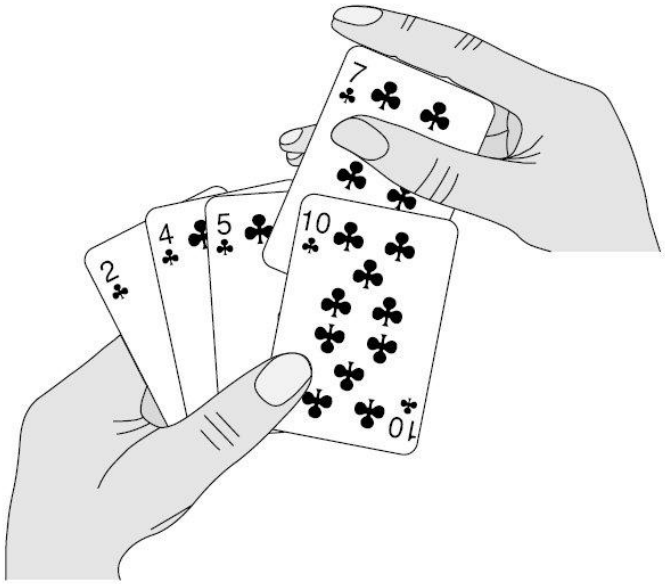
Ορθότητα ✓

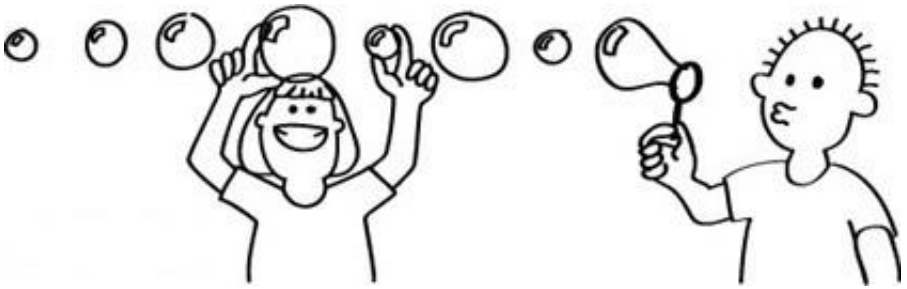
Πολυπλοκότητα $O(n^2)$

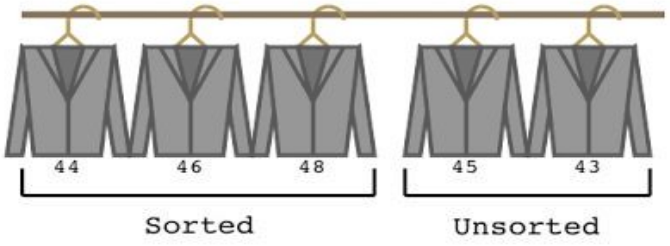
Το **min** της ακολουθίας **S[i,n]** στη θέση **i**

Αλγόριθμοι Bubble-Sort & Insertion-Sort

Φυσαλίδας και Παρεμβολής





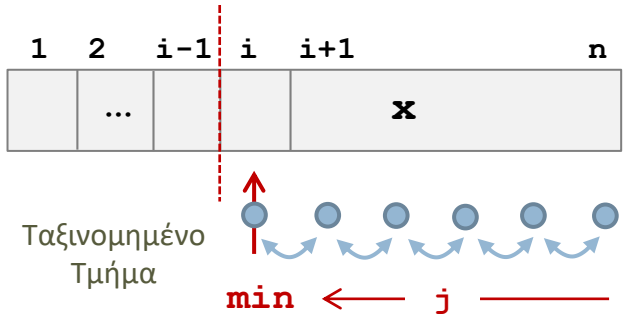


Αλγόριθμος Bubble-Sort (Φυσαλίδας)

```
Algorithm Bubble-sort
begin
      for i ← 1 to n-1 do
         for j ← n downto i+1 do
```

> Σύγκριση `S[j]` και `S[j-1]`
> και ανταλλαγή εάν χρειάζεται

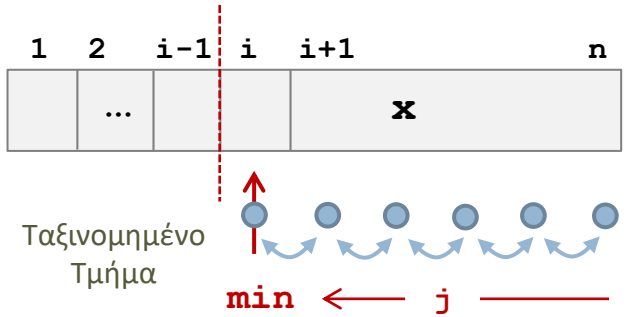
```
      end
      return S
end
```



Το **`min`** της ακολουθίας **`S[i,n]`**
στη θέση **`i`**

Αλγόριθμος Bubble-Sort (Φυσαλίδας)

```
Algorithm Bubble-sort
begin
      for i ← 1 to n-1 do
         for j ← n downto i+1 do
            if S[j] < S[j-1] then
               Swap(S[j], S[j-1])
             end
      end
      return S
end
```



Ορθότητα ✓

Πολυπλοκότητα $O(n^2)$

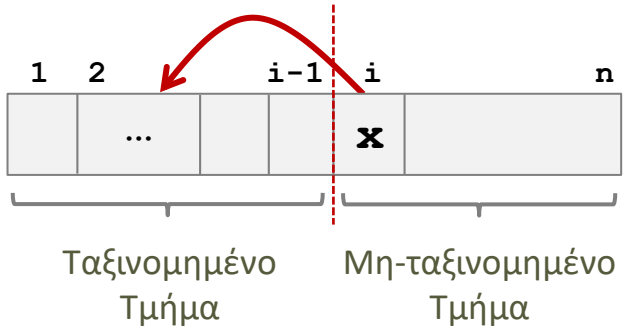
Το **min** της ακολουθίας **S[i,n]** στη θέση **i**

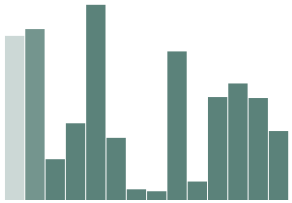
Αλγόριθμος Insertion-Sort (Παρεμβολής)

```
Algorithm Insertion-sort
begin
      for i ← 2 to n do
            Εισαγωγή του στοιχείου x της
            υποακολουθίας S[i+1,n]

            στην "σωστή" του θέση

            στην υποακολουθία S[1,i]
      end
      return S
end
```

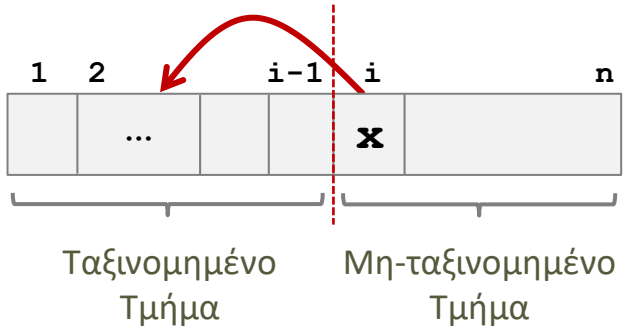




Ονομάζεται και Αλγόριθμος Εισαγωγής

Αλγόριθμος Insertion-Sort (Παρεμβολής)

```
Algorithm Insertion-sort
begin
      for i ← 2 to n do
            j ← i-1
            while j ≥ 1 and x < S[j] do
                  Swap(x, S[j])
                  j ← j-1
            end
      end
      return S
end
```



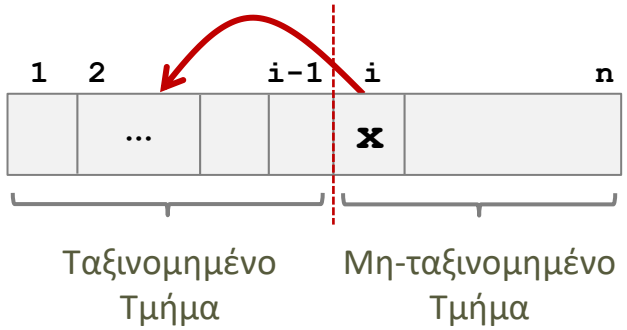
Ορθότητα

Πολυπλοκότητα $O(n^2)$

Μια πρώτη υλοποίηση του Insertion-Sort !

Αλγόριθμος Insertion-Sort (Παρεμβολής)

```
Algorithm Insertion-sort
begin
      for i ← 2 to n do
          x ← S[i];
          j ← i-1
          while j ≥ 1 and x < S[j] do
                S[j+1] ← S[j]
                j ← j-1
          end
          S[j+1] ← x
      end
      return S
end
```



Ορθότητα

Πολυπλοκότητα $O(n^2)$

Μια καλύτερη υλοποίηση του Insertion-Sort !!!

Αλγόριθμος Insertion-Sort (Παρεμβολής)

```
Algorithm Insertion-sort
begin
      for i ← 2 to n do
            εντολές
      end
      return S
end
```

Ορθότητα Insertion-sort

- Για τον αλγόριθμο **Insertion-sort** θεωρούμε την παρακάτω *αναλλοίωτη συνθήκη επανάληψης*:

*Κατά την έναρξη κάθε επανάληψη του εξωτερικού βρόχου (**for**), η υποακολουθία με στοιχεία από την θέση **1** έως την **i**, δηλ. **S[1,i]**, αποτελείται από τα ίδια στοιχεία που υπήρχαν αρχικά στην **S** στις θέσεις **1** έως **i**, αλλά ταξινομημένα (κάθε στοιχείο είναι μεγαλύτερο από όλα τα αριστερά του).*

Θα αποδείξουμε ότι η παραπάνω πρόταση αποτελεί αναλλοίωτη συνθήκη επανάληψης :

- ✓ **Αρχικοποίηση** - Η υποακολουθία **S[1,1]** που αποτελείται από το πρώτο στοιχείο είναι ταξινομημένη.
- ✓ **Διατήρηση** - Κάθε επανάληψη του βρόχου επεκτείνει την υποακολουθία **S[1,i]** διατηρώντας αναλλοίωτη την συνθήκη επανάληψης. Πράγματι, όταν το στοιχείο **x** εισάγεται στην **S[1,i]** είναι μεγαλύτερο από το στοιχείο **y** στα αριστερά του. Δεδομένου ότι τα στοιχεία στα αριστερά του **y** έχουν ήδη ταξινομηθεί, συνεπάγεται ότι το **x** είναι μεγαλύτερο από ***όλα*** τα στοιχεία αριστερά του. Άρα, η υποακολουθία **S[1,i]** παραμένει ταξινομημένη.
- ✓ **Τερματισμός** - Ο αλγόριθμος τερματίζει όταν το **i = n**, που σημαίνει ότι η ταξινομημένη υποακολουθία **S[1,i]** έχει επεκταθεί και τώρα είναι η **S[1,n]**. Άρα, η ακολουθία S είναι τώρα πλήρως ταξινομημένη.

Αλγόριθμος Insertion-Sort (Παρεμβολής)

Χαρακτηριστικά Αλγόριθμου

- Απλός (simple): Είναι απλός στην εφαρμογή του.
- Αποτελεσματικός (efficient): Για μικρά σύνολα δεδομένων.
- Αποδοτικός (efficient in practice): Πιο αποτελεσματικός στην πράξη από άλλους απλούς αλγορίθμους $O(n^2)$ (selection-Sort ή bubble-Sort), όπου στην «καλή» περίπτωση (σχεδόν ταξινομημένα τα στοιχεία κατά την είσοδο) ο χρόνος εκτέλεσης είναι $O(n)$.
- Σταθερός (stable): Δεν αλλάζει τη σχετική σειρά των στοιχείων που έχουν ίσα κλειδιά.
- Σε ίδιο χώρο (in-place): Απαιτεί μόνο μια σταθερή ποσότητα $O(1)$ επιπλέον χώρο μνήμης.
- Άμεσος (on-line): Μπορεί να ταξινομήσει μια ακολουθία καθώς τη λαμβάνει !!!

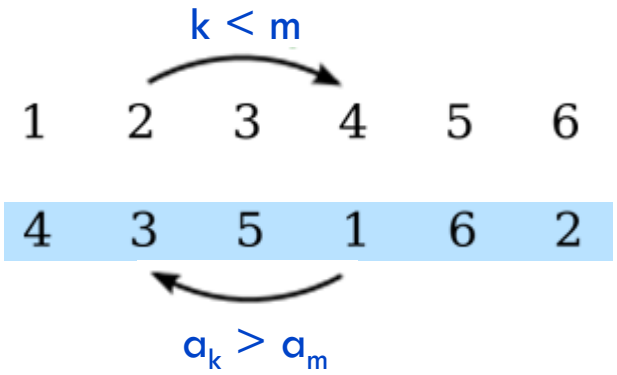
Συμπεριφορά Bubble-Sort & Insertion-Sort

- Κλάση Αλγορίθμων: Μετά από κάθε σύγκριση δύο στοιχείων:
  - Κανένα στοιχείο δεν μετακινείται
  - Ανταλλάσουν θέσεις

Όλοι οι αλγόριθμοι ταξινόμησης που εκτελούν τέτοιες «περιορισμένες» και «τοπικές» ανταλλαγές στοιχείων συμπεριφέρονται το ίδιο με τους αλγορίθμους Bubble-Sort και Insertion-Sort

Συμπεριφορά Bubble-Sort & Insertion-Sort

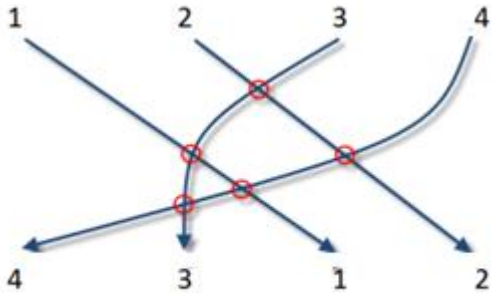
- Έστω μια μετάθεση $A = (a_1, a_2, \ldots, a_n)$ του συνόλου $N_n = \{1, 2, \ldots, n\}$.
- Ένα ζεύγος στοιχείων (a_k, a_m) της μετάθεσης A ονομάζεται *αντιστροφή* (inversion) εάν για τις θέσεις τους ισχύει $k < m$ και για τις τιμές του $a_k > a_m$.



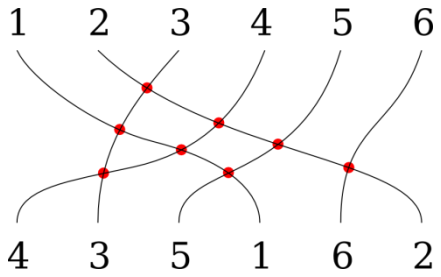
Το ζεύγος (3, 1) είναι μια αντιστροφή

Συμπεριφορά Bubble-Sort & Insertion-Sort

- Έστω μια μετάθεση $A = (a_1, a_2, \ldots, a_n)$ του συνόλου $N_n = \{1, 2, \ldots, n\}$.
- Ένα ζεύγος στοιχείων (a_k, a_m) της μετάθεσης A ονομάζεται *αντιστροφή* (inversion) εάν για τις θέσεις τους ισχύει $k < m$ και για τις τιμές του $a_k > a_m$.



5 αντιστροφές



8 αντιστροφές

Συμπεριφορά Bubble-Sort & Insertion-Sort

- Οποιοσδήποτε αλγόριθμος ταξινόμησης (όπως οι Bubble-Sort και Insertion-Sort) που εκτελεί «περιορισμένες» και «τοπικές» ανταλλαγές στοιχείων

Με κάθε σύγκριση «διορθώνει» το πολύ μία αντιστροφή !!!

Επομένως, το πλήθος των συγκρίσεων σε μια είσοδο $A = (a_1, a_2, \ldots, a_n)$ είναι τουλάχιστον ίση με το πλήθος των αντιστροφών στη μετάθεση A !!!

Πρόταση: Μια μετάθεση $A = (a_1, a_2, \ldots, a_n)$ μήκους n έχει το πολύ

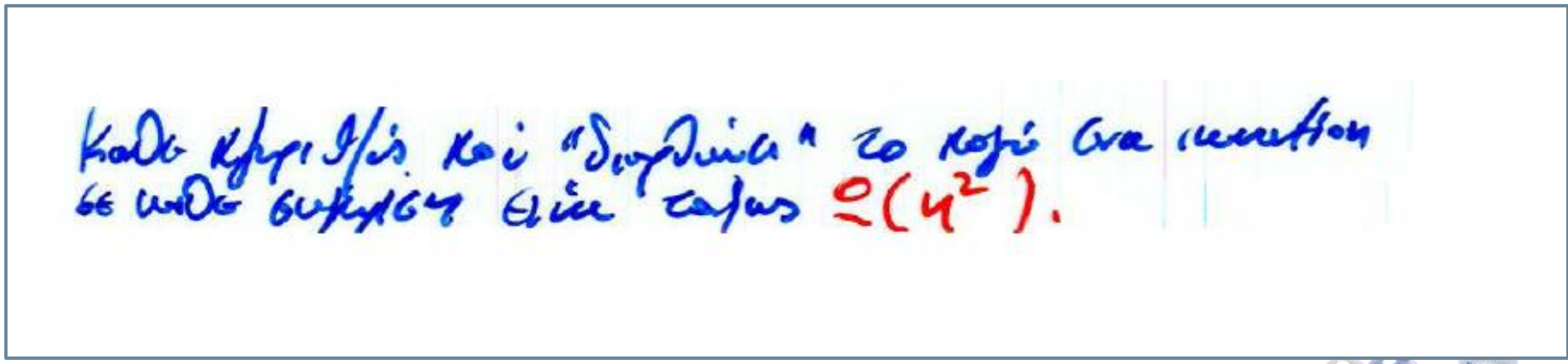
$$\frac{n(n-1)}{2}$$

πλήθος αντιστροφών.

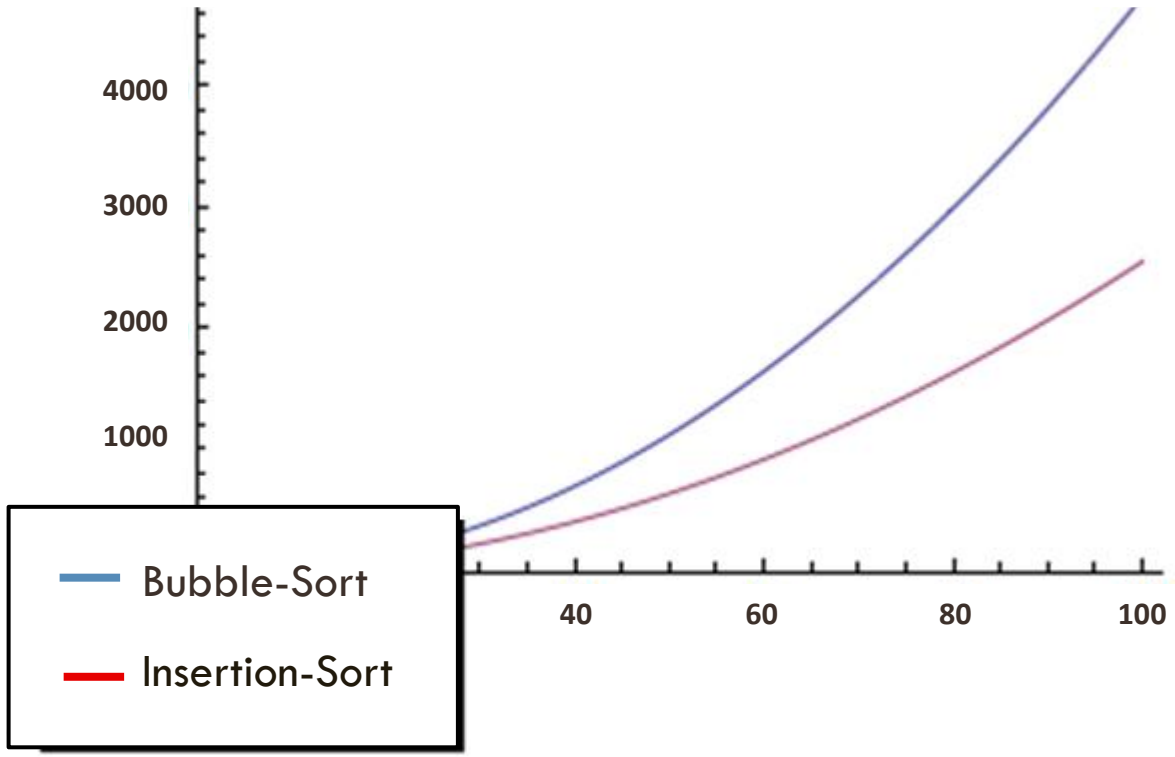
Συμπεριφορά Bubble-Sort & Insertion-Sort

- Οποιοσδήποτε αλγόριθμος ταξινόμησης (όπως οι Bubble-Sort και Insertion-Sort) που εκτελεί «περιορισμένες» και «τοπικές» ανταλλαγές στοιχείων

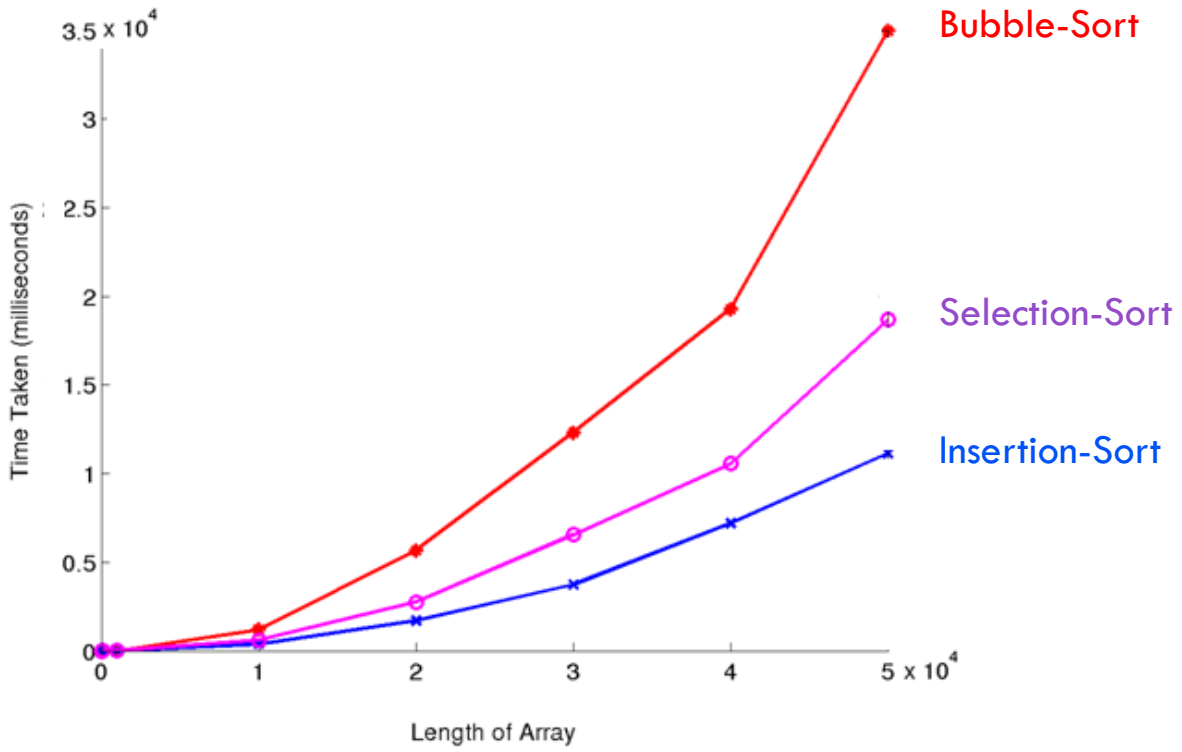
Με κάθε σύγκριση «διορθώνει» το πολύ μία αντιστροφή !!!



Σύγκριση Αλγορίθμων Ταξινόμησης



Σύγκριση Αλγορίθμων Ταξινόμησης



Αλγόριθμοι Ταξινόμησης

Selection-Sort

Bubble-Sort
Insertion -Sort

Quick -Sort
Merge-Sort
Heap-Sort

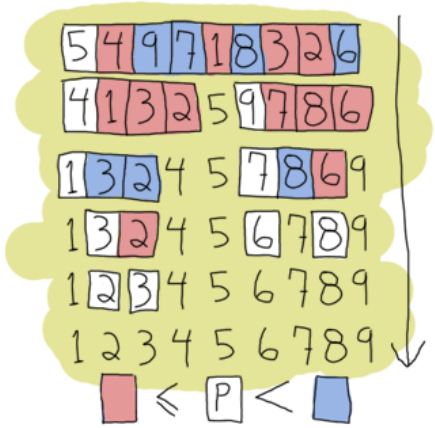
$O(n \log n)$

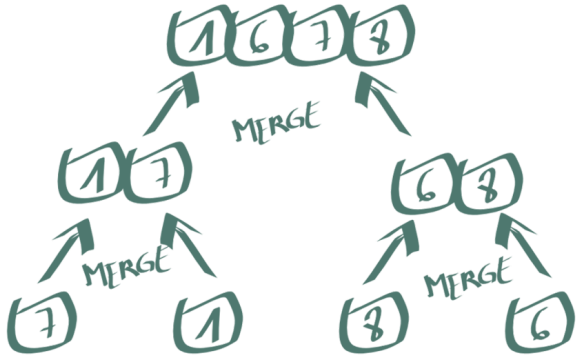
Counting-Sort
Bucket-Sort
Radix-Sort

Shell-sort

- **Αλγόριθμοι Quick-sort και Merge-sort**

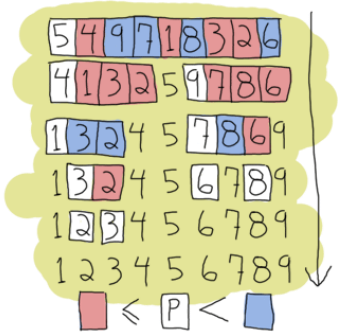
Divide & Conquer

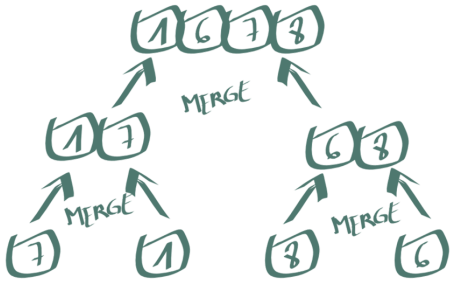




Αλγόριθμοι Quick-sort και Merge-sort

Divide & Conquer

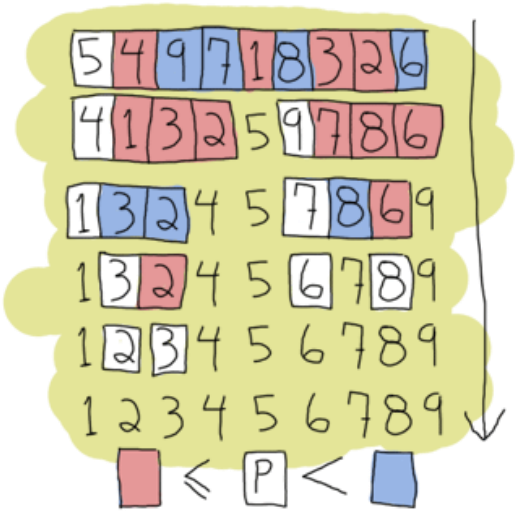




- **Quick-sort** *Διαμέριση σε "μικρότερα – pivot – μεγαλύτερα"*
- **Merge-sort** *Συγχώνευση ταξινομημένων ακολουθιών*

Αλγόριθμος Ταξινόμησης Quick-Sort

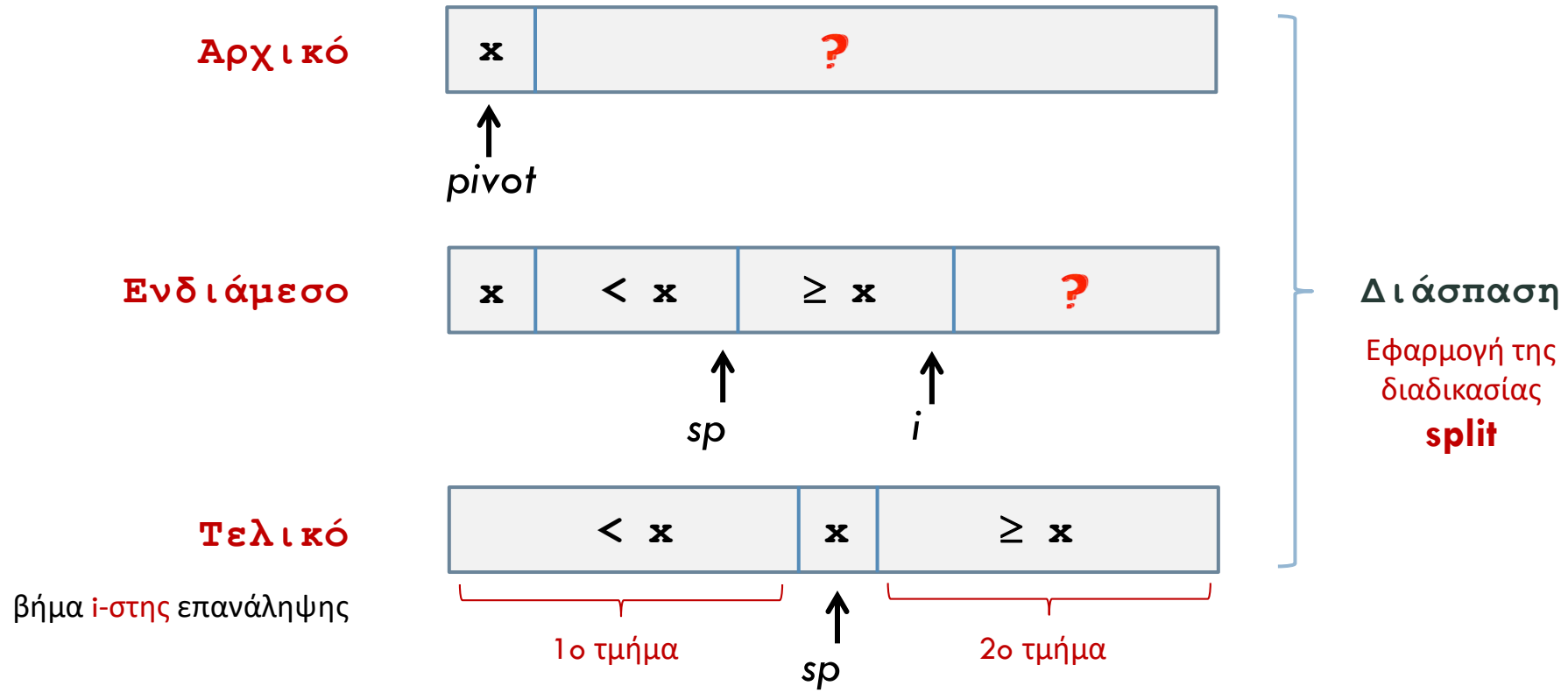
Divide & Conquer



- ✓ Τεχνική **Διαίρει-και-Βασίλευε**
- ✓ **Δύσκολη διαίρεση - *Εύκολη* συνένωση**
- ✓ Υπο-προβλήματα ***Διαφορετικού*** μήκους
- ✓ Χειρίστη Πολυπλοκότητα $\mathbf{O(n^2)}$
- ✓ Μέση Πολυπλοκότητα **O(nlogn)**

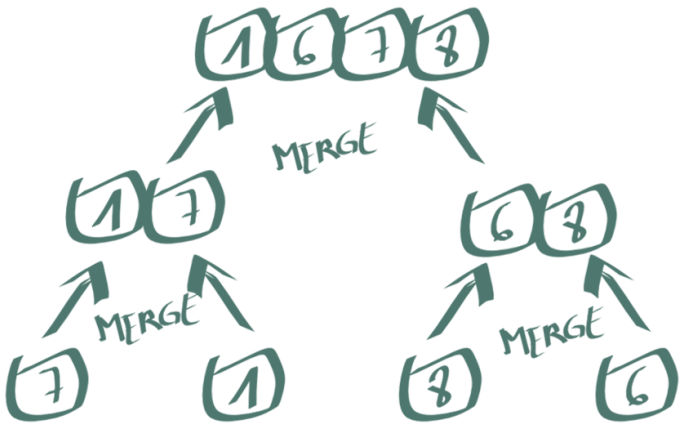
Αλγόριθμος Quick-Sort

Ιδέα Αλγόριθμου !!!



Αλγόριθμος Ταξινόμησης Merge-Sort

Divide & Conquer

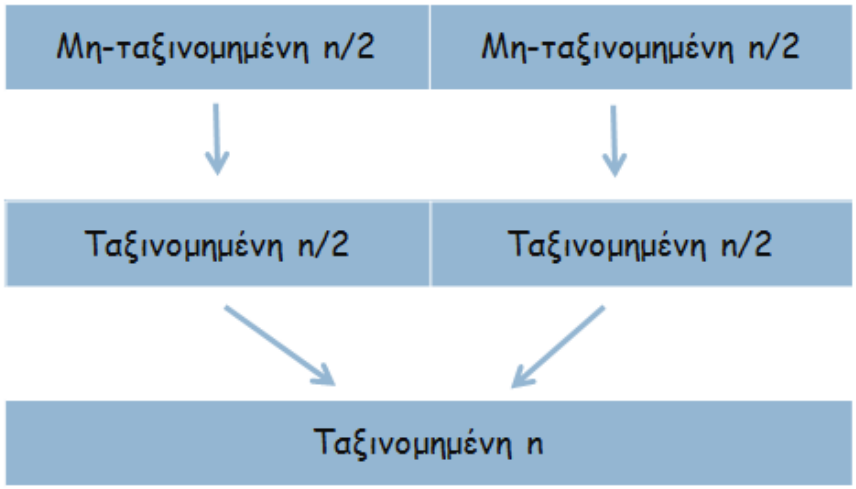


- ✓ Τεχνική **Διαίρει-και-Βασίλευε**
- ✓ ***Εύκολη* διαίρεση – Δύσκολη συνένωση**
- ✓ Υπο-προβλήματα ***ίδιου*** μήκους
- ✓ Χειρίστη Πολυπλοκότητα **O(nlogn)**
- ✓ Μέση Πολυπλοκότητα **O(nlogn)**

Αλγόριθμος Merge-Sort

Ιδέα Αλγόριθμου !!!

- Διαίρεση ακολουθίας εισόδου μήκους n σε δύο υποακολουθίες μήκους n/2.
- Ταξινόμηση των δύο υποακολουθιών (μισού μεγέθους) αναδρομικά.
- Συγχώνευση των δύο ταξινομημένων υποακολουθιών σε μία ταξινομημένη ακολουθία.



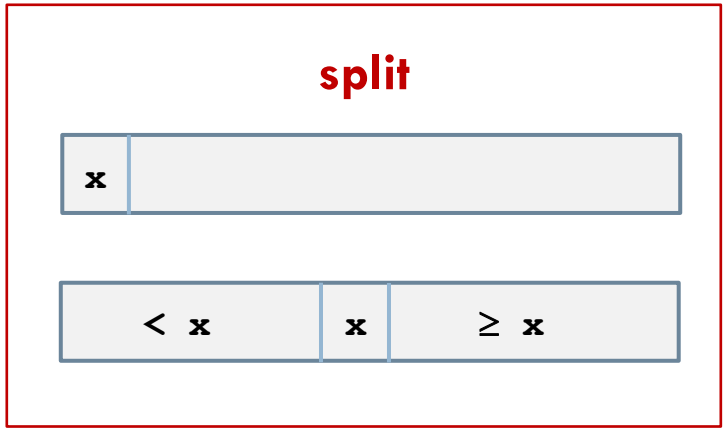
Διαίρεσε 1 (divide)

Ταξινόμησε 2 (conquer)

Συγχώνευσε 3 (combine)

Ευστάθεια Αλγορίθμων Quick-Sort και Marge-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```



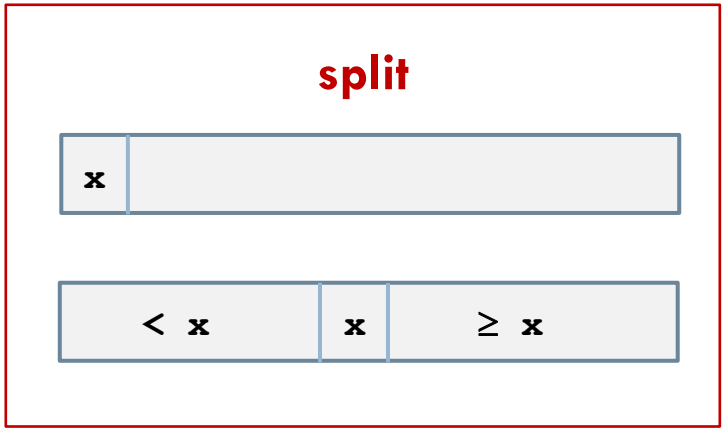

```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

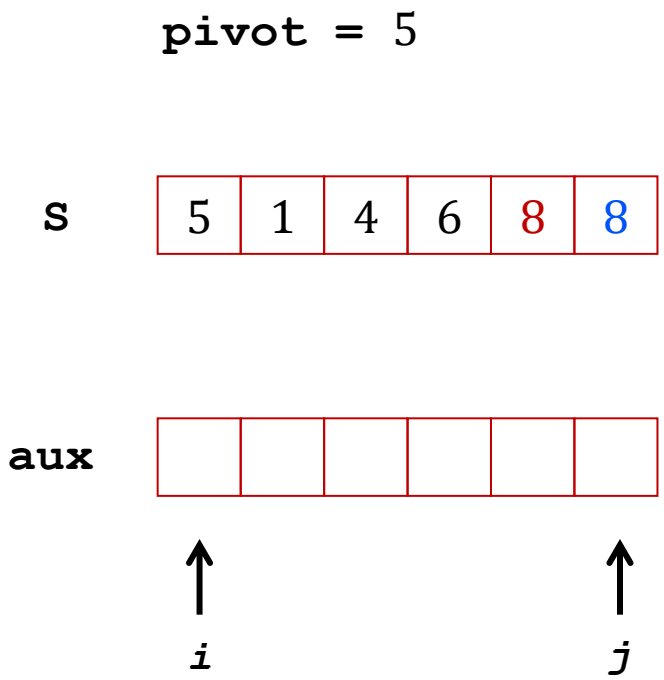
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```

Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```

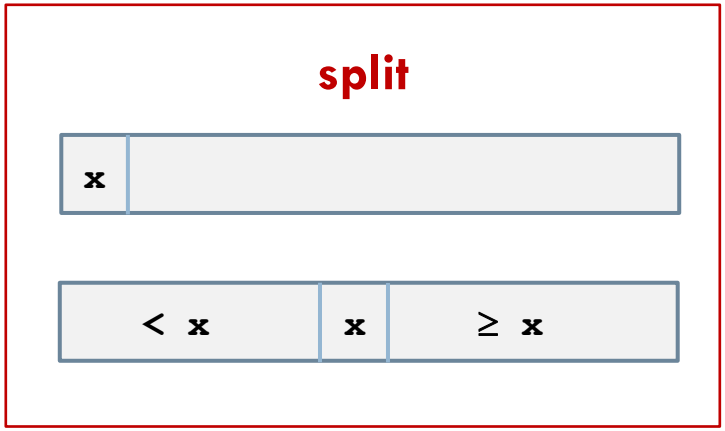



```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

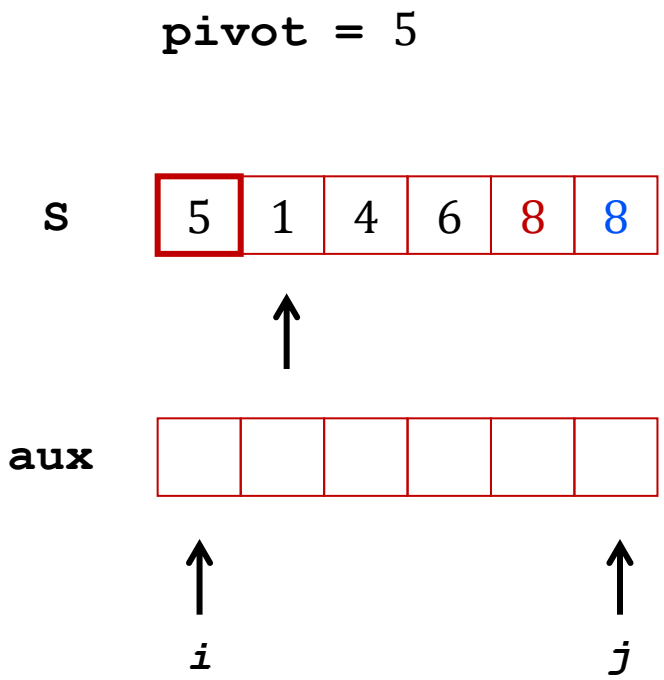


Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```

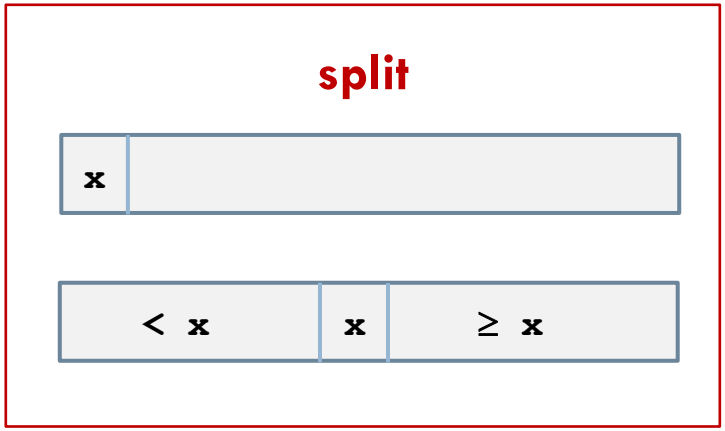



```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

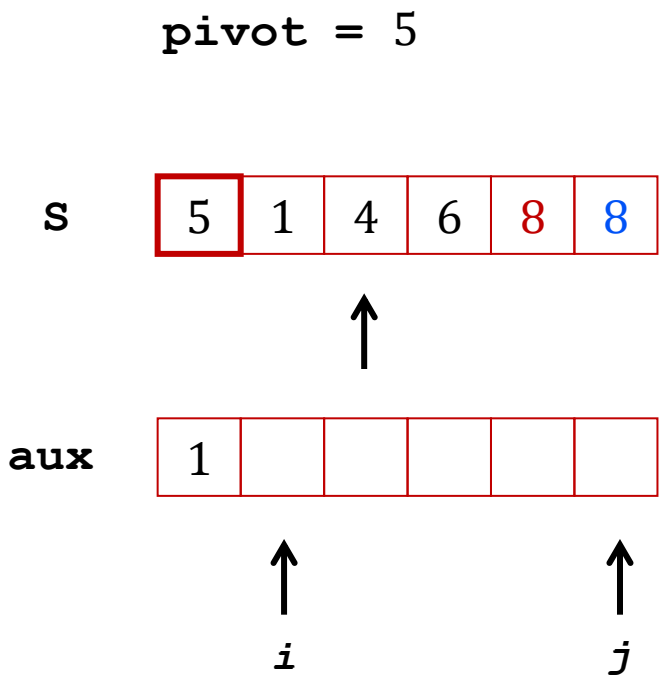


Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```

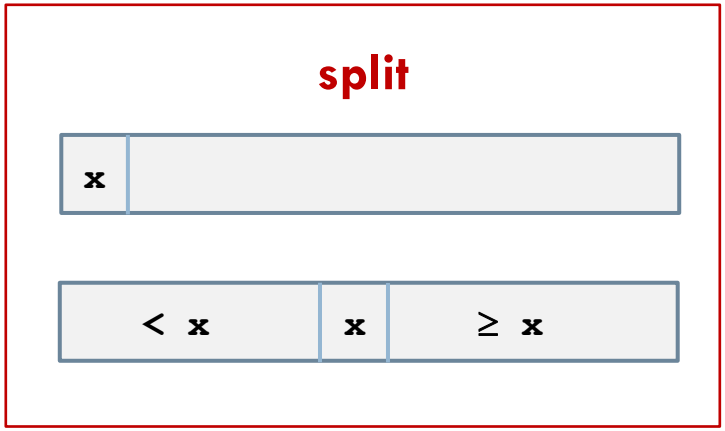



```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

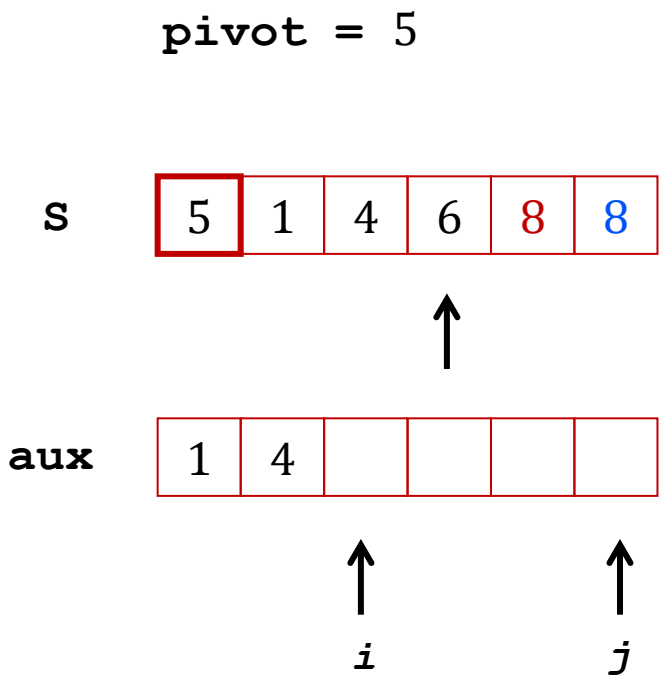


Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```

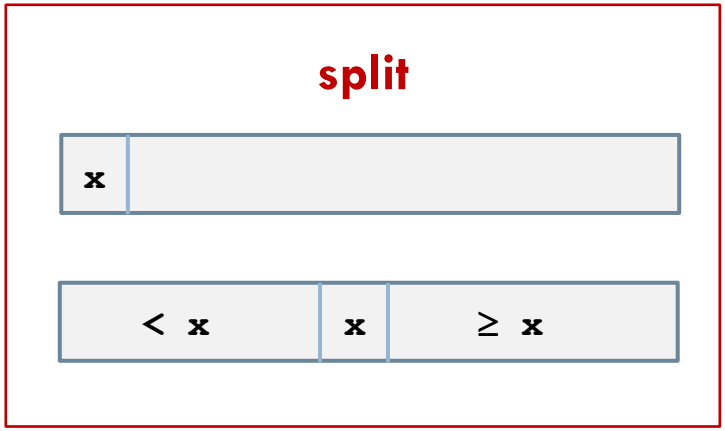



```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

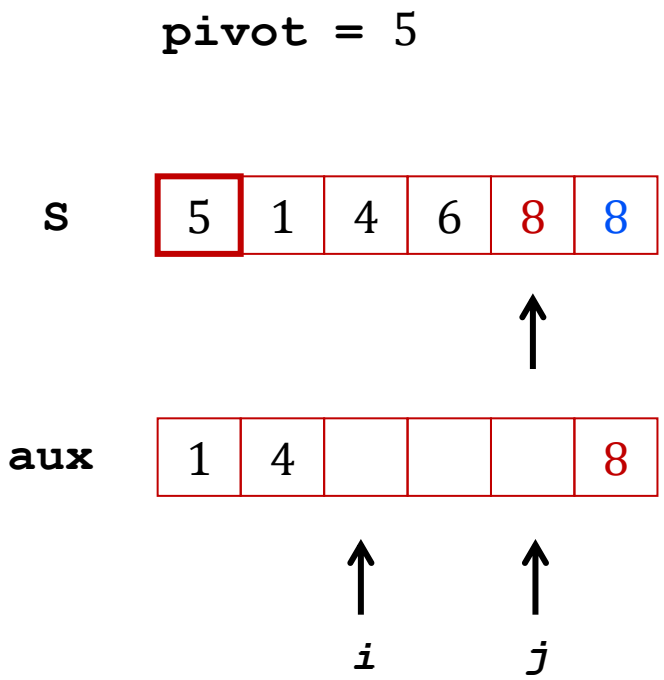


Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```

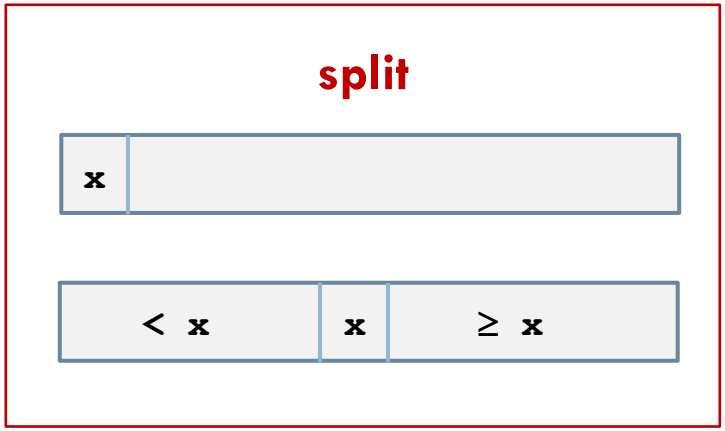



```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

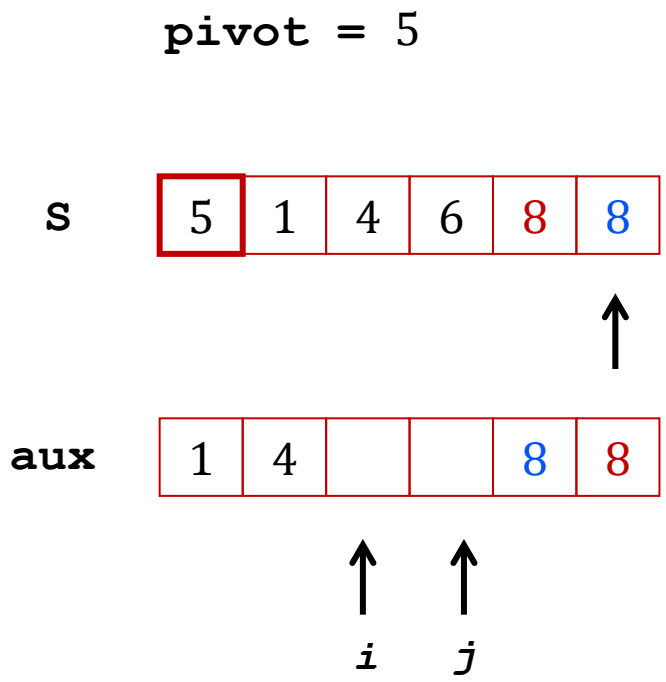


Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```



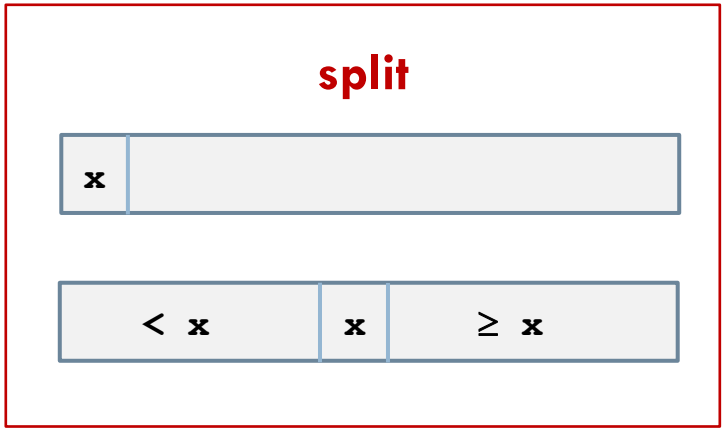

```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Μη-ευσταθής Αλγόριθμος

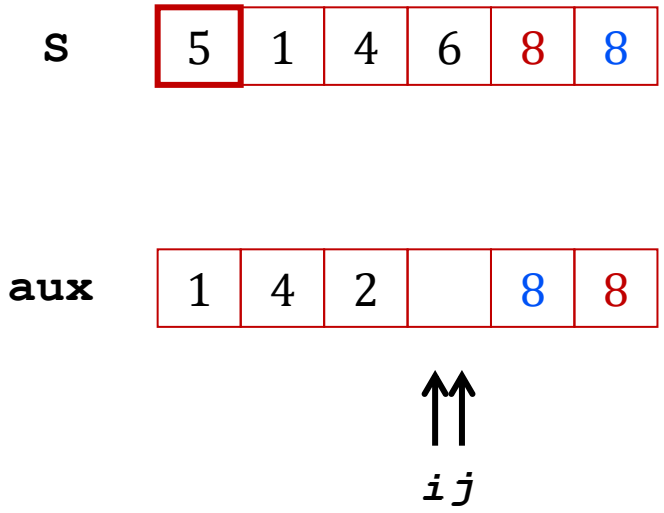
Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```




```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

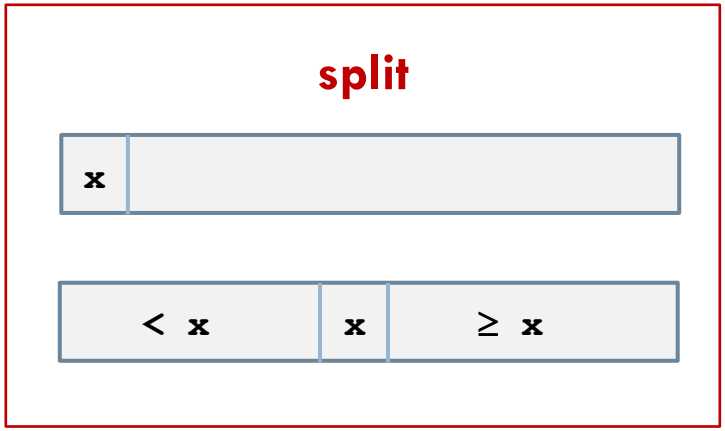
pivot = 5



Μη-ευσταθής Αλγόριθμος

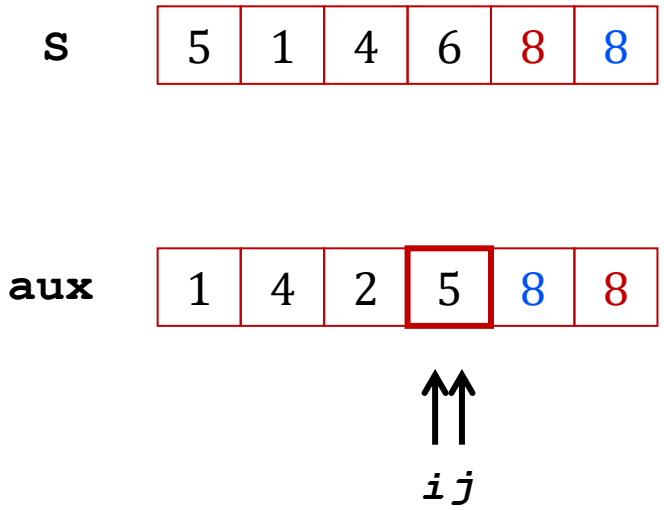
Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```




```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

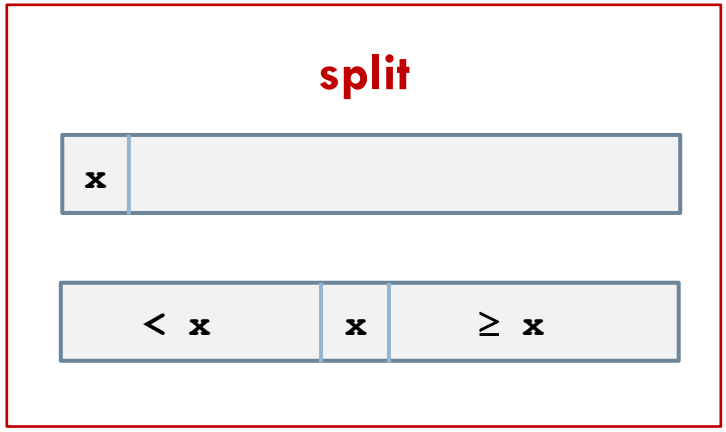
pivot = 5



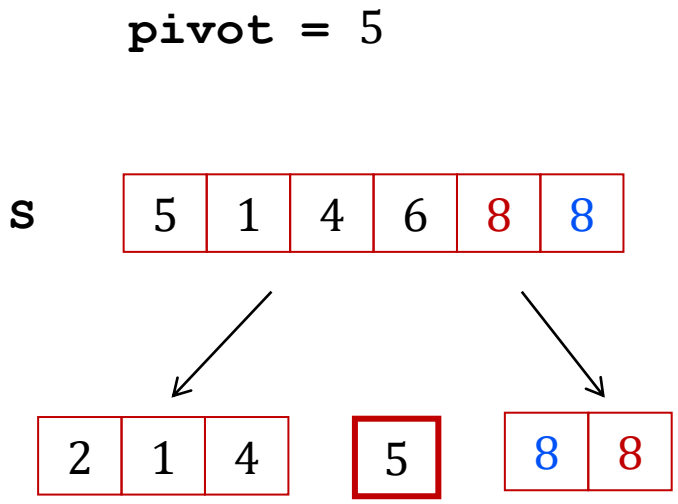
Μη-ευσταθής Αλγόριθμος

Ευστάθεια Quick-Sort

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
```




```
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



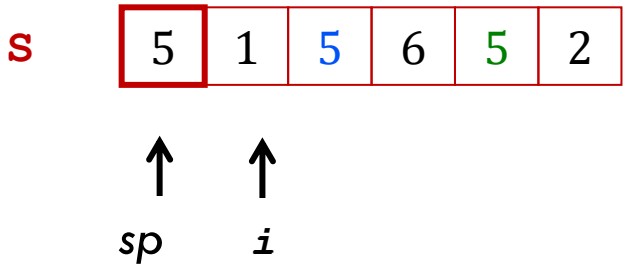
Μη-ευσταθής Αλγόριθμος

Αλγόριθμος Quick-Sort

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

pivot = 5



sp : splitpt

Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

pivot = 5

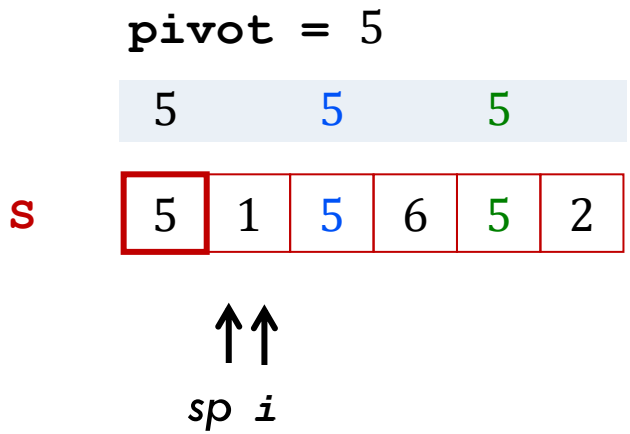
	5		5		5	
S	5	1	5	6	5	2
	↑ sp		↑ i			

sp : splitpt

Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

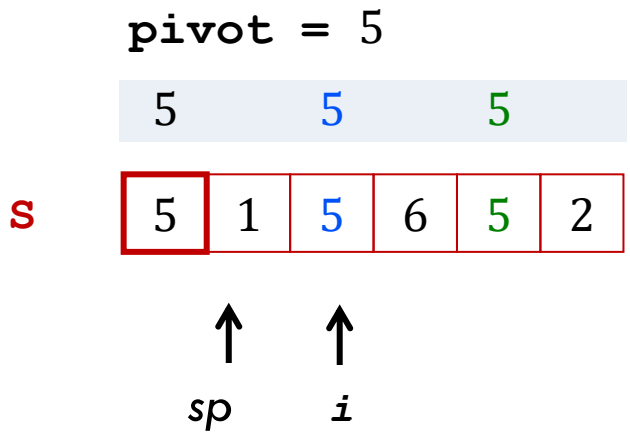
```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

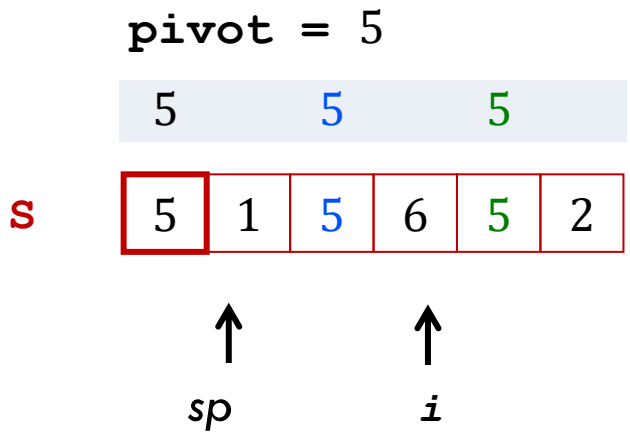
```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

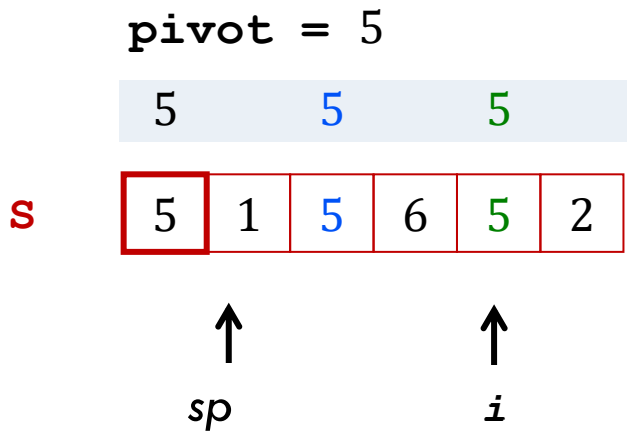


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

Ευστάθεια Quick-Sort (in-place)

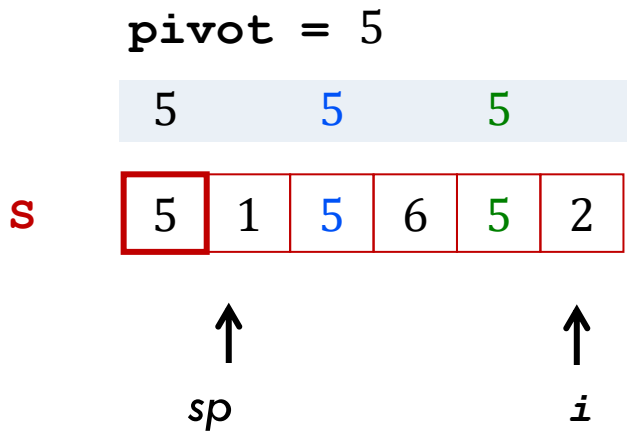
```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

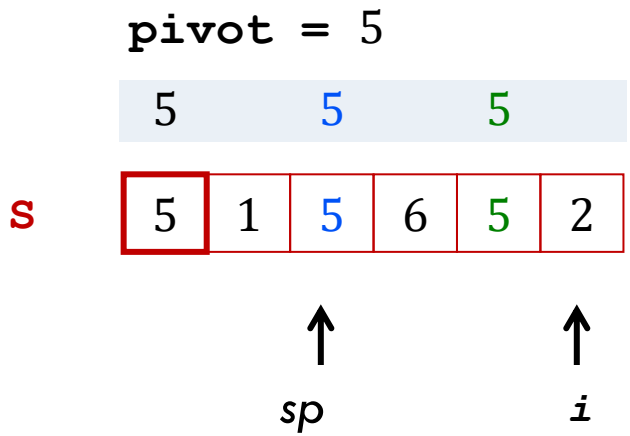
```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

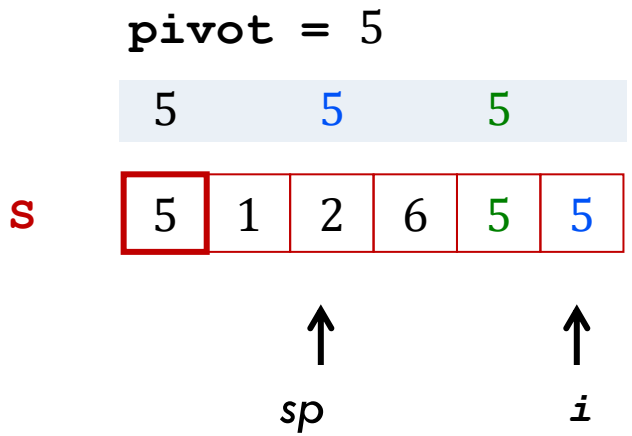
```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

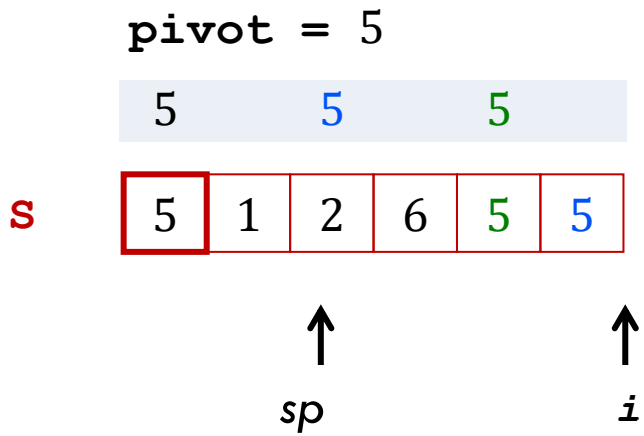


Μη-ευσταθής Αλγόριθμος

Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

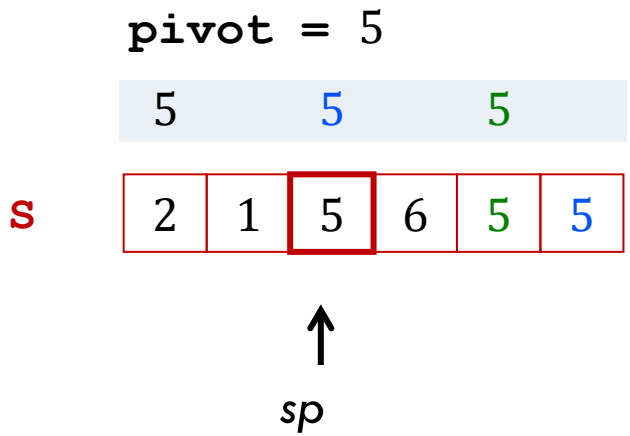


Μη-ευσταθής Αλγόριθμος

Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

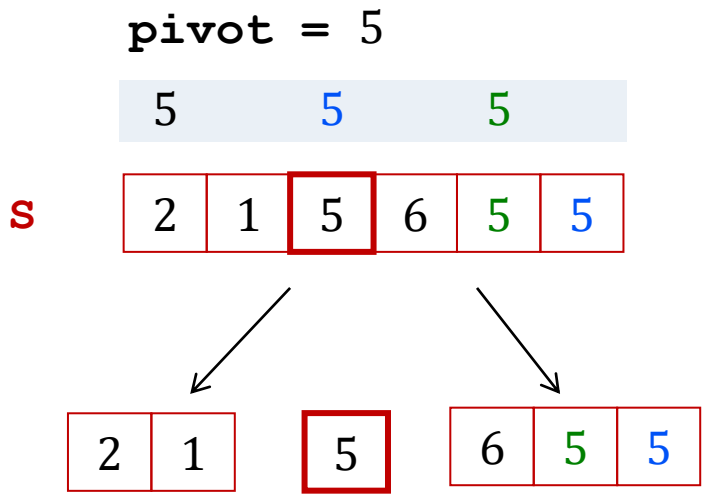


Μη-ευσταθής Αλγόριθμος

Split : επί τόπου διάσπαση

Ευστάθεια Quick-Sort (in-place)

```
Algorithm Quick-Sort(S, first, last)
begin
(1) if first < last then
        pivot ← S[first]
        splitpt ← first                          split
        for i ← first+1 to last do
           if S[i] < pivot then
                splitpt ← splitpt + 1
                Swap(S[splitpt], S[i])
           end;
        end;
        Swap (S[first], S[splitpt])
    end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



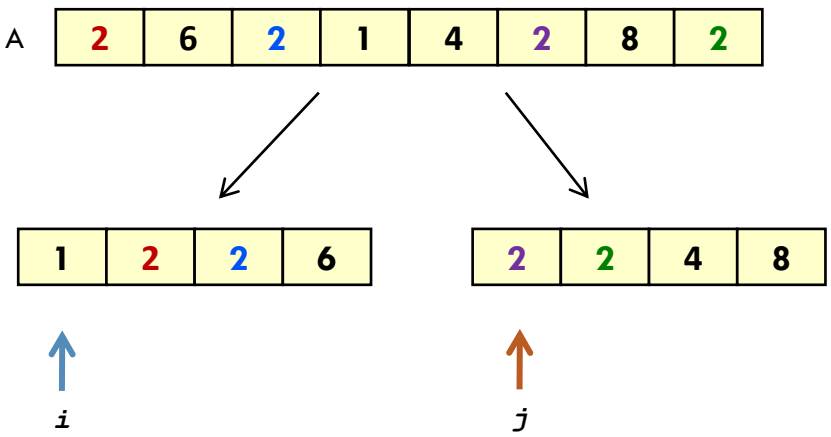
Μη-ευσταθής Αλγόριθμος

Split : επί τόπου διάσπαση

Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

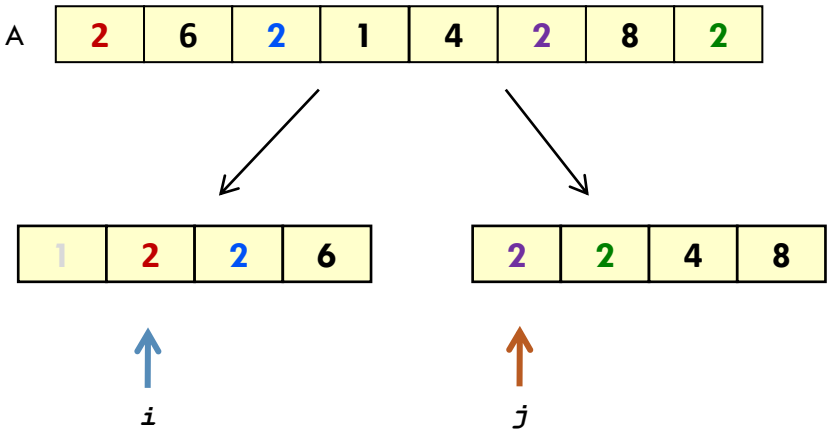
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

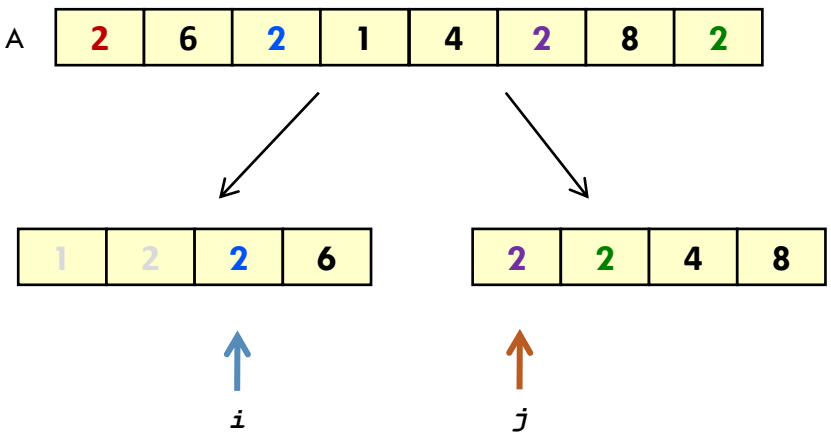
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

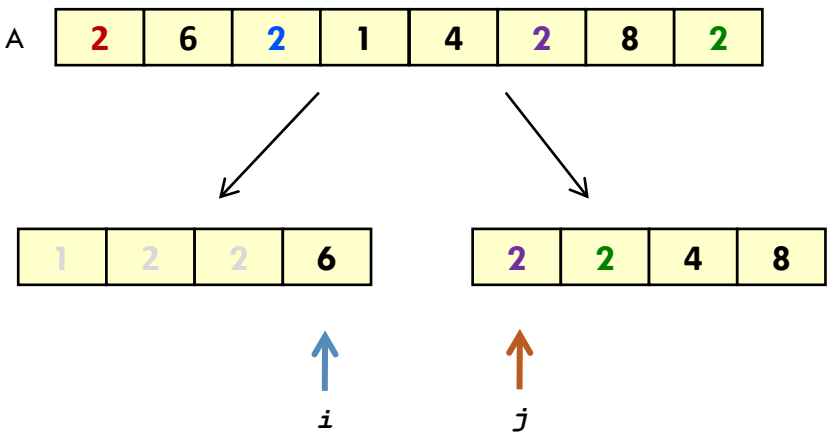
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

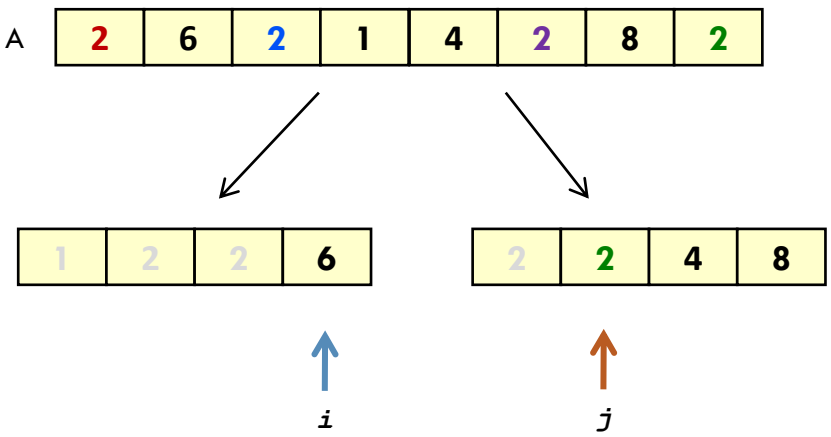
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

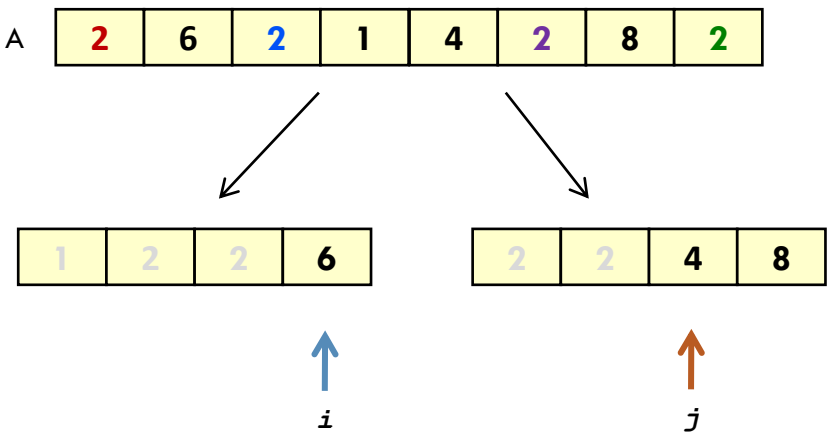
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

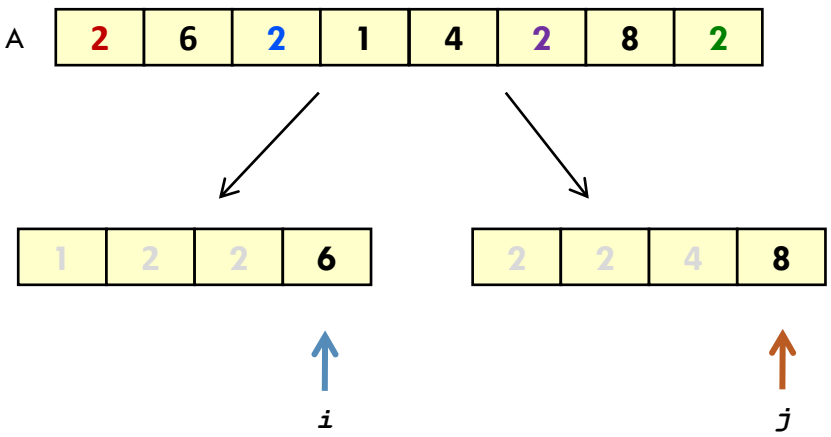
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

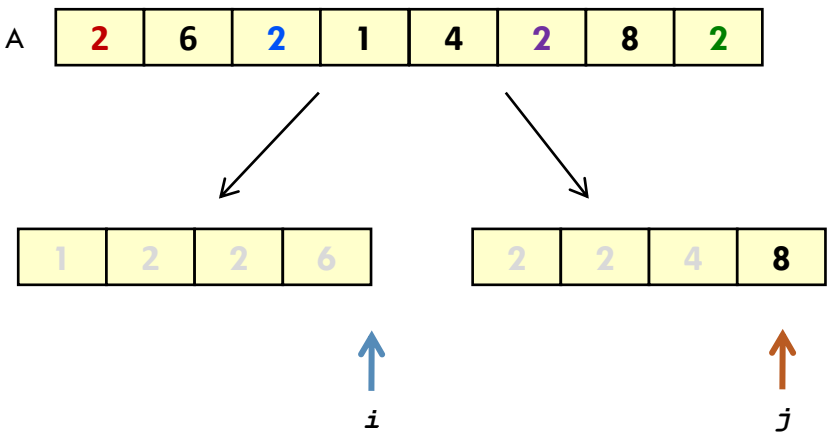
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

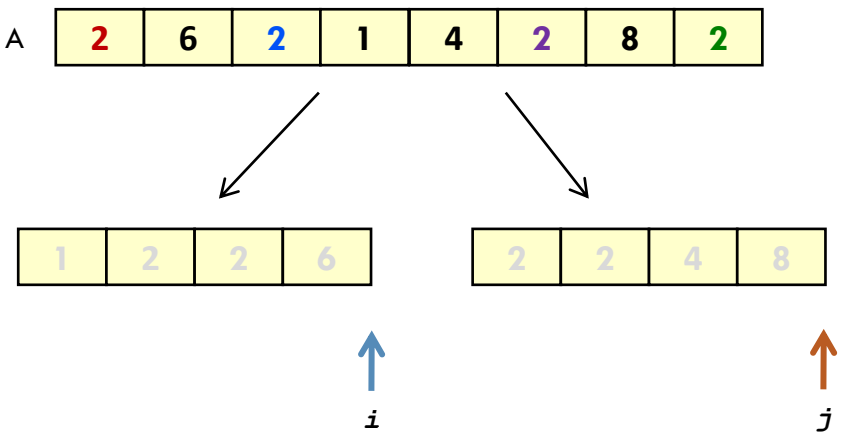
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```

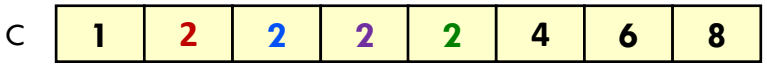


Ευστάθεια Merge-Sort

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left ≥ right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

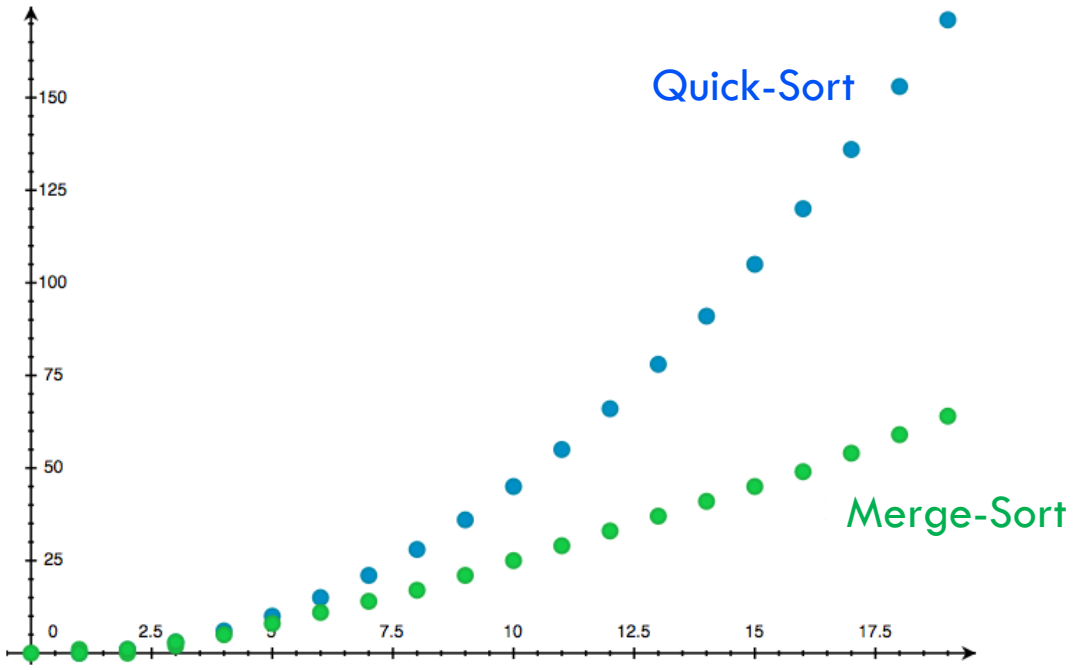
```
Algorithm Merge()
i ← j ← 1
while (A ≠ ∅ ∧ B ≠ ∅)
  if (a_i ≤ b_j) προσάρτηση a_i στη C και i ← i+1
  else{a_i > b_j}προσάρτηση b_j στη C και j ← j+1
end
  προσάρτηση των υπόλοιπων στοιχείων της
  μη-άδειας ακολουθίας A ή B στη C
end
```



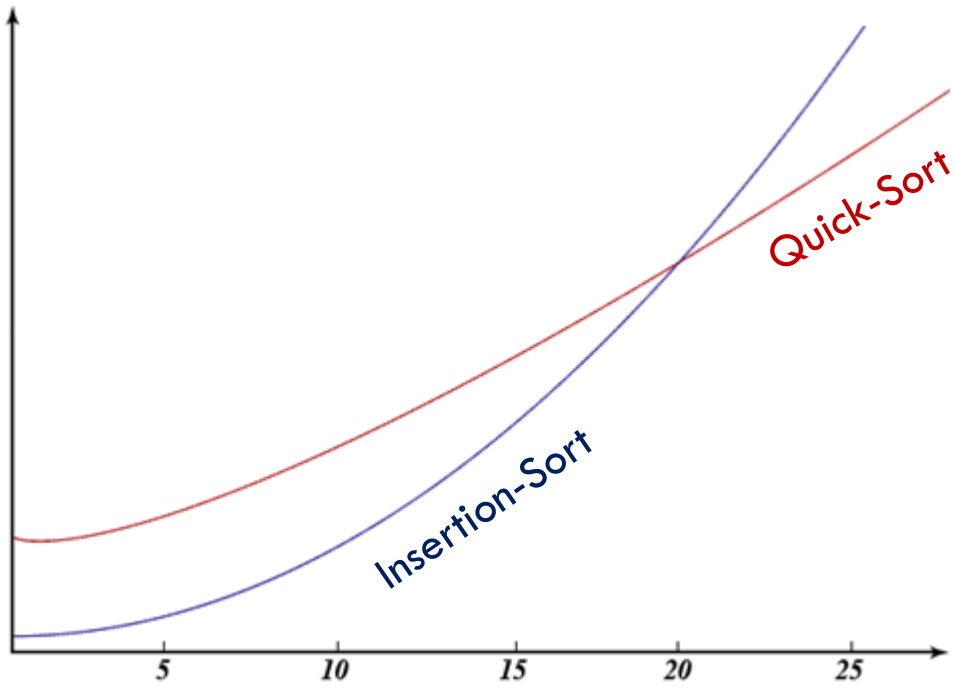


Ευσταθής Αλγόριθμος

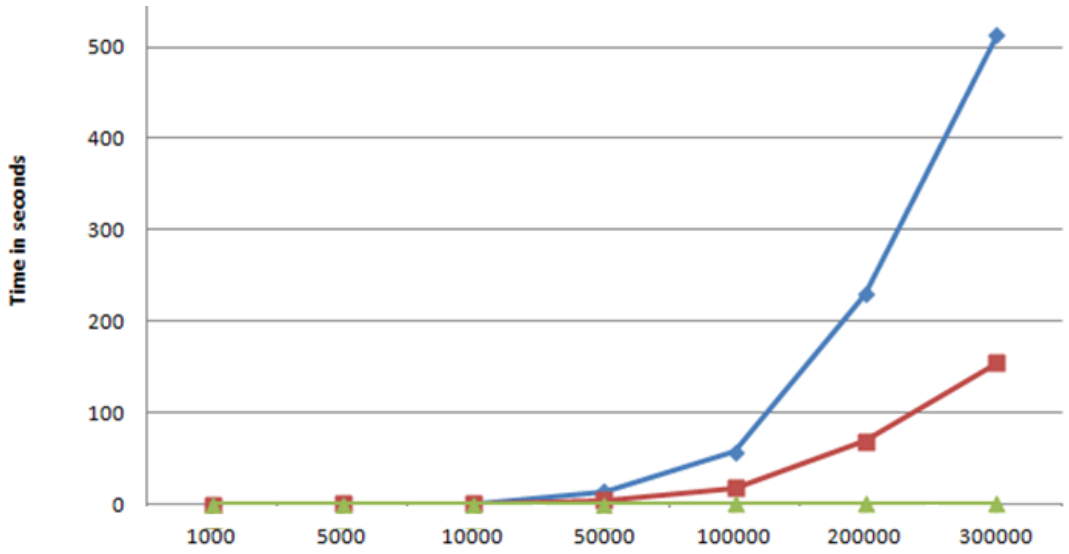
Σύγκριση Αλγορίθμων Ταξινόμησης



Σύγκριση Αλγορίθμων Ταξινόμησης

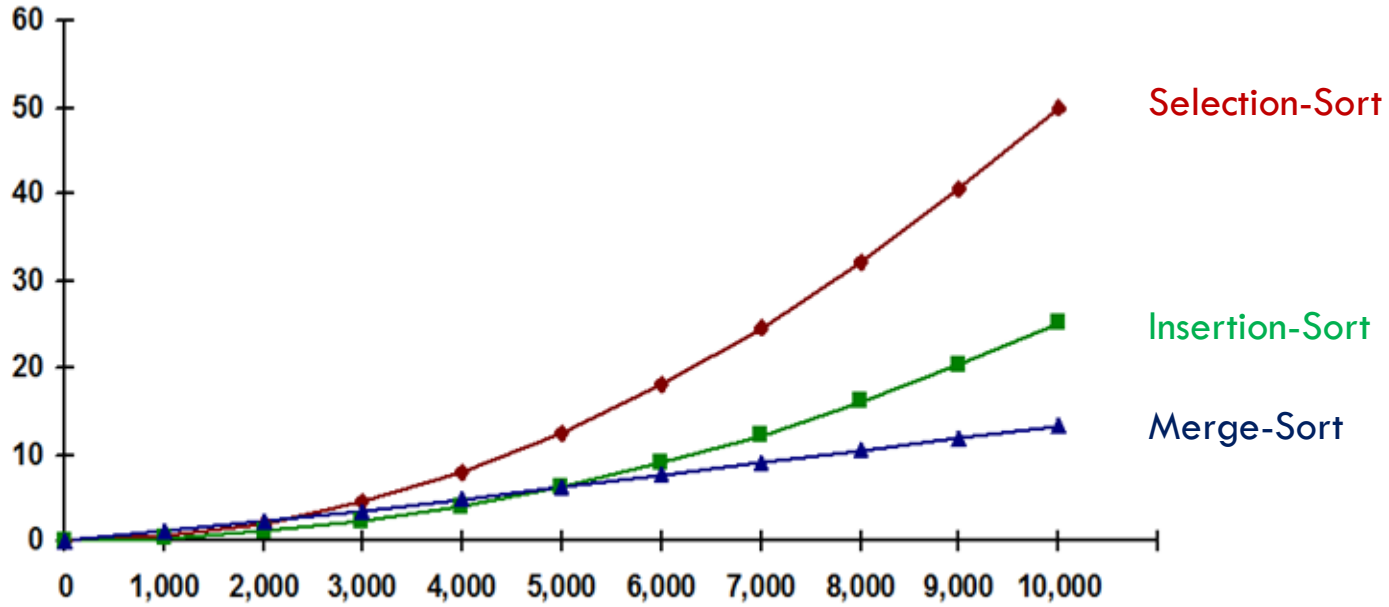


Σύγκριση Αλγορίθμων Ταξινόμησης

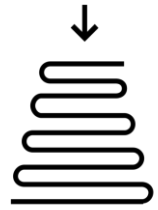


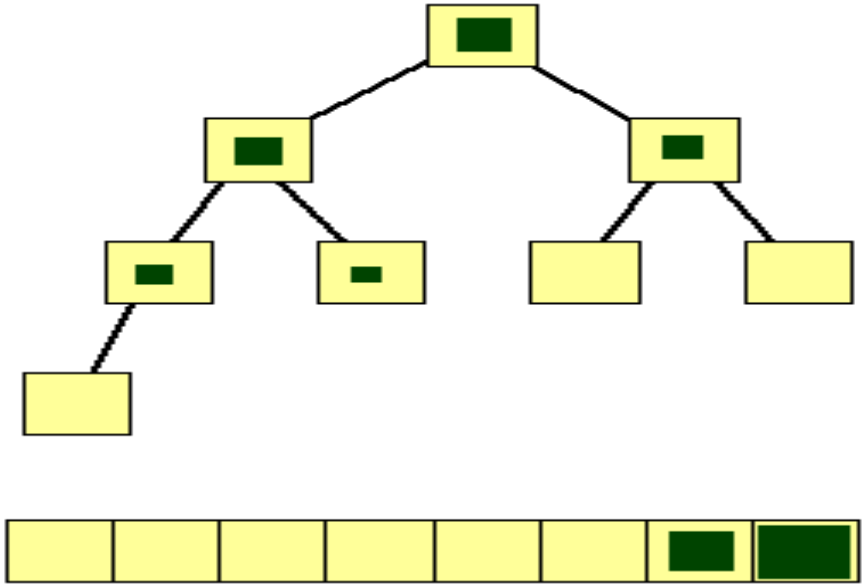
Bubble Sort	0	0.145	0.6	14.05	56.9	229.44	513.34
Insertion Sort	0	0.045	0.18	4.25	17.62	69.12	154.22
Quick Sort	0	0	0	0.01	0.02	0.05	0.065

Σύγκριση Αλγορίθμων Ταξινόμησης



Αλγόριθμος Heap-Sort



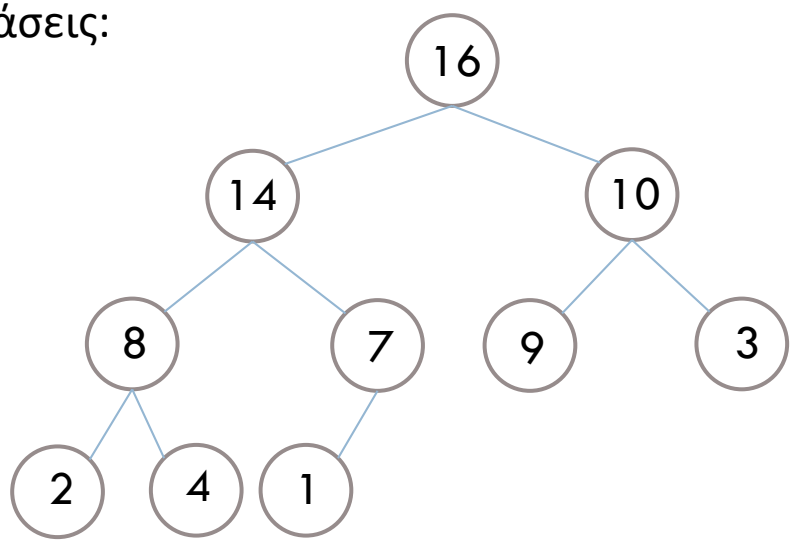


Αλγόριθμος Heap-Sort

Ιδέα Αλγόριθμου !!!

- Ο αλγόριθμος προτάθηκε από τον Williams το 1964.
- Βασίζεται στην ιδιότητα του σωρού: το max (ή min) στοιχείο βρίσκεται στην ρίζα.

- Ο αλγόριθμος Heap-Sort εκτελείται σε δύο φάσεις:
  (1) τη φάση της δημιουργίας του σωρού, και
  (2) τη φάση της επεξεργασίας του σωρού.



Σωρός Μεγίστων :

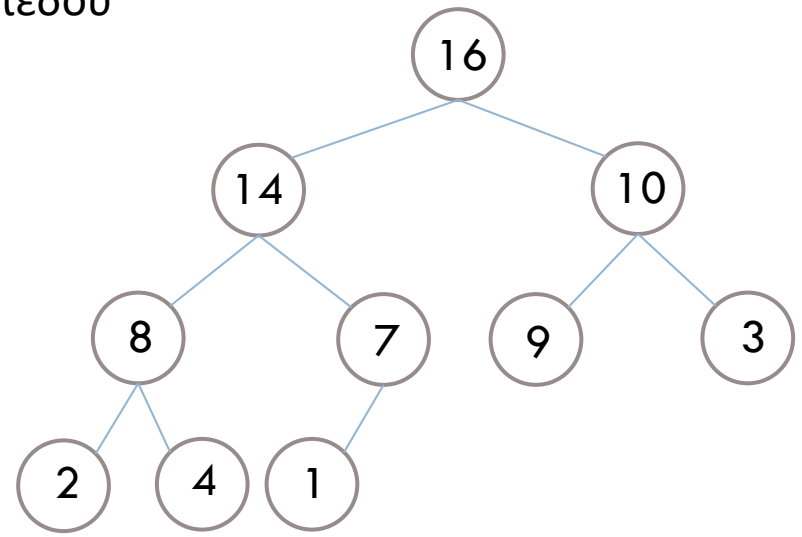
Αλγόριθμος Heap-Sort

Ιδιότητες Σωρού!!!

- Όλοι οι τερματικοί κόμβοι (φύλλα) του σωρού βρίσκονται στο h-1 και h επίπεδο.
- Όλες οι διαδρομές από την ρίζα σε ένα τερματικό κόμβο του h είναι **αριστερά** όλων των διαδρομών από την ρίζα σε ένα τερματικό κόμβο του h-1 επιπέδου.
- Ο δεξιότερος εσωτερικός κόμβος του h-1 επιπέδου έχει αριστερό παιδί.
- *Ιδιότητα μερικής διάταξης (partial order)*:

  Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του !

Σωρός Μεγίστων :



Αλγόριθμος Heap-Sort

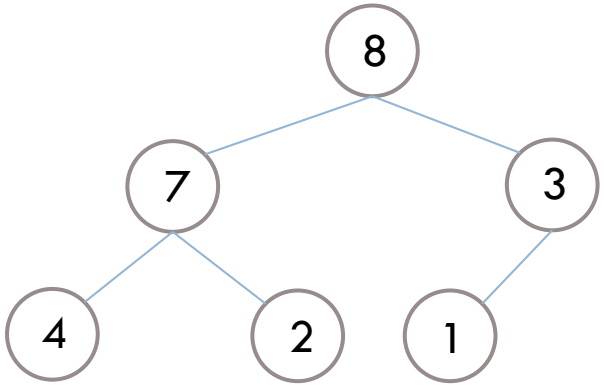
```
Algorithm Heap-sort
begin
```

Διάγραψε τη ρίζα (**x = max**) του σωρού με στοιχεία

S[1,i]

και την τιμή της καταχώρησε τη στη θέση **i** του πίνακα **A[1,n]**

```
    end
    return S
end
```

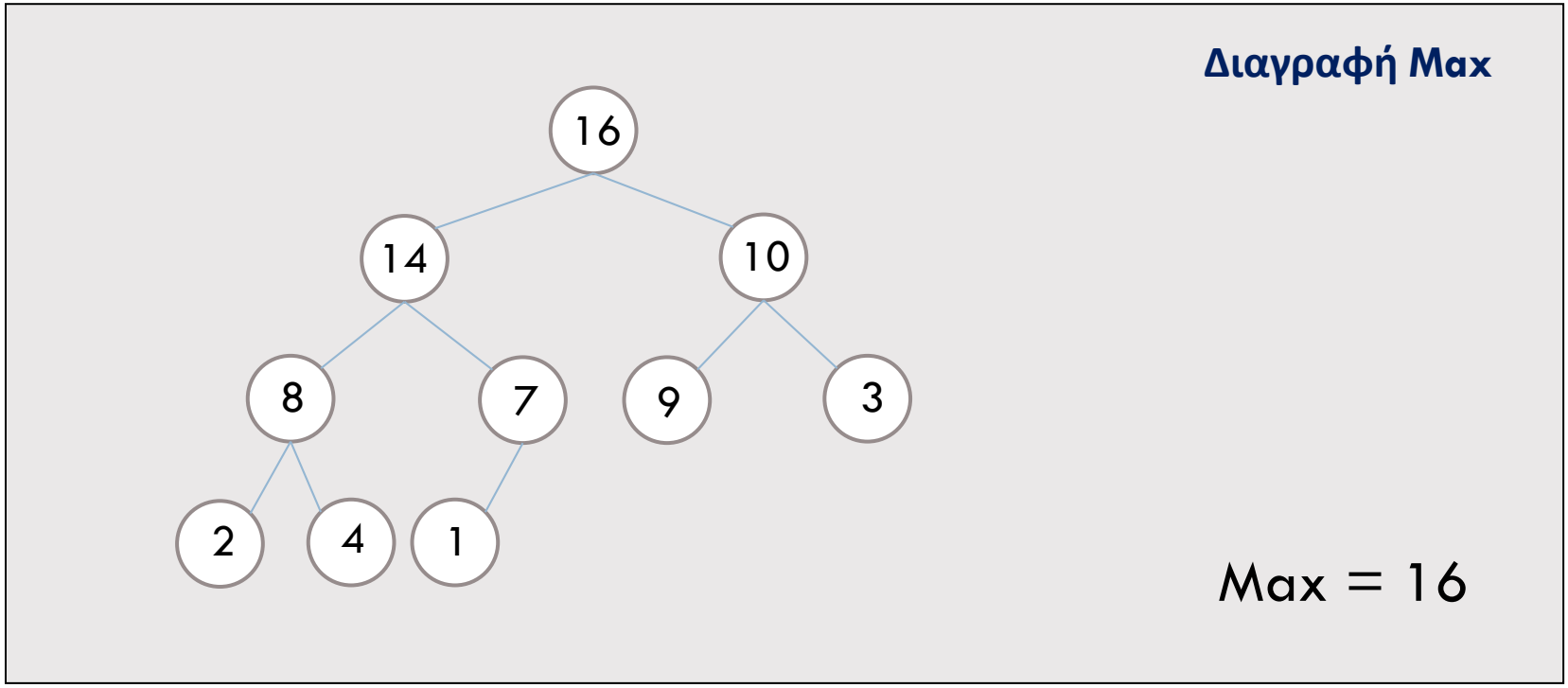


Διαγραφή Max = 8

						9	10	14	16

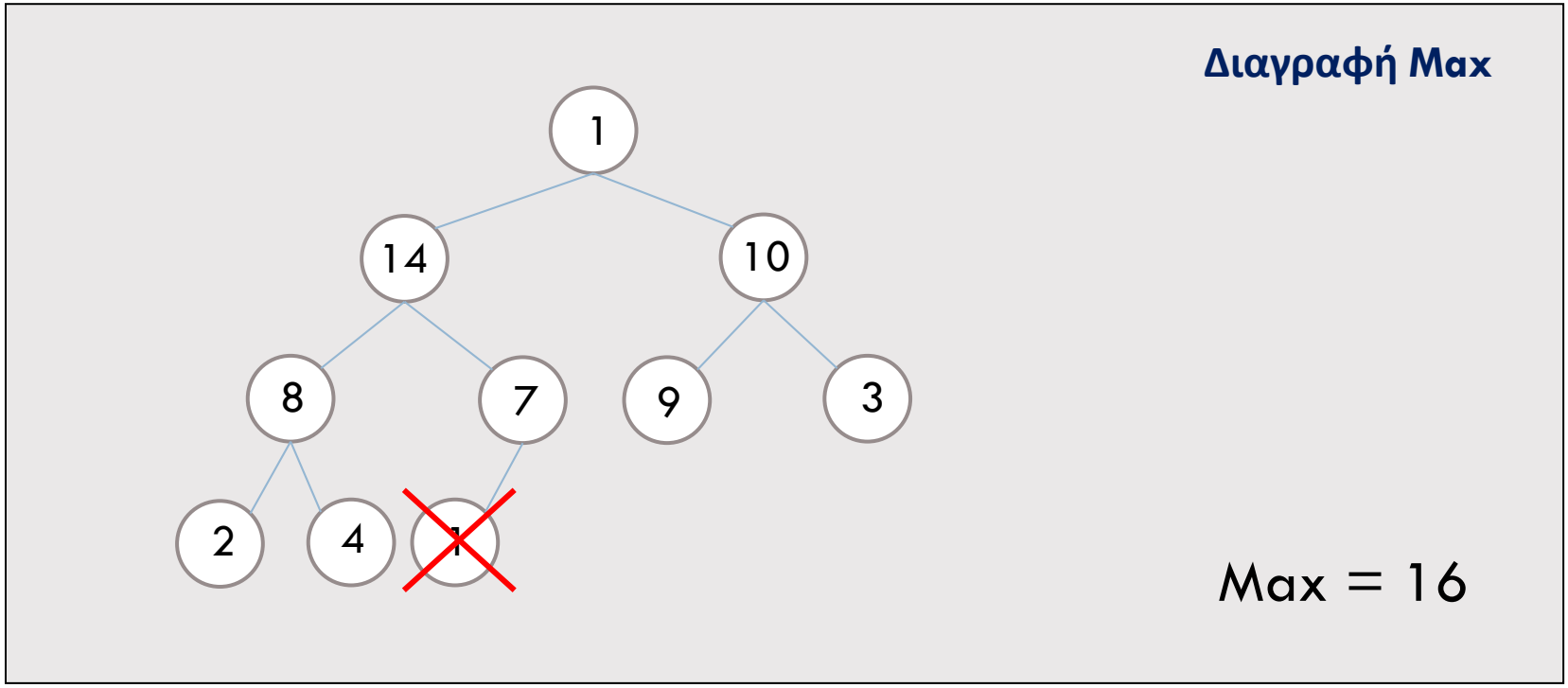
Αλγόριθμος Heap-Sort

Ας θυμηθούμε…



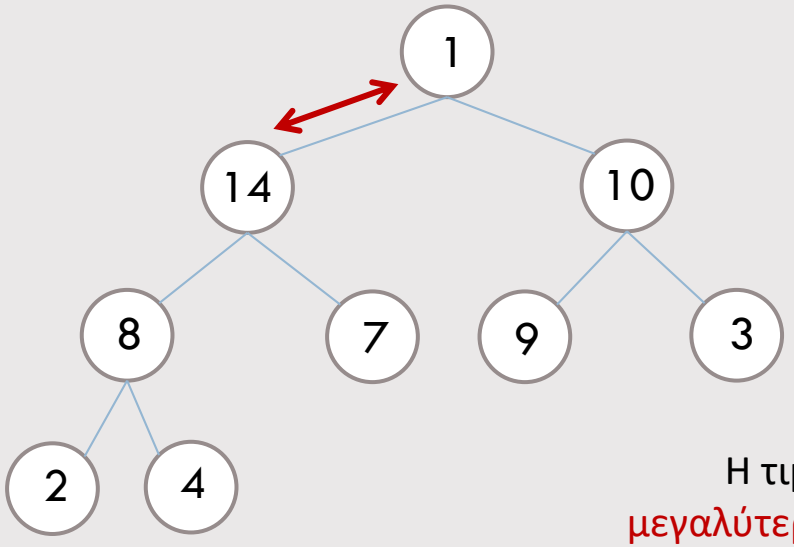
Αλγόριθμος Heap-Sort

Ας θυμηθούμε…



Αλγόριθμος Heap-Sort

Ας θυμηθούμε…



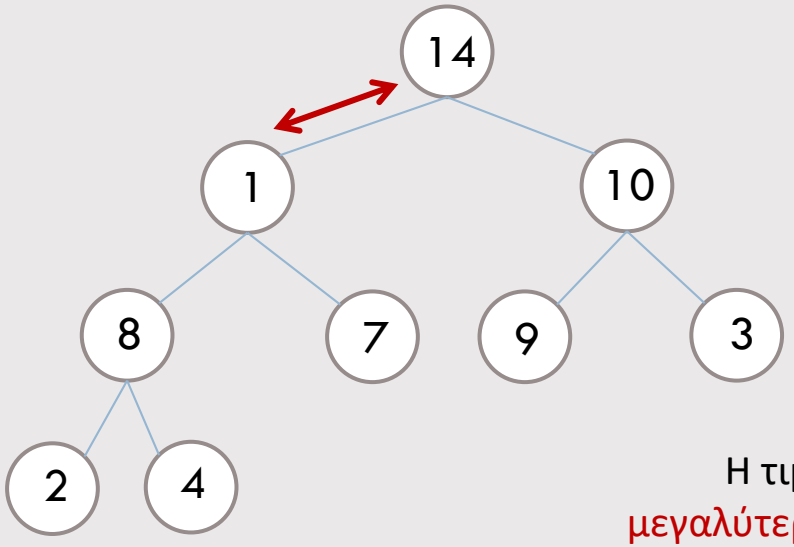
Διαγραφή Max

Αποκατάσταση Heap

Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του

Αλγόριθμος Heap-Sort

Ας θυμηθούμε…



Διαγραφή Max

Αποκατάσταση Heap

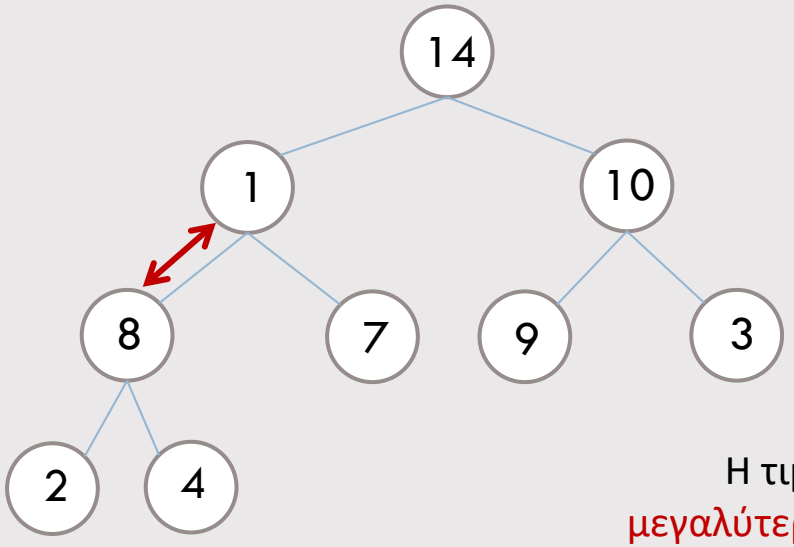
Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του

Αλγόριθμος Heap-Sort

Ας θυμηθούμε…

Διαγραφή Max

Αποκατάσταση Heap



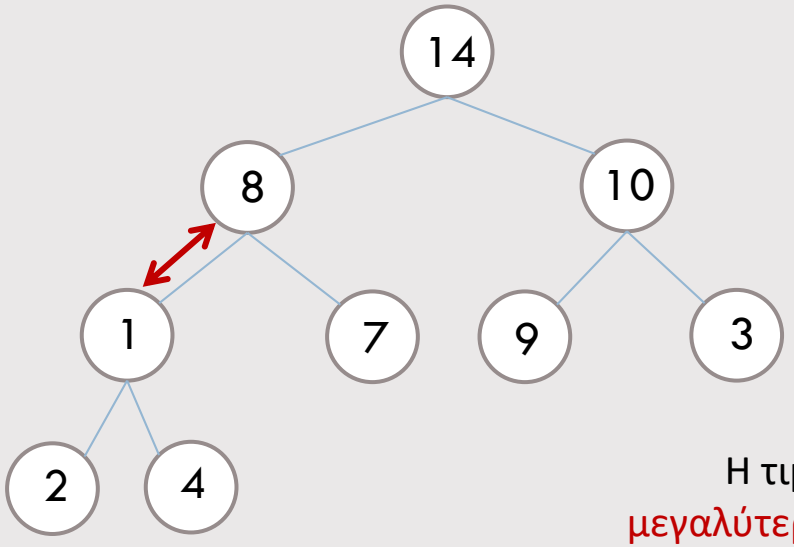
Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του

Αλγόριθμος Heap-Sort

Ας θυμηθούμε…

Διαγραφή Max

Αποκατάσταση Heap



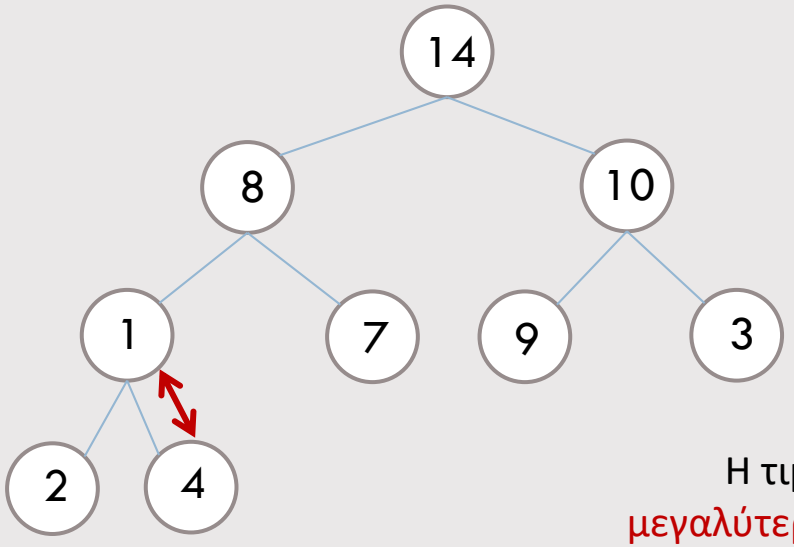
Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του

Αλγόριθμος Heap-Sort

Ας θυμηθούμε…

Διαγραφή Max

Αποκατάσταση Heap



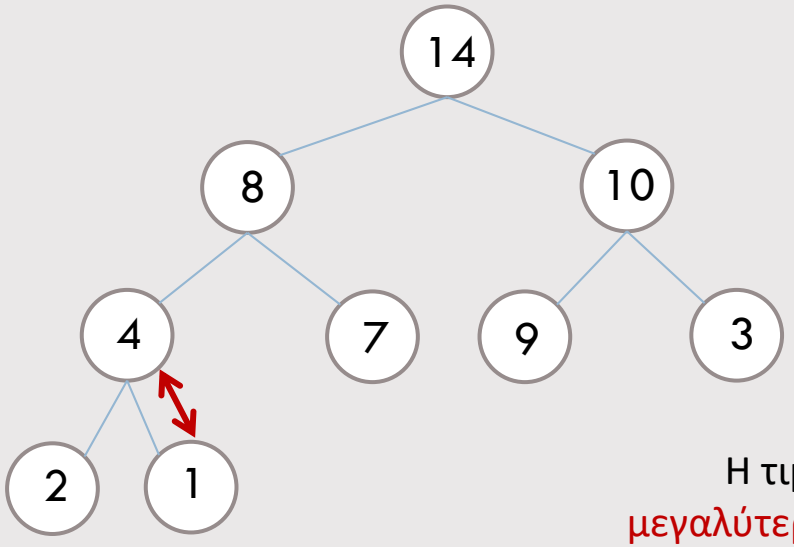
Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του

Αλγόριθμος Heap-Sort

Ας θυμηθούμε…

Διαγραφή Max

Αποκατάσταση Heap



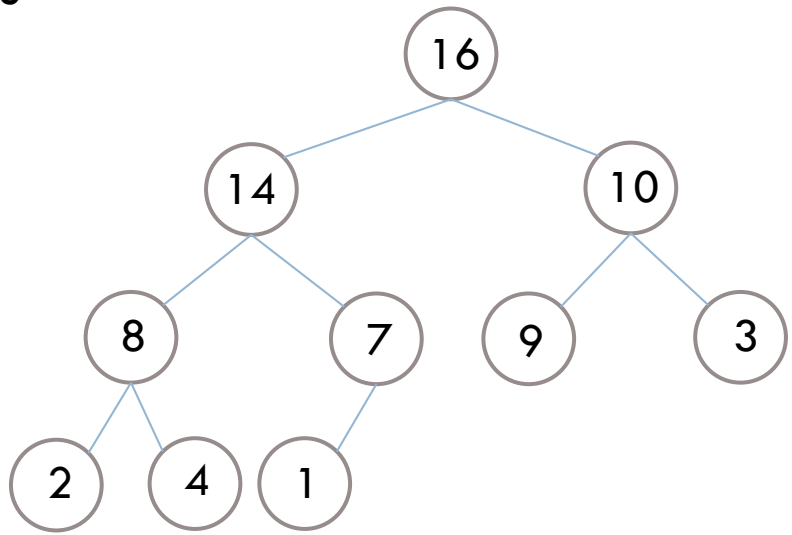
Η τιμή του κλειδιού ενός κόμβου είναι μεγαλύτερη ή ίση (σωρός μεγίστων) από τις τιμές των κλειδιών των παιδιών του

Αλγόριθμος Heap-Sort

Είσοδος Αλγόριθμου

Σωρός Μεγίστων

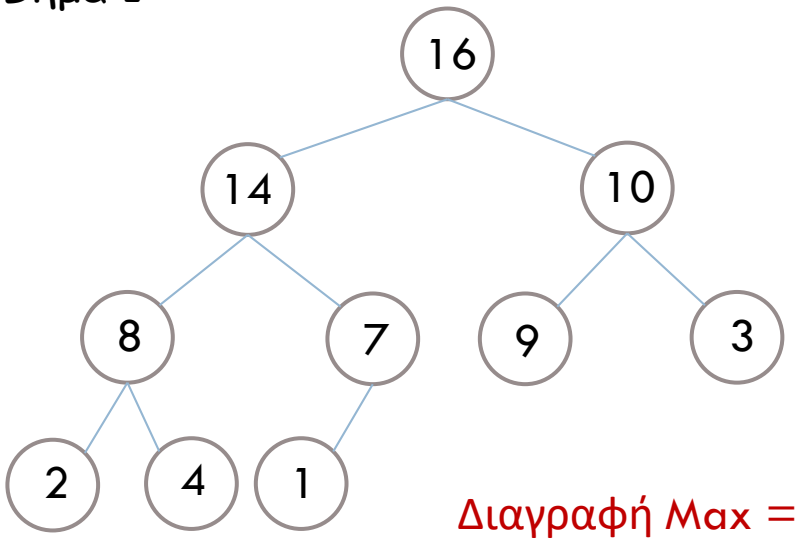
Παράδειγμα



Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 1

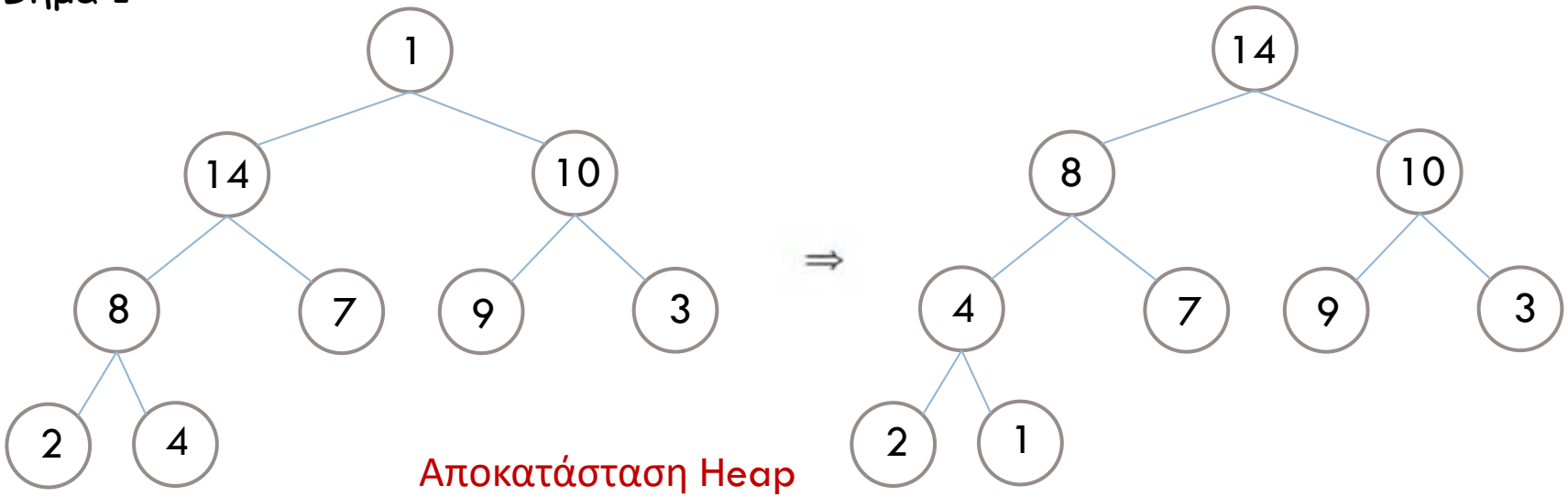


Διαγραφή Max =16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 1

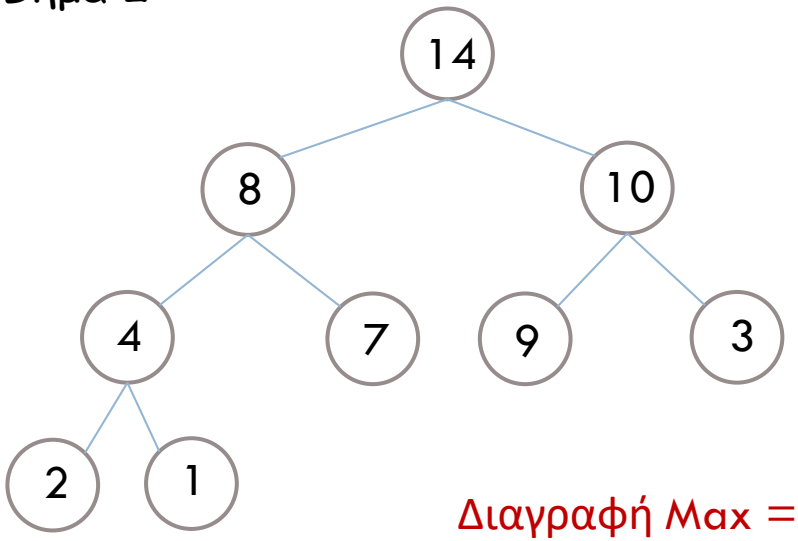


Αποκατάσταση Heap

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 2

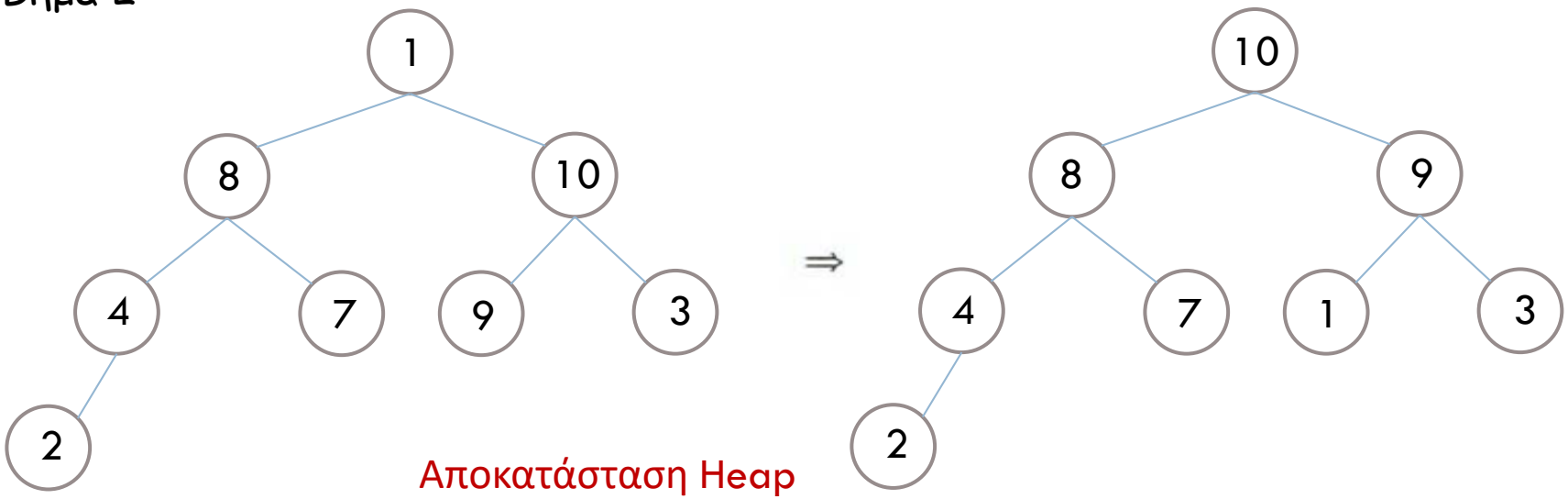


Διαγραφή Max = 14

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 2

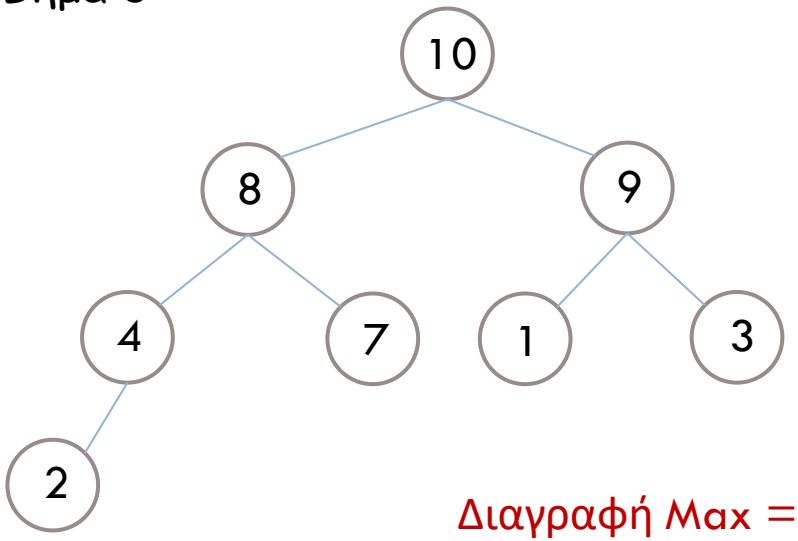


Αποκατάσταση Heap

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 3



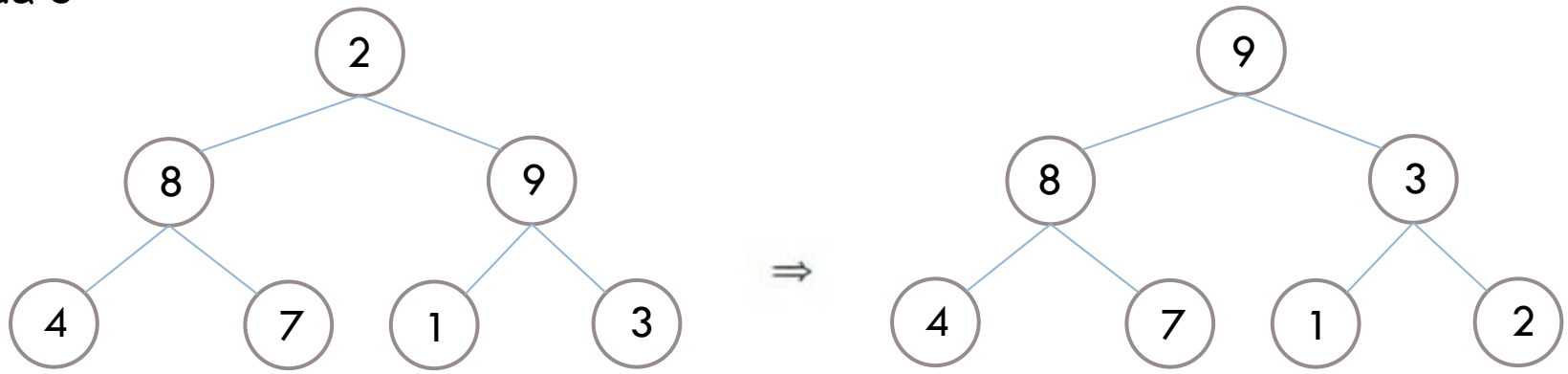
Διαγραφή Max = 10

								14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 3

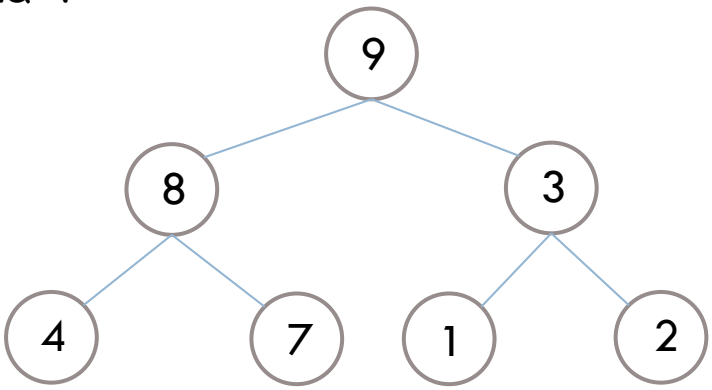


Αποκατάσταση Heap

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 4



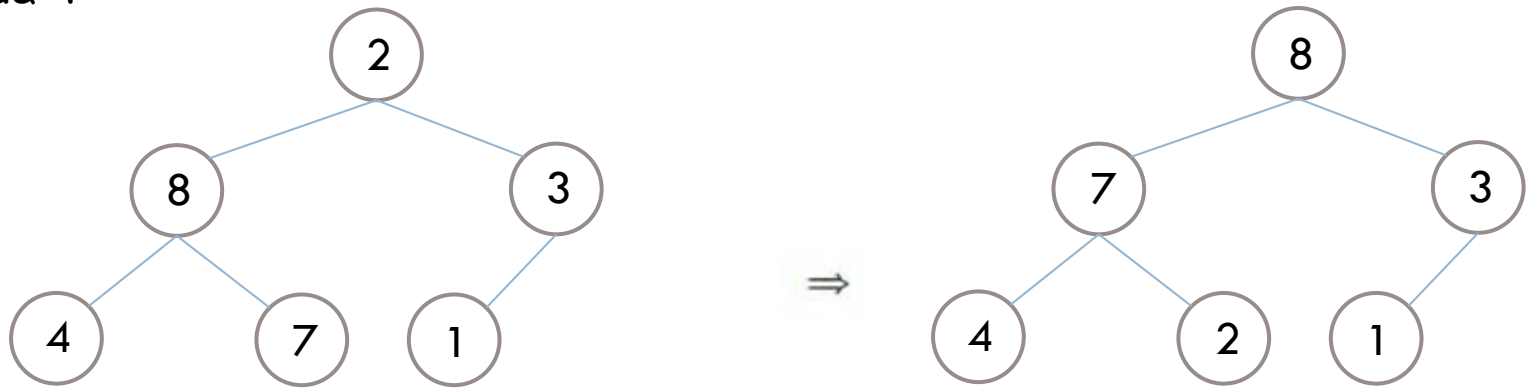
Διαγραφή Max = 9

							10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 4



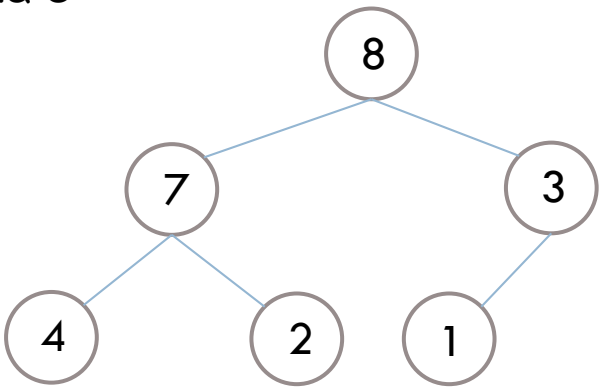
Αποκατάσταση Heap

						9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 5



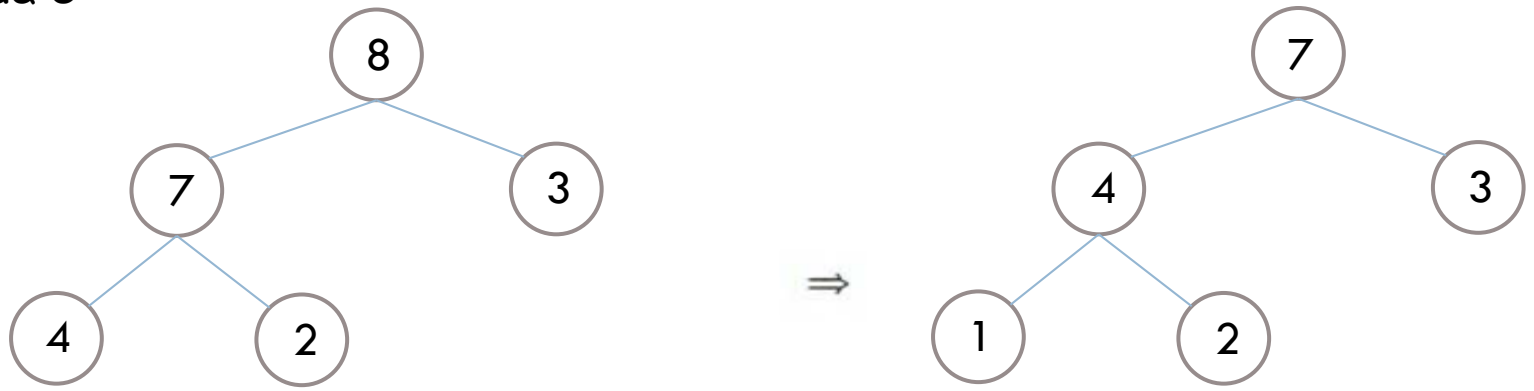
Διαγραφή Max = 8

						9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 5



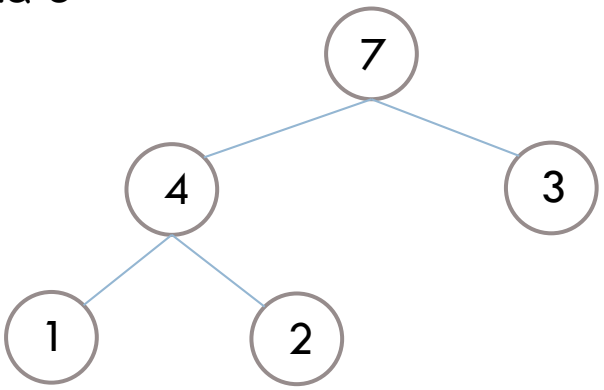
Αποκατάσταση Heap

					8	9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 6

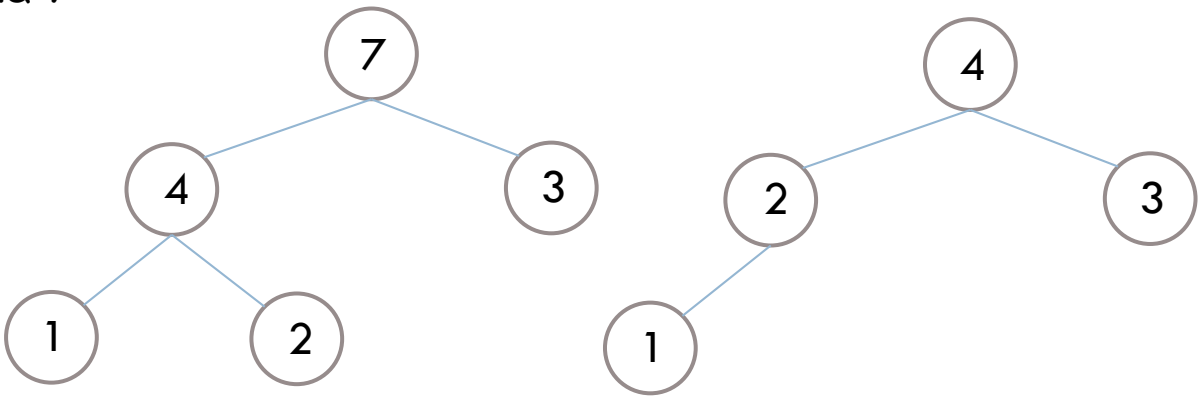


				7	8	9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 7

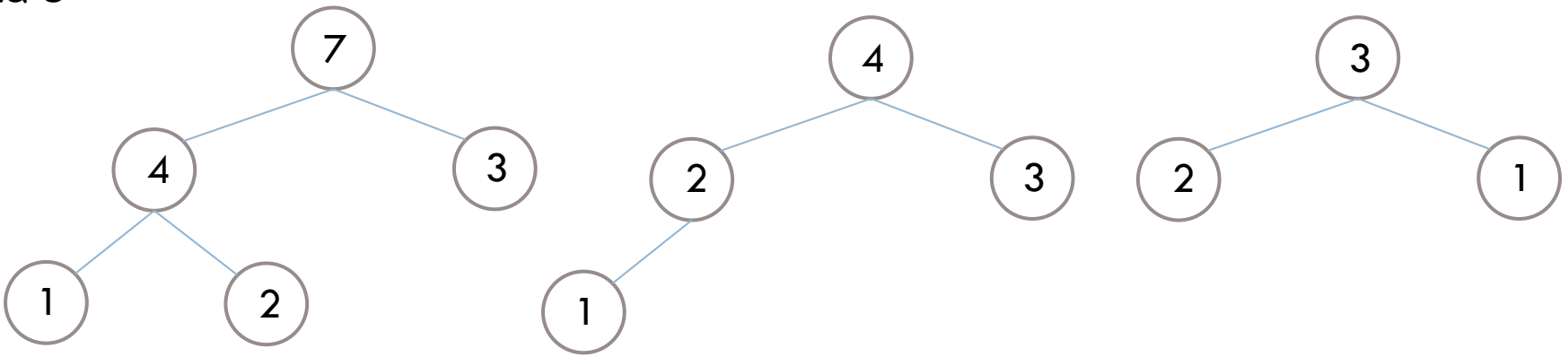


			4	7	8	9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 8

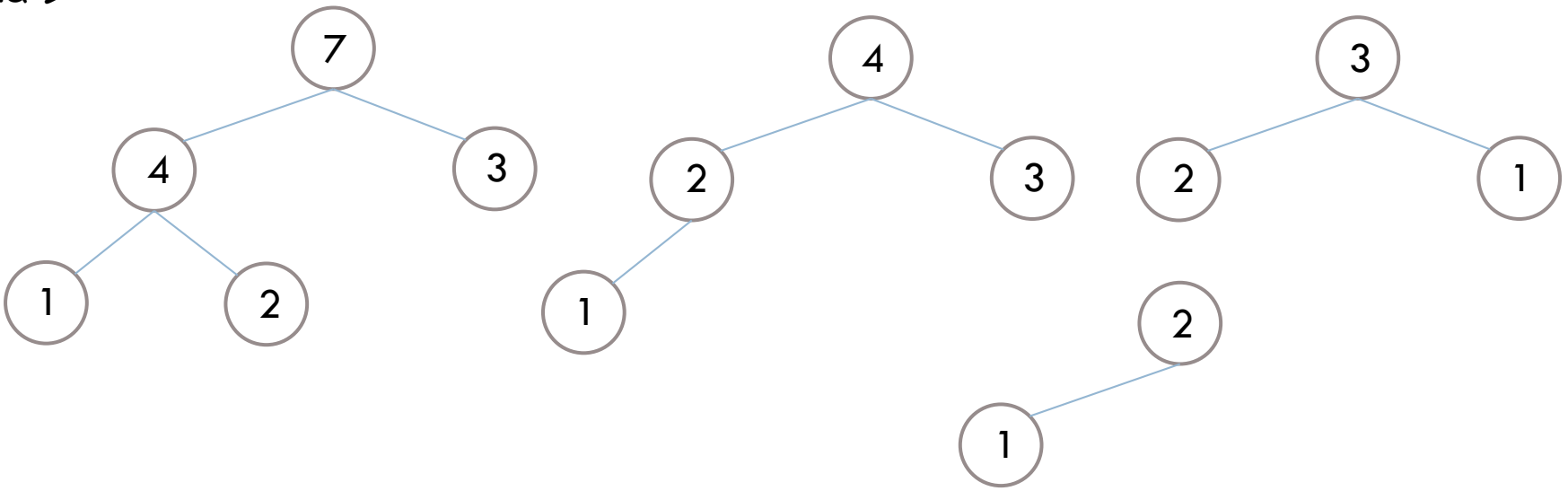


		3	4	7	8	9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 9

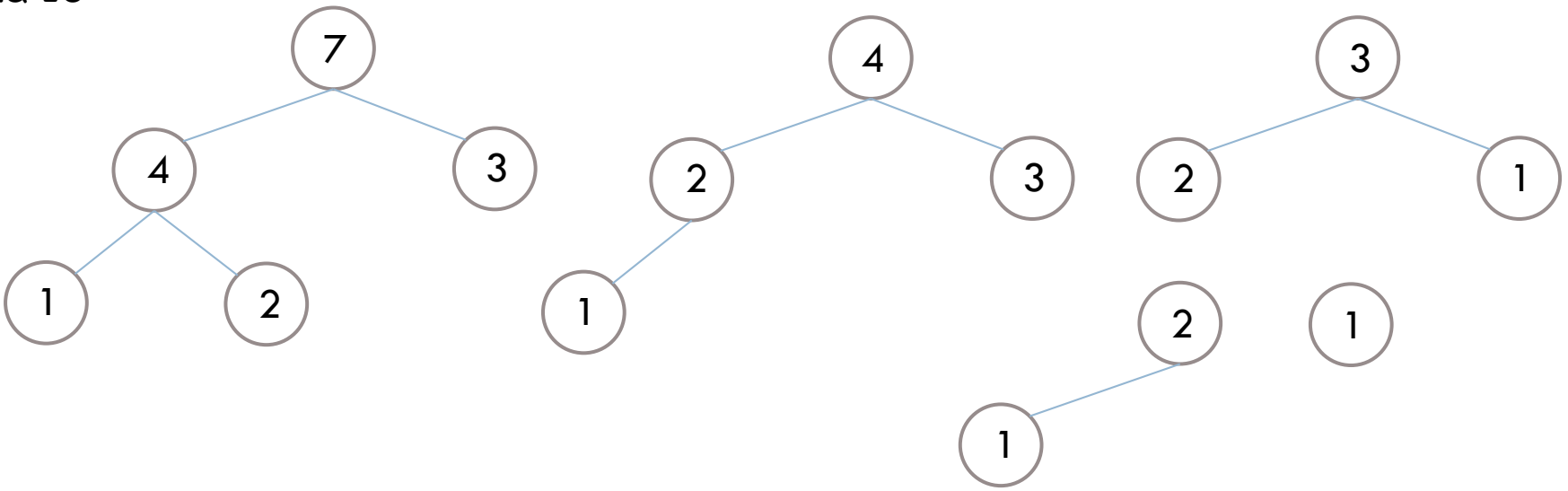


	2	3	4	7	8	9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Βήμα 10



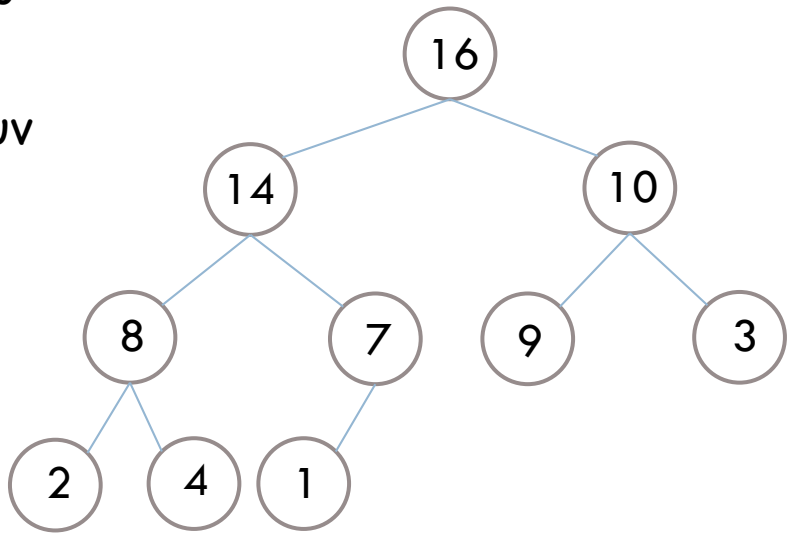
1	2	3	4	7	8	9	10	14	16

Αλγόριθμος Heap-Sort

Παράδειγμα

Έξοδος Αλγόριθμου

Ταξινομημένη ακολουθία στοιχείων



1	2	3	4	7	8	9	10	14	16

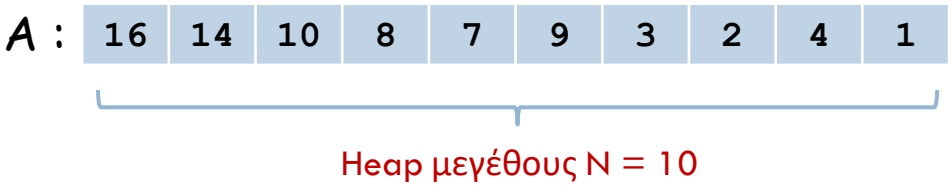
Αλγόριθμος Heap-Sort

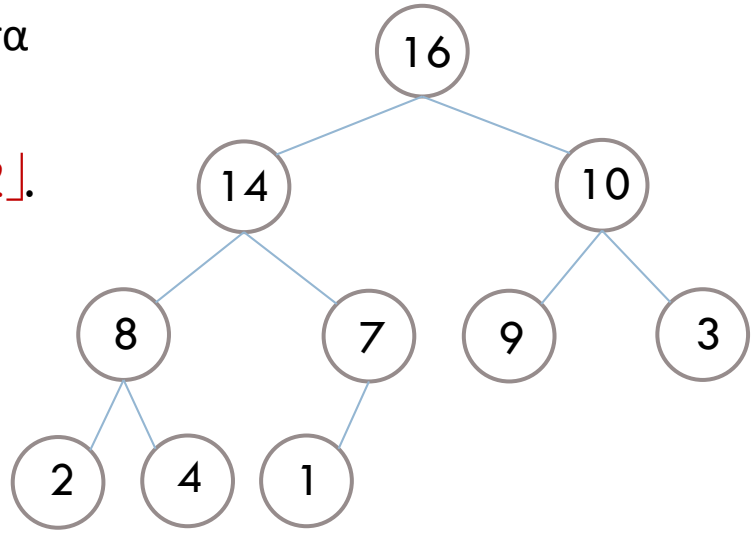
Διατήρηση σωρού σε πίνακα !!!

- Εκτός από την χρήση δεικτών (δομή δένδρο) για την αναπαράσταση ενός σωρού N στοιχείων, μπορούμε να χρησιμοποιήσουμε έναν πίνακα *A* μήκους N.
- Οι κόμβοι του σωρού αντιστοιχούν στις θέσεις του πίνακα H:

  A(1) είναι η ρίζα, και για κάθε κόμβο στην θέση i τα παιδιά του βρίσκονται στις θέσεις 2i και 2i+1.

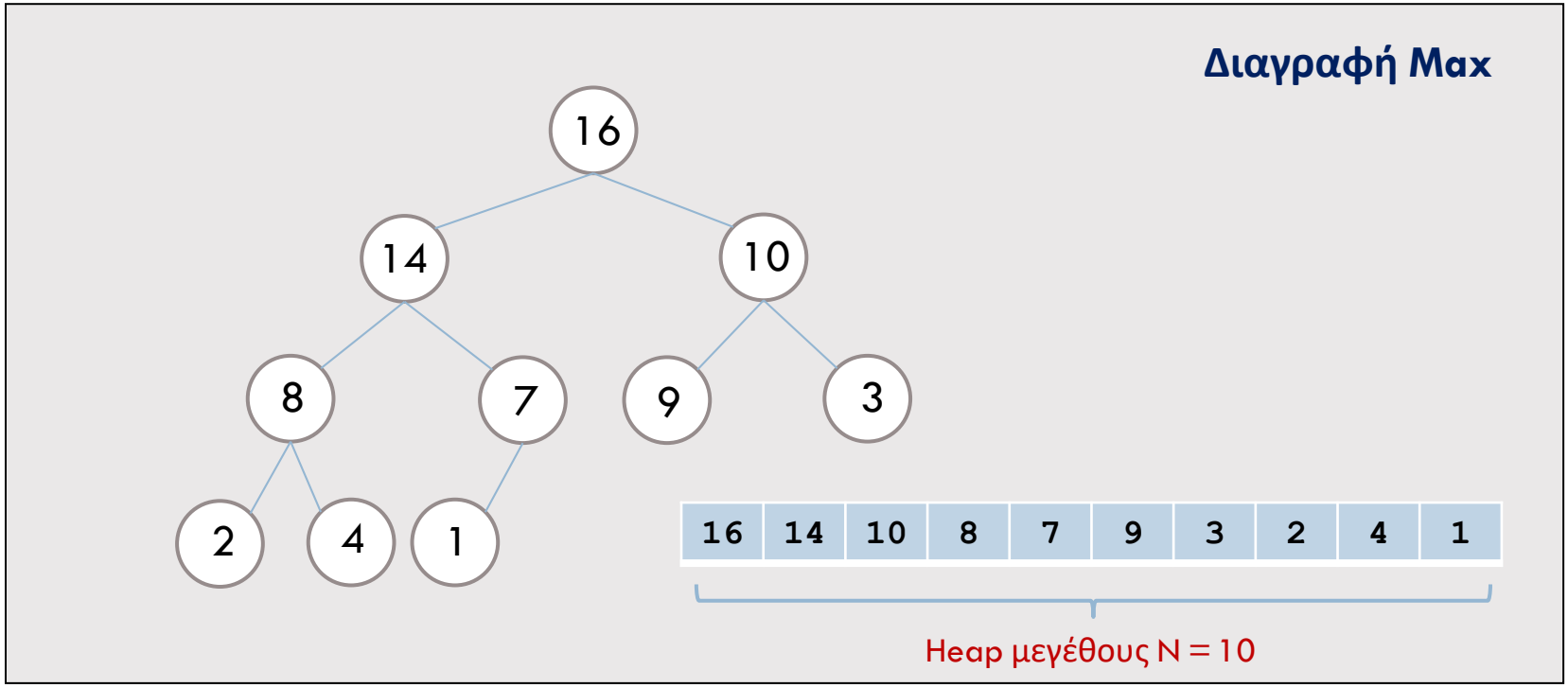
  Ο πατέρας του κόμβου i βρίσκεται στη θέση $\lfloor i/2 \rfloor$.





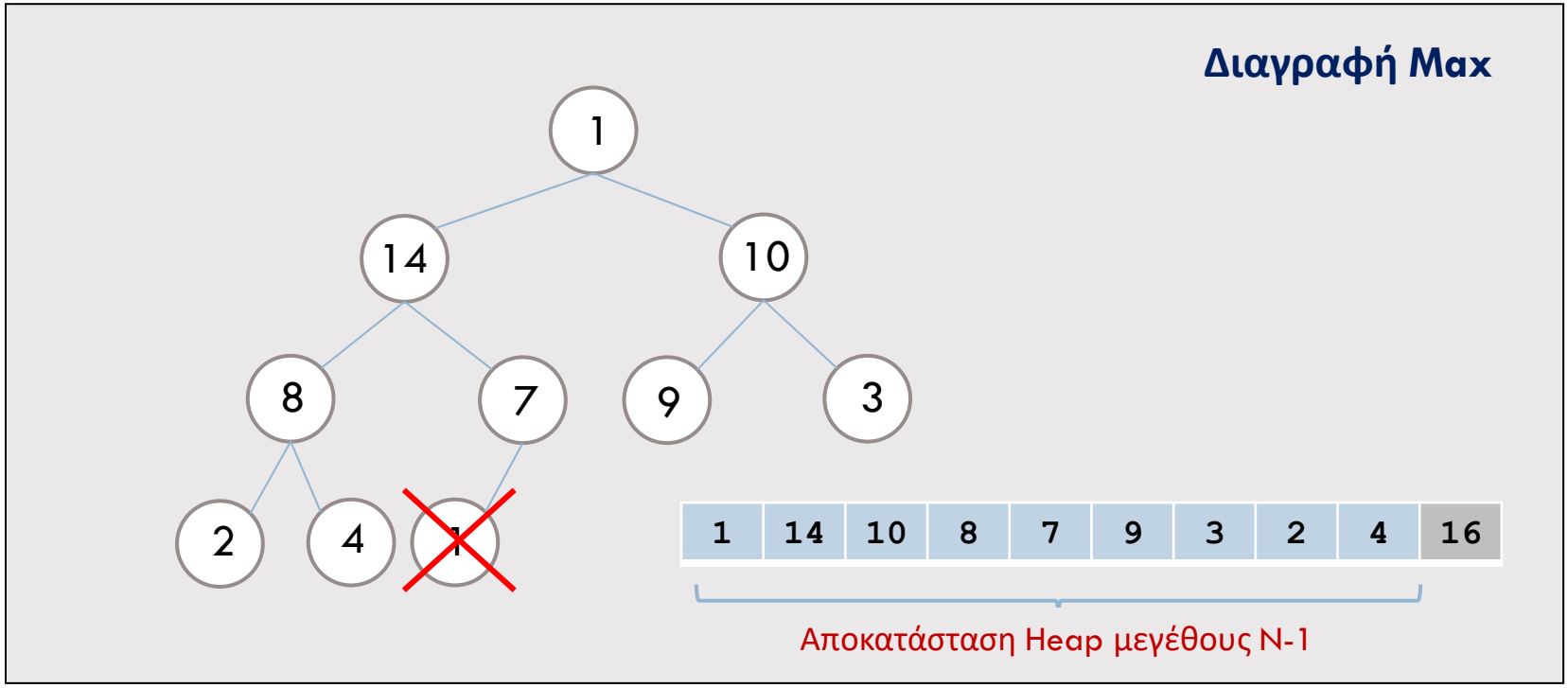
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



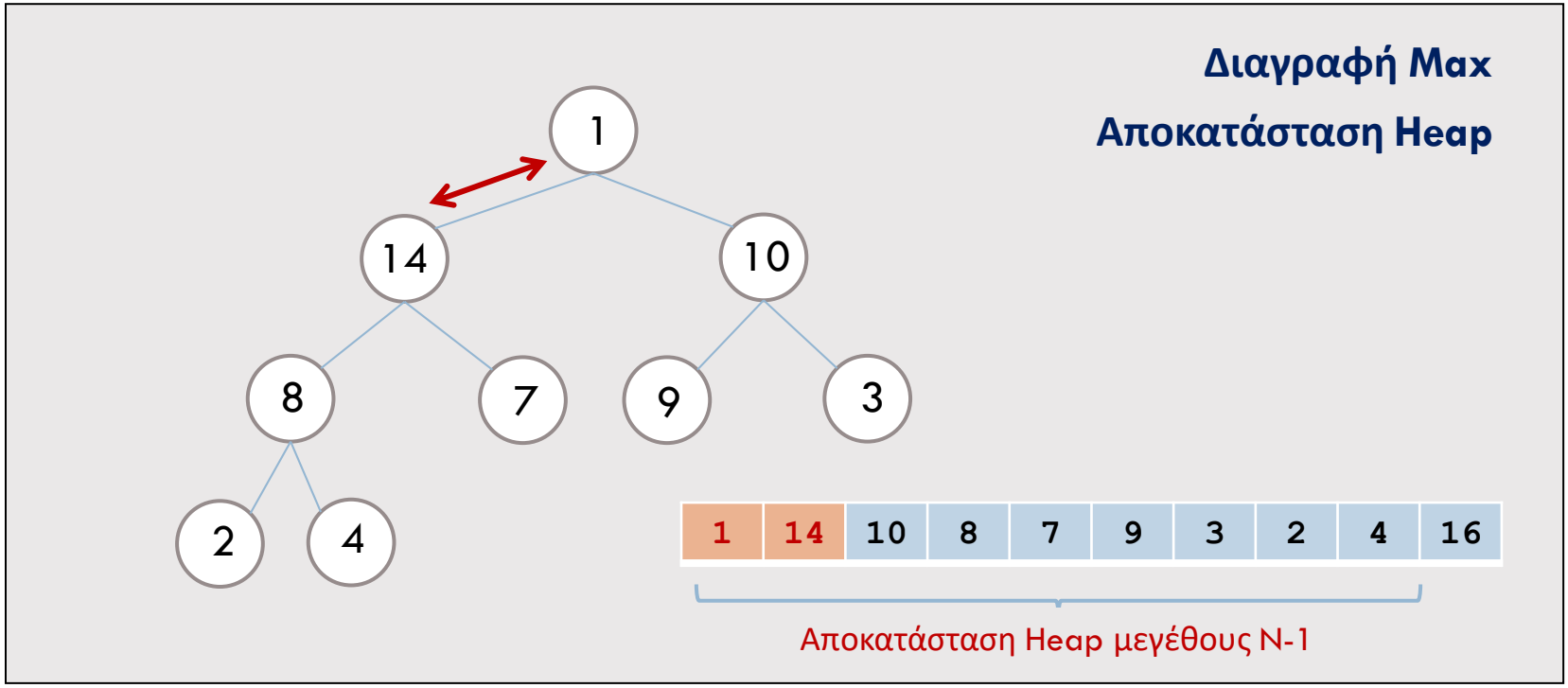
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



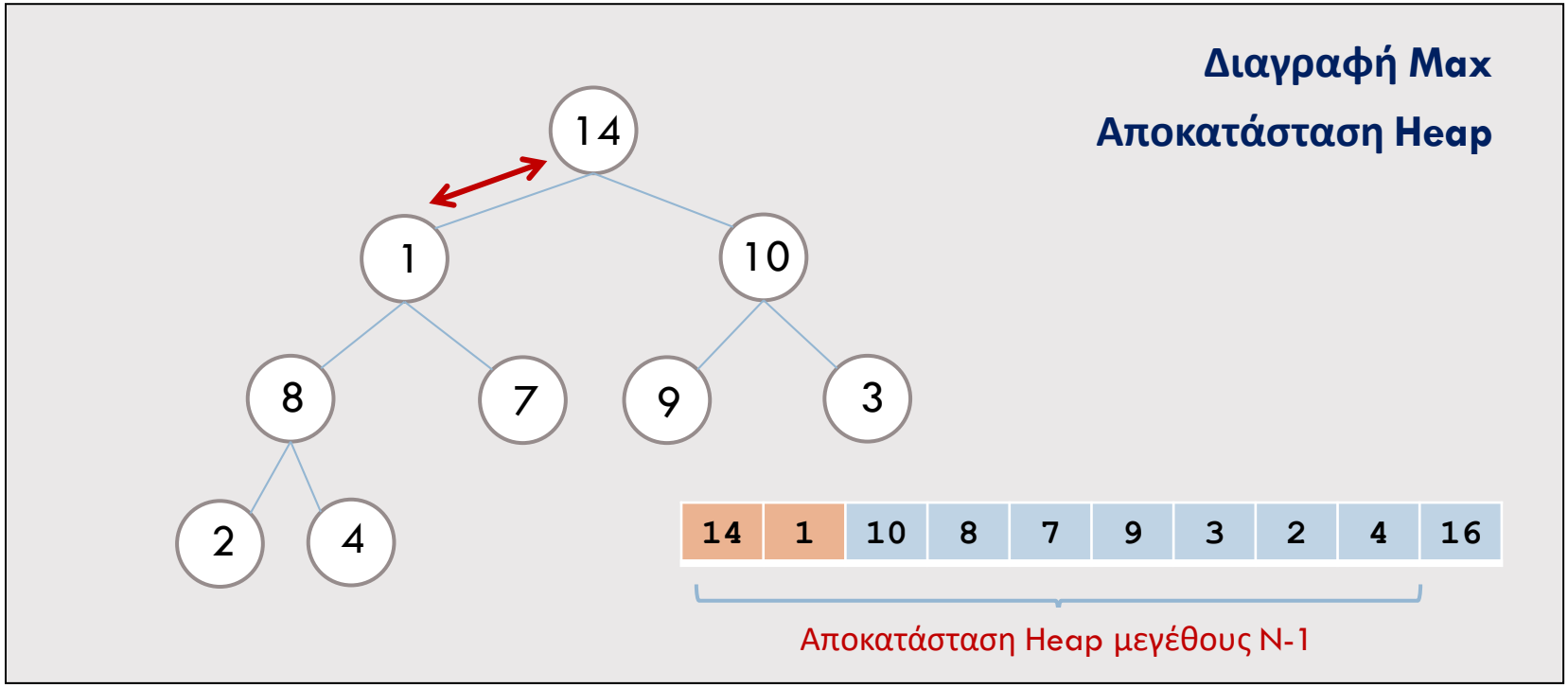
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



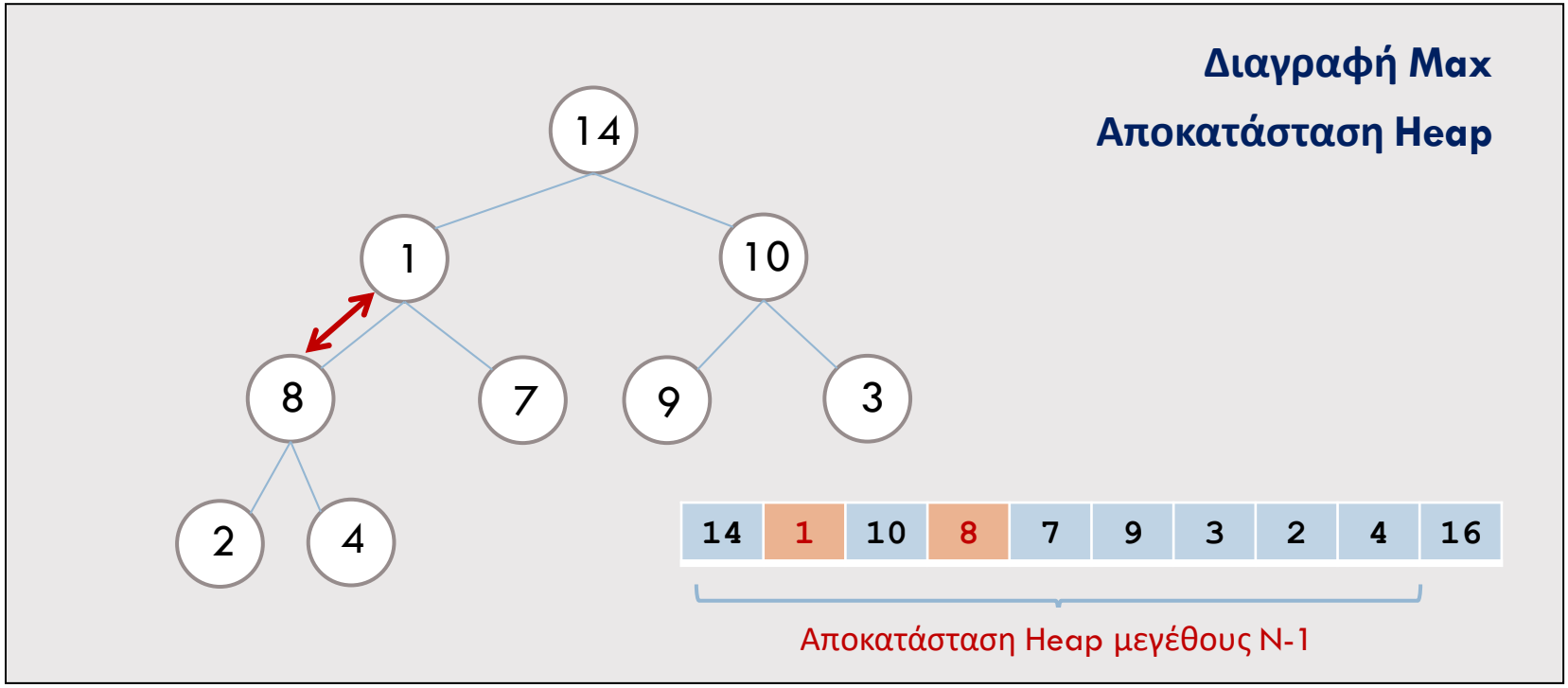
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



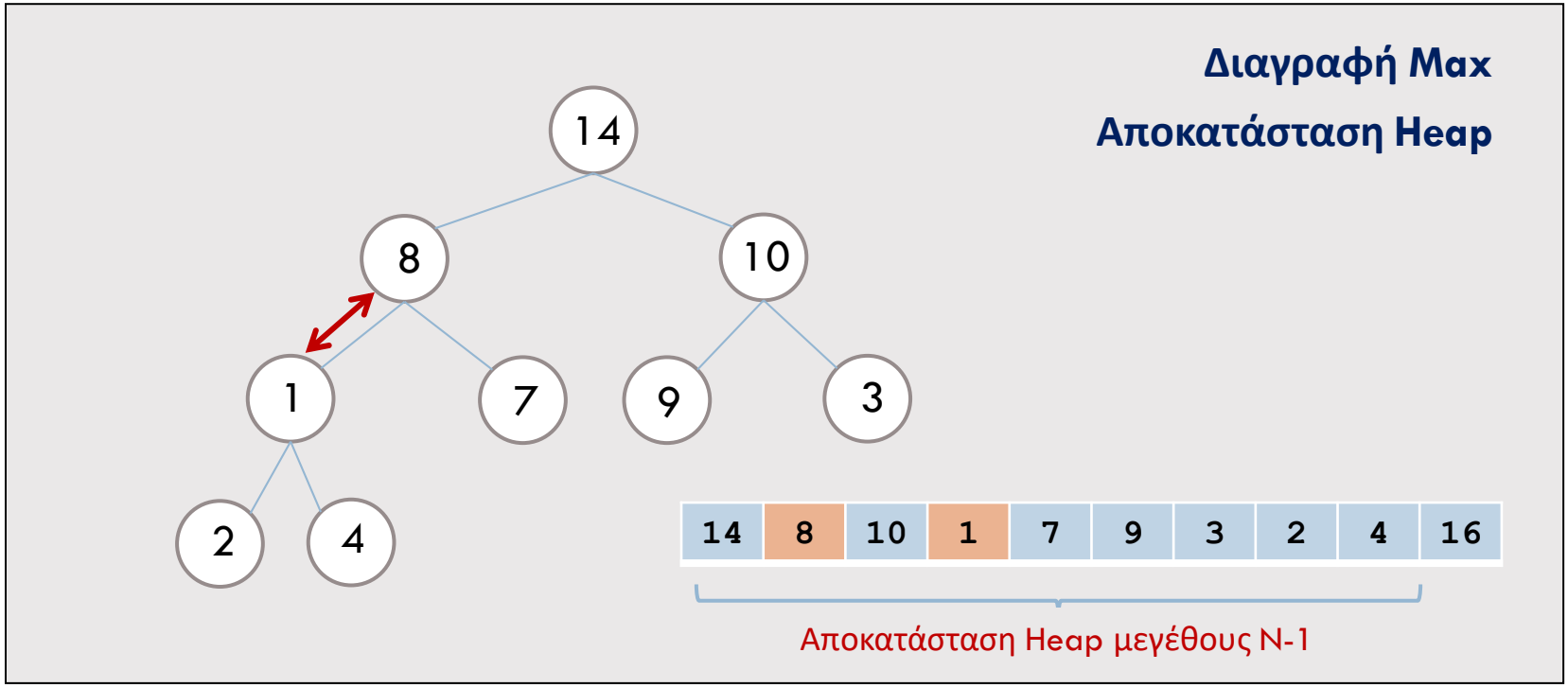
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



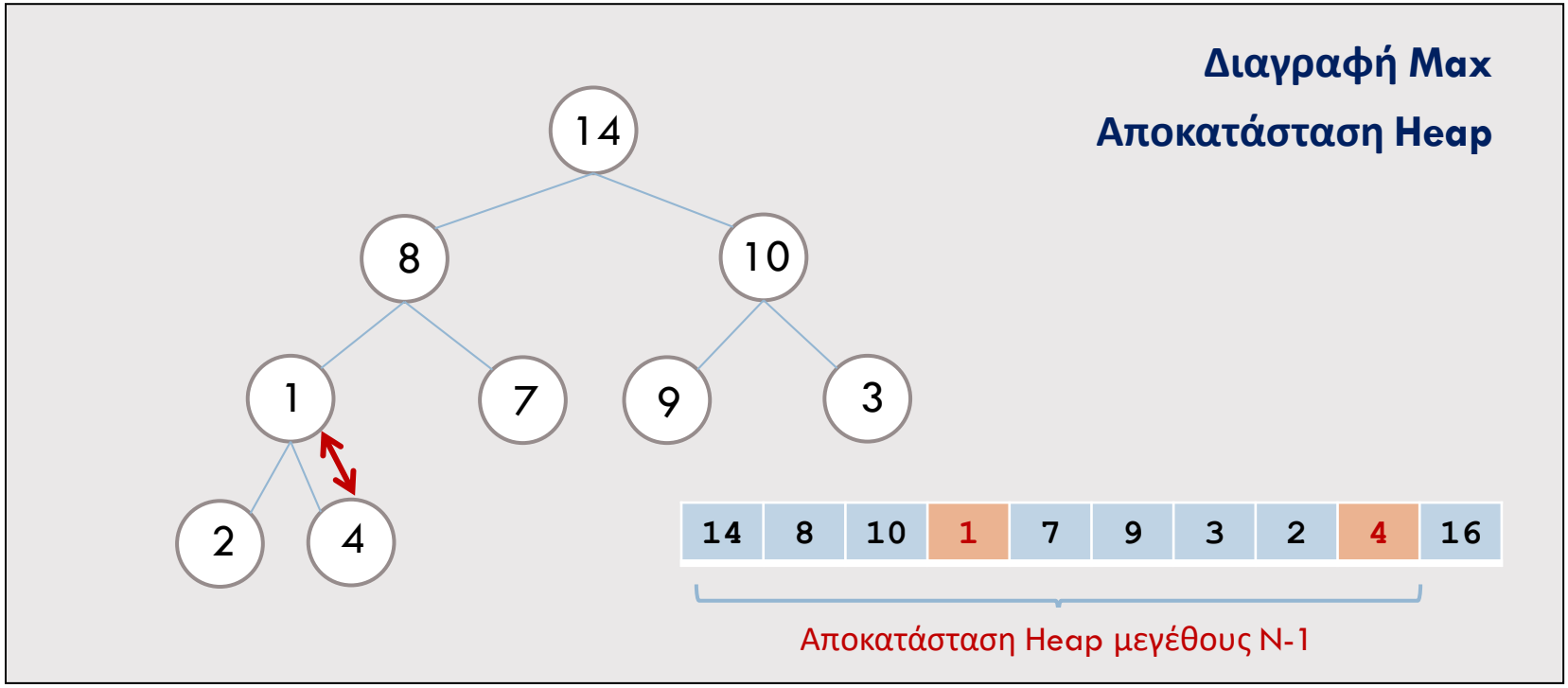
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



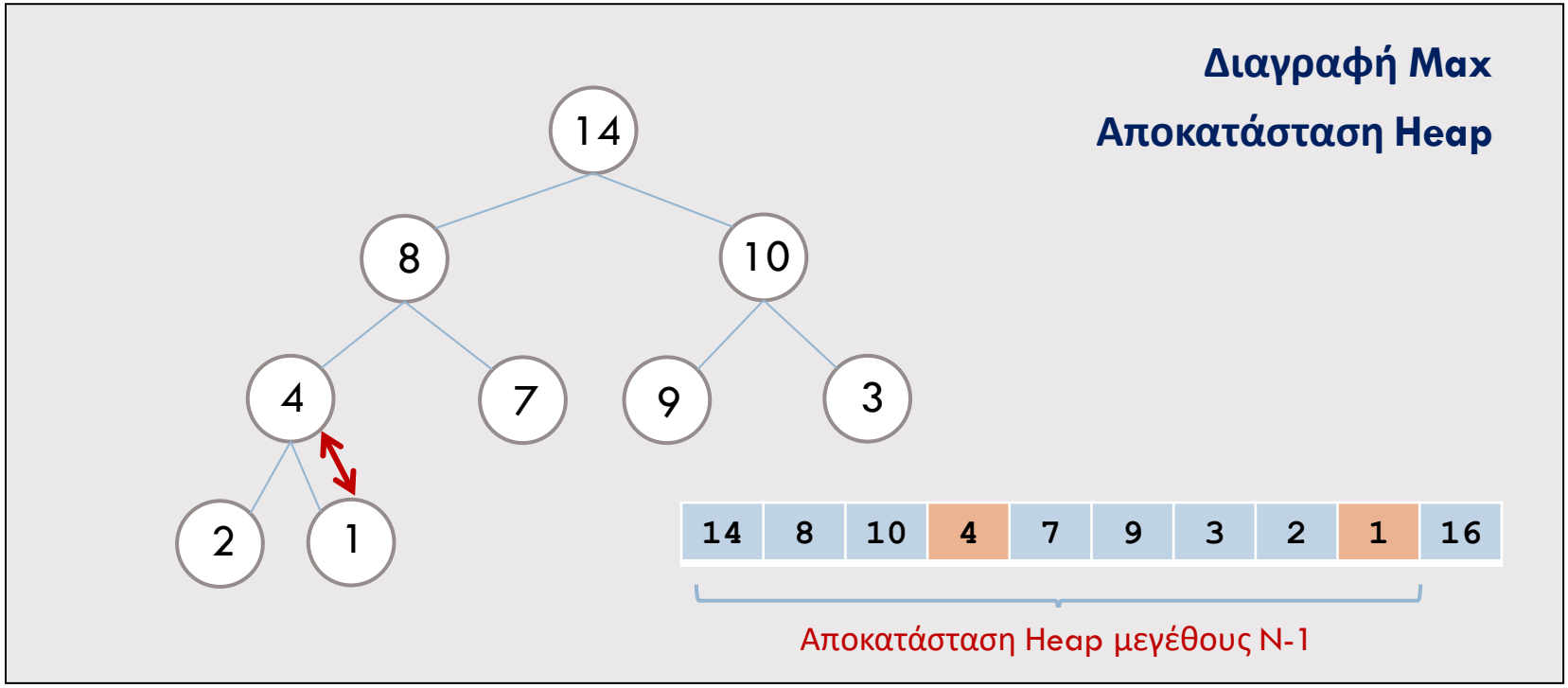
Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



Αλγόριθμος Heap-Sort

Διατήρηση σωρού σε πίνακα !!!



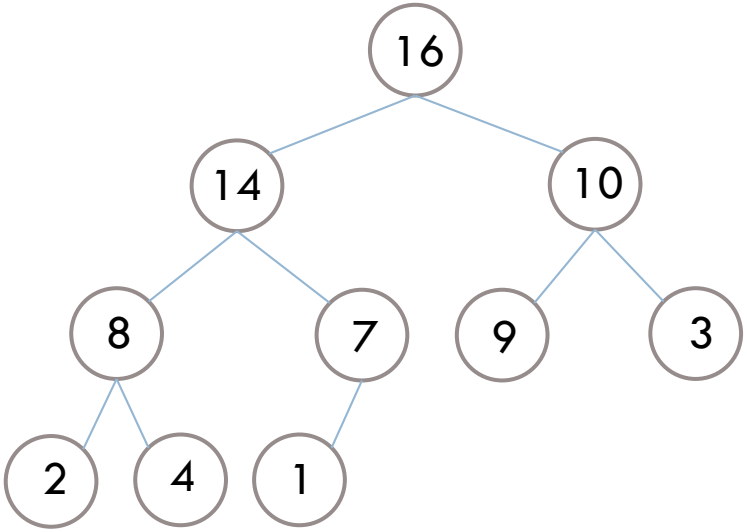
Αλγόριθμος Heap-Sort

```
Algorithm Heap-sort
begin
     Build-heap(A,n)
     heap-size(A) ← n
     for i ← heap-size(A) downto 2 do
         Swap(A[1], A[i])
         heap-size(A)--
         Build-heap(A, heap-size(A))
     end
     return A
end
```

A :	16	14	10	8	7	9	3	2	4	1

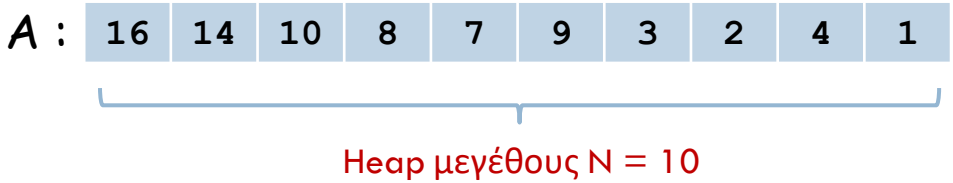
Heap μεγέθους N = 10

- Τα προς ταξινόμηση στοιχεία είναι καταχωρημένα στον πίνακα A μήκος N.
- Εάν κάποια στιγμή ο σωρός έχει n < N στοιχεία χρησιμοποιούμε τις n πρώτες θέσεις του πίνακα A.



Αλγόριθμος Heap-Sort

```
Algorithm Heap-sort
begin
     Build-heap(A,n)
     heap-size(A) ← n
     for i ← heap-size(A) downto 2 do
         Swap(A[1], A[i])
         heap-size(A)--
         Build-heap(A, heap-size(A))
     end
     return A
end
```

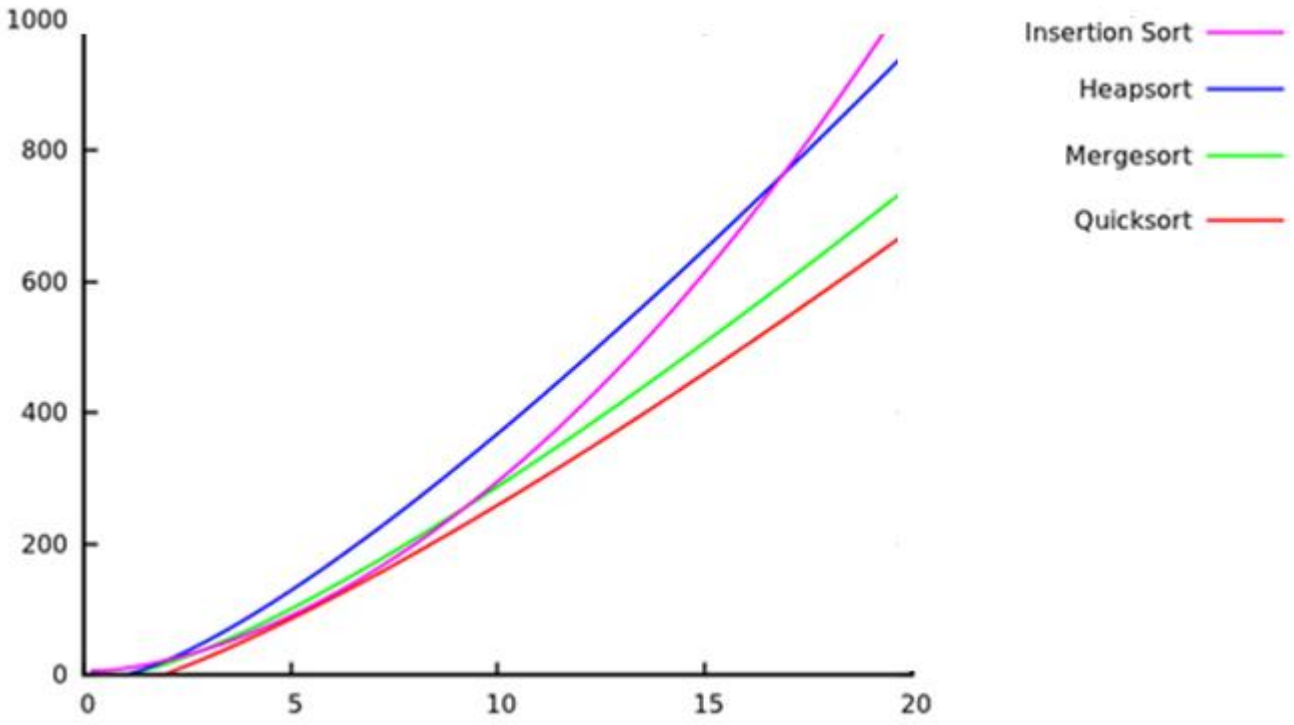


- Τα προς ταξινόμηση στοιχεία είναι καταχωρημένα στον πίνακα A μήκος N.
- Εάν κάποια στιγμή ο σωρός έχει n < N στοιχεία χρησιμοποιούμε τις n πρώτες θέσεις του πίνακα A.

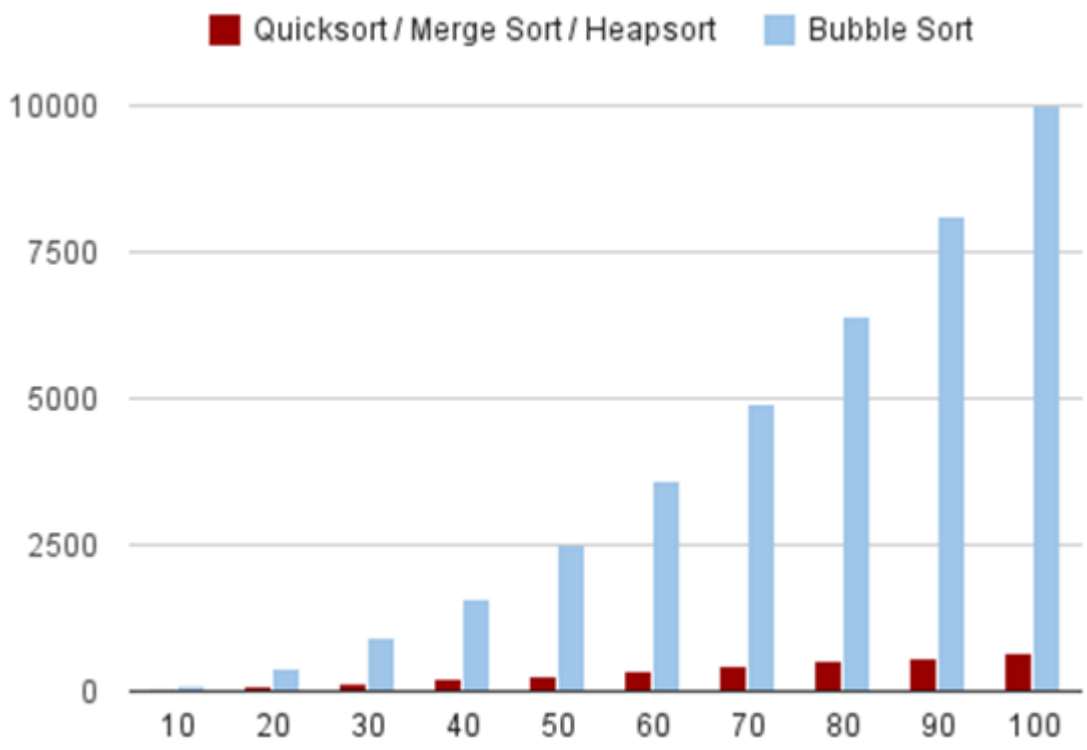
Ορθότητα ✓

Πολυπλοκότητα $O(n \log n)$

Σύγκριση Αλγορίθμων Ταξινόμησης



Σύγκριση Αλγορίθμων Ταξινόμησης



Σύγκριση Αλγορίθμων Ταξινόμησης

Name	Best	Average	Worst	Stable
Bubble sort	n	n^2	n^2	Yes
Selection sort	n^2	n^2	n^2	No
Insertion sort	n	n^2	n^2	Yes
Merge sort	$n \log n$	$n \log n$	$n \log n$	Yes
Quicksort	$n \log n$	$n \log n$	n^2	typical in-place sort is not stable;
Heapsort	$n \log n$	$n \log n$	$n \log n$	No

Κάτω Φράγμα Ταξινόμησης

Υπολόγισε ένα ελάχιστο πλήθος Βασικών Πράξεων (συγκρίσεων) που απαιτούνται για την επίλυση του Προβλήματος SORTING

- Είσοδος $(x_1, x_2, \ldots, x_n)$
- Αρχικά $n!$ περιπτώσεις:

$x_1 < x_2 < x_3 < \ldots < x_n$
$x_2 < x_1 < x_3 < \ldots < x_n$
$x_3 < x_1 < x_2 < \ldots < x_n$
$\ldots$
$x_n < x_{n-1} < \ldots < x_1$

- Σε κάθε σύγκριση το πλήθος των περιπτώσεων υποδιπλασιάζεται (στην καλύτερη περίπτωση)
- Έχουμε δείξει (Ενότητα 3):

Πλήθος συγκρίσεων ≥ $\log(n!) \geq (n \log n)/4$

Οποιοσδήποτε αλγόριθμος ταξινόμησης n στοιχείων χρειάζεται $\Omega(n \log n)$ συγκρίσεις

Κάτω φράγμα για ταξινόμηση που χρησιμοποιεί μόνο συγκρίσεις

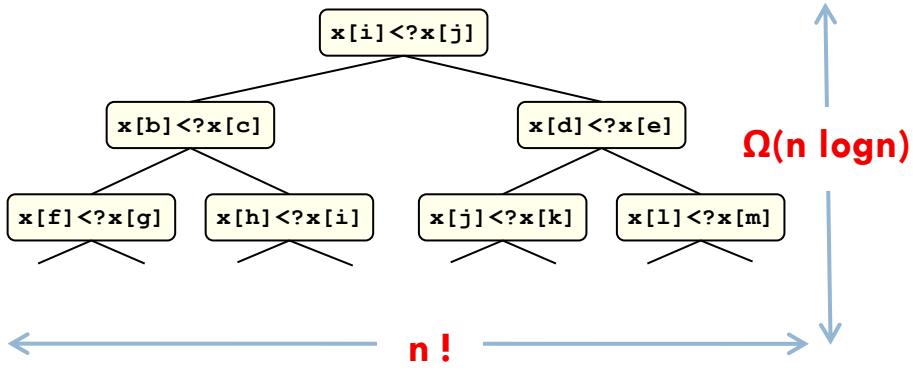
Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης

Οποιοσδήποτε αλγόριθμος ταξινόμησης με συγκρίσεις, ταξινομεί *n* στοιχεία απαιτώντας *τουλάχιστον*

n* log *n

συγκρίσεις.



Ύψος δυαδικού δένδρου με ***n*!** φύλλα

$$\Omega(\log n!) = \Omega(n \log n)$$

Λήμμα 3. Για δεδομένο **n**, το δένδρο απόφασης για οποιοδήποτε αλγόριθμο ταξινόμησης με συγκρίσεις των στοιχείων του ***έχει ύψος τουλάχιστον n* log n**.

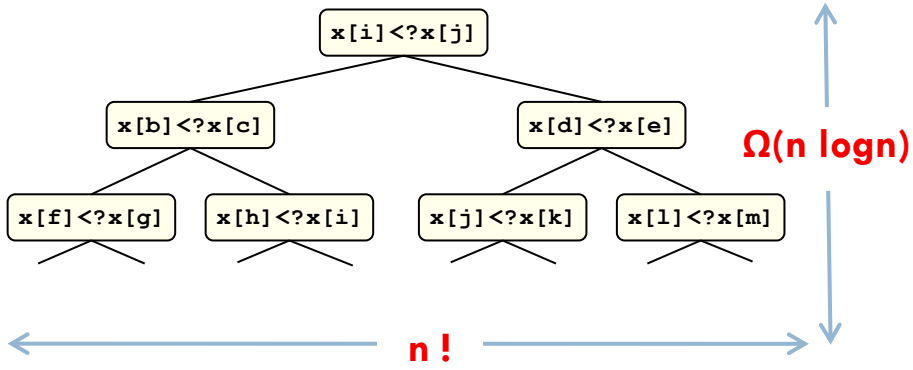
Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης

Οποιοσδήποτε αλγόριθμος ταξινόμησης με συγκρίσεις, ταξινομεί *n* στοιχεία απαιτώντας *τουλάχιστον*

n* log *n

συγκρίσεις.

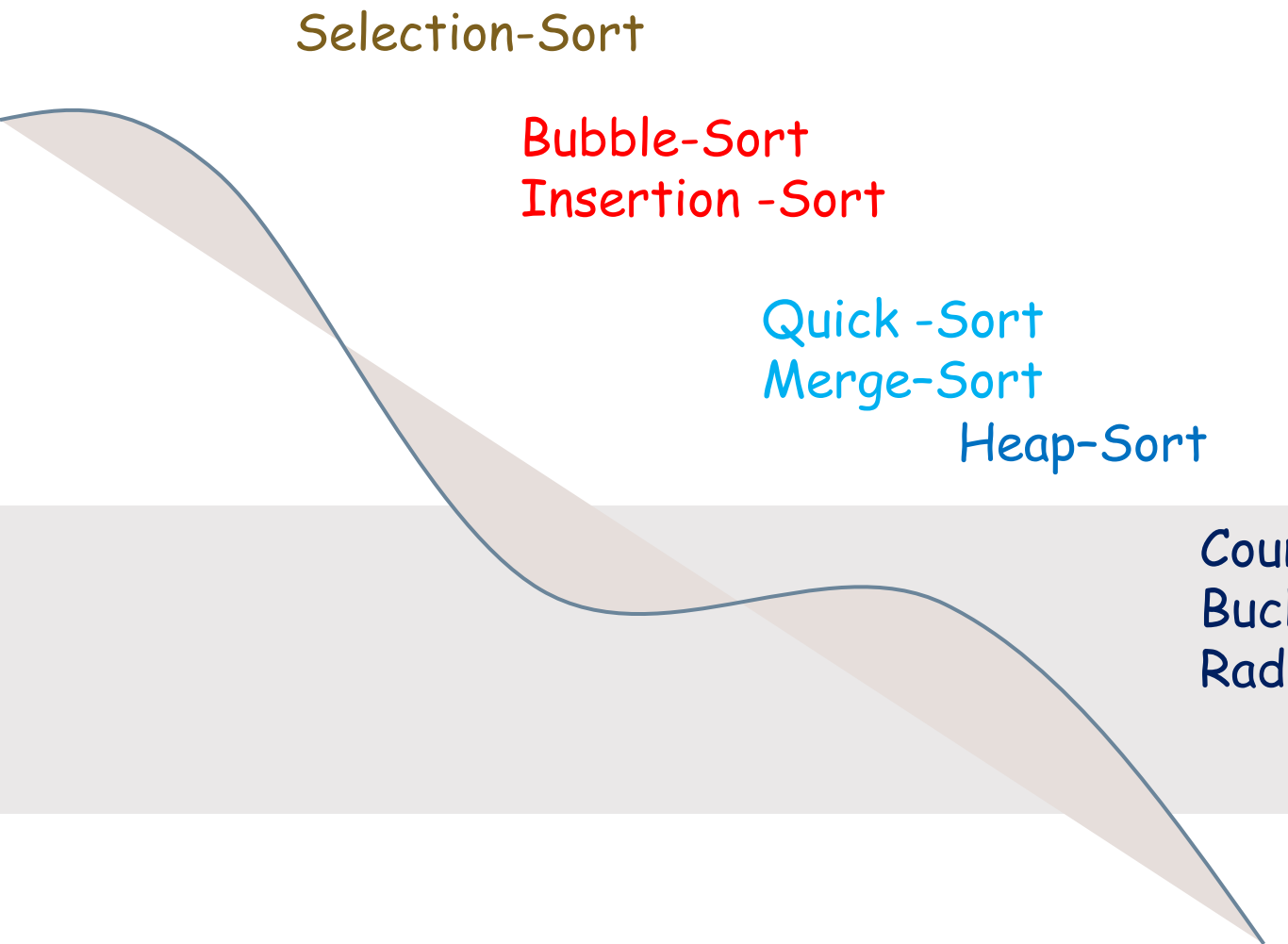


Ύψος δυαδικού δένδρου με ***n*!** φύλλα

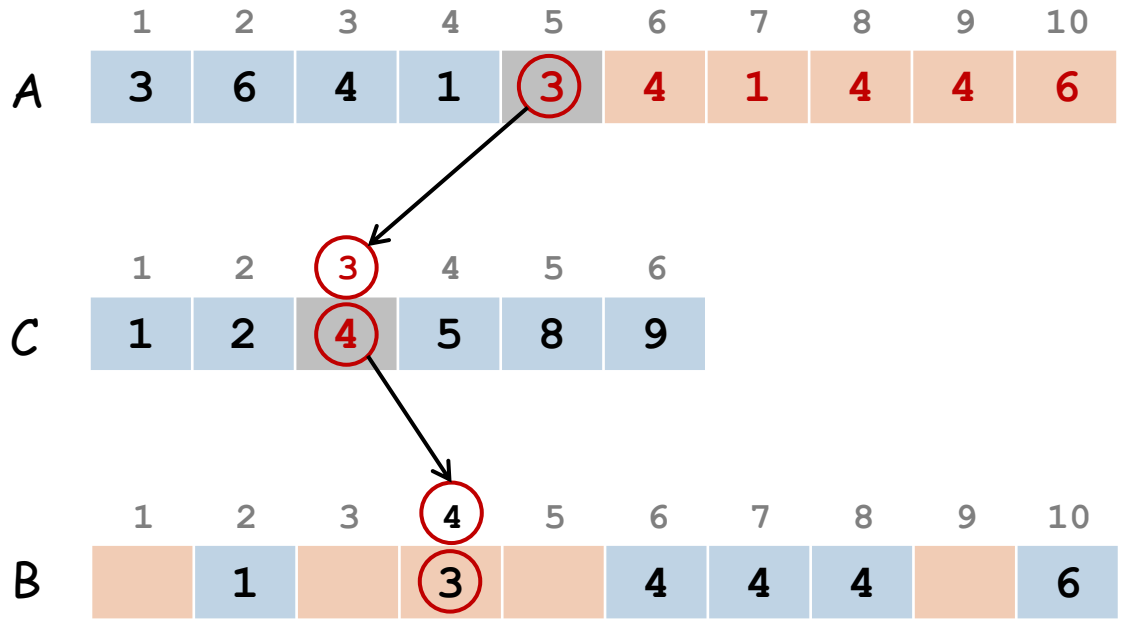
$$\Omega(\log n!) = \Omega(n \log n)$$

Κάτω φράγμα ταξινόμησης: $\Omega(n \log n)$

Αλγόριθμοι Ταξινόμησης



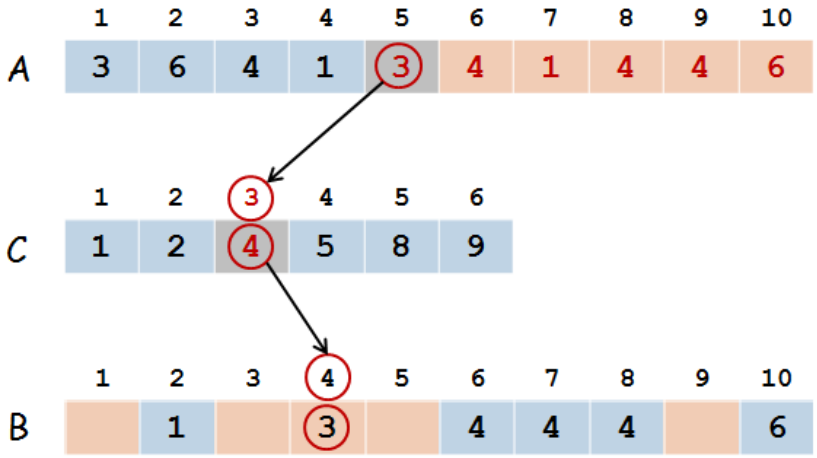
Αλγόριθμος Counting-Sort



Αλγόριθμος Counting-Sort

Ιδέα Αλγόριθμου !!!

- Ο αλγόριθμος ταξινόμησης της απαρίθμησης (Counting–Sort) δέχεται ότι τα n στοιχεία εισόδου (κλειδιά) είναι ακέραιοι στο διάστημα [1, k], όπου k θετικός ακέραιος.
- Όταν k = O(n), ο αλγόριθμος εκτελείται σε χρόνο O(n).



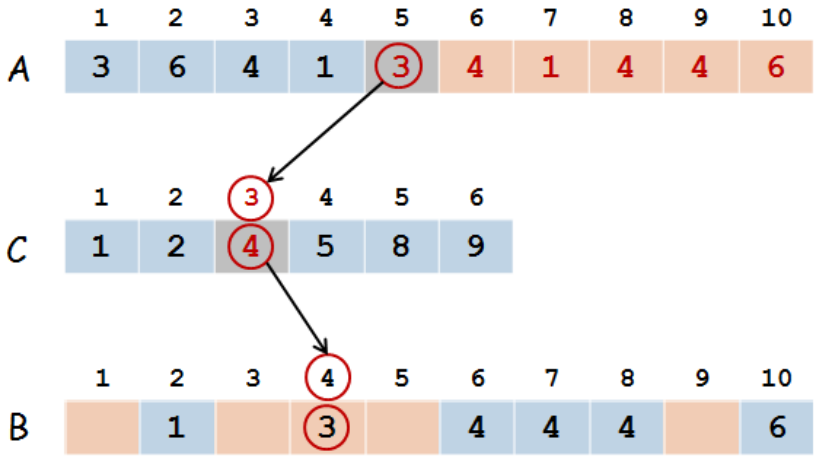
Βασική ιδέα: για κάθε στοιχείο x της εισόδου υπολογίζει το πλήθος των στοιχείων που είναι μικρότερα του x.

Για παράδειγμα, εάν υπάρχουν 3 στοιχεία μικρότερα του x, τότε το x θα καταλάβει τη θέση 4 στην ταξινομημένη ακολουθία.

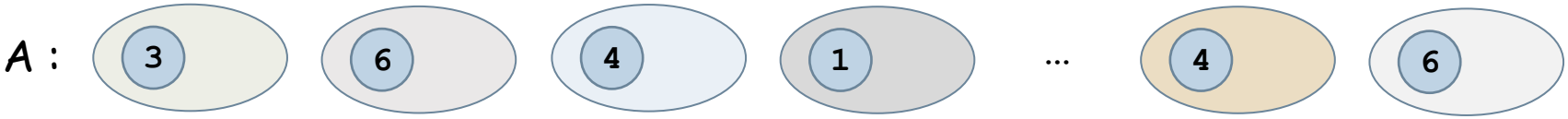
Αλγόριθμος Counting-Sort

Ιδέα Αλγόριθμου !!!

- Ο αλγόριθμος ταξινόμησης της απαρίθμησης (Counting–Sort) δέχεται ότι τα n στοιχεία εισόδου (κλειδιά) είναι ακέραιοι στο διάστημα [1, k], όπου k θετικός ακέραιος.
- Όταν k = O(n), ο αλγόριθμος εκτελείται σε χρόνο O(n).



Προσοχή !!!: Οι ακέραιοι προς ταξινόμηση είναι τα κλειδιά (keys).



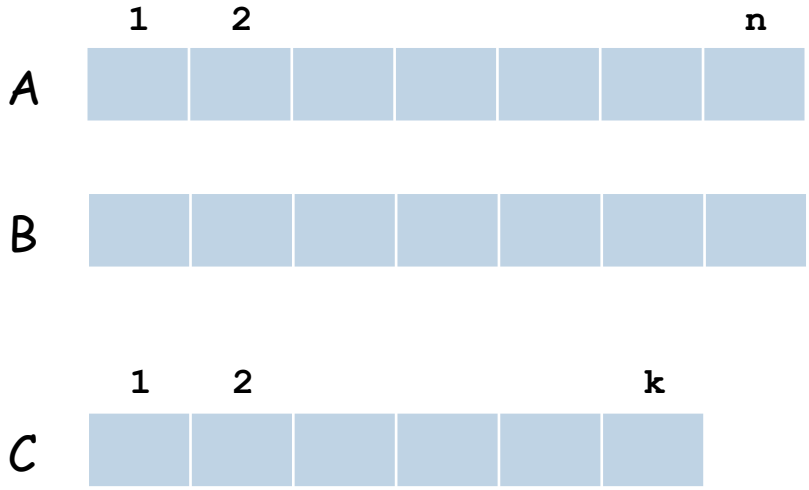
Αλγόριθμος Counting-Sort

Ιδέα Αλγόριθμου !!!

- Τα n στοιχεία εισόδου είναι καταχωρημένα στον πίνακα A[1..n].
- Ο αλγόριθμος απαιτεί δύο ακόμα βοηθητικούς πίνακες:

  B[1..n] για την καταχώρηση της ταξινομημένης ακολουθίας των στοιχείων.

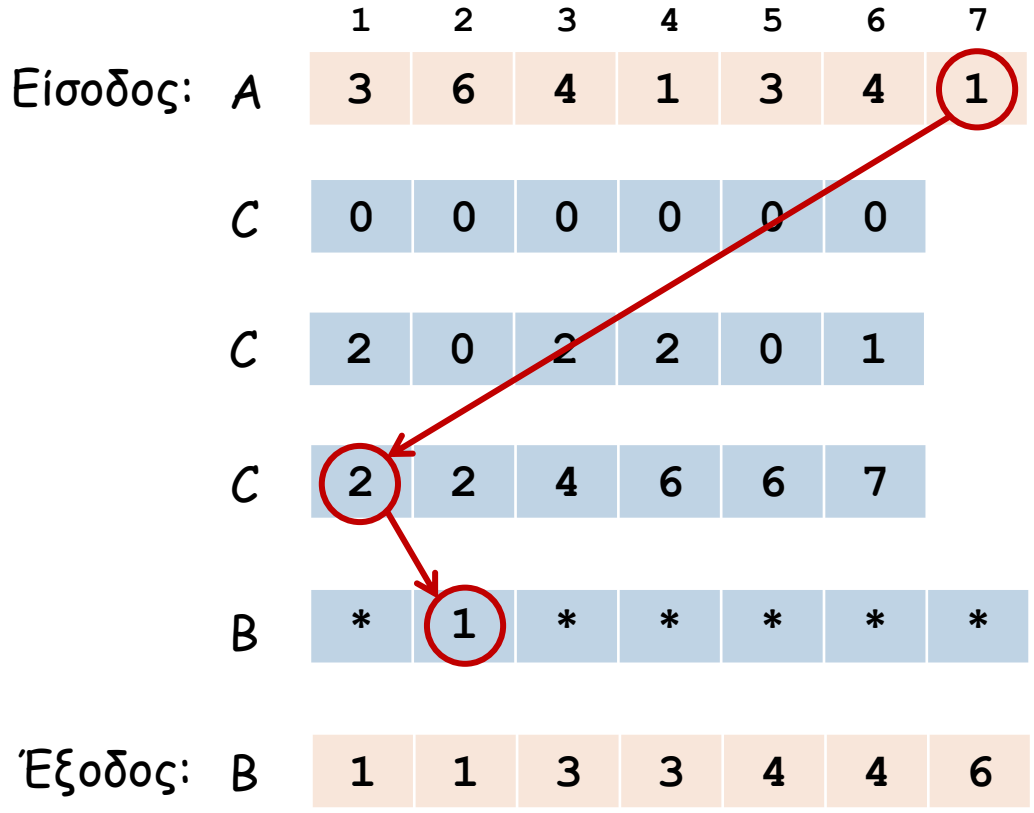
  C[1..k] για την καταχώρηση βοηθητικών πληροφοριών.



Αλγόριθμος Counting-Sort

Αλγόριθμος

```
Algorithm Counting-sort
begin
(1)   for i ← 1 to k do
        C[i] ← 0
(2)   for i ← 1 to n do
        C[A[i]]++
(3)   for i ← 2 to k do
        C[i] ← C[i] + C[i-1]
(4)   for i ← n downto 1 do
        B[C[A[i]]] ← A[i]
        C[A[i]]--
      end
      return B
end
```



Αλγόριθμος Counting-Sort

Παράδειγμα

	1	2	3	4	5	6	7	8	9	10
A	3	6	4	1	3	4	1	4	4	6

	1	2	3	4	5	6
C	2	0	2	4	0	2

Μέτρηση

	1	2	3	4	5	6
C	2	2	4	8	8	10

Προθεματικά Αθροίσματα

Αλγόριθμος Counting-Sort

Παράδειγμα

	1	2	3	4	5	6	7	8	9	10
A	3	6	4	1	3	4	1	4	4	6

Αρχικοποίηση

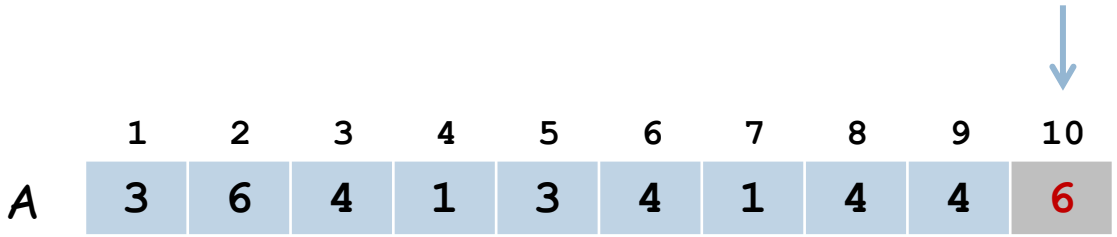
	1	2	3	4	5	6
C	2	2	4	8	8	10

	1	2	3	4	5	6	7	8	9	10
B										

Αλγόριθμος Counting-Sort

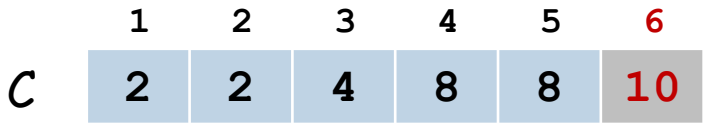
Παράδειγμα

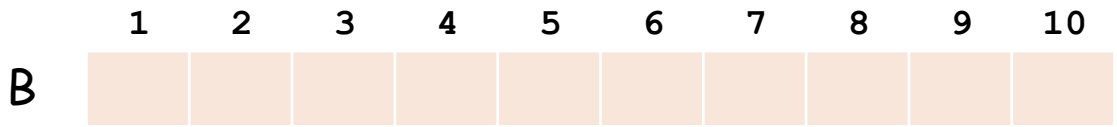
Βήμα 1



$A(10) = 6$

$C(6) = 10$

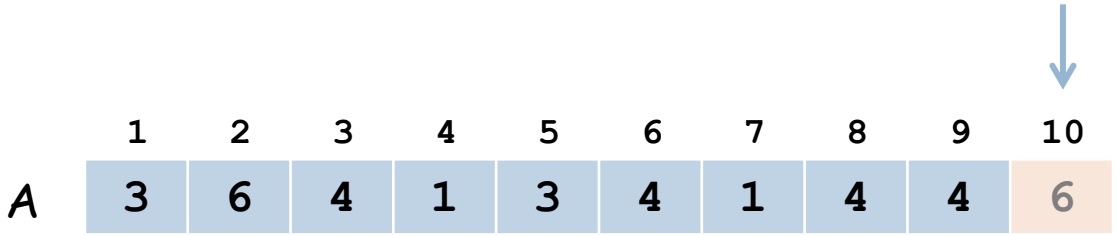




Αλγόριθμος Counting-Sort

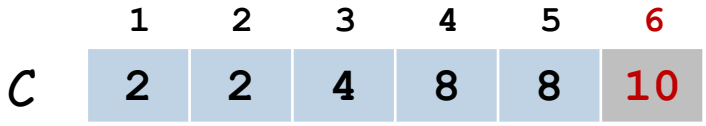
Παράδειγμα

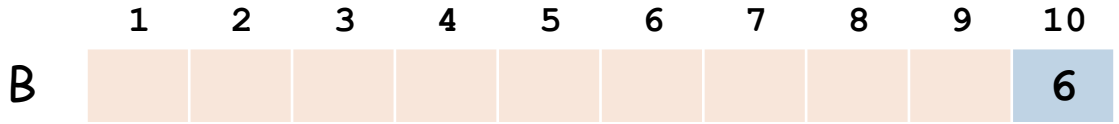
Βήμα 1



$A(10) = 6$

$C(6) = 10$



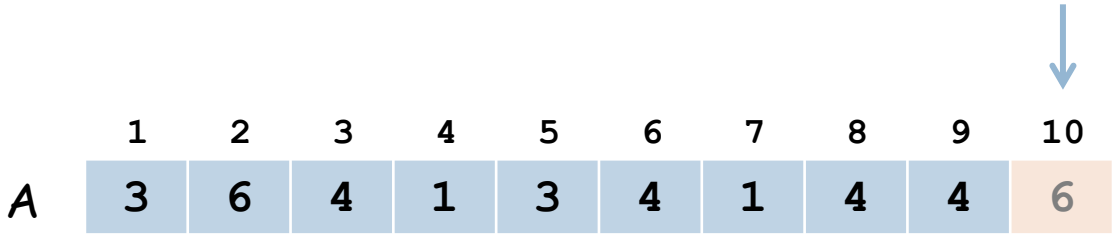


$B(10) = 6$

Αλγόριθμος Counting-Sort

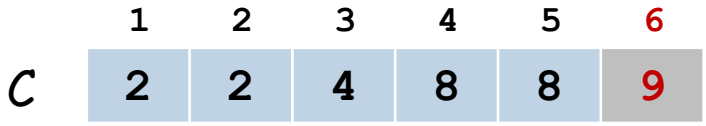
Παράδειγμα

Βήμα 1

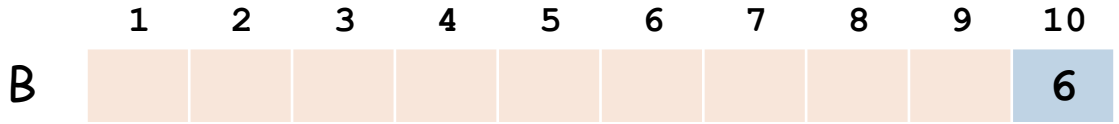


A(10) = 6

C(6) = 10



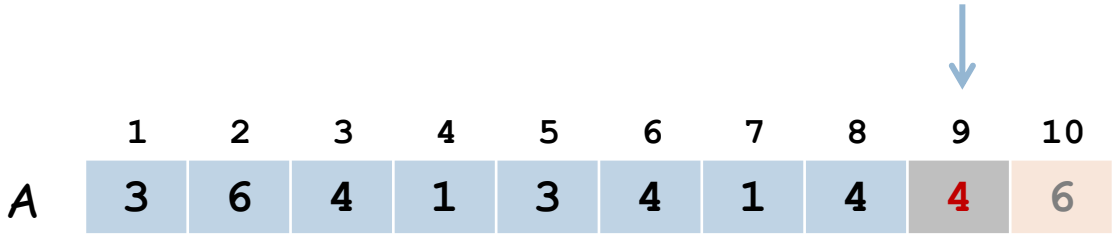
C(6) --

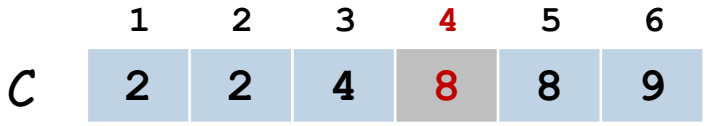


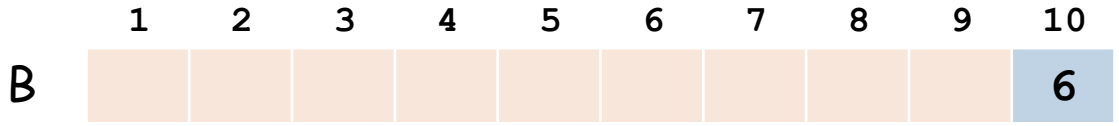
B(10) = 6

Αλγόριθμος Counting-Sort

Βήμα 2







Παράδειγμα

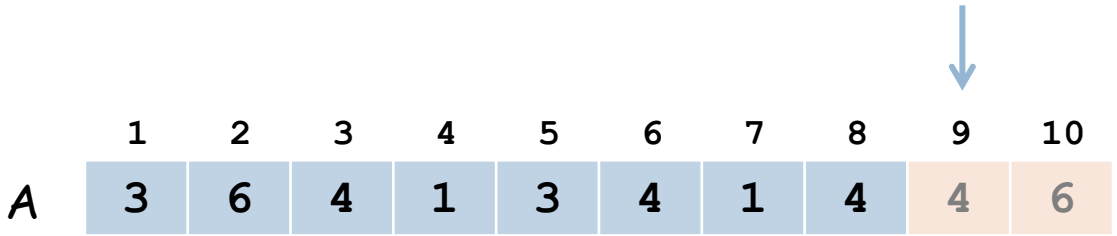
$A(9) = 4$

$C(4) = 8$

Αλγόριθμος Counting-Sort

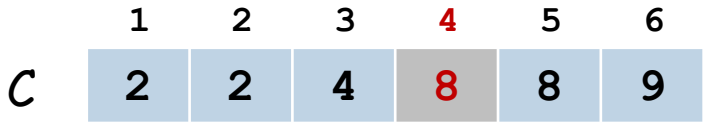
Παράδειγμα

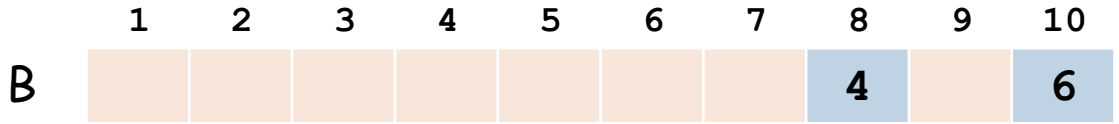
Βήμα 2



$A(9) = 4$

$C(4) = 8$



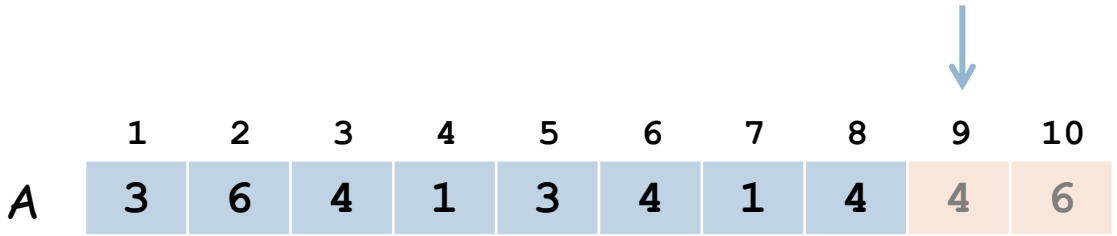


$B(8) = 4$

Αλγόριθμος Counting-Sort

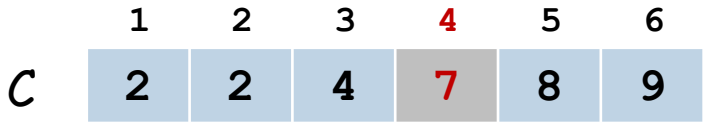
Παράδειγμα

Βήμα 2

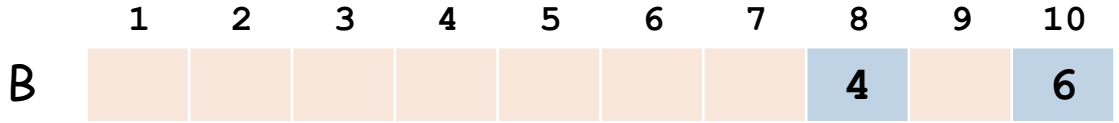


$A(9) = 4$

$C(4) = 8$



$C(4)$ --

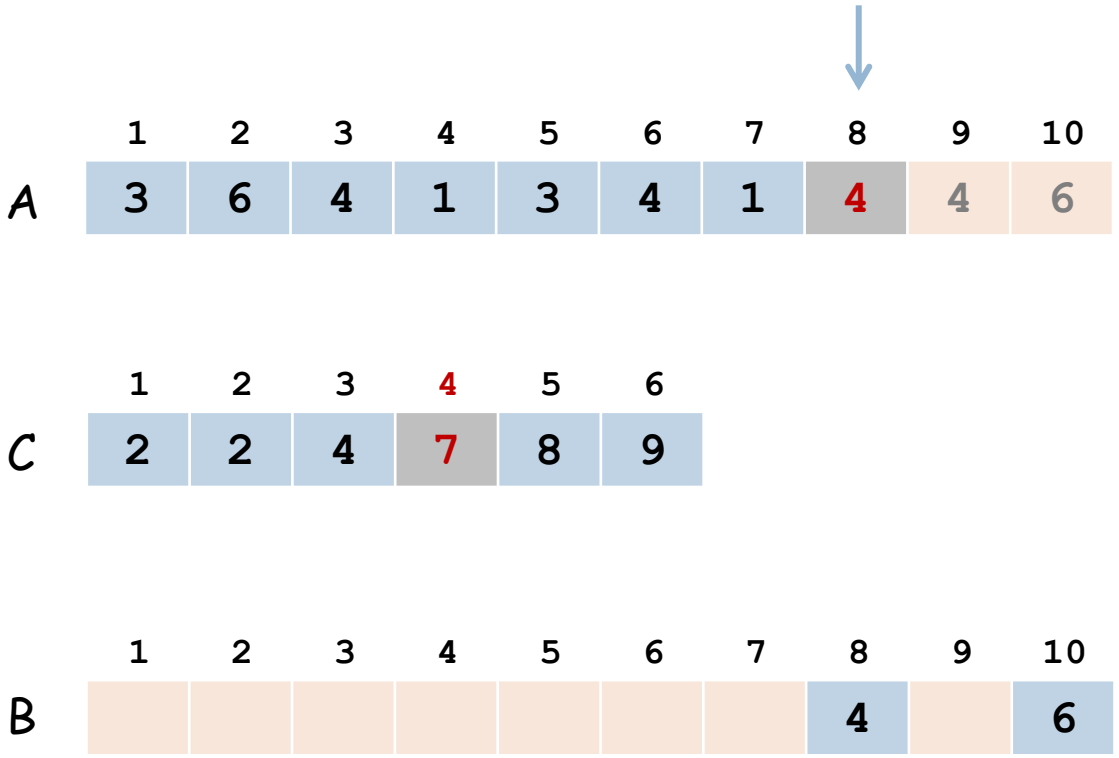


$B(8) = 4$

Αλγόριθμος Counting-Sort

Παράδειγμα

Βήμα 3



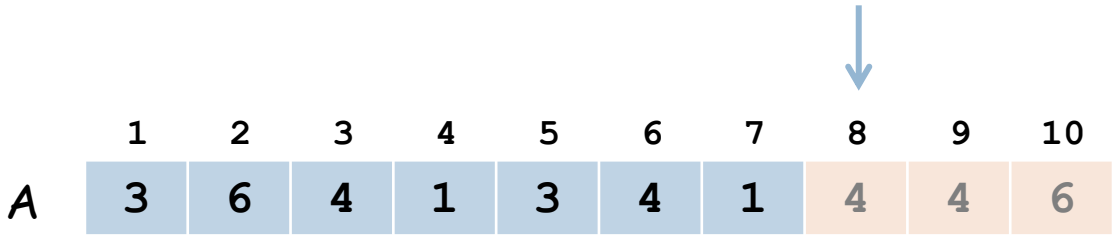
$A(8) = 4$

$C(4) = 7$

Αλγόριθμος Counting-Sort

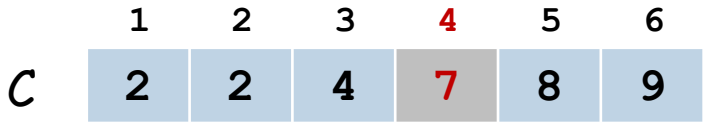
Παράδειγμα

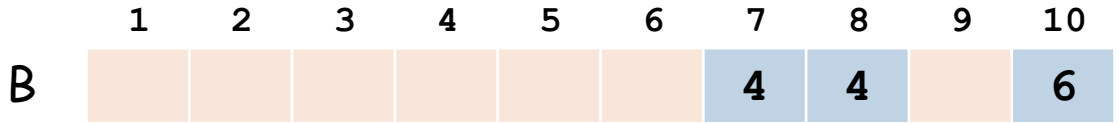
Βήμα 3



$A(8) = 4$

$C(4) = 7$



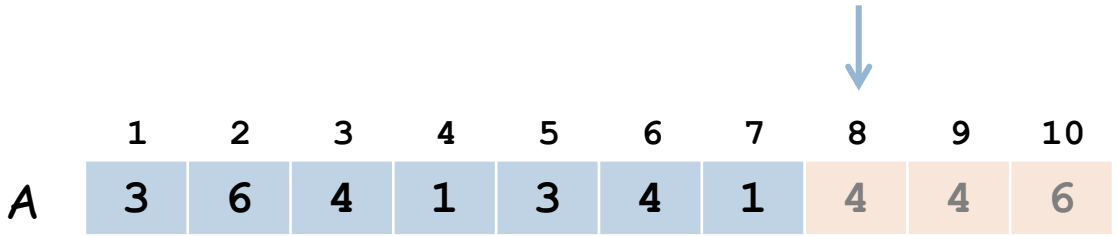


$B(7) = 4$

Αλγόριθμος Counting-Sort

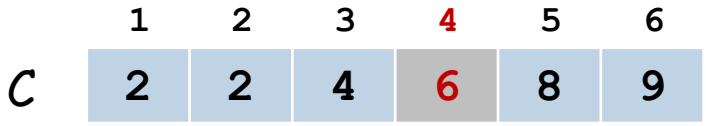
Παράδειγμα

Βήμα 3

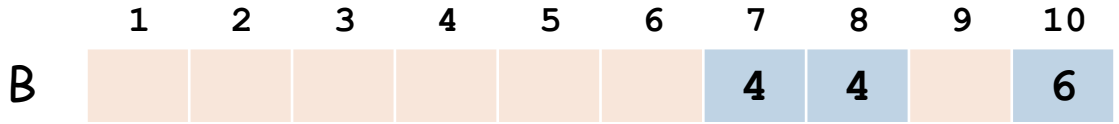


A(8) = 4

C(4) = 7



C(4) --

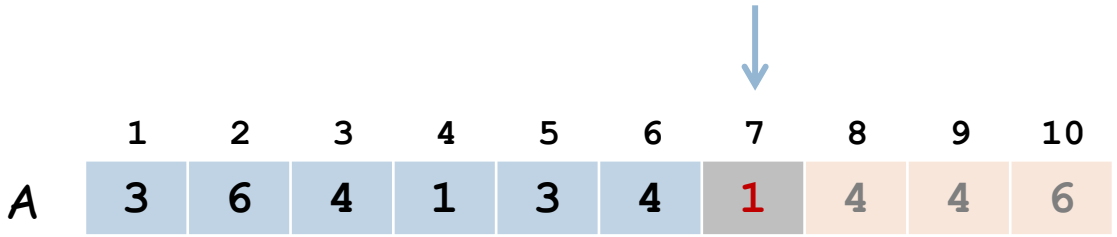


B(7) = 4

Αλγόριθμος Counting-Sort

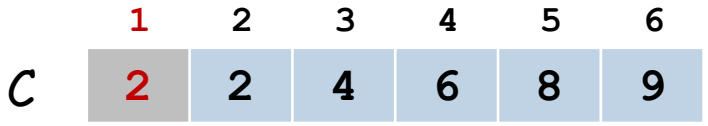
Παράδειγμα

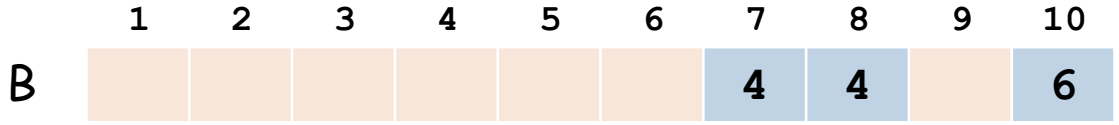
Βήμα 4



$A(7) = 1$

$C(1) = 2$

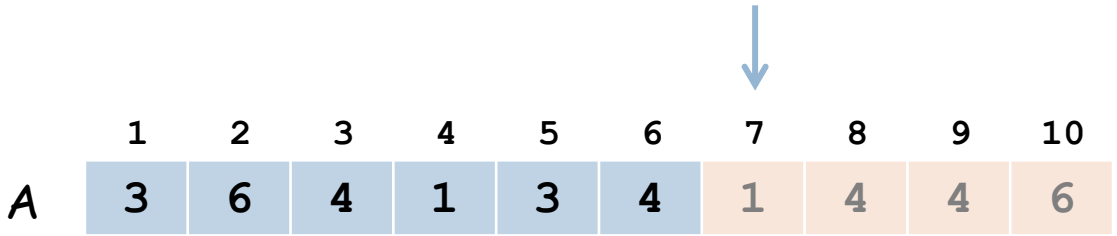




Αλγόριθμος Counting-Sort

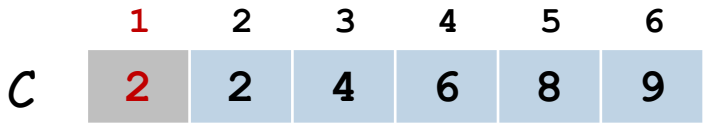
Παράδειγμα

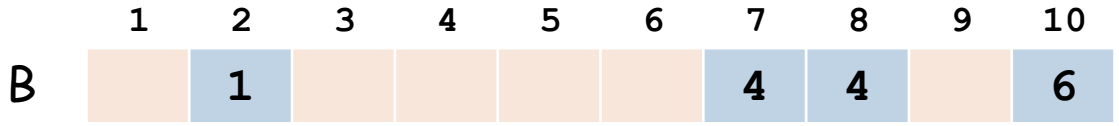
Βήμα 4



A(7) = 1

C(1) = 2



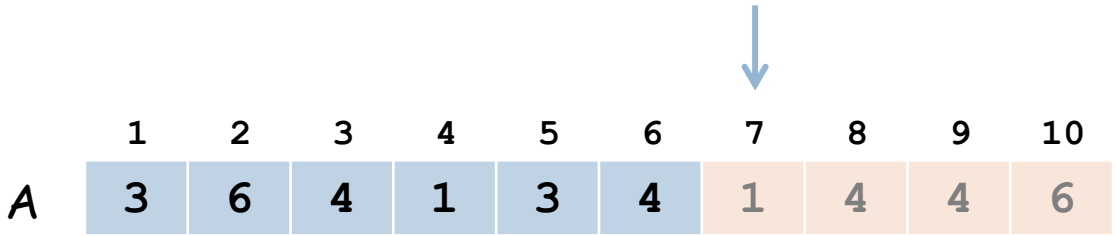


B(2) = 1

Αλγόριθμος Counting-Sort

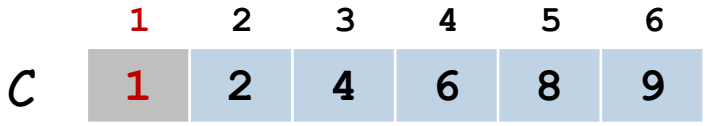
Παράδειγμα

Βήμα 4

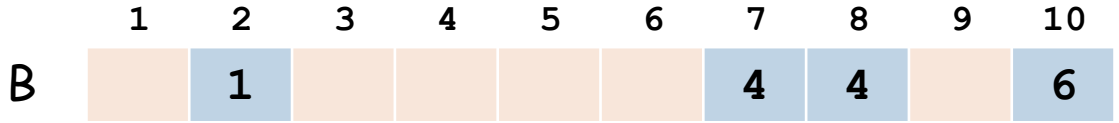


A(7) = 1

C(1) = 2



C(1) --

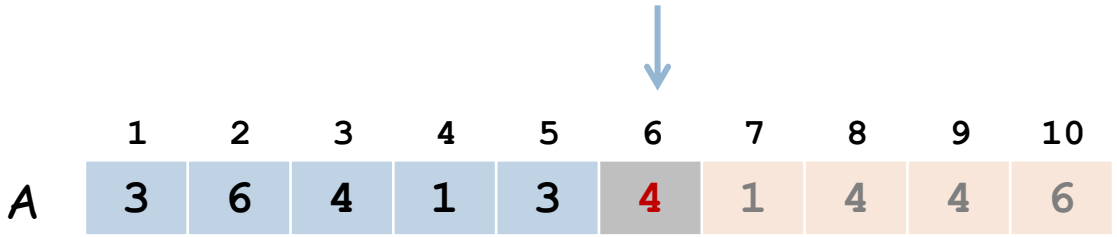


B(2) = 1

Αλγόριθμος Counting-Sort

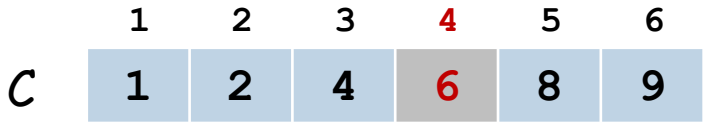
Παράδειγμα

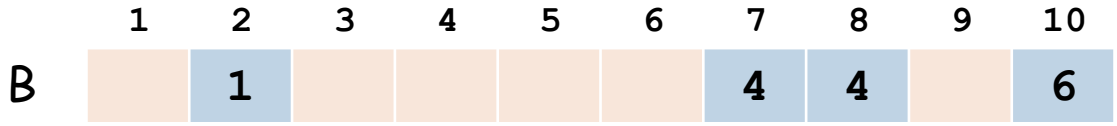
Βήμα 5



$A(6) = 4$

$C(4) = 6$

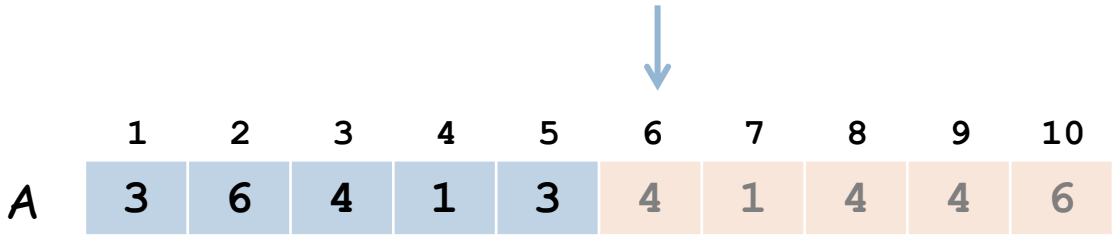




Αλγόριθμος Counting-Sort

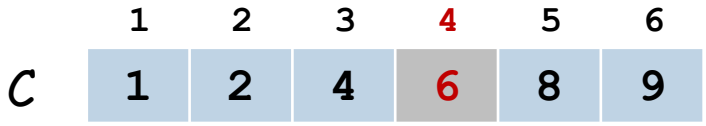
Παράδειγμα

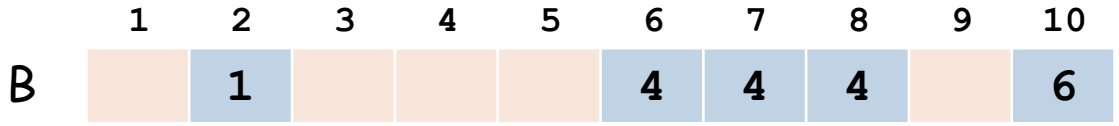
Βήμα 5



A(6) = 4

C(4) = 6



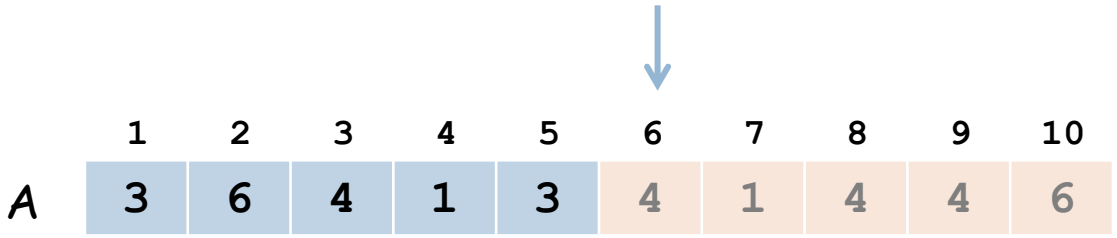


B(6) = 4

Αλγόριθμος Counting-Sort

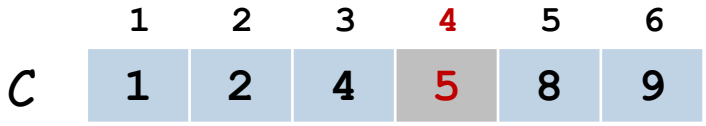
Παράδειγμα

Βήμα 5

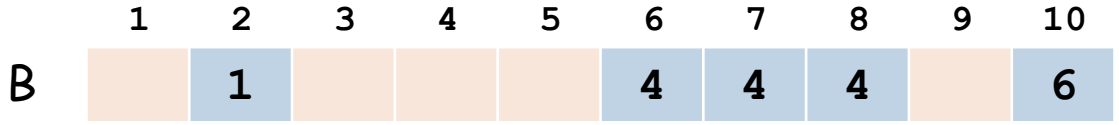


A(6) = 4

C(4) = 6



C(4) --

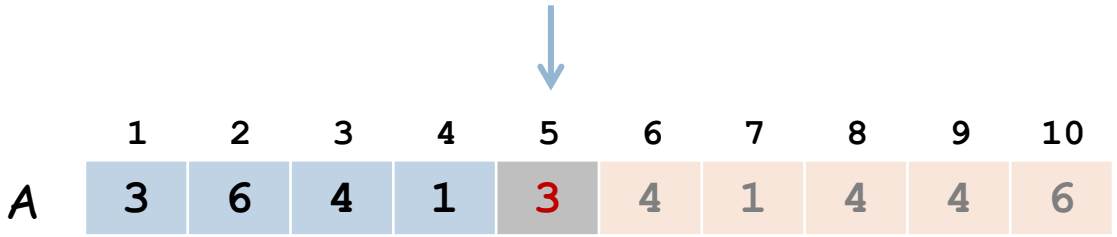


B(6) = 4

Αλγόριθμος Counting-Sort

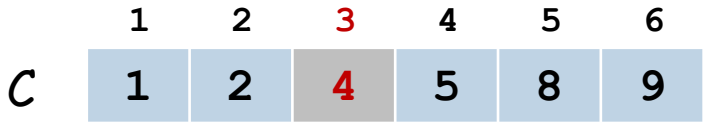
Παράδειγμα

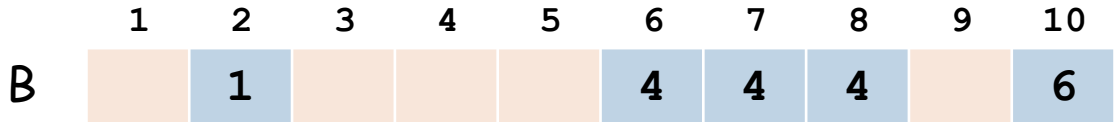
Βήμα 6



$A(5) = 3$

$C(3) = 4$

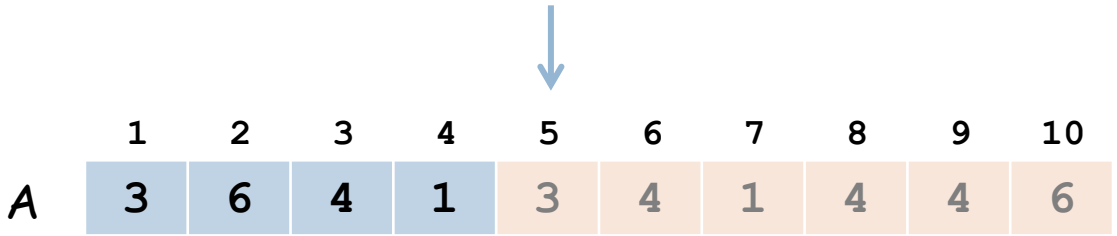




Αλγόριθμος Counting-Sort

Παράδειγμα

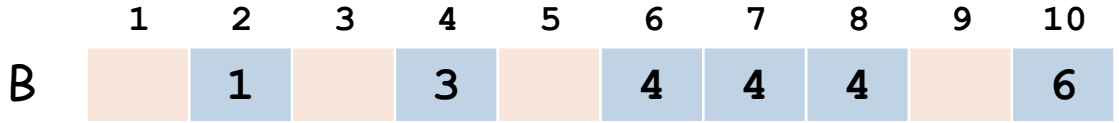
Βήμα 6



A(5) = 3

C(3) = 4

C(3) --

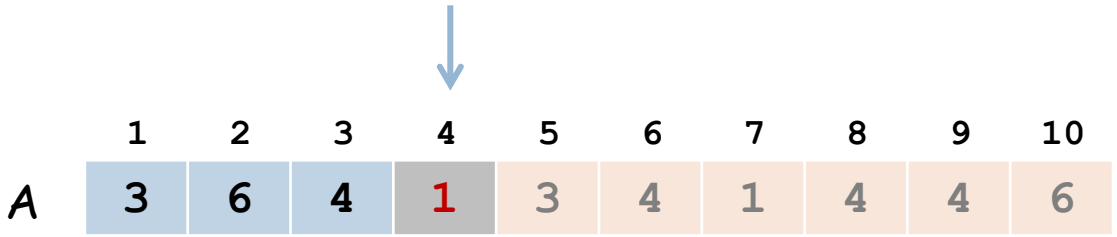


B(4) = 3

Αλγόριθμος Counting-Sort

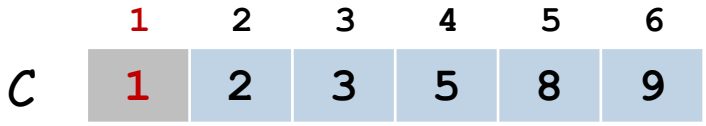
Παράδειγμα

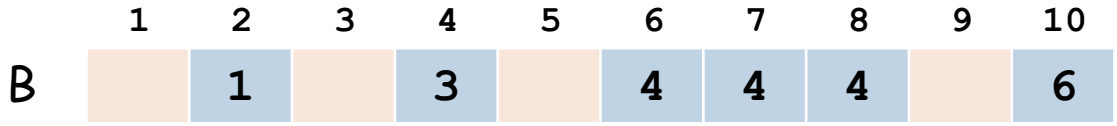
Βήμα 7



$A(4) = 1$

$C(1) = 1$

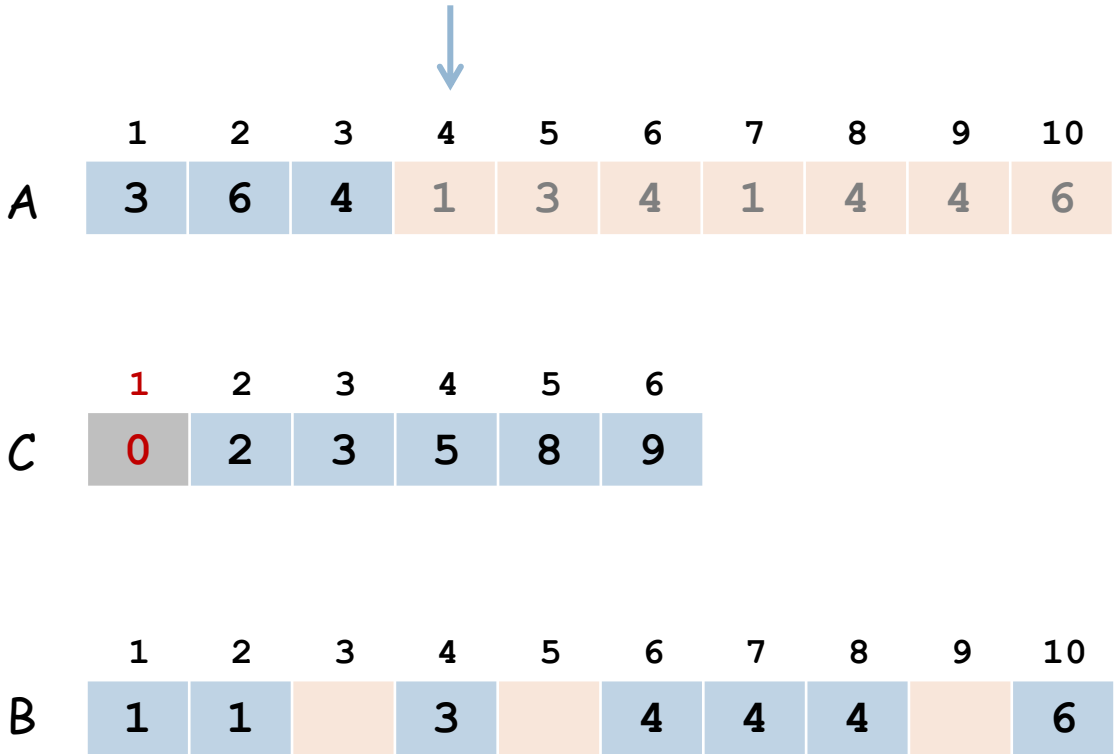




Αλγόριθμος Counting-Sort

Παράδειγμα

Βήμα 7



A(4) = 1

C(1) = 1

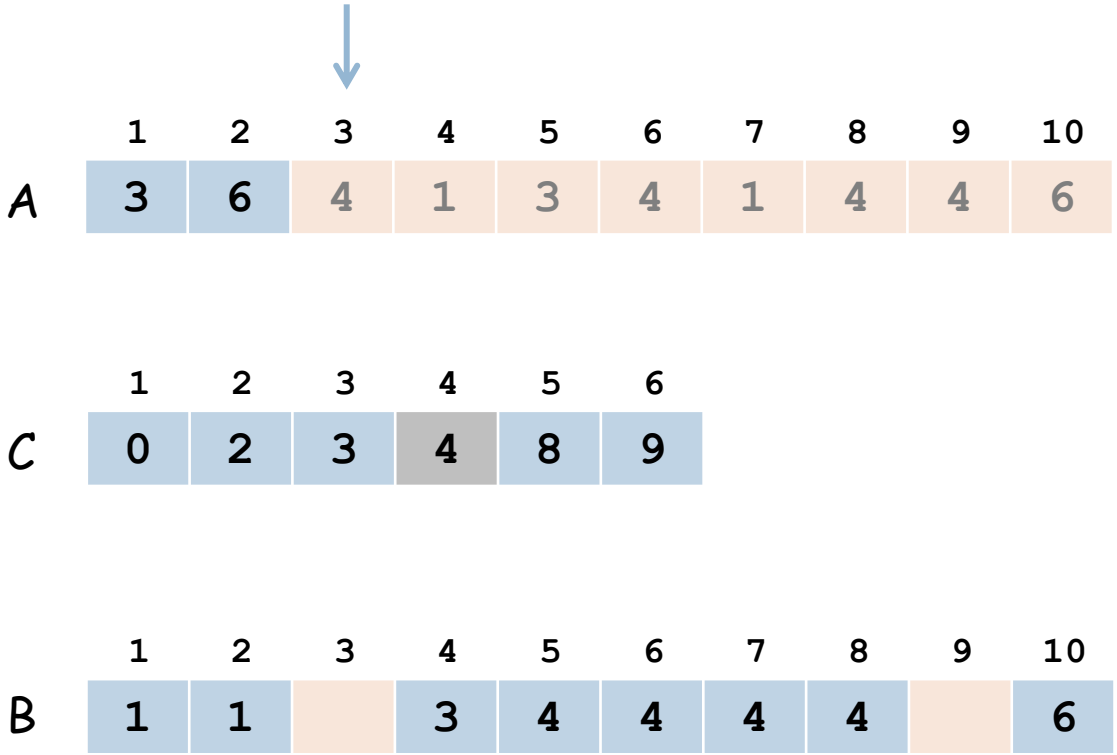
C(1) --

B(1) = 1

Αλγόριθμος Counting-Sort

Παράδειγμα

Βήμα 8



A(3) = 4

C(4) = 5

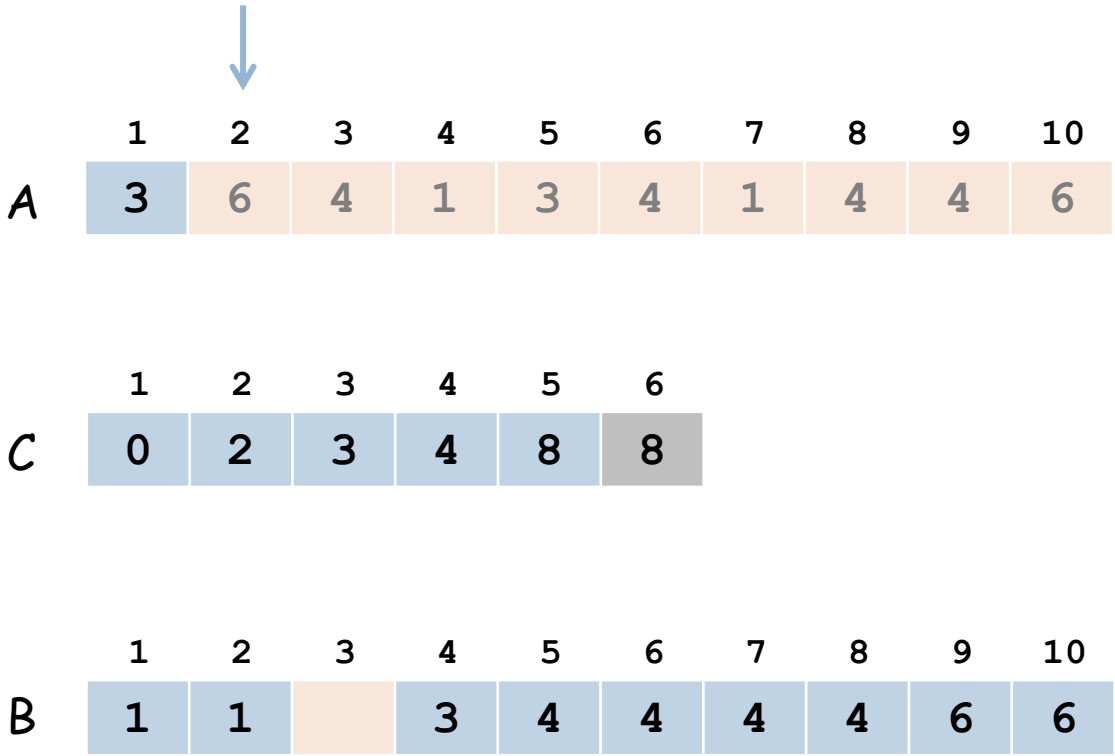
C(4) --

B(5) = 4

Αλγόριθμος Counting-Sort

Παράδειγμα

Βήμα 9



Αλγόριθμος Counting-Sort

Παράδειγμα

Βήμα 10

	1	2	3	4	5	6	7	8	9	10
A	3	6	4	1	3	4	1	4	4	6

A(1) = 3

C(3) = 3

	1	2	3	4	5	6
C	0	2	2	4	8	8

C(3) --

	1	2	3	4	5	6	7	8	9	10
B	1	1	3	3	4	4	4	4	6	6

B(3) = 3

Αλγόριθμος Counting-Sort

Παράδειγμα

	1	2	3	4	5	6	7	8	9	10
A	3	6	4	1	3	4	1	4	4	6

A(1) = 3

C(3) = 3

	1	2	3	4	5	6
C	0	2	2	4	8	8

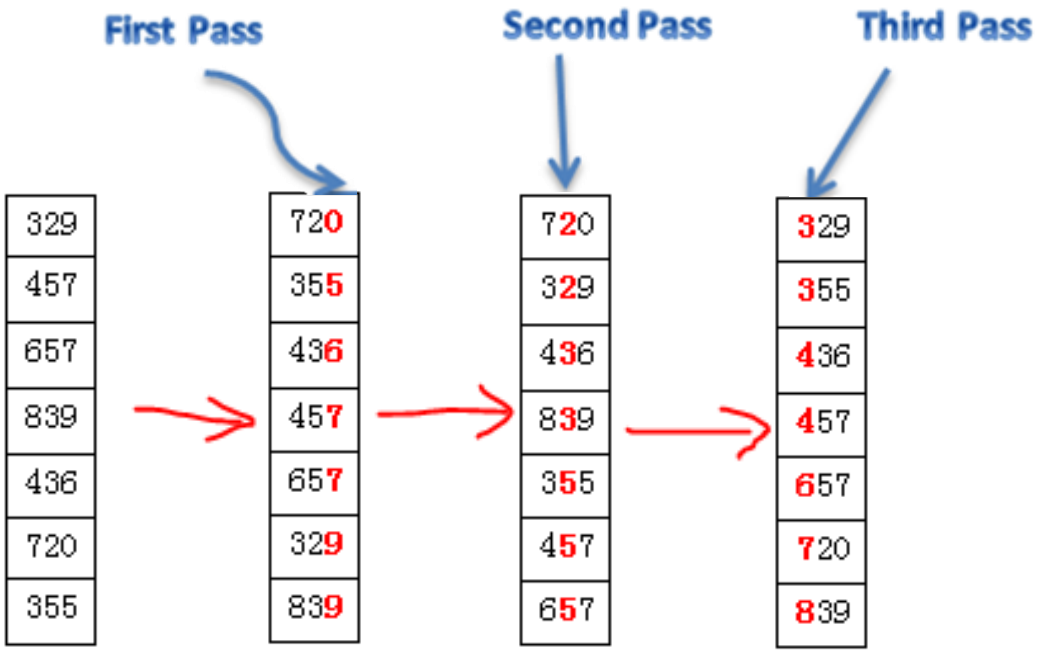
C(3) --

	1	2	3	4	5	6	7	8	9	10
B	1	1	3	3	4	4	4	4	6	6

B(3) = 3

Τέλος Αλγόριθμου

Αλγόριθμος Radix-Sort



Radix Sort

Αλγόριθμος Radix-Sort

Μια εφαρμογή ταξινόμησης!!!

- ✓ Έστω ότι θέλουμε να ταξινομήσουμε N άτομα ως προς την ημερομηνία γέννησής τους (έτος-μήνας-ημέρα) κατά φθίνουσα τάξη.

```
2016-05-22
1980-03-12
1995-16-03
2012-10-14
2016-11-26
2012-05-26
2016-09-17
2016-05-26
1980-03-15
```

Ένας απλός τρόπος με την χρήση του αλγόριθμου **Counting-Sort** !!!

- ✓ Ταξινομούμε αρχικά τα N άτομα ως προς το έτος γέννησής τους!
- ✓ Στη συνέχεια, άτομα με το ίδιο έτος γέννησης τα ταξινομούμε ως προς το μήνα γέννησης τους !!
- ✓ Τέλος, άτομα με το ίδιο έτος και μήνα γέννησης τα ταξινομούμε ως προς την ημέρα γέννησης τους !!

Υπάρχει κάποια «δυσκολία» σε αυτό τον τρόπο ?

Αλγόριθμος Radix-Sort

Μια εφαρμογή ταξινόμησης!!!

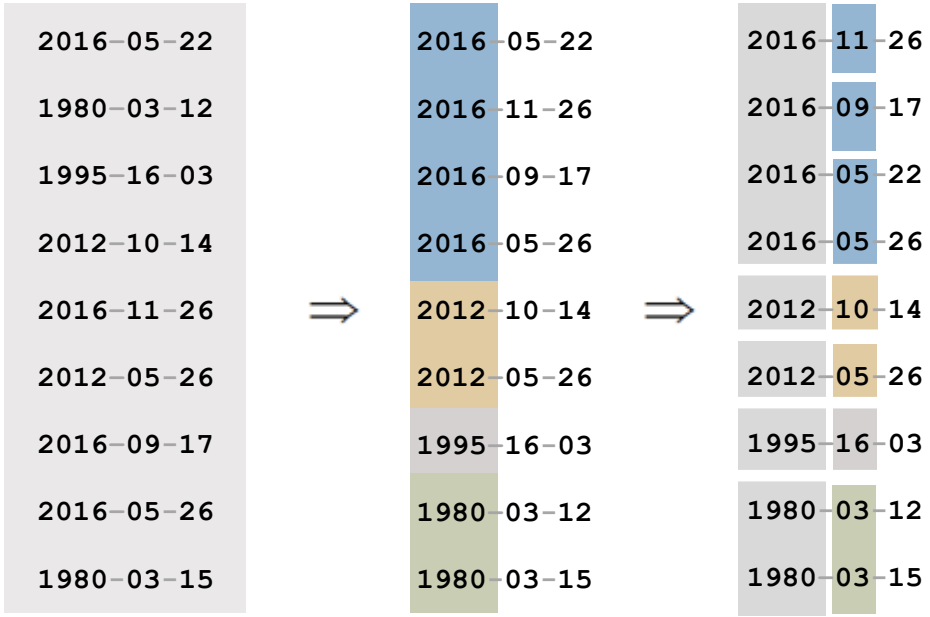
- ✓ Έστω ότι θέλουμε να ταξινομήσουμε N άτομα ως προς την ημερομηνία γέννησής τους (έτος-μήνας-ημέρα) κατά φθίνουσα τάξη.

2016-05-22		2016-05-22
1980-03-12		2016-11-26
1995-16-03		2016-09-17
2012-10-14		2016-05-26
2016-11-26	⇒	2012-10-14
2012-05-26		2012-05-26
2016-09-17		1995-16-03
2016-05-26		1980-03-12
1980-03-15		1980-03-15

Αλγόριθμος Radix-Sort

Μια εφαρμογή ταξινόμησης!!!

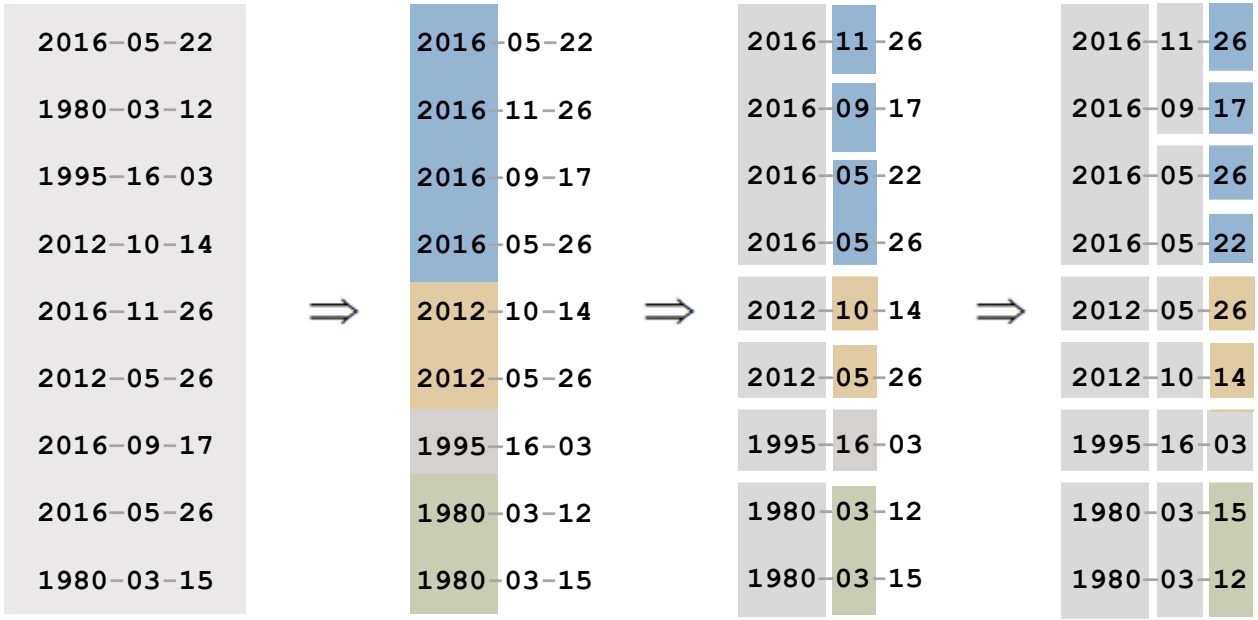
- ✓ Έστω ότι θέλουμε να ταξινομήσουμε N άτομα ως προς την ημερομηνία γέννησής τους (έτος-μήνας-ημέρα) κατά φθίνουσα τάξη.



Αλγόριθμος Radix-Sort

Μια εφαρμογή ταξινόμησης!!!

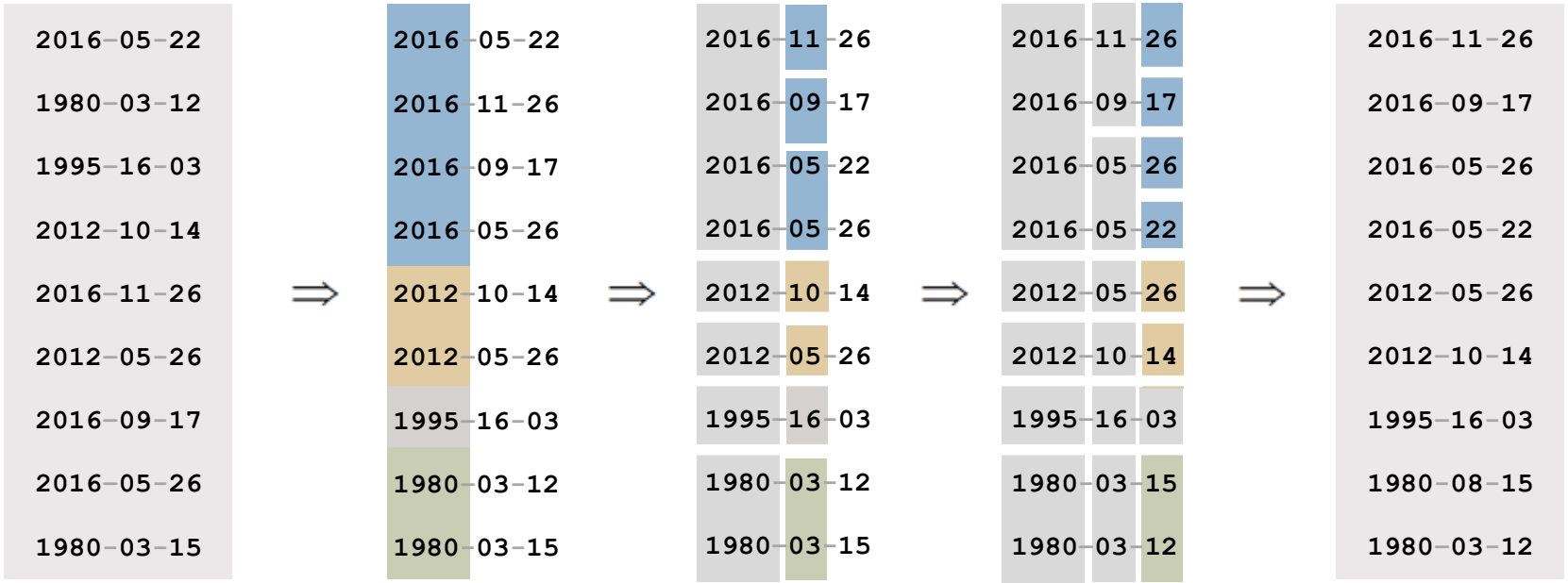
- ✓ Έστω ότι θέλουμε να ταξινομήσουμε N άτομα ως προς την ημερομηνία γέννησής τους (έτος-μήνας-ημέρα) κατά φθίνουσα τάξη.



Αλγόριθμος Radix-Sort

Μια εφαρμογή ταξινόμησης!!!

- ✓ Έστω ότι θέλουμε να ταξινομήσουμε N άτομα ως προς την ημερομηνία γέννησής τους (έτος-μήνας-ημέρα) κατά φθίνουσα τάξη.



Αλγόριθμος Radix-Sort

Ιδέα Αλγόριθμου !!!

- ✓ Έστω ότι τα στοιχεία είναι ακέραιοι με k ψηφία καταχωρημένα σε ένα πίνακα A.
- ✓ Ο αλγόριθμος **Radix-Sort** εργάζεται ως εξής:

(1) Ταξινόμησε ως προς το πιο δεξί ψηφίο
(i) ...
(k) Ταξινόμησε ως προς το πιο αριστερό ψηφίο

} k ταξινομήσεις Counting sort

Παράδειγμα :

329
457
657
839
436
720
355

Αλγόριθμος Radix-Sort

Ιδέα Αλγόριθμου !!!

- ✓ Έστω ότι τα στοιχεία είναι ακέραιοι με k ψηφία καταχωρημένα σε ένα πίνακα A.
- ✓ Ο αλγόριθμος **Radix-Sort** εργάζεται ως εξής:

(1) Ταξινόμησε ως προς το πιο δεξί ψηφίο
(i) ...
(k) Ταξινόμησε ως προς το πιο αριστερό ψηφίο

} k ταξινομήσεις Counting sort

Παράδειγμα :

329	3	2	9
457	4	5	7
657	6	5	7
839	8	3	9
436	4	3	6
720	7	2	0
355	3	5	5

Αλγόριθμος Radix-Sort

Ιδέα Αλγόριθμου !!!

- ✓ Έστω ότι τα στοιχεία είναι ακέραιοι με k ψηφία καταχωρημένα σε ένα πίνακα A.
- ✓ Ο αλγόριθμος **Radix-Sort** εργάζεται ως εξής:

(1) Ταξινόμησε ως προς το πιο δεξί ψηφίο
(i) ...
(k) Ταξινόμησε ως προς το πιο αριστερό ψηφίο

} k ταξινομήσεις Counting sort

Παράδειγμα :

329	3	2	9
457	4	5	7
657	6	5	7
839	8	3	9
436	4	3	6
720	7	2	0
355	3	5	5

$\Rightarrow$

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

Αλγόριθμος Radix-Sort

Ιδέα Αλγόριθμου !!!

- ✓ Έστω ότι τα στοιχεία είναι ακέραιοι με k ψηφία καταχωρημένα σε ένα πίνακα A.
- ✓ Ο αλγόριθμος **Radix-Sort** εργάζεται ως εξής:

(1) Ταξινόμησε ως προς το πιο δεξί ψηφίο
(i) ...
(k) Ταξινόμησε ως προς το πιο αριστερό ψηφίο

} k ταξινομήσεις Counting sort

Παράδειγμα :

329	3	2	9
457	4	5	7
657	6	5	7
839	8	3	9
436	4	3	6
720	7	2	0
355	3	5	5

⇒

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

⇒

7	2	0
3	2	9
4	3	6
8	3	9
3	5	5
4	5	7
6	5	7

Αλγόριθμος Radix-Sort

Ιδέα Αλγόριθμου !!!

- ✓ Έστω ότι τα στοιχεία είναι ακέραιοι με k ψηφία καταχωρημένα σε ένα πίνακα A.
- ✓ Ο αλγόριθμος **Radix-Sort** εργάζεται ως εξής:

(1) Ταξινόμησε ως προς το πιο δεξί ψηφίο
(i) ...
(k) Ταξινόμησε ως προς το πιο αριστερό ψηφίο

} k ταξινομήσεις Counting sort

Παράδειγμα :

329	3	2	9	⇒	7	2	0	⇒	7	2	0	⇒	3	2	9
457	4	5	7		3	5	5		3	2	9		3	5	5
657	6	5	7		4	3	6		4	3	6		4	3	6
839	8	3	9		4	5	7		8	3	9		4	5	7
436	4	3	6		6	5	7		3	5	5		6	5	7
720	7	2	0		3	2	9		4	5	7		7	2	0
355	3	5	5		8	3	9		6	5	7		8	3	9

Αλγόριθμος Radix-Sort

Πολυπλοκότητα

- ✓ $\Theta(n + 10)$ κάθε ταξινόμηση Counting Sort
  k ψηφία: $\Rightarrow$ $\Theta(n \cdot k)$
  $k = \log_{10}(A[i])$. Αν $A[i] = n^d = 10^{d'}$ τότε (για $n > 10$):
  $\Theta(n \cdot \log_{10}(10^{d'})) = \Theta(n \cdot d) = \Theta(n)$

Παράδειγμα :

329	3	2	9
457	4	5	7
657	6	5	7
839	8	3	9
436	4	3	6
720	7	2	0
355	3	5	5

$\Rightarrow$

7	2	0
3	5	5
4	3	6
4	5	7
6	5	7
3	2	9
8	3	9

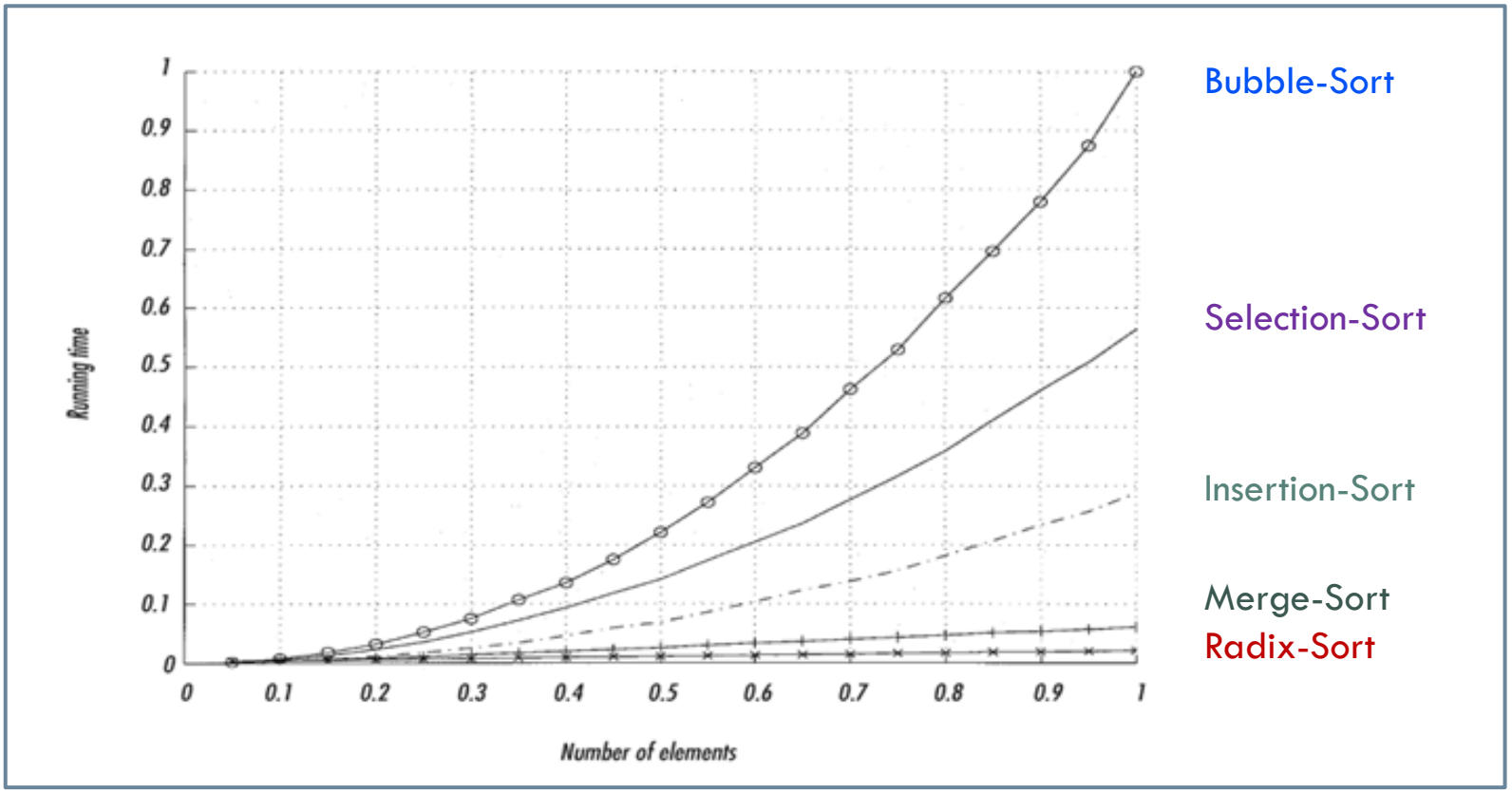
$\Rightarrow$

7	2	0
3	2	9
4	3	6
8	3	9
3	5	5
4	5	7
6	5	7

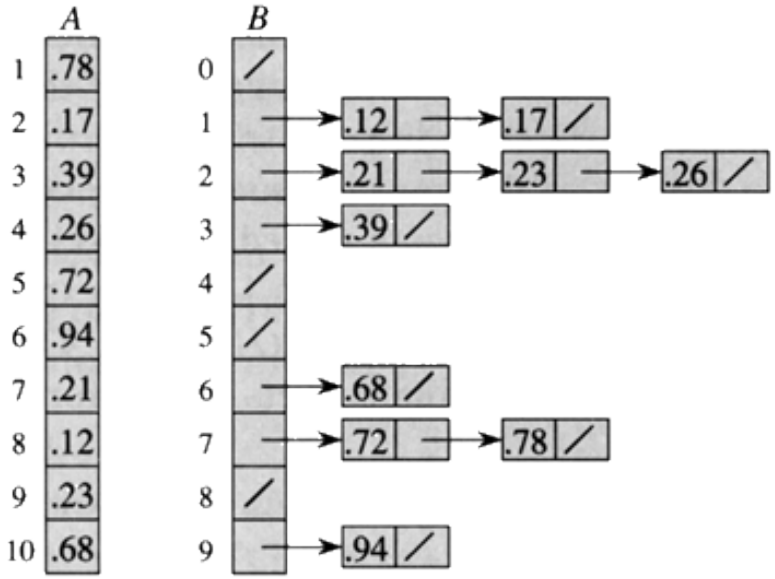
$\Rightarrow$

3	2	9
3	5	5
4	3	6
4	5	7
6	5	7
7	2	0
8	3	9

Σύγκριση Αλγορίθμων Ταξινόμησης



Αλγόριθμος Bucket-Sort



Αλγόριθμος Bucket-Sort

Ιδέα Αλγόριθμου !!!

✓ Ο αλγόριθμος της ταξινόμησης με κάδους (Bucket-Sort) δέχεται ότι τα n προς ταξινόμηση στοιχείων παράγονται από μια τυχαία διαδικασία η οποία κατανέμει τα στοιχεία ομοιόμορφα στο διάστημα [0, 1).

Τεχνική : Ο αλγόριθμος χωρίζει το διάστημα [0, 1) σε **n** ίσου μήκους υποδιαστήματα, ή «κάδους» όπως συνηθίζονται να λέγονται, και κατόπιν να κατανέμει τα **n** στοιχεία στους κάδους.

- Τα στοιχεία των κάδων ταξινομούνται με την χρήση ενός αλγόριθμου ταξινόμησης.
- Ο αλγόριθμος εκτελείται σε χρόνο **O(n)**

Αλγόριθμος Bucket-Sort

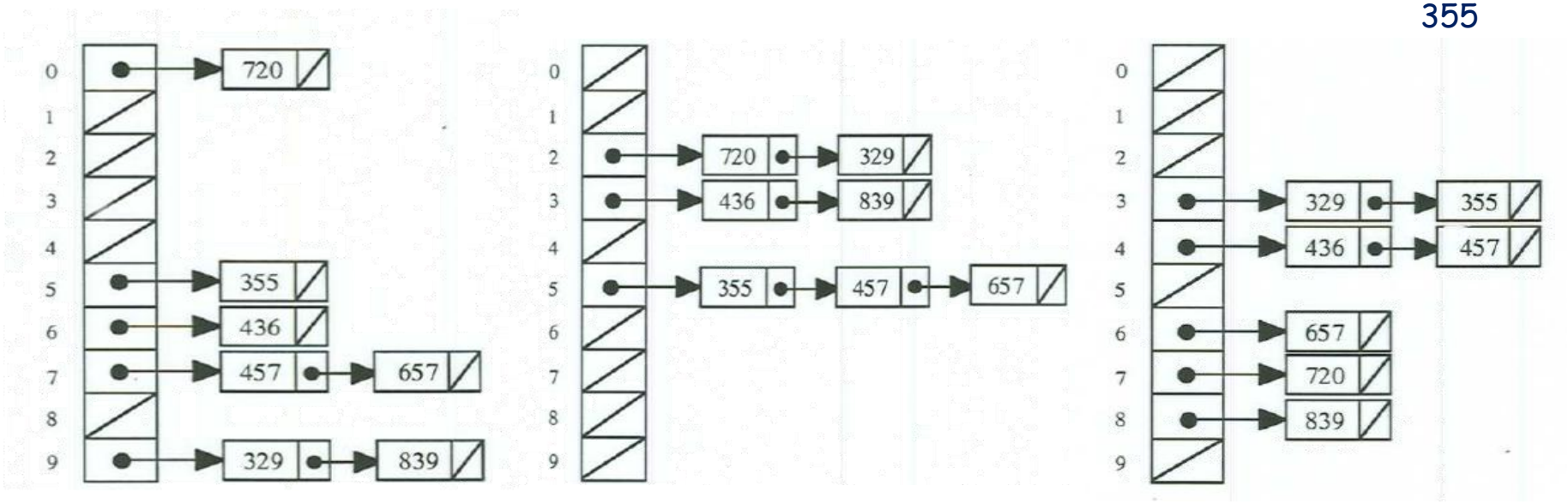
Παράδειγμα: Έστω ότι θέλουμε να ταξινομήσουμε τους 3-ψήφιους ακεραίους:

329
457
657
839
436
720
355

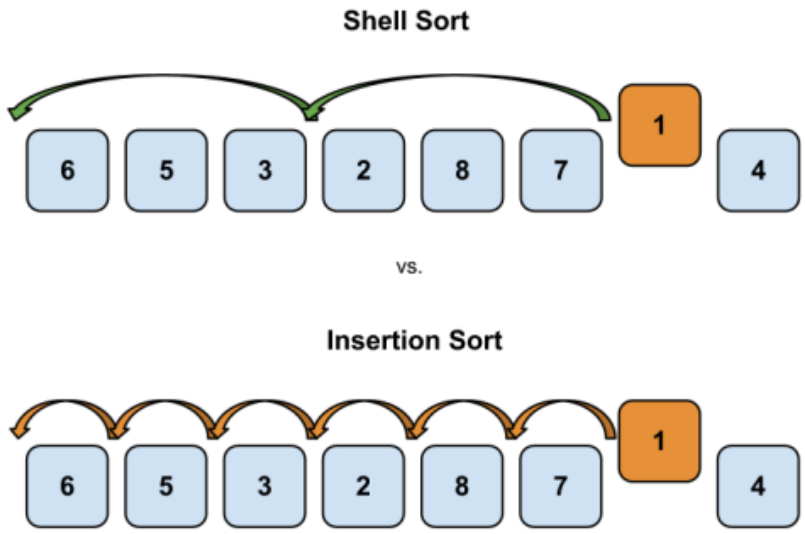
Πολυπλοκότητα: k ψηφία: $\Rightarrow$ $\Theta(n \cdot k)$

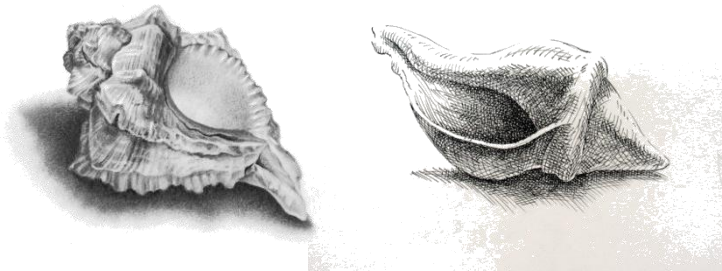
$k = \log_{10}(A[i])$. Αν $A[i] = n^d = 10^{d'}$ τότε (για $n > 10$):

$\Theta(n \cdot \log_{10}(10^{d'})) = \Theta(n \cdot d) = \Theta(n)$



Αλγόριθμος Shell-Sort





Αλγόριθμος Shell-Sort

Ιδέα Αλγόριθμου !!!

- ✓ Η τεχνική του αλγόριθμου είναι ενδιαφέρουσα, ο αλγόριθμος προγραμματίζεται εύκολα και εκτελείται γρήγορα.

Τεχνική : Ο αλγόριθμος ταξινομεί μια ακολουθία **S** με **n** στοιχεία με επιτυχή ταξινόμηση υποακολουθιών της ακολουθία **S**.

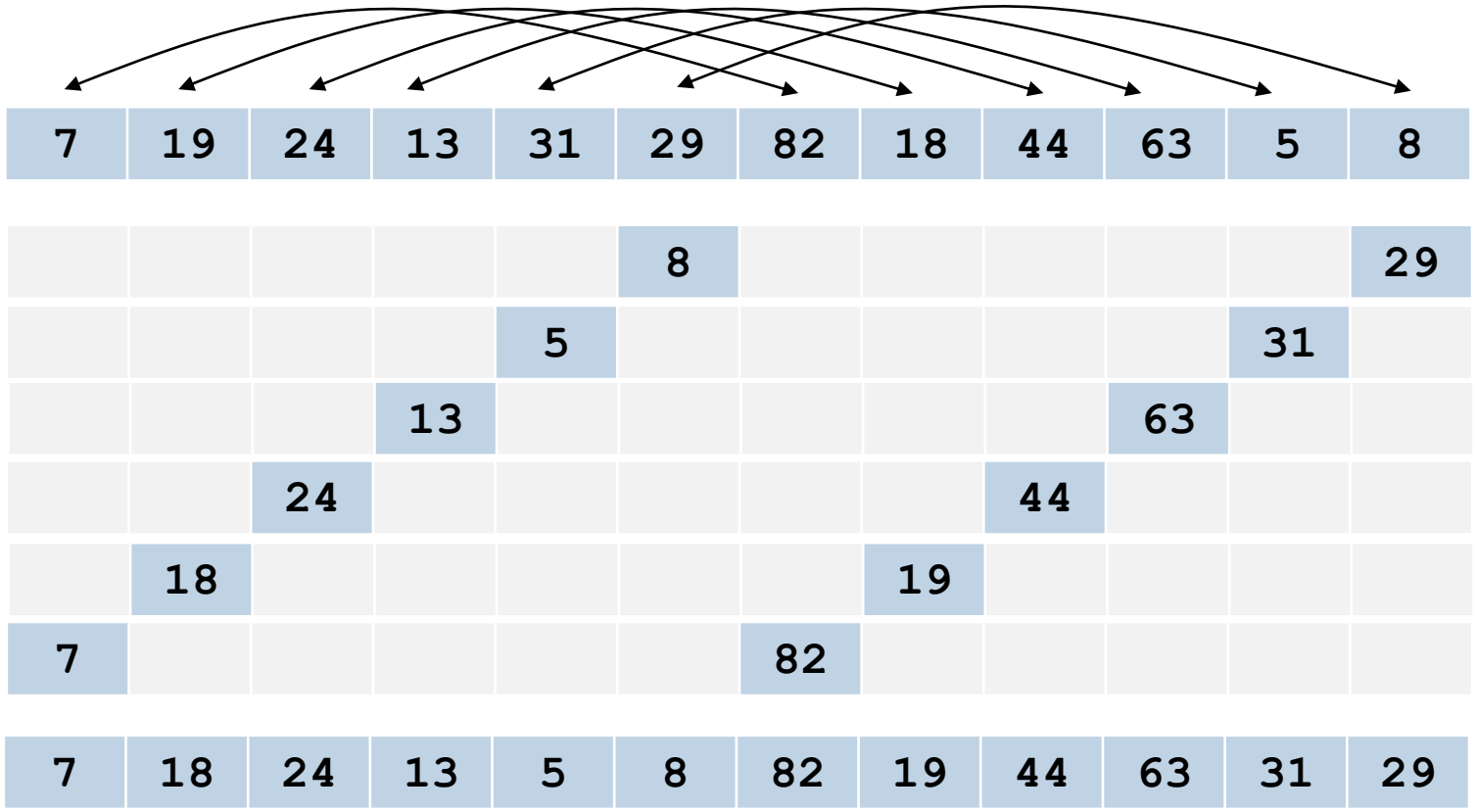
- Οι υποακολουθίες καθορίζονται από μια σειρά βημάτων προσαύξησης (increments):

$$h_1, h_2, h_3, \ldots, h_{t-1}, h_t$$

Αλγόριθμος Shell-Sort

Παράδειγμα

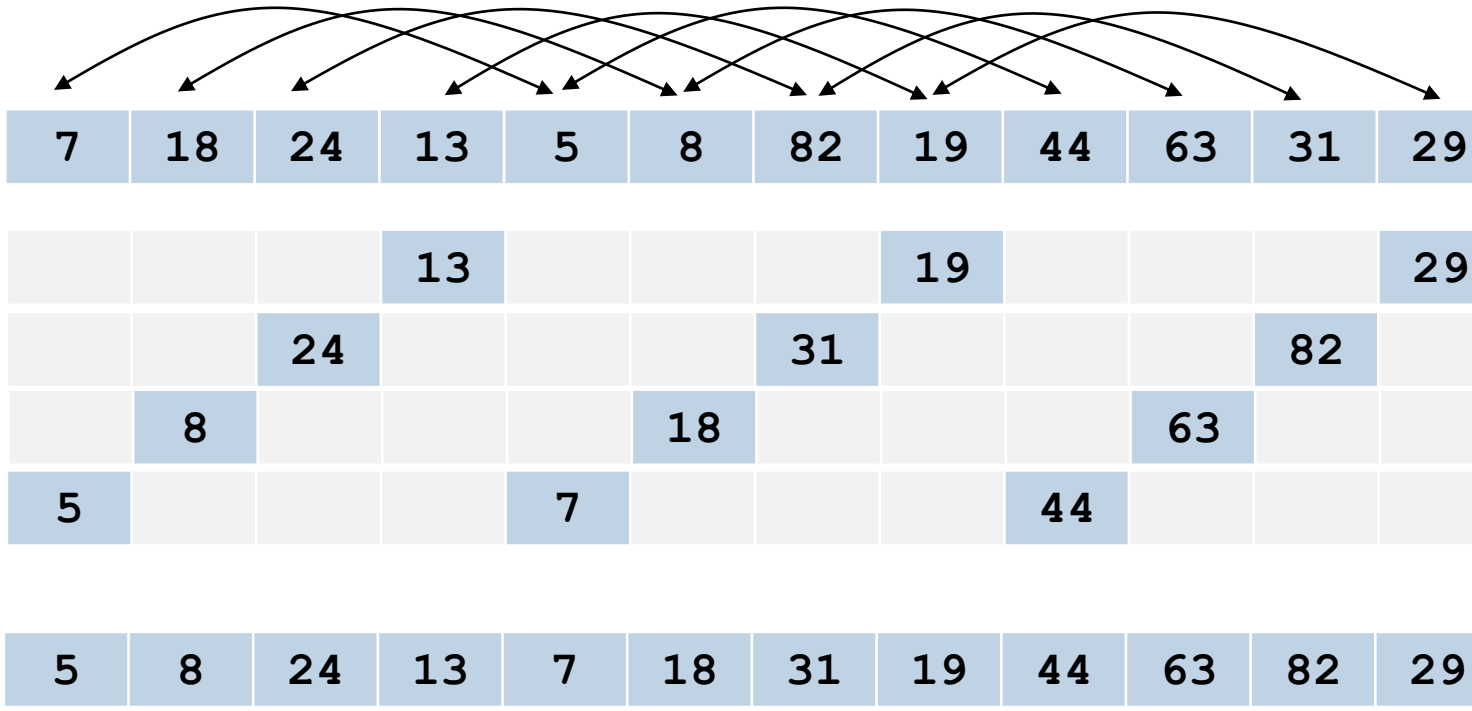
$h_1 = 6$



Αλγόριθμος Shell-Sort

Παράδειγμα

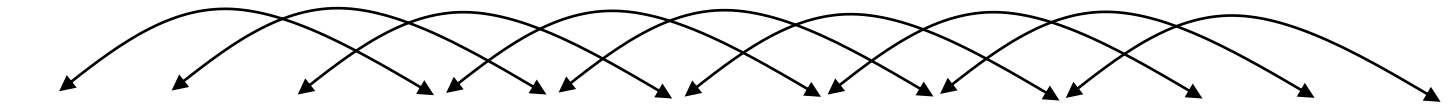
$h_2 = 4$



Αλγόριθμος Shell-Sort

Παράδειγμα

$h_3 = 3$



5	8	24	13	7	18	31	19	44	63	82	29

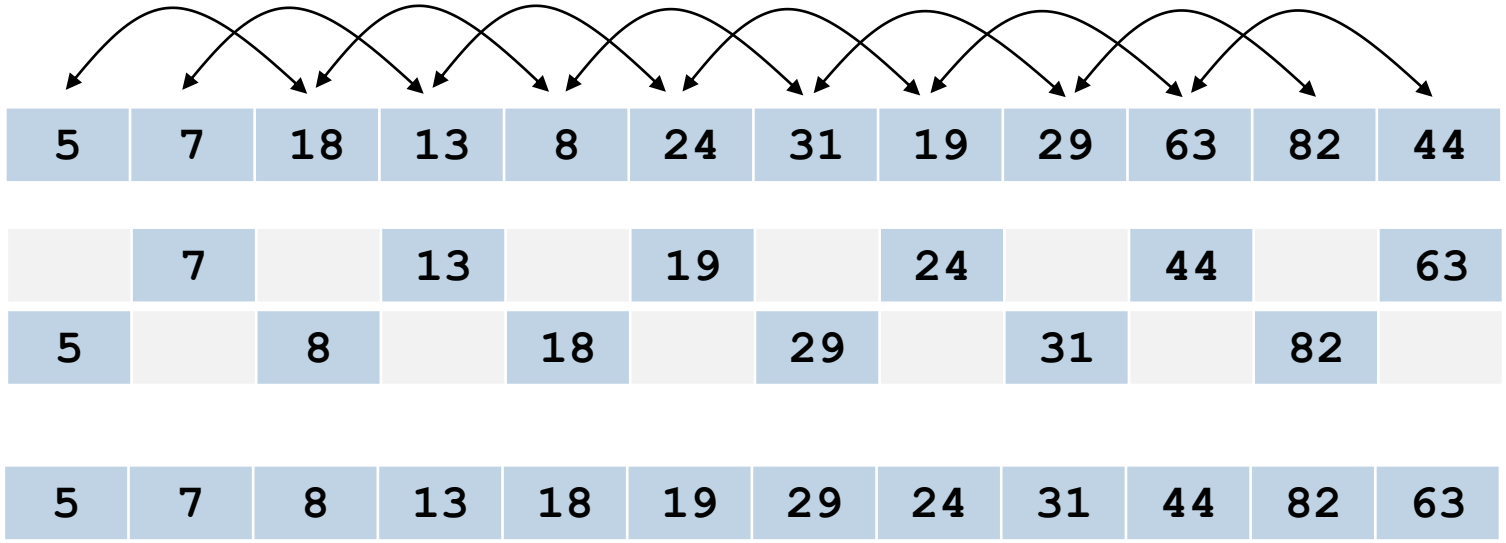
		18			24			29			44
	7			8			19			82	
5			13			31			63		

5	7	18	13	8	24	31	19	29	63	82	44

Αλγόριθμος Shell-Sort

Παράδειγμα

$h_4 = 2$



Αλγόριθμος Shell-Sort

Παράδειγμα

$h_5 = 1$

5	7	8	13	18	19	29	24	31	44	82	63

5	7	8	13	18	19	29	24	31	44	63	82

Ταξινομημένη Ακολουθία

5	7	8	13	18	19	29	24	31	44	63	82

Σύγκριση Αλγορίθμων Ταξινόμησης

algorithm	stable?	inplace?	running time
selection sort	no	yes	N^2
insertion sort	yes	yes	between N and N^2
shellsort	no	yes	$N^{6/5}$?
quicksort	no	yes	$N \lg N$
mergesort	yes	no	$N \lg N$
heapsort	no	yes	$N \lg N$

Ευχαριστώ για τη Συμμετοχή σας !!!

Σταύρος Δ. Νικολόπουλος

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

Webpage: www.cs.uoi.gr/~stavros