Σχεδίαση και Ανάλυση Αλγορίθμων

Ενότητα 3.0

Διαίρει και Βασίλευε

Quick-sort και Merge-sort

Μέτρηση Αντιστρόφων

Πλησιέστερο Ζεύγος Σημείων

Αλγόριθμος Strassen

Σταύρος Δ. Νικολόπουλος | 2017-18

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

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

Γενική Αρχή Τεχνικών Σχεδίασης και Ανάλυσης Αλγορίθμων

Καθορισμός *ενδιάμεσων μικρότερων στόχων* που μπορούν να επιτευχθούν πιο εύκολα και συμβάλουν στην επίτευξη του *τελικού στόχου*!!!

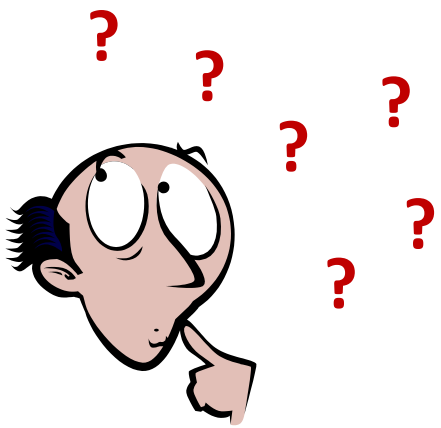
Ενδιάμεσοι μικρότεροι στόχοι $\Rightarrow$ Λύση μικρότερων προβλημάτων !!!

Τελικός στόχος $\Rightarrow$ Λύση αρχικού προβλήματος !!!

Αρχή της Εξισορρόπησης

Ισορροπία μεταξύ των μεγεθών

των μικρότερων προβλημάτων



Αρχή της Εξισορρόπησης

Γενικός Κανόνας: Η διατήρηση μια ισορροπίας, ή και ισότητας, μεταξύ των μεγεθών $\mathbf{n_1}$, $\mathbf{n_2}$, …, $\mathbf{n_k}$ των υπο-προβλημάτων Π_1, Π_2, …, Π_k που προκύπτουν από τη διάσπαση ενός προβλήματος Π μεγέθους **n** !!!

Επιχειρήματα υπέρ της εξισορρόπησης

- Πρέπει να υπάρχει ίση μεταχείριση !!!
- Ο χρόνος εκτέλεσης ενός έργου είναι αύξουσα συνάρτηση κάποιου μεγέθους που αντιστοιχεί στο έργο. Αν έχουμε να εκτελέσουμε δυο έργα είναι *προτιμότερο να έχουν* το *ίδιο μέγεθος !!!* (είτε τα εκτελέσουμε ακολουθιακά είτε παράλληλα).

Εξισορρόπηση - Θεώρημα

Θεώρημα: Έστω T(n) ο χρόνος επίλυσης ενός προβλήματος Π μεγέθους **n**, όπου οι συναρτήσεις T και T′ είναι αυστηρά αύξουσες, και έστω $\mathbf{n_1}$, $\mathbf{n_2}$ τα μεγέθη δύο υποπροβλημάτων $Π_1$ και $Π_2$ του Π με $\mathbf{n_1 + n_2 = n}$. Τότε, ο *ελάχιστος χρόνος* για την ακολουθιακή λύση των υποπροβλημάτων $Π_1$ και $Π_2$ επιτυγχάνεται όταν $\mathbf{n_1 = n_2 = n/2}$.

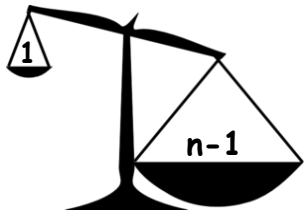
Απόδειξη:

- Συνολικός χρόνος επίλυσης $T(n_1) + T(n_2) = T(n_1) + T(n-n_1) = f(n_1)$.
- Για να υπάρχει ελάχιστο πρέπει $f'(n_1) = 0$, δηλαδή $T'(n_1) - T'(n-n_1) = 0$.
- Οπότε, $T'(n_1) = T'(n-n_1)$. Επειδή η συνάρτηση T′ είναι αυστηρά αύξουσα, η παραπάνω ισότητα ισχύει όταν $n_1 = n-n_1$, δηλαδή $n_1 = n_2 = n/2$.
- Ακόμη ισχύει $f''(n_1) = T''(n_1) + T''(n-n_1) > 0$. Άρα, για $n_1 = n_2 = n/2$ η συνάρτηση $f(n_1)$ εμφανίζει ελάχιστο.

Εξισορρόπηση - Παράδειγμα

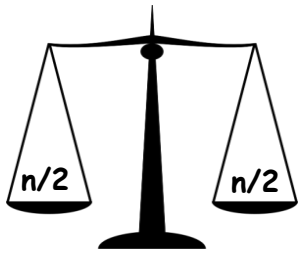
Έστω το πρόβλημα της αναζήτησης σε ακολουθία **n** στοιχείων.

- Στη ***σειριακή αναζήτηση*** η σύγκριση με το 1ο στοιχείο ανάγει την επίλυση του προβλήματος σε μικρότερο στιγμιότυπο μεγέθους **n-1** στοιχείων.
  - Η σειριακή αναζήτηση *δεν εφαρμόζει* την τεχνική της εξισορρόπησης !



- Στη ***δυαδική αναζήτηση*** η πρώτη διάσπαση παράγει δύο στιγμιότυπα μισού περίπου μεγέθους **n/2** από το αρχικό. Το ίδιο συμβαίνει και σε κάθε επόμενη διάσπαση.
  - Η δυαδική αναζήτηση *εφαρμόζει* την τεχνική της εξισορρόπησης !

  Αυτό την κάνει να είναι αποτελεσματικότερη από τη σειριακή αναζήτηση !!!

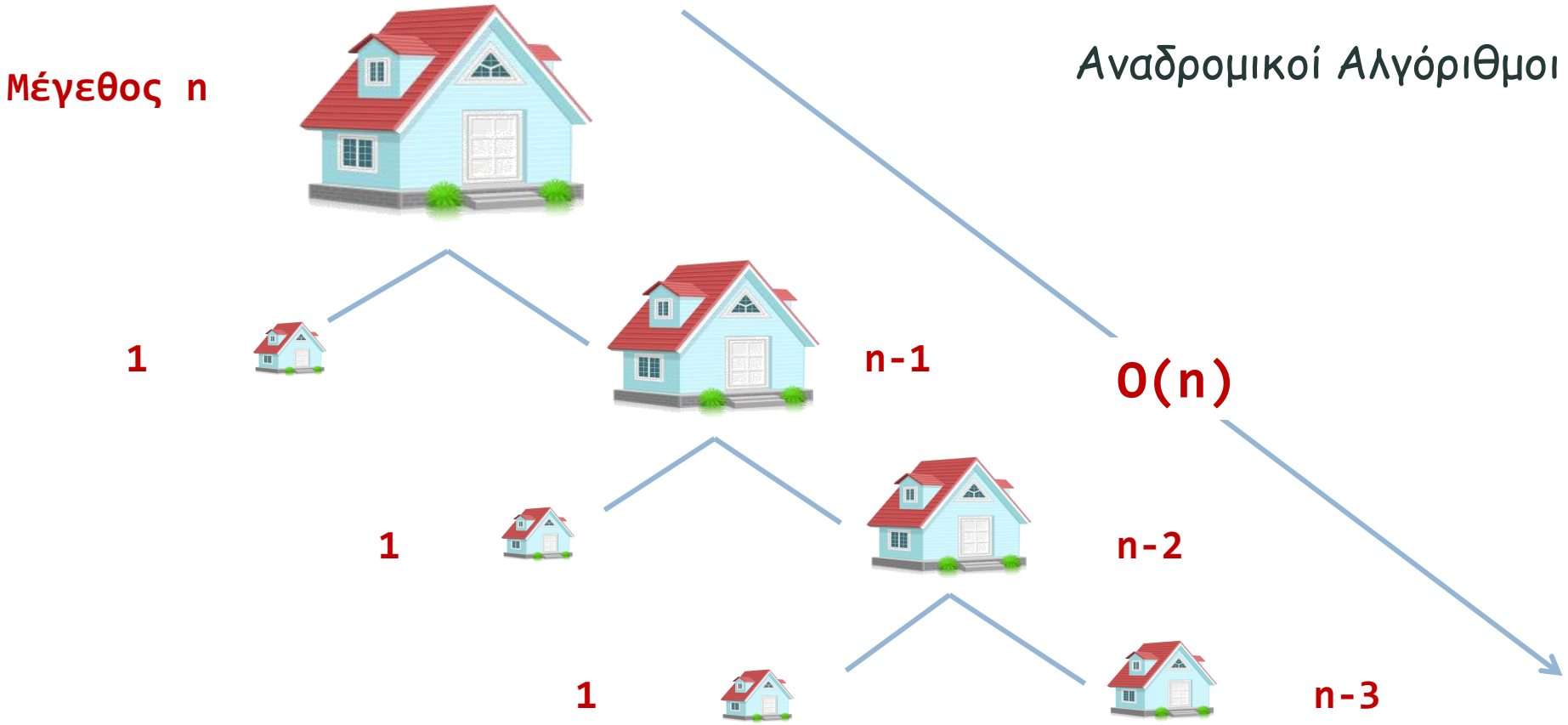


Εξισορρόπηση - Παράδειγμα

Αναδρομικοί Αλγόριθμοι !!!

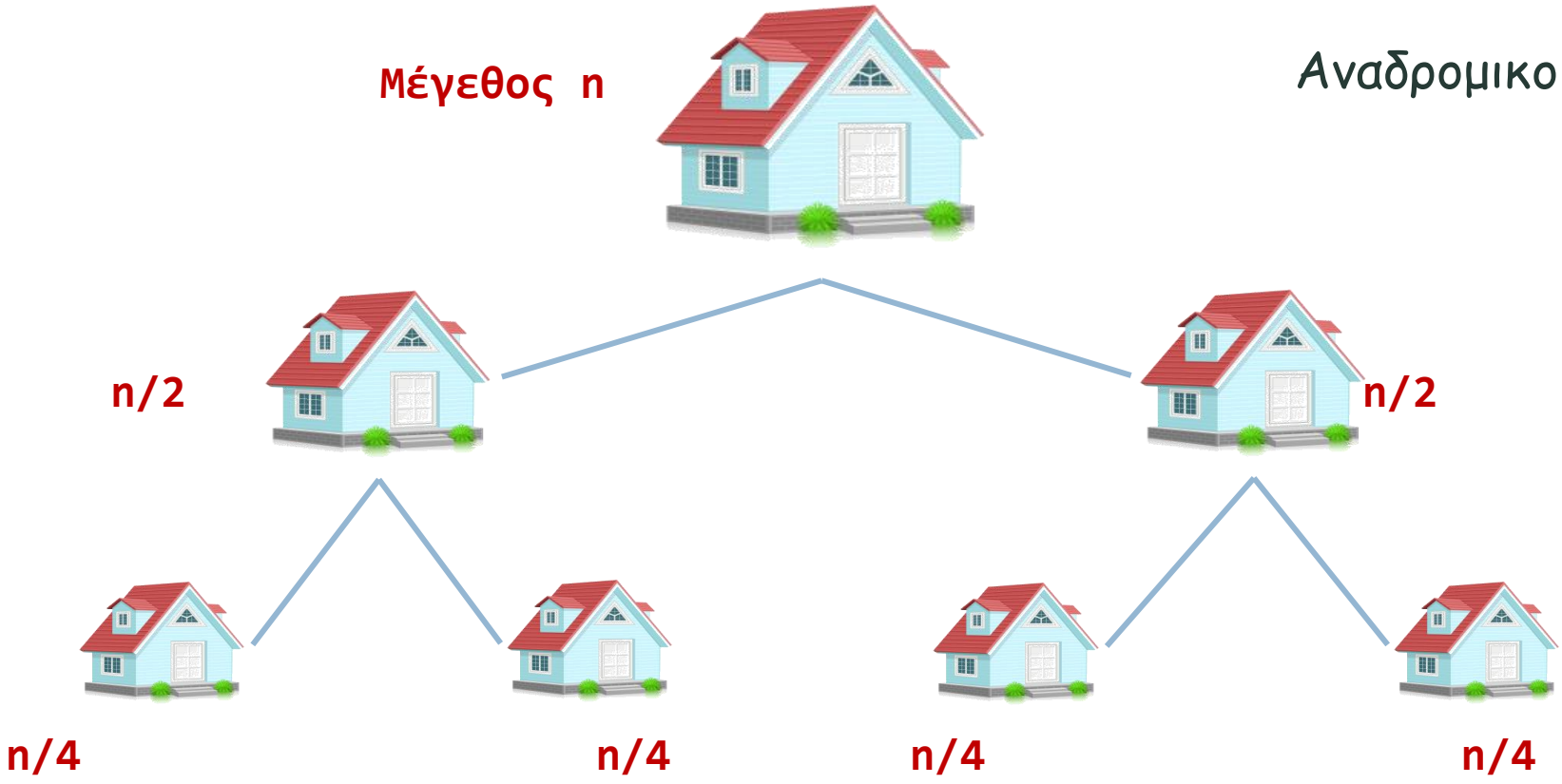
Εξισορρόπηση - Παράδειγμα

Αναδρομικοί Αλγόριθμοι !!!



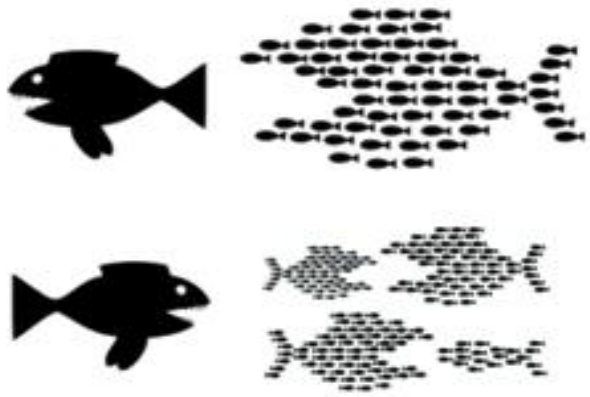
Εξισορρόπηση - Παράδειγμα

Αναδρομικοί Αλγόριθμοι !!!



Τεχνική Διαίρει-και-Βασίλευε

Divide & Conquer



Τεχνική Διαίρει-και-Βασίλευε

Αρκετά παλιά ιδέα ...

Divide et impera. Veni, vidi, vici.

Διαίρει και βασίλευε. Ήλθον, είδον, νίκησον.
Ιούλιος Καίσαρας (100 - 44 πΧ)

Προσφιλής πολεμική του τακτική:

διαίρεση του αντίπαλου στρατεύματος σε δύο μισά και επίθεση στο ένα μισό με ολόκληρο τον στρατό του !!!

Γενικός Κανόνας της Τεχνικής

Αντί να λύσουμε άμεσα ένα πρόβλημα Π με μεγάλο μέγεθος εισόδου, κάνουμε τα εξής:

- διασπάμε την είσοδο σε τμήματα μικρότερου μεγέθους,
- λύνουμε το πρόβλημα για κάθε μικρό τμήμα ξεχωριστά, και κατόπιν
- συνδυάζουμε τις επιμέρους λύσεις, πληρώνοντας ένα επιπλέον κόστος, για να δώσουμε λύση στο αρχικό πρόβλημα.

Μέθοδος που ανάγει ένα αρχικό πρόβλημα σε όλο και μικρότερα προβλήματα με την αναδρομική εφαρμογή τους !!!

Σύμφωνα με την αρχή της εξισορρόπησης τα μικρότερα τμήματα πρέπει να είναι ισομεγέθη !!!

Διαδικασίες της Τεχνικής

Divide and Conquer

1. **Διαίρει** (divide): Διαίρεσε το προβλήματος **Π** σε **k** (συνήθως k=2) υποπροβλήματα $\Pi_1, \Pi_2, \ldots, \Pi_k$ (στιγμιότυπα ίδιου τύπου με το **Π**, όσο δυνατόν ιδίου μεγέθους ή δυσκολίας).

2. **Κατάκτησε** (conquer): Αναδρομική επίλυση όλων των υποπροβλημάτων $\Pi_1, \Pi_2, \ldots, \Pi_k$. Εάν τα υποπροβλήματα είναι μικρού μεγέθους, απλώς επίλυσέ τα άμεσα (direct_solve).

3. **Συνδύασε** (combine): Κατάλληλος συνδυασμός των επιμέρους λύσεων των υποπροβλημάτων $\Pi_1, \Pi_2, \ldots, \Pi_k$ για τη λύση του αρχικού προβλήματος **Π**.

Χαρακτηριστικά της Τεχνικής

Divide and Conquer

1. **Διαίρει (divide)**
2. **Επίλυσε (direct_solve)**
3. **Συνδύασε (combine)**

Συνήθως, σε έναν αλγόριθμο Δ-και-Β είναι **εύκολη**:

- ✓ είτε η διαδικασία **divide**
- ✓ είτε η διαδικασία **combine**

Η διαδικασία **direct_solve** είναι πάντα εύκολη, απαιτεί απλούς υπολογισμούς !!!

Αναδρομική Σχέση

- Έστω **k** το πλήθος των υπο-προβλημάτων Π_1, Π_2, …, Π_k της αρχικής εισόδου (προβλήματος) μήκους **n**, και έστω:

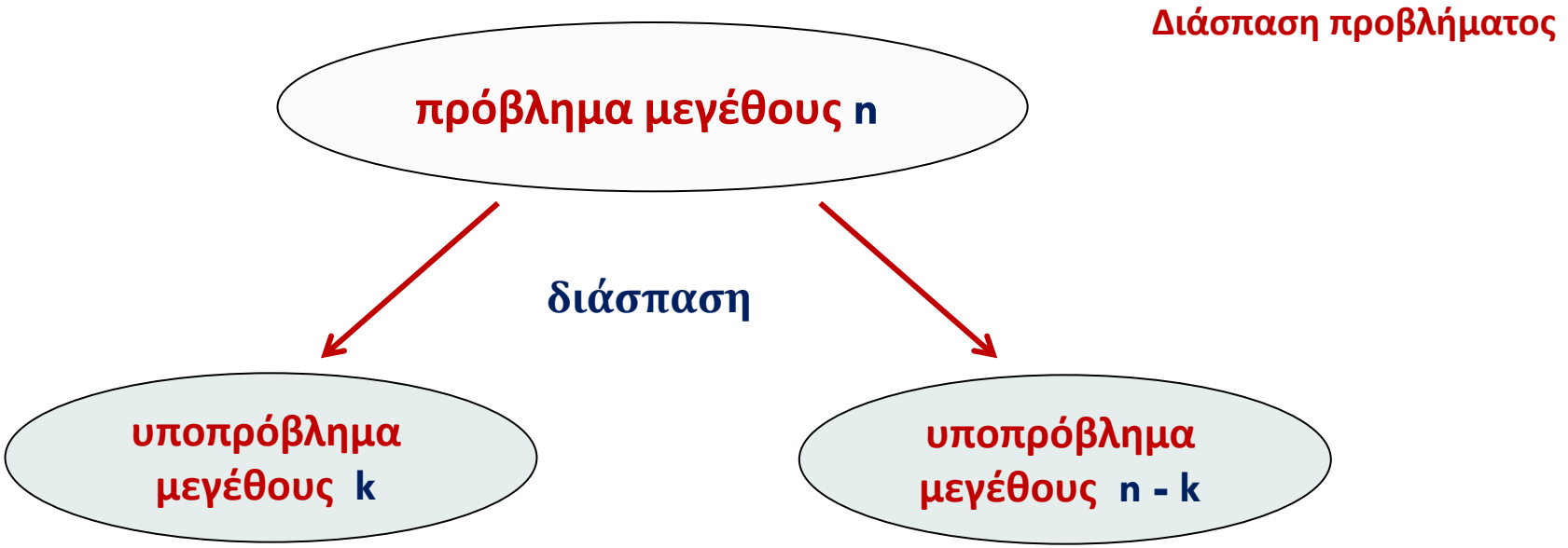
$D(n)$	το πλήθος των βημάτων της	`divide`
$S(n)$	>> >>	`direct_solve`
$C(n)$	>> >>	`combine`

Τότε

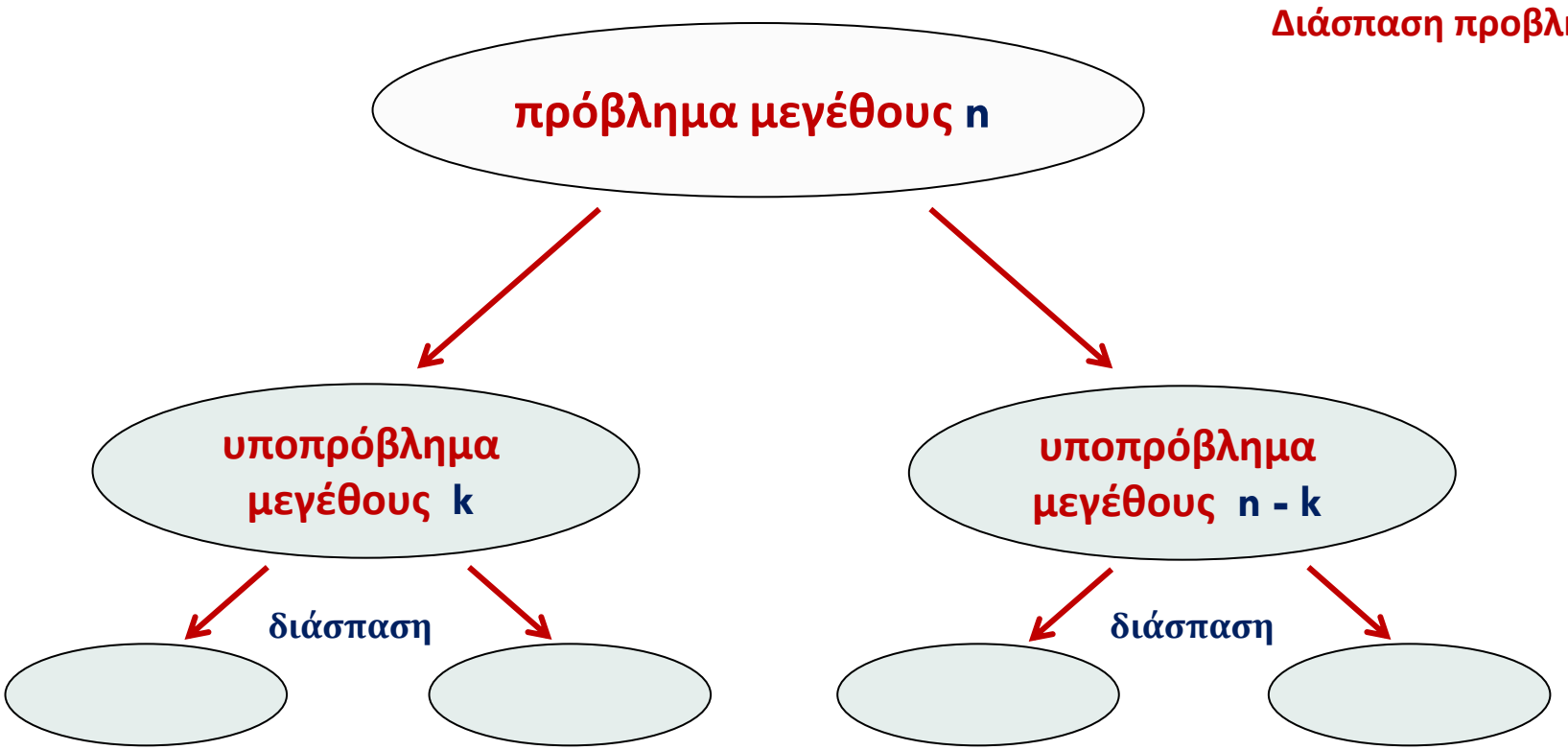
$$T(n) = D(n) + \sum_{i=1}^{k} S(n_i) + C(n)$$

$$\text{for } n > n_i = \text{smallsize}(I_i)$$

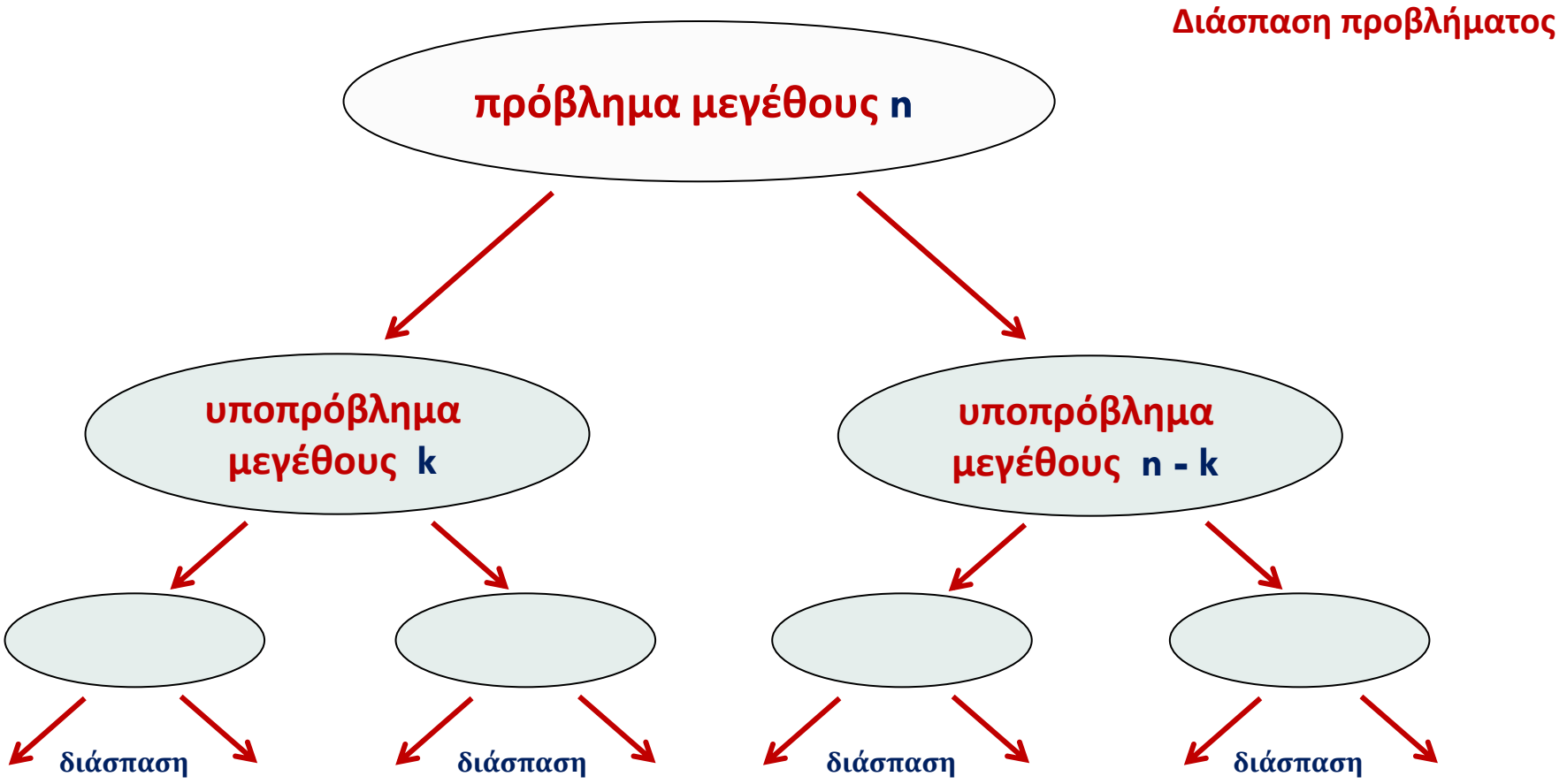
Αναδρομικό Σχήμα



Αναδρομικό Σχήμα



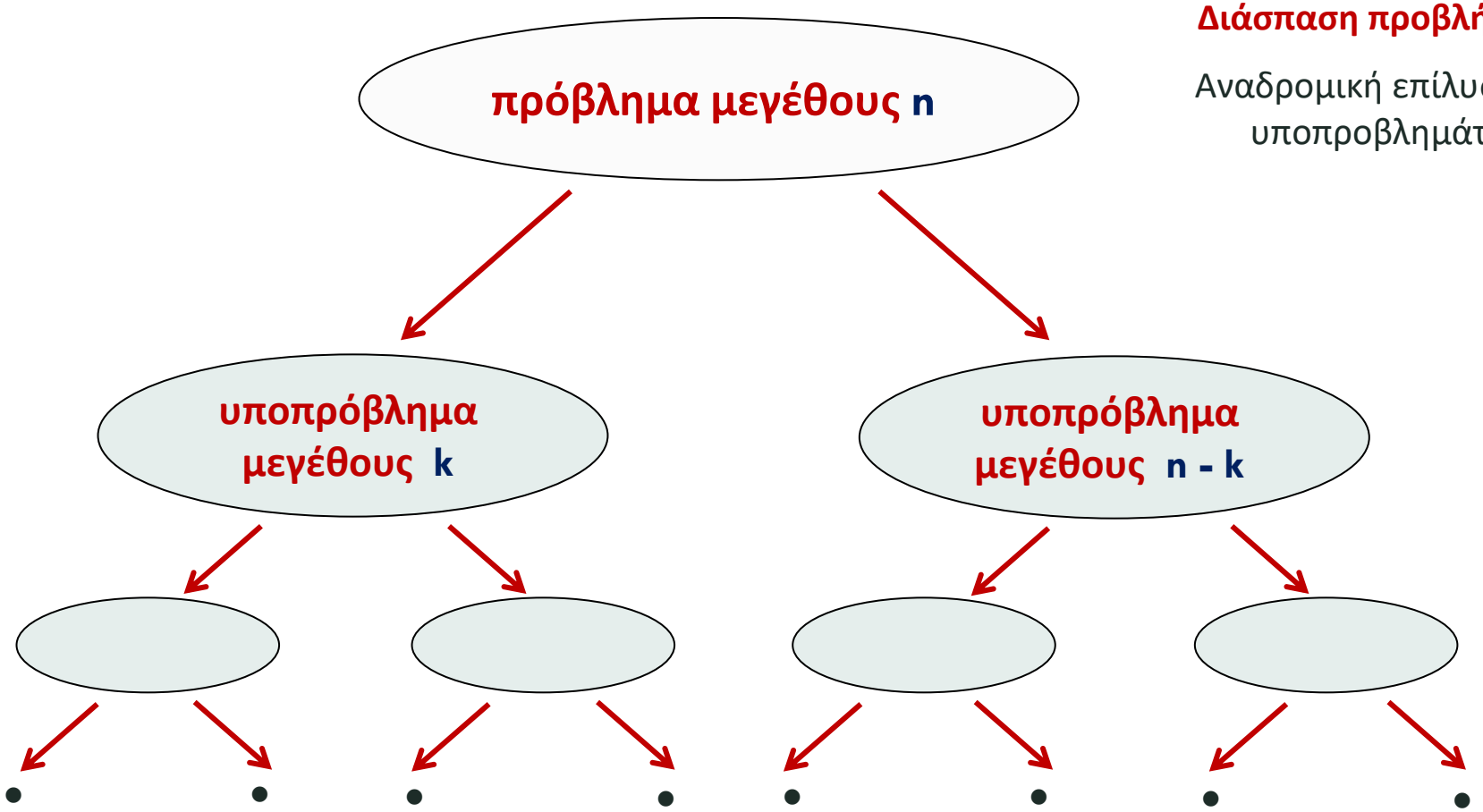
Αναδρομικό Σχήμα



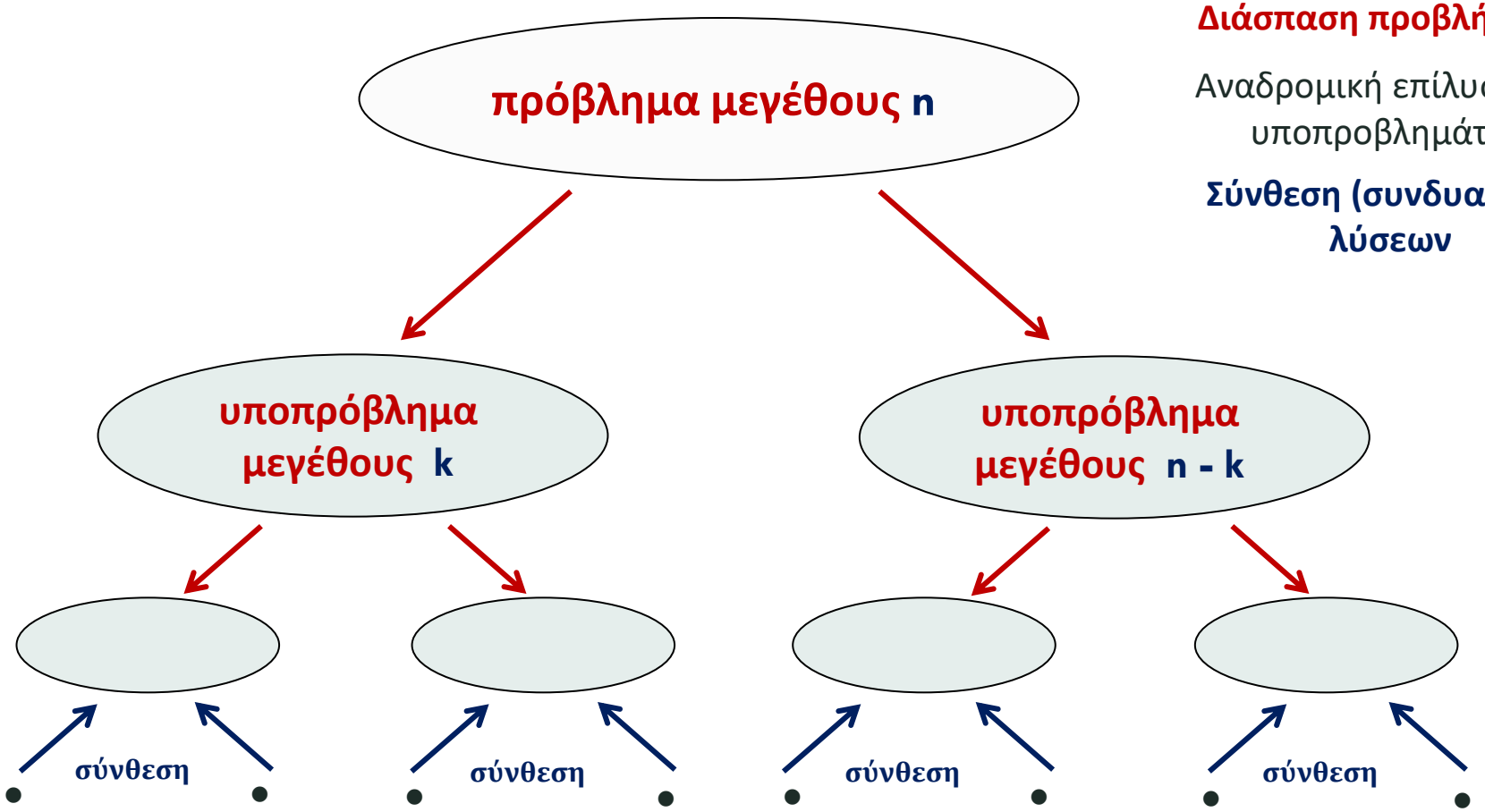
Αναδρομικό Σχήμα

Διάσπαση προβλήματος

Αναδρομική επίλυση των υποπροβλημάτων



Αναδρομικό Σχήμα

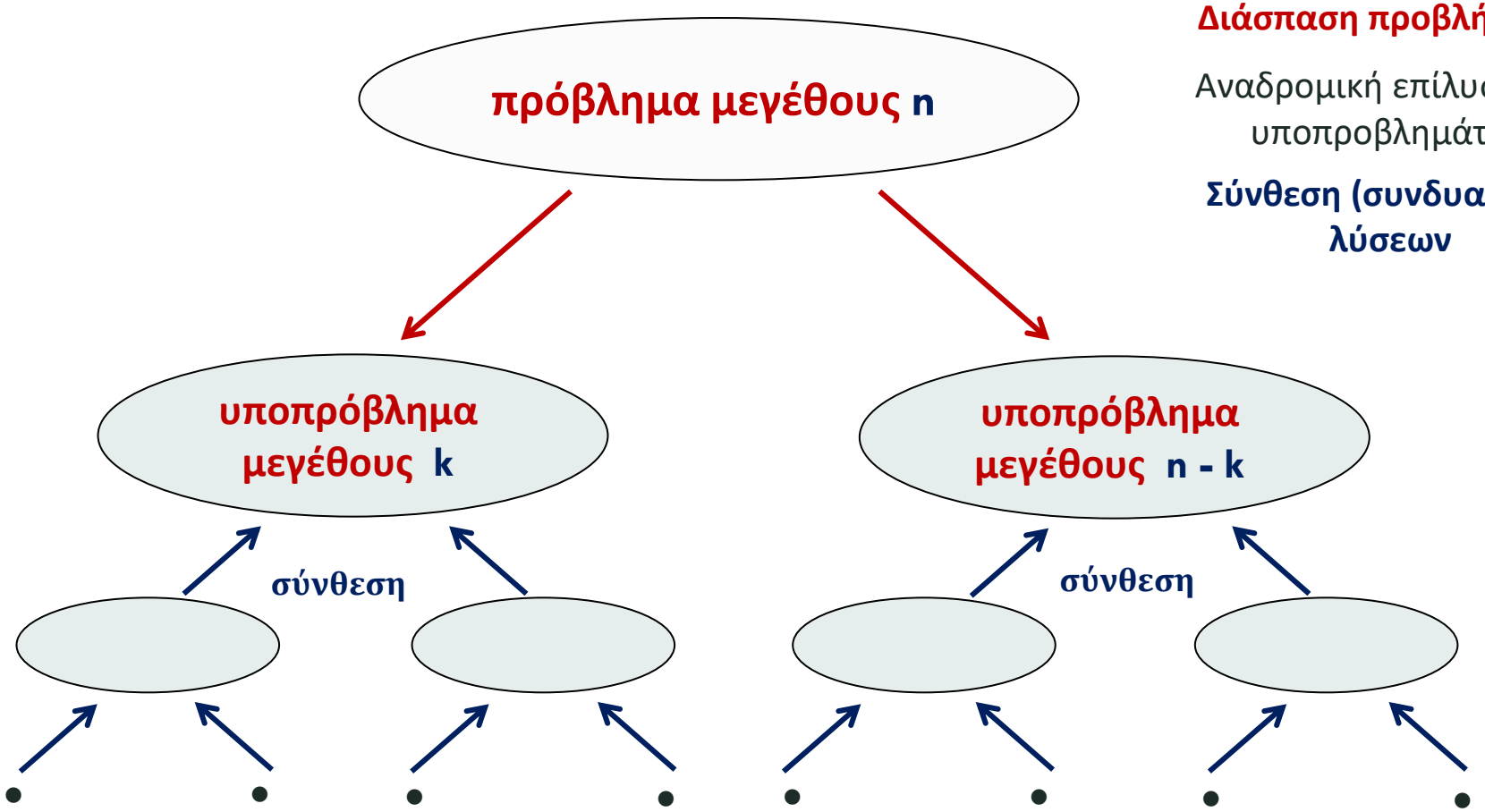


Διάσπαση προβλήματος

Αναδρομική επίλυση των υποπροβλημάτων

Σύνθεση (συνδυασμός) λύσεων

Αναδρομικό Σχήμα

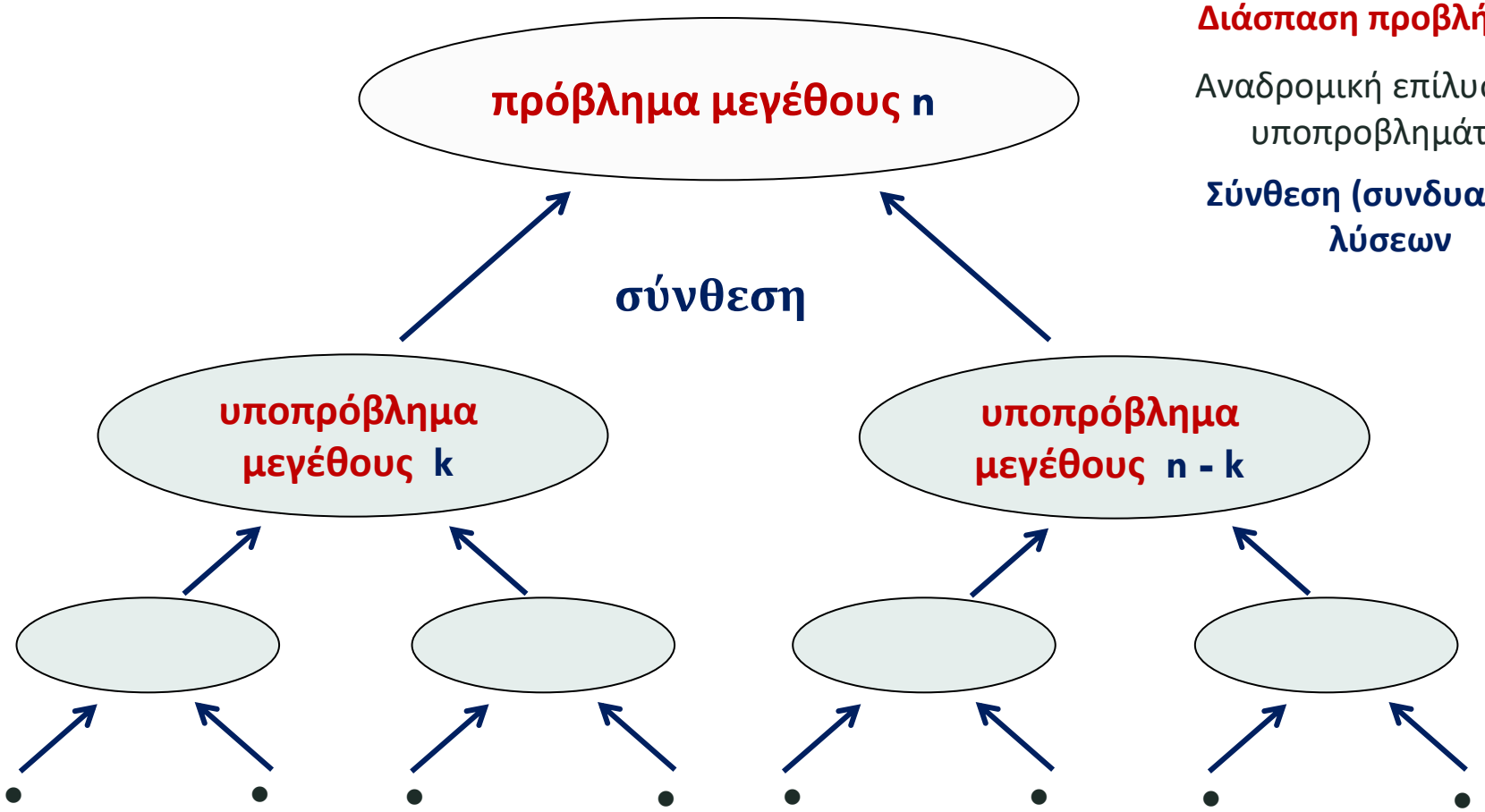


Διάσπαση προβλήματος

Αναδρομική επίλυση των υποπροβλημάτων

Σύνθεση (συνδυασμός) λύσεων

Αναδρομικό Σχήμα

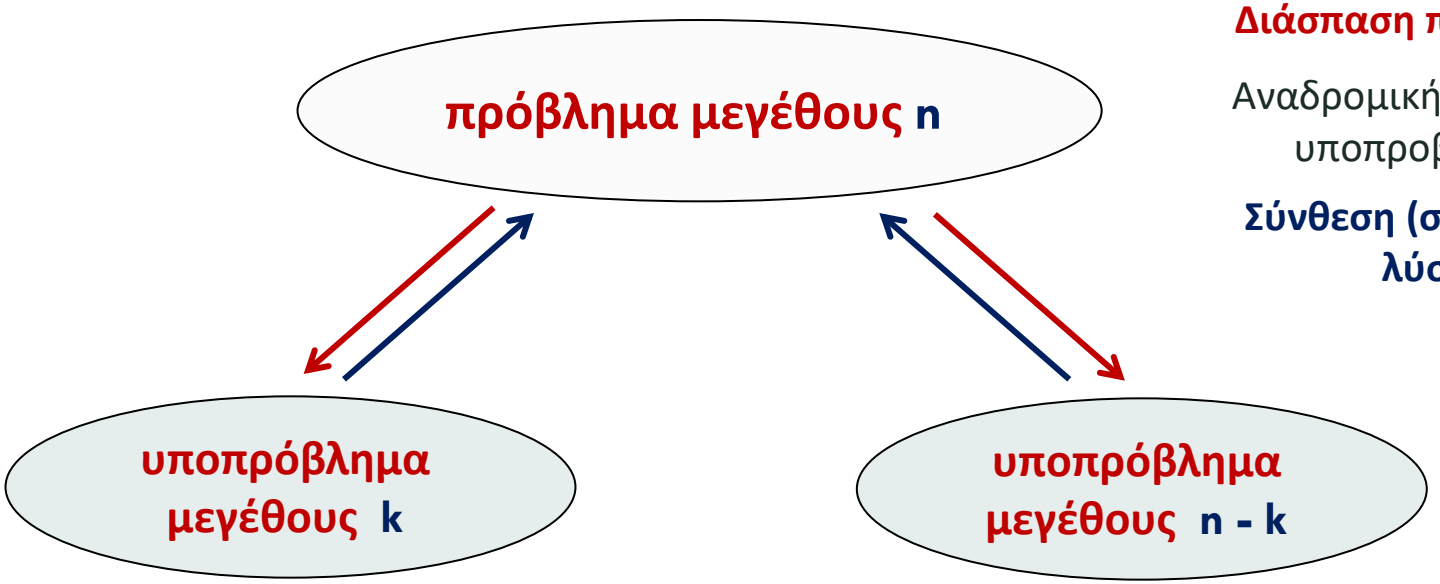


Διάσπαση προβλήματος

Αναδρομική επίλυση των υποπροβλημάτων

Σύνθεση (συνδυασμός) λύσεων

Αναδρομικό Σχήμα



Διάσπαση προβλήματος

Αναδρομική επίλυση των υποπροβλημάτων

Σύνθεση (συνδυασμός) λύσεων

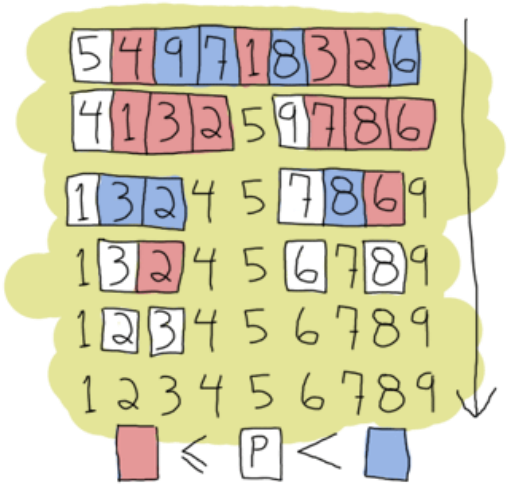
$$T(n) = T(k) + T(n-k) + D(n) + C(n)$$

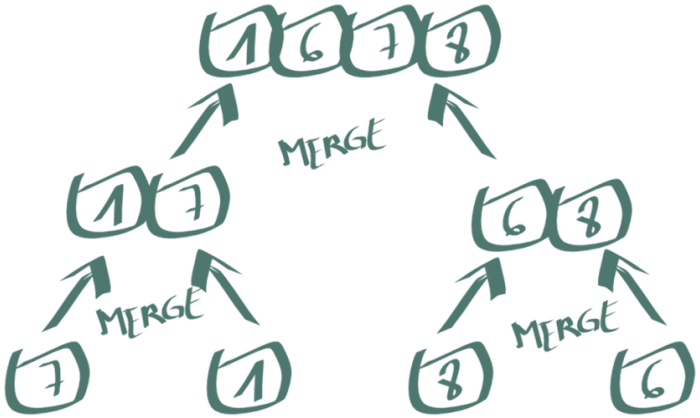
Χρόνος εκτέλεσης

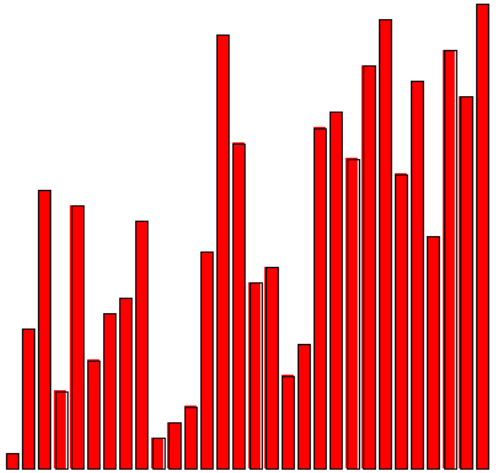
Χρόνος διάσπασης

Χρόνος σύνθεσης

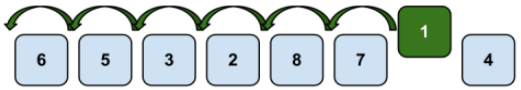
Το Πρόβλημα της Ταξινόμησης







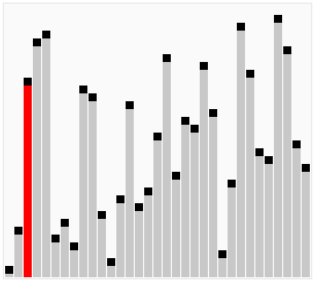
Το Πρόβλημα της Ταξινόμησης



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).

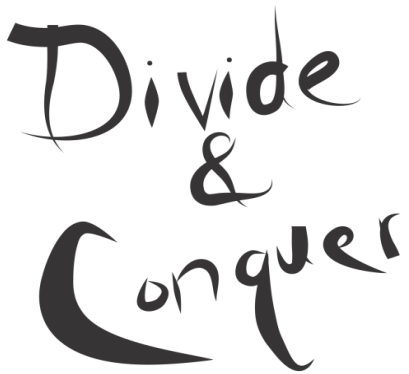
Ταχύτατη αναζήτηση σε ταξινομημένα δεδομένα.

Σημαντικές αλγοριθμικές ιδέες και τεχνικές !!!

Αλγόριθμοι και Μέθοδοι Ταξινόμησης

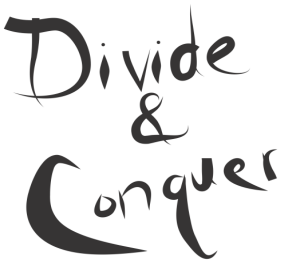
Αντιμετάθεση κάθε ζεύγους μη-ταξινομημένων στοιχείων.	Bubble-sort
Εισαγωγή στοιχείου σε κατάλληλη θέση σε έναν ταξινομημένο πίνακα.	Insertion-sort
Επιλογή μεγαλύτερου (μικρότερου) στοιχείου και τοποθέτηση στο τέλος (αρχή).	Max(Min)-sort Heap-sort
Διαμέριση σε μικρότερα και μεγαλύτερα από στοιχείο-διαχωρισμού και ταξινόμηση.	Quick-sort
Συγχώνευση ταξινομημένων ακολουθιών: Ισομερή διαίρεση, ταξινόμηση, συγχώνευση.	Merge-sort

Αλγόριθμοι Quick-sort και Merge-sort



- Διαμέριση σε μικρότερα και μεγαλύτερα από στοιχείο-διαχωρισμού και ταξινόμηση. **Quick-sort**
- Συγχώνευση ταξινομημένων ακολουθιών: Ισομερή διαίρεση, ταξινόμηση, συγχώνευση. **Merge-sort**

Αλγόριθμοι Quick-sort και Merge-sort



- Δύο ταχύτατοι (βέλτιστοι?) αλγόριθμοι ταξινόμησης
  οι οποίοι βασίζονται στην αλγοριθμική τεχνική του Δ-και-Β!!!

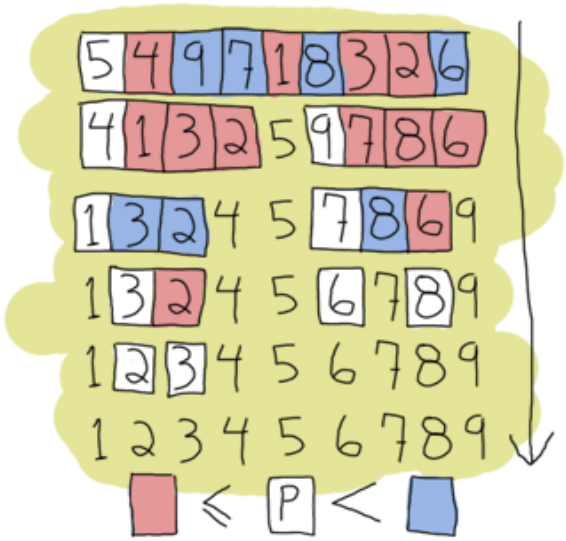
- Βασικές Διαφορές

(I)	Quick-sort	δύσκολη Διαίρεση	*εύκολη* Συνένωση
	Merge-sort	*εύκολη* Διαίρεση	δύσκολη Συνένωση

(II)	Quick-sort	*διαφορετικού* μήκους υπο-προβλήματα
	Merge-sort	*ίδιου* περίπου μήκους υπο-προβλήματα

Αλγόριθμος Ταξινόμησης Quick-Sort

Divide & Conquer

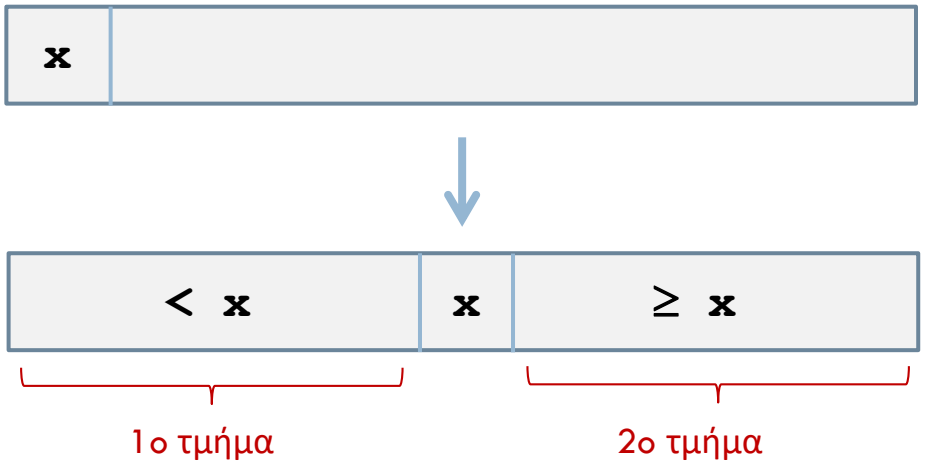


- ✓ Τεχνική **Διαίρει-και-Βασίλευε**
- ✓ **Δύσκολη διαίρεση** - ***Εύκολη*** **συνένωση**
- ✓ Υπο-προβλήματα ***Διαφορετικού*** μήκους
- ✓ Χειρίστη Πολυπλοκότητα $\mathbf{O(n^2)}$
- ✓ Μέση Πολυπλοκότητα **O(nlogn)**

Αλγόριθμος Quick-Sort

Ιδέα Αλγόριθμου !!!

`x = pivot`



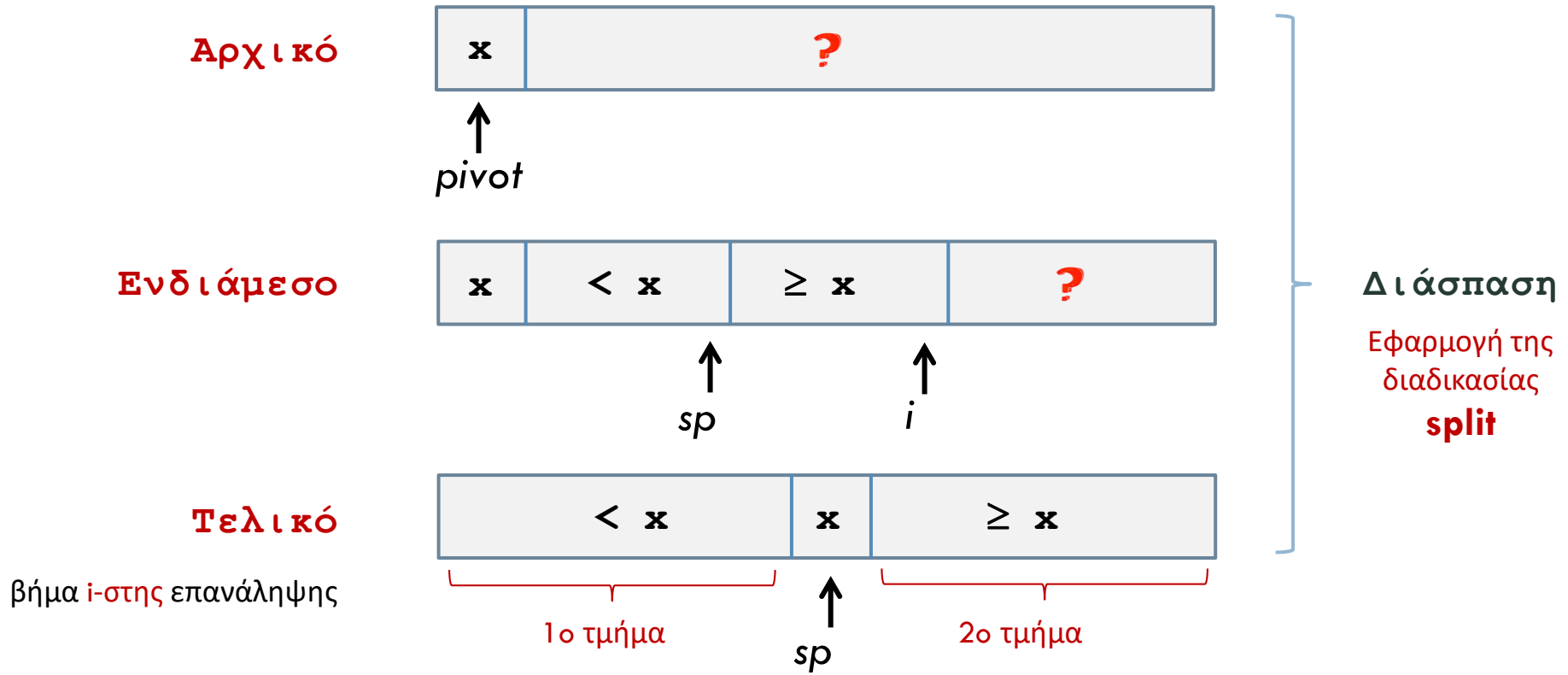
$$T(n) = D(n) + \sum_{i=1}^{2} T(size(I_i)) + C(n)$$

Χρόνος διάσπασης

Χρόνος σύνθεσης $= 0$

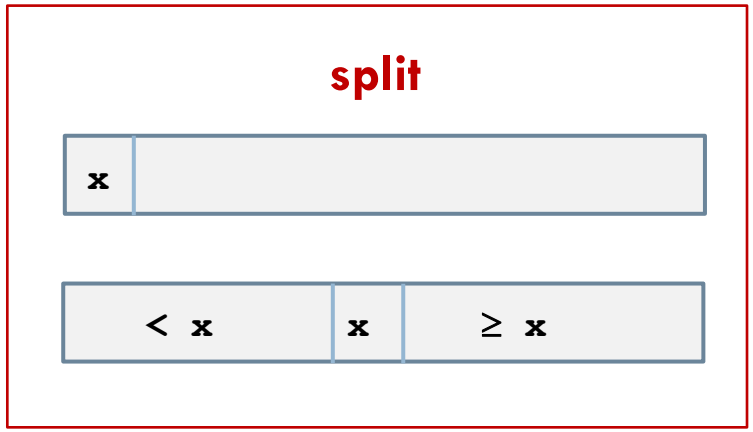
Αλγόριθμος Quick-Sort

Ιδέα Αλγόριθμου !!!

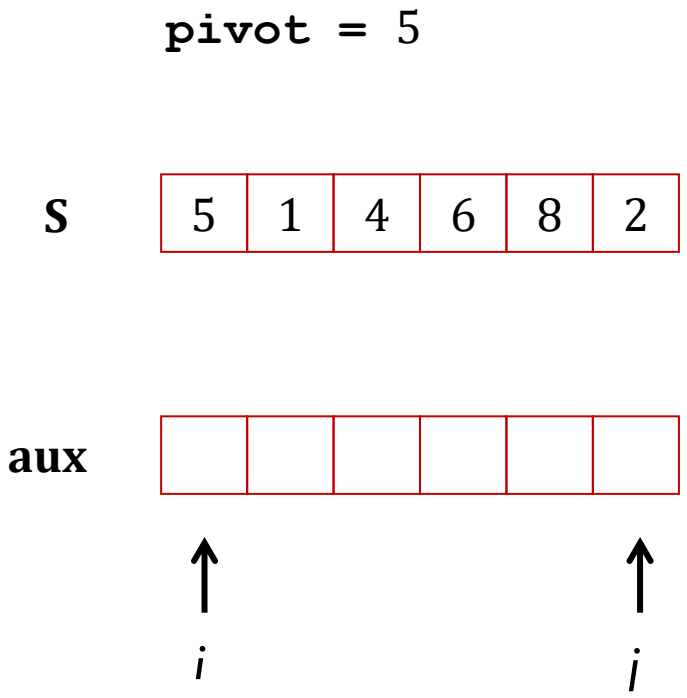


Αλγόριθμος Quick-Sort

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

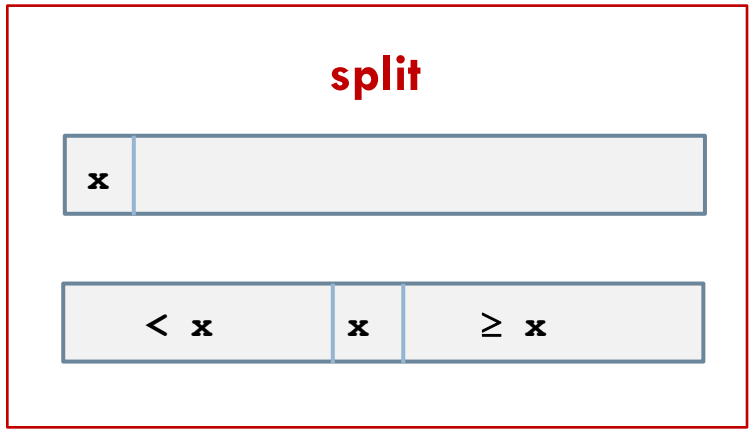



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

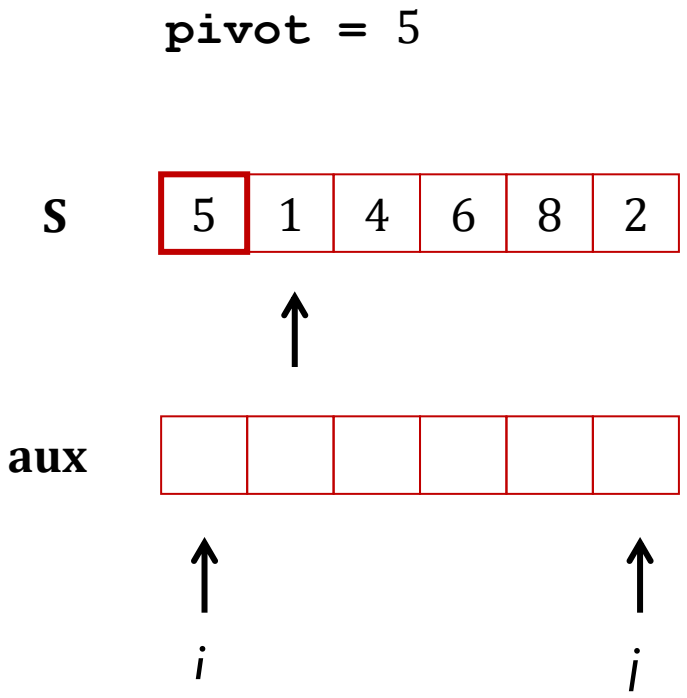


Αλγόριθμος Quick-Sort

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

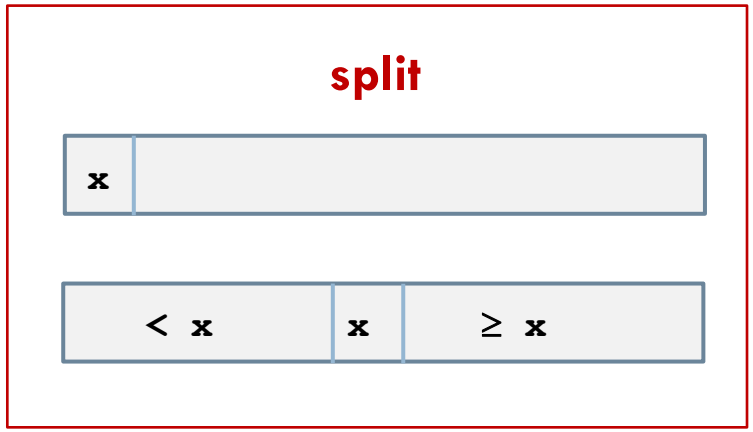



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

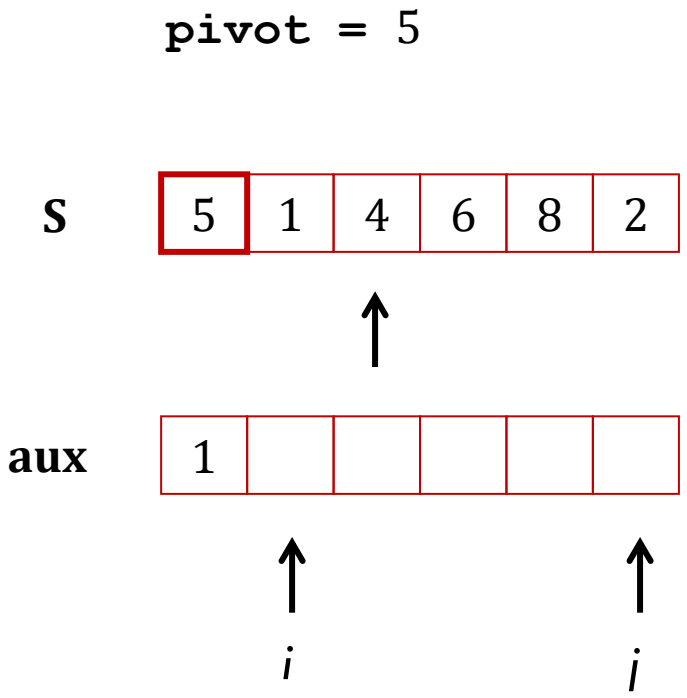


Αλγόριθμος Quick-Sort

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

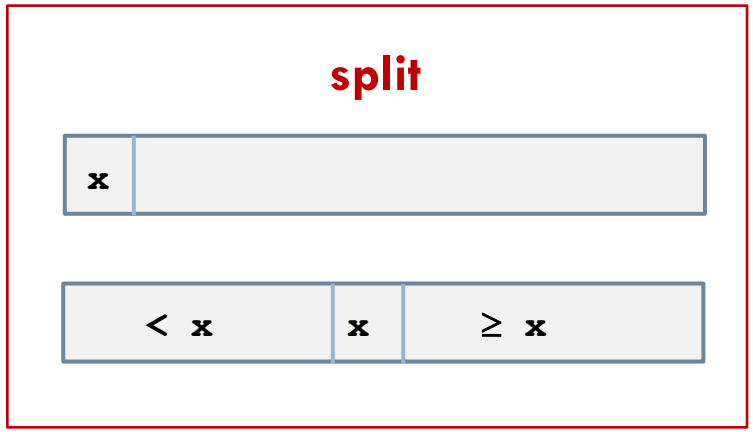



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

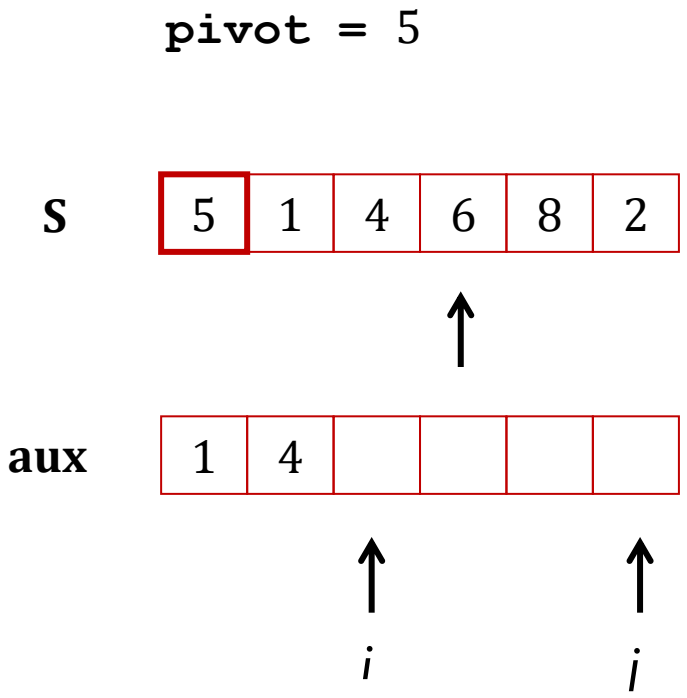


Αλγόριθμος Quick-Sort

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

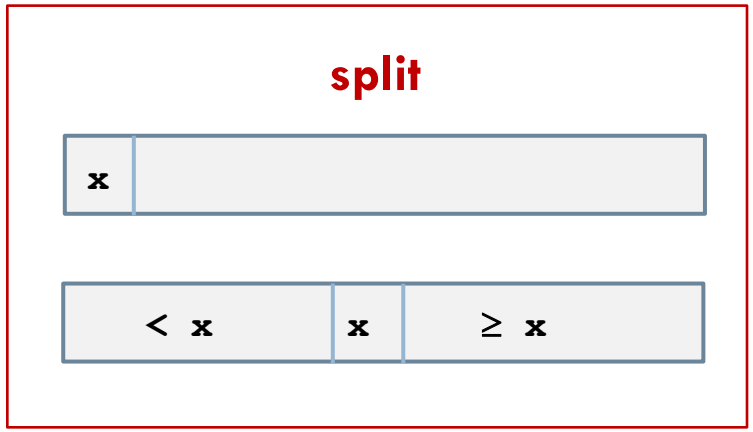



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

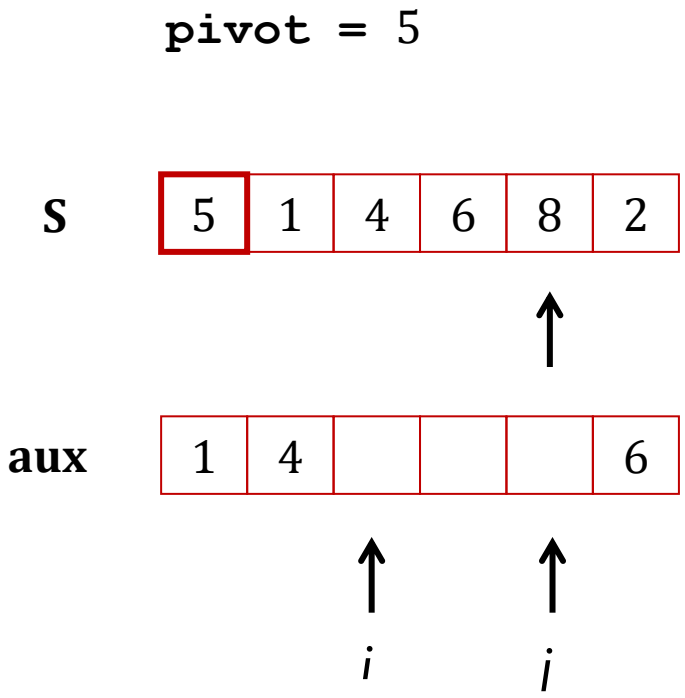


Αλγόριθμος Quick-Sort

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

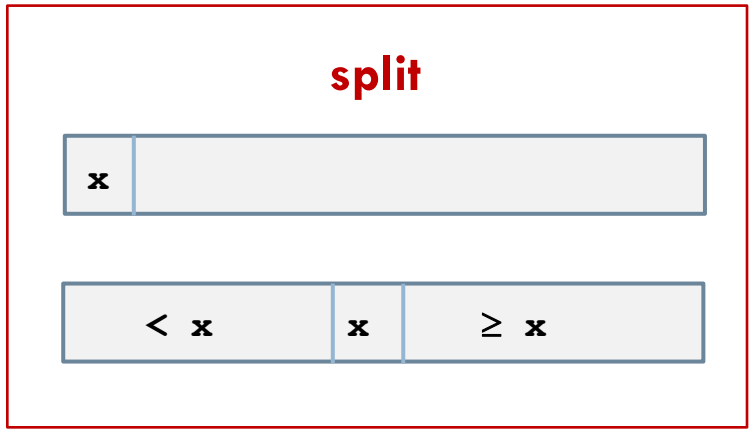



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

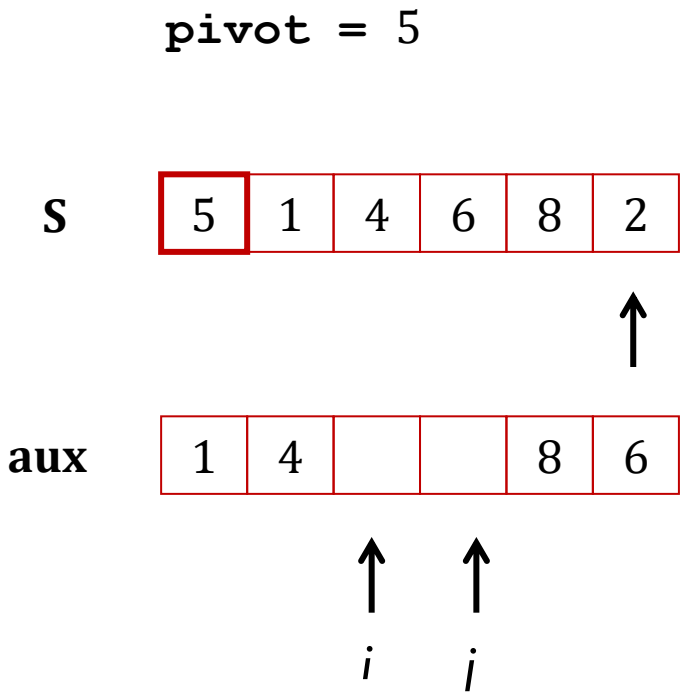


Αλγόριθμος Quick-Sort

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

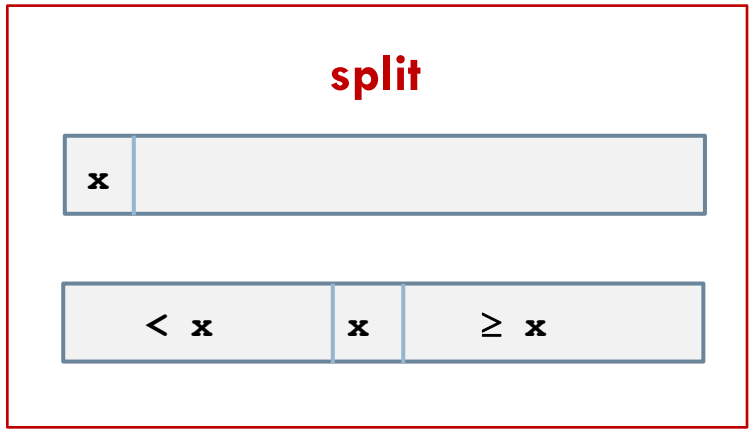



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

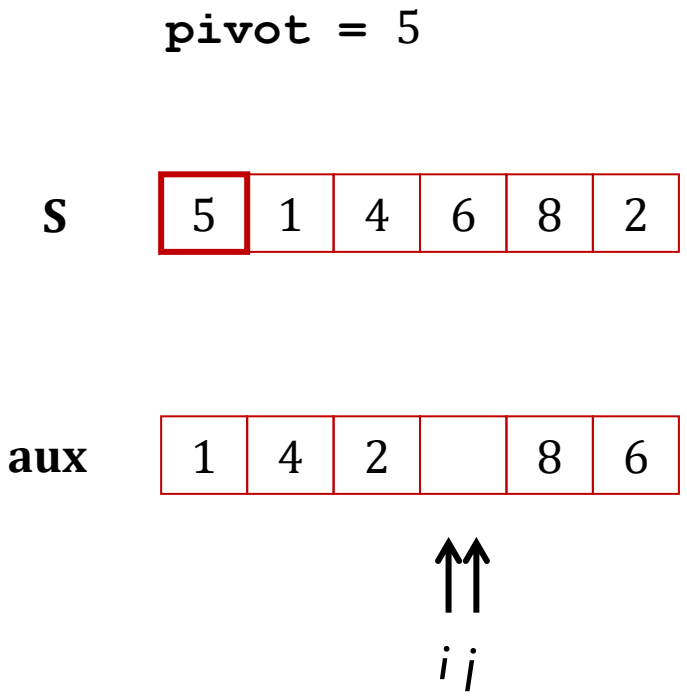


Αλγόριθμος Quick-Sort

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

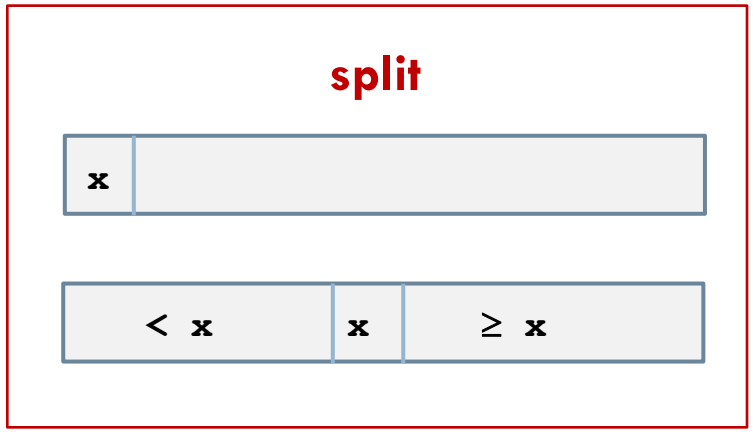



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



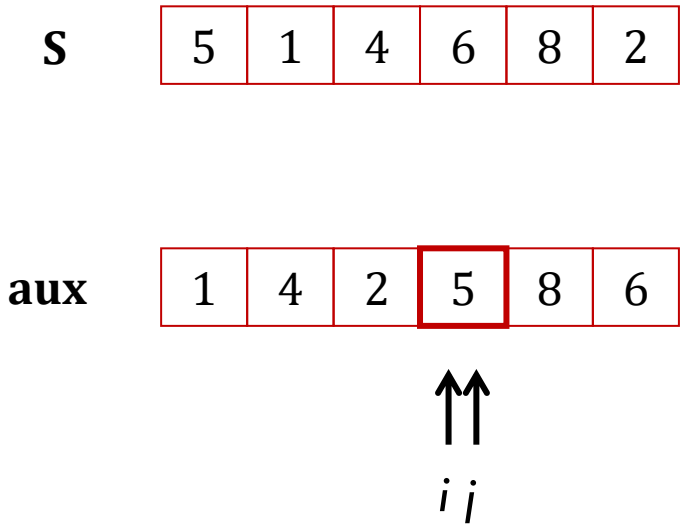
Αλγόριθμος Quick-Sort

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



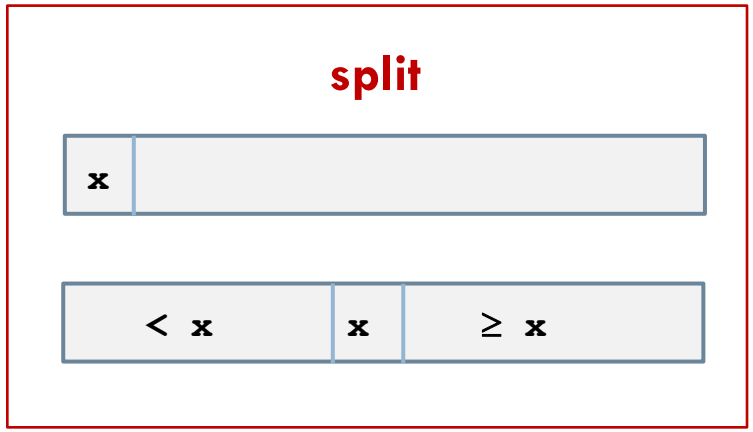

```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

pivot = 5



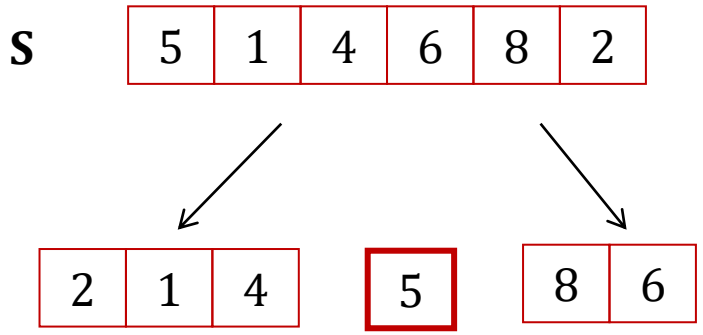
Αλγόριθμος Quick-Sort

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



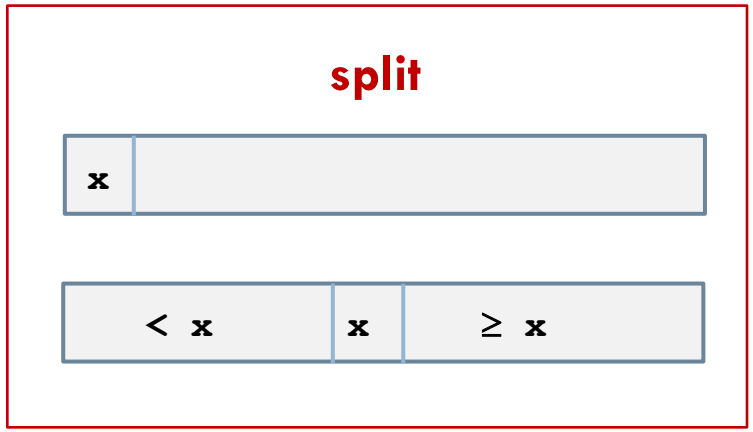

```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

pivot = 5

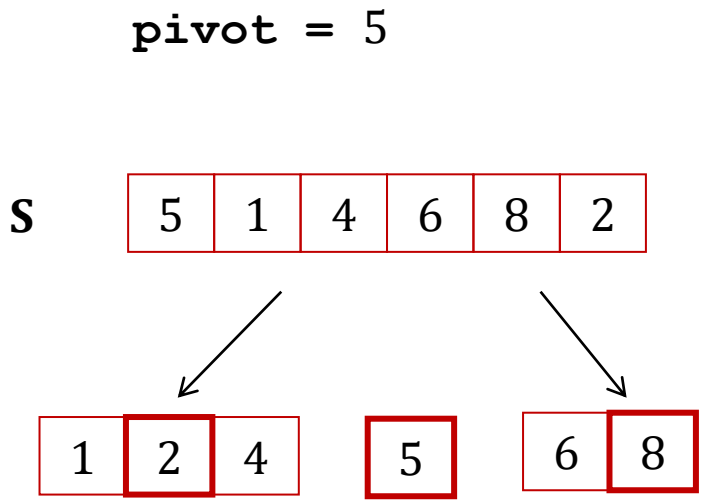


Αλγόριθμος Quick-Sort

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

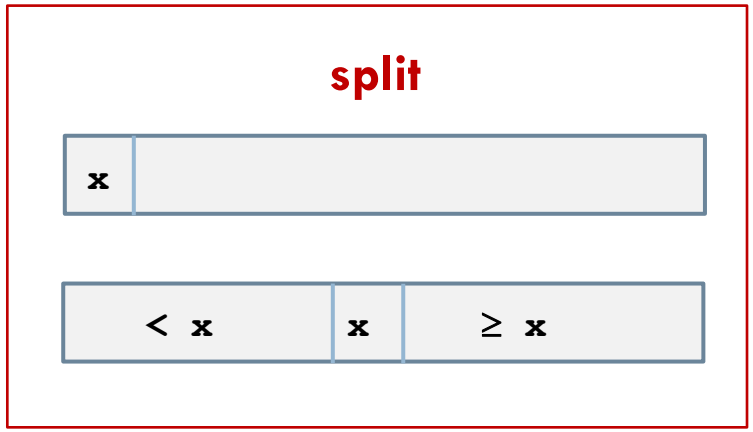



```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```

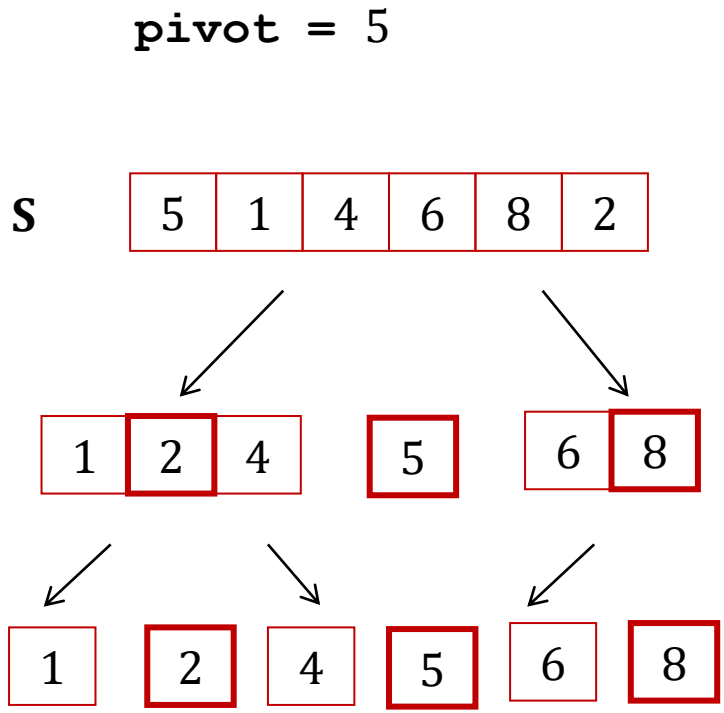


Αλγόριθμος Quick-Sort

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



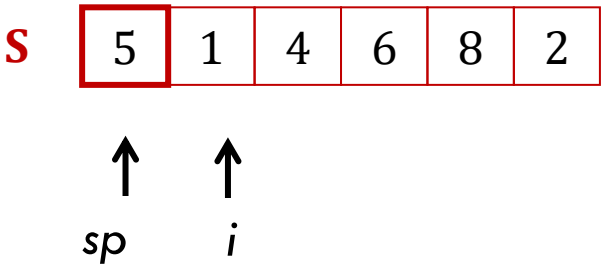

```
   end
(2) Quick-Sort(S, first, splitpt-1)
(3) Quick-Sort(S, splitpt+1, last)
    return S
end
```



Αλγόριθμος Quick-Sort

```
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



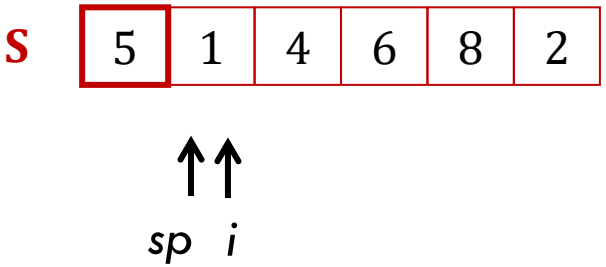
sp : splitpt

Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

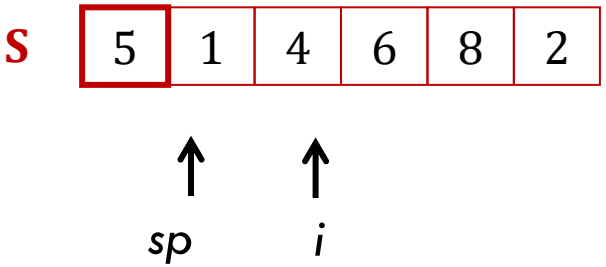


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

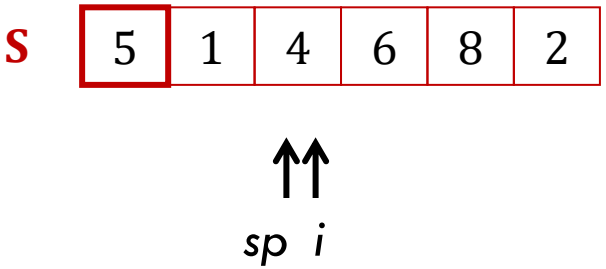


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

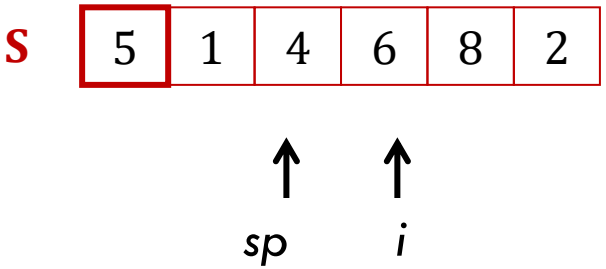


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

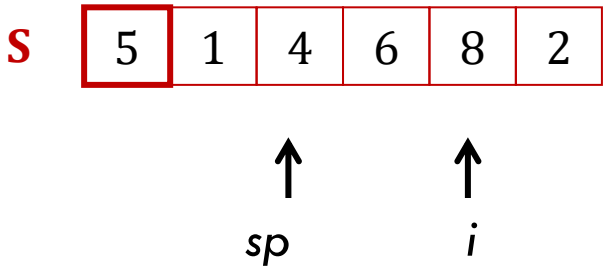


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

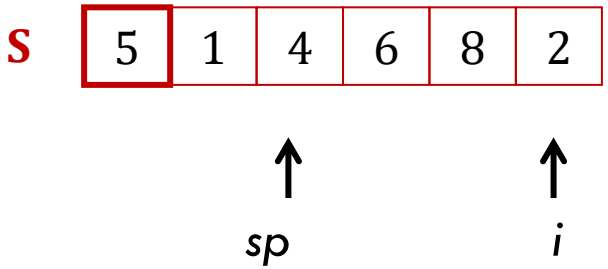


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

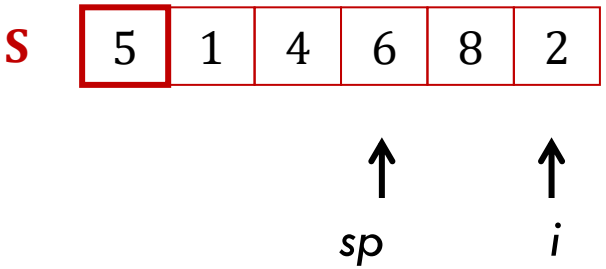


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

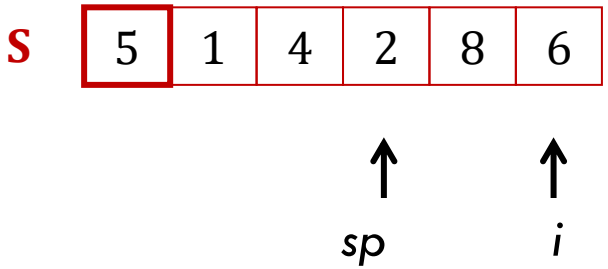


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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

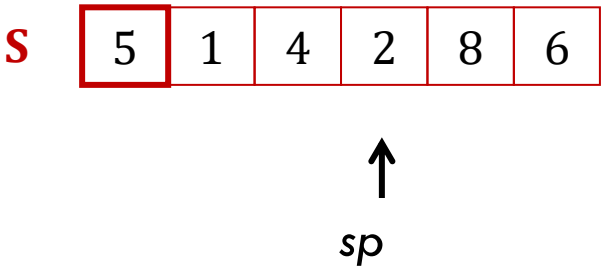


Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

```
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



Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

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

split

```
splitpt ← first
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

Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

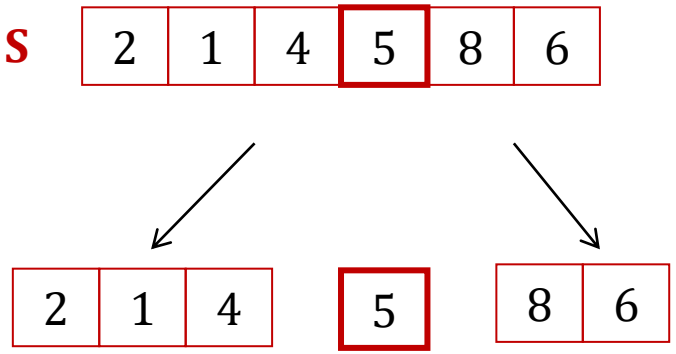
```
Algorithm Quick-Sort(S, first, last)
begin
(1)if first < last then
       pivot ← S[first]
```

split

```
splitpt ← first
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



Split : επί τόπου διάσπαση

Αλγόριθμος Quick-Sort

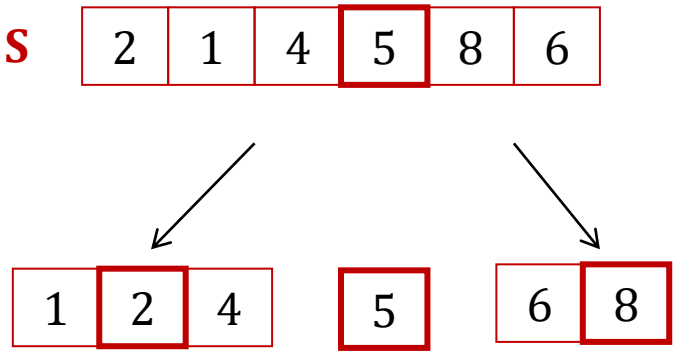
```
Algorithm Quick-Sort(S, first, last)
begin
(1)if first < last then
       pivot ← S[first]
```

split

```
splitpt ← first
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



Split : επιτόπια διάσπαση

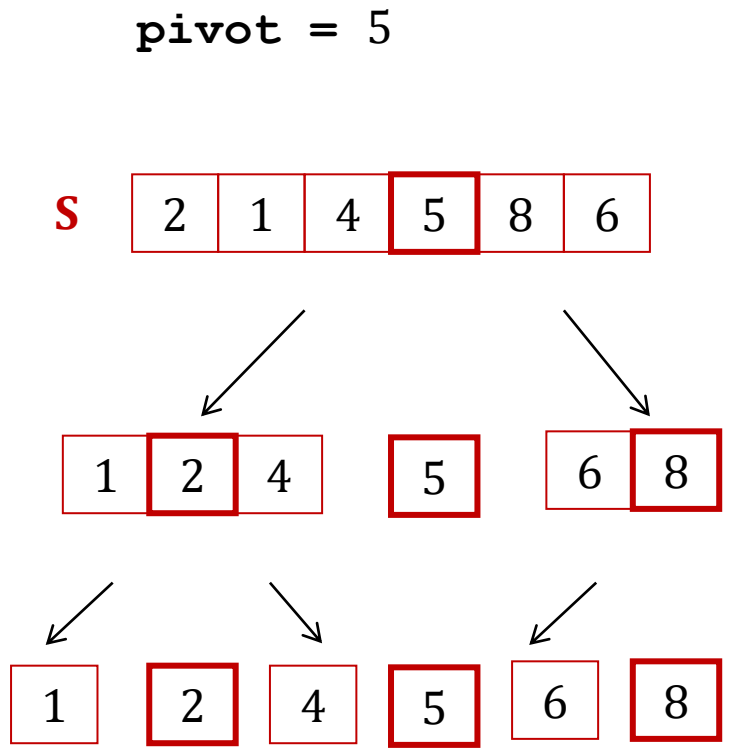
Αλγόριθμος Quick-Sort

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

split

```
splitpt ← first
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 : επιτόπια διάσπαση

Ορθότητα Αλγόριθμου Quick-Sort

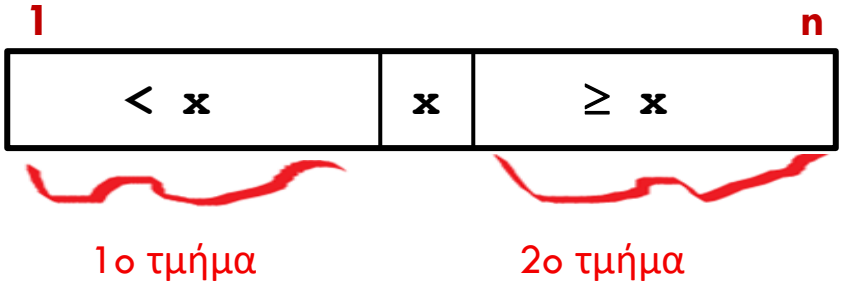
- Απόδειξη με επαγωγή:

  Ισχυρισμός P(n): Ο αλγόριθμος Quick-Sort ταξινομεί σωστά κάθε ακολουθία εισόδου S μήκους n.

  Βάση επαγωγής: Κάθε ακολουθία εισόδου S μήκους 1 είναι ταξινομημένη, δηλ. ισχύει P(1).

  Επαγωγική υπόθεση: Έστω S μήκους $n \geq 2$, για την οποία ισχύει P(k) $\forall$ $k < n$.

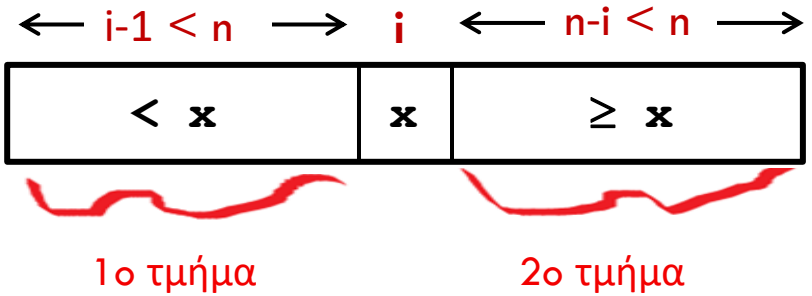
  Επαγωγικό βήμα : Δείχνουμε ότι ισχύει P(n).



Ορθότητα Αλγόριθμου Quick-Sort

- Απόδειξη με επαγωγή:

Θεωρούμε pivot x και τη διαμέριση του S ως προς x (έστω **i** η θέση του pivot, $1 \leq i \leq n$).



Έστω $\mathbf{k_1} = i-1$ το μήκος του 1ου τμήματος των στοιχείων της S που είναι $< x$ και $\mathbf{k_2} = n-i$ του 2ου τμήματος που είναι $\geq x$.

Από την ορθότητα της διαδικασίας split (εύκολη απόδειξη), το στοιχείο pivot x καταλαμβάνει τη σωστή θέση στην ακολουθία εξόδου S.

Από επαγωγική υπόθεση: Ισχύει $P(k_1)$ και $P(k_2)$ επειδή $\mathbf{k_1} < n$ και $\mathbf{k_1} < n$, δηλ., ο αλγόριθμος Quick-Sort ταξινομεί σωστά το 1ο και 2ο τμήμα της S.

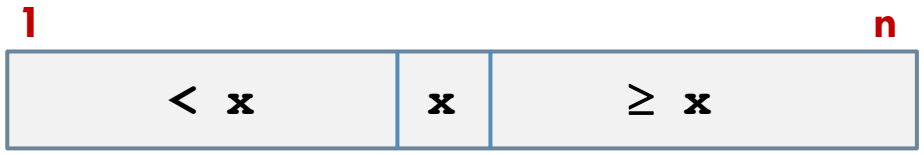
Άρα, ταξινομεί σωστά ολόκληρη την ακολουθία S. QED

Πολυπλοκότητα Αλγόριθμου Quick-Sort

❑ Χειρίστη Πολυπλοκότητα

Στη διαδικασία split το στοιχείο pivot συγκρίνεται με κάθε στοιχείο της ακολουθίας S.

Επομένως, εάν **n** είναι το μήκος της ακολουθίας εισόδου σε κάποια επανάληψη της διαδικασίας split, τότε στη διαδικασία αυτή εκτελούνται **n-1** συγκρίσεις (ΒΠ).



Βασική Πράξη: `S[i] < pivot`

W(n) = **O(n^2)**

```
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])
```

S	5	1	4	6	8	2

n = 6

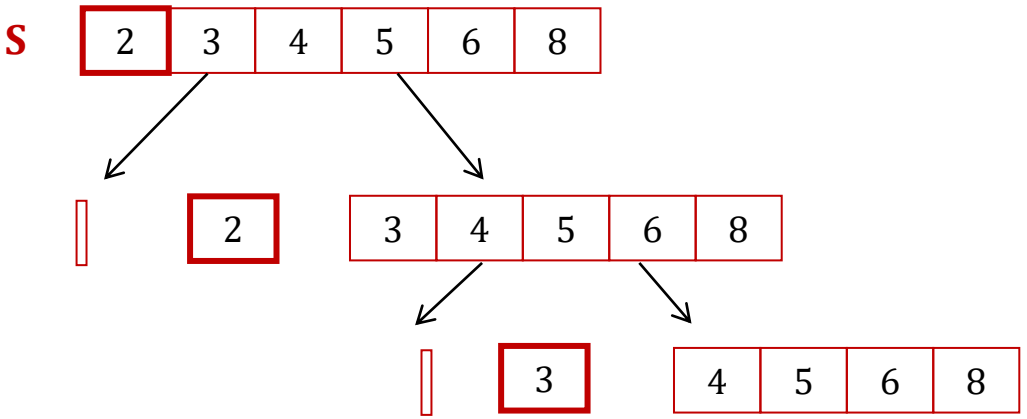
Πλήθος συγκρίσεων = 5

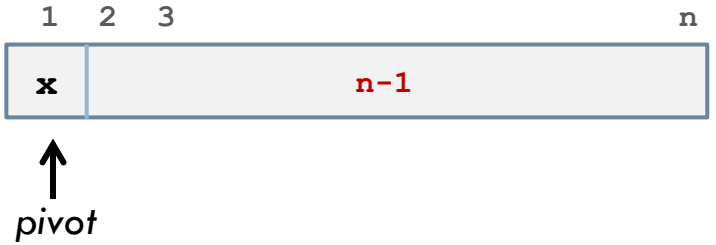
Πολυπλοκότητα Αλγόριθμου Quick-Sort

Χειρίστη Πολυπλοκότητα

Εάν το pivot = S[first] είναι το μικρότερο στοιχείο της ακολουθίας σε κάποια επανάληψη, τότε μετά την εκτέλεση της split θα ισχύει splitpoint = first.

Επομένως, η πρώτη αναδρομική κλήση του Quick-Sort δεν θα εκτελεστεί, ενώ η δεύτερη κλήση θα εκτελεστεί με ακολουθία μήκους n-1, n-2, …, 2.






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

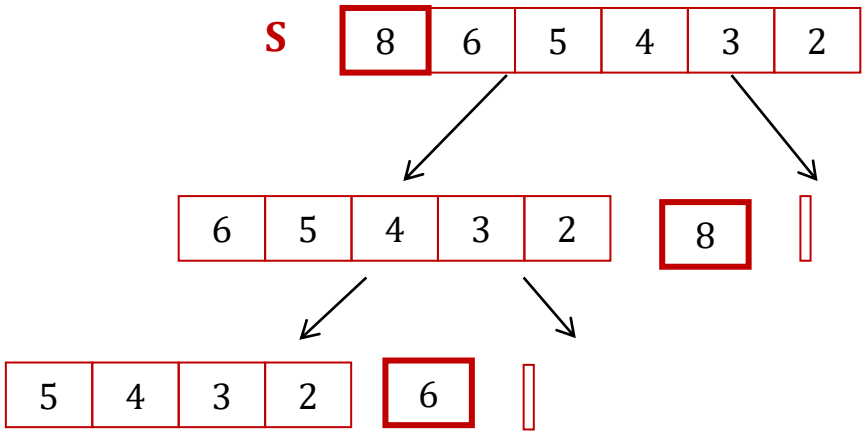
```
Quick-Sort(S, 1, 0)
Quick-Sort(S, 2, n)
```

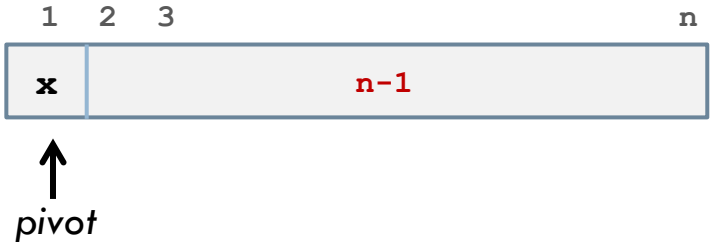
Πολυπλοκότητα Αλγόριθμου Quick-Sort

Χειρίστη Πολυπλοκότητα

Εάν το pivot = S[first] είναι το μεγαλύτερο στοιχείο της ακολουθίας σε κάποια επανάληψη, τότε μετά την εκτέλεση της split θα ισχύει splitpoint = last.

Επομένως, η δεύτερη αναδρομική κλήση του Quick-Sort δεν θα εκτελεστεί, ενώ η πρώτη κλήση θα εκτελεστεί με ακολουθία μήκους n-1, n-2, …, 2.






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

```
Quick-Sort(S, 1, n-1)
Quick-Sort(S, n+1, n)
```

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Χειρίστη Πολυπλοκότητα

Άρα, στην χειρίστη περίπτωσή για την πολυπλοκότητα τον αλγόριθμου Quick-Sort ισχύει:

$$W(n) = \begin{cases} 1 & \text{if } n = 2 \\ (n-1) + W(n-1) & \text{if } n > 2 \end{cases}$$

Επιλύουμε την αναδρομική σχέση:

$$\begin{aligned} W(n) &= (n-1) + W(n-1) \\ &= (n-1) + (n-2) + W(n-2) \\ &\cdot \\ &\cdot \\ &\cdot \\ &= (n-1) + (n-2) + (n-3) + \ldots + 2 + 1 \\ &= n(n-1)/2 \end{aligned}$$

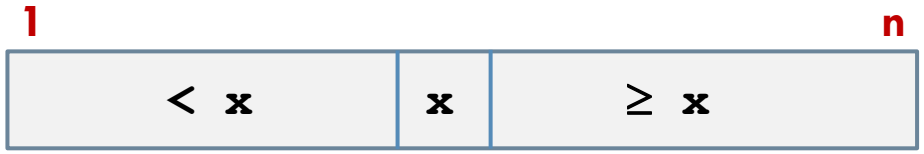
Ισχύει : $W(n) \in \Theta(n^2)$

Πολυπλοκότητα Αλγόριθμου Quick-Sort

❑ Μέση Πολυπλοκότητα

Έστω n το μήκος της ακολουθίας εισόδου σε κάποια εκτέλεση της διαδικασίας split.

Τότε στη διαδικασία αυτή εκτελούνται **n-1 συγκρίσεις** (βασικές πράξεις).



Βασική Πράξη: `S[i] < pivot`

A(n) = **O(n logn)**

Split

```
splitpoint ← first
for i ← first+1 to last do
    if S[i] < pivot then
        splitpoint ← splitpoint + 1
        Swap(S[splitpoint], S[i])
    end;
end;
Swap (S[first], S[splitpoint])
```

S	5	1	4	6	8	2

n = 6

Πλήθος συγκρίσεων = 5

Πολυπλοκότητα Αλγόριθμου Quick-Sort

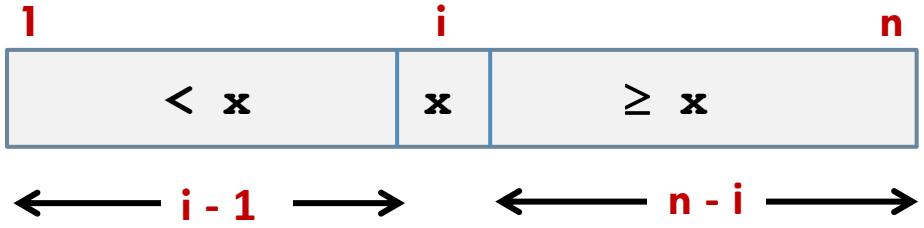
Μέση Πολυπλοκότητα

Θεωρούμε τη γενική περίπτωση κατά την οποία μετά την εκτέλεση της διαδικασία split με μήκος ακολουθίας **n**, το στοιχείο pivot καταλαμβάνει την **i-στη** θέση.

Άρα, **i-1** στοιχεία είναι μικρότερα του **pivot** και φυσικά **n-i** μεγαλύτερα από αυτό.

Δεχόμαστε ότι το στοιχείο **pivot** έχει την ίδια πιθανότητα **1/n** να καταλάβει οποιαδήποτε θέση μεταξύ **1** και **n**:

το **pivot** καταλαμβάνει τη θέση **i** με πιθανότητα **1/n**.



Μήκος 1ου τμήματος	Θέση pivot	Μήκος 2ου τμήματος
(0)	1	(n-1)
(1)	2	(n-2)
(2)	3	(n-3)
.	.	.
.	.	.
(n-2)	n-1	(1)
(n-1)	n	(0)

Πολυπλοκότητα Αλγόριθμου Quick-Sort

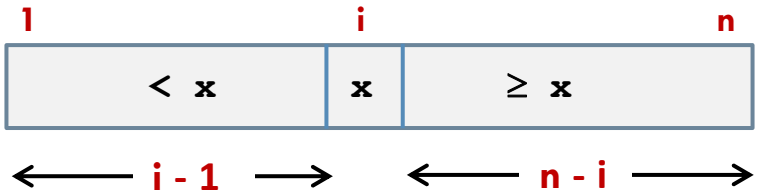
Μέση Πολυπλοκότητα

Ορίζουμε:

A(n) = μέσος όρος συγκρίσεων σε ακολουθία μήκους **n**.

Έτσι, καταλήγουμε στην αναδρομική σχέση:

$$A(n) = \begin{cases} 0 & n < 2 \\ (n-1) + \sum_{i=1}^{n} \frac{1}{n} \cdot (A(i-1) + A(n-i)) & n \geq 2 \end{cases}$$



Πολυπλοκότητα Αλγόριθμου Quick-Sort

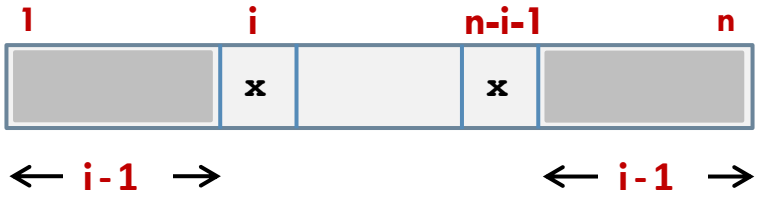
Μέση Πολυπλοκότητα

Μπορούμε εύκολα να απλοποιήσουμε την αναδρομική σχέση

$$A(n) = \begin{cases} 0 & n < 2 \\ (n-1) + \sum_{i=1}^{n} \frac{1}{n} \cdot (A(i-1) + A(n-i)) & n \geq 2 \end{cases}$$

παρατηρώντας ότι:

Μια υποακολουθία με μήκος **i-1** ($1 \leq i \leq n$) εμφανίζεται **2** φορές: όταν το pivot καταλάβει την **i** και **n-i-1** θέση !!!



Μήκος 1ου τμήματος	Θέση pivot	Μήκος 2ου τμήματος
(0)	1	(n-1)
(1)	2	(n-2)
(2)	3	(n-3)
.	.	.
.	.	.
(n-2)	n-1	(1)
(n-1)	n	(0)

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

Μπορούμε εύκολα να απλοποιήσουμε την αναδρομική σχέση

$$A(n) = \begin{cases} 0 & n < 2 \\ (n-1) + \sum_{i=1}^{n} \frac{1}{n} \cdot (A(i-1) + A(n-i)) & n \geq 2 \end{cases}$$

και να πάρουμε:

$$A(n) = n - 1 + \frac{1}{n} \cdot \sum_{i=0}^{n-1} 2 \cdot A(i), \qquad n \geq 1$$

Αυτή είναι μια πιο περίπλοκη αναδρομική σχέση σε σχέση με αυτές που έχουμε δει μέχρι τώρα, διότι η τιμή **A(n)** *εξαρτάται από όλες τις προηγούμενες τιμές* !!!

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

$$A(n) = n - 1 + \frac{2}{n} \cdot \sum_{i=0}^{n-1} A(i), \qquad n \geq 1$$

Μπορούμε να χρησιμοποιήσουμε κάποιο ***ευφυή τρόπο*** για τη λύση της ή να ***εικάσουμε*** τη λύση της και να την αποδείξουμε με επαγωγή !

Η τεχνική της ***εικασίας*** είναι μια καλή προσέγγιση για αναδρομικούς αλγορίθμους !

Θα ακολουθήσουμε αυτό την προσέγγιση…
θα ***εικάσουμε*** την τιμή της **A(n)** *και θα την αποδείξουμε με επαγωγή…*

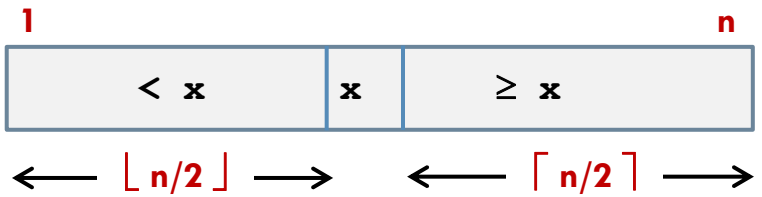
Για να σχηματίσουμε μια εικασία για την τιμή της **A(n)**, εξετάζουμε αρχικά την περίπτωση που ο αλγόριθμος Quick-sort εργάζεται ιδανικά!!!... **Ισομερής Ανάλυση**

Πολυπλοκότητα Αλγόριθμου Quick-Sort

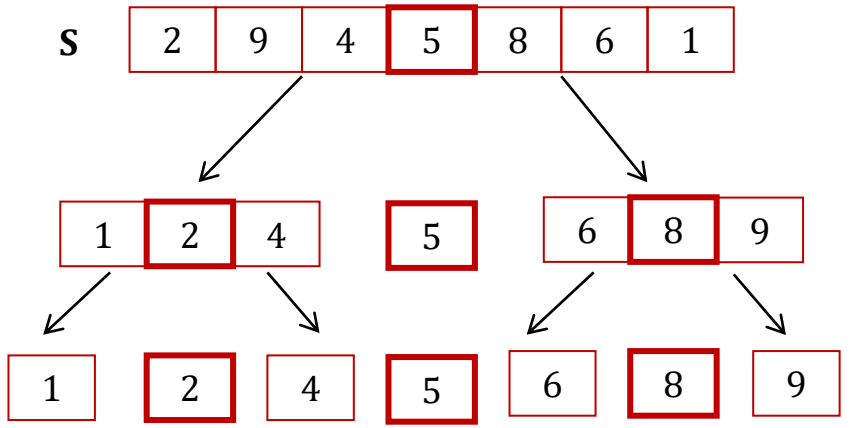
Μέση Πολυπλοκότητα

Ισομερής Ανάλυση

Θεωρούμε την περίπτωση κατά την οποία σε κάθε επανάληψη της διαδικασίας split τα μισά στοιχεία (~**n / 2**) της S είναι μικρότερα του στοιχείου pivot.



Θα θεωρήσουμε το μέγεθος των δύο υποακολουθιών να είναι n/2, χωρίς να υπάρχει ανησυχία για το εάν το n/2 είναι ακέραιος !!!



Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

Ισομερής Ανάλυση

Θεωρούμε την περίπτωση κατά την οποία σε κάθε επανάληψη της διαδικασίας split τα μισά στοιχεία (**n / 2**) της S είναι μικρότερα του στοιχείου pivot.

Άρα, καταλήγουμε στην παρακάτω αναδρομική σχέση:

$$A(n) = \begin{cases} 0 & n < 2 \\ (n-1) + 2 \cdot A(n/2) & n \geq 2 \end{cases}$$

Ισχύει : $A(n) \in \Theta(n \log n)$

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

Συμπέρασμα

Εάν σε κάθε επανάληψη η αναμενόμενη τιμή E[first] του pivot είναι *κοντά στη διάμεσο* (ισομερής ανάλυση) τότε το πλήθος των συγκρίσεων (Βασικών Πράξεων) είναι **Θ(n logn)**, σημαντικά μικρότερο από **Θ(n^2)** !!!

Άρα, ισχύει : **Θ(n logn)** ≤ **A(n)** ≤ **Θ(n^2)**

Ερώτημα!!!... εάν *όλες οι μεταθέσεις έχουν την ίδια πιθανότητα εμφάνισης*, τότε μήπως υπάρχουν πολλές «καλές» μεταθέσεις (ακολουθίες εισόδου) που επηρεάσουν θετικά τη μέση πολυπλοκότητα A(n) του **Quick-Sort** ώστε A(n) = Θ(n logn);

Αποδεικνύουμε : ΝΑΙ... αυτό ισχύει !!!

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

$$A(n) = n - 1 + \frac{2}{n} \cdot \sum_{i=0}^{n-1} A(i), \qquad n \geq 1$$

Πρόταση. **A(n) = c n ln n** για n ≥ 1, όπου c σταθερά.

Απόδειξη:

Με επαγωγή στο πλήθος **n** των στοιχείων εισόδου.

Βάση επαγωγής: Για n = 1, A(1) = 0 και **c n ln n = 0.**

Επαγωγική υπόθεση: Έστω ότι ισχύει **A(i) ≤ c i ln i** για 1 ≤ i < n, όπου n > 1.

Επαγωγικό βήμα : **A(n) ≤ c n ln n**, όπου n > 1.

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

$$A(n) = n - 1 + \frac{2}{n} \cdot \sum_{i=0}^{n-1} A(i), \qquad n \geq 1$$

Από επαγωγική υπόθεση:

$$A(n) = n - 1 + \frac{2}{n} \sum_{i=1}^{n-1} A(i) \;\leq\; n - 1 + \frac{2}{n} \sum_{i=1}^{n-1} c \cdot i \cdot \ln\, i$$

Ισχύει:

$$\sum_{i=1}^{n-1} c \cdot i \cdot \ln\, i \;\leq\; c \cdot \int_{1}^{n} x \cdot \ln\, x \, dx \;=\; \frac{1}{2} n^2 \ln n - \frac{1}{4} n^2$$

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

$$A(n) = n - 1 + \frac{2}{n} \cdot \sum_{i=0}^{n-1} A(i), \qquad n \geq 1$$

Άρα, έχουμε:

$$A(n) \leq n - 1 + \frac{2c}{n} \sum_{i=1}^{n-1} i \cdot \ln i = n - 1 + \frac{2c}{n}(\frac{1}{2}n^2 \ln n - \frac{1}{4}n^2)$$

$$= c\, n \ln n + n(1 - \frac{c}{2}) - 1$$

$$n(1 - \frac{c}{2}) \leq 0 \quad \text{for } c \geq 2.$$

Πολυπλοκότητα Αλγόριθμου Quick-Sort

Μέση Πολυπλοκότητα

$$A(n) = n - 1 + \frac{2}{n} \cdot \sum_{i=0}^{n-1} A(i), \qquad n \geq 1$$

Άρα, για $c = 2$ ισχύει :

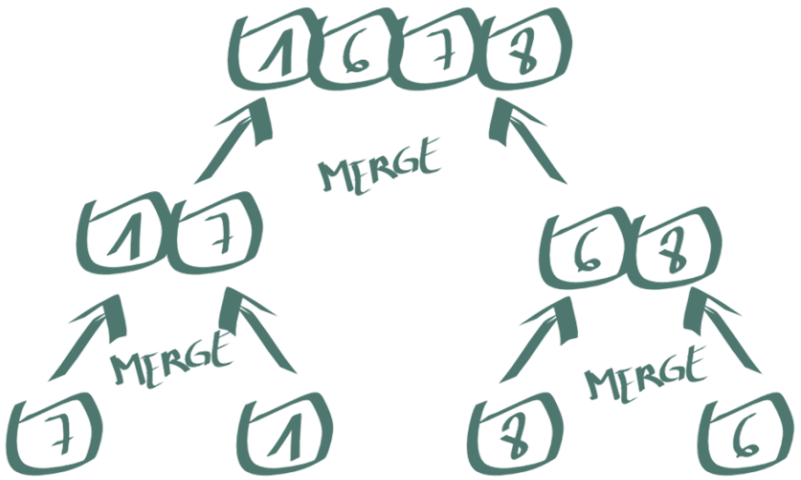
$$A(n) \leq 2\, n\, \ln n$$ QED

Επειδή, $\ln n \approx 0.693 \log n$

Ισχύει : $A(n) \in \Theta(n \log n)$

Αλγόριθμος Ταξινόμησης Merge-Sort

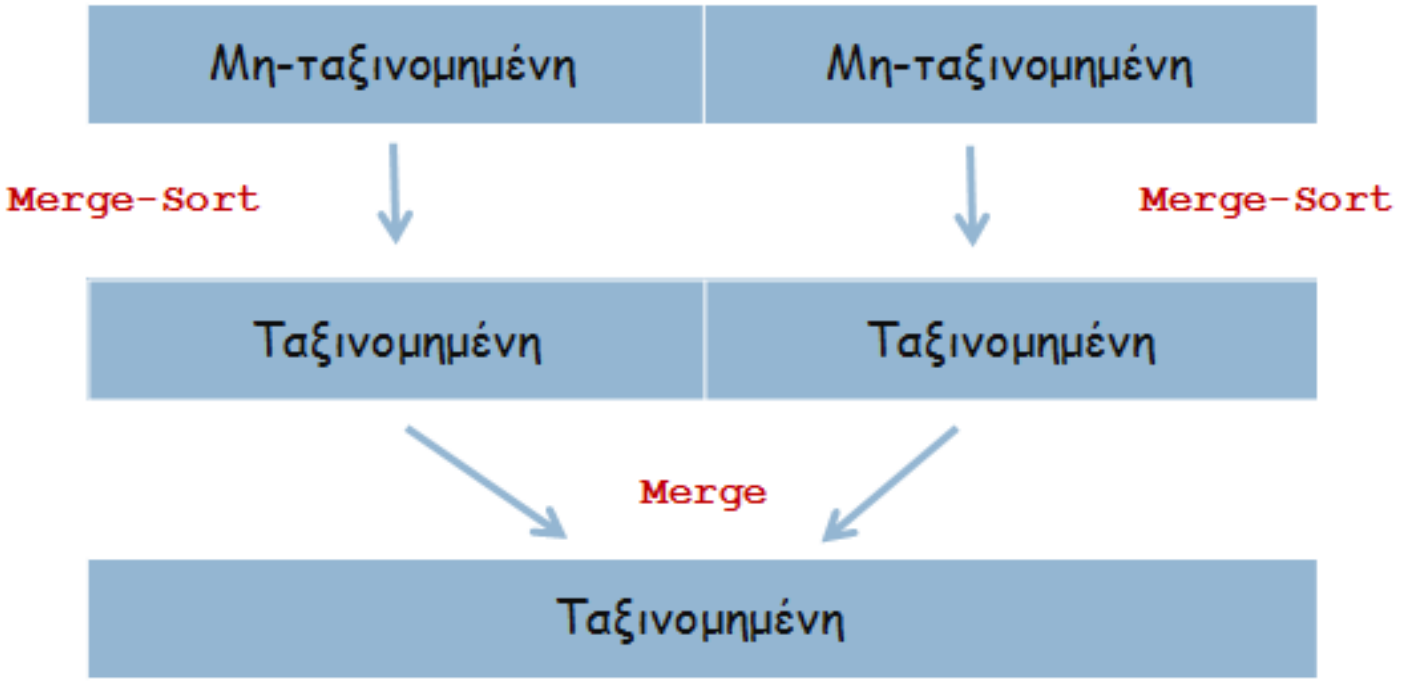
Divide & Conquer



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

Αλγόριθμος Merge-Sort

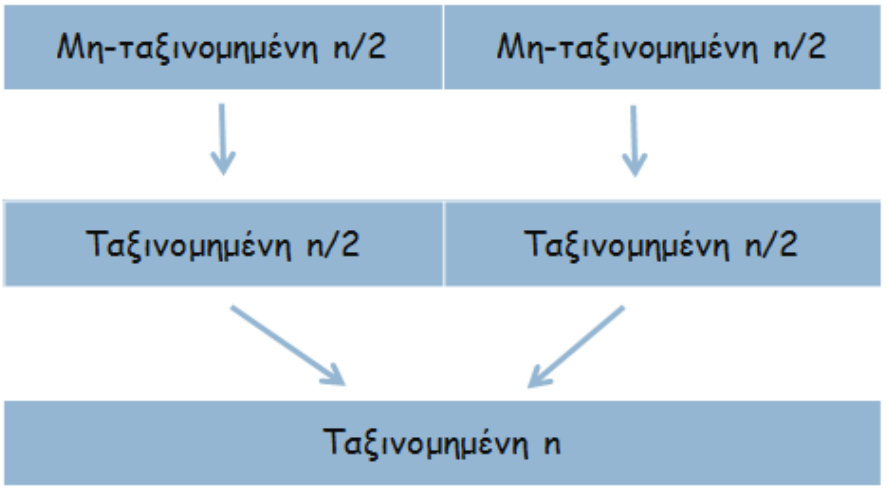
Ιδέα Αλγόριθμου !!!... Ταξινόμηση με Συγχώνευση



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)

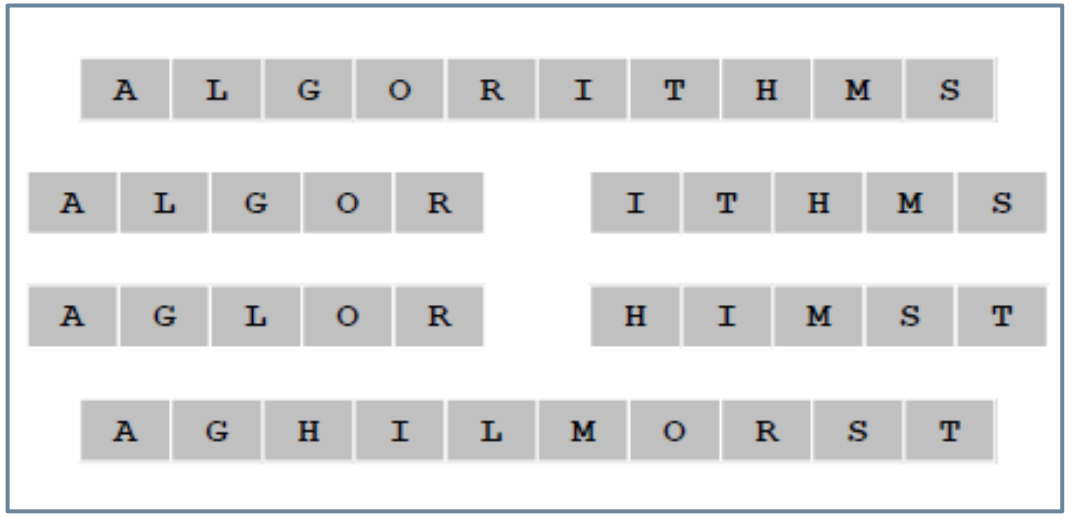
Ταξινόμησε 2 (conquer)

Συγχώνευσε 3 (combine)

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)

Αλγόριθμος Merge-Sort

Αλγόριθμος

- 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.

```
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
```

Αλγόριθμος Merge-Sort

Συγχώνευση (Merge)

Συγχώνευση δύο ταξινομημένων ακολουθιών

$A = a_1, a_2, \ldots, a_n \qquad B = b_1, b_2, \ldots, b_n$

σε μία μόνο ταξινομημένη ακολουθία C (πίνακα ή λίστα) !!!

Algorithm Merge()

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

Συγχώνευση (Merge)

Παράδειγμα

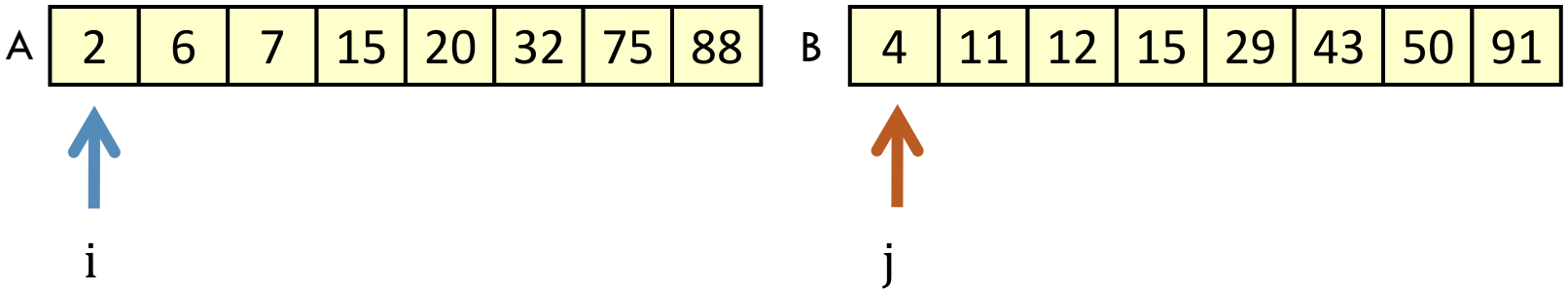
Συγχώνευση δύο ταξινομημένων ακολουθιών

A	2	6	7	15	20	32	75	88

B	4	11	12	15	29	43	50	91

Συγχώνευση (Merge)

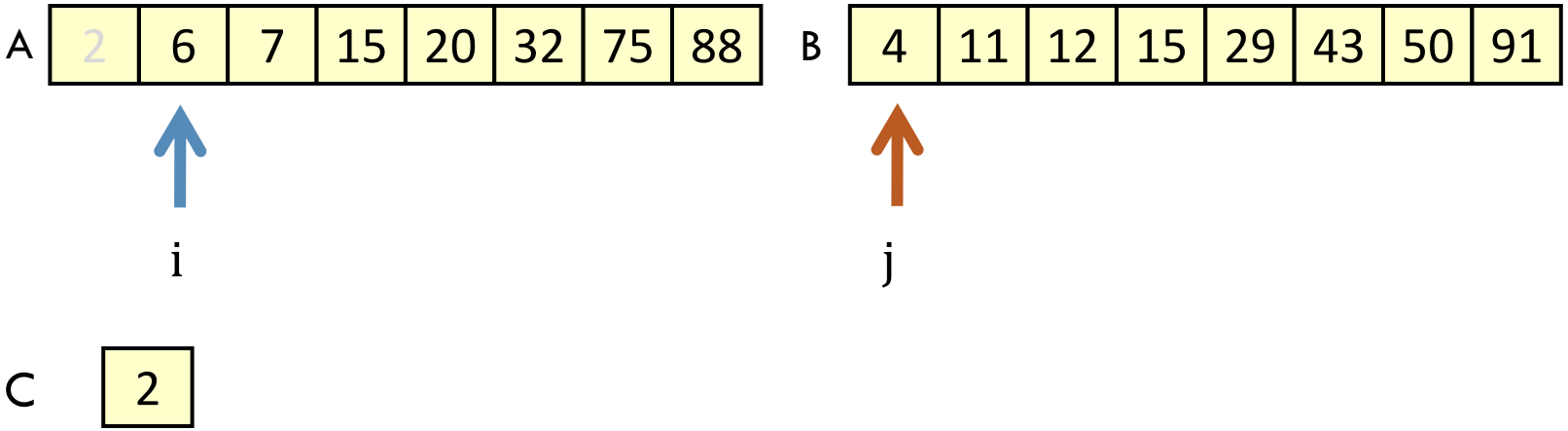
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

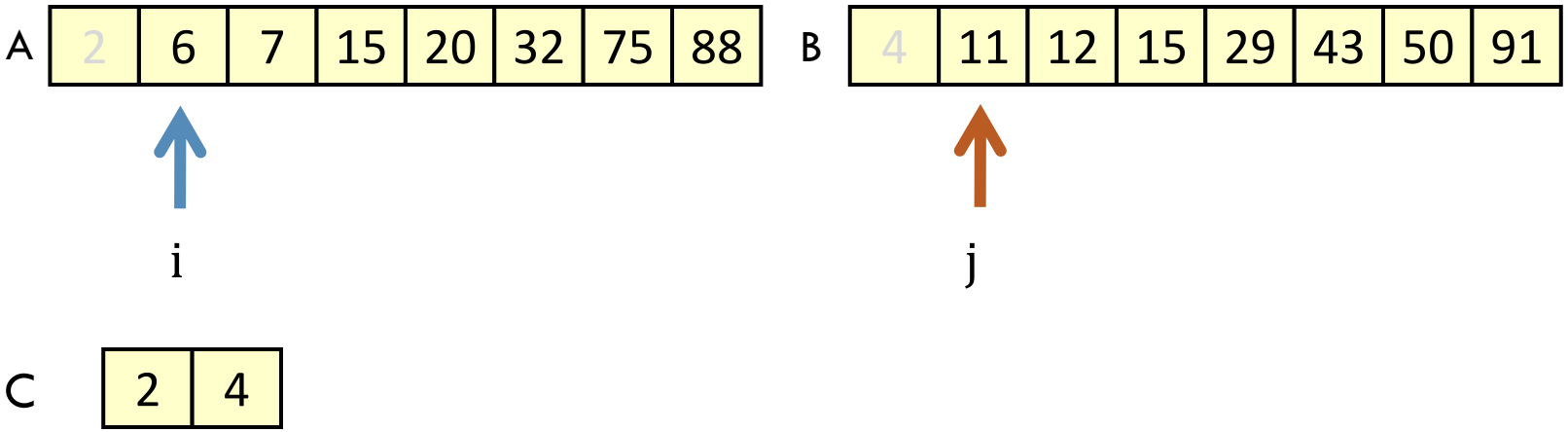
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

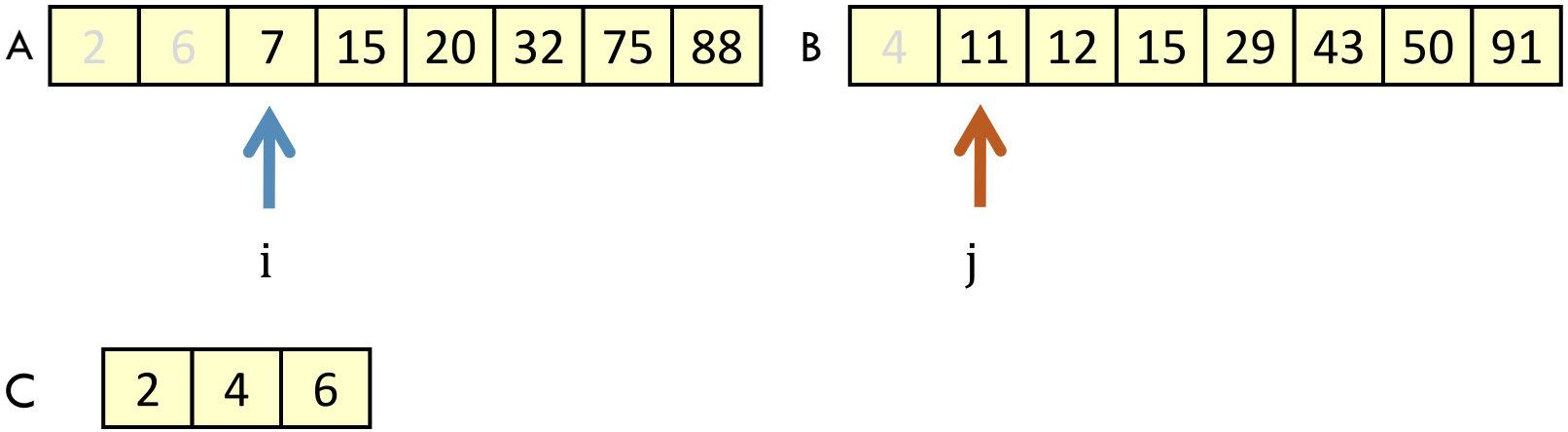
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

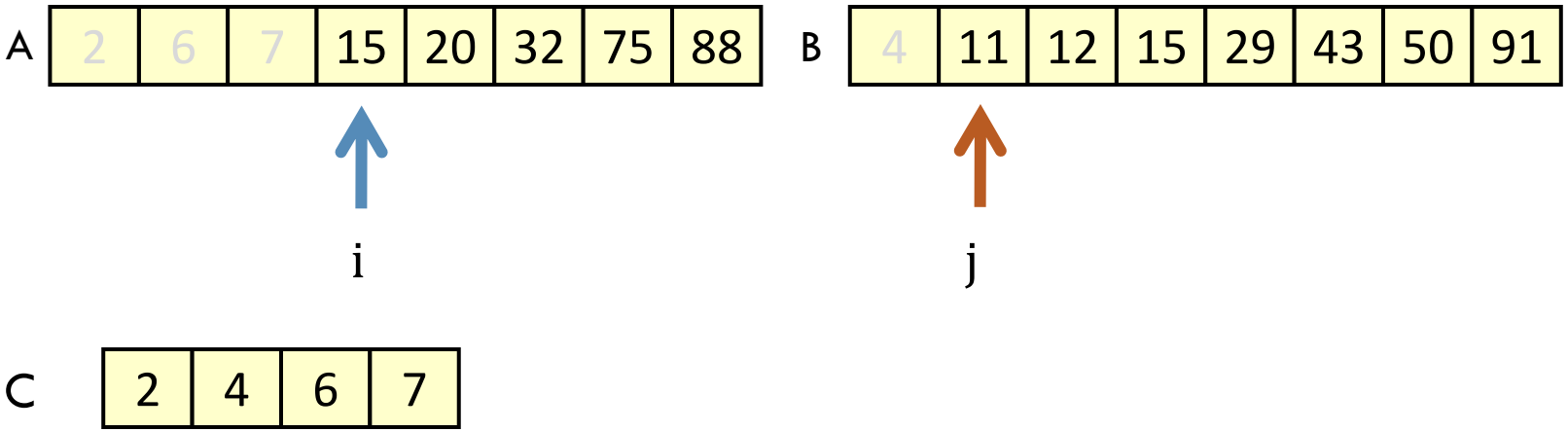
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

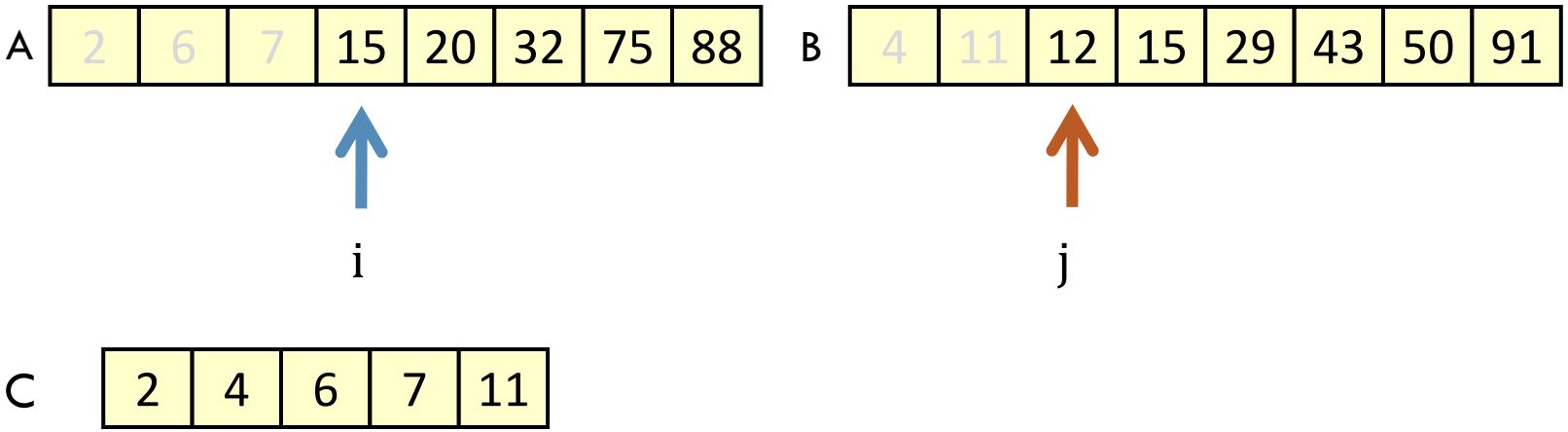
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

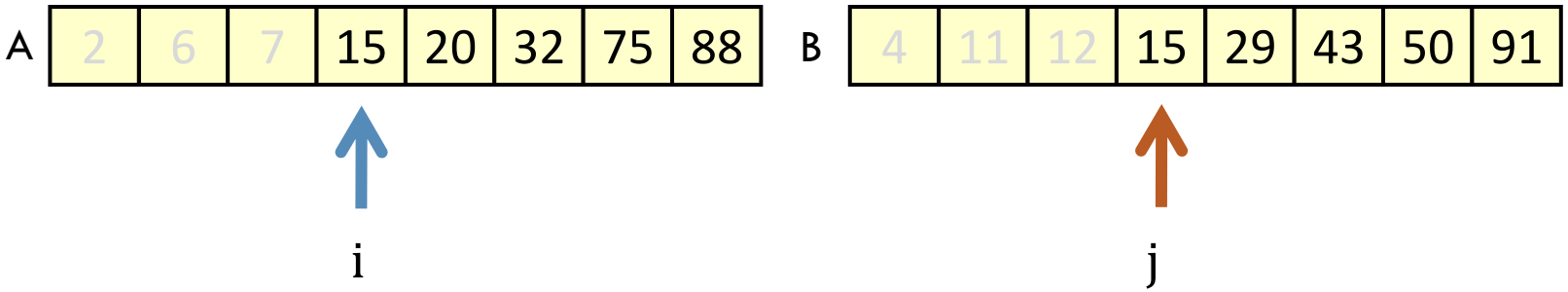
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

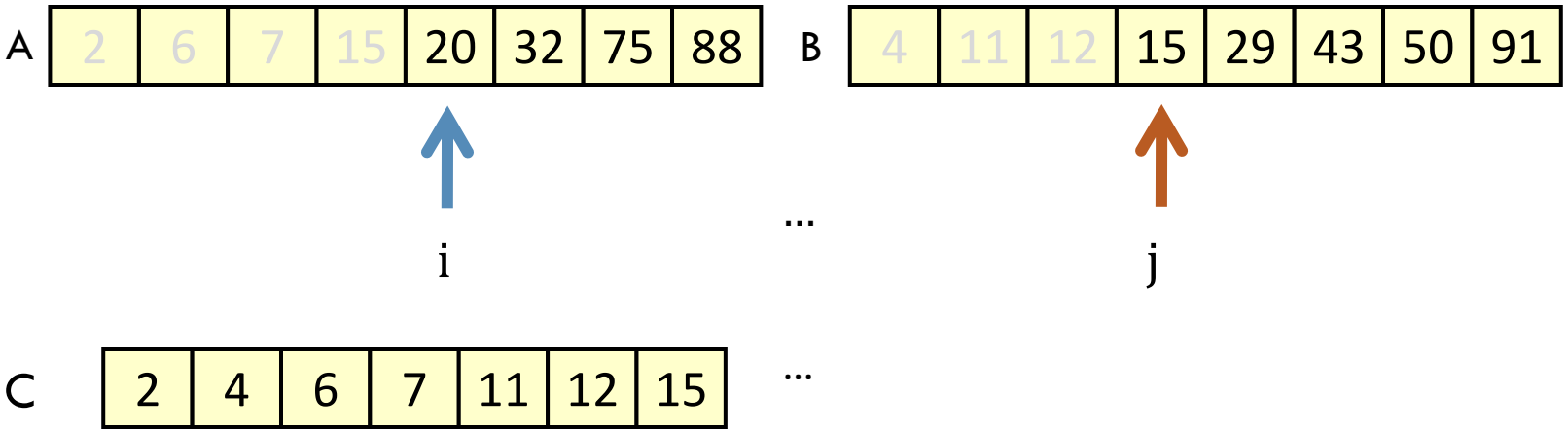
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

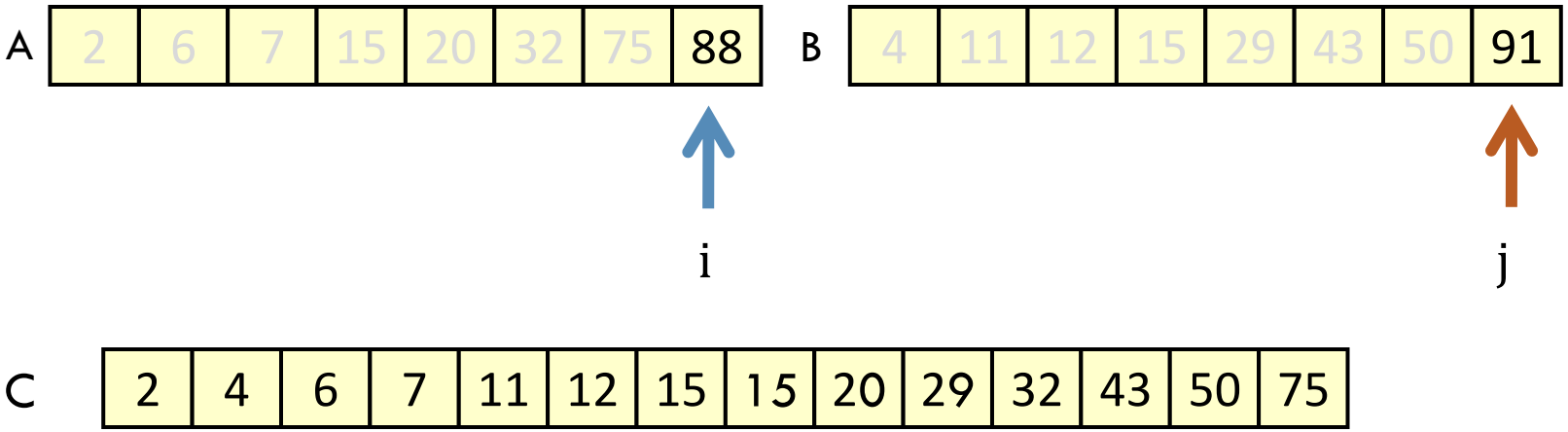
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

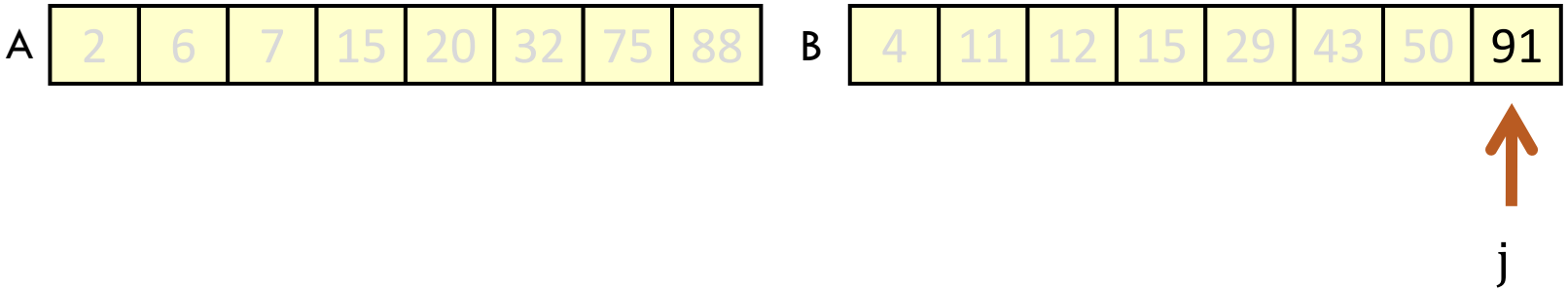
Παράδειγμα



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

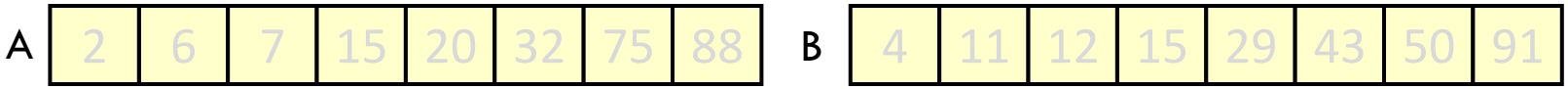
Παράδειγμα

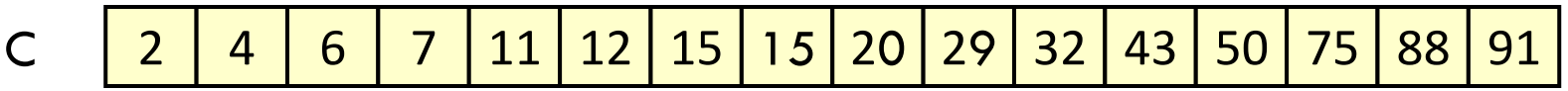


Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

Παράδειγμα





Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

Συγχώνευση δύο ταξινομημένων ακολουθιών

Απαιτεί
3 ξεχωριστούς πίνακες

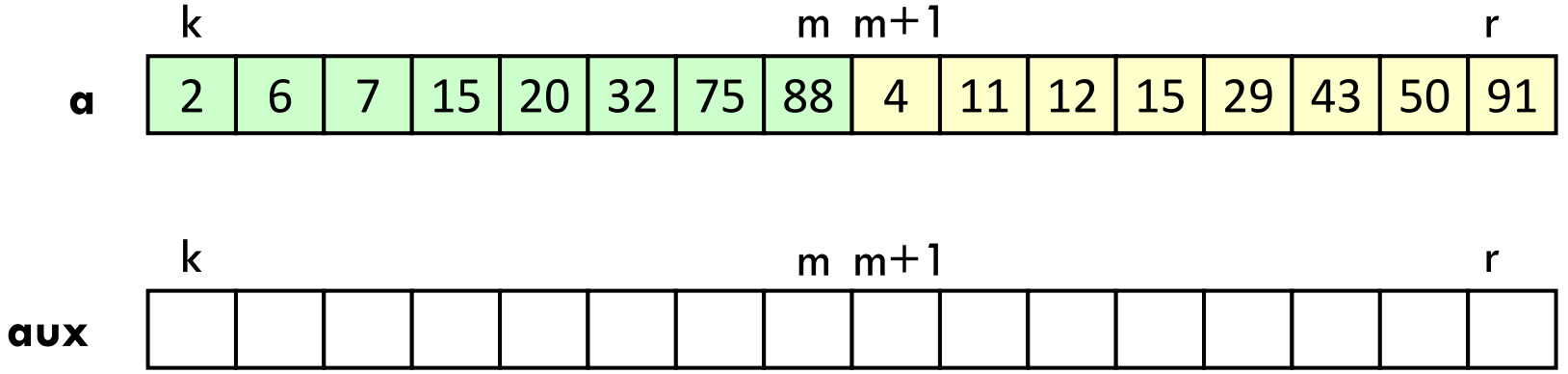
A	2	6	7	15	20	32	75	88

B	4	11	12	15	29	43	50	91

C	2	4	6	7	11	12	15	15	20	29	32	43	50	75	88	91

Συγχώνευση (Merge)

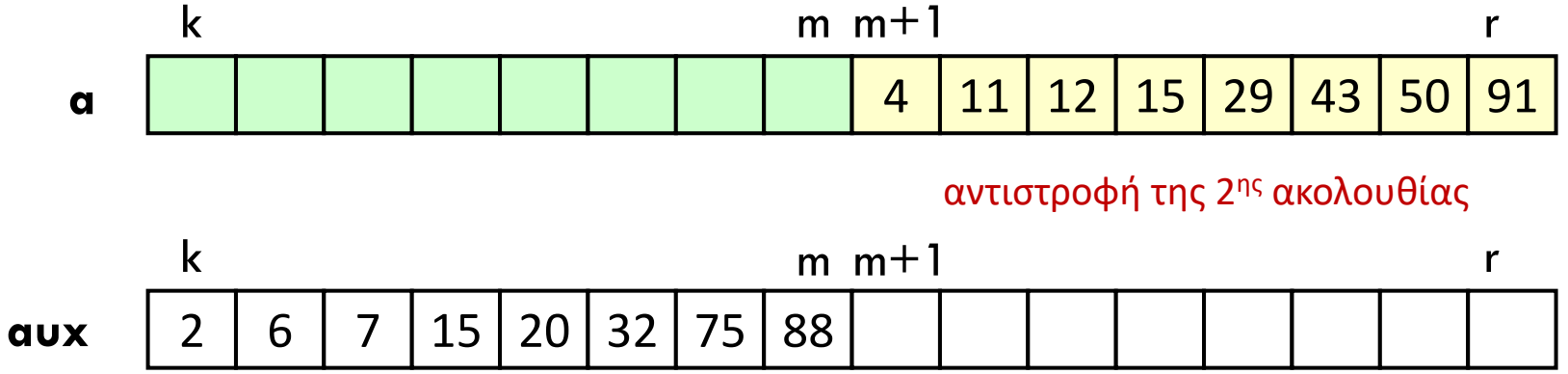
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

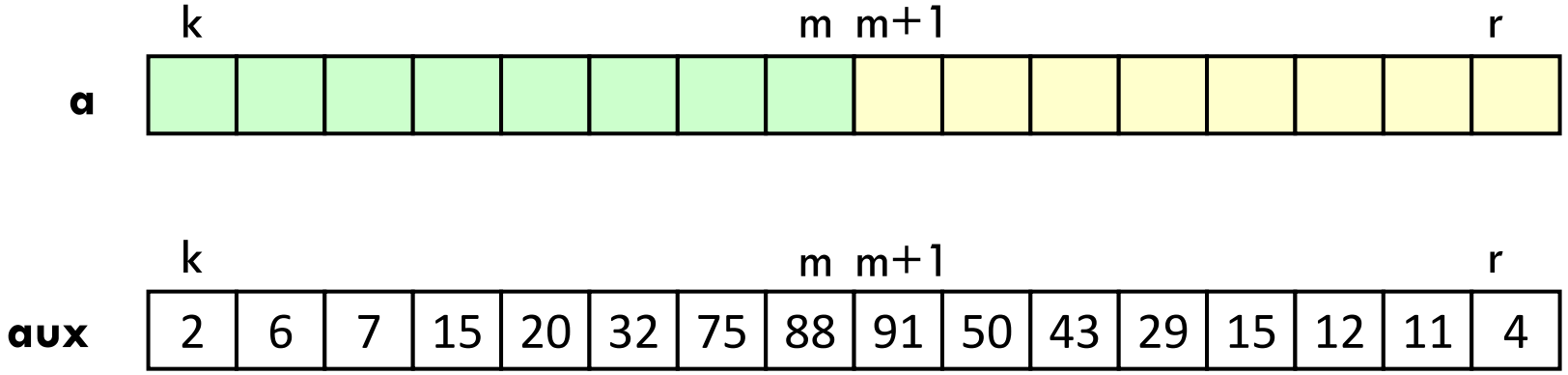
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

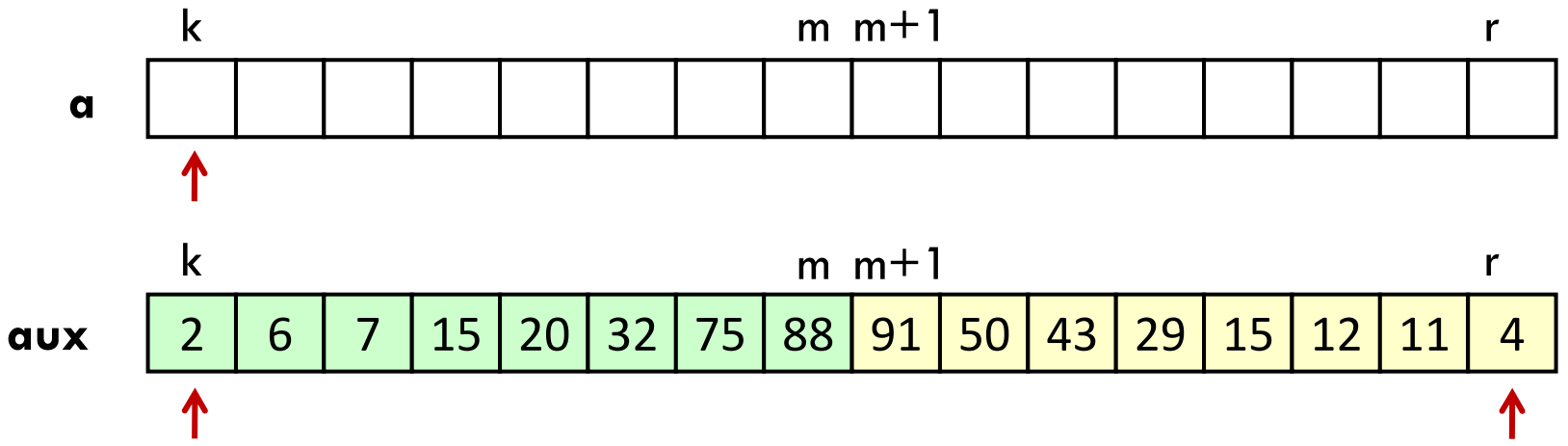
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

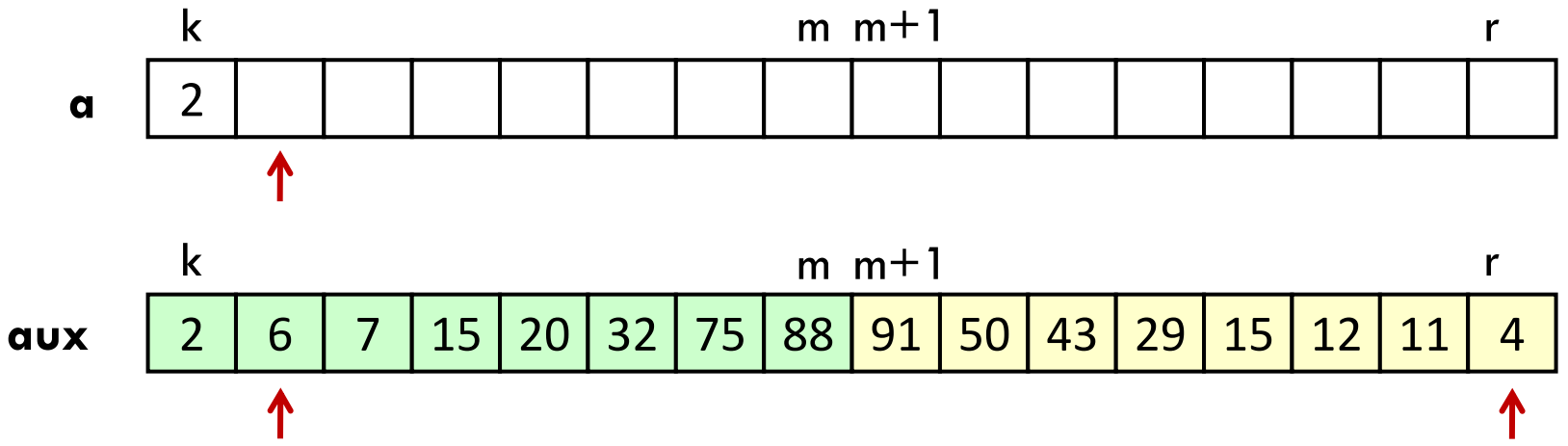
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

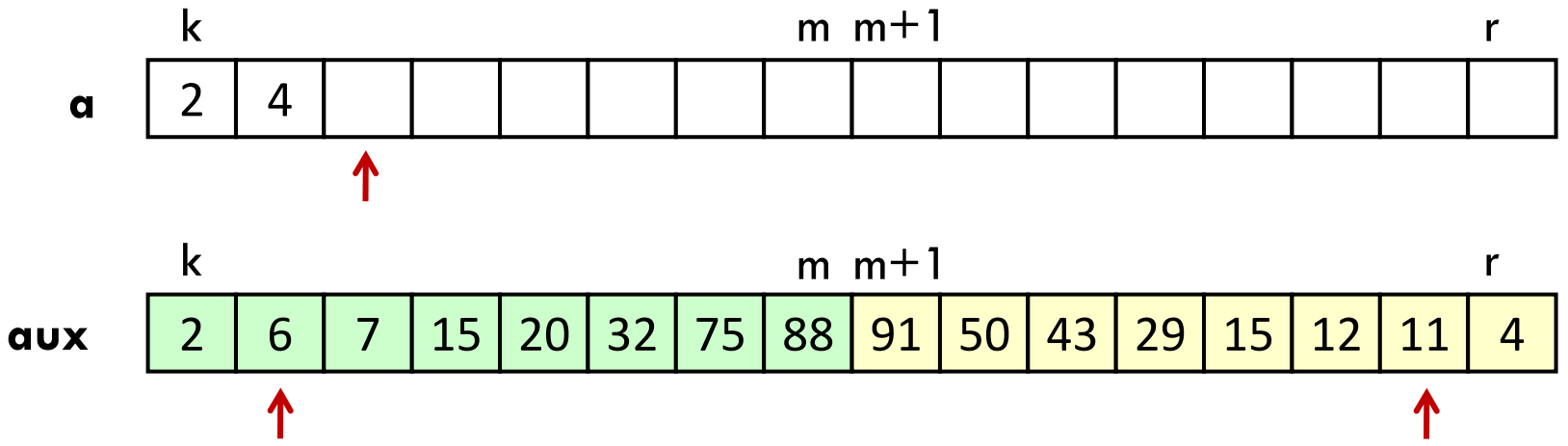
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

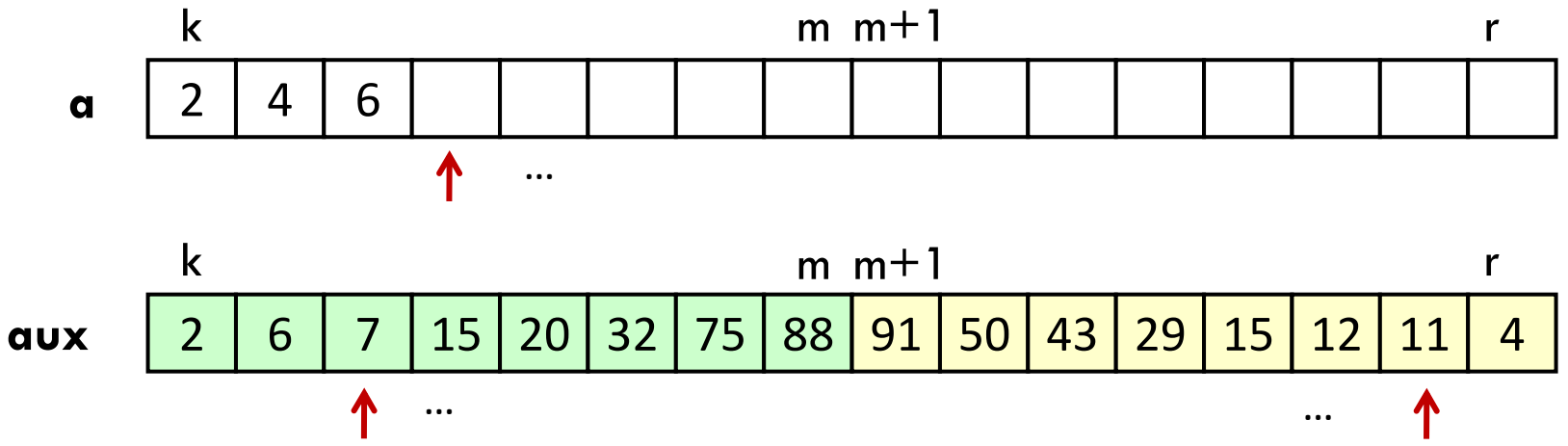
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

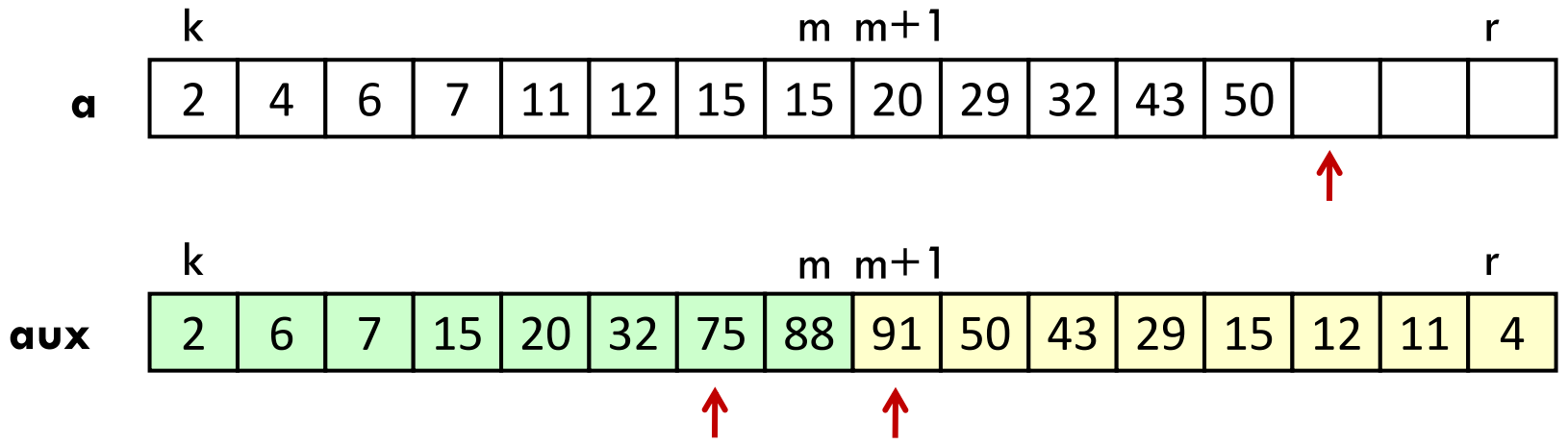
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

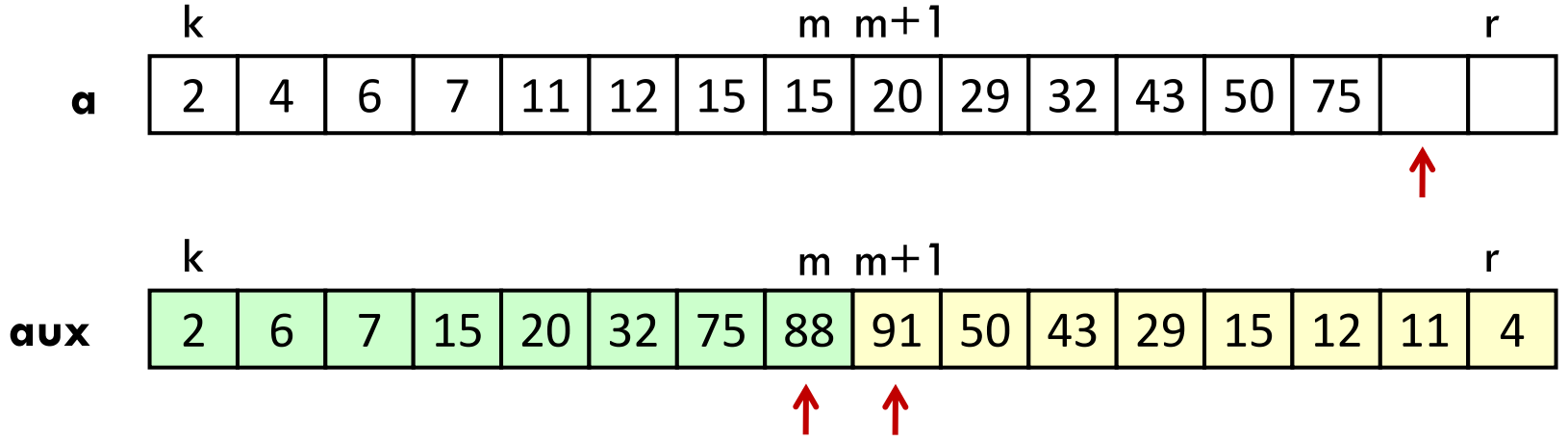
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

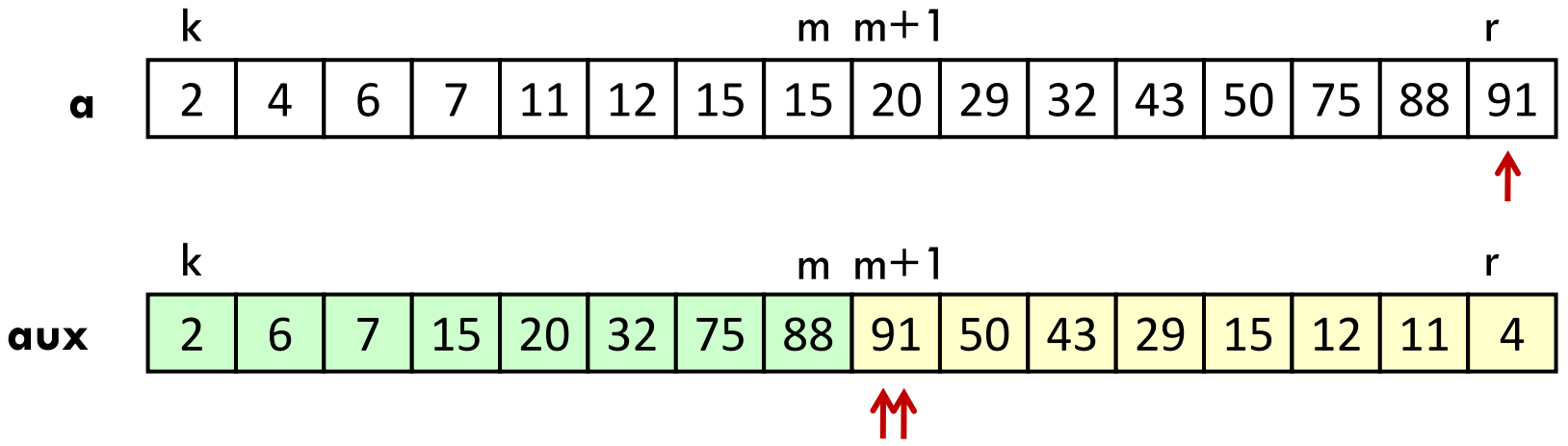
Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

Συγχώνευση με
2 πίνακες



Συγχώνευση δύο ταξινομημένων ακολουθιών

Συγχώνευση (Merge)

Συγχώνευση δύο ταξινομημένων ακολουθιών

Απαιτεί
1 βοηθητικό πίνακα

a	2	4	6	7	11	12	15	15	20	29	32	43	50	75	88	91

aux	2	6	7	15	20	32	75	88	91	50	43	29	15	12	11	4

Αλγόριθμος Merge-Sort

Συγχώνευση (Merge)

Συγχώνευση δύο ταξινομημένων ακολουθιών

Απαιτεί
1 βοηθητικό πίνακα

a	2	4	6	7	11	12	15	15	20	29	32	43	50	75	88	91

Μπορούμε χωρίς Βοηθητικό Πίνακα

Πρόκληση για αυτούς που ενδιαφέρονται!... Επί τόπου συγχώνευση [Kronrud, 1969]

Χρησιμοποιώντας επιπλέον μόνο ένα σταθερό ποσό μνήμης !!!

Αλγόριθμος Merge-Sort

Συγχώνευση (Merge)

Συγχώνευση δύο ταξινομημένων ακολουθιών

$A = a_1, a_2, \ldots, a_n$ $B = b_1, b_2, \ldots, b_n$

σε μία μόνο ταξινομημένη ακολουθία C (πίνακα ή λίστα) !!!

Algorithm Merge()

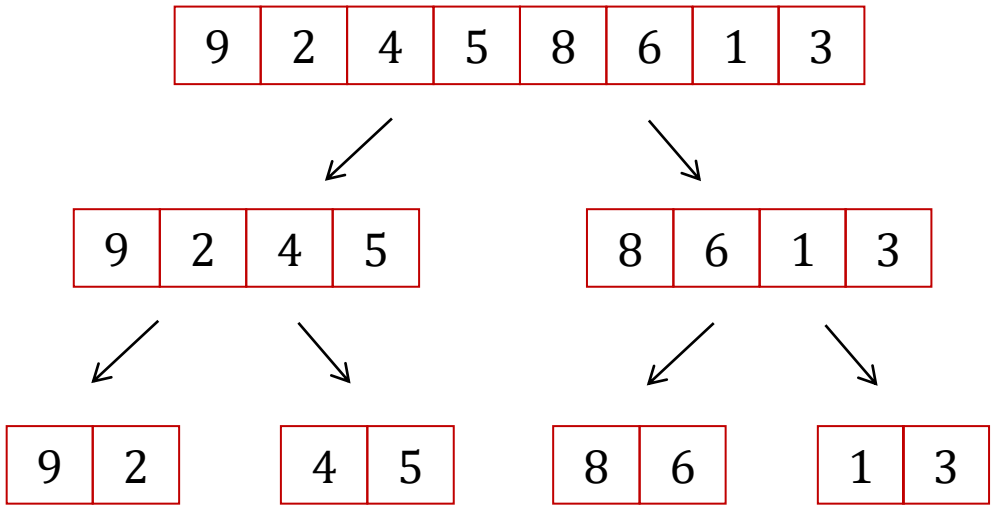
Ισχυρισμός. Η συγχώνευση δύο ταξινομημένων ακολουθιών A και B μήκους n απαιτεί χρόνο O(n).

Απόδειξη: Μετά από κάθε σύγκριση, το μέγεθος της ακολουθίας εξόδου C αυξάνεται κατά 1.

Αλγόριθμος 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
```

Παράδειγμα εφαρμογής Αλγόριθμου



Διαίρεση

Αλγόριθμος 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
```

Παράδειγμα εφαρμογής Αλγόριθμου

Ταξινόμηση

Αλγόριθμος 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
```

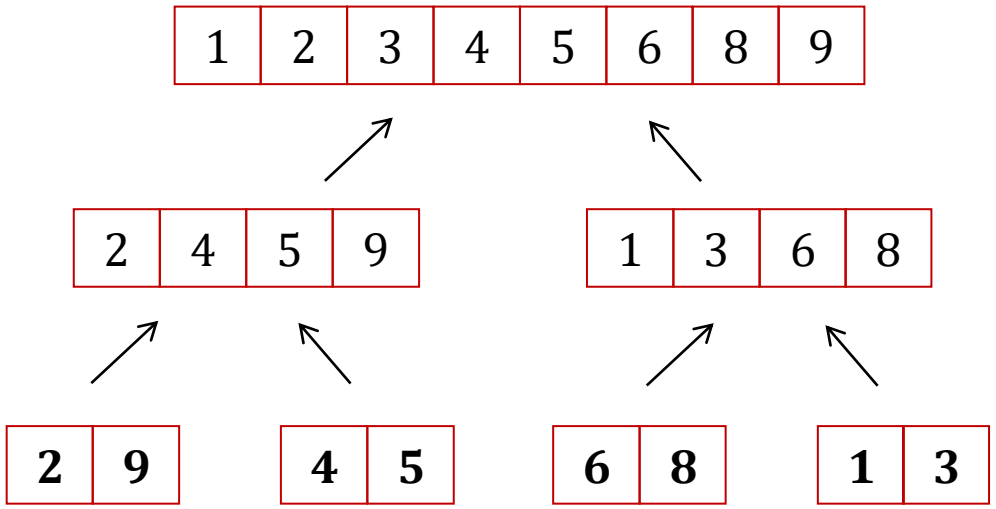
Παράδειγμα εφαρμογής Αλγόριθμου

Ταξινόμηση

Αλγόριθμος 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
```

Παράδειγμα εφαρμογής Αλγόριθμου



Συγχώνευση

Ορθότητα Αλγόριθμου Merge-Sort

Ορθότητα αλγόριθμου Merge:

- Οι δύο υποακολουθίες A και B είναι ταξινομημένες.
- Όταν ένα στοιχείο μεταφέρεται στον (βοηθητικό) πίνακα C, δεν υπάρχει μικρότερο διαθέσιμο στοιχείο στις A και B.

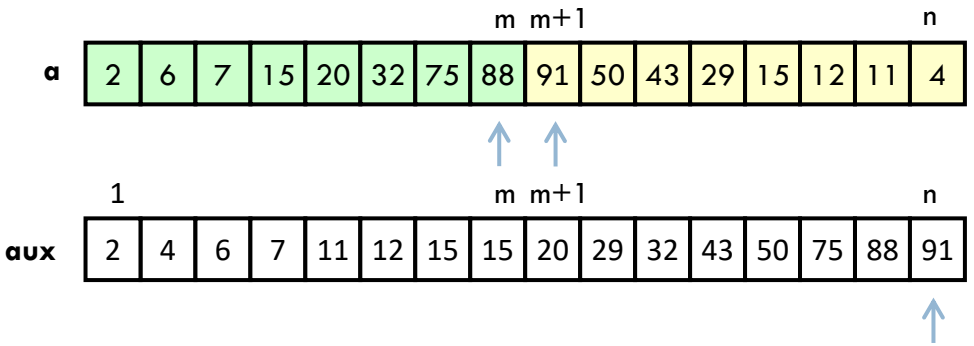
Η ορθότητα του αλγορίθμου Merge-Sort αποδεικνύεται επαγωγικά:

- Βάση επαγωγής (ένα στοιχείο) τετριμμένη.
- Οι δύο υποακολουθίες ταξινομούνται σωστά (επαγωγική υπόθεση), και συγχωνεύονται σωστά (ορθότητα merge).
- Άρα, η ακολουθία A είναι σωστά ταξινομημένη.

Πολυπλοκότητα Αλγόριθμου Merge-Sort

Πολυπλοκότητα Αλγόριθμου Merge

Χειρίστη Πολυπλοκότητα



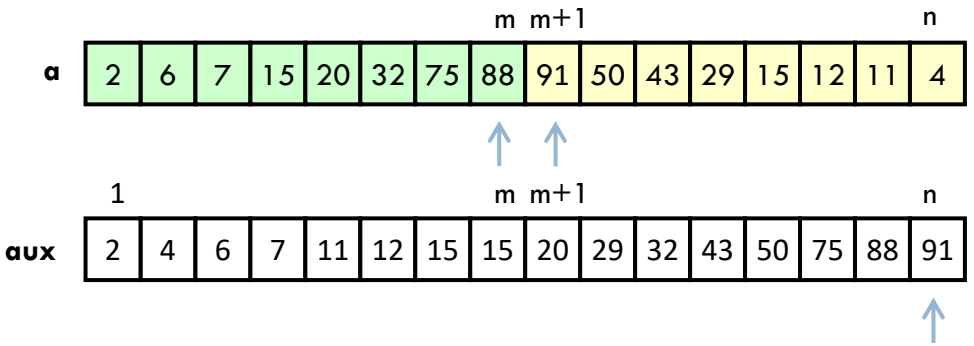

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left >= right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

- Κάθε φορά συγκρίνονται δύο στοιχεία και *ένα μόνο* στοιχείο **x** μεταφέρεται στον βοηθητικό πίνακα **aux** μήκους **n** και *ποτέ δεν ξανασυγκρίνεται* !!!
- Στην τελευταία σύγκριση δύο στοιχεία **x** και **y** δεν έχουν μεταφερθεί στον **aux**: το min(x,y) μεταφέρεται άμεσα και το max(x,y) αντιγράφεται.

Πολυπλοκότητα Αλγόριθμου Merge-Sort

Πολυπλοκότητα Αλγόριθμου Merge

Χειρίστη Πολυπλοκότητα




```
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
```

- Άρα, απαιτούνται το πολύ **n-1** συγκρίσεις για τη συγχώνευση **n** στοιχείων (δύο ταξινομημένων ακολουθιών με **m** και **n-m** στοιχεία).
- Χειρίστη Πολυπλοκότητα Merge : **W(n) = n-1**

Ασυμπωτική πολυπλοκότητα χειρίστης περίπτωσης Merge : $W(n) \in \Theta(n)$

Πολυπλοκότητα Αλγόριθμου Merge-Sort

- **Χειρίστη Πολυπλοκότητα Merge-Sort**

  Η αναδρομική σχέση της πολυπλοκότητας του αλγόριθμου Merge-Sort είναι η εξής:

$$W(n) = W(\lfloor \frac{n}{2} \rfloor) + W(\lceil \frac{n}{2} \rceil) + (n-1)$$

$$W(1) = 0$$

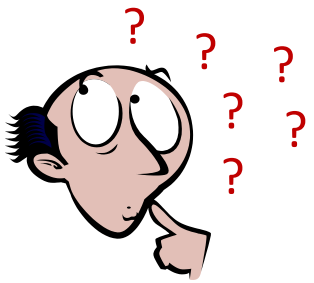
```
Algorithm Merge-Sort(A, left, right)
begin
      if (left >= right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

- Έχουμε $n-1 \in O(n)$ (πολυπλοκότητα συγχώνευσης – αλγόριθμος Merge)
- Κύριο Θεώρημα ($2^{η}$ περίπτωση $c = \log_2 2 = 1$) : **W(n) = O(n log n)**

Ασυμπωτική πολυπλοκότητα χειρίστης περίπτωσης Merge-Sort : $W(n) \in \Theta(n \log n)$

Πολυπλοκότητα Αλγόριθμου Merge-Sort

- **Μέση Πολυπλοκότητα 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
```

- Αντί να έρθουμε άμεσα να υπολογίσουμε τη μέση περίπτωση της πολυπλοκότητας του **Merge-Sort**, ας περιμένουμε έως να ελέγξουμε ένα κάτω-φράγμα για το πρόβλημα της ταξινόμησης με σύγκριση στοιχείων.

Ασυμπωτική πολυπλοκότητα μέσης περίπτωσης Merge-Sort : $A(n) = ?$

Άνω Φράγμα Ταξινόμησης

Αλγόριθμος : **Merge-Sort**

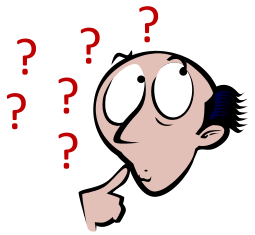
```
begin
    .
    .
   Merge-Sort
    .
    .
    .
end;
```

$W(n) \in \Theta(n \log n)$

Άνω φράγμα για το πρόβλημα της Ταξινόμησης !!!

Έχει ο Merge-Sort καλύτερη πολυπλοκότητα στη μέση περίπτωση ;

Υπάρχει καλύτερος αλγόριθμος ταξινόμησης από τον Merge-Sort;



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:

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

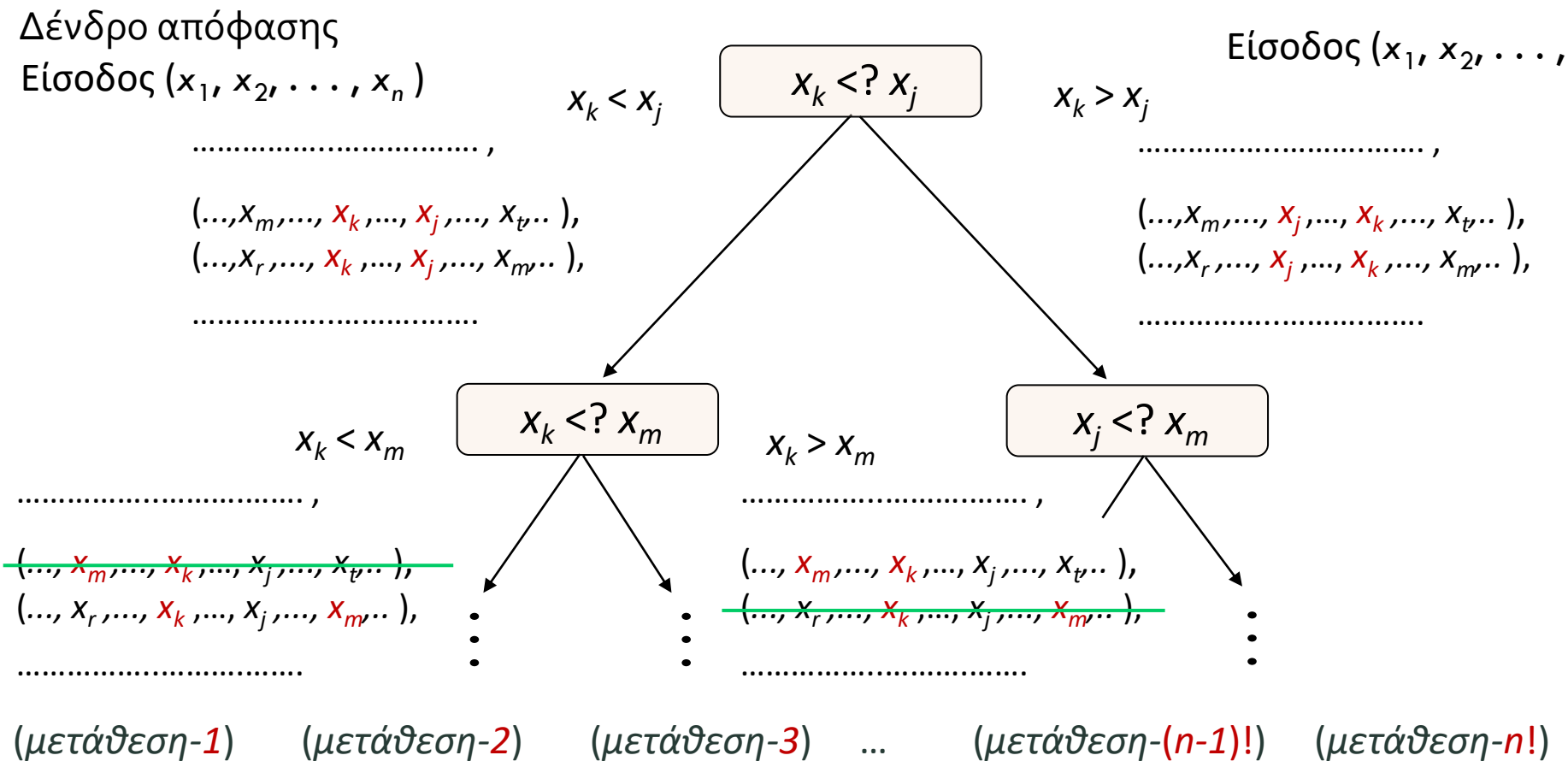
- Σε κάθε σύγκριση το πλήθος των περιπτώσεων υποδιπλασιάζεται (στην καλύτερη περίπτωση)
- Θα δείξουμε:

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

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

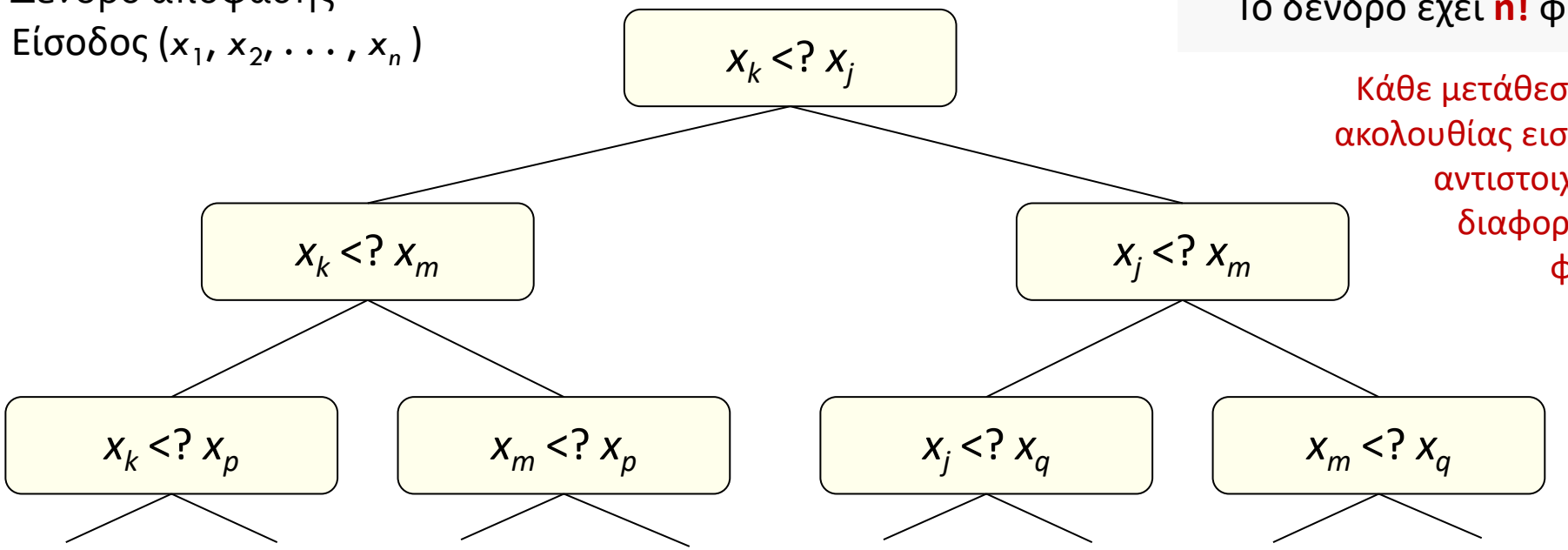
Κάτω φράγμα για ταξινόμηση που χρησιμοποιεί μόνο συγκρίσεις

Κάτω Φράγμα Ταξινόμησης



Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης
Είσοδος $(x_1, x_2, \ldots, x_n)$



Το δένδρο έχει **n!** φύλλα

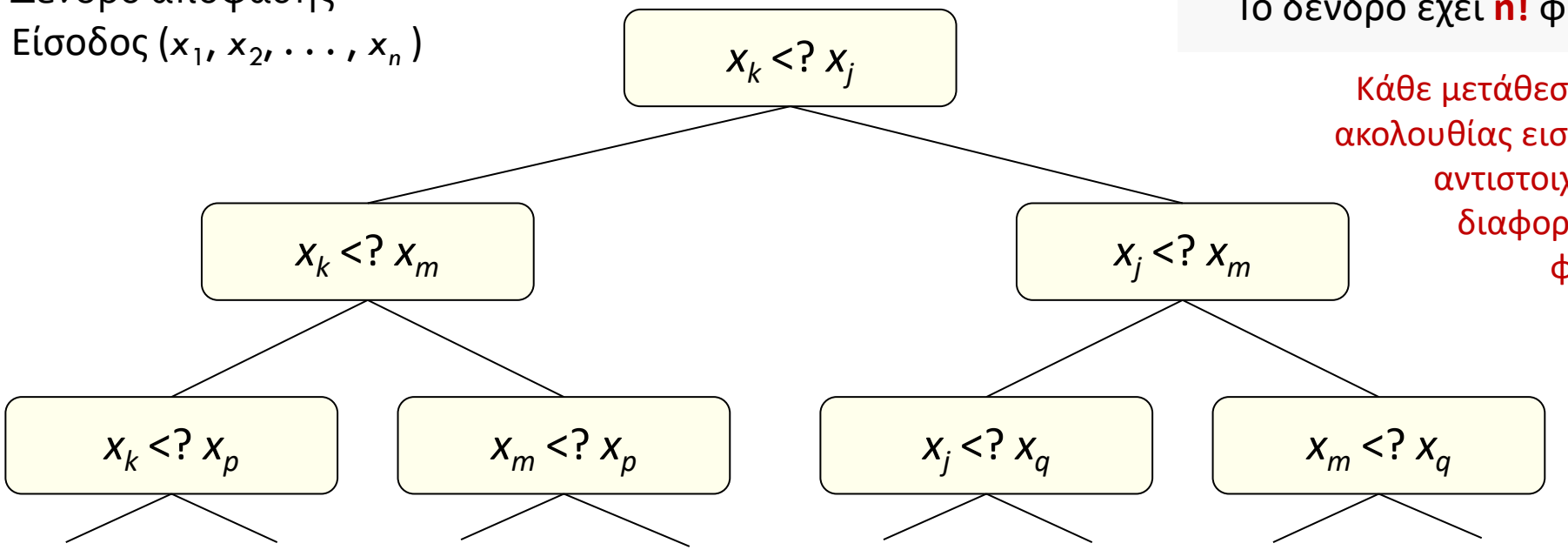
Κάθε μετάθεση της ακολουθίας εισόδου αντιστοιχεί σε διαφορετικό φύλλο

(*μετάθεση-1*) (*μετάθεση-2*) (*μετάθεση-3*) ... (*μετάθεση-(n-1)!*) (*μετάθεση-n!*)

Η λύση είναι μια μετάθεση που βρίσκεται σε ένα φύλλο του δένδρου

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης
Είσοδος $(x_1, x_2, \ldots, x_n)$



Το δένδρο έχει **n!** φύλλα

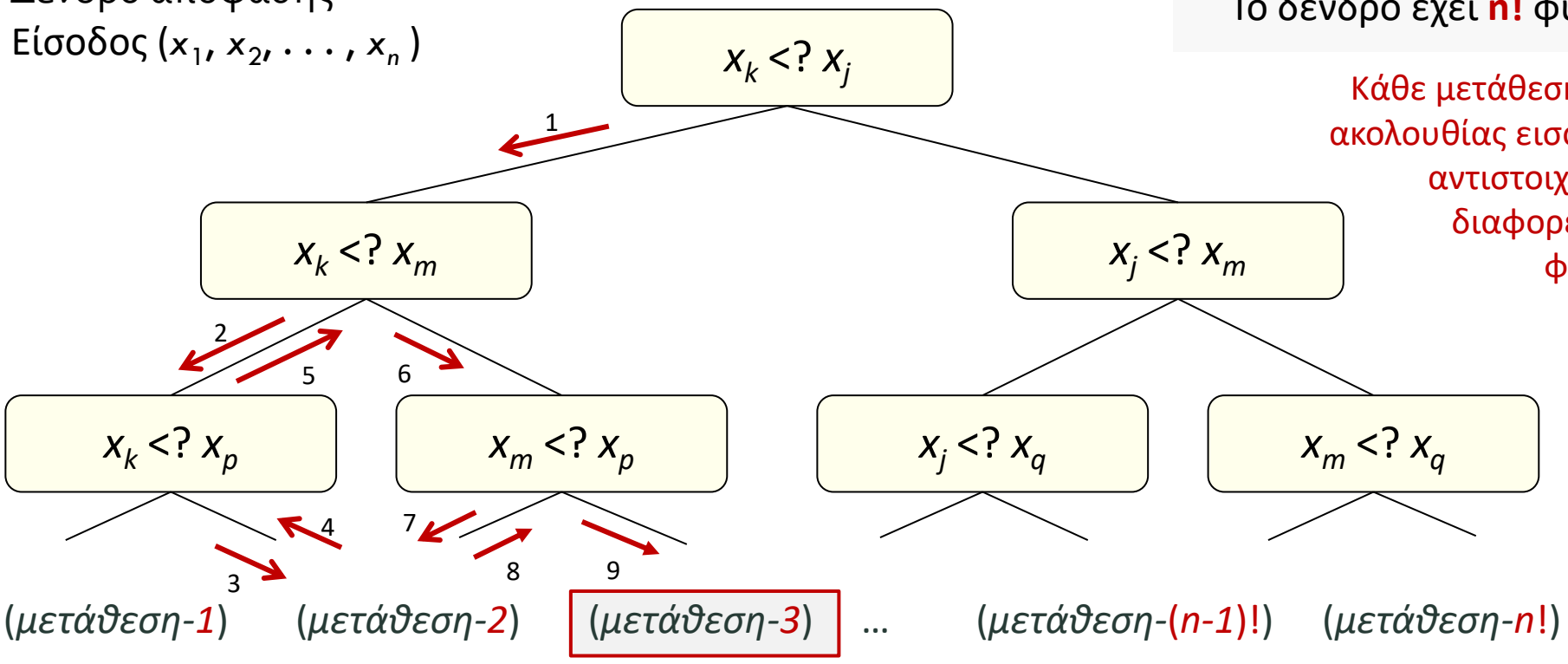
Κάθε μετάθεση της ακολουθίας εισόδου αντιστοιχεί σε διαφορετικό φύλλο

(*μετάθεση-1*) (*μετάθεση-2*) (*μετάθεση-3*) ... (*μετάθεση-(n-1)!*) (*μετάθεση-n!*)

Έστω η λύση είναι η (μετάθεση-3)

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης
Είσοδος $(x_1, x_2, \ldots, x_n)$



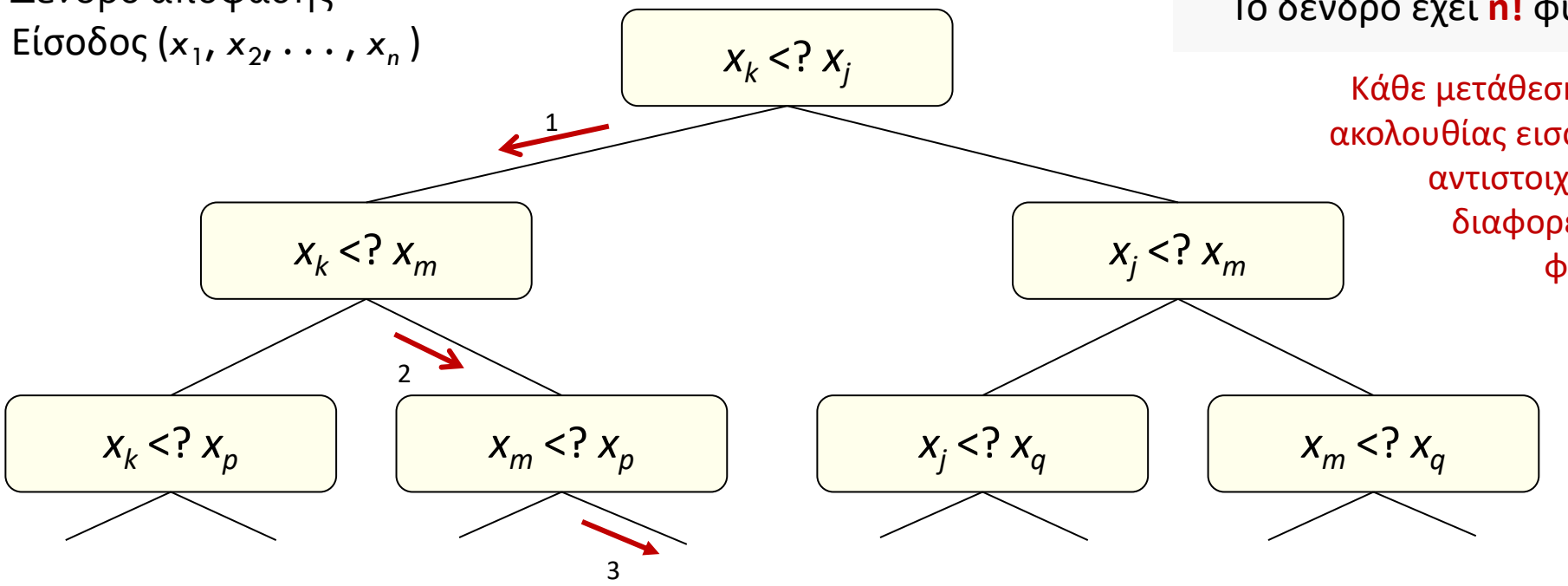
Το δένδρο έχει **n!** φύλλα

Κάθε μετάθεση της ακολουθίας εισόδου αντιστοιχεί σε διαφορετικό φύλλο

Έστω η λύση είναι η (μετάθεση-3)

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης
Είσοδος $(x_1, x_2, \ldots, x_n)$



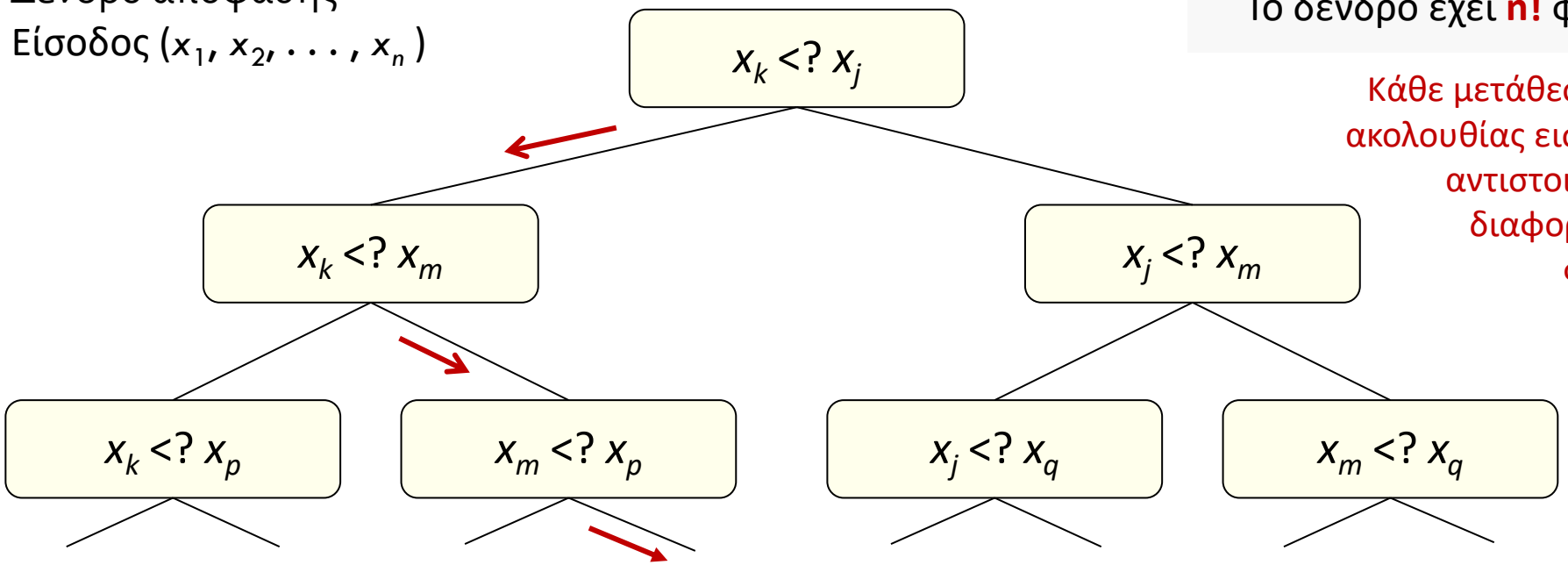
Το δένδρο έχει **n!** φύλλα

Κάθε μετάθεση της ακολουθίας εισόδου αντιστοιχεί σε διαφορετικό φύλλο

Έστω η λύση είναι η (μετάθεση-3)

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης
Είσοδος $(x_1, x_2, \ldots, x_n)$



Το δένδρο έχει **n!** φύλλα

Κάθε μετάθεση της ακολουθίας εισόδου αντιστοιχεί σε διαφορετικό φύλλο

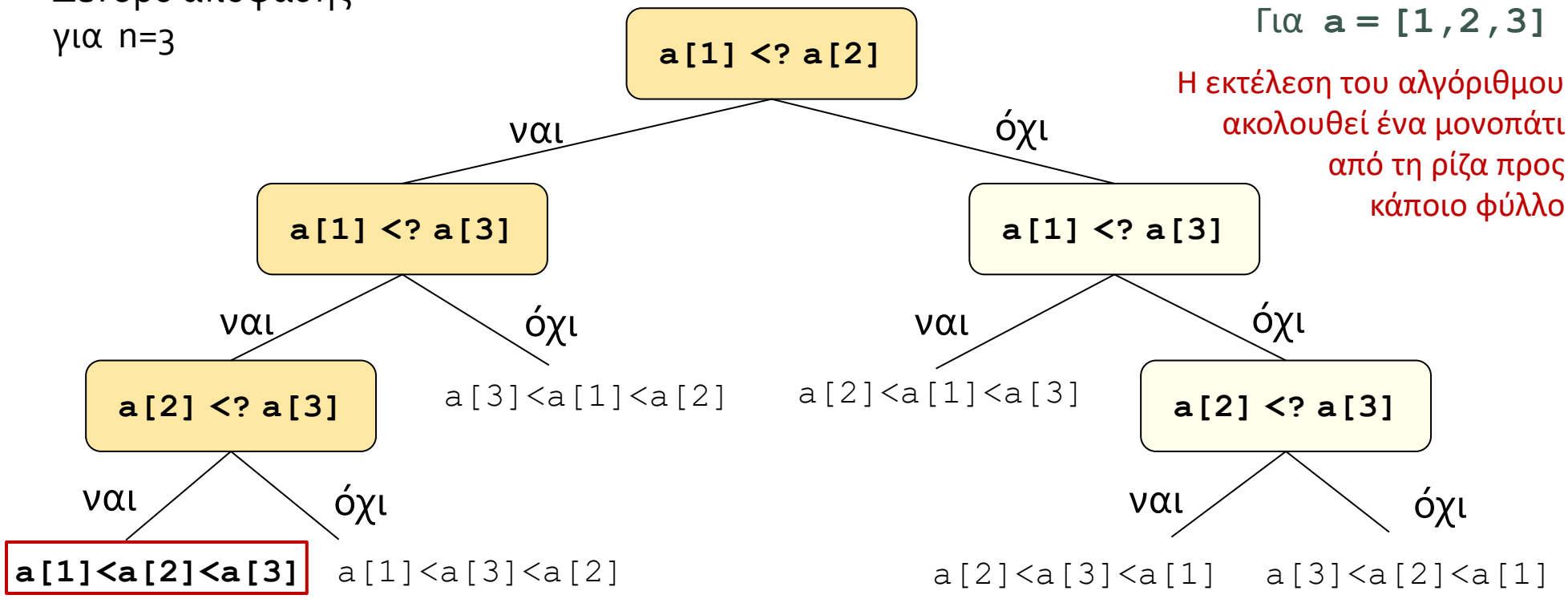
Δεν μπορώ να βρω την λύση με λιγότερες συγκρίσεις από το ύψος του δένδρου !!!

Στη χειρίστη περίπτωση.

Κάτω Φράγμα Ταξινόμησης

Παράδειγμα

Δένδρο απόφασης
για n=3



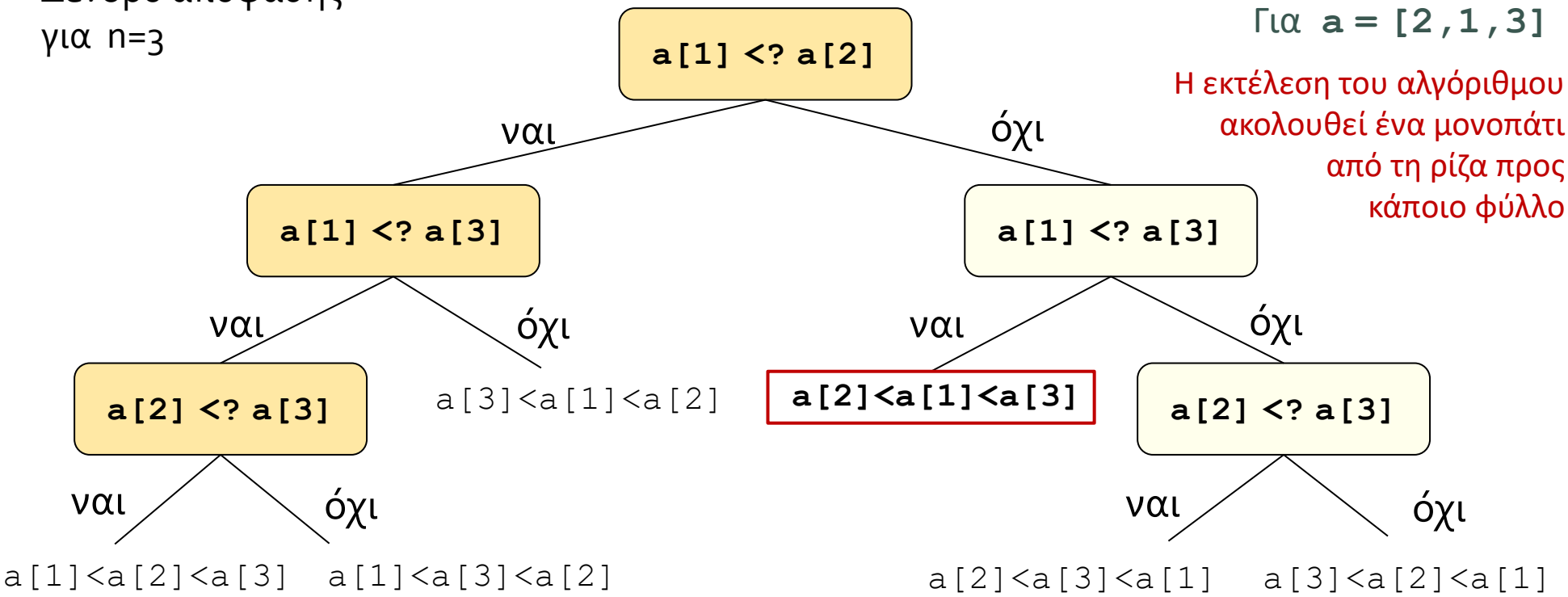
Για `a = [1,2,3]`

Η εκτέλεση του αλγόριθμου ακολουθεί ένα μονοπάτι από τη ρίζα προς κάποιο φύλλο

Κάτω Φράγμα Ταξινόμησης

Παράδειγμα

Δένδρο απόφασης
για n=3

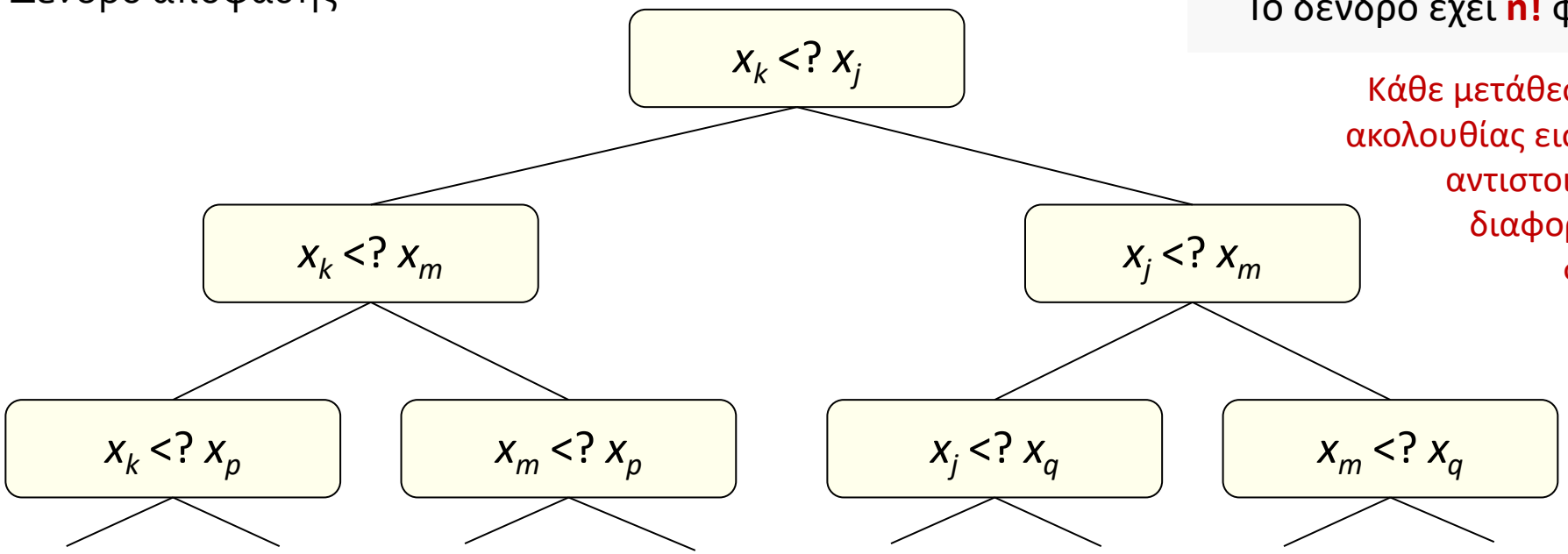


Για `a = [2,1,3]`

Η εκτέλεση του αλγόριθμου ακολουθεί ένα μονοπάτι από τη ρίζα προς κάποιο φύλλο

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης



Το δένδρο έχει **n!** φύλλα

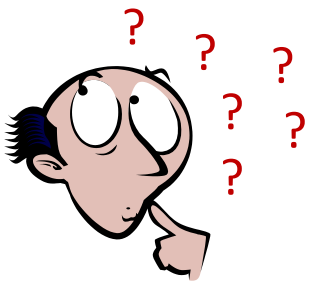
Κάθε μετάθεση της ακολουθίας εισόδου αντιστοιχεί σε διαφορετικό φύλλο

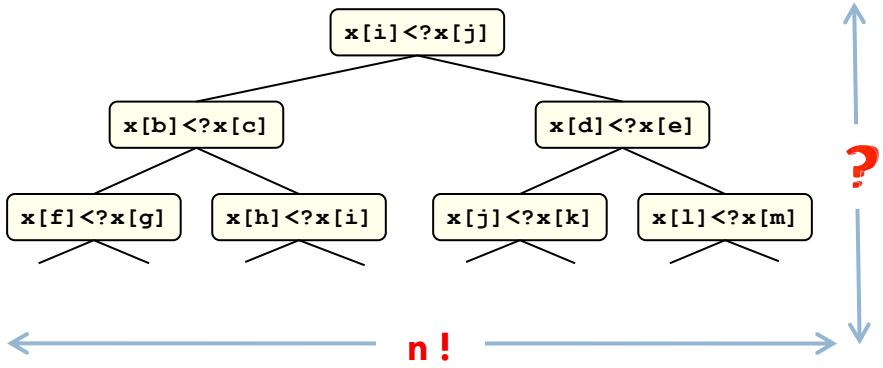
Το *πλήθος των συγκρίσεων* που πραγματοποιεί ένας *οποιουδήποτε αλγόριθμος* ταξινόμησης είναι τουλάχιστον *ίσος με το ύψος του δένδρου* !!!

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης

Το δένδρο έχει **n!** φύλλα





Ύψος δένδρου **?**

Το *πλήθος των συγκρίσεων* που πραγματοποιεί ένας *οποιουδήποτε αλγόριθμος* ταξινόμησης είναι τουλάχιστον *ίσος με το ύψος του δένδρου* !!!

Κάτω Φράγμα Ταξινόμησης

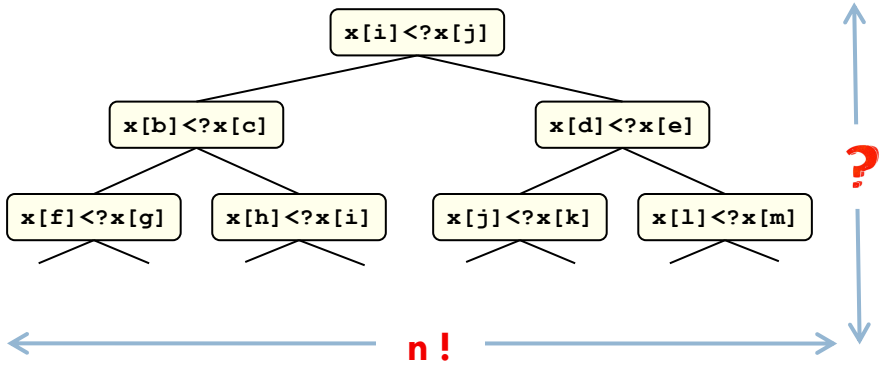
Δένδρο απόφασης

Λήμμα 1. Έστω L το πλήθος των φύλλων ενός δυαδικού δένδρου και έστω h το ύψος του.

Τότε ισχύει $L \leq 2^h$.

Λήμμα 2. Έστω L και h του Λήμματος 1.

Τότε ισχύει $h \geq \lceil \log L \rceil$.



Ύψος δένδρου **?**

Λήμμα 3. Για δεδομένο **n**, το δένδρο απόφασης για οποιοδήποτε αλγόριθμο ταξινόμησης με συγκρίσεις των στοιχείων του ***έχει ύψος τουλάχιστον*** $\lceil \log(n!) \rceil$.

Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης

Log(n!) ?

$$(n/2)^{n/2} \leq$$

$n/2$ ακέραιοι με τιμή $\geq n/2$

$$n! = 1 \times 2 \times 3 \times \ldots \times n/2 \times \ldots \times n$$

$$\leq n^n$$

n ακέραιοι με τιμή $\leq n$

$$n/2 \log(n/2) \leq \log(n!) \leq n \log n$$

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

Το δένδρο έχει **n!** φύλλα
και ύψος **log(n!)**

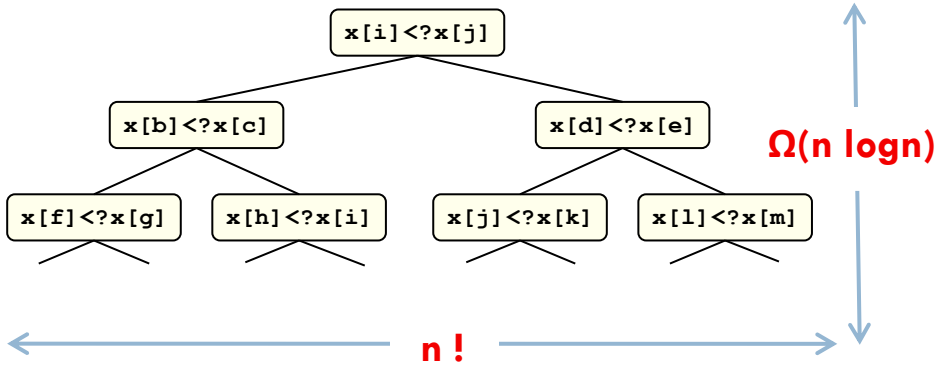
Κάτω Φράγμα Ταξινόμησης

Δένδρο απόφασης

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

n* log *n

συγκρίσεις.



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

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

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

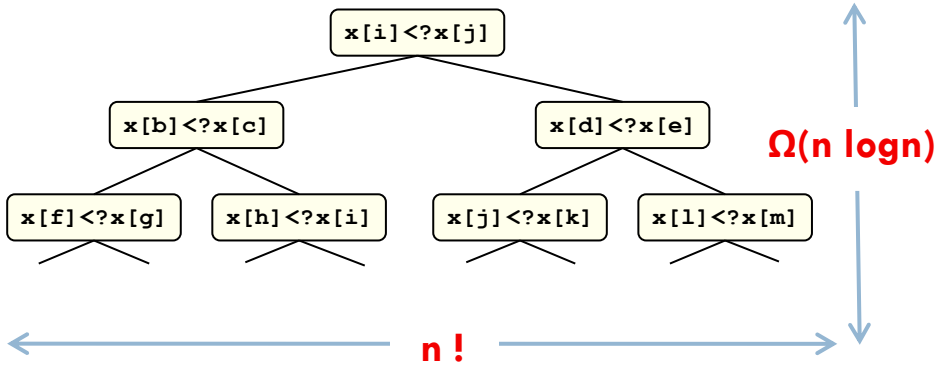
Κάτω Φράγμα Ταξινόμησης

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: $\Omega(n \log n)$

Άνω και Κάτω Φράγματα Ταξινόμησης

Πρόβλημα Ταξινόμησης

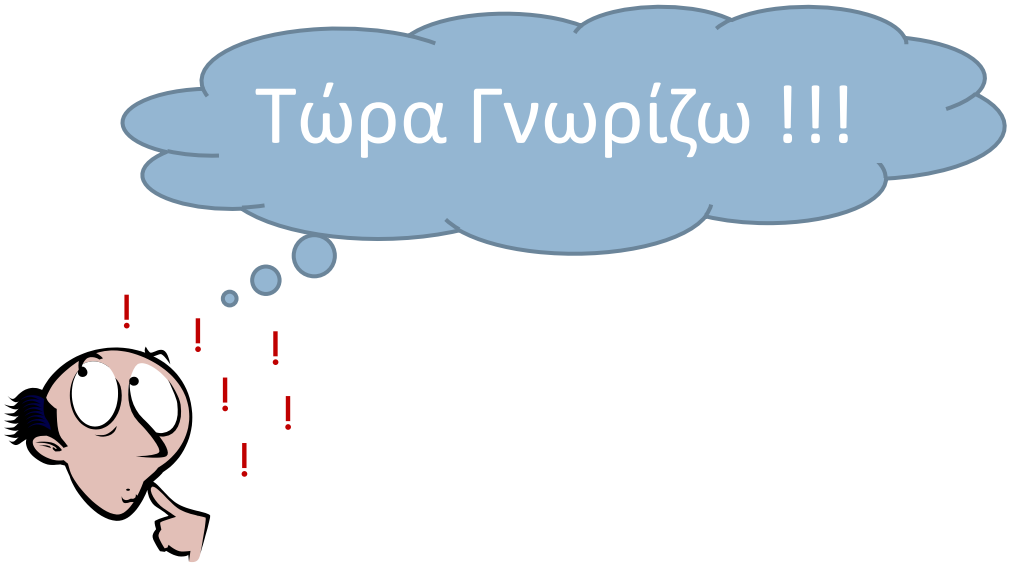
- Η πολυπλοκότητα του Αλγόριθμου **Merge-Sort** είναι **O(n logn)**
- Το ελάχιστο πλήθος Βασικών Πράξεων που απαιτούνται για την επίλυση του **Προβλήματος της Ταξινόμησης** είναι **Ω(n logn)**

Άνω Φράγμα O(n logn) ↓

Κάτω Φράγμα Ω(n logn) ↑

Πολυπλοκότητα Αλγόριθμου Merge-Sort

- **Μέση Πολυπλοκότητα Merge-Sort**



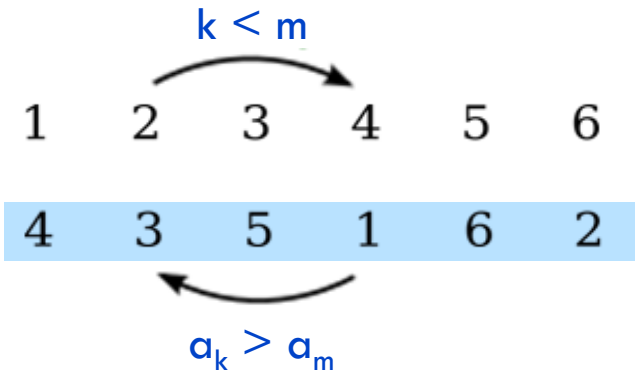

```
Algorithm Merge-Sort(A, left, right)
begin
      if (left >= right) return
      mid = (left + right) / 2
      Merge-Sort(A, left, mid)
      Merge-Sort(A, mid+1, right)
      Merge(A, left, mid, right)
end
```

Ασυμπωτική πολυπλοκότητα μέσης περίπτωσης Merge-Sort : $A(n) \in \Theta(n \log n)$

Μέτρηση Αντιστροφών

Μέτρηση Αντιστροφών

- 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 ισχύει: $k < m$ και $a_k > a_m$.



Το ζεύγος (3, 1) είναι μια αντιστροφή

Μέτρηση Αντιστροφών

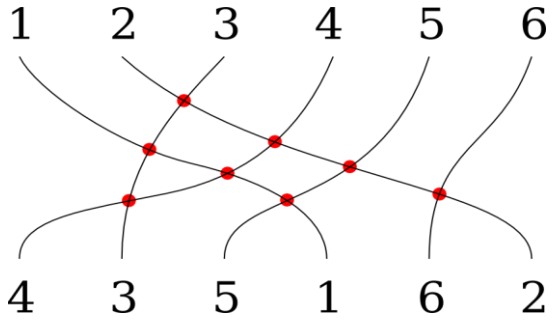
- Έστω μια μετάθεση $A = (a_1, a_2, \ldots, a_n)$ του συνόλου $N_n = \{1, 2, \ldots, n\}$.
- Ένα ζεύγος στοιχείων (a_k, a_m) της μετάθεσης A ονομάζεται *αντιστροφή* (inversion) εάν ισχύει: $k < m$ και $a_k > a_m$.
- Για παράδειγμα, οι αντιστροφές της μετάθεσης

  $A = (4, 3, 5, 1, 6, 2)$

  είναι τα ζεύγη

  (4, 3) (4, 1) (4, 2) (3, 1) (3, 2)

  (5, 1) (5, 2) (6, 2)



Μέτρηση Αντιστροφών

- 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 ισχύει: $k < m$ και $a_k > a_m$.

Αλγόριθμος

Σχεδιάστε αλγόριθμο ο οποίος θα δέχεται ως είσοδο μία μετάθεση A του συνόλου N_n και θα υπολογίζει το πλήθος των αντιστροφών της.

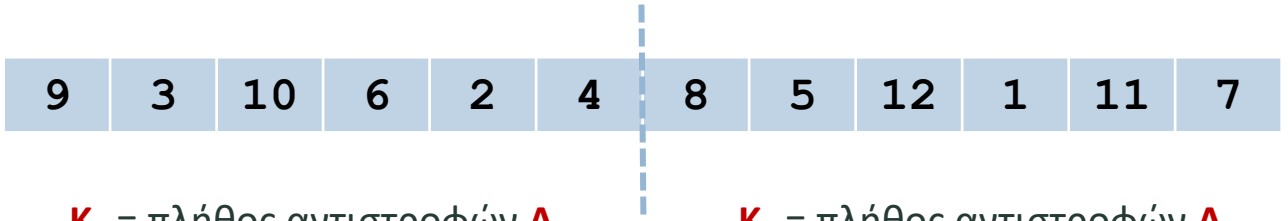
Προφανής αλγόριθμος: $W(n) \in \Theta(n^2)$

Θέλουμε αλγόριθμο με πολυπλοκότητα : $W(n) \in \Theta(n \log n)$

Divide & Conquer

Μέτρηση Αντιστροφών

- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



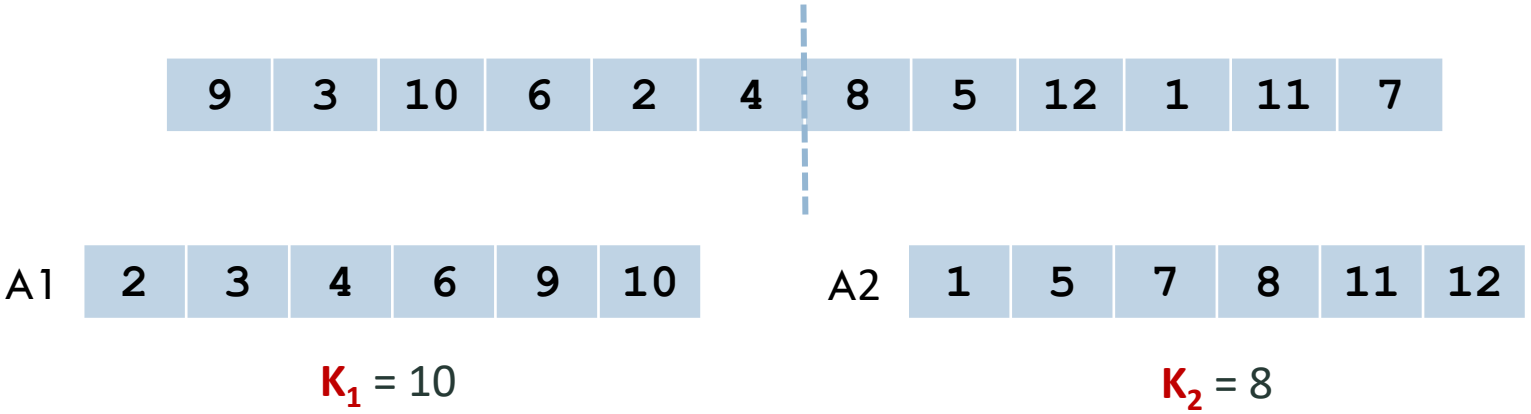
K_1 = πλήθος αντιστροφών A_1 K_2 = πλήθος αντιστροφών A_2

K_{12} = πλήθος αντιστροφών (x, y) **με** $x \in A_1$ και $y \in A_2$

Πλήθος αντιστροφών $K_1 + K_2 + K_{12}$

Μέτρηση Αντιστροφών

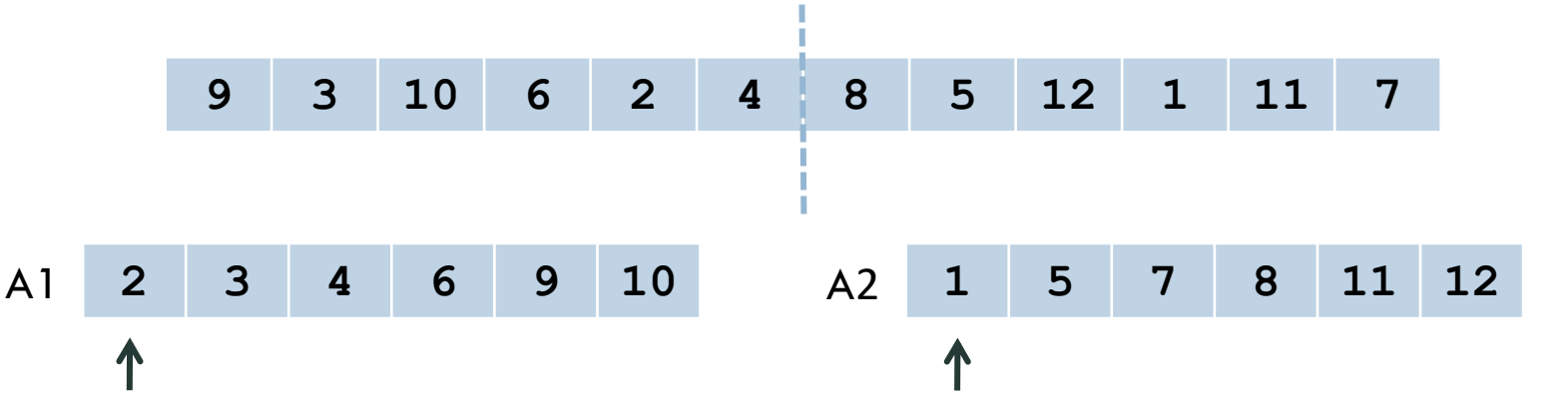
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



Πλήθος αντιστροφών $K_1 + K_2 + K_{12}$

Μέτρηση Αντιστροφών

- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα

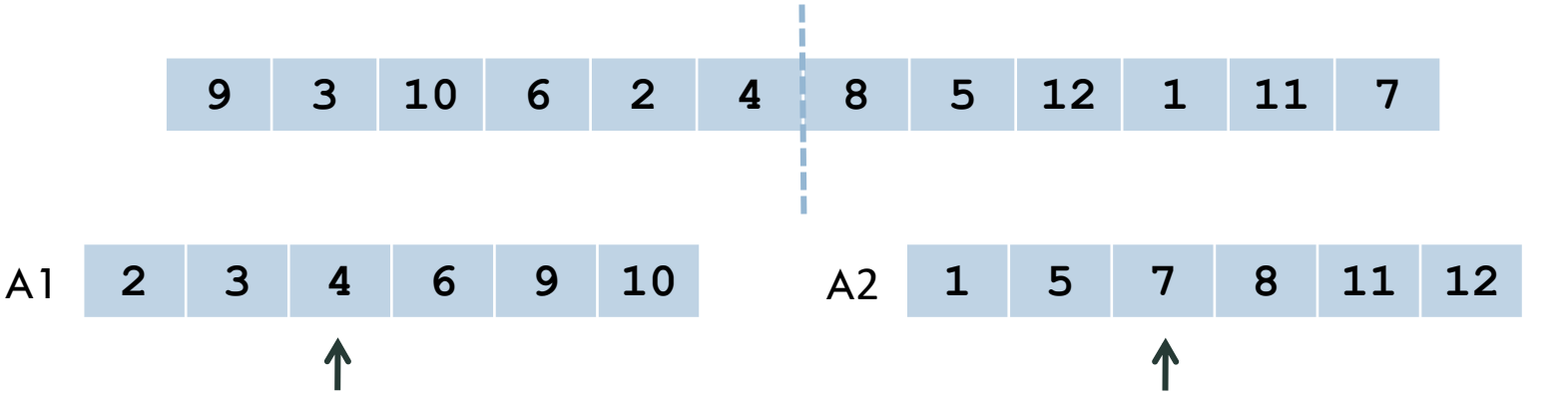


6 αντιστροφές όταν A1[1] > A2[1]

$K_1 = 10$ $K_2 = 8$

Μέτρηση Αντιστροφών

- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



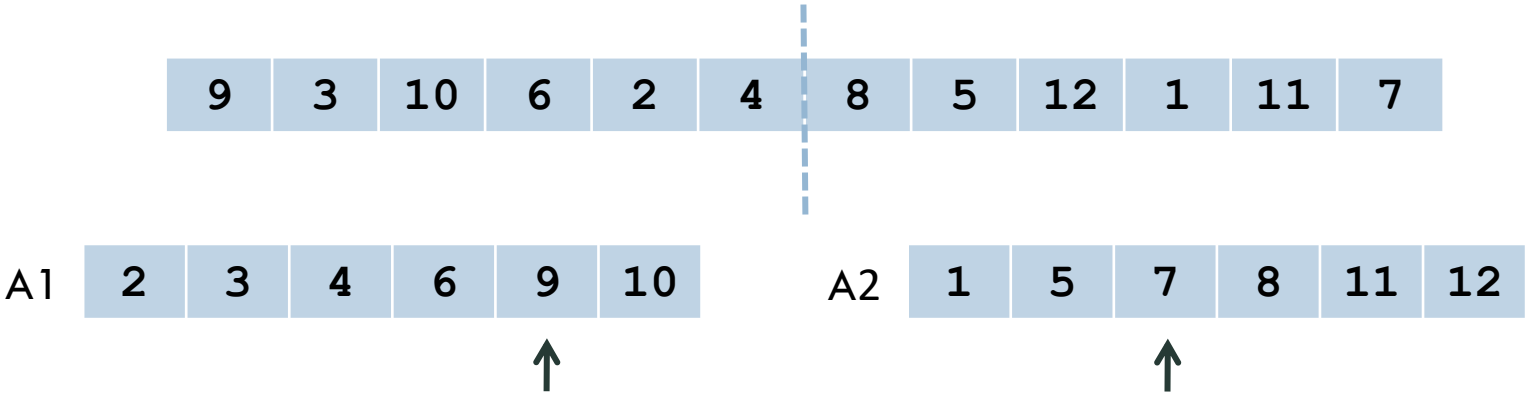
6 αντιστροφές όταν A1[1] > A2[1]

0 >> A1[3] < A2[3]

$K_1 = 10$ $K_2 = 8$

Μέτρηση Αντιστροφών

- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



6 αντιστροφές όταν A1[1] > A2[1]

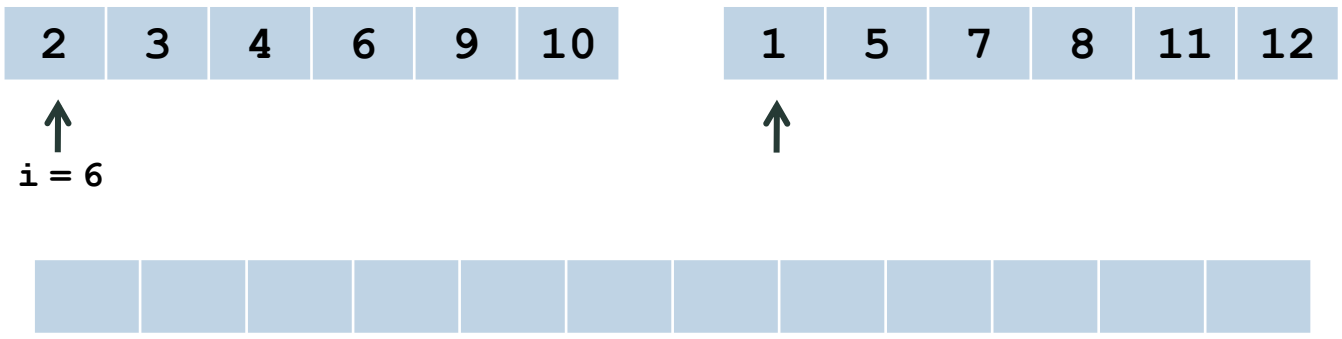
0 >> A1[3] < A2[3]

2 >> A1[5] > A2[3]

$K_1 = 10$ $K_2 = 8$

Μέτρηση Αντιστροφών

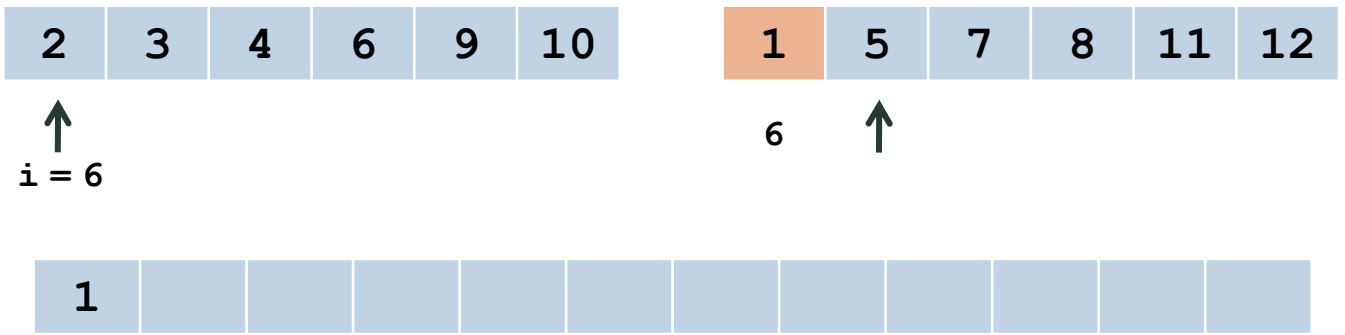
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = ?$

Μέτρηση Αντιστροφών

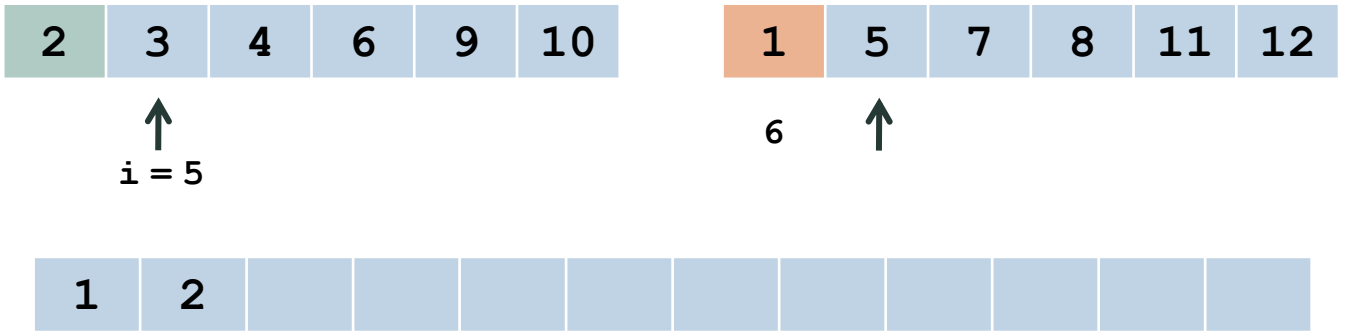
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6$

Μέτρηση Αντιστροφών

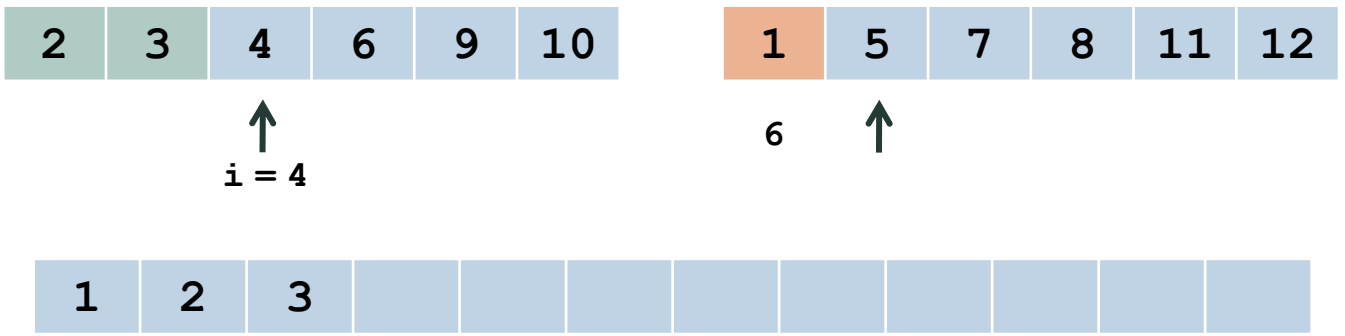
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6$

Μέτρηση Αντιστροφών

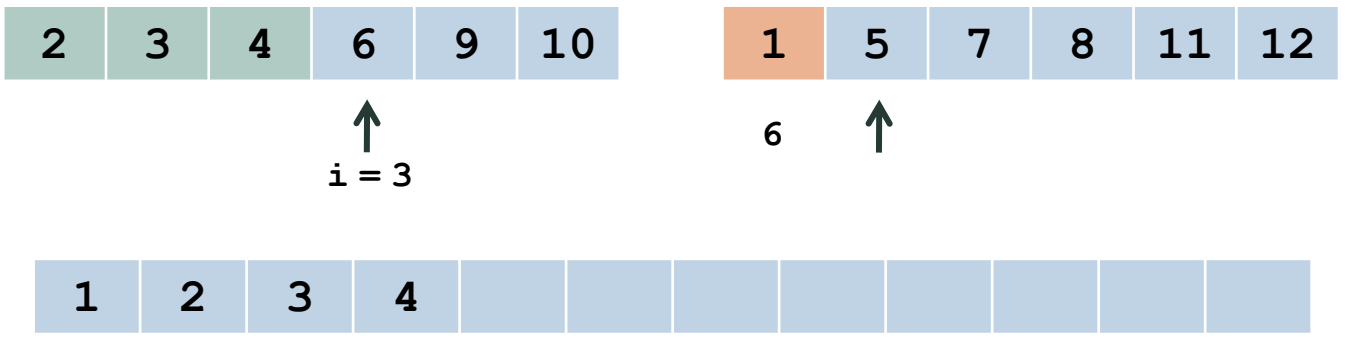
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6$

Μέτρηση Αντιστροφών

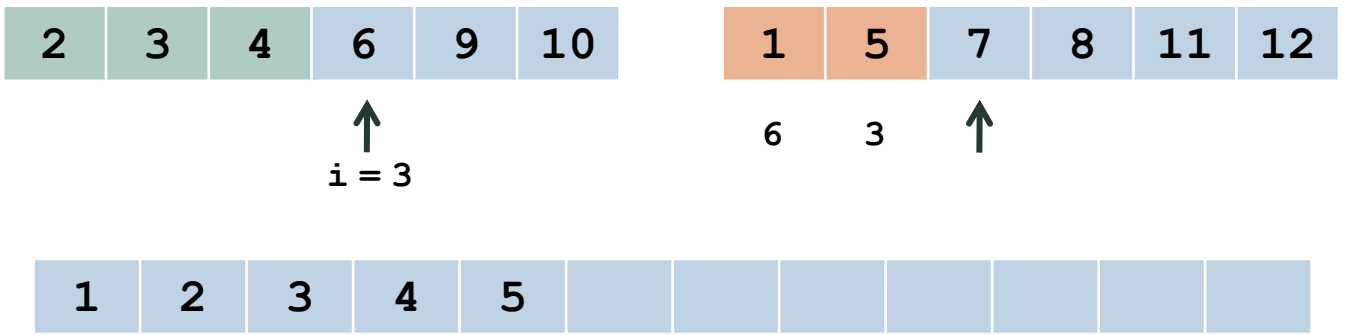
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6$

Μέτρηση Αντιστροφών

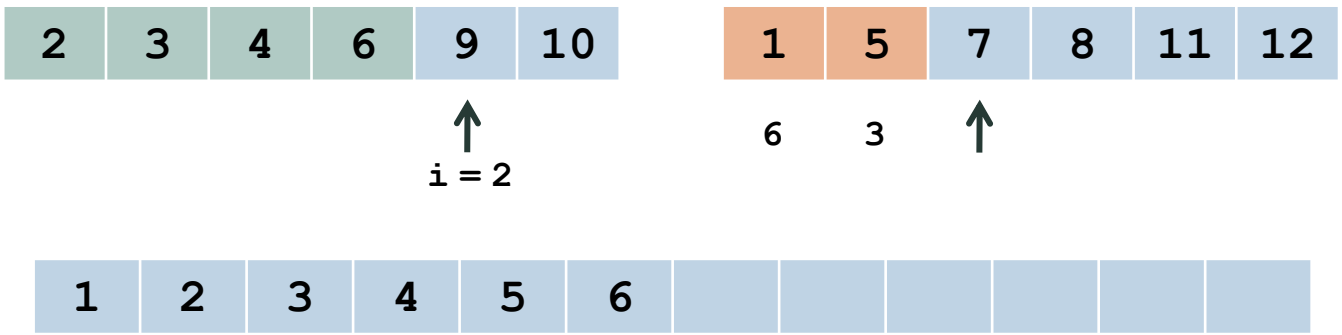
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3$

Μέτρηση Αντιστροφών

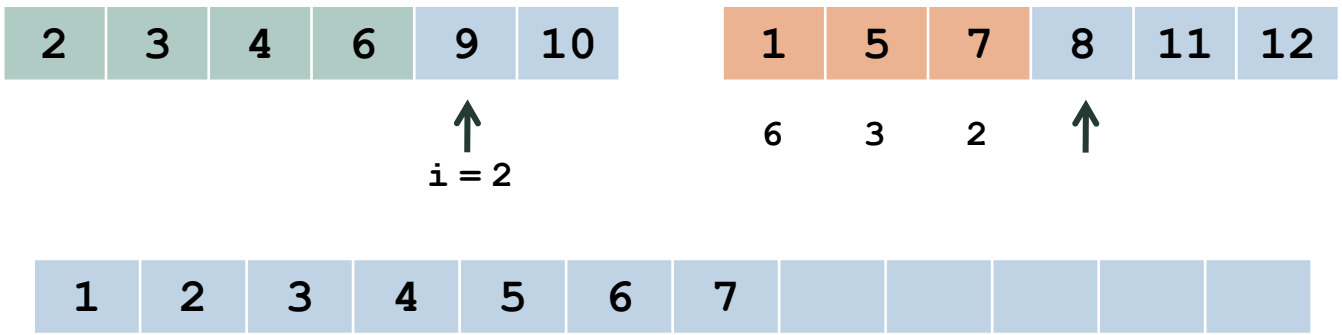
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3$

Μέτρηση Αντιστροφών

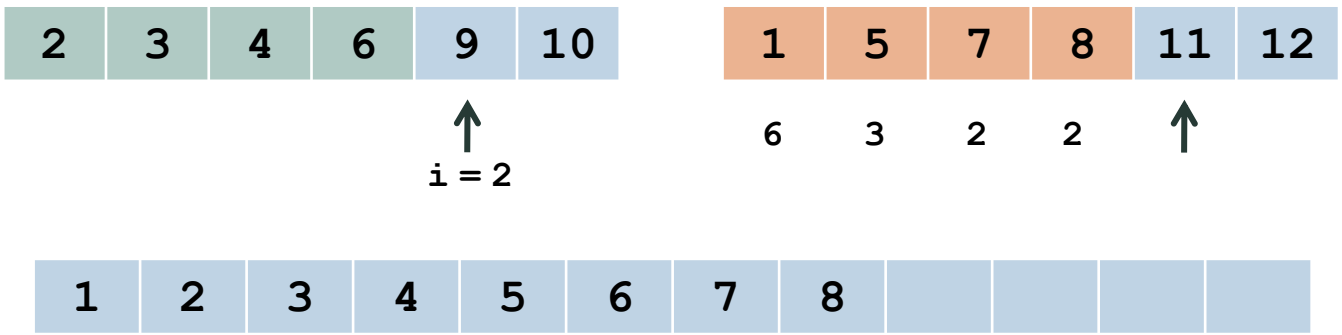
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3 + 2$

Μέτρηση Αντιστροφών

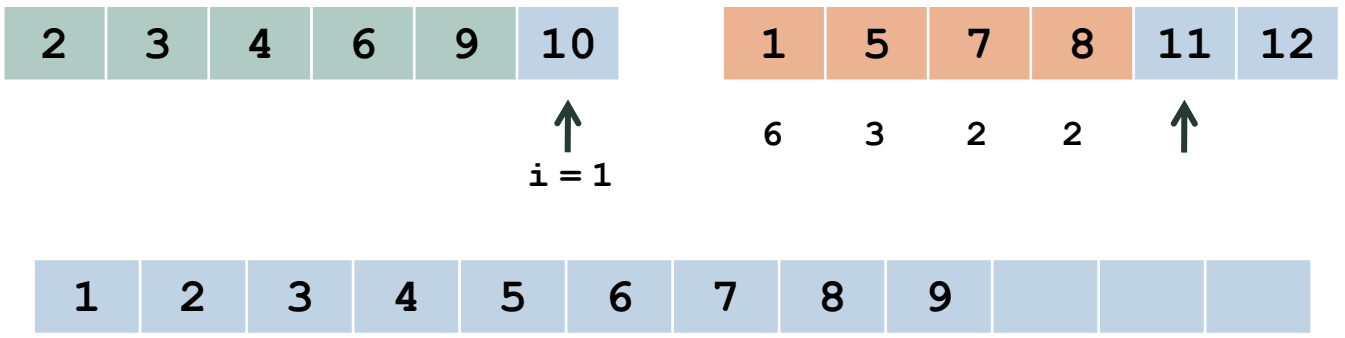
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3 + 2 + 2$

Μέτρηση Αντιστροφών

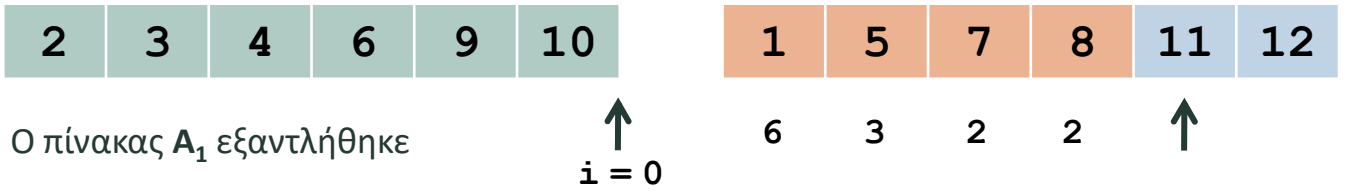
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3 + 2 + 2$

Μέτρηση Αντιστροφών

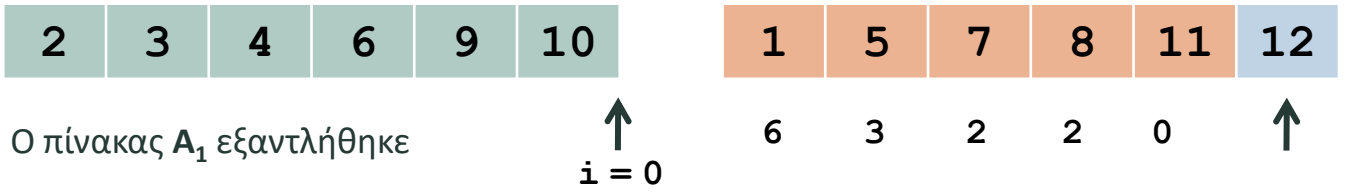
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3 + 2 + 2$

Μέτρηση Αντιστροφών

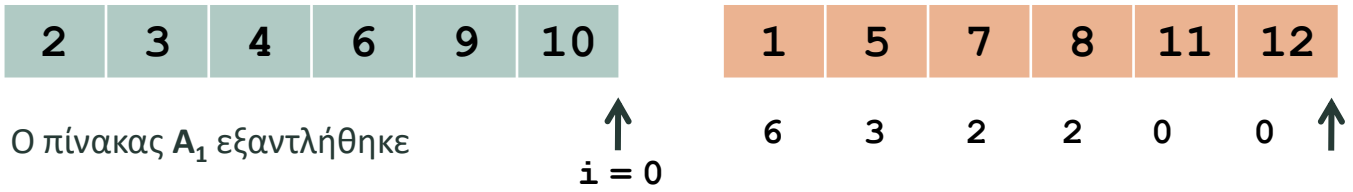
- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα



$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3 + 2 + 2$

Μέτρηση Αντιστροφών

- Παρουσίαση της ιδέας του Αλγόριθμου με ένα παράδειγμα

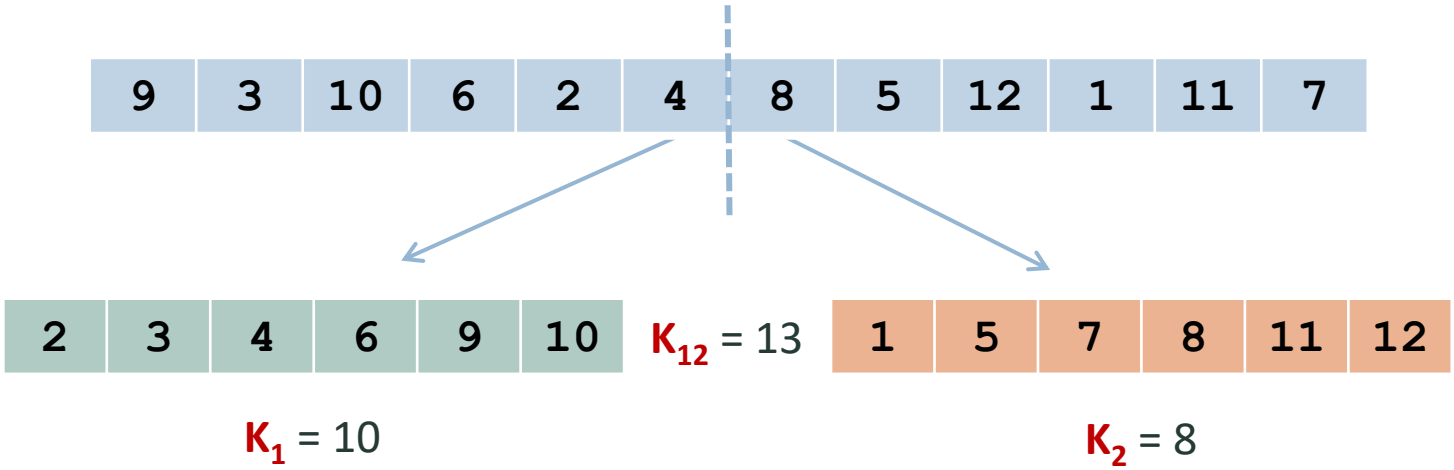


$K_1 = 10$ $K_2 = 8$ $K_{12} = 6 + 3 + 2 + 2 = 13$

Πολυπλοκότητα υπολογισμού των **K** αντιστροφών : $W(n) \in \Theta(n)$

Μέτρηση Αντιστροφών

- Αλγόριθμος Διαίρει-και-Βασίλευε



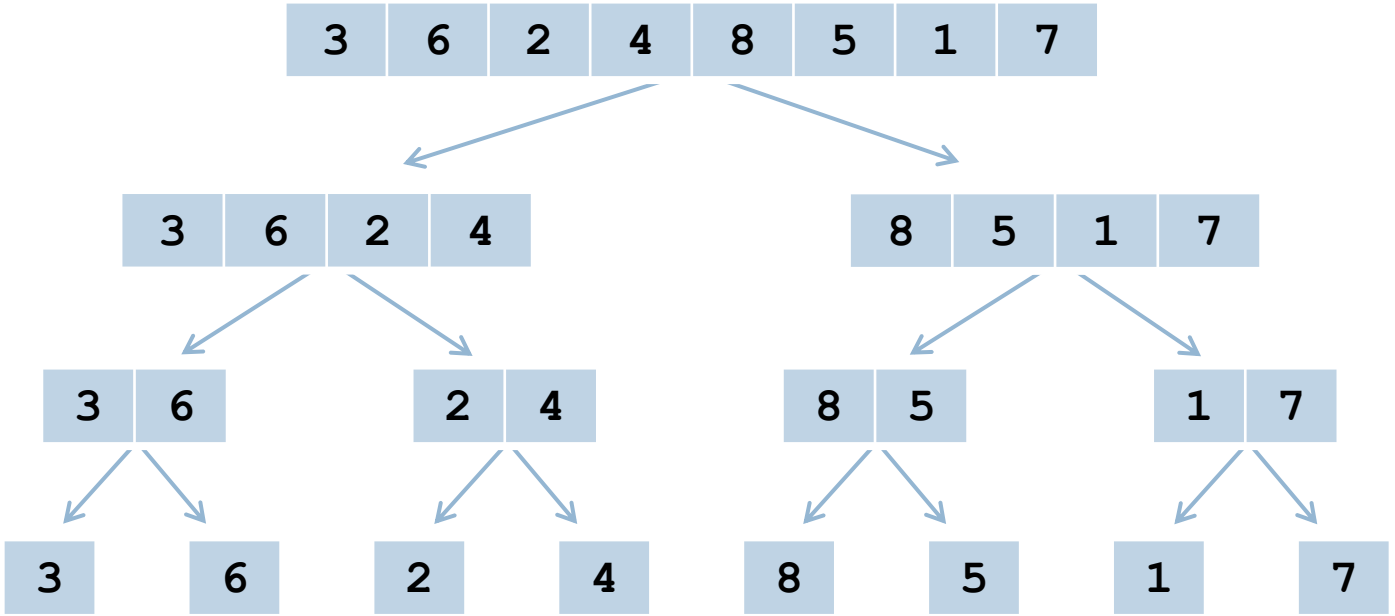
Πλήθος αντιστροφών **31**

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Divide & Conquer

Μέτρηση Αντιστροφών

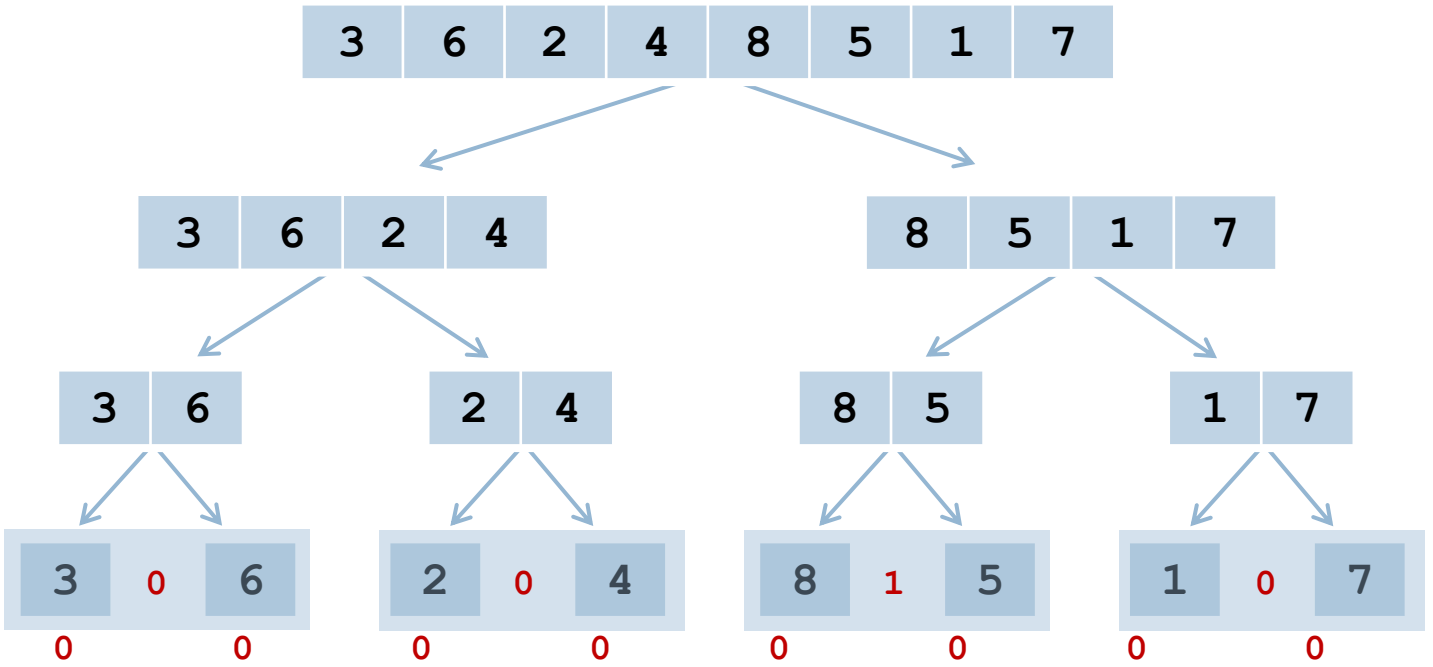
- Παράδειγμα Εκτέλεσης του αλγόριθμου



Πλήθος αντιστροφών ?

Μέτρηση Αντιστροφών

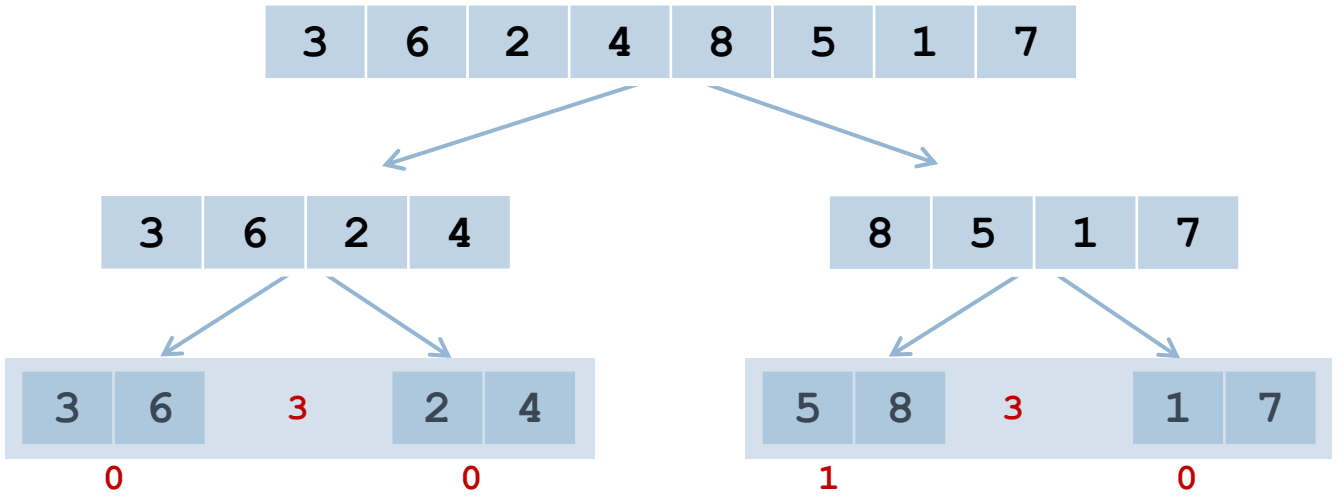
- Παράδειγμα Εκτέλεσης του αλγόριθμου



Πλήθος αντιστροφών **?**

Μέτρηση Αντιστροφών

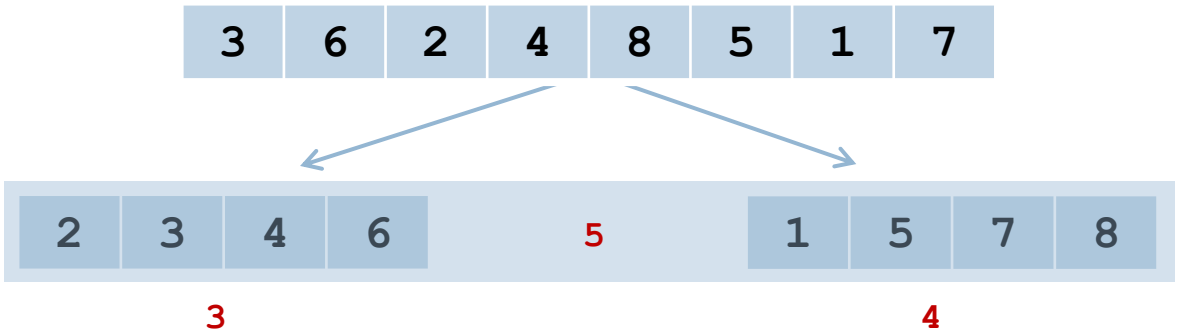
- Παράδειγμα Εκτέλεσης του αλγόριθμου



Πλήθος αντιστροφών ?

Μέτρηση Αντιστροφών

- Παράδειγμα Εκτέλεσης του αλγόριθμου



Πλήθος αντιστροφών ?

Μέτρηση Αντιστροφών

- Παράδειγμα Εκτέλεσης του αλγόριθμου

3	6	2	4	8	5	1	7

12

Πράγματι:

- (3, 2) (3, 1)
- (6, 2) (6, 4) (6, 5) (6, 1)
- (2, 1)
- (4, 1)
- (8, 5) (8, 1) (8, 7)
- (5, 1)

Πλήθος αντιστροφών **12**

Μέτρηση Αντιστροφών

- Παράδειγμα Εκτέλεσης του αλγόριθμου

3	6	2	4	8	5	1	7

- ✓ Τεχνική Διαίρει-και-Βασίλευε
- ✓ Χρήση Ταξινόμησης (Merge-Sort)
- ✓ Ορθότητα εύκολη
- ✓ Πολυπλοκότητα Merge-Sort

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Divide & Conquer

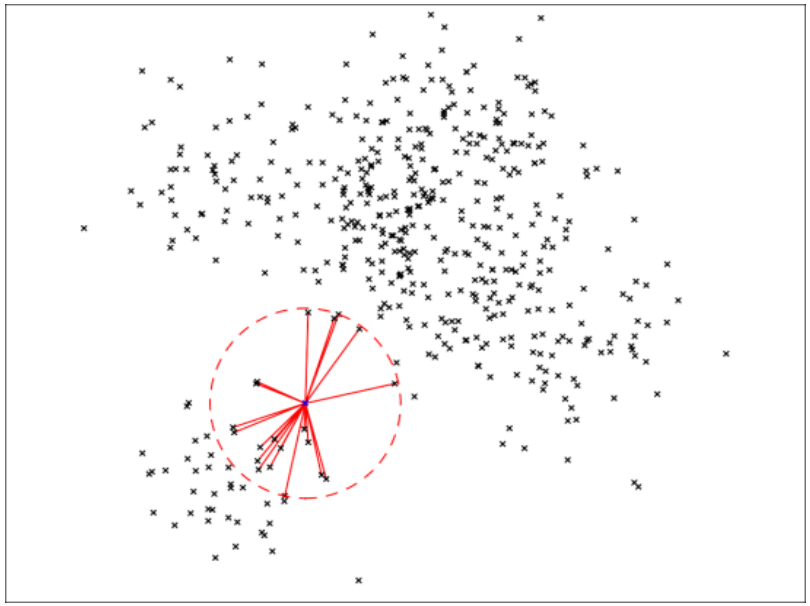
Μέτρηση Αντιστροφών

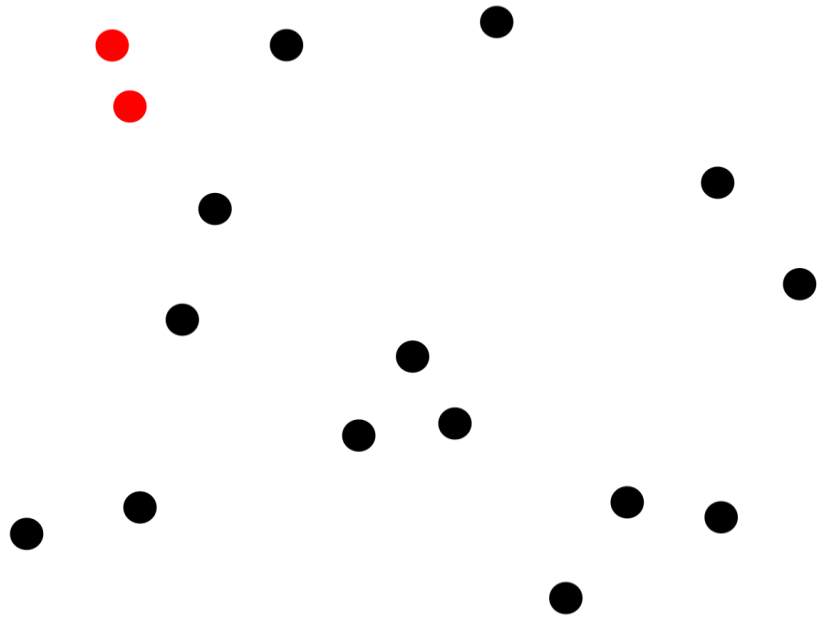
Αλγόριθμος $O(n^2)$ σε σχέση με έναν $O(n \log n)$

n	logn	n^2	n logn
16	4	256	64
1024	10	1.048.576	10.240
2048	11	4.194.304	22.528
4096	12	16.777.216	49.152
1048576	20	1.099.511.627.776	20.971.520
33554432	25	1.125.899.906.842.624	838.860.800

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Πλησιέστερο Ζεύγος Σημείων





Η εταιρεία MS-Control

Η εταιρεία MS-Control με κύρια δραστηριότητα σε υπηρεσίες «παγκόσμιας ασφαλούς πλοήγησης», διαθέτει ένα ισχυρό δορυφορικό σύστημα το οποίο σε σταθερό χρόνο μπορεί να εντοπίσει όλα τα μέσα πολιτικής αερομεταφοράς και πλεύσης

$$S_1, S_2, \ldots, S_n \ (n \geq 1)$$

τα οποία βρίσκονται σε οπουδήποτε σημείο του κόσμου και να υπολογίσει τις συντεταγμένες

$$S_i = (x_i, y_i)$$

της θέσης τους.

Η εταιρεία MS-Control

Η εταιρεία θέλει να δημιουργήσει ένα σύστημα το οποίο να εντοπίζει γρήγορα ένα ζεύγος αντικείμενων (δύο αεροπλάνα ή πλοία)

$$(S_p, S_q)$$

τα οποία στο χρονικό διάστημα $[t_1, t_2]$ απέχουν μεταξύ τους την μικρότερη απόσταση

$$d(S_p, S_q) = \min \{d(S_i, S_j) \mid 1 \le i, j \le n\}$$

μεταξύ όλων των ζευγών των n αντικειμένων που εντοπίζει, έτσι ώστε εάν

$$d(S_p, S_q) < Dmin$$

να στέλνει «SOS» για αποφυγή πιθανής σύγκρουσης !!!

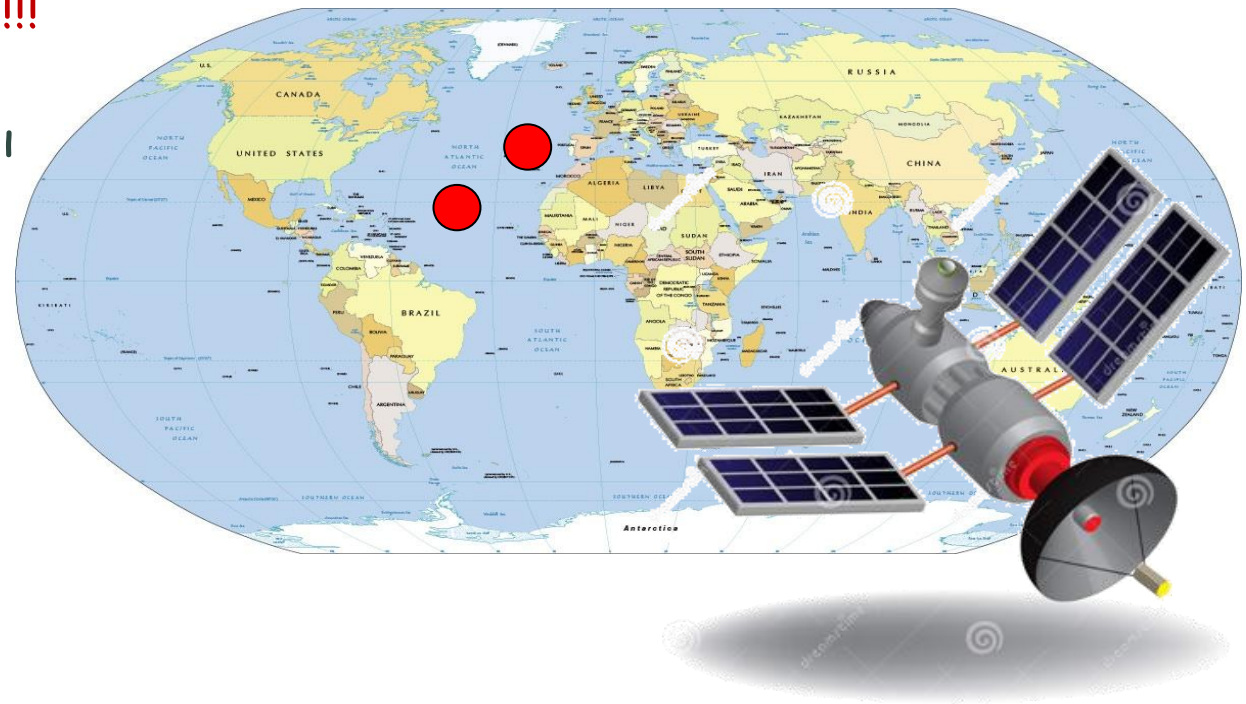
Η εταιρεία MS-Control

Στο λογισμικό του συστήματος πλοήγησης πρέπει να ενσωματωθεί αλγόριθμος ο οποίος «πολύ γρήγορα» θα υπολογίζει το πλησιέστερο ζεύγος αεροσκαφών ή πλοίων (S_p, S_q) σε οποιοδήποτε σημείο του κόσμου !!!

Καλείστε να τον σχεδιάσετε !!!

Ο αλγόριθμός σας θα πρέπει να είναι :

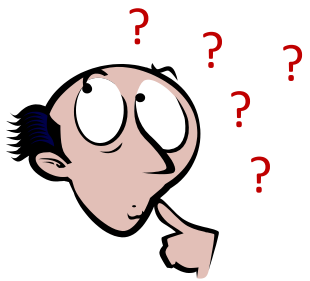
- Σωστός
- Πολύ γρήγορος !!!



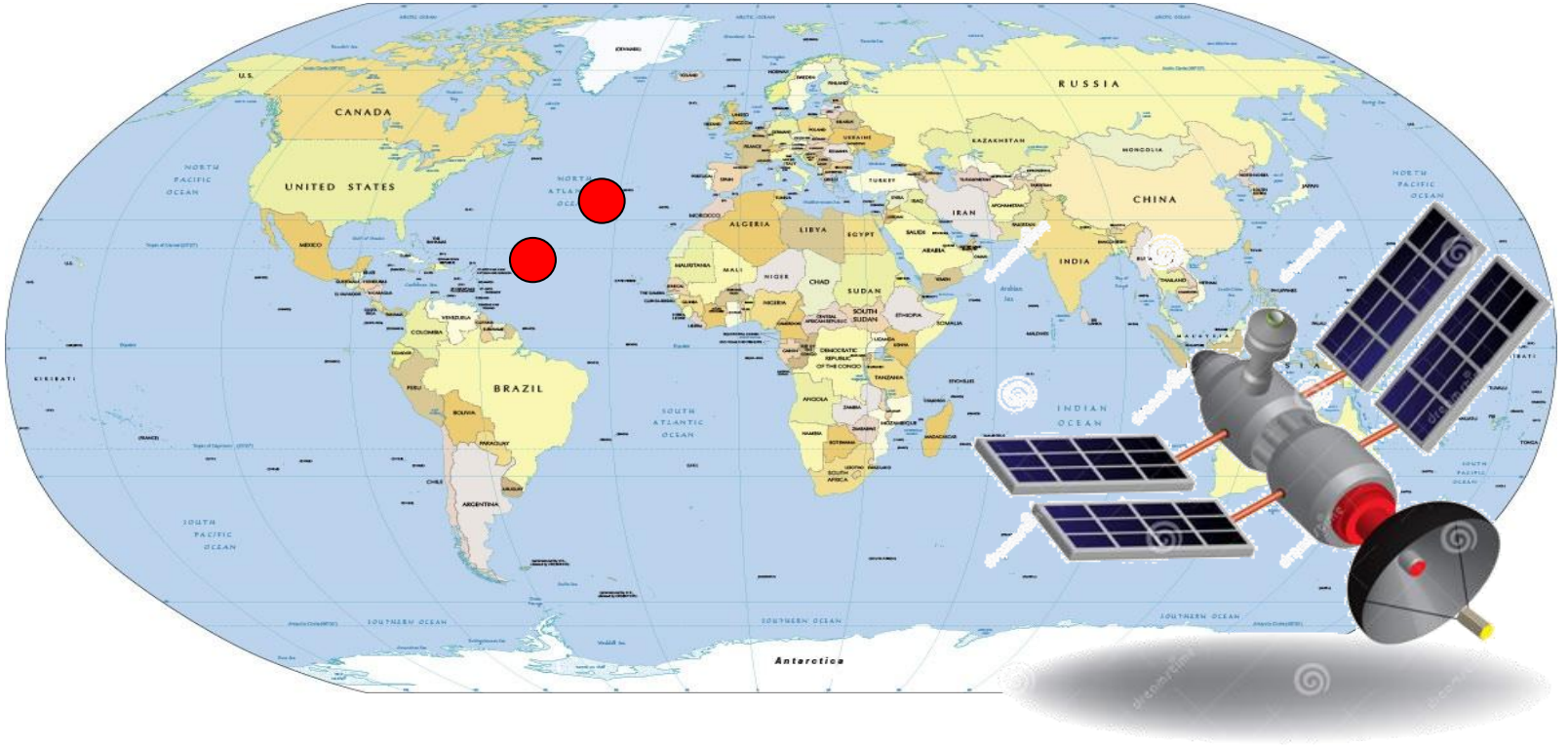
Η εταιρεία MS-Control

Δεν θα θέλατε να σχεδιάσετε !!!

έναν αλγόριθμο πολυπλοκότητας $O(n^2)$, $O(n^3)$ ή μεγαλύτερης, για τη συγκεκριμένη εφαρμογή…



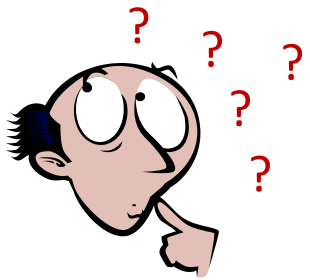
Γιατί ?



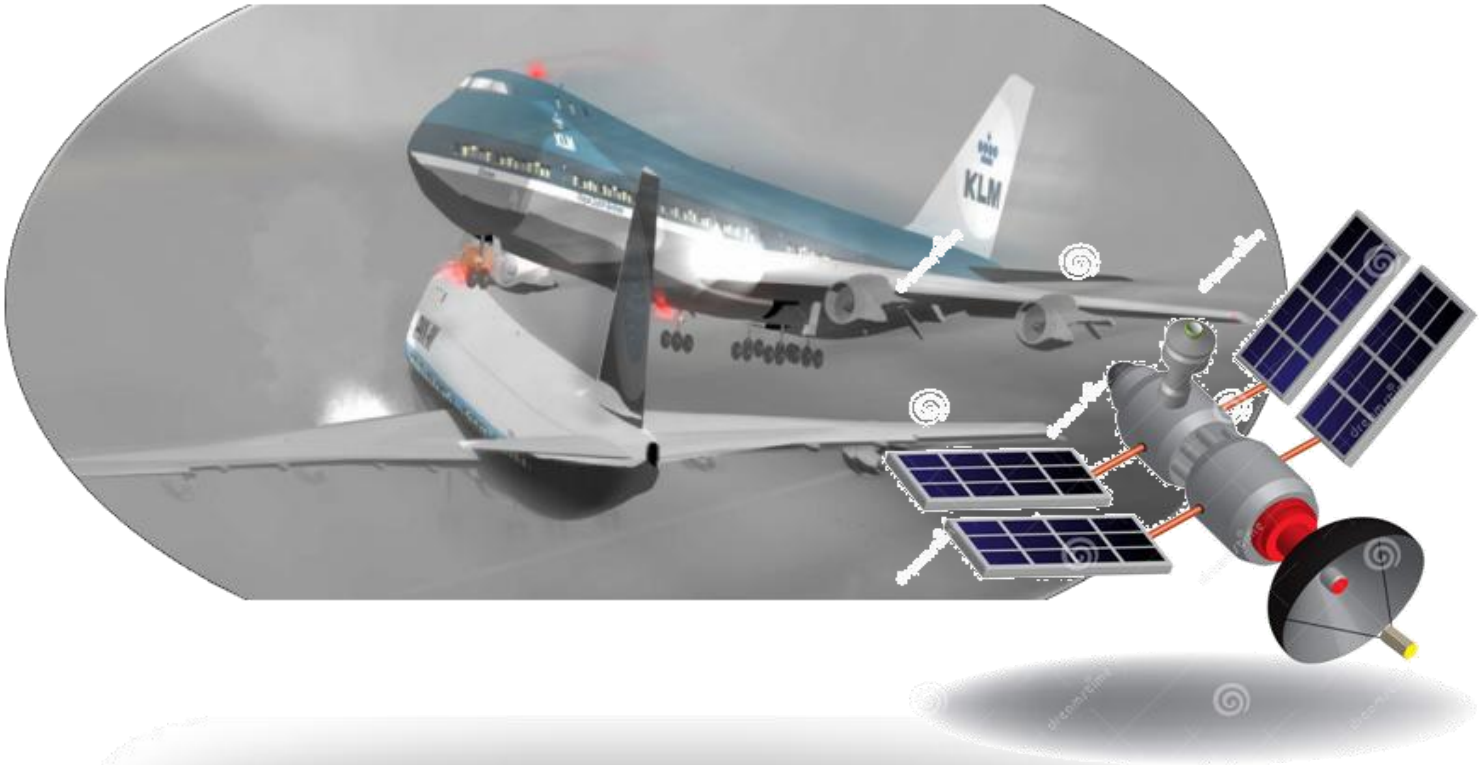
Η εταιρεία MS-Control

Δεν θα θέλατε να σχεδιάσετε !!!

έναν αλγόριθμο πολυπλοκότητας $O(n^2)$, $O(n^3)$ ή μεγαλύτερης, για τη συγκεκριμένη εφαρμογή…



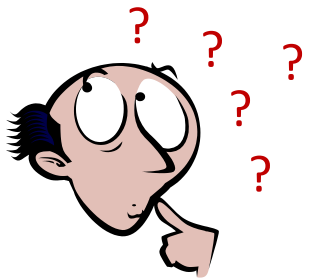
Δεν θα θέλατε να συμβεί αυτό !!!



Η εταιρεία MS-Control

Δεν θα θέλατε να σχεδιάσετε !!!

έναν αλγόριθμο πολυπλοκότητας $O(n^2)$, $O(n^3)$ ή μεγαλύτερης, για τη συγκεκριμένη εφαρμογή…

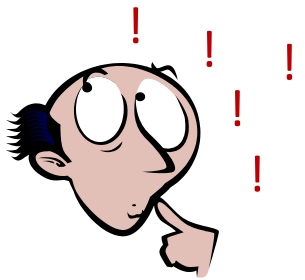


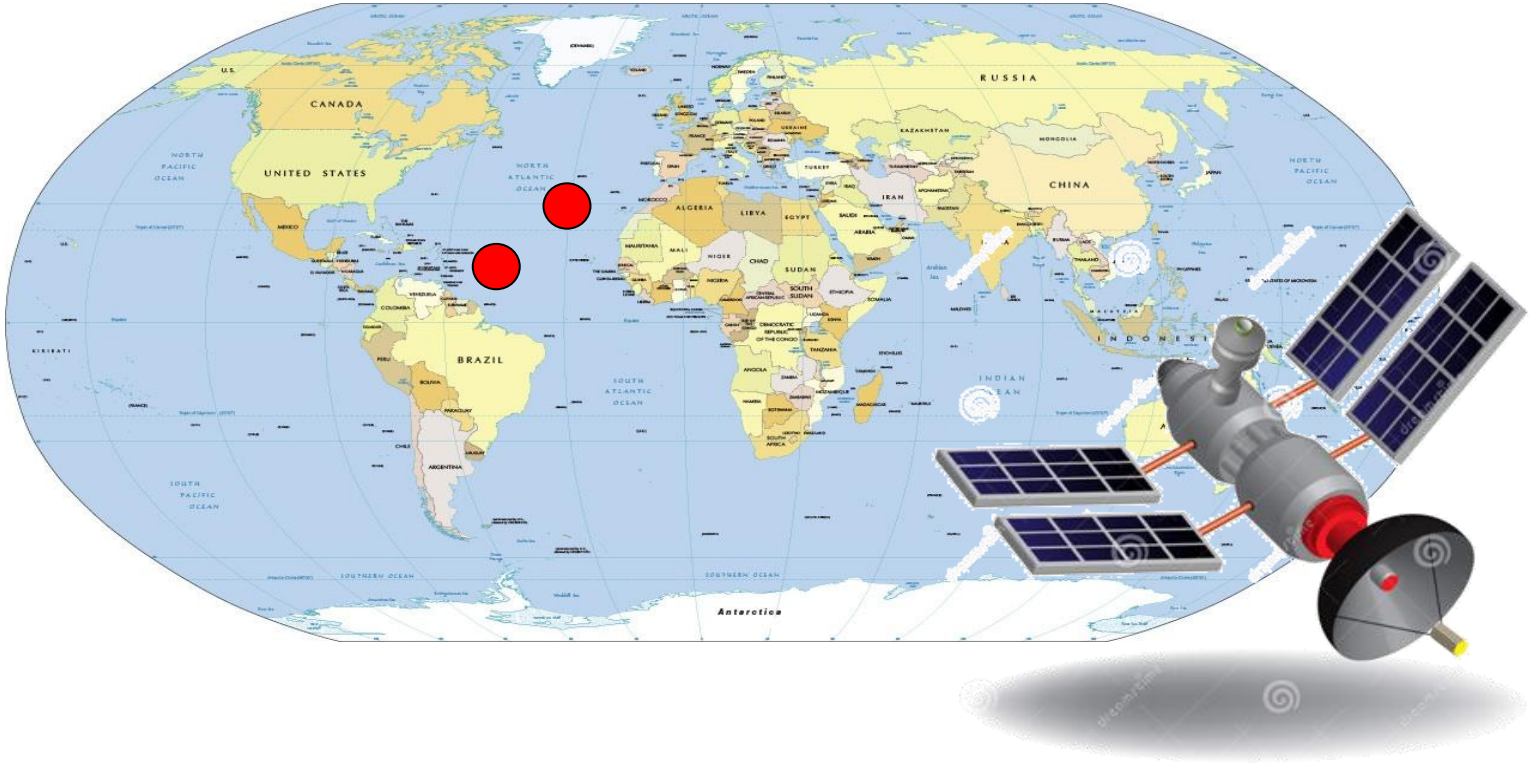
Δεν θα θέλατε να συμβεί αυτό !!!

Η εταιρεία MS-Control

Θέλω έναν αλγόριθμο Σωστό και «Ταχύτατο» !!!

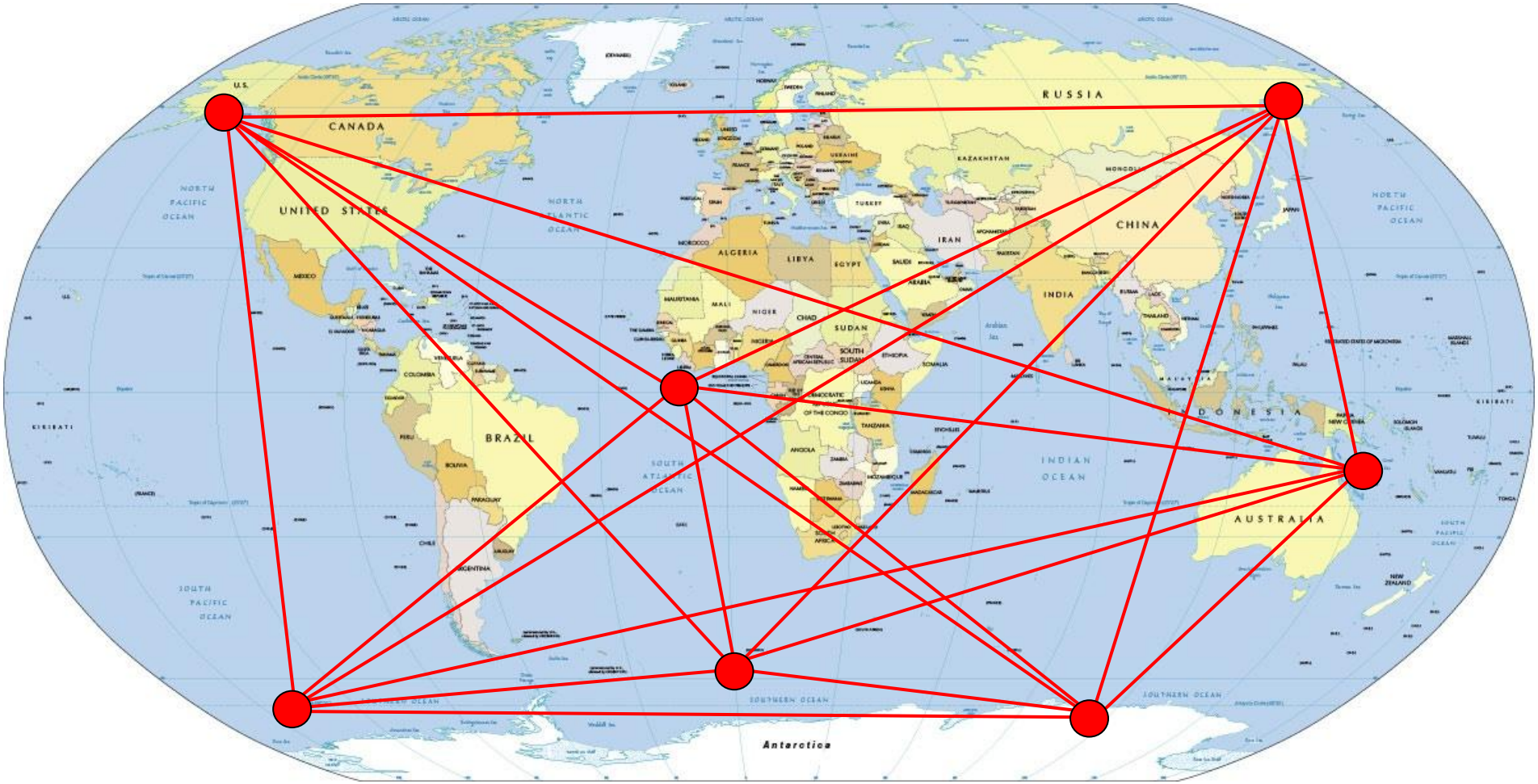
Μελέτη Προβλήματος… και Επινόηση Καλής Τεχνικής και Ιδέας !!!





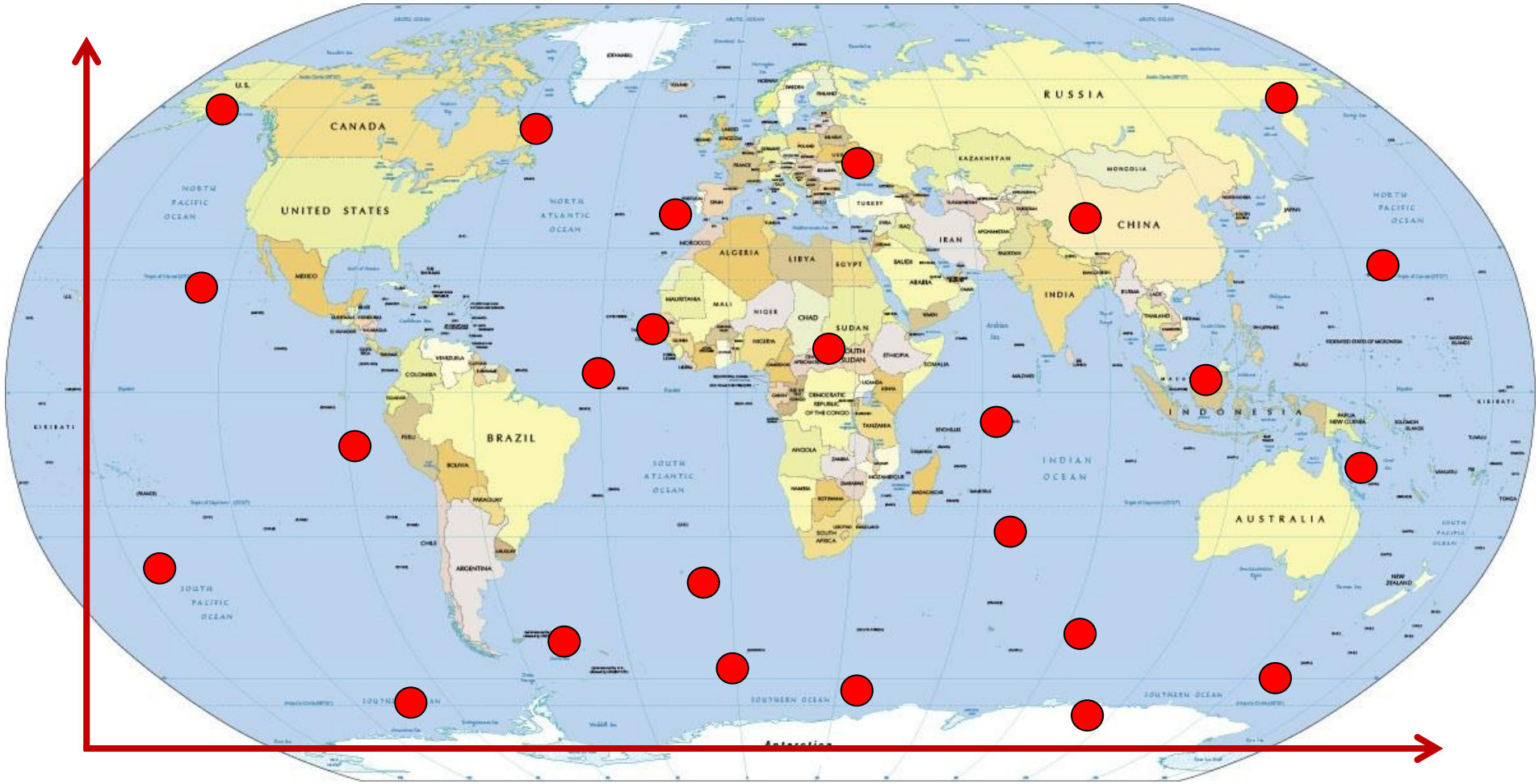
Πλησιέστερο Ζεύγος Σημείων

Ορισμός του Προβλήματος

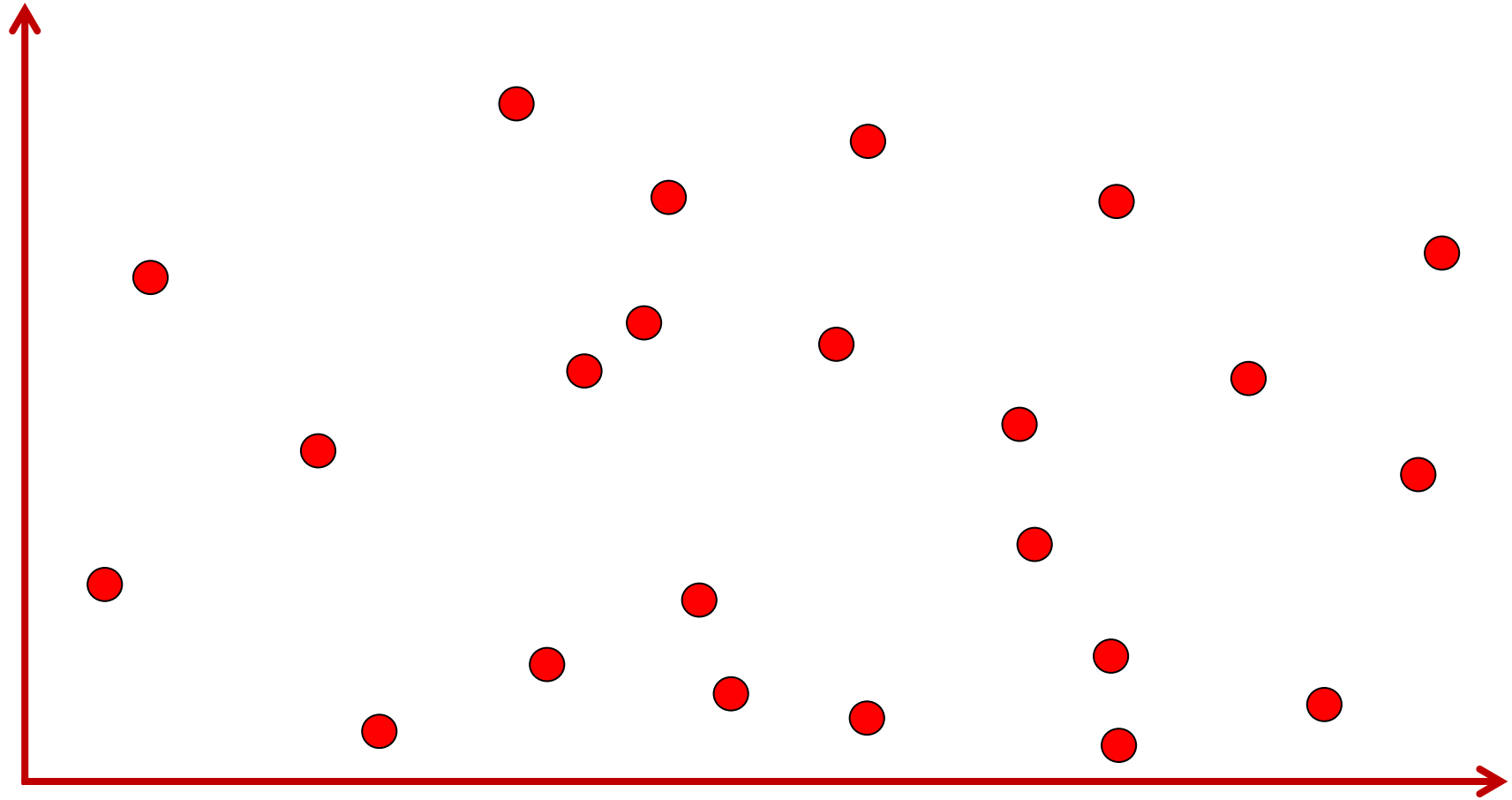


Πλησιέστερο Ζεύγος Σημείων

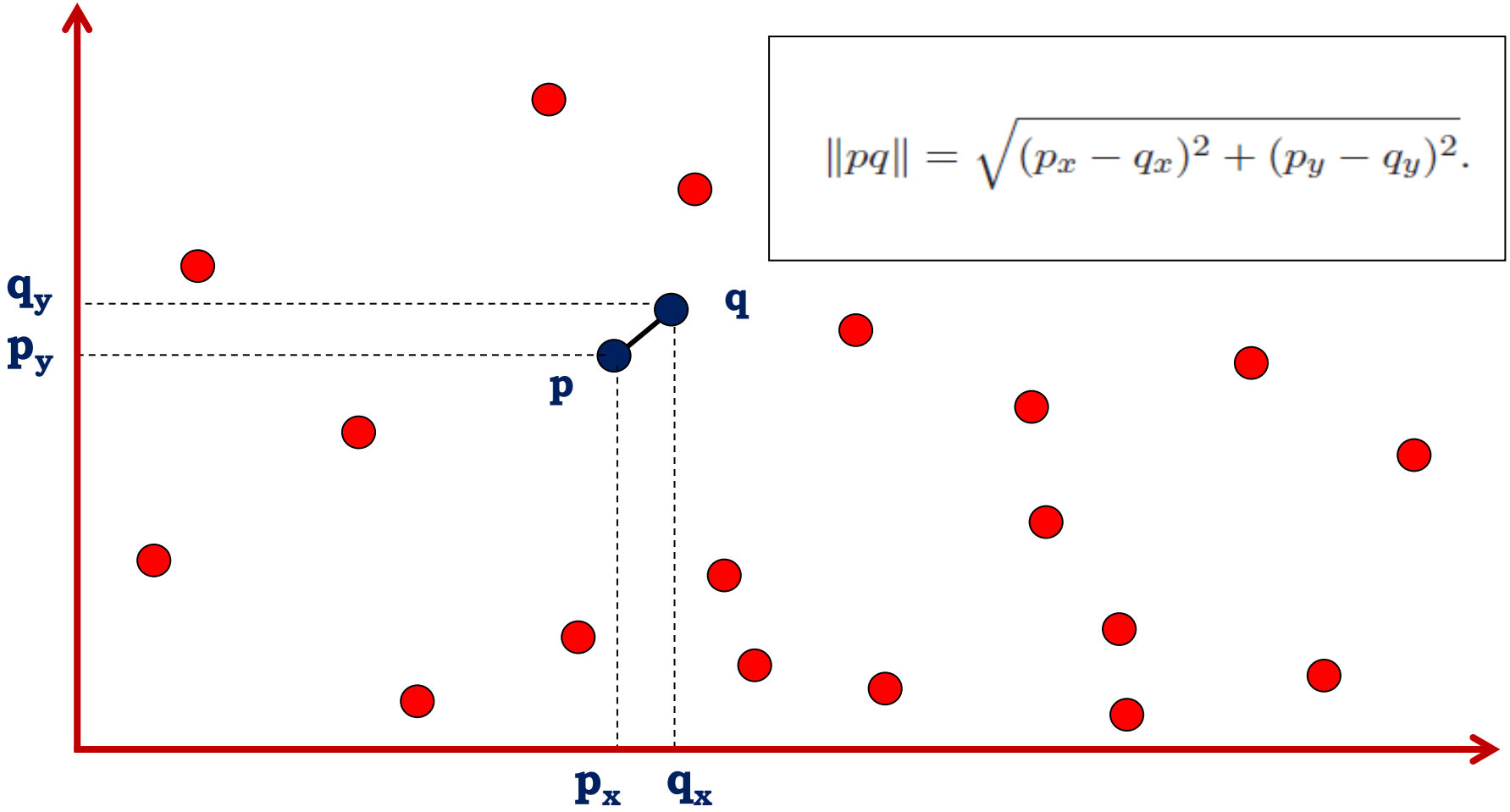
Ορισμός του Προβλήματος



Ορισμός του Προβλήματος



Ορισμός του Προβλήματος



Σχεδίαση Αλγόριθμου

- Είσοδος :

  Οι συντεταμένες n σημείων p_i στο επίπεδο και οι μεταξύ τους αποστάσεις $d(p_i, p_j)$, $1 \leq i, j \leq n$

- Έξοδος :

  Το ζεύγος σημείων (p_k, p_m) με τη μικρότερη μεταξύ τους απόσταση D !!!

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

Υπάρχει προφανής και εύκολος αλγόριθμος πολυπλοκότητας $O(n^2)$

Θέλουμε καλύτερο !!!... πολυπλοκότητας $O(n \log n)$

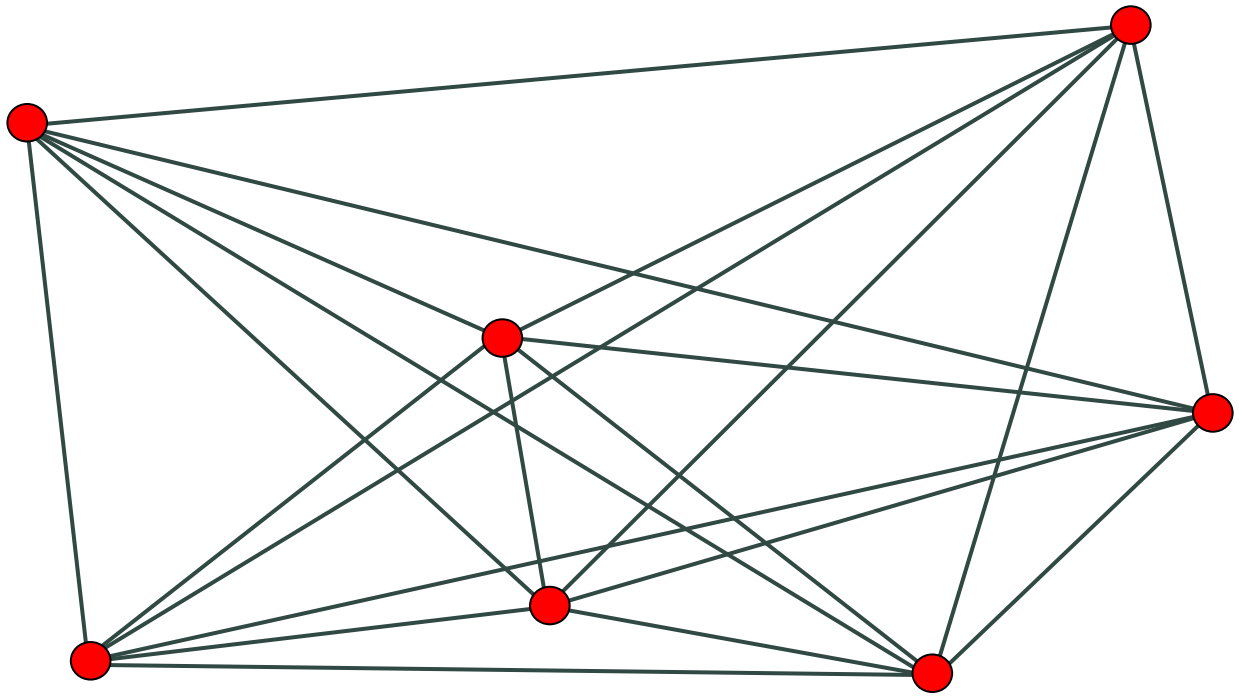
Μπορούμε να σχεδιάσουμε καλύτερο ?

Αλγόριθμος $O(n^2)$

Τετριμμένος… δυσκολίας Λυκείου !!!

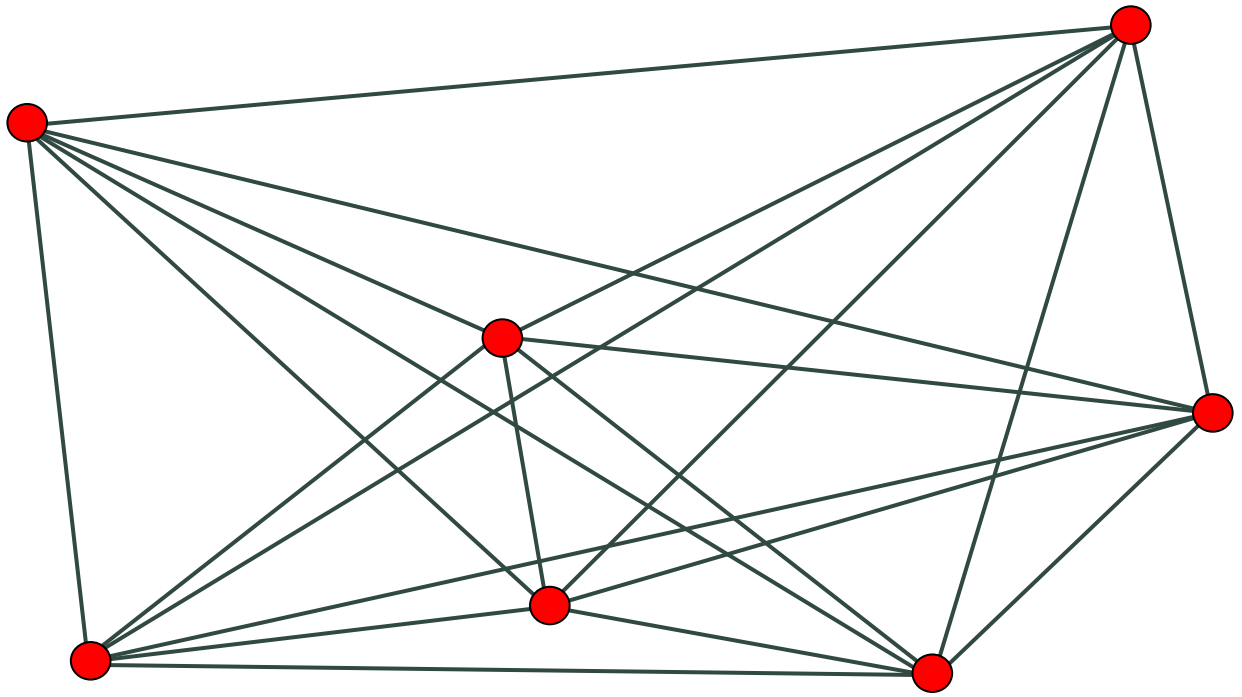
Max από τα n^2 ζεύγη σημείων

Πολυπλοκότητα Αλγόριθμου $O(n^2)$



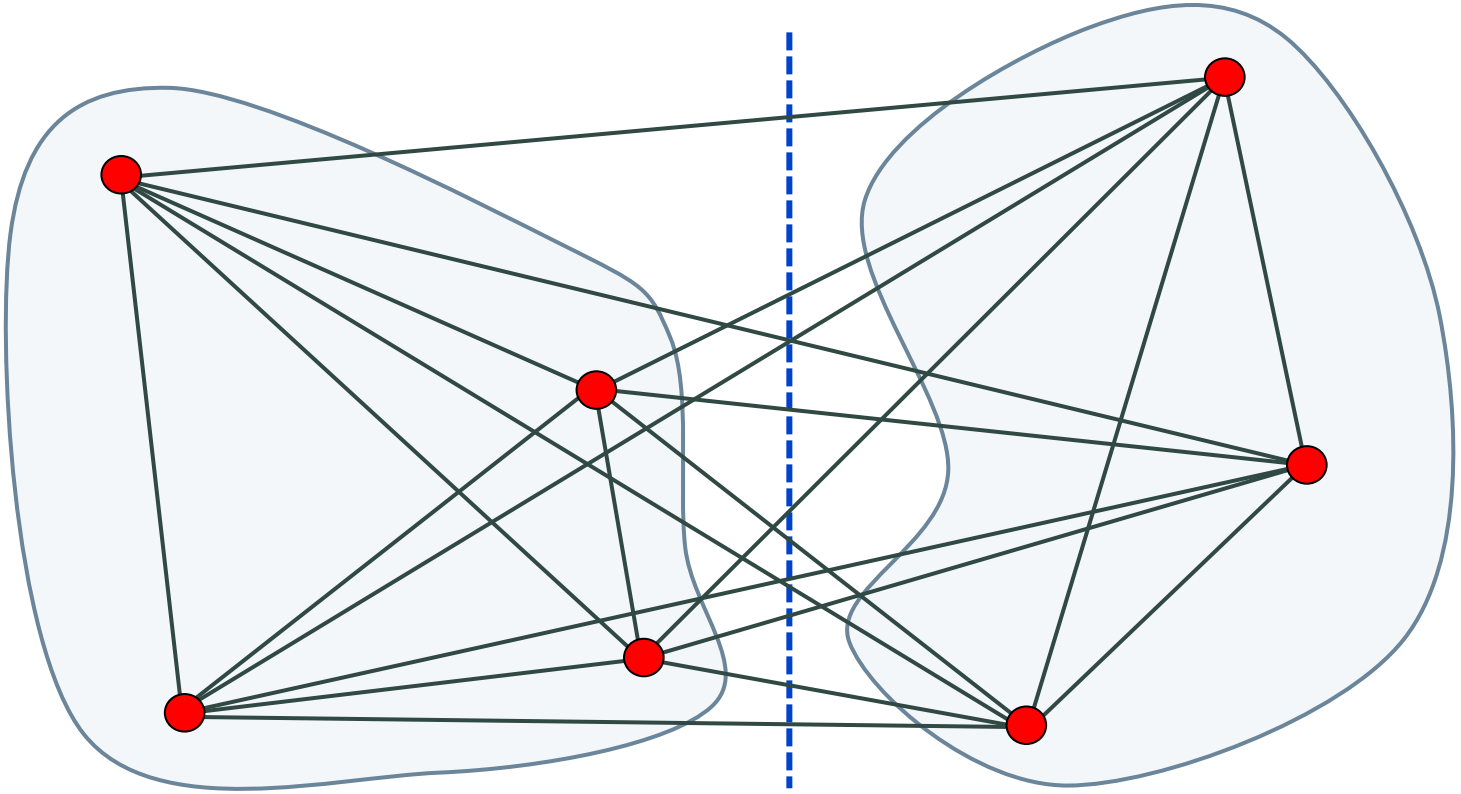
Αλγόριθμος O(n logn)

Επινόηση Καλής Τεχνικής και Ιδέας !!!



Αλγόριθμος O(n logn)

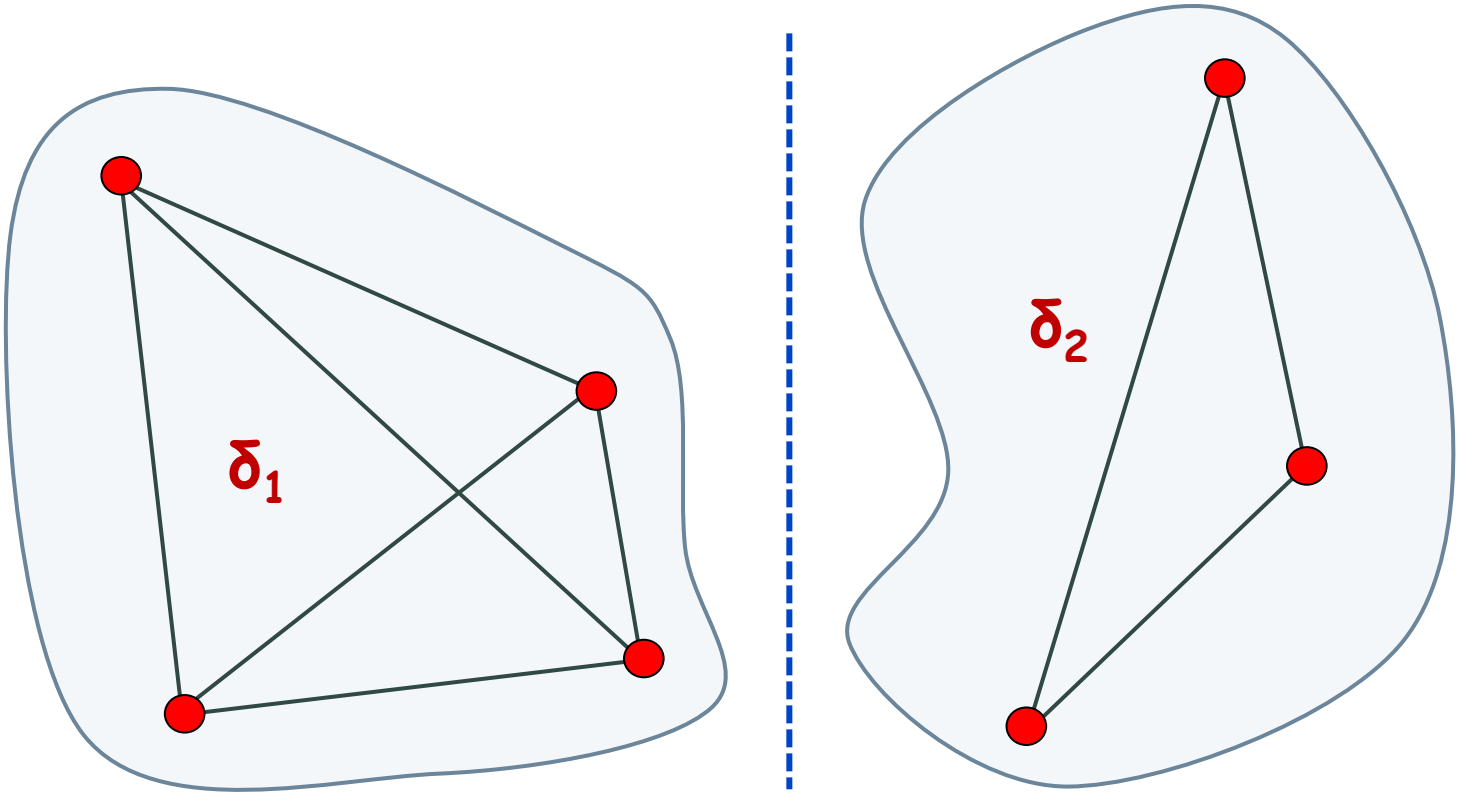
Τεχνική Διαίρει-και-Βασίλευε !!!



Αλγόριθμος O(n logn)

Τεχνική Διαίρει-και-Βασίλευε !!!

$D = \min\{\delta_1, \delta_2\}$

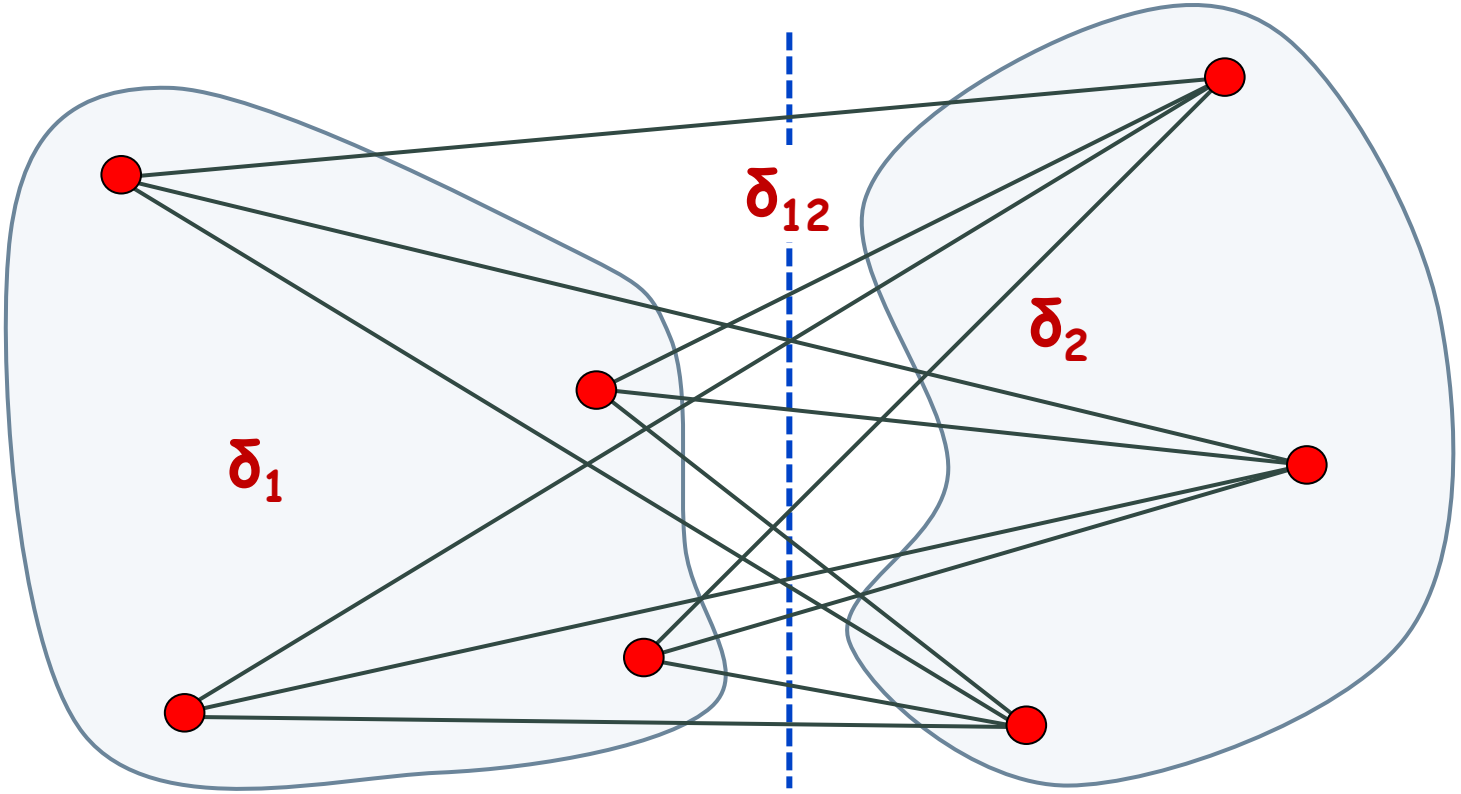


Αλγόριθμος O(n logn)

Τεχνική Διαίρει-και-Βασίλευε !!!

$D = \min\{\delta_1, \delta_2, \delta_{12}\}$

?



Εύρεση Πλησιέστερου Ζεύγους Σημείων

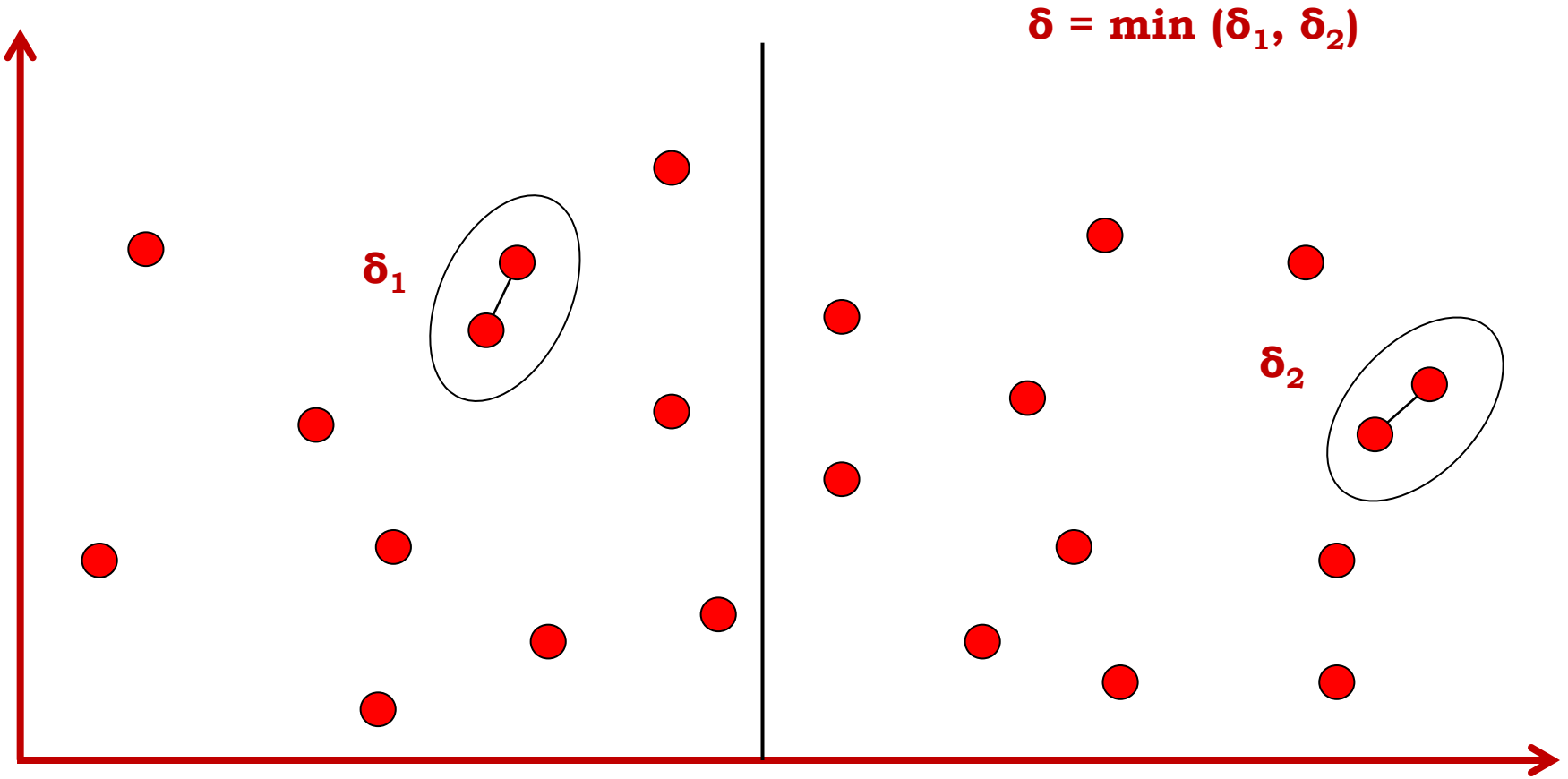
Τεχνική Διαίρει-και-Βασίλευε !

- ✓ Αλγόριθμος **Closest-Pair**

1. Ταξινόμησε τα σημεία σύμφωνα με τις x-συντεταγμένες τους.
2. Χωρίστε το σύνολο των σημείων σε δύο ίσου μεγέθους υποσύνολα με μια κάθετη γραμμή $x = x_{mid}$.
3. Λύσε το πρόβλημα αναδρομικά στα αριστερά και δεξιά υποσύνολα και υπολόγισε τις ελάχιστες αποστάσεις δ_1 και δ_2 στο αριστερό και δεξιό υποσύνολο, αντίστοιχα.
4. Υπολόγισε την ελάχιστη απόσταση δ_{12} μεταξύ των σημείων εκατέρωθεν της καθέτου $x = x_{mid}$ και τα οποία βρίσκονται σε απόσταση $\delta = \min\{\delta_1, \delta_2\}$ από αυτήν.
5. Επίστρεψε τη λύση D ως το ελάχιστο μεταξύ των δ_1, δ_2 και δ_{12}.

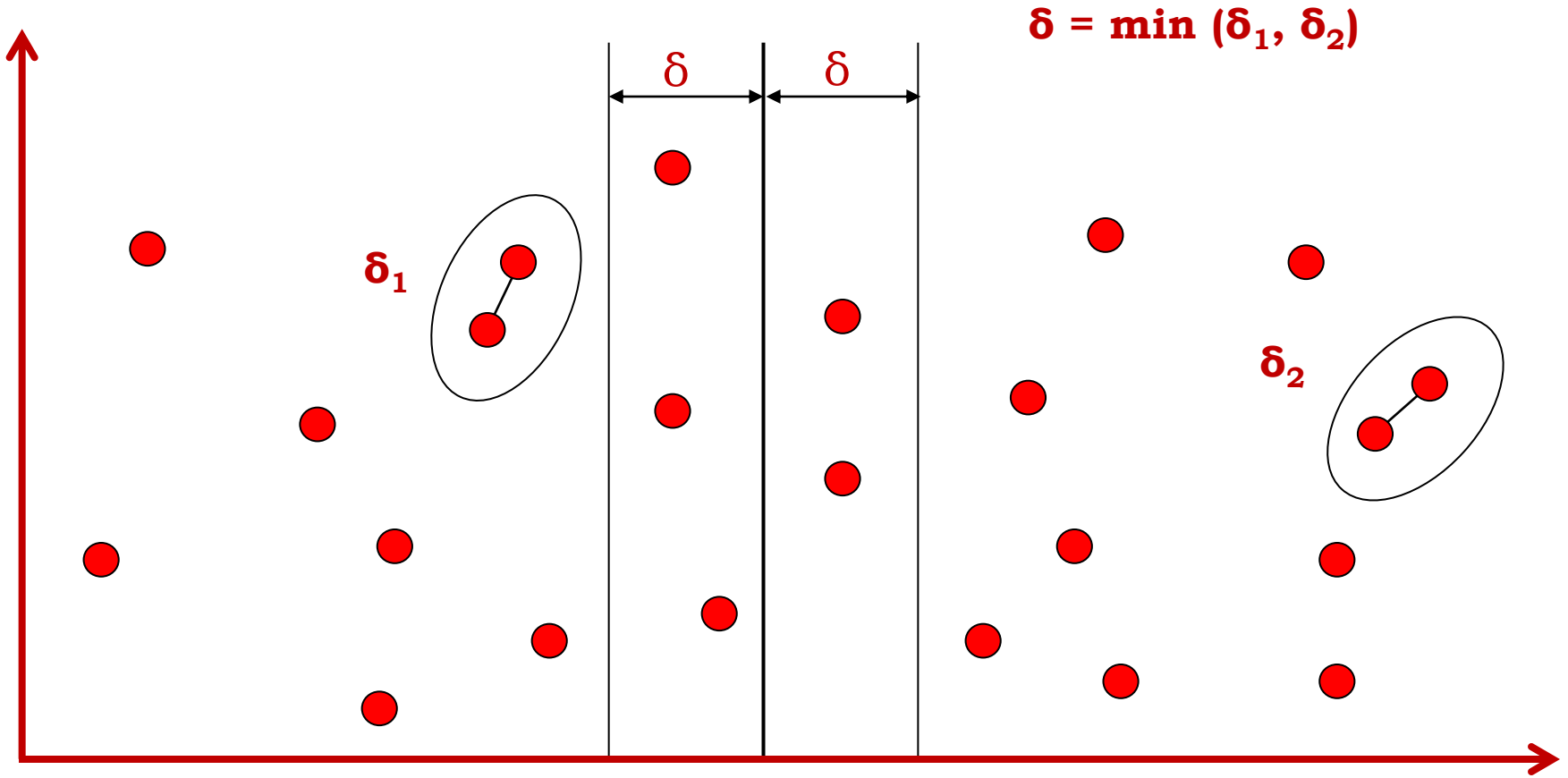
Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



Εύρεση Πλησιέστερου Ζεύγους Σημείων

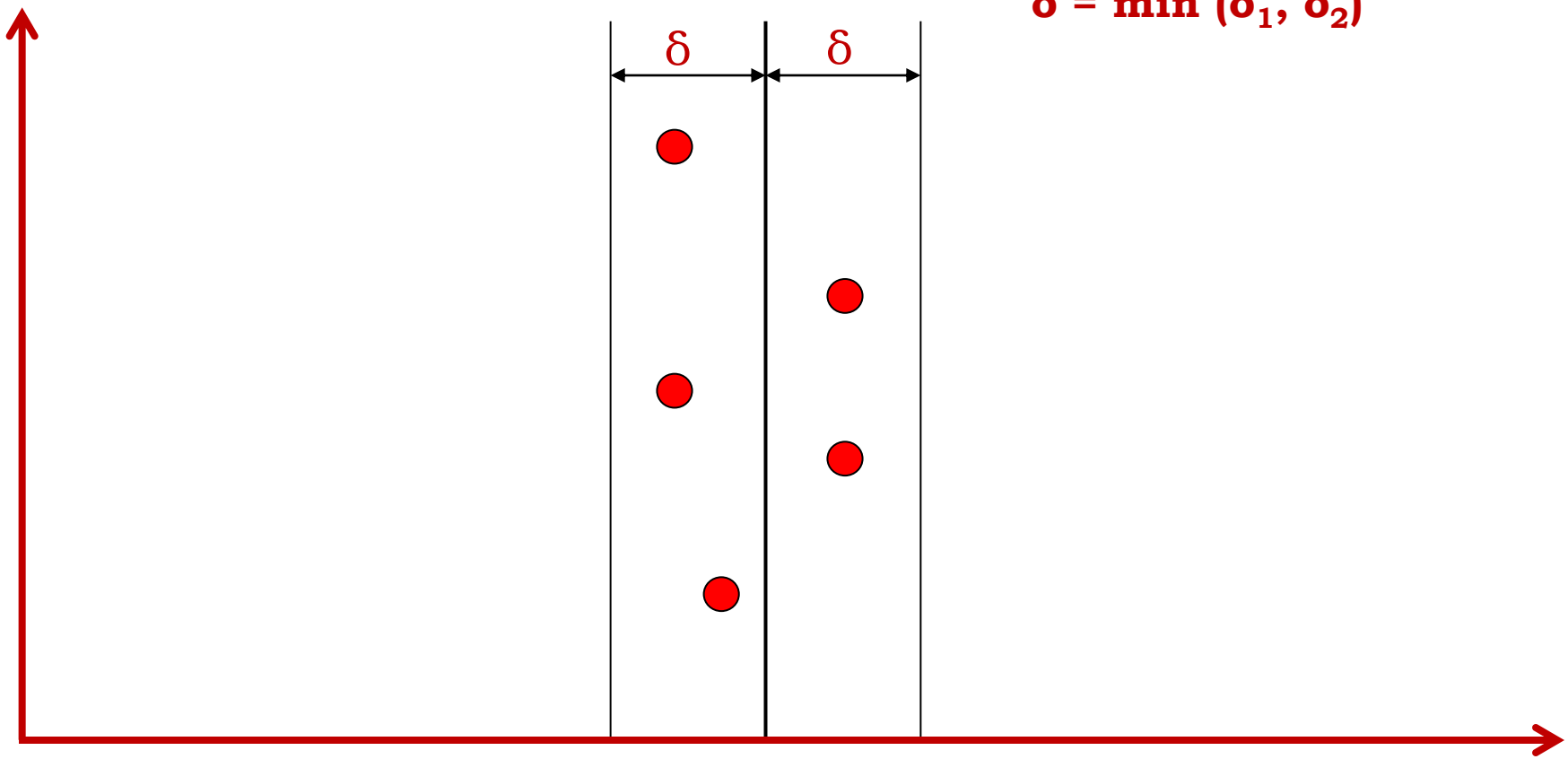
- ✓ Αλγόριθμος **Closest-Pair**



Εύρεση Πλησιέστερου Ζεύγους Σημείων

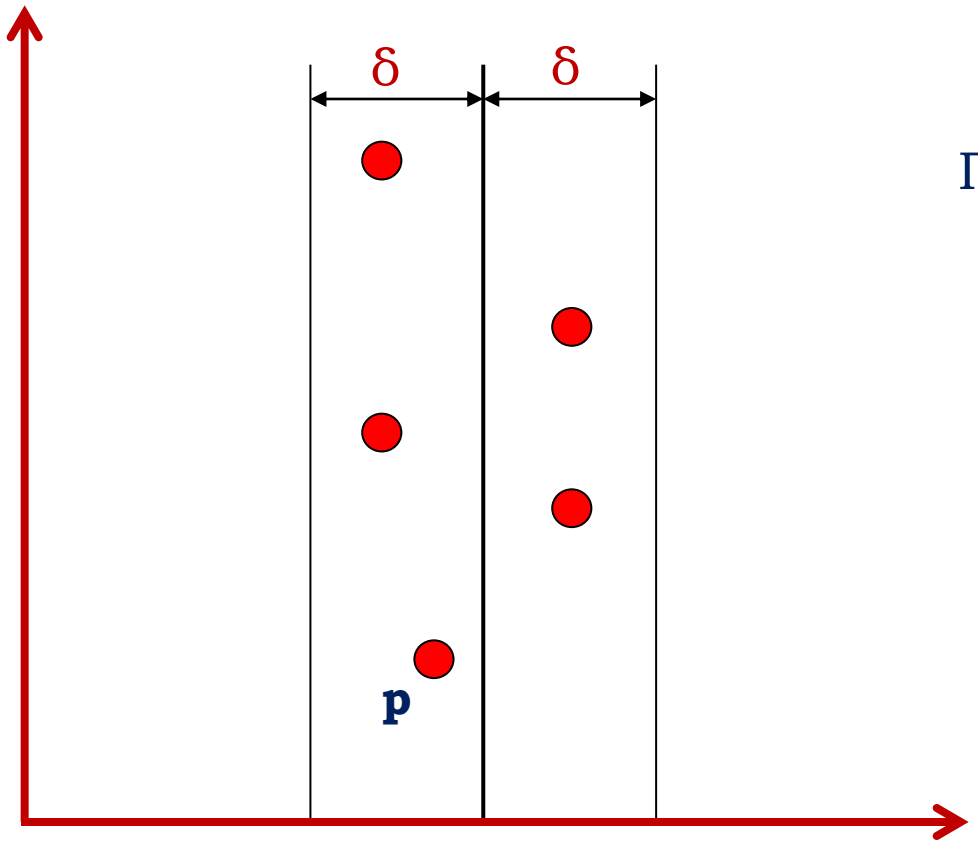
- ✓ Αλγόριθμος **Closest-Pair**

$\delta = \min(\delta_1, \delta_2)$



Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**

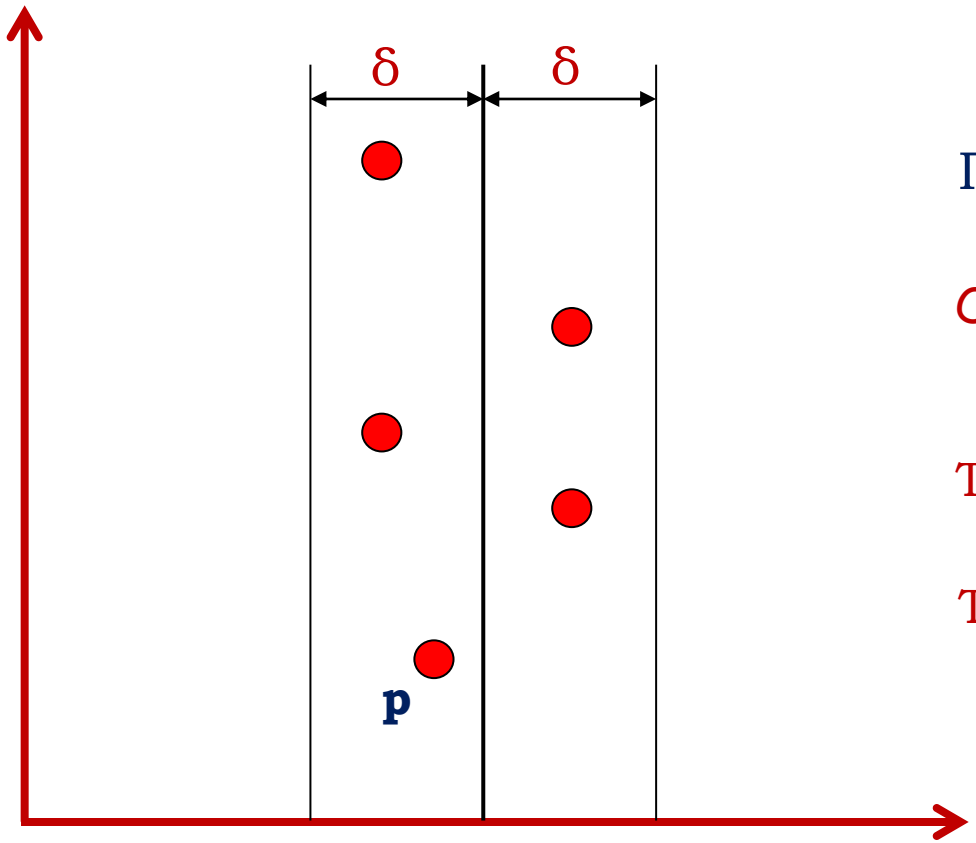


$\delta = \min(\delta_1, \delta_2)$

Πόσα σημεία **p** υπάρχουν ?

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$\delta = \min(\delta_1, \delta_2)$

Πόσα σημεία **p** υπάρχουν ?

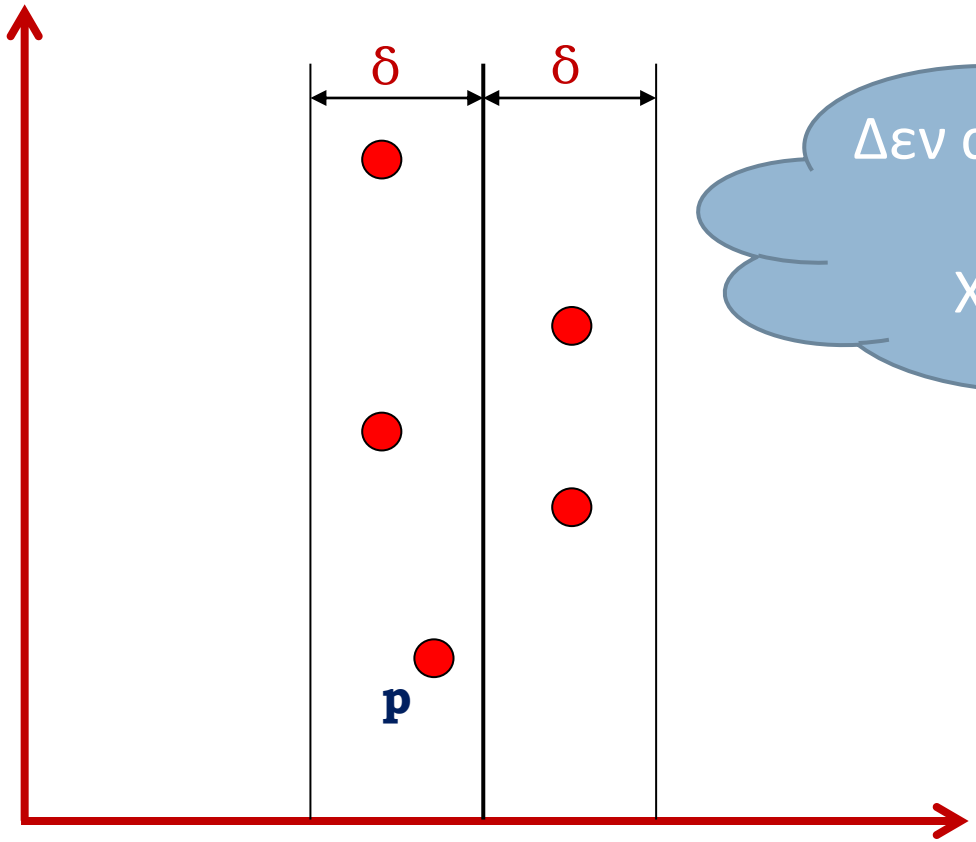
$O(n) \Rightarrow O(n^2)$ συγκρίσεις !

$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n^2)$

$T(n) = O(n^2)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

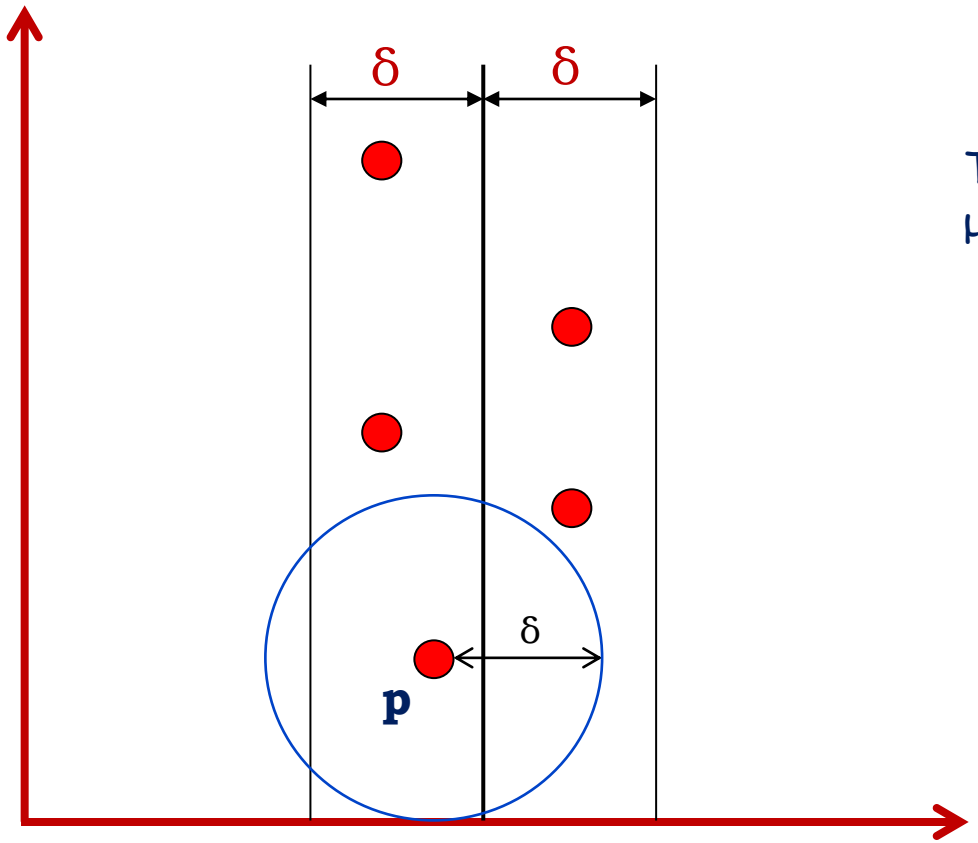
- ✓ Αλγόριθμος **Closest-Pair**



Δεν αρκεί μόνο η καλή Αλγοριθμική Τεχνική !!!
Χρειάζεται και Καλή Ιδέα !!!

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**

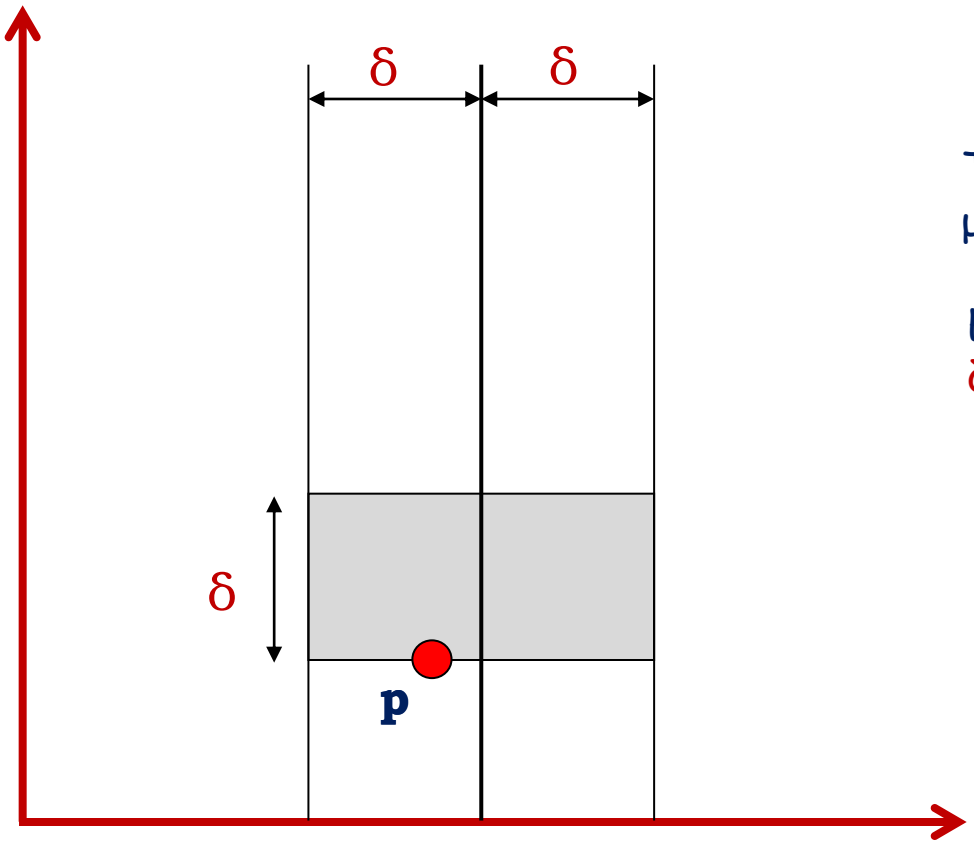


$$\delta = \min(\delta_1, \delta_2)$$

Το σημείο **p** με πόσα το λιγότερο σημεία μπορεί να συγκριθεί ?

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



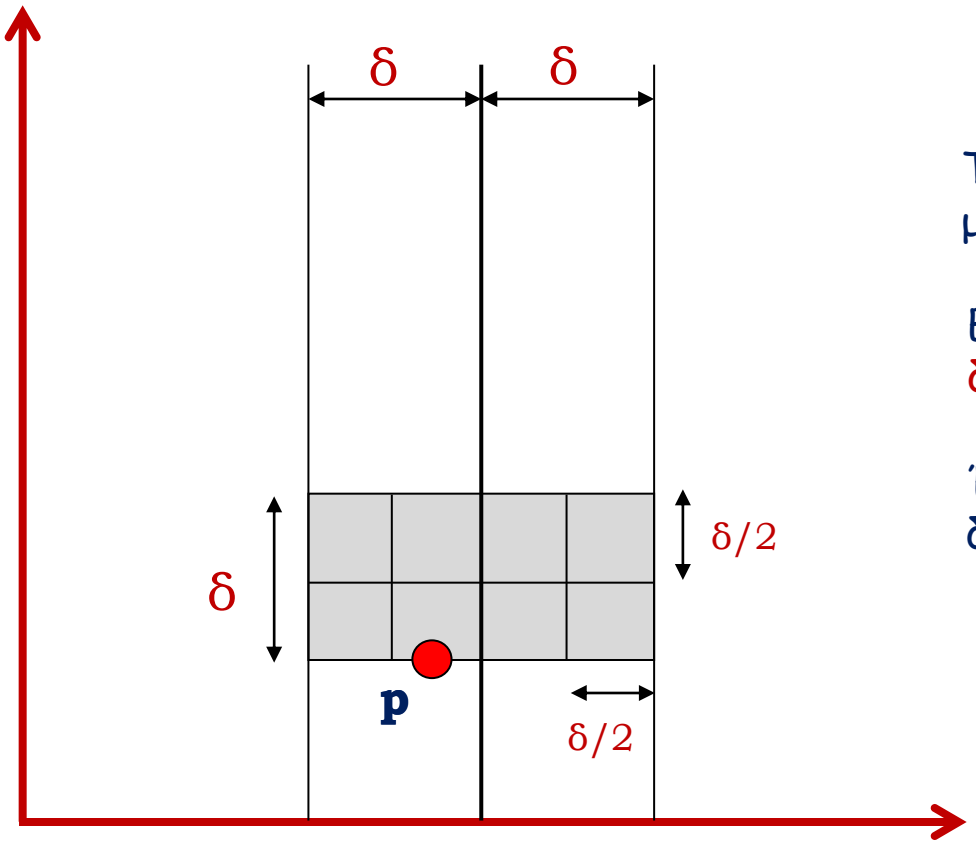
$$\delta = \min(\delta_1, \delta_2)$$

Το σημείο **p** με πόσα το λιγότερο σημεία μπορεί να συγκριθεί ?

Εντός του παραλληλογράμμου διαστάσεων δ x 2δ υπάρχουν το πολύ **8** σημεία

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

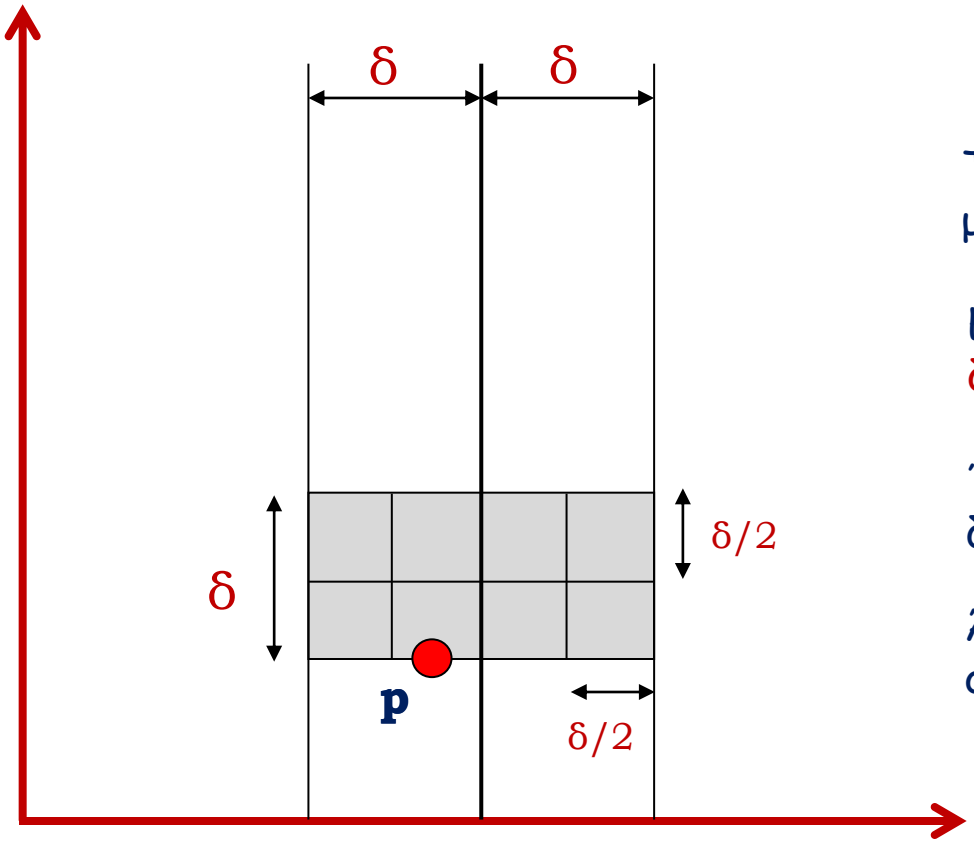
Το σημείο **p** με πόσα το λιγότερο σημεία μπορεί να συγκριθεί ?

Εντός του παραλληλογράμμου διαστάσεων δ x 2δ υπάρχουν το πολύ **8** σημεία

Ένα σημείο σε κάθε παραλληλόγραμμο διαστάσεων δ/2 x δ/2

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

Το σημείο **p** με πόσα το λιγότερο σημεία μπορεί να συγκριθεί ?

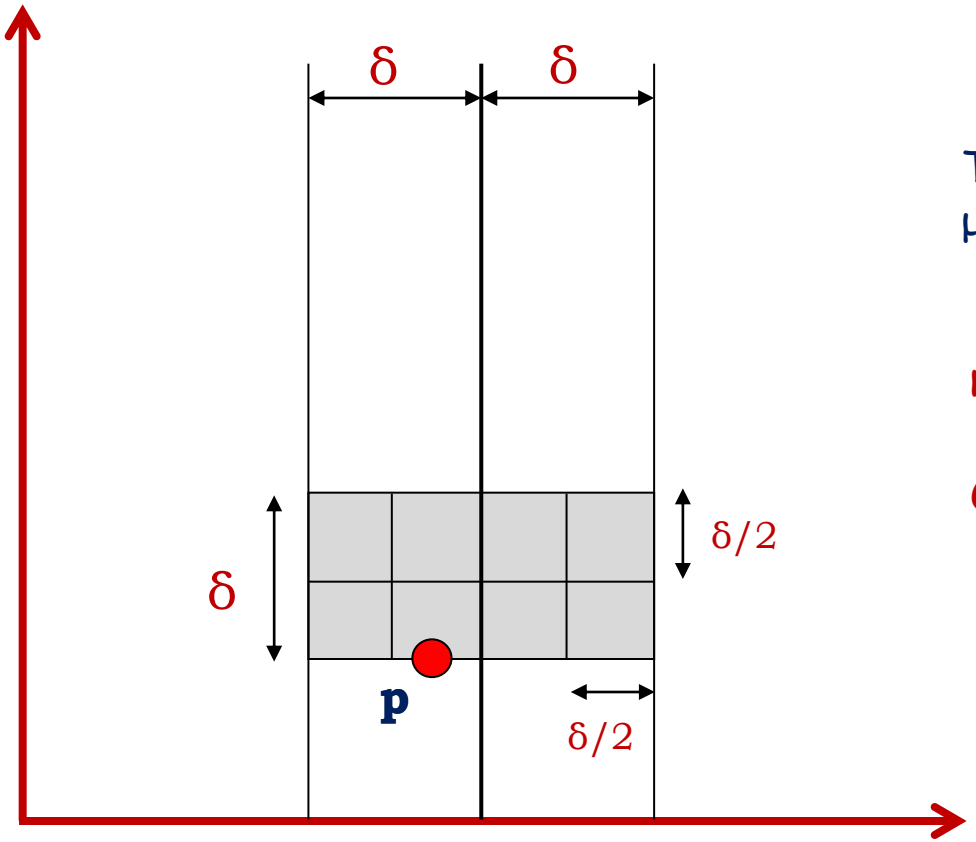
Εντός του παραλληλογράμμου διαστάσεων δ x 2δ υπάρχουν το πολύ **8** σημεία

Ένα σημείο σε κάθε παραλληλόγραμμο διαστάσεων δ/2 x δ/2

Άρα το σημείο **p** θα συγκριθεί με **7** το πολύ σημεία !!!

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

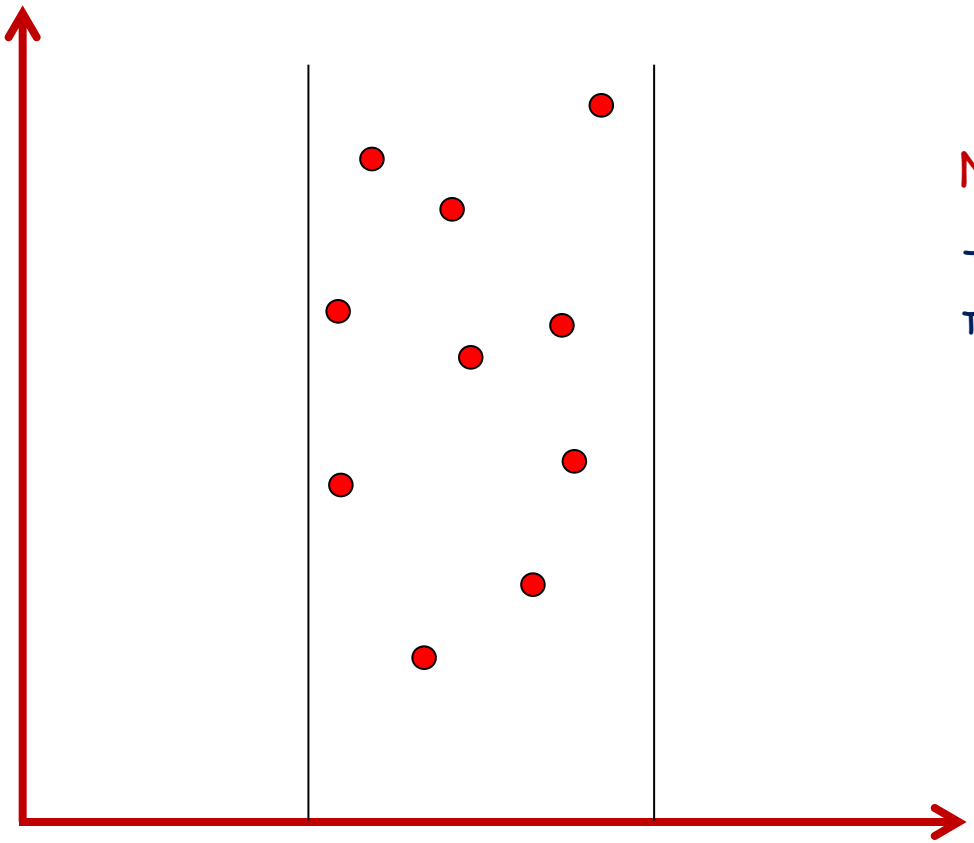
Το σημείο **p** με πόσα το λιγότερο σημεία μπορεί να συγκριθεί ?

m σημεία $\Rightarrow$ 7 m συγκρίσεις !

$O(n)$ σημεία $\Rightarrow$ $O(n)$ συγκρίσεις !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

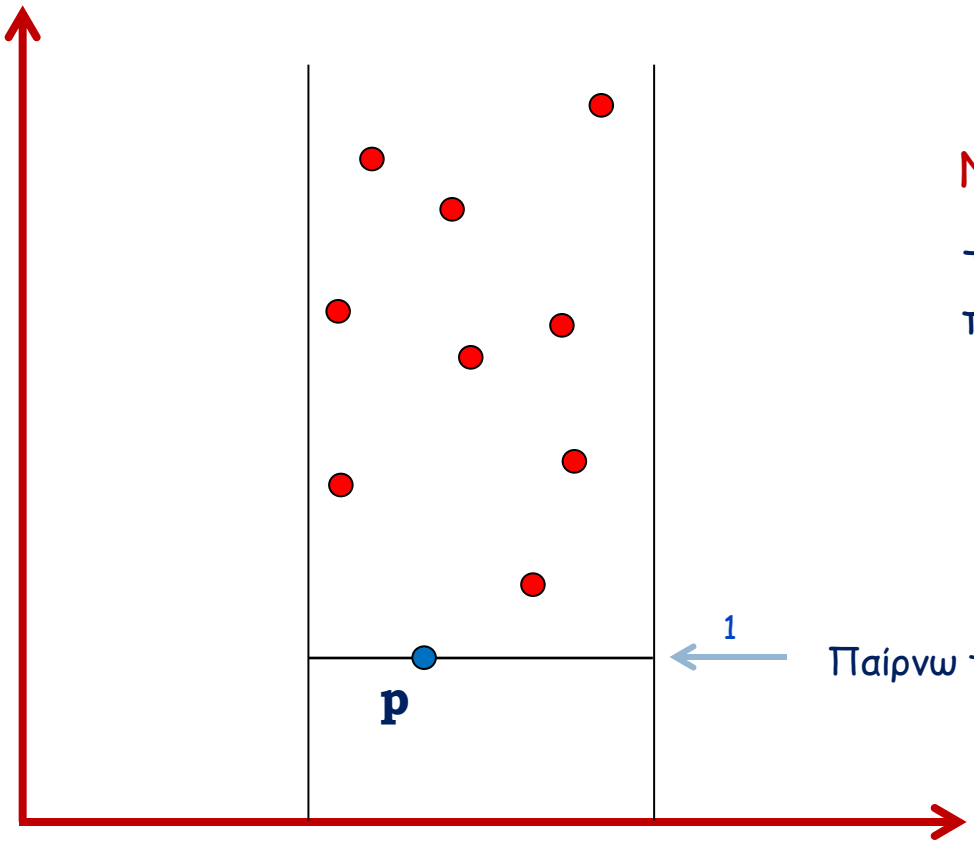
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**

$$\delta = \min(\delta_1, \delta_2)$$



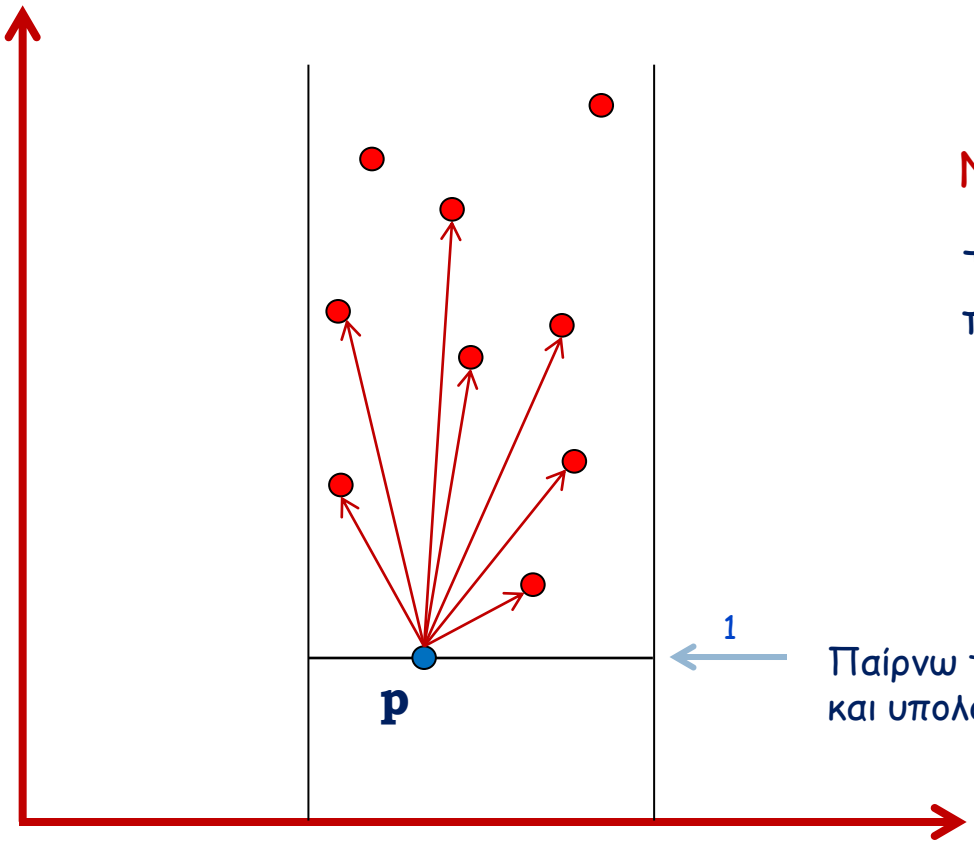
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Παίρνω το **p** στη «Λωρίδας» με την min y-συντεταγμένη

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$\delta = \min(\delta_1, \delta_2)$

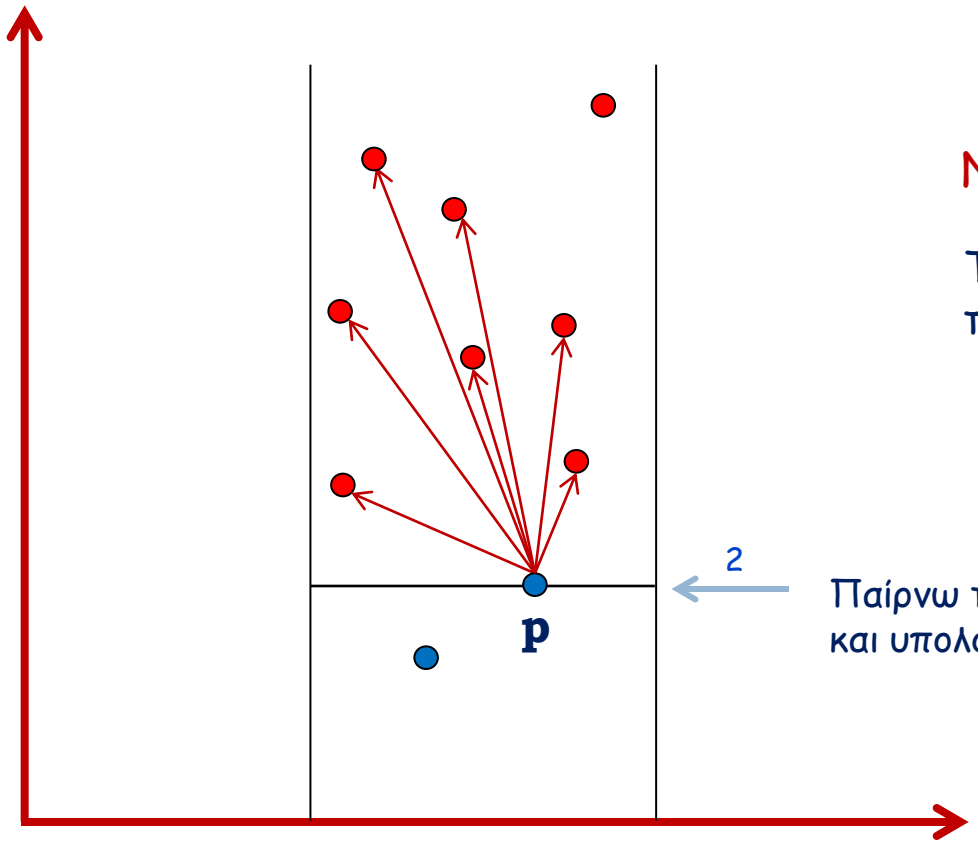
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Παίρνω το **p** στη «Λωρίδας» με την min y-συντεταγμένη και υπολογίζω την απόσταση του από τα 7 πλησιέστερα

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$\delta = \min(\delta_1, \delta_2)$

Να πως θα δουλέψει ο αλγόριθμος !

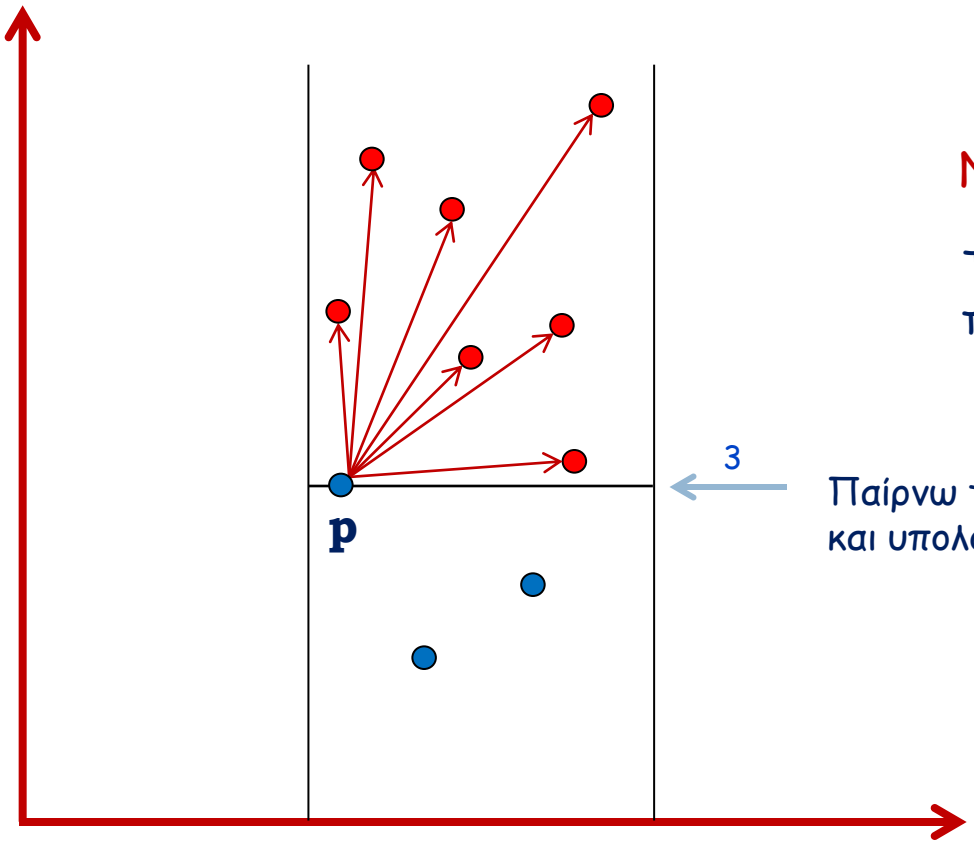
Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Παίρνω το **p** με την αμέσως min y-συντεταγμένη και υπολογίζω την απόσταση του από τα 7 πλησιέστερα

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**

$$\delta = \min(\delta_1, \delta_2)$$



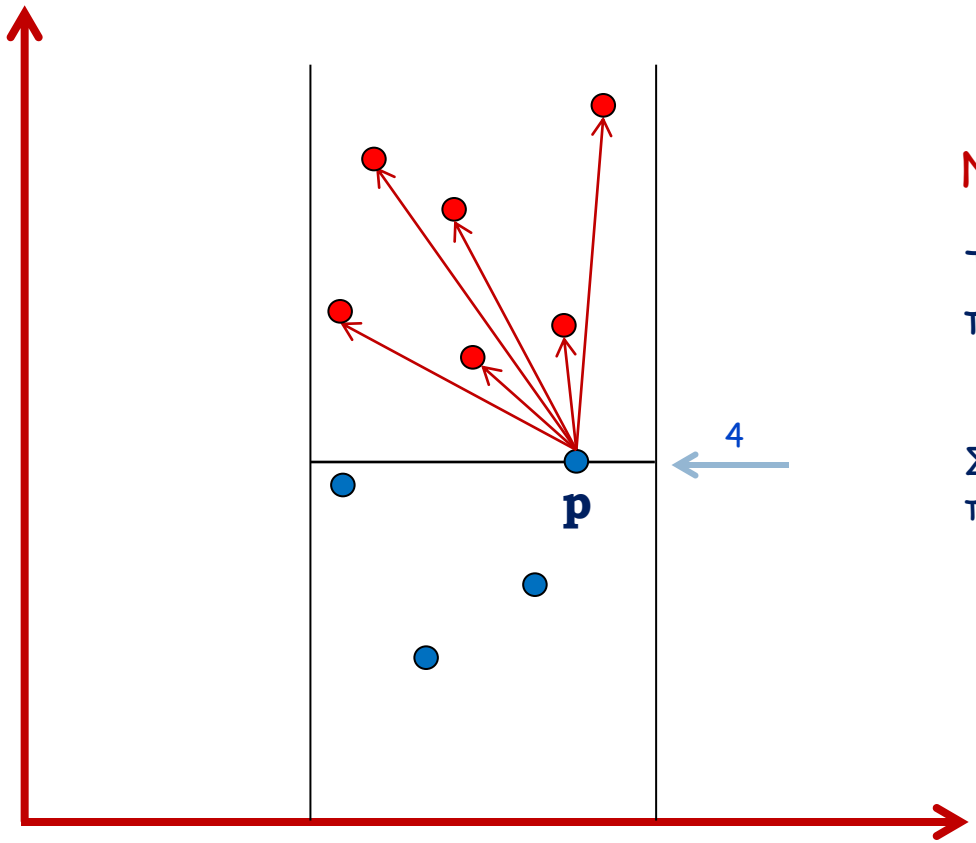
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Παίρνω το αμέσως επόμενο με min y-συντεταγμένη και υπολογίζω την απόσταση του από τα 7 πλησιέστερα

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

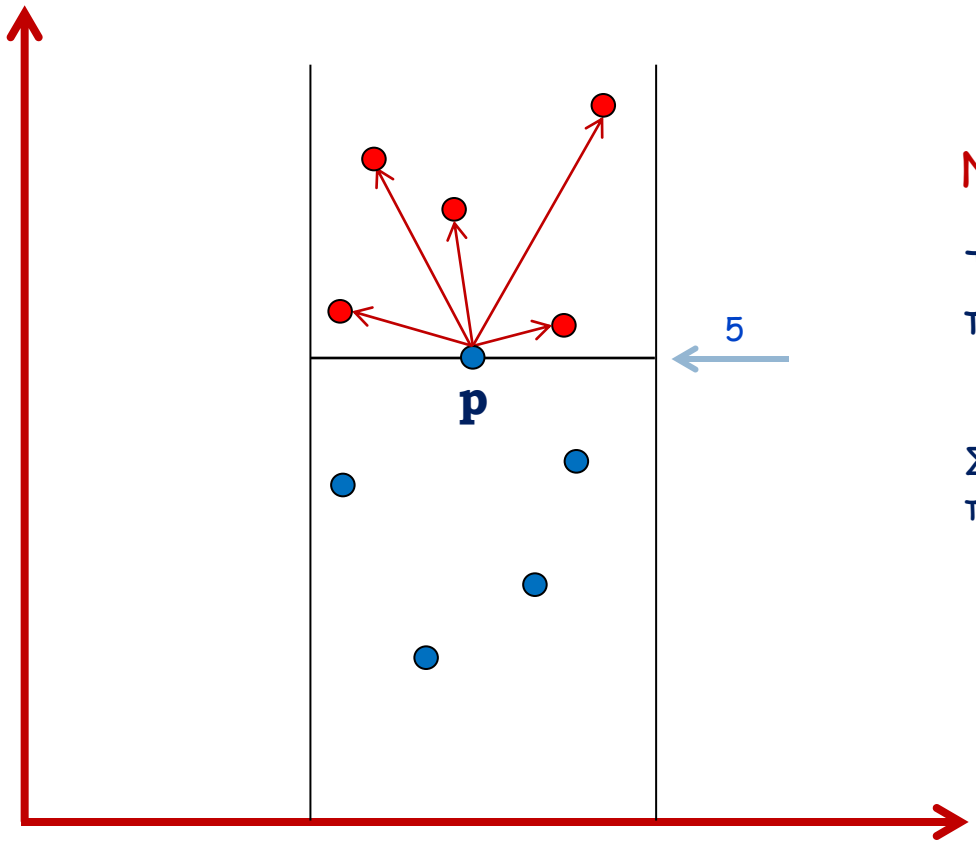
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

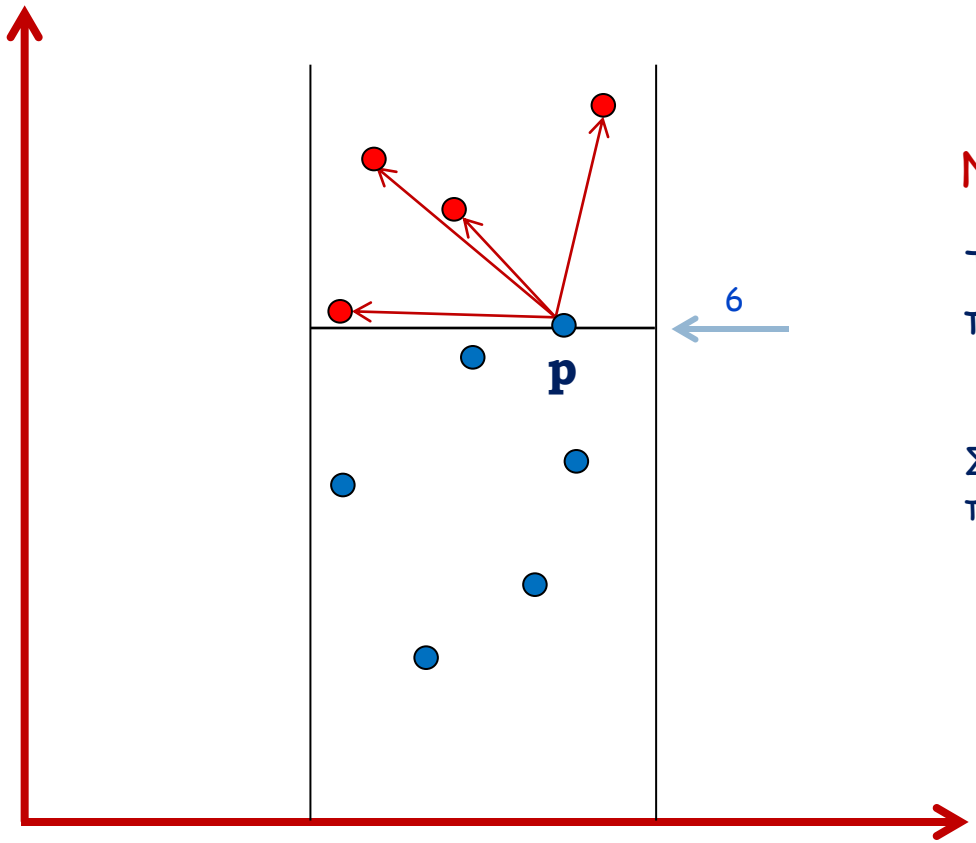
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

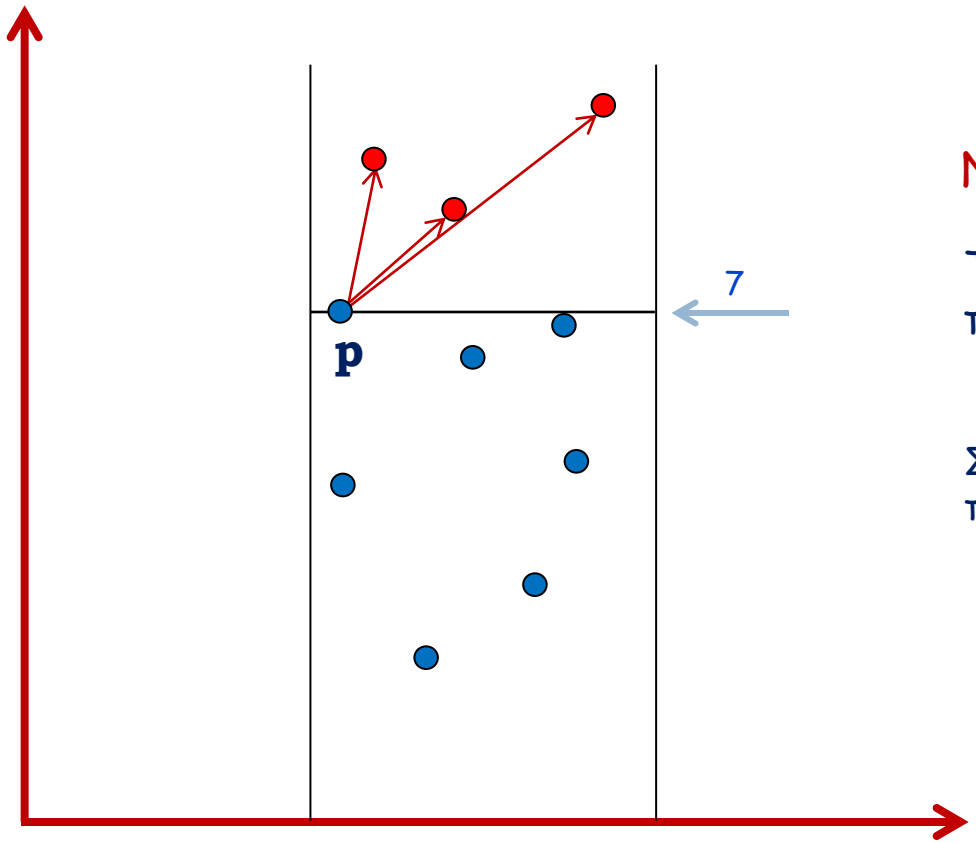
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

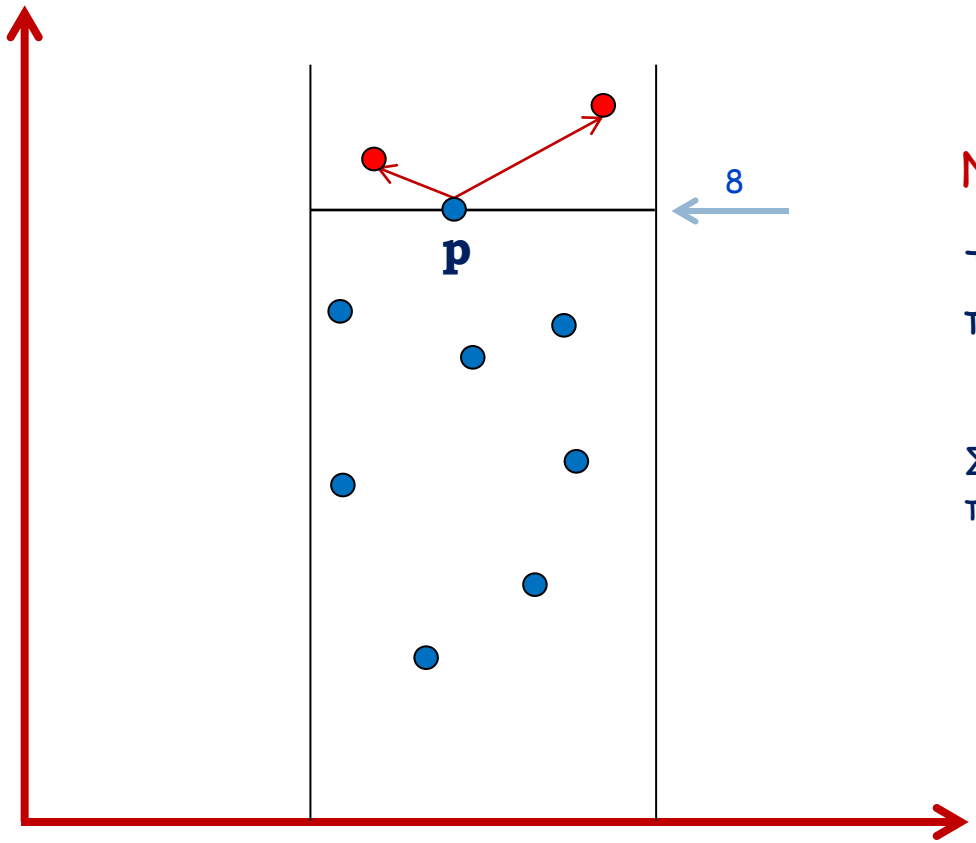
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

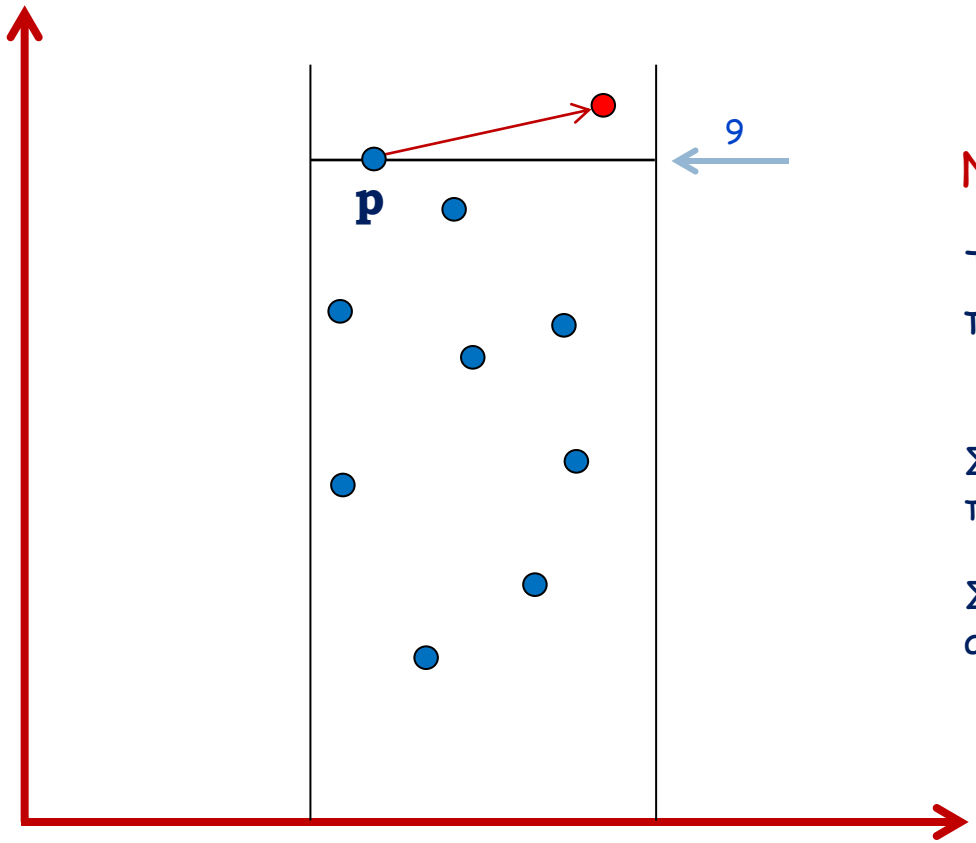
Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

Να πως θα δουλέψει ο αλγόριθμος !

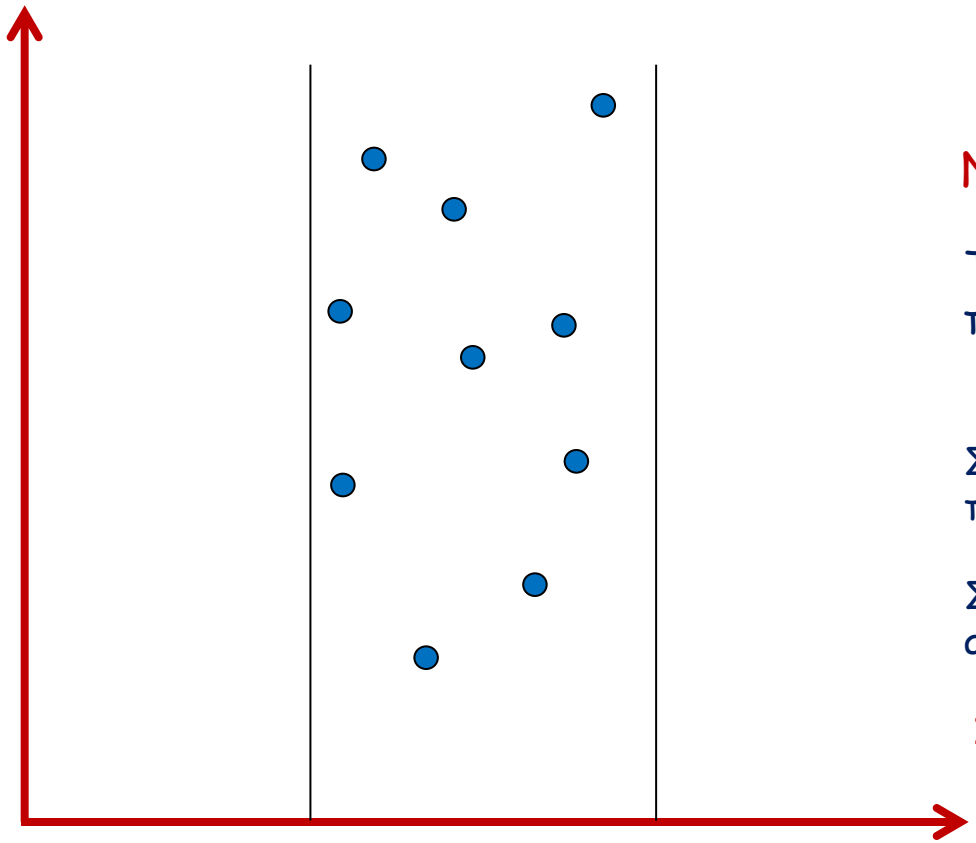
Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Στο τέλος υπολογίζω το δ_{12} να είναι το min όλων αυτών των αποστάσεων !

Εύρεση Πλησιέστερου Ζεύγους Σημείων

- ✓ Αλγόριθμος **Closest-Pair**



$$\delta = \min(\delta_1, \delta_2)$$

Να πως θα δουλέψει ο αλγόριθμος !

Ταξινομώ τα στοιχεία της «λωρίδας» ως προς την y-συντεταγμένη

Συνεχίζω έως ότου εξαντληθούν όλα τα στοιχεία της «Λωρίδας» !

Στο τέλος υπολογίζω το δ_{12} να είναι το min όλων αυτών των αποστάσεων !

$$D = \min(\delta_1, \delta_2, \delta_{12})$$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ορθότητα αλγόριθμου

- Η απόδειξη της ορθότητας του αλγόριθμου **Closest-Pair** είναι σχεδόν προφανής.

✓ Χρειάζεται λίγη μόνο προσοχή στο να δειχθεί ότι για τον υπολογισμό της απόστασης δ_{12} είναι αρκετές οι 7 συγκρίσεις για κάθε σημείο στη «λωρίδα».

Ορθότητα αλγόριθμου: Σωστό αποτέλεσμα για κάθε είσοδο !!!

Divide & Conquer

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Πολυπλοκότητα αλγόριθμου

- Η αναδρομική σχέση που εκφράζει το χρόνο εκτέλεσης του αλγόριθμου:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn$$ ⟵ **c** σταθερά (με τιμή **c** = 7)

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$$

- Από Κύριο Θεώρημα:

$$T(n) = O(n \log n)$$

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

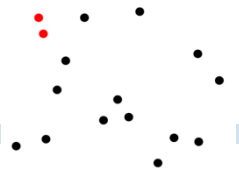
Divide & Conquer

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Αλγόριθμος $O(n^2)$ σε σχέση με έναν $O(n \log n)$

n	logn	n^2	n logn
16	4	256	64
1024	10	1.048.576	10.240
2048	11	4.194.304	22.528
4096	12	16.777.216	49.152
1048576	20	1.099.511.627.776	20.971.520
33554432	25	1.125.899.906.842.624	838.860.800

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$



Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

Θέλω… «γρηγορότερο» αλγόριθμο !!!

Εντάξει… θα προσπαθήσω !!!

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

Προσπαθώ !!!

αλλά δεν μπορώ να σχεδιάσω κάποιον «γρηγορότερο» από $O(n \log n)$

Χμμ…

δεν θα αποφύγω την απόλυση !!!

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

Το πρόβλημα της διακριτότητας στοιχείων (**element distinctness problem**) ή της μοναδικότητας στοιχείων (**element uniqueness problem**) είναι το πρόβλημα του καθορισμού αν όλα τα στοιχεία μιας ακολουθίας είναι διαφορετικά.

Για πρόσεχε λίγο αυτό το πρόβλημα !!!

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

Διακριτότητα Στοιχείων

Απέδειξα ότι το πρόβλημα αυτό έχει *κάτω φράγμα* επίλυσης $\Omega(n \log n)$

Και λοιπόν… εγώ ασχολούμαι με το πρόβλημα

Πλησιέστερου Ζεύγους Σημείων

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

Αναγωγές !!!

Το πρόβλημα **Διακριτότητα Στοιχείων**

Ανάγεται

στο πρόβλημα **Πλησιέστερου Ζεύγους Σημείων**

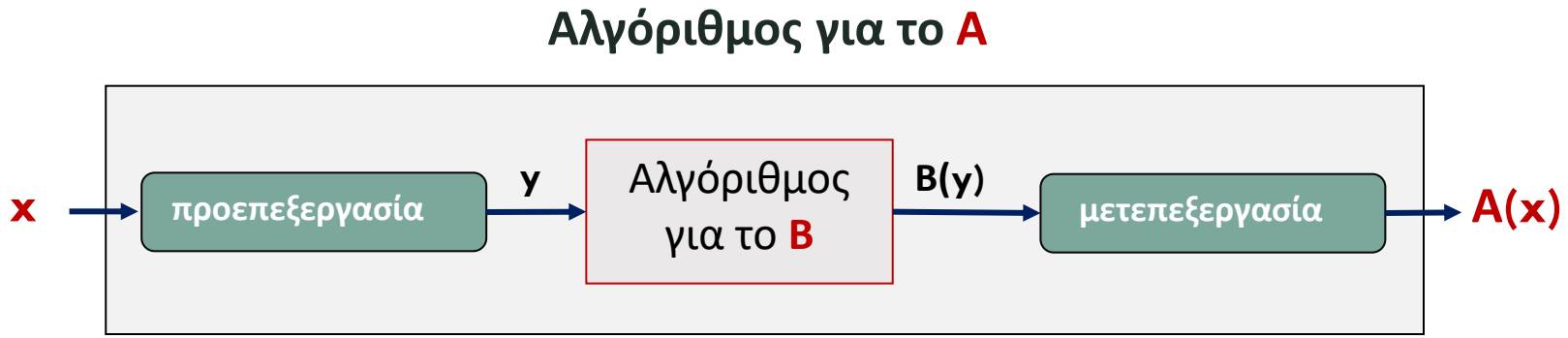
Αν αποδείξεις αυτό,
έχεις αποφύγει την απόλυση !

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Αναγωγές !!!

- Λέμε ότι το πρόβλημα **A ανάγεται** στο πρόβλημα **B** και το συμβολίζουμε **A ≤ B,** εάν μπορούμε να σχεδιάσουμε έναν αλγόριθμο πολυωνυμικού χρόνου για την επίλυση του **A** *χρησιμοποιώντας* έναν αλγόριθμο για την επίλυση του **B** !!!



Εύρεση Πλησιέστερου Ζεύγους Σημείων

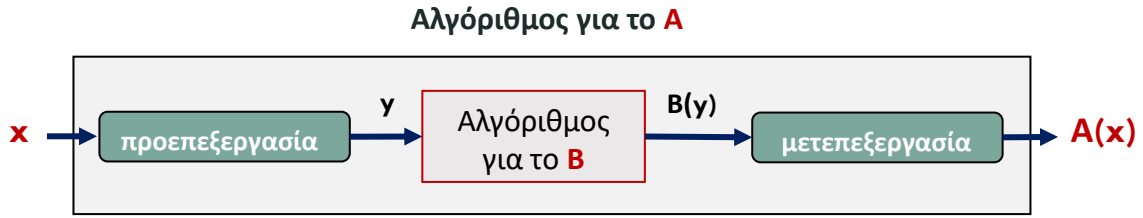
Γιατί Αναγωγές ?

- **Αυξάνουν την ισχύ των αλγορίθμων μας**:

  Εάν **A** ≤ **B** και έχουμε έναν αλγόριθμο για το **B**, τότε μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο **B** για να λύσουμε το πρόβλημα **A** !!!

- **Επιτρέπουν συγκρίσεις και κατηγοριοποιήσεις**:

  Μια αναγωγή **A** ≤ **B** μας επιτρέπει να *συγκρίνουμε* τις πολυπλοκότητες των προβλημάτων **A** και **B** και να *κατηγοριοποιήσουμε αυτά* ανάλογα με τη δυσκολία τους !!!



Εύρεση Πλησιέστερου Ζεύγους Σημείων

Αναγωγή:

Διακριτότητα Στοιχείων ≤ Πλησιέστερο Ζεύγος Σημείων

- **Τι σημαίνει αυτό !!!**

  Μπορώ να λύσω το πρόβλημα

  Διακριτότητα Στοιχείων

  χρησιμοποιώντας έναν αλγόριθμο για το πρόβλημα

  Πλησιέστερο Ζεύγος Σημείων

  Για παράδειγμα, τον **Closest-Pair**

Να πως !!!

```
Algorithm Element-Distinctness(A)
begin
    Algorithm Closest-Pair(A)
    Return D

    if (D > 0) then return Yes
               else return No
end
```

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Αναγωγή:

Διακριτότητα Στοιχείων $\leq$ Πλησιέστερο Ζεύγος Σημείων

- **Τότε, έχουμε αυτό που θέλουμε !!!**

  Εάν μπορούσα να σχεδιάσω ένα αλγόριθμο για το «**Πλησιέστερο Ζεύγος Σημείων**» πολυπλοκότητα $o(n\log n)$, δηλ. «γρηγορότερο» από $O(n\log n)$, τότε

  θα μπορούσα να λύσω το πρόβλημα «**Διακριτότητα Στοιχείων**» σε χρόνο $o(n\log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Αναγωγή:

Διακριτότητα Στοιχείων $\leq$ Πλησιέστερο Ζεύγος Σημείων

- **Τότε, έχουμε αυτό που θέλουμε !!!**

  Εάν μπορούσα να σχεδιάσω ένα αλγόριθμο για το «**Πλησιέστερο Ζεύγος Σημείων**» πολυπλοκότητα $o(n\log n)$, δηλ. «γρηγορότερο» από $O(n\log n)$, τότε

  θα μπορούσα να λύσω το πρόβλημα «**Διακριτότητα Στοιχείων**» σε χρόνο $o(n\log n)$

  Αυτό είναι αδύνατον να συμβεί, διότι $O(n\log n)$ είναι το *κάτω φράγμα* επίλυσης του προβλήματος «**Διακριτότητα Στοιχείων**» !!!

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

Τώρα γνωρίζω…

Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Εύρεση Πλησιέστερου Ζεύγους Σημείων

Ένα πολύ ενδιαφέρον Σενάριο !!!

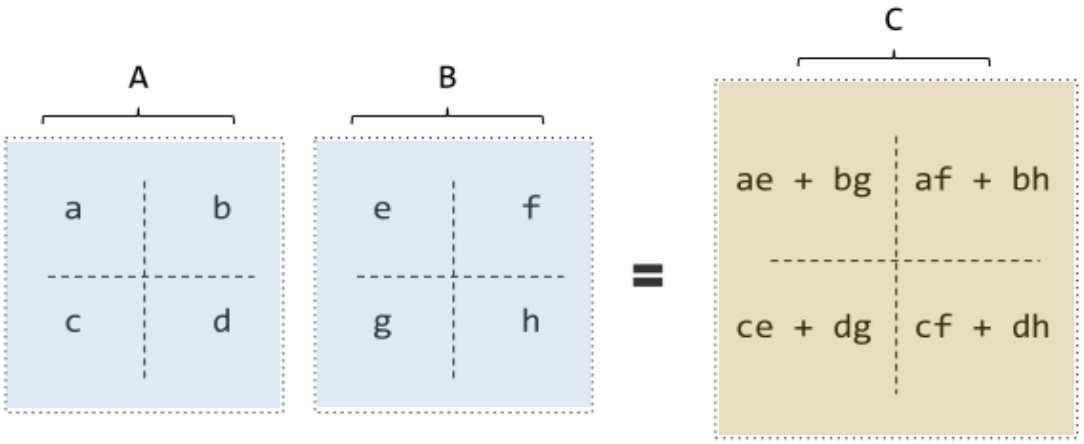
Τώρα γνωρίζω…

Κοίτα "boss" !!!... Κανένας δεν μπορεί να σχεδιάσει «γρηγορότερο» αλγόριθμο από εμένα !!!

Και μπορώ να στο αποδείξω !!!

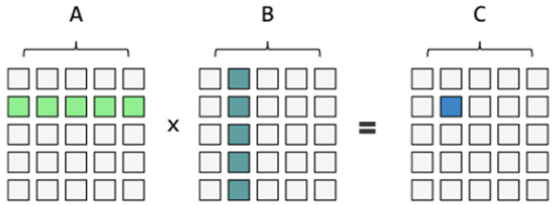
Πολυπλοκότητα αλγόριθμου: $W(n) \in \Theta(n \log n)$

Πολλαπλασιασμός Πινάκων



Strassen's Matrix Multiplication

Αλγόριθμος… Λυκείου !!!



Πολλαπλασιασμός Πινάκων

- Έστω $A = (a_{ij})$ και $B=(b_{ij})$ δύο τετραγωνικοί $n \times n$ πίνακες.
- Υπάρχει ο κλασσικός μας αλγόριθμος πολλαπλασιασμού των δύο πινάκων $C = A \times B$ που βασίζεται στον ορισμό:

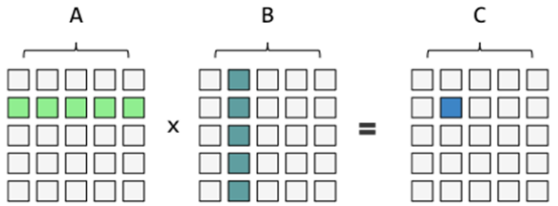
$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

όπου c_{ij} το στοιχείο του πίνακα C στην ij θέση, $i, j = 1, 2, \ldots, n$.

$$C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$\begin{aligned} c_{11} &= a_{11} \cdot b_{11} + a_{12} \cdot b_{21} \\ c_{12} &= a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ c_{21} &= a_{21} \cdot b_{11} + a_{22} \cdot b_{21} \\ c_{22} &= a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{aligned}$$

Αλγόριθμος… Λυκείου !!!



Πολλαπλασιασμός Πινάκων

- Ο κλασσικός μας αλγόριθμος πολλαπλασιασμού :

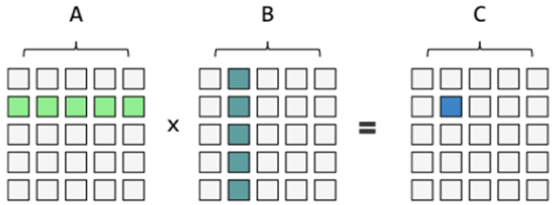
$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

όπου c_{ij} είναι το στοιχείο του πίνακα C στην ij θέση, i, j = 1, 2, …, n.

- Το στοιχείο c_{ij} του πίνακα C υπολογίζεται σε χρόνο $O(n)$, θεωρώντας βασικές πράξεις την πρόσθεση και τον πολλαπλασιασμό.

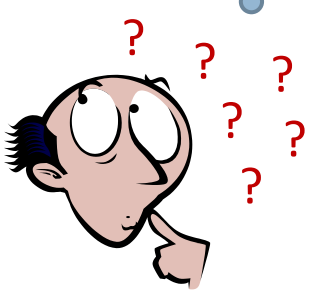
Επομένως, ο κλασσικός αλγόριθμος πολλαπλασιασμού δύο $n \times n$ πινάκων έχει πολυπλοκότητα $O(n^3)$.

Αλγόριθμος… Λυκείου !!!



Πολλαπλασιασμός Πινάκων

Μπορώ να χρησιμοποιήσω την τεχνική του Διαίρει-και-Βασίλευε ?



ΝΑΙ !!!

Δεν ξεχνώ !!!… Κάθε αλγόριθμος *Διαίρει-και-Βασίλευε* περιλαμβάνει τρείς φάσεις:

1. Διαίρεσε το πρόβλημα σε μικρότερα υποπροβλήματα.
2. Κατάκτησε τα υποπροβλήματα, επιλύοντάς τα αναδρομικά. Εάν τα υποπροβλήματα είναι μικρού μεγέθους, απλώς επίλυσέ τα άμεσα.
3. Συνδύασε τις λύσεις των υποπροβλημάτων για να βρεις τη λύση του αρχικού προβλήματος.

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

- Παρουσιάζουμε έναν απλό αναδρομικό αλγόριθμο για τον υπολογισμό του γινομένου $C = A \cdot B$ δύο τετραγωνικών $n \times n$ πινάκων.
- Υποθέτουμε, για απλότητα, ότι το n είναι δύναμη του $2 \Rightarrow n/2$ ακέραιος.
- Διαμερίζουμε καθένα από τους A, B, και C σε τέσσερις $n/2 \times n/2$ πίνακες:

$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \qquad B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) \qquad C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

- Οι πίνακες A_{ij}, B_{ij}, και C_{ij} είναι διαστάσεων $n/2 \times n/2$.

$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \qquad B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) \qquad C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

- Οπότε, η εξίσωση $C = A \cdot B$ μπορεί να γραφεί :

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

- Η εξίσωση :

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

αντιστοιχεί στις 4 εξισώσεις :

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

- Κάθε μία από τις 4 εξισώσεις περιγράφει 2 πολλαπλασιασμούς $n/2 \times n/2$ πινάκων και την πρόσθεση των $n/2 \times n/2$ γινομένων τους !!!

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

```
Algorithm RM(A,B)
begin
    n ← A.rows
    let C be a new n×n matrix
    if n == 1
        c11 = a11 + b11
    else
        divide A,B,C in n/2 x n/2 submx
        C11 = RM(A11,B11) + RM(A12,B21)
        C12 = RM(A11,B12) + RM(A12,B22)
        C21 = RM(A21,B11) + RM(A22,B21)
        C22 = RM(A21,B12) + RM(A22,B22)
    end
    return C
end
```

- Με βάση τις 4 αυτές εξισώσεις

$$
\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}
$$

σχεδιάζουμε έναν απλό αναδρομικό αλγόριθμο Διαίρει-και-Βασίλευε !!!

- 8 πολλαπλασιασμοί $n/2 \times n/2$ πινάκων
- 4 προσθέσεις των γινομένων τους

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

- ```
  Algorithm RM(A,B)
  begin
      n ← A.rows
      let C be a new n×n matrix
      if n == 1
          c11 = a11 + b11
      else
          divide A,B,C in n/2 x n/2 submx
          C11 = RM(A11,B11) + RM(A12,B21)
          C12 = RM(A11,B12) + RM(A12,B22)
          C21 = RM(A21,B11) + RM(A22,B21)
          C22 = RM(A21,B12) + RM(A22,B22)
      end
      return C
  end
  ```

- Πολυπλοκότητα αλγόριθμου RM

$$T(n) = \begin{cases} O(1) & n = 1 \\ 8\,T(n/2) + O(n^2) & n \geq 1 \end{cases}$$
  ```

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

- Θυμηθείτε !!!

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{εάν} \quad c < \log_b a \quad (1) \\ O(n^c \log n) & \text{εάν} \quad c = \log_b a \quad (2) \\ O(n^c) & \text{εάν} \quad c > \log_b a \quad (3) \end{cases}$$

$$T(n) = 8\,T(n/2) + O(n^2)$$

$c = 2$

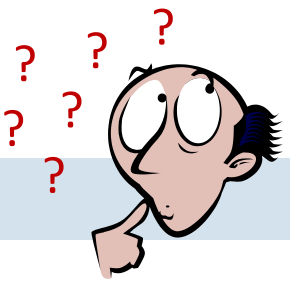
$a = 8 \quad b = 2 \implies \log_2 8 = 3$

- Πολυπλοκότητα αλγόριθμου ΡΜ

$$T(n) = \begin{cases} O(1) & n = 1 \\ 8\,T(n/2) + O(n^2) & n \geq 1 \end{cases}$$

- Από Κύριο Θεώρημα:

$$T(n) = O(n^3)$$



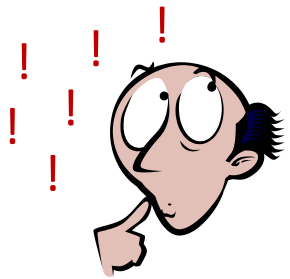
Καμία βελτίωση ως προς τον κλασσικό μας αλγόριθμο !!!

Αλγόριθμος Διαίρει-και-Βασίλευε

Αναδρομικός Πολλαπλασιασμός Πινάκων

- **`Algorithm RM(A,B)`**

Δεν αρκεί μόνο η καλή Αλγοριθμική Τεχνική !!!
Χρειάζεται και Ευφυΐα !!!



Αλγόριθμος Strassen

Αναδρομικός Πολλαπλασιασμός Πινάκων

- Παρουσιάζουμε τον αλγόριθμο του Strassen για τον υπολογισμό του γινομένου $C = A \cdot B$ δύο τετραγωνικών $n \times n$ πινάκων A και B, που εκτελείται σε χρόνο :

$$T(n) = O(n^{\log 7})$$

- Δεδομένου ότι η τιμή $\log 7$ είναι μεταξύ 2.80 και 2.81, ο αλγόριθμος του Strassen έχει χρόνο εκτέλεσης :

$$T(n) = O(n^{2.81})$$

και, επομένως, είναι ασυμπτωτικά ταχύτερος από τον κλασσικό αλγόριθμο του Λυκείου μας !!!

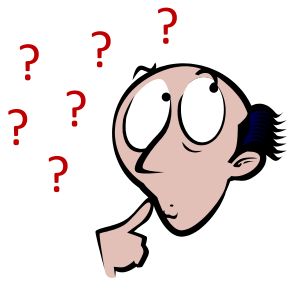
Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

- Είναι αλγόριθμος τεχνικής *Διαίρει-και-Βασίλευε* !!!
- Η ιδέα κλειδί του αλγόριθμου βασίζεται στην εξής επινόηση του **Strassen** :

> Ο πολλαπλασιασμός δύο 2×2 πινάκων A και B μπορεί να εκτελεστεί μόνο με **7** πολλαπλασιασμούς αντί για 8 (και 18 προσθέσεις/αφαιρέσεις αντί για 4) !!!

- Πόσο σημαντική είναι η εξάλειψη ενός πολλαπλασιασμού ?



- Ας δούμε πρώτα την ιδέα του !!!
  και μετά δείχνουμε *εάν είναι σημαντική* και *πόσο* !!!

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο 2×2 πινάκων A και B μπορεί να εκτελεστεί μόνο με **7** πολλαπλασιασμούς αντί για 8 (και 18 προσθέσεις/αφαιρέσεις αντί για 4) !!!

✓ Να η ιδέα του Strassen !!!

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

$$\begin{aligned} P_1 &= a \cdot (g - h) \\ P_2 &= (a + b) \cdot h \\ P_3 &= (c + d) \cdot e \\ P_4 &= d \cdot (f - e) \\ P_5 &= (a + d) \cdot (e + h) \\ P_6 &= (b - d) \cdot (f + h) \\ P_7 &= (a - c) \cdot (e + g) \end{aligned}$$

Όρισε τις ποσότητες $P_1, P_2, \ldots, P_7$ και στη συνέχεια...

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ s &= P_1 + P_2 \\ t &= P_3 + P_4 \\ u &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο 2×2 πινάκων A και B μπορεί να εκτελεστεί μόνο με **7** πολλαπλασιασμούς αντί για 8 (και 18 προσθέσεις/αφαιρέσεις αντί για 4) !!!

✓ Να η ιδέα του Strassen !!!

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

$$\begin{aligned} P_1 &= a \cdot (g - h) \\ P_2 &= (a + b) \cdot h \\ P_3 &= (c + d) \cdot e \\ P_4 &= d \cdot (f - e) \\ P_5 &= (a + d) \cdot (e + h) \\ P_6 &= (b - d) \cdot (f + h) \\ P_7 &= (a - c) \cdot (e + g) \end{aligned}$$

Πράγματι... η ιδέα του Strassen είναι σωστή !!!

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ &= (a + d) \cdot (e + h) + d \cdot (f - e) - (a + h) \cdot h + (b - d) \cdot (f + h) \\ &= ae + ah + de + dh + df - de - ah - bh + bf + bh - df - dh \\ &= ae + bf \end{aligned}$$

Επαλήθευση της τιμής του r !!!

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο 2×2 πινάκων **A** και **B** μπορεί να εκτελεστεί μόνο με **7** πολλαπλασιασμούς αντί για 8 (και 18 προσθέσεις/αφαιρέσεις αντί για 4) !!!

✓ Να η ιδέα του Strassen !!!

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

$$\begin{aligned} P_1 &= a \cdot (g - h) \\ P_2 &= (a + b) \cdot h \\ P_3 &= (c + d) \cdot e \\ P_4 &= d \cdot (f - e) \\ P_5 &= (a + d) \cdot (e + h) \\ P_6 &= (b - d) \cdot (f + h) \\ P_7 &= (a - c) \cdot (e + g) \end{aligned}$$

Πράγματι... η ιδέα του Strassen είναι σωστή !!!

$$\begin{aligned} s &= P_1 + P_2 = a \cdot g + b \cdot h \\ t &= P_3 + P_4 = c \cdot e + d \cdot f \\ u &= P_5 + P_1 - P_3 - P_7 = c \cdot h + d \cdot h \end{aligned}$$

Εύκολα μπορούμε να επαληθεύουμε τις τιμές των **s**, **t** και **u** !!!

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο 2×2 πινάκων **A** και **B** μπορεί να εκτελεστεί μόνο με **7** πολλαπλασιασμούς αντί για 8 (και 18 προσθέσεις/αφαιρέσεις αντί για 4) !!!

✓ Η ιδέα του Strassen είναι σωστή !!!

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

- Πόσο καλή όμως είναι η ιδέα ;
- Μας γλιτώνει ένα *πολλαπλασιασμό* !
- Όμως μας κοστίζει 14 επιπρόσθετες *προσθέσεις/αφαιρέσεις* !!
- Δεν θα είχε καμία αξία εάν η χρησιμότητά της περιοριζόταν μόνο σε πολλαπλασιασμό 2×2 πινάκων !!!

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο 2×2 πινάκων A και B μπορεί να εκτελεστεί μόνο με **7** πολλαπλασιασμούς αντί για 8 (και 18 προσθέσεις/αφαιρέσεις αντί για 4) !!!

Έχει γενικότερη χρησιμότητα !!!

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

Γιατί ?

- ✓ Κρίσιμο σημείο της Ιδέας !!!

  Η αντιμεταθετικότητα της πράξης του πολλαπλασιασμού δεν χρησιμοποιείται στους τύπους υπολογισμού των ποσοτήτων $P_1, P_2, \ldots, P_7$

- ✓ Άρα, μπορεί να εφαρμοστεί όταν τα a, b, c, d, e, f, g και h είναι πίνακες !!!

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο $n \times n$ πινάκων A και B μπορεί να εκτελεστεί μόνο με **7** αναδρομικούς πολλαπλασιασμούς $n/2 \times n/2$ πινάκων αντί για 8 !!!

Αναδρομική εφαρμογή της μεθόδου του Strassen σε *πίνακες* !!!

- Η εξάλειψη *ενός πολλαπλασιασμού πινάκων* κοστίζει μερικές επιπλέον νέες προσθέσεις $n/2 \times n/2$ πινάκων !!!
- Όμως, το πλήθος αυτών των προσθέσεων είναι σταθερό !!!

  και θα απορροφηθεί στον όρο $O(\cdot)$ της αναδρομικής σχέσης που εκφράζει το χρόνο εκτέλεσης του αλγόριθμου.

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο **n** × **n** πινάκων **A** και **B** μπορεί να εκτελεστεί μόνο με **7** αναδρομικούς πολλαπλασιασμούς **n/2** × **n/2** πινάκων αντί για **8** !!!

✓ Γενίκευση της ιδέας !!!

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 \\ P_2 &= S_2 \cdot B_{22} \\ P_3 &= S_3 \cdot B_{11} \\ P_4 &= A_{22} \cdot S_4 \\ P_5 &= S_5 \cdot S_6 \\ P_6 &= S_7 \cdot S_8 \\ P_7 &= S_9 \cdot S_{10} \end{aligned}$$

όπου $S_1, S_2, \ldots, S_{10}$ πίνακες διαστάσεων **n/2** × **n/2** :

$$\begin{aligned} S_1 &= B_{12} + B_{22} \\ S_2 &= A_{11} + A_{12} \\ S_3 &= A_{21} + A_{22} \\ S_4 &= B_{21} + B_{11} \\ S_5 &= A_{11} + A_{22} \end{aligned} \qquad \begin{aligned} S_6 &= B_{11} + B_{22} \\ S_7 &= A_{12} + A_{22} \\ S_8 &= B_{21} + B_{22} \\ S_9 &= A_{11} + A_{21} \\ S_{10} &= B_{11} + B_{12} \end{aligned}$$

Αλγόριθμος Strassen

Ιδέα Αλγόριθμου !!!...

Ο πολλαπλασιασμός δύο **n** × **n** πινάκων **A** και **B** μπορεί να εκτελεστεί μόνο με **7** αναδρομικούς πολλαπλασιασμούς **n/2** × **n/2** πινάκων αντί για **8** !!!

✓ Γενίκευση της ιδέας !!!

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 \\
P_2 &= S_2 \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 \\
P_5 &= S_5 \cdot S_6 \\
P_6 &= S_7 \cdot S_8 \\
P_7 &= S_9 \cdot S_{10}
\end{aligned}
$$

$$
\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}
$$

Αναδρομικός υπολογισμός $P_1, P_2, \ldots, P_7$ και έξοδος ο C :

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

Αλγόριθμος Strassen

Γενική περιγραφή

Αλγόριθμος

Η ιδέα αυτή υπαγορεύει τον παρακάτω αναδρομικό αλγόριθμο του Strassen για τον πολλαπλασιασμό δύο $n \times n$ πινάκων:

1. "Διαίρεσε" τους πίνακες A και B σε 4 υποπίνακες $A_{11}, A_{12}, A_{21}, A_{22}$ και $B_{11}, B_{12}, B_{21}, B_{22}$ τον καθένα διαστάσεων $n/2 \times n/2$.

2. "Κατέκτησε" τους πίνακες $P_1, P_2, \ldots, P_7$ κάνοντας 7 πολλαπλασιασμούς $n/2 \times n/2$ πινάκων αναδρομικά (για τον υπολογισμό των πινάκων $P_1, P_2, \ldots, P_7$ χρησιμοποίησε 14 συνολικά $n/2 \times n/2$ πίνακες - τους 8 πίνακες $A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22}$ και τους 10 πίνακες $S_1, S_2, \ldots, S_{10}$).

3. "Συνδύασε" τα επιμέρους αποτελέσματα και υπολόγισε τον πίνακα $C = A \cdot B$ χρησιμοποιώντας προσθέσεις/αφαιρέσεις πάνω στους 7 πίνακες $P_1, P_2, \ldots, P_7$.

Αλγόριθμος Strassen

Αναλυτική περιγραφή

Αλγόριθμος

1. Διαίρεσε τούς πίνακες A και B και τον πίνακα εξόδου C σε $n/2 \times n/2$ υποπίνακες. Χρόνος $\Theta(1)$.

$$A = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \quad B = \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) \quad C = \left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right)$$

Δημιούργησε 10 πίνακες $S_1, S_2, \ldots, S_{10}$ διαστάσεων $n/2 \times n/2$ που είναι το άθροισμα ή η διαφορά δύο πινάκων που δημιουργήθηκαν στο βήμα 1. Χρόνος $\Theta(n^2)$.

$$\begin{aligned} S_1 &= B_{12} + B_{22} \\ S_2 &= A_{11} + A_{12} \\ S_3 &= A_{21} + A_{22} \\ S_4 &= B_{21} + B_{11} \\ S_5 &= A_{11} + A_{22} \end{aligned} \qquad \begin{aligned} S_6 &= B_{11} + B_{22} \\ S_7 &= A_{12} + A_{22} \\ S_8 &= B_{21} + B_{22} \\ S_9 &= A_{11} + A_{21} \\ S_{10} &= B_{11} + B_{12} \end{aligned}$$

Αλγόριθμος Strassen

Αναλυτική περιγραφή

Αλγόριθμος

2. Υπολόγισε αναδρομικά επτά γινόμενα πινάκων $P_1, P_2, \ldots, P_7$ χρησιμοποιώντας τους υποπίνακες των βημάτων 1 και 2. Κάθε P_i έχει διαστάσεις $n/2 \times n/2$.

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}
$$

3. Υπολόγισε τους ζητούμενους υποπίνακες $C_{11}, C_{12}, C_{21}, C_{22}$ του τελικού πίνακα C προσθέτοντας και αφαιρώντας διάφορους συνδυασμούς των πινάκων P_i. Χρόνος $\Theta(n^2)$.

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

Αλγόριθμος Strassen

Ορθότητα Αλγόριθμου

- Στο βήμα 2, υπολογίζει τους πίνακες P_1, P_2, …, P_7 κάνοντας 7 πολλαπλασιασμούς $n/2 \times n/2$ πινάκων αναδρομικά, όπου καθένας είναι άθροισμα ή διαφορά γινομένων των υποπινάκων A_{11}, A_{12}, A_{21}, A_{22}, B_{11}, B_{12}, B_{21}, B_{22} των A και B:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}
$$

- Στο βήμα 3, προσθέτει και αφαιρεί τους πίνακες P_1, P_2, …, P_7 και κατασκευάζει τους τέσσερις $n/2 \times n/2$ υποπίνακες C_{11}, C_{12}, C_{21}, C_{22} του γινομένου $C = A \cdot B$.

Αλγόριθμος Strassen

Ορθότητα Αλγόριθμου

- Ας ξεκινήσουμε από τον όρο C_{11} :

$$C_{11} = P_5 + P_4 - P_2 + P_6$$

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}$$

Αν γράψουμε αναλυτικά το δεξί μέλος, αναπτύσσοντας κάθε P_i σε μία γραμμή και στοιχίζοντας κατακόρυφα όρους που απαλείφονται, βλέπουμε ότι το C_{11} ισούται με:

$$\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}$$

$$\begin{array}{llllllll}
A_{11} \cdot B_{11} & + A_{11} \cdot B_{22} & + A_{22} \cdot B_{11} & + A_{22} \cdot B_{22} & & & & \\
 & & - A_{22} \cdot B_{11} & & + A_{22} \cdot B_{21} & & & \\
 & - A_{11} \cdot B_{22} & & & & - A_{12} \cdot B_{22} & & \\
 & & & - A_{22} \cdot B_{22} & - A_{22} \cdot B_{21} & + A_{12} \cdot B_{22} & + A_{12} \cdot B_{21} & \\
\hline
A_{11} \cdot B_{11} & & & & & & A_{12} \cdot B_{21} &
\end{array}$$

Αλγόριθμος Strassen

Ορθότητα Αλγόριθμου

- Αντίστοιχα, παίρνουμε τον όρο C_{12} :

$$C_{12} = P_1 + P_2$$

οπότε αυτός ισούται με:

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}$$

$$\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}$$

$$\begin{array}{l}
A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
\qquad + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
\hline
A_{11} \cdot B_{12} \qquad\qquad + A_{12} \cdot B_{22}
\end{array}$$

Αλγόριθμος Strassen

Ορθότητα Αλγόριθμου

- Συνεχίζοντας, παίρνουμε τον όρο C_{21} :

$$C_{21} = P_3 + P_4$$

οπότε αυτός ισούται με:

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}$$

$$\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}$$

$$\begin{array}{l}
A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
\qquad - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\
\hline
A_{21} \cdot B_{11} \qquad\qquad + A_{22} \cdot B_{21}
\end{array}$$

Αλγόριθμος Strassen

Ορθότητα Αλγόριθμου

- Τέλος, ο τέταρτος όρος C_{22} :

$$C_{22} = P_5 + P_1 - P_3 - P_7$$

ισούται με:

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{22} \\
P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}
\end{aligned}$$

$$\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}$$

$$\begin{array}{llll}
A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} & & & \\
- A_{11} \cdot B_{22} & & + A_{11} \cdot B_{12} & \\
- A_{22} \cdot B_{11} & & - A_{21} \cdot B_{11} & \\
 & & - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} & \\
\hline
A_{22} \cdot B_{22} & & A_{12} \cdot B_{12} &
\end{array}$$

Αλγόριθμος Strassen

Πολυπλοκότητα Αλγόριθμου

- Η αναδρομική σχέση που εκφράζει το χρόνο εκτέλεσης του αλγόριθμου:

$$T(n) = \begin{cases} O(1) & n = 1 \\ 7\,T(n/2) + O(n^2) & n \geq 1 \end{cases}$$

Ο αλγόριθμος εκτελεί
7 πολλαπλασιασμούς και
18 προσθέσεις/αφαιρέσεις
σε πίνακες διατάσεων n/2 × n/2

$$T(n) = 7 \cdot T(\frac{n}{2}) + 18 \cdot (\frac{n}{2})^2$$
$$= 7 \cdot T(\frac{n}{2}) + O(n^2)$$

- Από Κύριο Θεώρημα: $T(n) = O(n^{\log 7})$

$c = 2$

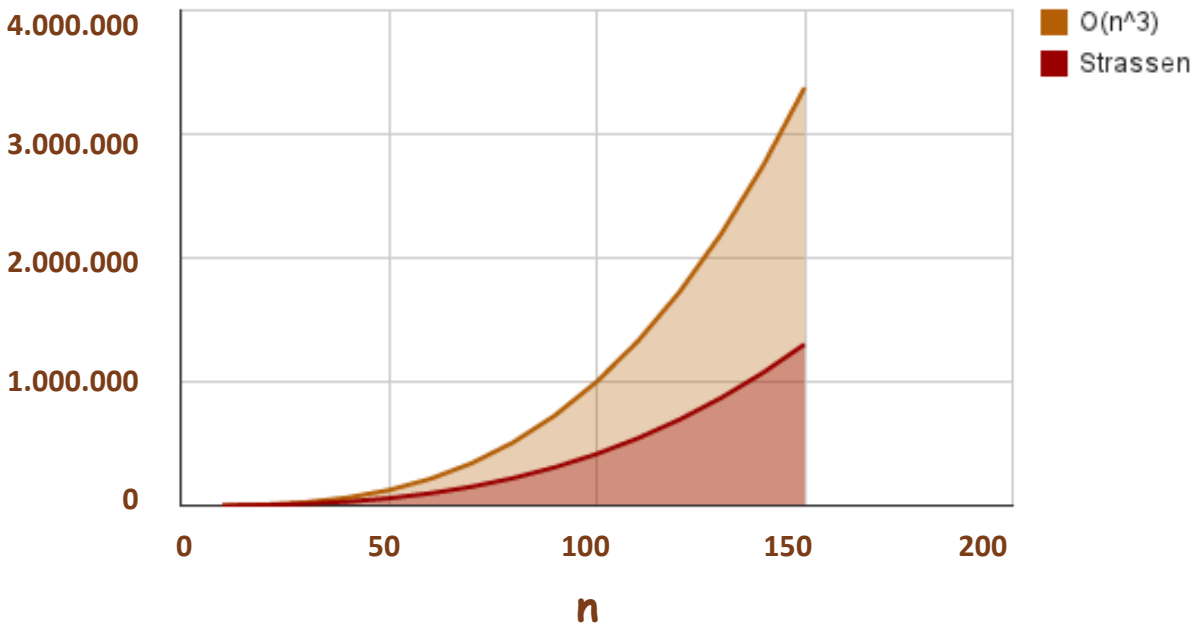
$a = 7 \quad b = 2 \implies \log_2 7 > c = 2$

Πολυπλοκότητα αλγόριθμου: $T(n) = O(n^{2.81})$

Divide & Conquer

Πολλαπλασιασμός Πινάκων

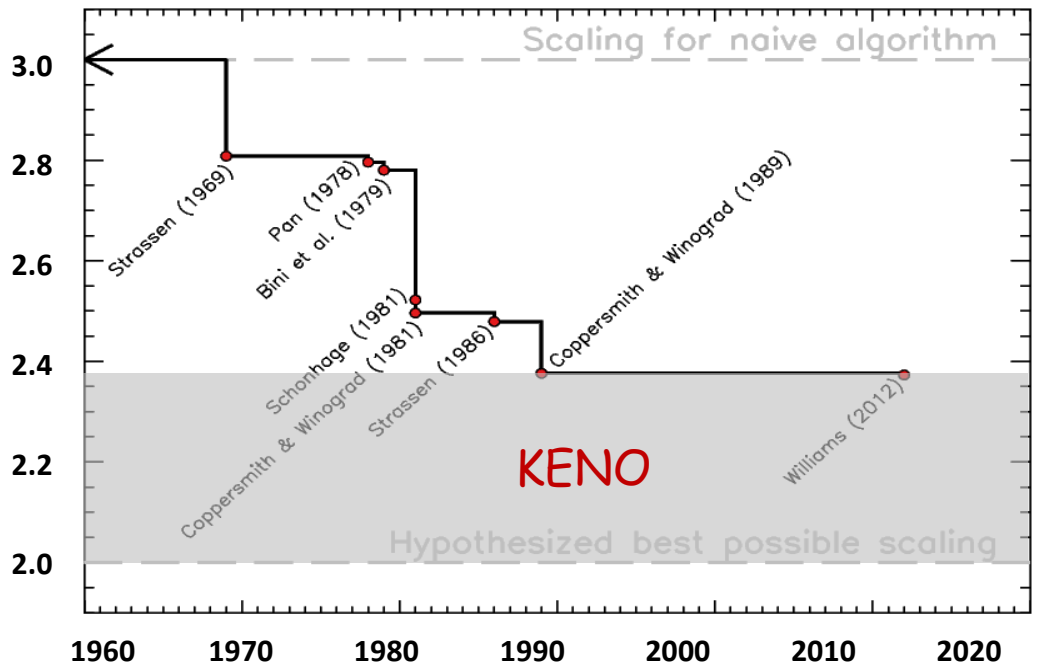
Αλγόριθμος $O(n^3)$ σε σχέση με τον αλγόριθμο του Strassen $O(n^{2.81})$



Strassens's Matrix Multiplication

Πολλαπλασιασμός Πινάκων

Και λίγα Ιστορικά στοιχεία για την «Πρόοδο» του Προβλήματος



Πολυπλοκότητα $O(n^3)$ Κλασσικού Αλγορίθμου

Strassen $O(n^{2.81})$

Άνω φράγμα του προβλήματος $O(n^{2.373})$

Κάτω φράγμα του προβλήματος $\Omega(n^2)$

Strassens's Matrix Multiplication

Ευχαριστώ για τη Συμμετοχή σας !!!

Σταύρος Δ. Νικολόπουλος

Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Πανεπιστήμιο Ιωαννίνων

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