Αλγόριθμοι και Πολυπλοκότητα

Ν. Μ. Μισυρλής

Τμήμα Πληροφορικής και Τηλεπικοινωνιών,
Πανεπιστήμιο Αθηνών

Αναδρομή

Διαίρει και Βασίλευε (Divide and Conquer)

- Η *αναδρομή* είναι μια ισχυρή αλγοριθμική μέθοδος καθώς και μια τεχνική προγραμματισμού (αναδρομικές συναρτήσεις, αναδρομικός ορισμός τύπου).
- Ένας αναδρομικός αλγόριθμος καλεί τον εαυτό του επιλύοντας υποπροβλήματα του αρχικού προβλήματος.
- Υποπρόβλημα: ένα πρόβλημα της ίδιας φύσης με το αρχικό, αλλά μικρότερου μεγέθους.

Αναδρομή

Διαίρει και Βασίλευε (Divide and Conquer)

- **διαίρει και βασίλευε (Divide and Conquer)** το αρχικό πρόβλημα διασπάται σε υποπροβλήματα τα οποία επιλύονται και συνδυάζονται οι λύσεις τους για να δημιουργηθεί η λύση του αρχικού προβλήματος.

  Διαίρεση: Διαίρεση του προβλήματος σε μικρότερα προβλήματα (υποπροβλήματα).

  Επίλυση: Επίλυση των μικρότερων προβλημάτων αναδρομικά.

  Συνδυασμός: Συνδυασμός των λύσεων για την εύρεση της αρχικής λύσης.

Αναδρομή

Merge-Sort

- **Διαίρεση:** Διαίρεση του πίνακα n στοιχείων σε δύο υποπίνακες $\frac{n}{2}$ στοιχείων ο καθένας.
- **Επίλυση:** Αναδρομική ταξινόμηση των δύο υποπινάκων με χρήση της MERGE-SORT.
- **Συνδυασμός:** Συγχώνευση των δύο ταξινομημένων υποπινάκων.
- Το βήμα **Επίλυση** σταματά όταν οι υποπίνακες οι οποίοι προκύπτουν μετά τη διάσπαση έχουν μόνο ένα στοιχείο.
- Η συγχώνευση δύο ταξινομημένων υποπινάκων γίνεται με τον αλγόριθμο MERGE

Αναδρομή

Merge-Sort

```
     Merge (a, b, c)
1.   i = 1
2.   j = 1
3.   for k = 1 to m + n do
4.        if a_i ≤ b_j then
5.             c_k = a_i
6.             i = i + 1
7.        else
8.             c_k = b_j
9.             j = j + 1
10.       end if
11.  end for
```

Σχήμα: Αλγόριθμος MERGE για την συγχώνευση των ταξινομημένων πινάκων a (μήκους m) και b (μήκους n) στον πίνακα c (μήκους $m + n$).

Αναδρομή

Merge-Sort

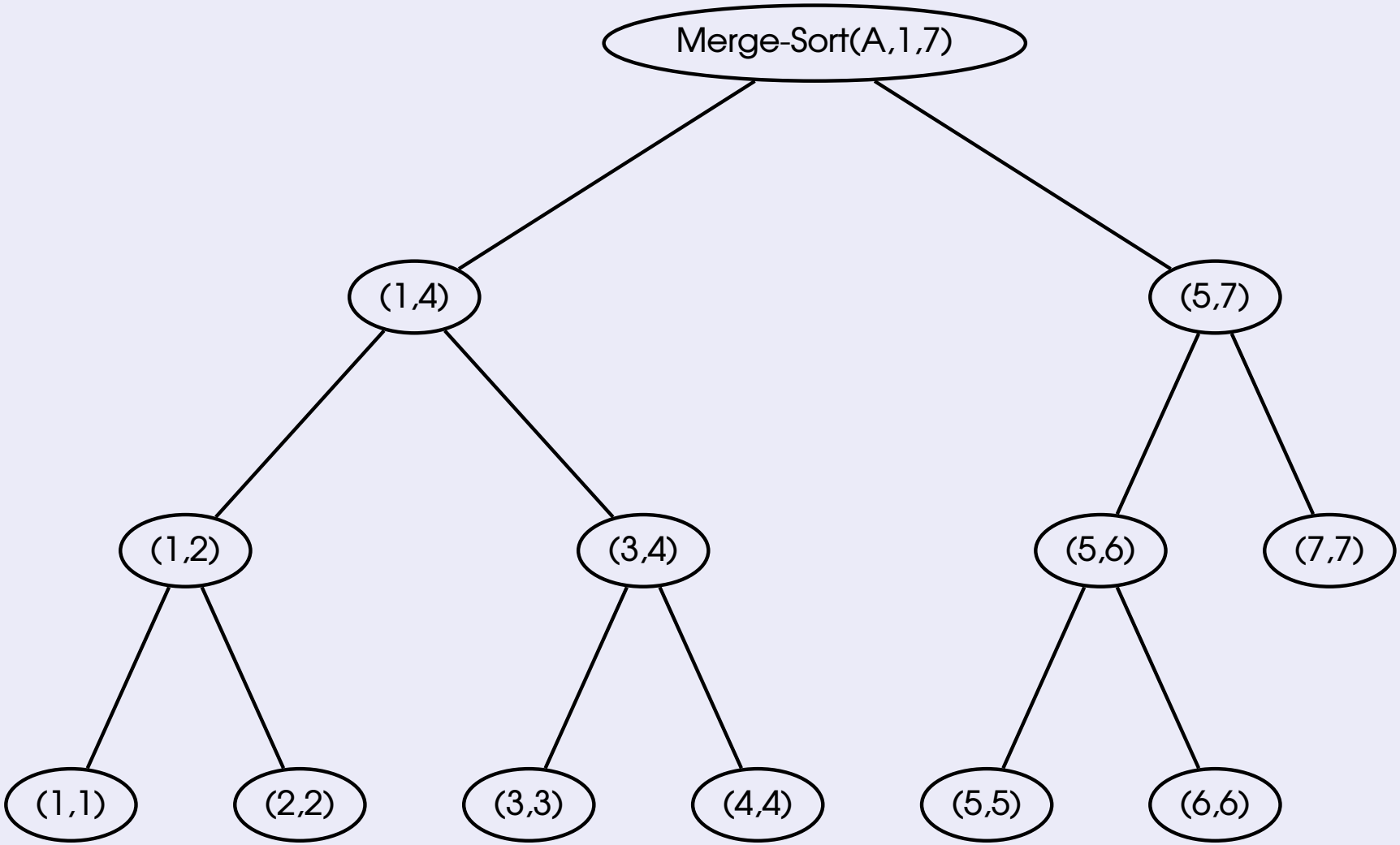
Η πολυπλοκότητα του αλγορίθμου MERGE είναι $O(m+n)$.

```
     MergeSort (A, l, r)
1.   if l < r
2.         q ← ⌊(l + r)/2⌋
3.         MergeSort(A,l,q)
4.          MergeSort(A,q+1,r)
5.          Merge /* Τους δύο υποπίνακες */
```

Σχήμα: Αναδρομικός αλγόριθμος MERGESORT για την ταξινόμηση των στοιχείων ενός πίνακα.

Αναδρομή

Merge-Sort



Αναδρομή

Merge-Sort

- *Αναδρομική εξίσωση* περιγράφει το συνολικό χρόνο εκτέλεσης του αλγορίθμου σε ένα πρόβλημα μεγέθους n χρησιμοποιώντας το χρόνο εκτέλεσης σε προβλήματα μικρότερου μεγέθους.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(\frac{n}{2}) + \Theta(n) & \text{if } n > 1. \end{cases}$$

- $T(n) = \Theta(n \log n)$
- Γενικά, η υποδιαίρεση του αρχικού προβλήματος σε a υποπροβλήματα το καθένα από τα οποία έχει μέγεθος $1/b$ του αρχικού
  και
- το βήμα συνδυασμός χρειάζεται χρόνο $d(n)$, τότε

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ aT(\frac{n}{b}) + d(n) & \text{if } n > 1. \end{cases}$$

Αναδρομή

Merge-Sort

Μέθοδος της *αντικατάστασης (substitution)*

$$
\begin{aligned}
T(n) &= aT(n/b) + d(n) \\
&= a[aT(n/b^2) + d(n/b)] + d(n) \\
&= a^2T(n/b^2) + ad(n/b) + d(n) \\
&= a^2[aT(n/b^3) + d(n/b^2)] + ad(n/b) + d(n) \\
&= a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n) \\
&= \dots \\
&= a^iT(n/b^i) + \sum_{j=0}^{i-1} a^jd(n/b^j)
\end{aligned}
$$

Αναδρομή

Merge-Sort

Αν $n/b^k = 1$ ή $n = b^k$ τότε $T(n/b^k) = T(1) = 1$ και για $i = k$:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

$$k = \log_b n$$

$$a^k = a^{\log_b n} = n^{\log_b a}.$$

Αναδρομή

Merge-Sort

Στην περίπτωση της MERGE-SORT έχουμε $a = b = 2$ και $d(n) = n - 1 (= \Theta(n))$.

$$
\begin{aligned}
T(n) &= n + \sum_{j=0}^{k-1} 2^j (2^{k-j} - 1) \\
&= n + \sum_{j=0}^{k-1} (2^k - 2^j) \\
&= n + 2^k k - \frac{(2^k - 1)}{(2 - 1)} \\
&= n + n \log n - n + 1 \\
&= n \log n + 1
\end{aligned}
$$

Δυαδική Αναζήτηση (Binary Search)

Το Πρόβλημα

- **Είσοδος :** ο πίνακας $A[1..n]$, του οποίου τα στοιχεία είναι ταξινομημένα και ένα στοιχείο x.
- **΄Εξοδος :** Η εύρεση της θέσης του x στον A.

Αναδρομή

Binary Search

Διαίρεση: Βρίσκουμε το μεσαίο στοιχείο του πίνακα και τον χωρίζουμε σε δύο υποπίνακες.

Επίλυση: Επαναλαμβάνουμε την παραπάνω διαδικασία στον επιλεγμένο υποπίνακα, μεχρίς ότου βρούμε το στοιχείο σαν μεσαίο ή μέχρι να έχουμε κενούς υποπίνακες.

Συνδυασμός: Δεν υπάρχει βήμα Συνδυασμός.

Αναδρομή

Binary Search

```
     BinarySearch (A, l, r, x)
1.   if l < r
2.         q ← ⌊(l + r)/2⌋
3.          if x < A[q]
4.                BinarySearch(A,l,q,x)
5.          else if x > A[q]
6.                BinarySearch(A,q+1,r,x)
7.          else
8.                return q
```

Σχήμα: Αναδρομικός αλγόριθμος BinarySearch.

$$T(n) = T(n/2) + c = O(logn)$$

Αντιστροφή (Inversion)

Το Πρόβλημα

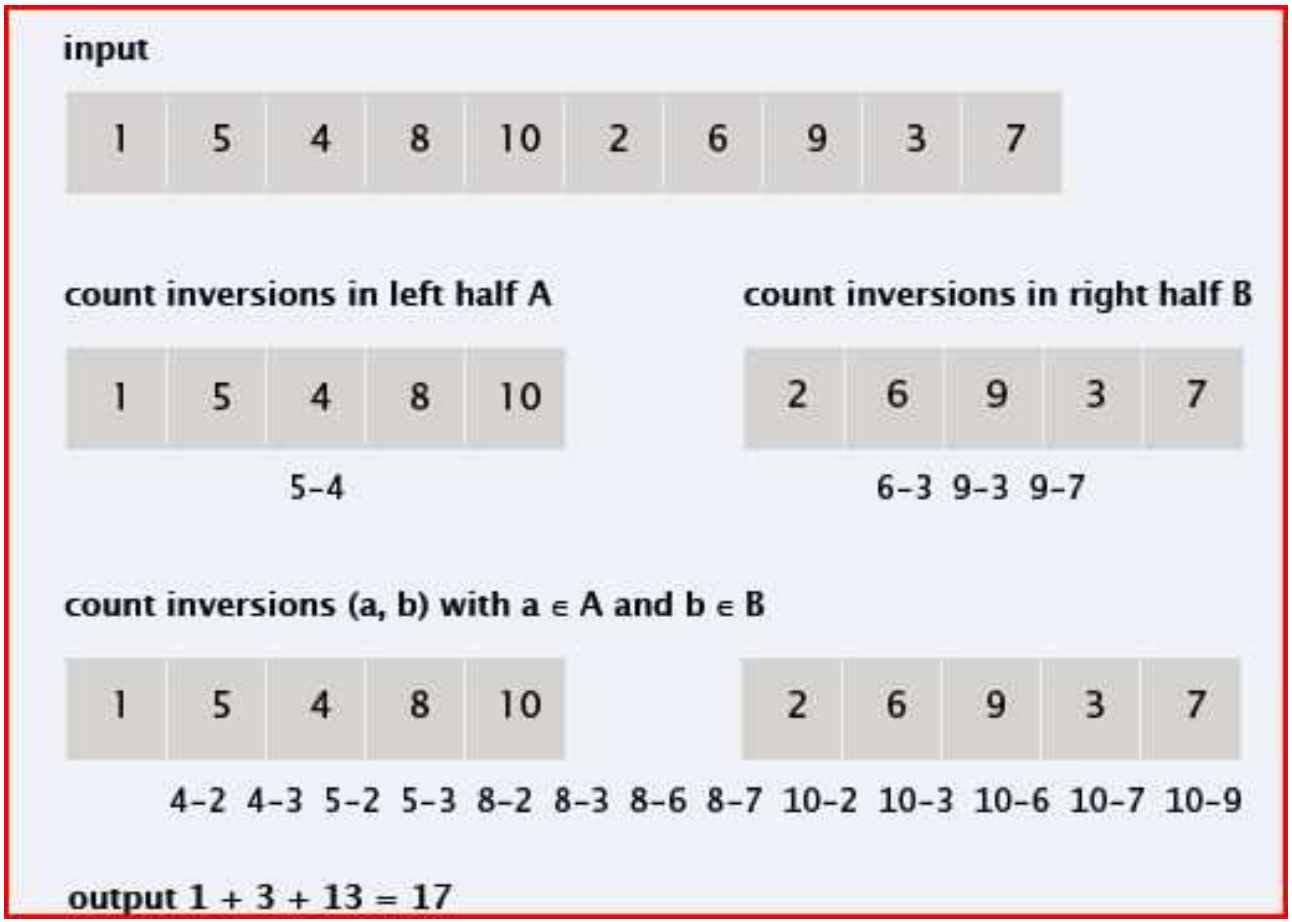
- **Είσοδος :** ο πίνακας L περιέχει τους αριθμούς $1, 2, 3, \ldots, n$ με κάποια διάταξη.
- **΄Εξοδος :** πλήθος αντιστροφών = πλήθος των ζευγών (i, j) θέσεων πίνακα με $i < j$ και $L[i] > L[j]$

Αντιστροφή (Inversion)

Σχεδιασμός και Ανάλυση Αλγορίθμου

- Ποιός είναι ο απλούστερος αλγόριθμος ;
- Ωμή βία (Brute force) με πολυπλοκότητα $O(n^2)$.
- Μπορούμε να βελτιώσουμε τον ανωτέρω αλγόριθμο ;
- Ναι, με πολυπλοκότητα $O(n \log n)$.
- Παράξενο, γιατί αν έχουμε n^2 αντιστροφές, τότε αναμένουμε $O(n^2)$
- Συνεπώς ο αλγόριθμος θα πρέπει να υπολογίζει το πλήθος των αντιστροφών χωρίς να εξετάζει μεμονωμένα κάθε αντιστροφή.

Διαίρει και Βασίλευε- Αντιστροφή



Διαίρει και Βασίλευε- Αντιστροφή

Άντιστροφή-Inversion

- **Διαίρει:** Διαίρεση του πίνακα n στοιχείων σε δύο υποπίνακες $\frac{n}{2}$ στοιχείων ο καθένας A, B.
- **Βασίλευε:** Αναδρομική κλήση της Αντιστροφής σε κάθε υποπίνακα για τον υπολογισμό του πλήθους των αντιστροφών του.
- **Συνδυασμός:** Υπολογισμός του πλήθους των αντιστροφών (a, b) όπου $a \in A$ και $b \in B$.
- Επιστρέφει το άθροισμα του πλήθους των τριών αντιστροφών.

Αντιστροφή

Αλγόριθμος- 1η Εκδοχή

```
    Count (L, n)
1.  if n = 1 return 0
2.  else
3.        x=Count(A,n/2)
4.        y= Count(B,n/2)
5.        z= CountSplitInv(A,B) /* δεν έχει υλοποιηθεί */
6.  return x+y+z
```

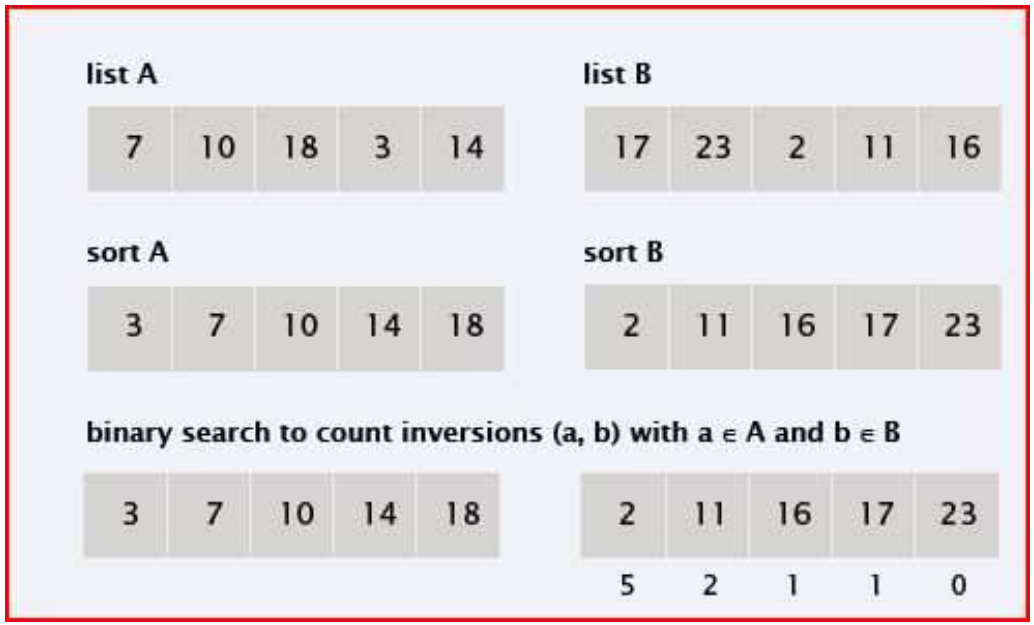
Σχήμα: Αναδρομικός αλγόριθμος Count για την εύρεση του πλήθους των αντιστροφών των στοιχείων ενός πίνακα.

Στόχος : Η CountSplitInv(L,n) να έχει γραμμική $O(n)$ πολυπλοκότητα γιατί τότε η Count θα έχει πολυπλοκότητα $O(n \log n)$ όπως και η Merge-Sort.

Αντιστροφή

Αλγόριθμος-Ανάλυση

- Ταξινόμηση των A, B.
- Για κάθε στοιχείο του $b \in B$, δυαδική αναζήτηση στον A για την εύρεση του πλήθους των στοιχείων του που είναι μεγαλύτερα από το b.



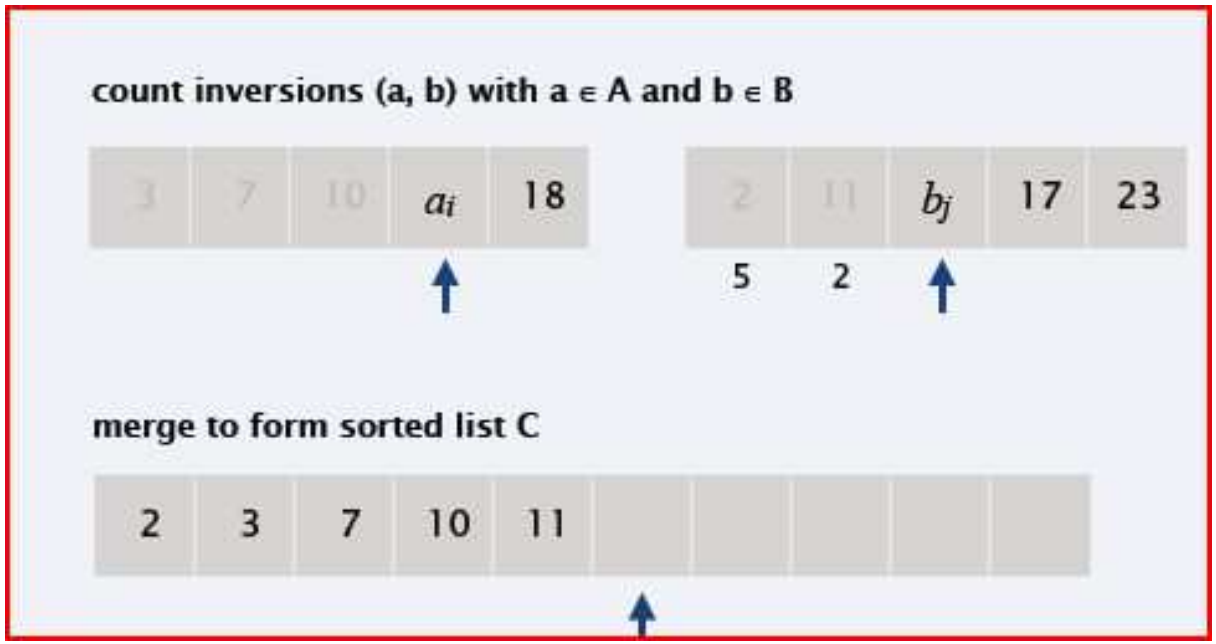
Αντιστροφή

Υπολογισμός του πλήθους των αντιστροφών (a, b)

Αν οι πίνακες A, B είναι ταξινομημένοι, τότε Συγχώνευση (Merge).

- Σάρωση των A, B από αριστερά προς τα δεξιά.
- Σύγκριση των a_i, b_j.
- Αν $a_i < b_j$ τότε το a_i δεν αντιστρέφεται με κανένα εναπομένον στοιχείο του B.
- Αν $a_i > b_j$ τότε το b_j αντιστρέφεται με όλα τα εναπομένοντα στοιχεία του A.
- Τοποθέτηση του μικρότερου στοιχείου στον ταξινομημένο πίνακα C.

Αντιστροφή


count inversions (a, b) with a ∈ A and b ∈ B
3
7
10
a_i
18
2
11
b_j
17
23
5
2
merge to form sorted list C
2
3
7
10
11

Αντιστροφή

Αλγόριθμος-Ανάλυση

- Όπως στην Merge-Sort, οι αναδρομικές κλήσεις να βρίσκουν το πλήθος των αντιστροφών και να ταξινομούν.
- Κατ' αυτόν τον τρόπο η Merge εντοπίζει τις αντιστροφές που υπάρχουν στους A και B.

Αντιστροφή

Αλγόριθμος

ΕΙΣΟΔΟΣ: Ο πίνακας L.

ΕΞΟΔΟΣ: Το πλήθος των αντιστροφών στον L και ο ταξινομημένος πίνακας L'.

```
    SortCount (L, n)
1.  if n = 1 return 0
2.  else
3.          (x, A)=SortCount(A,n/2)
4.          (y, B)= SortCount(B,n/2)
5.          (z, L')= MergeCountSplitInv(A,B) /* όμοια της Merge */
6.  return (x+y+z, L')
```

Σχήμα: Αναδρομικός αλγόριθμος SortCount για την εύρεση του πλήθους των αντιστροφών των στοιχείων ενός πίνακα.

Αντιστροφή- Ανάλυση αλγορίθμου

Χρόνος χειρότερης περίπτωσης

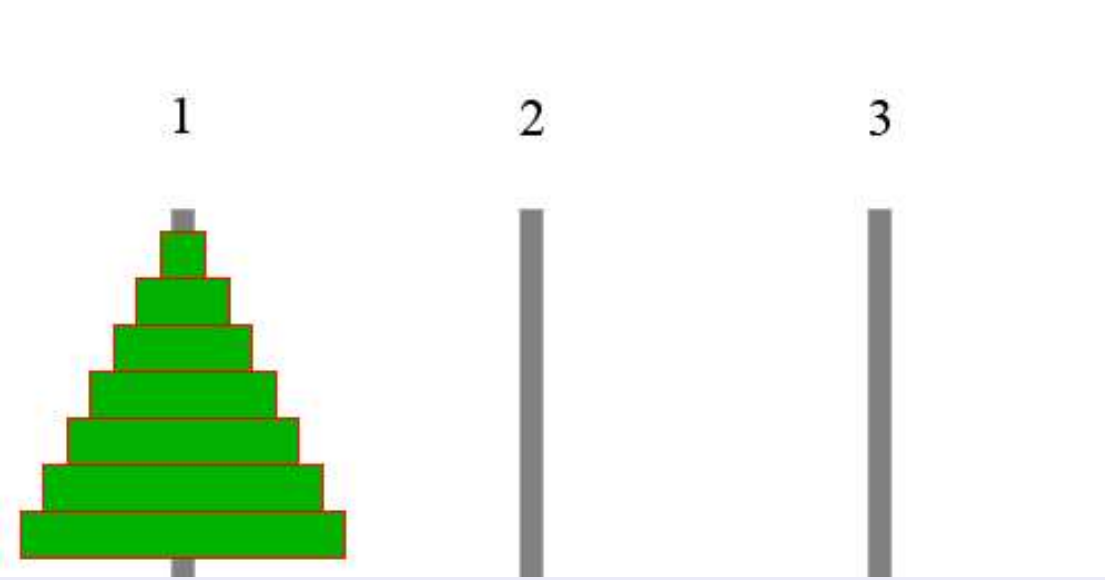
$$T(n) = \begin{cases} \Theta(1) & , \text{για } n = 1 \\ 2T(\frac{n}{2}) + \Theta(n) & , \text{για } n > 1 \end{cases}$$

Χρόνος εκτέλεσης SortCount

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

Αναδρομή

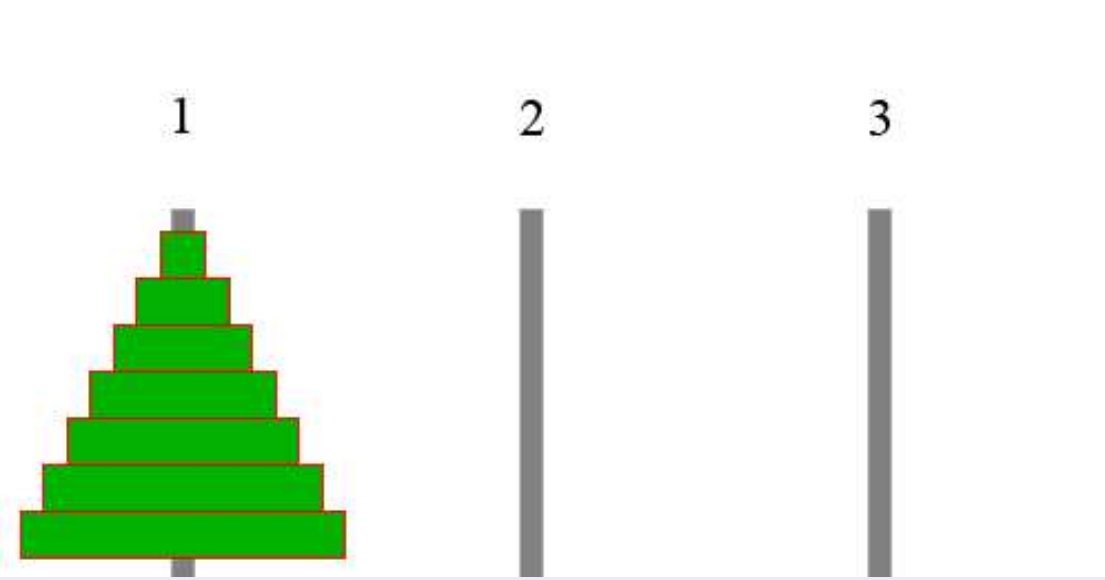
Οι πύργοι του Hanoi



Σχήμα: Οι πύργοι του Hanoi

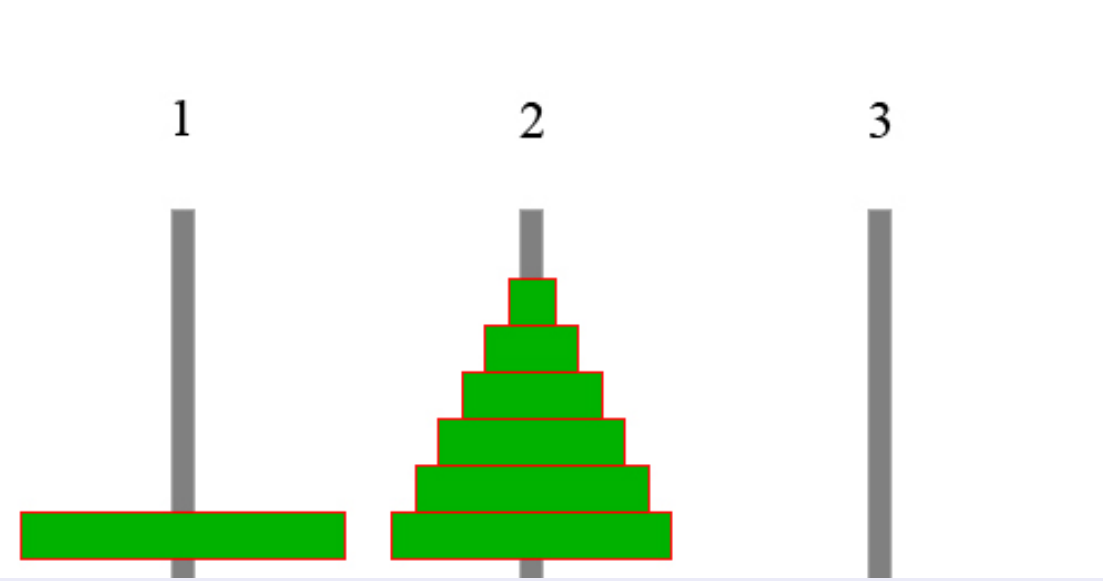
Αναδρομή

Οι πύργοι του Hanoi - Σχηματική λειτουργία του αλγορίθμου



Σχήμα: Αρχική κατάσταση.

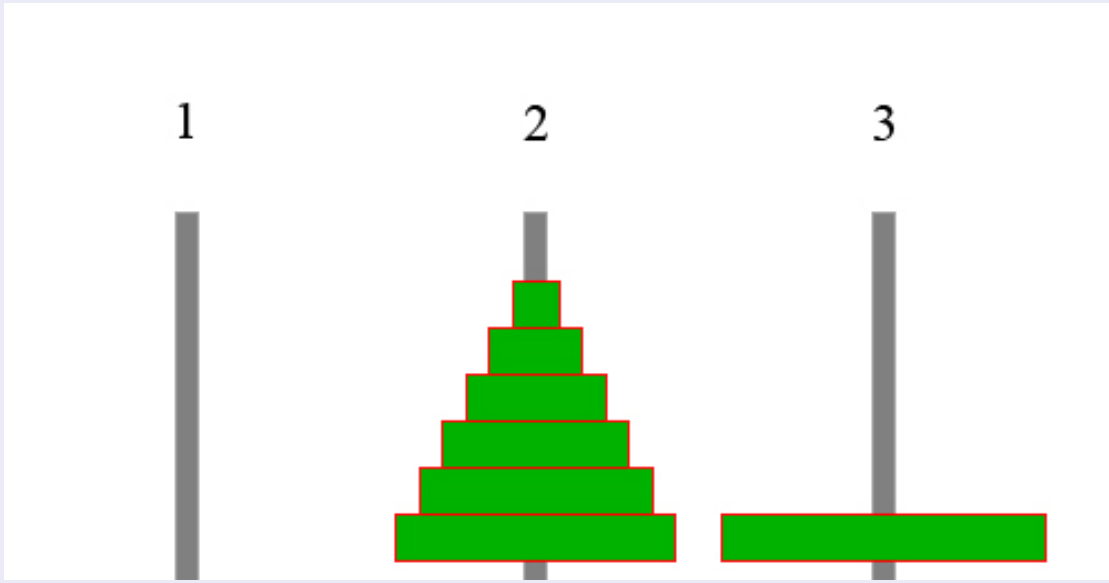
Οι πύργοι του Hanoi - Σχηματική λειτουργία του αλγορίθμου



Σχήμα: $Hanoi(n-1,1,2)$

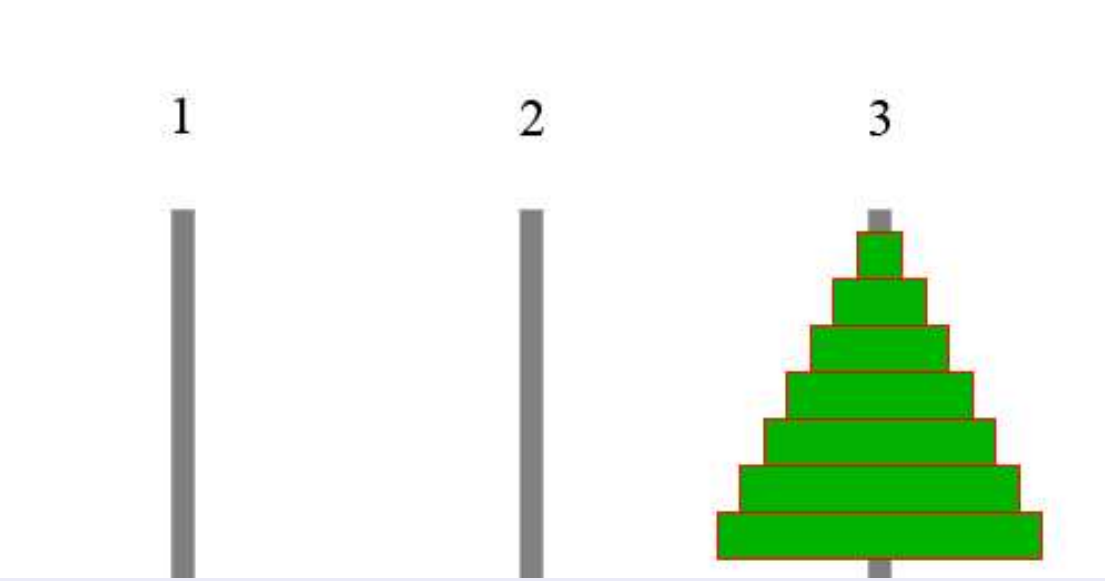
Αναδρομή

Οι πύργοι του Hanoi - Σχηματική λειτουργία του αλγορίθμου



Σχήμα: Μετακίνηση του μεγάλου δίσκου από το στύλο 1 στον στύλο 3.

Οι πύργοι του Hanoi - Σχηματική λειτουργία του αλγορίθμου



Σχήμα: $Hanoi(n-1,2,3)$

Αναδρομή

Οι πύργοι του Hanoi

HANOI (n, i, j)

1. **if** $n \geq 1$ **then**
2. HANOI$(n-1, i, 6-(i+j))$ /* χρησιμοποιώντας τον στύλο j */
3. Μεταφορά του μεγάλου δίσκου από τον στύλο i στον στύλο j
4. HANOI$(n-1, 6-(i+j), j)$ /* χρησιμοποιώντας τον στύλο i */
5. **end if**

Αναδρομή

Οι πύργοι του Hanoi

```
    Hanoi (n, 1, 3)
1.  if n ≥ 1 then
2.      Hanoi(n − 1, 1, 2)   /* χρησιμοποιώντας τον στύλο 3 */
3.      Μεταφορά του μεγάλου δίσκου από τον στύλο 1 στον στύλο 3
4.      Hanoi(n − 1, 2, 3)   /* χρησιμοποιώντας τον στύλο 1 */
5.  end if
```

Αναδρομή

Οι πύργοι του Hanoi

$$T(n) = \begin{cases} 0 & \text{if } n = 0, \\ 2T(n-1) + 1 & \text{if } n \geq 1. \end{cases}$$

Μέθοδος των *αθροιζομένων παραγόντων*

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \quad (\times 2^0) \\ T(n-1) &= 2T(n-2) + 1 \quad (\times 2^1) \\ T(n-2) &= 2T(n-3) + 1 \quad (\times 2^2) \\ &\dots \\ T(2) &= 2T(1) + 1 \quad (\times 2^{n-2}) \\ T(1) &= 2T(0) + 1 \quad (\times 2^{n-1}) \end{aligned}$$

Αναδρομή

Οι πύργοι του Hanoi

Αθροίζοντας

$$T(n) = 2^n T(0) + (1 + \ldots + 2^{n-1}) = 2^n 0 + \frac{2^n - 1}{2 - 1}$$

και τελικά

$$T(n) = 2^n - 1, \qquad n \geq 0$$

Γενική μορφή μιας αναδρομικής εξίσωσης

$$a(n)T_n = b(n)T_{n-1} + c(n)$$

με T_0 σταθερά και $a(n), b(n), c(n)$ συναρτήσεις του n.

Αναδρομή

Άσκηση:

Να επιλυθεί η αναδρομική εξίσωση $T(n) = T(n-1) + 2$ με $T(1) = 1$.

Λύση:

$$\begin{aligned} T(n) &= 2 + T(n-1) \\ &= 2 + 2 + T(n-2) \\ &\dots \\ &= 2 + \dots + 2 + T(1) \\ &= (n-1) \times 2 + 1 \\ &\leq 2n \end{aligned}$$

$$T(n) = \Theta(n)$$

Πολλαπλασιασμός ακεραίων

Έστω δυο ακέραιοι x και y με n ψηφία

Παράδειγμα

$$x = 1234 \quad y = 5678$$

Ο αλγόριθμος του πολλαπλασιασμού έχει πολυπλοκότητα

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

Αναδρομικός αλγόριθμος

Παρατηρήστε ότι αν χωρίσουμε στο μέσον τους ανωτέρω αριθμούς, τότε

$$x = 1234 = 12 \cdot 10^2 + 34 \text{ και } y = 5678 = 56 \cdot 10^2 + 78$$

Πολλαπλασιασμός ακεραίων

Αναδρομικός Αλγόριθμος

Γενικά

$$x = x_1 \cdot 10^{n/2} + x_0 \tag{1}$$

όπου x_1 είναι τα πρώτα $n/2$ και ο x_0 είναι τα τέλευταία $n/2$ ψηφία του x.

Όμοια

$$y = y_1 \cdot 10^{n/2} + y_0 \tag{2}$$

Από τις (1), (2) έχουμε

$$\begin{aligned} xy &= (x_1 \cdot 10^{n/2} + x_0)(y_1 \cdot 10^{n/2} + y_0) \\ &= x_1 y_1 \cdot 10^n + (x_1 y_0 + x_0 y_1) 10^{n/2} + x_0 y_0 \end{aligned} \tag{3}$$

Πολλαπλασιασμός ακεραίων

Αναδρομικός Αλγόριθμος

- Ο αναδρομικός αλγόριθμος υπολογίζει τα τέσσερα γινόμενα x_1y_1, x_1y_0, x_0y_1, x_0y_0
- στη συνέχεια συνδυάζει τα αποτελέσματα με τη χρήση της σχέσης (3).

1η Εκδοχή

RecursiveMultiply(x, y)

$x = x_1 \cdot 10^{n/2} + x_0$

$y = y_1 \cdot 10^{n/2} + y_0$

$x_1y_1 =$ *RecursiveMultiply*(x_1, y_1)

$x_1y_0 =$ *RecursiveMultiply*(x_1, y_0)

$x_0y_1 =$ *RecursiveMultiply*(x_0, y_1)

$x_0y_0 =$ *RecursiveMultiply*(x_0, y_0)

Return $x_1y_1 \cdot 10^n + (x_1y_0 + x_0y_1) \cdot 10^{n/2} + x_0y_0$

Πολλαπλασιασμός ακεραίων

Πολυπλοκότητα

- Στον ανωτέρω αλγόριθμο έχουν παραλειφθεί οι περιπτώσεις διακοπής της αναδρομής (περιπτώσεις βάσης).
- Η (3) απαιτεί ένα σταθερό πλήθος προσθέσεων αριθμών με $O(n)$ ψηφία συνεπώς έχει $O(n)$ χρόνο.
- Η πολυπλοκότητα του *RecursiveMultiply* δίνεται από την

$$T(n) \leq 4T(n/2) + cn \tag{4}$$

όπου $c =$ σταθερά.

- Η λύση της (4) είναι

$$T(n) \leq O(n^2)$$

δηλαδη πάλι τετραγωνικός χρόνος!

Πολλαπλασιασμός ακεραίων

Η τεχνική του Gauss

Παρατηρήστε ότι

$$(x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0$$

άρα

$$x_1y_0 + x_0y_1 = (x_1 + x_0)(y_1 + y_0) - x_1y_1 - x_0y_0 \tag{5}$$

που σημαίνει ότι αρκούν τρεις αναδρομικές κλήσεις αντί για τέσσερις.

Πράγματι

Πολλαπλασιασμός ακεραίων

Η τεχνική του Gauss - Αλγόριθμος

RecursiveMultiply(x, y)

$x = x_1 \cdot 10^{n/2} + x_0$

$y = y_1 \cdot 10^{n/2} + y_0$

$z =$ *RecursiveMultiply*$(x_1 + x_0, y_1 + y_0)$

$x_1 y_1 =$ *RecursiveMultiply*(x_1, y_1)

$x_0 y_0 =$ *RecursiveMultiply*(x_0, y_0)

Return $x_1 y_1 \cdot 10^n + (z - x_1 y_1 - x_0 y_0) \cdot 10^{n/2} + x_0 y_0$

Πολυπλοκότητα

Συνεπώς

$$T(n) \leq 3T(n/2) + cn$$

όπου $c =$ σταθερά.

ή

$$T(n) = O(n^{log3}) = O(n^{1.59}).$$

Πλησιέστερο ζεύγος σημείων

Το πρόβλημα

- Είσοδος: Ένα σύνολο $P = \{p_1, p_2, ..., p_n\}$ που περιέχει n σημεία του επιπέδου ($\mathbb{R}^2$).
- Συμβολισμός: Συμβολίζουμε με $d(p_i, p_j)$ την Ευκλείδια απόσταση. Επομένως, αν $p_i = (x_i, y_i)$ και $p_j = (x_j, y_j)$,

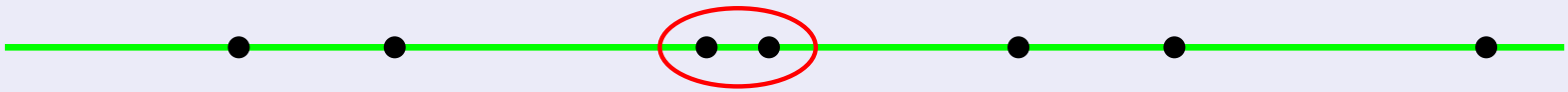
$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- Έξοδος: Ένα ζεύγος $p^*, q^* \in P$ διακεκριμένων σημείων που να ελαχιστοποιούν την απόσταση $d(p, q)$ όταν $p, q \in P$.

Πλησιέστερο ζεύγος σημείων

Αρχικές Παρατηρήσεις

- Υποθέτουμε (για διευκόλυνση) ότι όλα τα σημεία έχουν διαφορετικές συντεταγμένες x και διαφορετικές συντεταγμένες y.
- Το πρόβλημα μπορεί να επιλυθεί με εξαντλητική αναζήτηση (Brute force) σε χρόνο $O(n^2)$.
- Μπορούμε να το επιλύσουμε σε καλύτερο χρόνο;
- Η εκδοχή του προβλήματος στη μία διάσταση ($\mathbb{R}$).



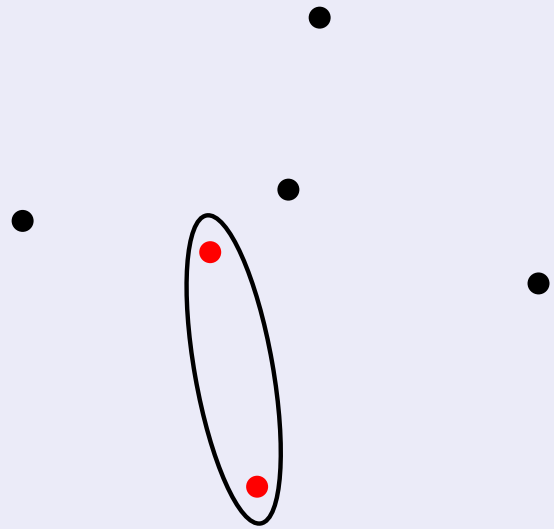
1. Ταξινόμηση των σημείων σε χρόνο $O(nlogn)$.
2. Εύρεση πλησιέστερου ζεύγους διαδοχικών σημείων σε χρόνο $O(n)$.

- Στόχος μας να βρούμε αλγόριθμο χρόνου $O(nlogn)$ για την εκδοχή του προβλήματος στο επίπεδο ($\mathbb{R}^2$)

Πλησιέστερο ζεύγος σημείων

Προσέγγιση Υψηλού Επιπέδου

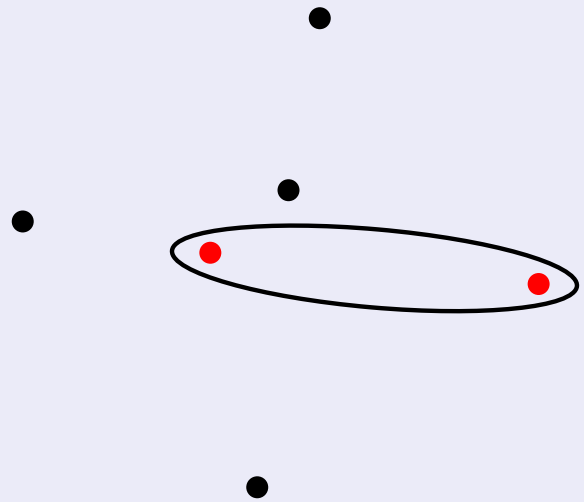
- Δημιουργία αντιγράφων των σημείων ταξινομημένων ως προς την συνιστώσα x (P_x) και ως προς τη συνιστώσα y (P_y) σε χρόνο $O(nlogn)$.
- Αυτό δεν είναι αρκετό.
- Εφαρμογή του Διαίρει και Βασίλευε.



Πλησιέστερο ζεύγος σημείων

Προσέγγιση Υψηλού Επιπέδου

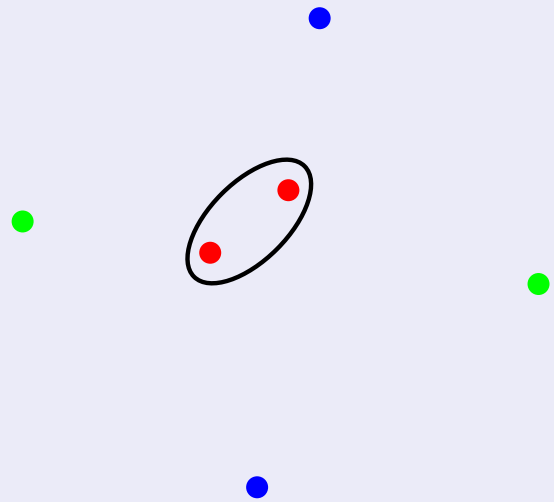
- Δημιουργία αντιγράφων των σημείων ταξινομημένων ως προς την συνιστώσα x (P_x) και ως προς τη συνιστώσα y (P_y) σε χρόνο $O(nlogn)$.
- Αυτό δεν είναι αρκετό.
- Εφαρμογή του Διαίρει και Βασίλευε.



Πλησιέστερο ζεύγος σημείων

Προσέγγιση Υψηλού Επιπέδου

- Δημιουργία αντιγράφων των σημείων ταξινομημένων ως προς την συνιστώσα x (P_x) και ως προς τη συνιστώσα y (P_y) σε χρόνο $O(nlogn)$.
- Αυτό δεν είναι αρκετό.
- Εφαρμογή του Διαίρει και Βασίλευε.



Πλησιέστερο ζεύγος σημείων

Διαίρει και Βασίλευε (Divide and Conquer)

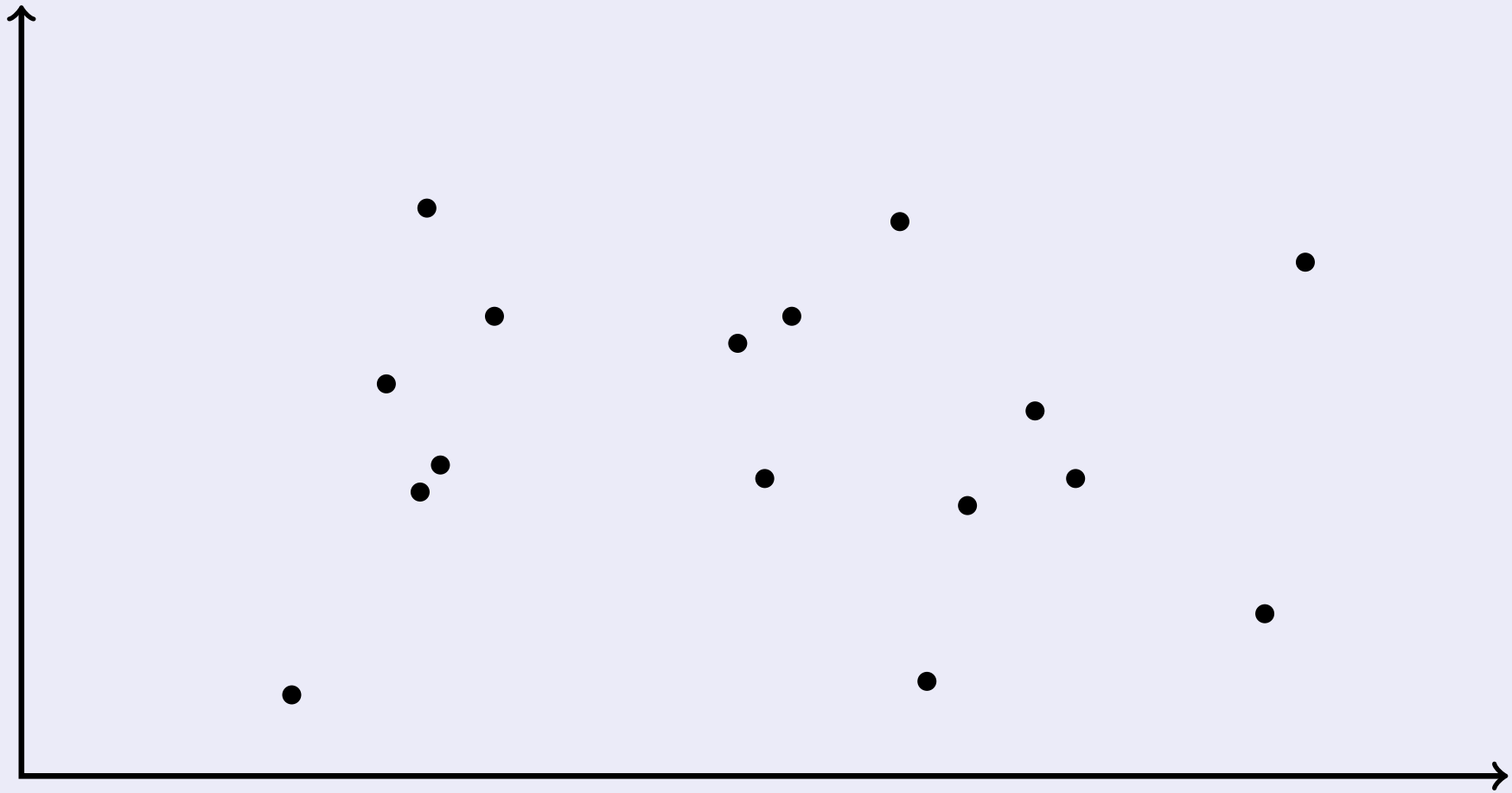
Διαίρεση: Διαίρεση του προβλήματος σε μικρότερα προβλήματα (υποπροβλήματα).

Επίλυση: Επίλυση των μικρότερων προβλημάτων αναδρομικά.

Συνδυασμός: Συνδυασμός των λύσεων για την εύρεση της αρχικής λύσης.

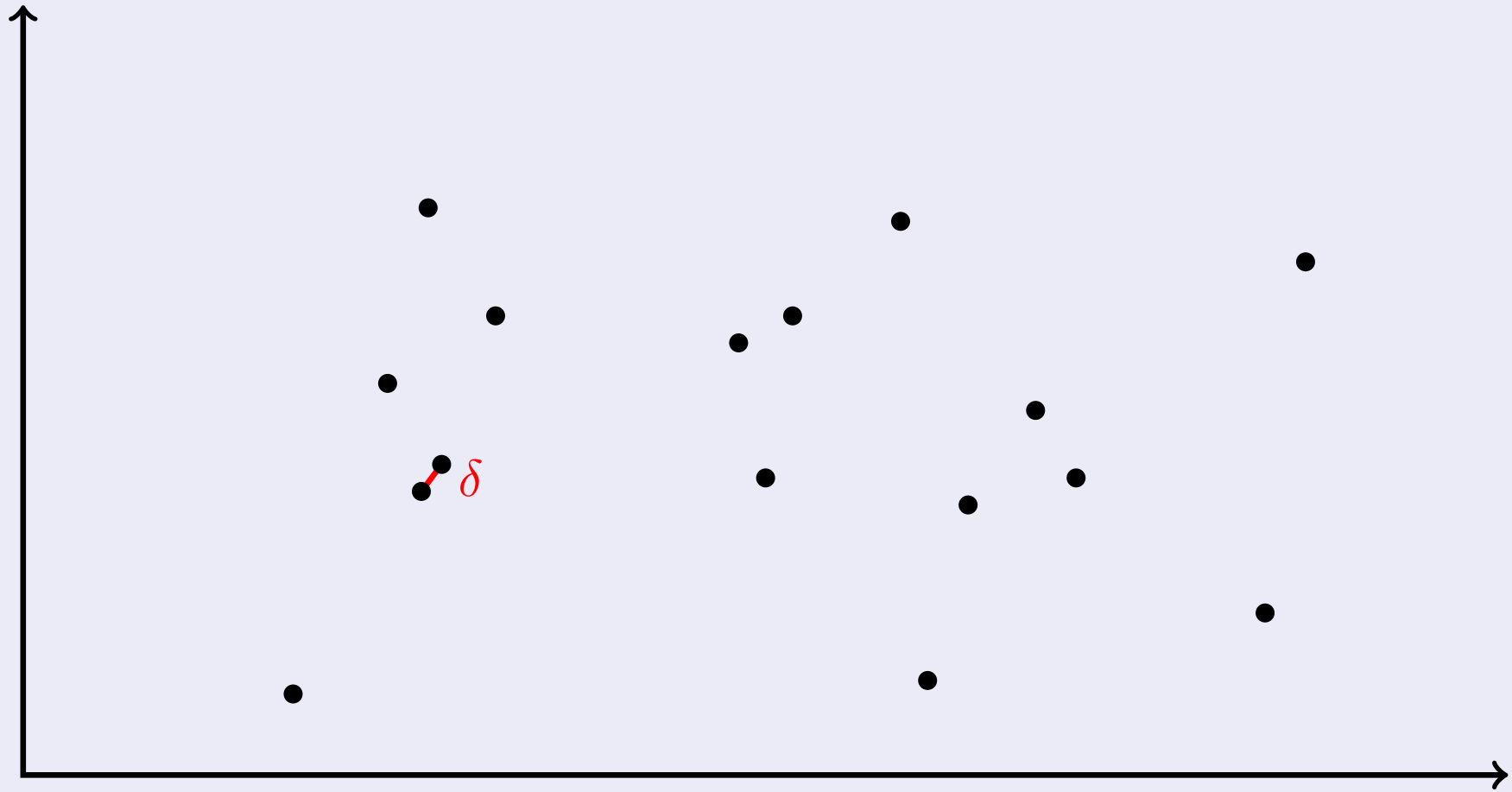
Πλησιέστερο ζεύγος σημείων

Σχηματικά



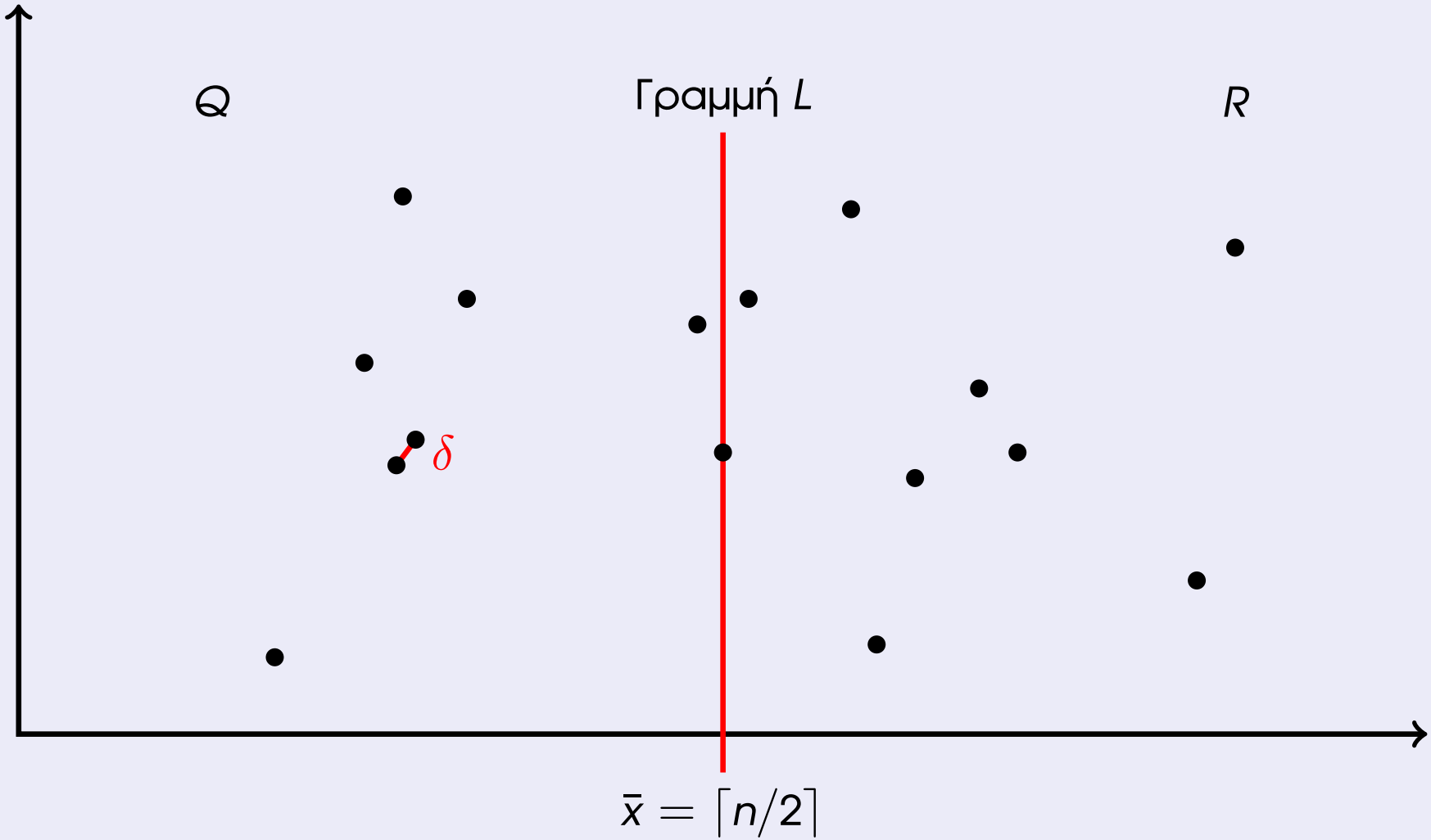
Πλησιέστερο ζεύγος σημείων

Σχηματικά



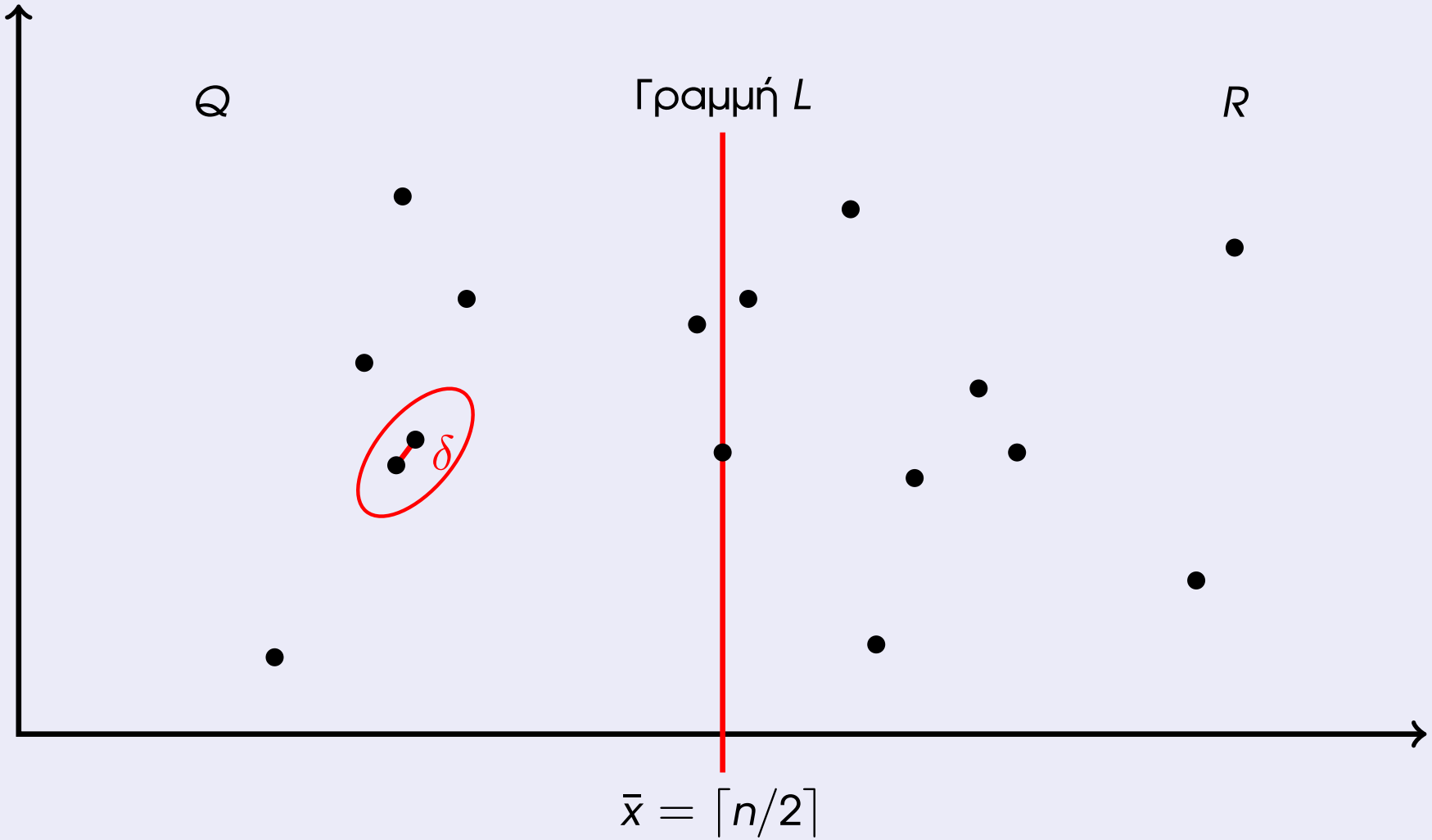
Πλησιέστερο ζεύγος σημείων

Σχηματικά



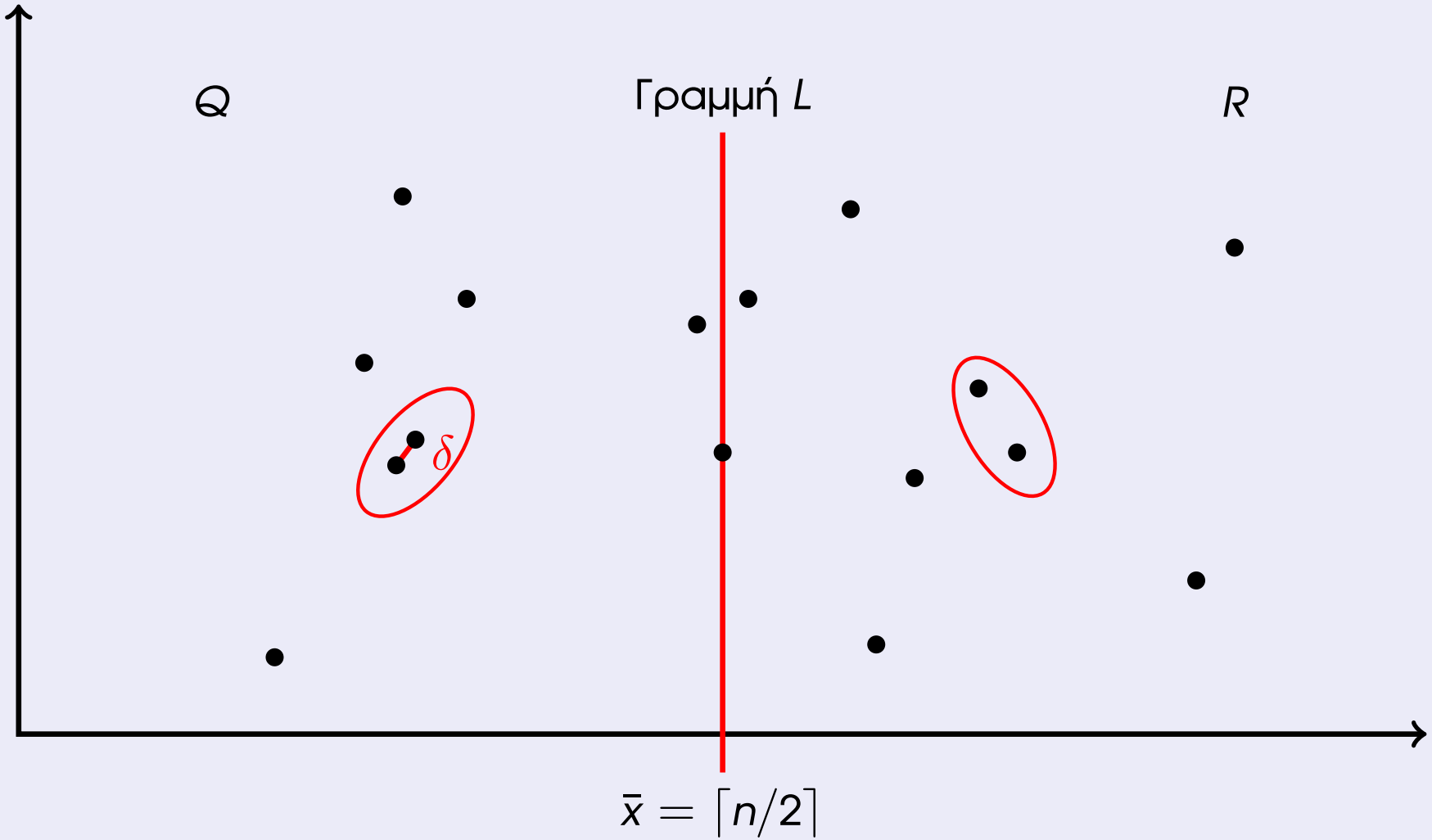
Πλησιέστερο ζεύγος σημείων

Σχηματικά



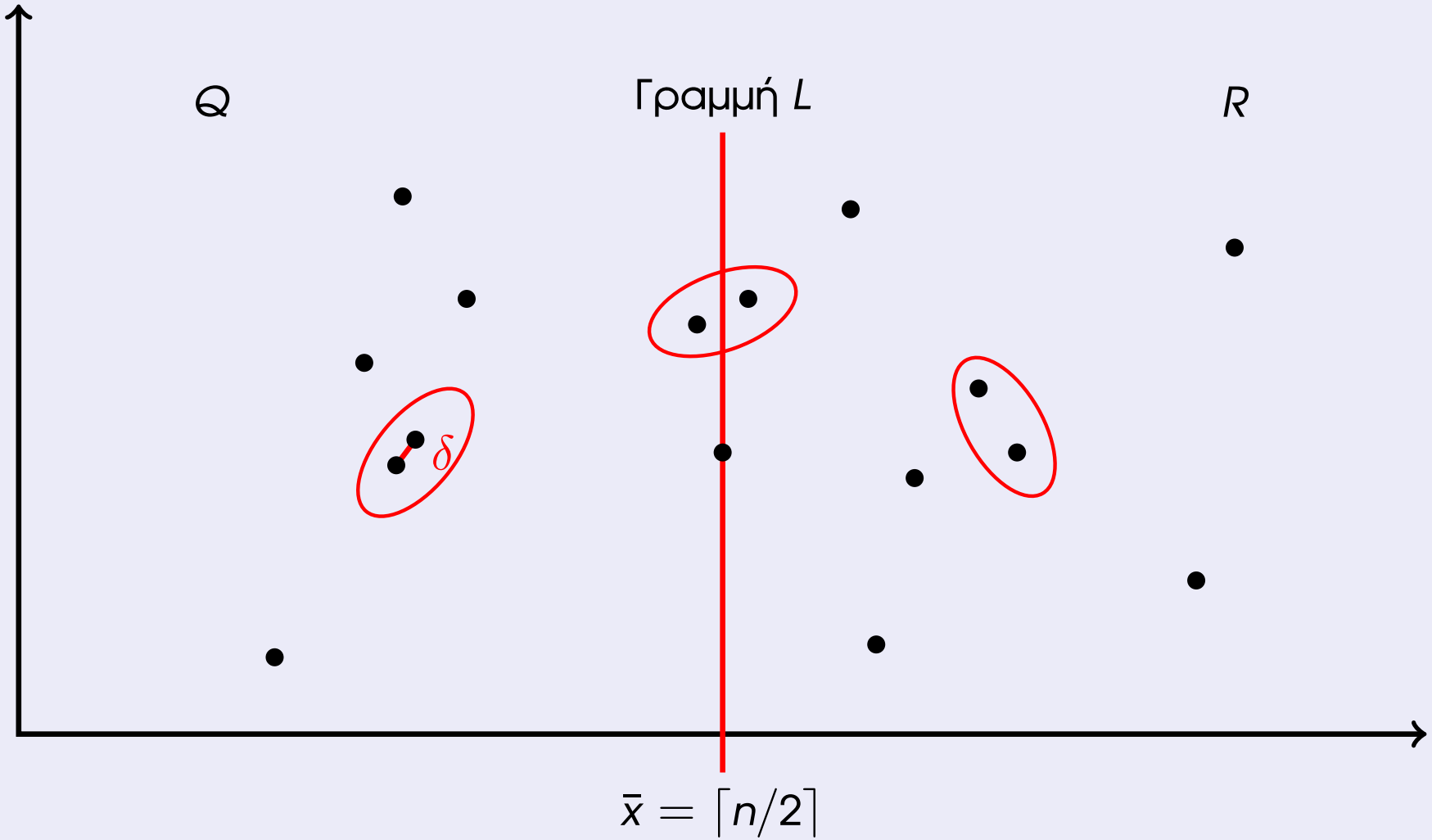
Πλησιέστερο ζεύγος σημείων

Σχηματικά



Πλησιέστερο ζεύγος σημείων

Σχηματικά



Πλησιέστερο ζεύγος σημείων

ClosestPair(P_x, P_y) 1η Εκδοχή

1. Ορίζουμε το Q ως το σύνολο των σημείων που βρίσκονται στις πρώτες $\lceil n/2 \rceil$ θέσεις της λίστας P_x (¨αριστερό μισό¨) και
2. το R ως το σύνολο των σημείων που βρίσκονται στις τελευταίες $\lfloor n/2 \rfloor$ θέσεις της λίστας P_x (¨δεξιό μισό¨).
3. Με μια μόνο διέλευση μέσω των P_x και P_y δημιουργούμε τις λίστες Q_x, Q_y, R_x, R_y ($O(n)$ χρόνο).
4. $(p_1, q_1) = \text{ClosestPair}(Q_x, Q_y)$
5. $(p_2, q_2) = \text{ClosestPair}(R_x, R_y)$
6. $(p_3, q_3) = \text{ClosestSplitPair}(P_x, P_y)$
7. Επίστρεψε το καλύτερο από τα $(p_1, q_1), (p_2, q_2), (p_3, q_3)$.

Πλησιέστερο ζεύγος σημείων

Ερώτημα

- Έστω $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$.
- Υπάρχουν σημεία $p \in Q$ και $q \in R$ για τα οποία $d(p, q) < \delta$;
- Αν δεν υπάρχουν τότε έχουμε ήδη βρει το πλησιέστερο ζευγάρι σε μία από τις αναδρομικές μας κλήσεις.
- Αν όμως υπάρχουν, τότε τα πλησιέστερα αυτά σημεία p και q σχηματίζουν το πλησιέστερο ζεύγος στο P.
- Στη συνέχεια θα ασχοληθούμε με τη δημιουργία του αλγορίθμου ClosestSplitPair

Πλησιέστερο ζεύγος σημείων

Πολυπλοκότητα

Έστω ότι μπορούμε να υλοποιήσουμε το ClosestSplitPair σε $O(n)$ χρόνο. Ποιός θα είναι ο συνολικός χρόνος εκτέλεσης του αλγορίθμου ClosestPair; Επιλέξτε το ελάχιστο άνω φράγμα.

- $O(n)$
- $O(n\log n)$
- $O(n(\log n)^2)$
- $O(n^2)$

Πλησιέστερο ζεύγος σημείων

Πολυπλοκότητα

Έστω ότι μπορούμε να υλοποιήσουμε το ClosestSplitPair σε $O(n)$ χρόνο. Ποιός θα είναι ο συνολικός χρόνος εκτέλεσης του αλγορίθμου Closest Pair; Επιλέξτε το ελάχιστο άνω φράγμα.

- $O(n)$
- $O(n \log n)$
- $O(n(\log n)^2)$
- $O(n^2)$

Πλησιέστερο ζεύγος σημείων

Βασική ιδέα

- Αρκεί να υπολογίσουμε το πλησιέστερο διαχωρισμένο ζεύγος στην περίπτωση $p \in Q$ και $q \in R$.
- Δηλαδή, όταν η απόσταση του διαχωρισμένου ζεύγους είναι μικρότερη από τα $d(p_1, q_1)$ και $d(p_2, q_2)$.
- Τα $d(p_1, q_1)$ και $d(p_2, q_2)$ είναι τα αποτελέσματα της πρώτης και της δεύτερης αναδρομικής κλήσης αντίστοιχα.

Πλησιέστερο ζεύγος σημείων

ClosestPair(P_x, P_y) 2η Εκδοχή

1. Ορίζουμε το Q ως το σύνολο των σημείων στις πρώτες $\lceil n/2 \rceil$ θέσεις της λίστας P_x (¨αριστερό μισό¨) και
2. το R ως το σύνολο των σημείων στις τελευταίες $\lfloor n/2 \rfloor$ θέσεις της λίστας P_x (¨δεξιό μισό¨).
3. Με μια μόνο διέλευση μέσω των P_x και P_y δημιουργούμε τις λίστες Q_x, Q_y, R_x, R_y (σε $O(n)$ χρόνο).
4. $(p_1, q_1) = \text{ClosestPair}(Q_x, Q_y)$
5. $(p_2, q_2) = \text{ClosestPair}(R_x, R_y)$
6. Έστω $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
7. $(p_3, q_3) = \text{ClosestSplitPair}(P_x, P_y, \delta)$
8. Επίστρεψε το καλύτερο από τα (p_1, q_1), (p_2, q_2), (p_3, q_3).

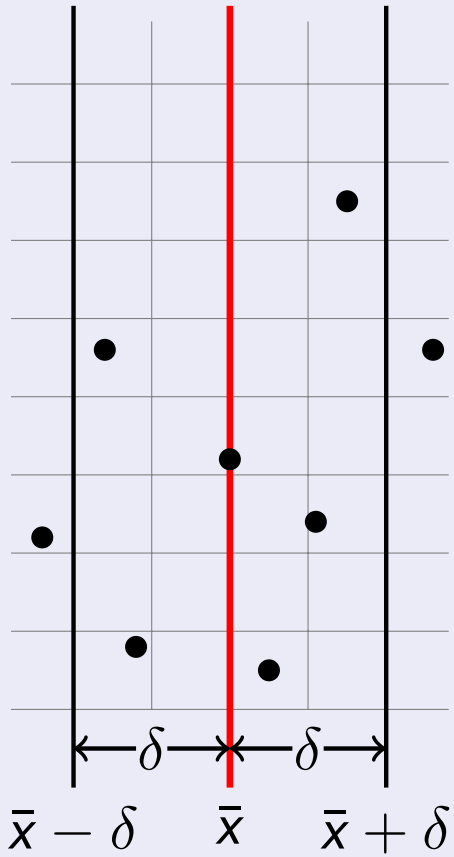
Πλησιέστερο ζεύγος σημείων

ClosestSplitPair(P_x, P_y, δ)

- Έστω $\bar{x}$ η μεγαλύτερη συντεταγμένη x στο αριστερό μέρος του P.
- Έστω S_y τα σημεία του P με συντεταγμένη x στο διάστημα $[\bar{x} - \delta, \bar{x} + \delta]$, ταξινομημένα ως προς τη συντεταγμένη y (χρόνος $O(n)$).
- Τα σημεία που πρέπει να εξεταστούν από την ClosestSplitPair είναι μόνον αυτά που βρίσκονται στην S_y

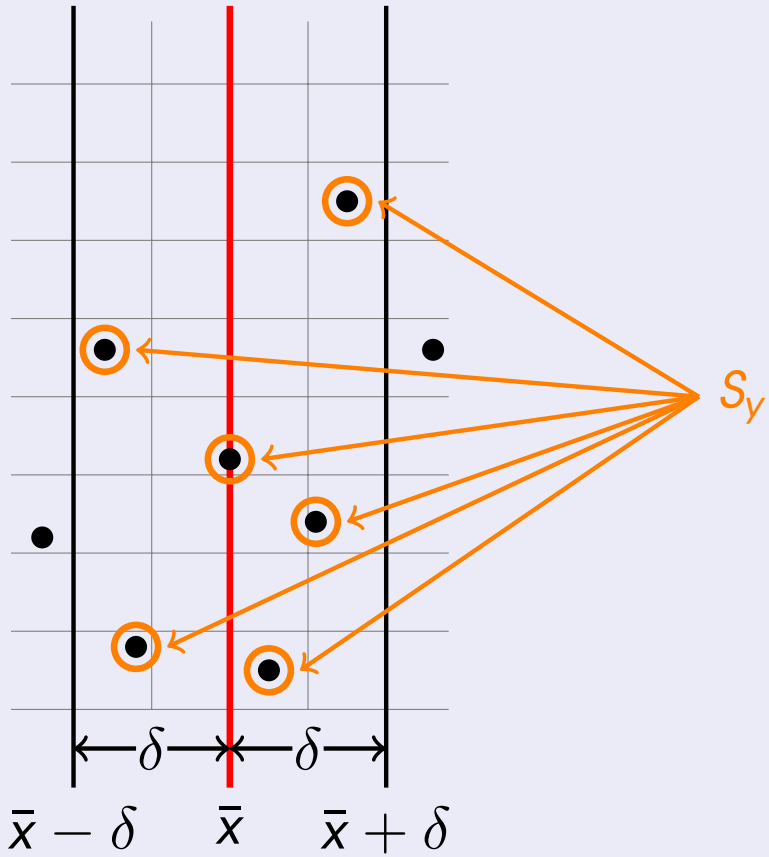
Πλησιέστερο ζεύγος σημείων

ClosestSplitPair(P_x, P_y, δ)



Πλησιέστερο ζεύγος σημείων

ClosestSplitPair(P_x, P_y, δ)



Αλγόριθμος

ClosestSplitPair(P_x, P_y, δ)

1. best $= \delta$, bestpair $=$ NULL
2. για $i = 1$ έως $|S_y| - 1$ $\quad O(n)$
3. για $j = 1$ έως $min\{7, |S_y| - 1\}$ $\quad O(1)$
4. $p \leftarrow i$-οστό σημείο του S_y $\quad (O(n)$, το S_y μπορεί να είναι $= P)$
5. $q \leftarrow (i + j)$-οστό σημείο του S_y
6. αν $d(p, q) <$ best τότε
7. bestpair $= (p, q)$, best $= d(p, q)$
8. Επίστρεψε bestpair

Πλησιέστερο ζεύγος σημείων

Ισχυρισμός ορθότητας

- **Ισχυρισμός:** Έστω ότι $\exists(p, q)$με $p \in Q, q \in R : d(p, q) < \delta$, όπου

$$\delta = min\{d(p_1, q_1), d(p_2, q_2)\}. \tag{6}$$

Τότε:

Α. $p, q \in S_y$.

Β. Τα p και q απέχουν μεταξύ τους το πολύ 7 θέσεις στο S_y.

- Αν ο ισχυρισμός είναι ορθός τότε ισχύουν τα ακόλουθα δύο πορίσματα.

Πόρισμα 1

Αν το πλησιέστερο ζεύγος σημείων είναι ένα διαχωρισμένο ζεύγος τότε ο ClosestSplitPair το βρίσκει.

Πόρισμα 2

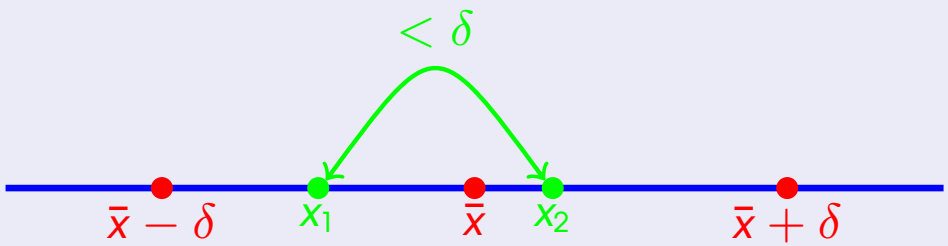
Ο ClosestPair είναι ορθός και τρέχει σε χρόνο $O(nlogn)$.

Πλησιέστερο ζεύγος σημείων

Απόδειξη του ισχυρισμού ορθότητας (Α)

- Έστω $p = (x_1, y_1) \in Q$, $q = (x_2, y_2) \in R$ και $d(p, q) < \delta$.
- Αφού $d(p, q) < \delta$ έχουμε

$$|x_1 - x_2| < \delta \quad \text{και} \quad |y_1 - y_2| < \delta. \tag{7}$$



- Αλλά

$$x_1 \leq \bar{x} \leq x_2. \tag{8}$$

Πλησιέστερο ζεύγος σημείων

Απόδειξη του ισχυρισμού ορθότητας (A)

- Έτσι από (7) (8) έχουμε

$$\bar{x} - x_1 \leq x_2 - x_1 < \delta, \tag{9}$$

$$x_2 - \bar{x} \leq x_2 - x_1 < \delta \tag{10}$$

- Από (9) και (10) έχουμε

$$\bar{x} - \delta < x_1 \tag{11}$$

και

$$x_2 < \bar{x} + \delta \tag{12}$$

οπότε οι (11) και (12) δηλώνουν ότι

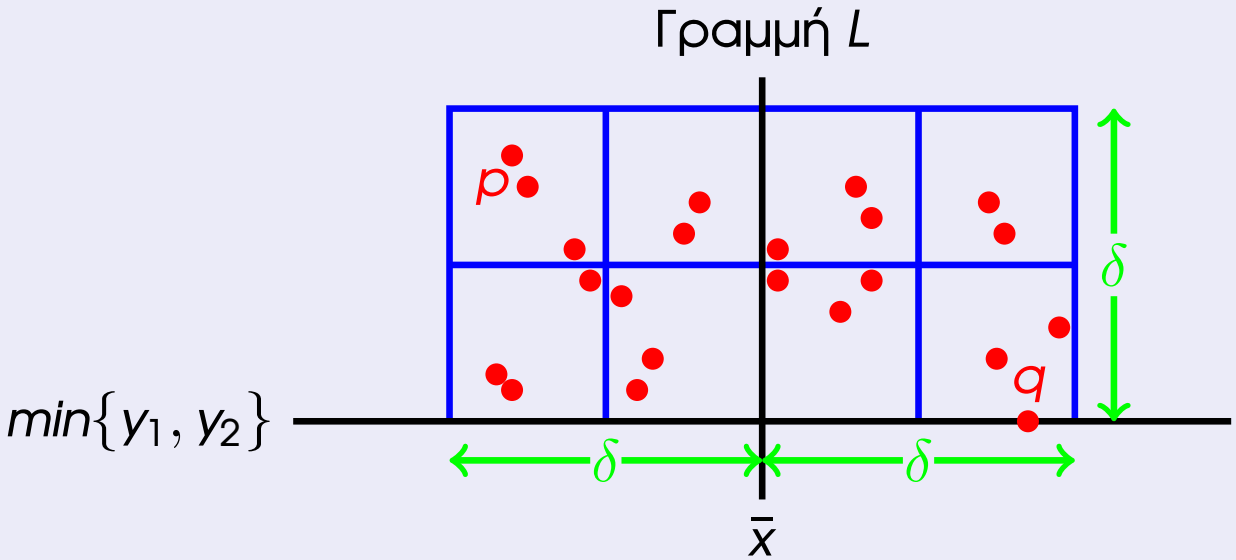
$$p, q \in S_y$$

.

Πλησιέστερο ζεύγος σημείων

Απόδειξη του ισχυρισμού ορθότητας (Β)

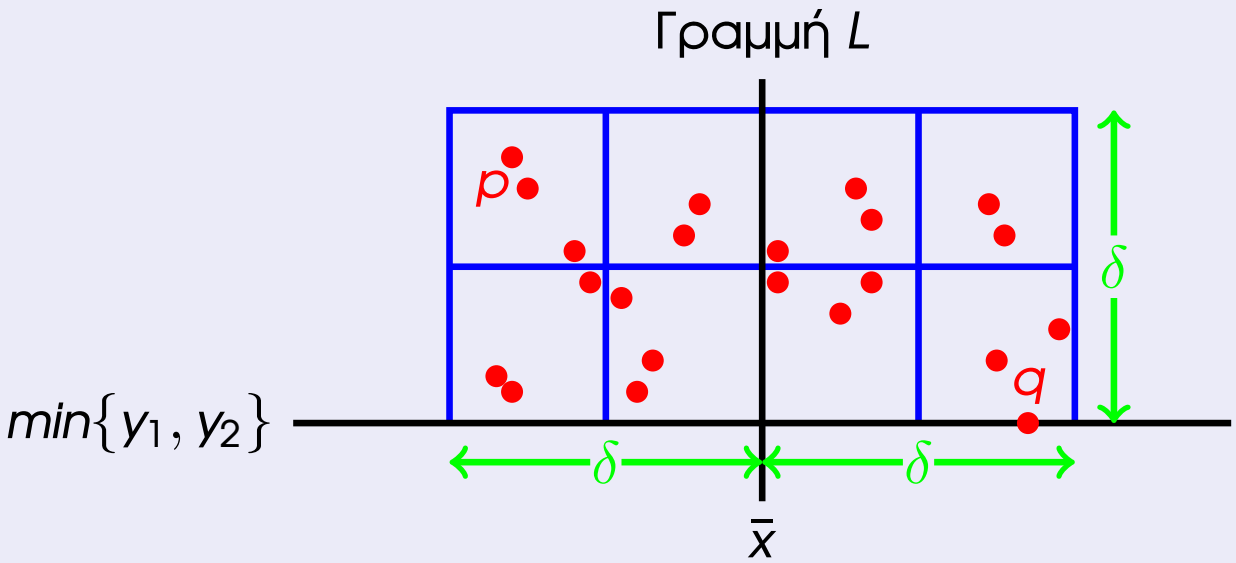
- Έστω $p = (x_1, y_1)$ και $q = (x_2, y_2)$ διαχωρισμένο ζεύγος με $d(p, q) < \delta$. Θα δείξουμε ότι τα p, q απέχουν μεταξύ τους το πολύ 7 θέσεις στο S_y.
- Σχεδιάζουμε 8 τετράγωνα πλευράς $\frac{\delta}{2}$ με κέντρο $\bar{x}$ και βάση $min\{y_1, y_2\}$.



Πλησιέστερο ζεύγος σημείων

Απόδειξη του ισχυρισμού ορθότητας (Β)

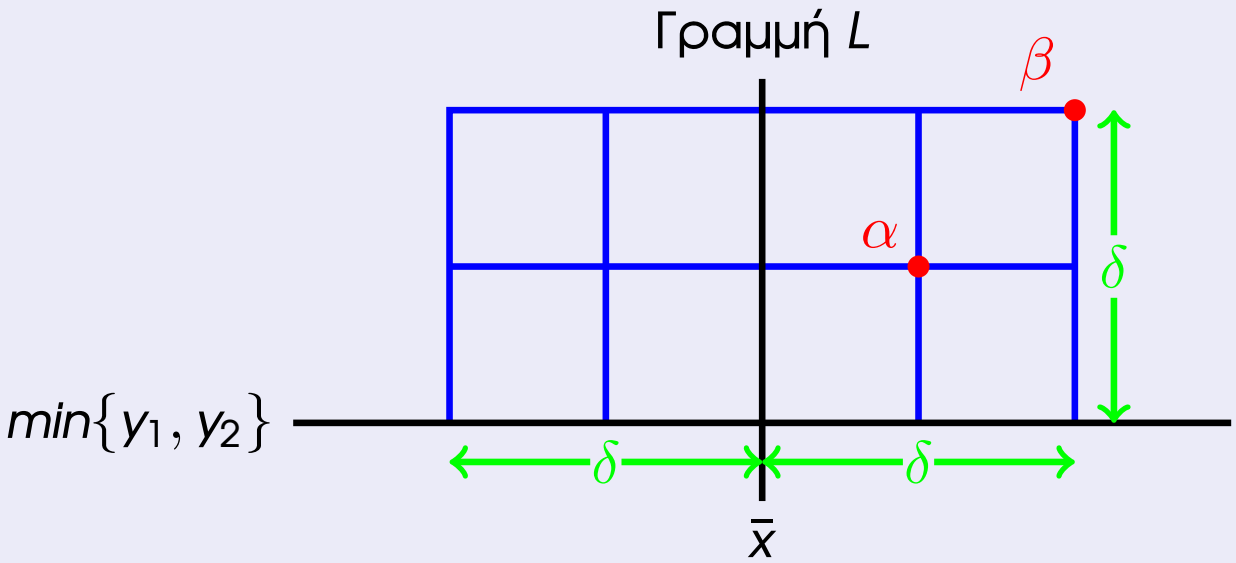
- **Λήμμα 1:** Όλα τα σημεία του S_y με συντεταγμένη y μεταξύ των συντεταγμένων y των p και q, συμπεριλαμβανομένων, βρίσκονται σε ένα από αυτά τα τετράγωνα.
- **Απόδειξη:** Αρχικά, έχουμε υποθέσει ότι οι συντεταγμένες y των p και q διαφέρουν το πολύ κατά δ. Δεύτερον, από τον ορισμό του S_y, όλα έχουν συντεταγμένες x μεταξύ του $\bar{x} - \delta$ και $\bar{x} + \delta$.



Πλησιέστερο ζεύγος σημείων

Απόδειξη του ισχυρισμού ορθότητας (Β)

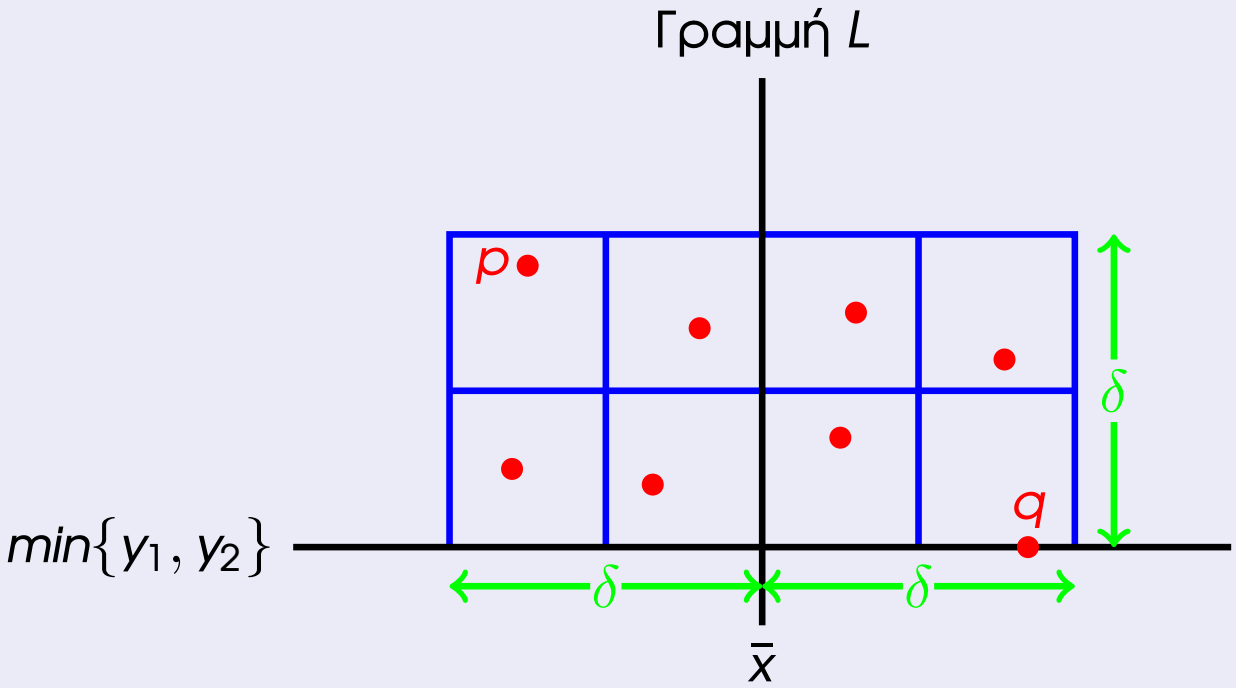
- **Λήμμα 2:** Το πολύ 1 σημείο του P βρίσκεται σε κάθε τετράγωνο.
- **Απόδειξη:** Με εις άτοπον απαγωγή. Έστω δύο σημεία α και β βρίσκονται στο ίδιο τετράγωνο. Τότε:
  (i) Τα α, β βρίσκονται και τα δύο στο Q ή και τα δύο στο R.
  (ii) $d(\alpha, \beta) \leq \frac{\delta}{2}\sqrt{2} < \delta$
  Όμως τα (i),(ii) έρχονται σε αντίθεση με τον ορισμό του δ (η μικρότερη απόσταση μεταξύ ζεύγους σημείων στο Q ή στο R). Άτοπο.



Πλησιέστερο ζεύγος σημείων

Τελική ανακεφαλαίωση της απόδειξης

- Τα λήμματα 1 και 2 εγγυώνται ότι στο παρακάτω σχήμα θα υπάρχουν το πολύ 8 σημεία (συμπεριλαμβανομένων των p και q).
- Άρα η θέση των p και q στο S_y διαφέρει το πολύ κατά 7 θέσεις.



Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

1. Closest-Pair(P)
2. Δημιούργησε τα P_x και P_y
3. (p, q) =Closest-Pair-Rec(P_x, P_y)
4.
5. Closest-Pair-Rec(P_x, P_y)
6. αν $|P| \leq 3$ τότε
7. Βρες το πλησιέστερο ζεύγος με μέτρηση όλων των αποστάσεων των ζευγαριών
8.
9. Δημιούργησε τα Q_x, Q_y, R_x, R_y
10. (p_1, q_1) =Closest-Pair-Rec(Q_x, Q_y)
11. (p_2, q_2) =Closest-Pair-Rec(R_x, R_y)
12.
13. $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
14. $\bar{x}$ = μέγιστη συντεταγμένη x ενός σημείου του συνόλου Q

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

1. Closest-Pair(P)
2. Δημιούργησε τα P_x και P_y (χρόνος $O(n\log n)$)
3. (p, q) =Closest-Pair-Rec(P_x, P_y)
4.
5. Closest-Pair-Rec(P_x, P_y)
6. αν $|P| \leq 3$ τότε
7. Βρες το πλησιέστερο ζεύγος με μέτρηση όλων των αποστάσεων των ζευγαριών
8.
9. Δημιούργησε τα Q_x, Q_y, R_x, R_y
10. (p_1, q_1) =Closest-Pair-Rec(Q_x, Q_y)
11. (p_2, q_2) =Closest-Pair-Rec(R_x, R_y)
12.
13. $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
14. $\bar{x}$ = μέγιστη συντεταγμένη x ενός σημείου του συνόλου Q

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

1. Closest-Pair(P)
2. Δημιούργησε τα P_x και P_y (χρόνος $O(n\log n)$)
3. (p, q) =Closest-Pair-Rec(P_x, P_y)
4.
5. Closest-Pair-Rec(P_x, P_y)
6. αν $|P| \leq 3$ τότε
7. Βρες το πλησιέστερο ζεύγος με μέτρηση όλων των αποστάσεων των ζευγαριών (χρόνος $O(1)$)
8.
9. Δημιούργησε τα Q_x, Q_y, R_x, R_y
10. (p_1, q_1) =Closest-Pair-Rec(Q_x, Q_y)
11. (p_2, q_2) =Closest-Pair-Rec(R_x, R_y)
12.
13. $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
14. $\bar{x}$ = μέγιστη συντεταγμένη x ενός σημείου του συνόλου Q

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

1. Closest-Pair(P)
2. Δημιούργησε τα P_x και P_y (χρόνος $O(n\log n)$)
3. (p, q) =Closest-Pair-Rec(P_x, P_y)
4.
5. Closest-Pair-Rec(P_x, P_y)
6. αν $|P| \leq 3$ τότε
7. Βρες το πλησιέστερο ζεύγος με μέτρηση όλων των αποστάσεων των ζευγαριών (χρόνος $O(1)$)
8.
9. Δημιούργησε τα Q_x, Q_y, R_x, R_y (χρόνος $O(n)$)
10. (p_1, q_1) =Closest-Pair-Rec(Q_x, Q_y)
11. (p_2, q_2) =Closest-Pair-Rec(R_x, R_y)
12.
13. $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
14. $\bar{x}$ = μέγιστη συντεταγμένη x ενός σημείου του συνόλου Q

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

1. Closest-Pair(P)
2. Δημιούργησε τα P_x και P_y (χρόνος $O(n\log n)$)
3. (p, q) =Closest-Pair-Rec(P_x, P_y)
4.
5. Closest-Pair-Rec(P_x, P_y)
6. αν $|P| \leq 3$ τότε
7. Βρες το πλησιέστερο ζεύγος με μέτρηση όλων των αποστάσεων των ζευγαριών (χρόνος $O(1)$)
8.
9. Δημιούργησε τα Q_x, Q_y, R_x, R_y (χρόνος $O(n)$)
10. (p_1, q_1) =Closest-Pair-Rec(Q_x, Q_y)
11. (p_2, q_2) =Closest-Pair-Rec(R_x, R_y)
12.
13. $\delta = min\{d(p_1, q_1), d(p_2, q_2)\}$
14. $\bar{x}$ = μέγιστη συντεταγμένη x ενός σημείου του συνόλου Q (χρ. $O(n)$)

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

15. $L = \{(x, y) : x = \bar{x}\}$
16. $S =$ σημεία του P με απόσταση μέχρι δ από τη γραμμή L
17. Δημιούργησε το S_y
18. Για κάθε σημείο $q \in S_y$, υπολόγισε την απόσταση του q από τα επόμενα 7 σημεία του S_y ClosestSplitPair (P_x, P_y, δ)
19. Έστω p_3, q_3 το ζευγάρι που επιστρέφει η ClosestSplitPair που είναι το ελάχιστο αυτών των αποστάσεων
20.
21. Αν $d(p_3, q_3) < \delta$ τότε
22. Επίστρεψε (p_3, q_3)
23. Αλλιώς αν $d(p_1, q_1) < d(p_2, q_2)$ τότε
24. Επίστρεψε (p_1, q_1)
25. Αλλιώς
26. Επίστρεψε (p_2, q_2)

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

15. $L = \{(x, y) : x = \bar{x}\}$
16. $S =$ σημεία του P με απόσταση μέχρι δ από τη γραμμή L
17. Δημιούργησε το S_y (χρόνος $O(n)$)
18. Για κάθε σημείο $s \in S_y$, υπολόγισε την απόσταση του s από τα επόμενα 7 σημεία του S_y
19. Έστω p_3, q_3 ένα ζευγάρι που επιτυγχάνει το ελάχιστο αυτών των αποστάσεων
20.
21. Αν $d(p_3, q_3) < \delta$ τότε
22. Επίστρεψε (p_3, q_3)
23. Αλλιώς αν $d(p_1, q_1) < d(p_2, q_2)$ τότε
24. Επίστρεψε (p_1, q_1)
25. Αλλιώς
26. Επίστρεψε (p_2, q_2)

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

15. $L = \{(x, y) : x = \bar{x}\}$
16. $S =$ σημεία του P με απόσταση μέχρι δ από τη γραμμή L
17. Δημιούργησε το S_y (χρόνος $O(n)$)
18. Για κάθε σημείο $s \in S_y$, υπολόγισε την απόσταση του s από τα επόμενα 7 σημεία του S_y (χρόνος $O(n)$, το S_y μπορεί να είναι $= P$)
19. Έστω p_3, q_3 ένα ζευγάρι που επιτυγχάνει το ελάχιστο αυτών των αποστάσεων
20.
21. Αν $d(p_3, q_3) < \delta$ τότε
22. Επίστρεψε (p_3, q_3)
23. Αλλιώς αν $d(p_1, q_1) < d(p_2, q_2)$ τότε
24. Επίστρεψε (p_1, q_1)
25. Αλλιώς
26. Επίστρεψε (p_2, q_2)

Πλησιέστερο ζεύγος σημείων

Αλγόριθμος

15. $L = \{(x, y) : x = \bar{x}\}$
16. $S =$ σημεία του P με απόσταση μέχρι δ από τη γραμμή L
17. Δημιούργησε το S_y (χρόνος $O(n)$)
18. Για κάθε σημείο $s \in S_y$, υπολόγισε την απόσταση του s από τα επόμενα 7 σημεία του S_y (χρόνος $O(n)$, το S_y μπορεί να είναι $= P$)
19. Έστω p_3, q_3 ένα ζευγάρι που επιτυγχάνει το ελάχιστο αυτών των αποστάσεων (χρόνος $O(1)$)
20.
21. Αν $d(p_3, q_3) < \delta$ τότε
22. Επίστρεψε (p_3, q_3)
23. Αλλιώς αν $d(p_1, q_1) < d(p_2, q_2)$ τότε
24. Επίστρεψε (p_1, q_1)
25. Αλλιώς
26. Επίστρεψε (p_2, q_2)

Αναδρομή

Η ακολουθία Fibonacci

$$F(n) = \begin{cases} 1 & \text{if } n = 0,\ n = 1, \\ F(n-1) + F(n-2) & \text{if } n > 1. \end{cases}$$

```
    FIBONACCI (n)
1.  f0 = 1
2.  f1 = 1
3.  for i = 3 to n do
4.          temp = f0 + f1
5.          f0 = f1
6.          f1 = temp
7.  end for   O(n)
8.  return f
```

Σχήμα: Επαναληπτικός Αλγόριθμος εύρεσης του n-οστού αριθμού Fibonacci.

Η πολυπλοκότητα του επαναληπτικού αλγορίθμου είναι $O(n)$.

Απόδειξη Ορθότητας Αναδρομικών αλγορίθμων

Πρόβλημα

Να δοθεί αναδρομικός αλγόριθμος για την εύρεση του n-οστού αριθμού Fibonacci. Στην συνέχεια να αποδειχθεί η ορθότητά του.

Λύση

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2$$

$$F_0 = F_1 = 1$$

Απόδειξη Ορθότητας Αναδρομικών αλγορίθμων

Αλγόριθμος

Fib(n)
1 **αν** $n = 0$ ή 1
2 **τότε**
3 Fib $\leftarrow 1$
4 **άλλως**
5 Fib $\leftarrow$ Fib$(n-1)+$ Fib$(n-2)$
6 **επιστροφή** Fib

Απόδειξη Ορθότητας Αναδρομικών αλγορίθμων

Ορθότητα

Με επαγωγή.

Βάση.

Για $n = 0$ έχουμε

$$\text{Fib}(0) = 1 = F_0 \quad \text{ισχύει.}$$

Επαγωγικό βήμα.

Υποθέτουμε ότι

$$\text{Fib}(k) = F_k \quad \text{για } k < n$$

και θα δείξουμε ότι

$$\text{Fib}(n) = F_n.$$

Απόδειξη.

$$\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2) = F_{n-1} + F_{n-2} = F_n.$$

Αναδρομή

Πολυπλοκότητα του Αναδρομικού Αλγορίθμου

$$
\begin{aligned}
T(n) &= 1 + T(n-1) + T(n-2) \\
&= 1 + 1 + T(n-2) + T(n-3) + T(n-2) \quad (n \geq 3) \\
T(n) &\geq 2T(n-2) \\
&\geq 2^2 T(n-4) \\
&\cdots \\
&\geq 2^{\frac{n}{2}} T(n - 2\frac{n}{2})
\end{aligned}
$$

Αναδρομή

Η ακολουθία Fibonacci

Άρα θα έχουμε

$$T(n) \geq 2^{\frac{n}{2}} T(0)$$

εάν n άρτιος και

$$T(n) \geq 2^{\frac{n-1}{2}} T(1)$$

εάν n περιττός, οπότε τελικά

$$T(n) \geq 2^{\frac{n}{2}},$$

δεδομένου ότι $T(0) = T(1) = 1$.
Ο αναδρομικός αλγόριθμος έχει εκθετική πολυπλοκότητα ενώ ο επαναληπτικός πολυωνυμική $\Theta(n)$.

Ύψωση σε δύναμη

Το πρόβλημα

Για δεδομένο αριθμό x και ακέραιο $n \geq 0$ να υπολογιστεί το x^n

Απλός αλγόριθμος

$$x \cdot x \cdot x \cdots x = x^n, \; T(n) = \Theta(n)$$

Διαίρει και βασίλευε αλγόριθμος

$$x^n = \begin{cases} x^{n/2} \cdot x^{n/2} & \text{, για } n \text{ άρτιο} \\ x^{\frac{n-1}{2}} \cdot x^{\frac{n-1}{2}} \cdot x & \text{, για } n \text{ περιττό} \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + \Theta(1) \\ &= \Theta(\log n) \end{aligned}$$

Αναδρομή

Αριθμοί Fibonacci

$$F_n = \begin{cases} 0 & , \text{για } n = 0 \\ 1 & , \text{για } n = 1 \\ F_{n-1} + F_{n-2} & , \text{για } n \geq 2 \end{cases}$$

Απλός αναδρομικός αλγόριθμος

$$T(0) = 0, \quad T(1) = 1$$

$$T(n) = T(n-1) + T(n-2) + 1 \tag{13}$$

$$T(n) = \Omega(\varphi^n), \quad \varphi = \frac{1+\sqrt{5}}{2} \text{ εκθετικός χρόνος!}$$

Επαναληπτικός Αλγόριθμος

$$T(n) = \Theta(n)$$

Αναδρομή

Αναδρομικός τετραγωνισμός

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \Rightarrow T(n) = \Theta(\log n) \text{ λόγω ύψωσης σε δύναμη}$$

Απόδειξη με επαγωγή στο n

Βάση $n = 1$

$$\begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1 \text{ ισχύει}$$

ό Βήμα

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Αναδρομή

Πολλαπλασιασμός πινάκων

Δίνονται δυο $n \times n$ πινάκες $A = (a_{i,j})$ και $B = (b_{i,j})$.

Να υπολογιστεί το γινόμενο

$$C = AB$$

όπου

$$c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j} \tag{14}$$

Χρόνος εκτέλεσης

Από την (14) έχουμε

$$T(n) = \Theta(n^3). \tag{15}$$

Αναδρομή

Το 1969 ο Strassen πρότεινε έναν αλγόριθμο διαίρει και βασίλευε με

$$T(n) = \Theta(n^{\log_2 7}). \tag{16}$$

Διαμερίζουμε τους πίνακες A και B σε τέσσερις $(n/2) \times (n/2)$ υποπίνακες ως εξής

$$AB = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \left[\begin{array}{c|c} A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} \\ \hline A_{10}B_{00} + A_{11}B_{10} & A_{10}B_{01} + A_{11}B_{11} \end{array}\right] \tag{17}$$

Για τον υπολογισμό του AB απαιτούνται, λόγω της (20), οκτώ πολλαπλασιασμοί $(n/2) \times (n/2)$ πινάκων, δηλαδή

$$T(n) = 8T\left(\frac{n}{2}\right) + ?? \tag{18}$$

Αναδρομή

Το 1969 ο Strassen πρότεινε έναν αλγόριθμο διαίρει και βασίλευε με

$$T(n) = \Theta(n^{\log_2 7}). \tag{19}$$

Διαμερίζουμε τους πίνακες A και B σε τέσσερις $(n/2) \times (n/2)$ υποπίνακες ως εξής

$$AB = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \left[\begin{array}{c|c} A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} \\ \hline A_{10}B_{00} + A_{11}B_{10} & A_{10}B_{01} + A_{11}B_{11} \end{array}\right] \tag{20}$$

Για τον υπολογισμό του AB απαιτούνται, λόγω της (20), οκτώ πολλαπλασιασμοί $(n/2) \times (n/2)$ πινάκων, δηλαδή

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^3). \tag{21}$$

Αναδρομή

Ο Strassen πρότεινε τον υπολογισμό του AB με τους ακόλουθους εφτά πολλαπλασιασμούς πινάκων.

$$
\begin{aligned}
M_1 &= (A_{00} + A_{11})(B_{00} + B_{11}) \\
M_2 &= (A_{10} + A_{11})B_{00} \\
M_3 &= A_{00}(B_{01} - B_{11}) \\
M_4 &= A_{11}(B_{10} - B_{00}) \\
M_5 &= (A_{00} + A_{01})B_{11} \\
M_6 &= (A_{10} - A_{00})(B_{00} + B_{01}) \\
M_7 &= (A_{01} - A_{11})(B_{10} + B_{11})
\end{aligned}
\tag{22}
$$

Αναδρομή

Τότε

$$AB = \left[\begin{array}{c|c} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ \hline M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{array}\right] \tag{23}$$

Από τις (22) και (23) έχουμε

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2), \quad n > 1 \tag{24}$$
$$T(1) = 1$$

Επιλύοντας την (24) με τη μέθοδό της αντικατάστασης έχουμε

$$T(n) = \Theta(n^{\log_2 7}). \tag{25}$$

Επειδή $n^{\log_2 7} = 2.81$ συνεπάγεται ότι ο αλγόριθμος του Strassen είναι σημαντικά καλύτερος από τον κλασικό αλγόριθμό πολυπλοκότητας $\Theta(n^3)$.

Αναδρομή

Οι τύποι (22) και (23) του Strassen έχουν 18 προσθέσεις (ή αφαιρέσεις). Ο Winograd ανακάλυψε το ακόλουθο σύνολο τύπων που απαιτεί, για τον υπολογισμό 2×2 πινάκων, μόνο 15 προσθέσεις (ή αφαιρέσεις) διατηρώντας το ίδιο πλήθος (επτά) πολλαπλασιασμών

$$
\begin{aligned}
M_1 &= (A_{10} + A_{11} - A_{00})(B_{11} - B_{01} + B_{00}) \\
M_2 &= A_{00}B_{00} \\
M_3 &= A_{01}B_{10} \\
M_4 &= (A_{00} - A_{10})(B_{11} - B_{01}) \\
M_5 &= (A_{10} + A_{11})(B_{01} - B_{00}) \\
M_6 &= (A_{01} - A_{10} + A_{00} - A_{11})B_{11} \\
M_7 &= A_{11}(B_{00} + B_{11} - B_{01} - B_{10})
\end{aligned}
\tag{26}
$$

Αναδρομή

Τότε

$$AB = \left[\begin{array}{c|c} M_2 + M_3 & M_1 + M_2 + M_5 + M_6 \\ \hline M_1 + M_2 + M_4 - M_7 & M_1 + M_2 + M_4 + M_5 \end{array}\right] \quad (27)$$

- Αν και η (27) απαιτεί 24 προσθέσεις (ή αφαιρέσεις) μπορεί να αποδειχθεί ότι χρειάζεται μόνο 15 προσθέσεις.
- Συνεπώς έχουμε τρεις λιγότερες προσθέσεις (ή αφαιρέσεις) από τους τύπους του Strassen ενώ το πλήθος των πολλαπλασιασμών παραμένει επτά.

Αναδρομή

Η μέθοδος βασίζεται στην ακόλουθη ταυτότητα για διανύσματα τάξης n

$$x^T y = \sum_{i=1}^{n/2}(x_{2i-1} + y_{2i})(x_{2i} + y_{2i-1}) - \sum_{i=1}^{n/2} x_{2i-1}x_{2i} - \sum_{i=1}^{n/2} y_{2i-1}y_{2i}. \quad (28)$$

Όταν εφαρμοστεί (28) στον πολλαπλασιασμό AB με x μια γραμμή του A και y μια στήλη του B, το δεύτερο και τρίτο άθροισμα είναι κοινό στα άλλα εσωτερικά γινόμενα, τα οποία σχηματίζονται από την x γραμμή ή y στήλη και κατά συνέπεια υπολογίζονται μια φορά και στη συνέχεια επαναχρησιμοποιούνται.

Αναδρομή

Αν n είναι άρτιος και

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \in \mathbb{R}^{n \times n}, \quad A_{i,j} \in \mathbb{R}^{(n/2)\times(n/2)}$$

τότε ο υπολογισμός του A^{-1} δίνεται από τον ακόλουθο τύπο

$$A^{-1} = \left[\begin{array}{c|c} P_1 - P_3 P_6 P_2 & P_3 P_6 \\ \hline P_6 P_2 & -P_6 \end{array}\right] \tag{29}$$

όπου

$$\begin{aligned} P_1 &= A_{00}^{-1} & P_2 &= A_{10} P_1 \\ P_3 &= P_1 A_{01} & P_4 &= A_{10} P_3 \\ P_5 &= P_4 - A_{11} & P_6 &= P_5^{-1}. \end{aligned} \tag{30}$$

Οι πολλαπλασιασμοί πινάκων εκτελούνται με την μέθοδο του Strassen και οι υπολογισμοί των αντιστρόφων των P_1 και P_6 γίνονται με την αναδρομική κλήση της ίδιας της παρούσας μεθόδου.

Άσκηση

- Να επαληθευθεί η ισχύς των (29) ,(30) με την χρήση μιας LU παραγοντοποίησης του A και να αποδειχθεί ότι $T(n) = \Theta(n^{\log_2 7})$.

Αναδρομή

Ο αλγόριθμος Min-Max

Το πρόβλημα της εύρεσης του μέγιστου και του ελάχιστου στοιχείου ενός πίνακα A με n στοιχεία.

```
MinMax (A)
1.  min = a_1
2.  max = a_1
3.  for i = 2 to n do
4.          if a_i > max then
5.                  max = a_i
6.          else if a_i < min then
7.                  min = a_i
8.          end if
9.   end for
```

Σχήμα: Αλγόριθμος εύρεσης του μέγιστου και του ελάχιστου στοιχείου ενός πίνακα

Αναδρομή

Ο αλγόριθμος Min-Max

- Στη χείριστη περίπτωση, όπου τα στοιχεία του πίνακα θα είναι ταξινομημένα σε φθίνουσα σειρά η ακριβής πολυπλοκότητα χρόνου θα είναι $3(n-1)$,
- Στην περίπτωση όπου τα στοιχεία του πίνακα είναι ταξινομημένα σε αύξουσα σειρά η ακριβής πολυπλοκότητα θα είναι $2(n-1)$,
- Συνολικά θα έχουμε $\Theta(n)$.

Αναδρομή

Ο αλγόριθμος Min-Max

```
     MinMax (A, i, j, min, max)
 1.   if i ≥ j − 1 then
 2.         if a_i < a_j then
 3.               max = a_j
 4.               min = a_i
 5.         else
 6.               max = a_i
 7.               min = a_j
 8.          end if
 9.   else
10.         m = ⌊(i + j)/2⌋
11.         MinMax(A, i, m, min_1, max_1)
12.         MinMax(A, m, j, min_2, max_2)
13.         min = fmin(min_1, min_2)
14.         max = fmax(max_1, max_2)
15.    end if
```

Σχήμα: Αναδρομικός Αλγόριθμος εύρεσης του μέγιστου και του ελάχιστου στοιχείου ενός πίνακα

Αναδρομή

Ο αλγόριθμος Min-Max

- Ο έλεγχος $i \geq j - 1$ καλύπτει τις δύο περιπτώσεις $i = j$ και $i = j - 1$
- Οι συναρτήσεις fmax() και fmin() απαιτούν μία σύγκριση η κάθε μία για την εύρεση του μέγιστου ή του ελάχιστου (αντίστοιχα) μεταξύ δύο στοιχείων.

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \text{ or } n = 1, \\ 2T(\frac{n}{2}) + 3 & \text{if } n > 2. \end{cases}$$

Αναδρομή

Ο αλγόριθμος Min-Max

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + 3 \\
&= 2[2T(\frac{n}{2^2}) + 3] + 3 = 2^2 T(\frac{n}{2^2}) + 2 \times 3 + 3 \\
&= 2^2[2T(\frac{n}{2^3}) + 3] + 2 \times 3 + 3 = 2^3 T(\frac{n}{2^3}) + 2^2 \times 3 + 2 \times 3 + 3 \\
&= \ldots \\
&= 2^{k-1} T(\frac{n}{2^{k-1}}) + 2^{k-2} \times 3 + \ldots + 2 \times 3 + 3
\end{aligned}
$$

Αναδρομή

Ο αλγόριθμος Min-Max

και επομένως αν $n = 2^k$ έχουμε

$$\begin{aligned} T(n) &= \frac{n}{2} \times T(2) + [\frac{2^{k-1} - 1}{2 - 1}] \times 3 \\ &= n + 3 \times 2^{k-1} - 3 \\ &= n + 3\frac{n}{2} - 3 \\ &= 5\frac{n}{2} - 3 \end{aligned}$$

Η ακριβής πολυπλοκότητα του αναδρομικού αλγορίθμου είναι μικρότερη από την αντίστοιχη του επαναληπτικού, σε αντίθεση με το πρόβλημα των αριθμών Fibonacci

Αναδρομή

QuickSort

- Αλγόριθμος ταξινόμησης: δημιουργήθηκε από τον C. A. R. Hoare στη χείριστη περίπτωση $\Theta(n^2)$ είναι συχνά η καλύτερη πρακτική επιλογή, γιατί είναι αποδοτικός στη μέση περίπτωση $\Theta(nlogn)$
- έχει επίσης το πλεονέκτημα να ταξινομεί στον ίδιο πίνακα
- Η QuickSort και η MergeSort, ακολουθούν το Divide-and-Conquer.

Αναδρομή

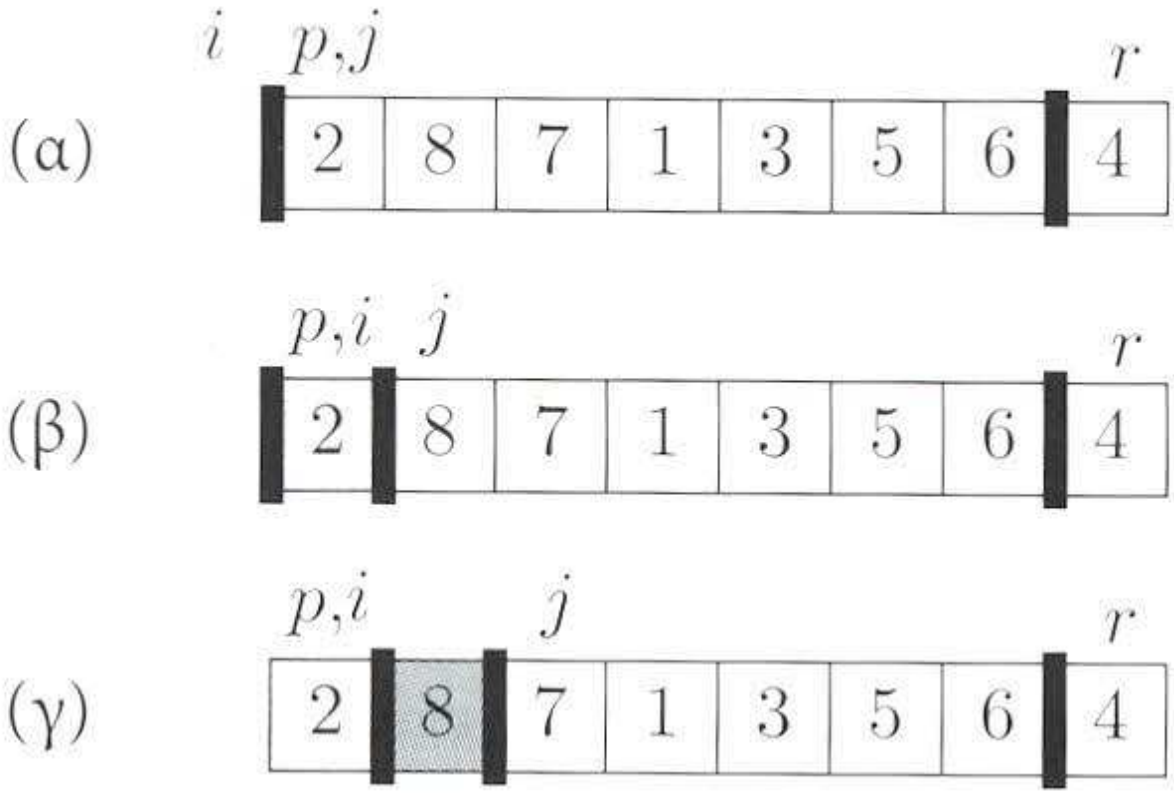
QuickSort

Divide: Διαμέριση του πίνακα $A[p \ldots r]$ σε δύο, πιθανώς κενούς, υποπίνακες $A[p \ldots q-1]$ και $A[q+1 \ldots r]$. Διαμερίζοντας, βρίσκουμε τη θέση του $A[q]$ στον πίνακα $A[p \ldots r]$.

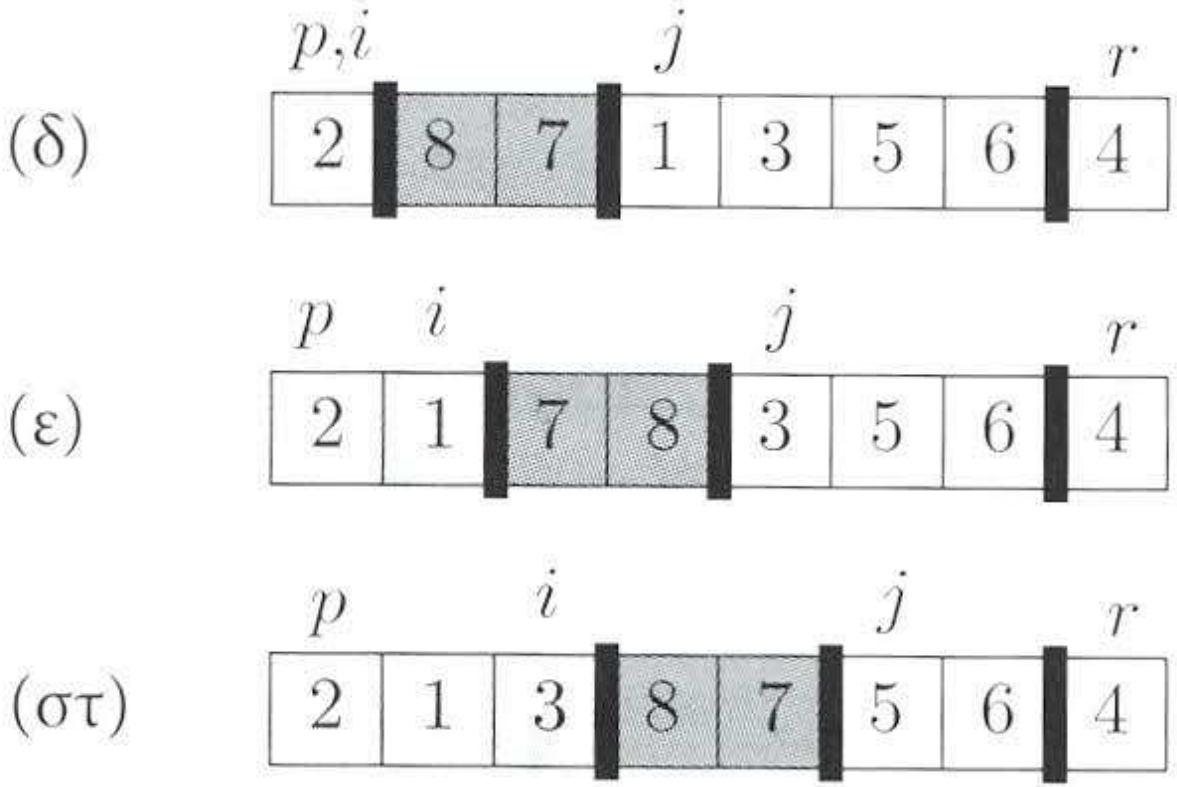
Conquer: Ταξινόμηση των δύο υποπινάκων $A[p \ldots q-1]$ και $A[q+1 \ldots r]$ με αναδρομικές κλήσεις της QuickSort.

Combine: Αφού οι υποπίνακες ταξινομούνται στον ίδιο πίνακα δε χρειάζεται παραπάνω εργασία για να συνδυάσουμε τις λύσεις με τους υποπίνακες.

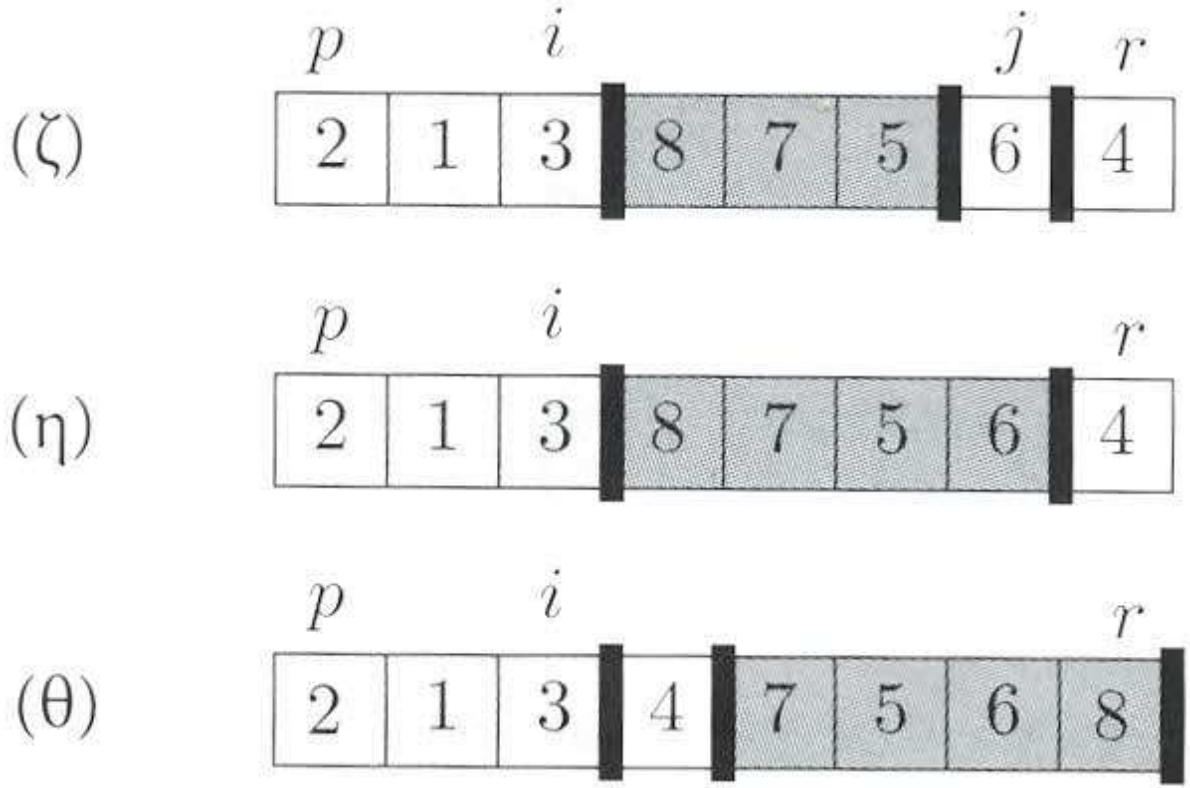
Αναδρομή



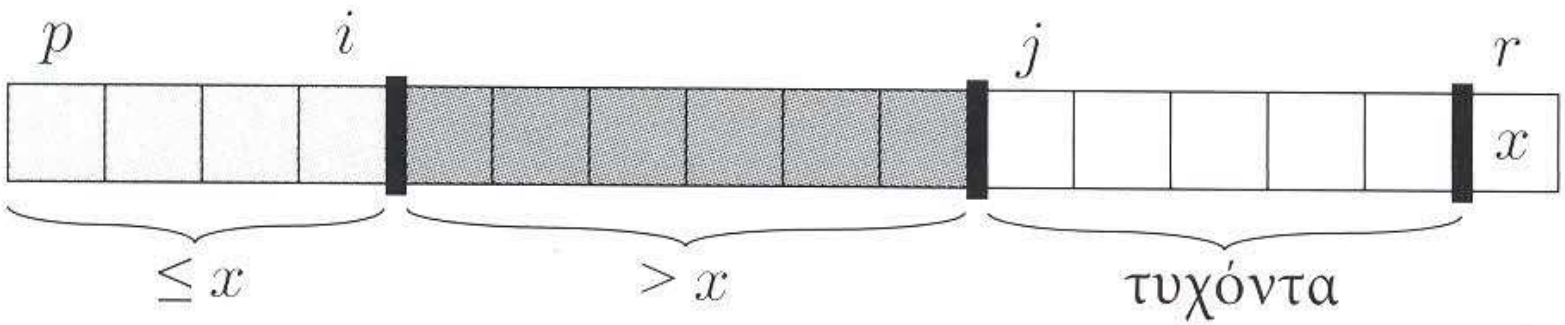
Αναδρομή



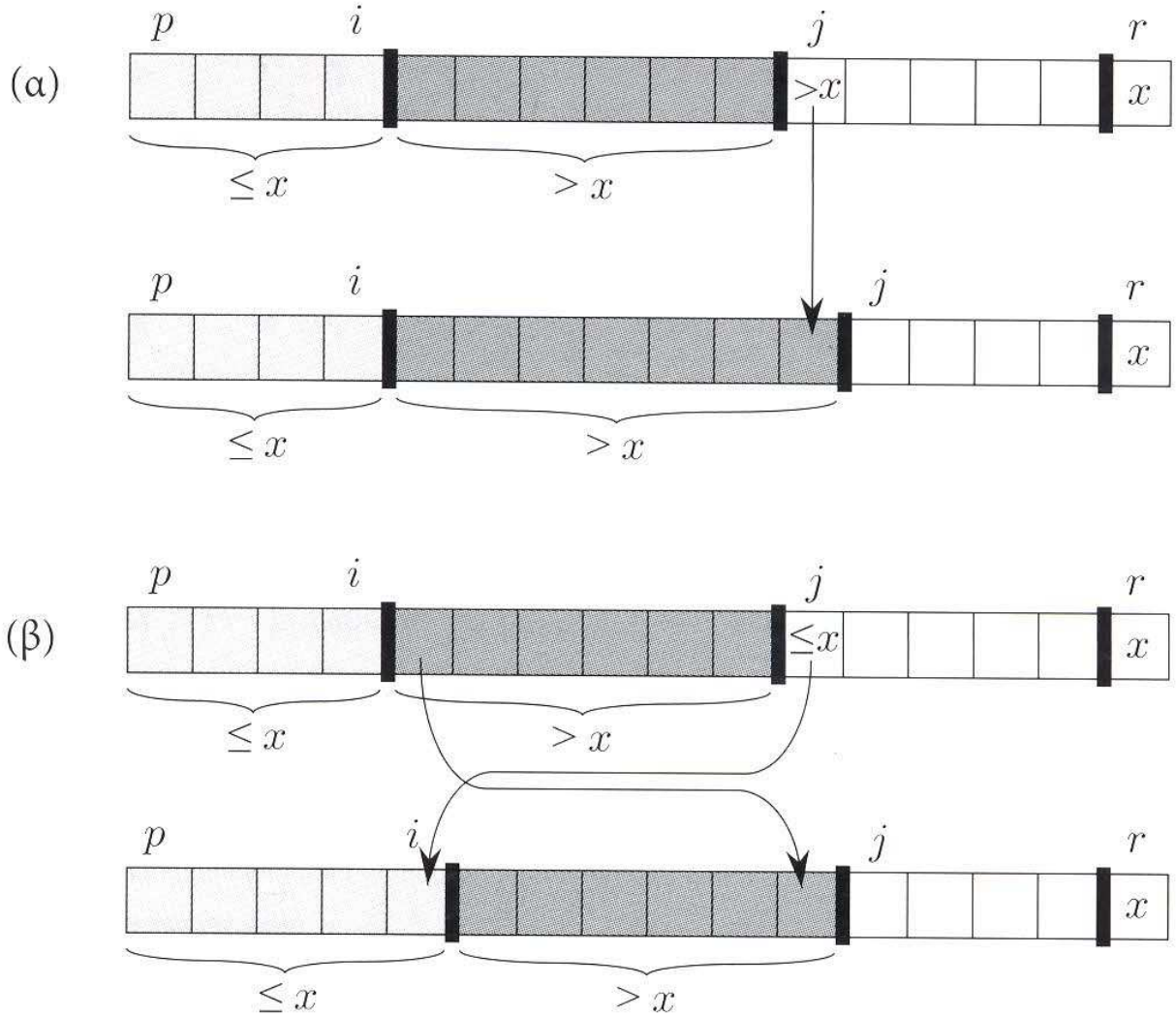
Αναδρομή



Αναδρομή



Αναδρομή



Αναδρομή

QuickSort

```
     QuickSort (A, p, r)
1.   if p < r then
2.         q =Partition(A, p, r)
3.          QuickSort (A, p, q − 1)
4.         QuickSort (A, q + 1, r)
5.   end if
```

Σχήμα: Αλγόριθμος QuickSort

Αναδρομή

QuickSort

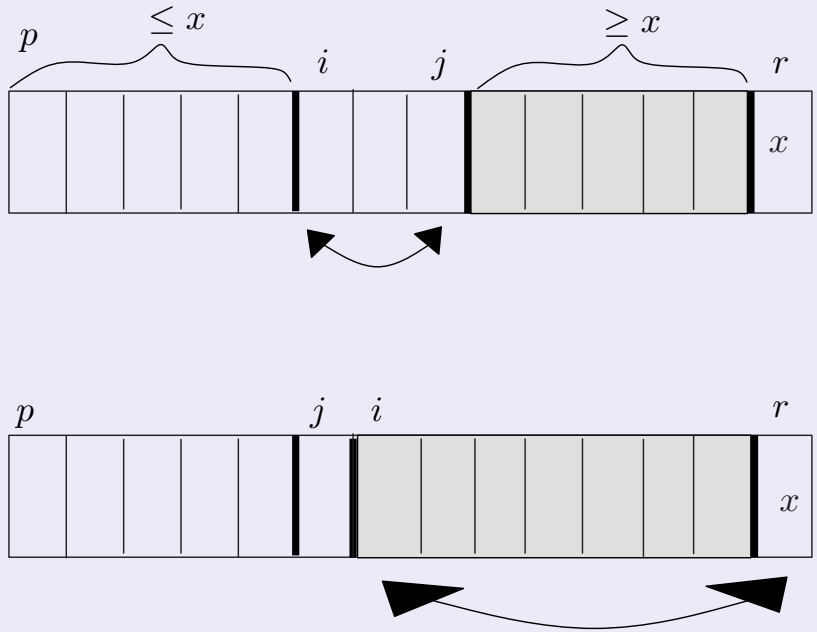
Το κλειδί είναι η διαδικασία της διαμέρισης (Partition)

```
     PARTITION (A, p, r)
1.   x = A[r]
2.   i = p − 1
3.   for j = p to r − 1 do
4.        if A[j] ≤ x
5.             i = i + 1
6.             swap(A[i], A[j])
7.        end if
8.   end for
9.   swap(A[i + 1], A[r])
10.  return i + 1
```

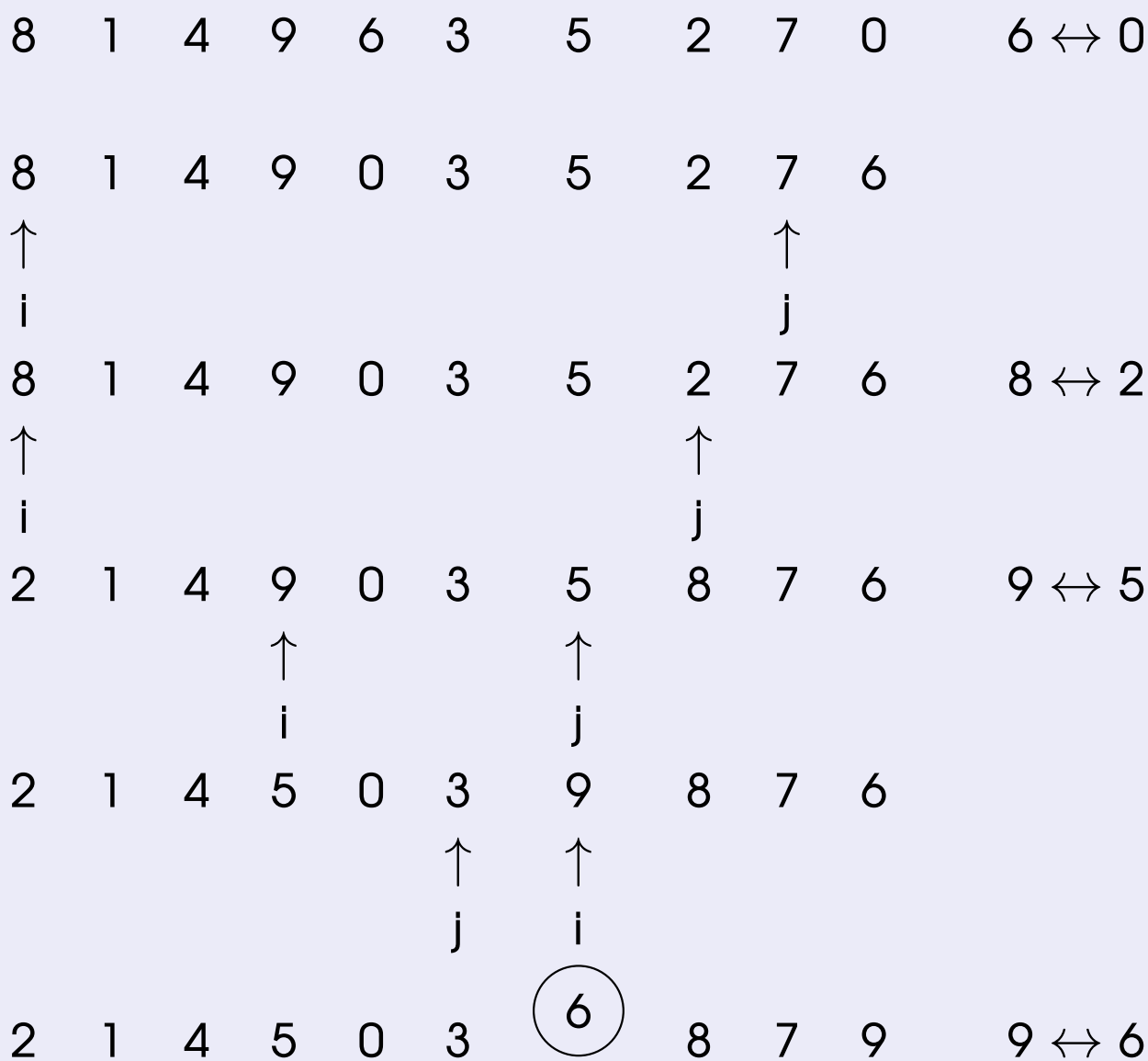
Σχήμα: Διαδικασία διαμέρισης με βάση το οδηγό στοιχείο.

Διαμέριση του Hoare(A,p,r)

$x = A[r]$
$i = p - 1$
$j = r + 1$
while(TRUE)
    repeat
        $j = j - 1$
    until $A[j] \leq x$
    repeat
        $i = i + 1$
    until $A[i] \geq x$
    if $i < j$ then
        swap$(A[i], A[j])$
    else
        return i
end while



Medium of Three Partitioning



QuickSort

Ορθότητα QuickSort

Αν η διαμέριση (Partition) είναι ορθή, τότε Π(n)= ¨Η QuickSort ταξινομεί σωστά κάθε πίνακα μεγέθους n¨

Απόδειξη με επαγωγή

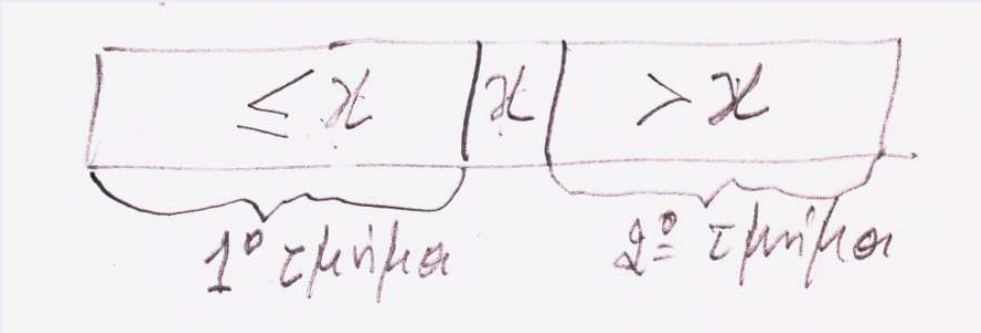
1. Βάση

Κάθε πίνακας μεγέθους $n = 1$ είναι ήδη ταξινομημένος. Η QuickSort επιστρέφει τον πίνακα αυτό, συνεπώς ταξινομεί σωστά τον πίνακα αυτό (τετριμένη περίπτωση).

2. Επαγωγικό βήμα

Έστω ένας πίνακας μεγέθους $n \geq 2$. Θα δείξουμε ότι αν η Π(k) ισχύει για κάθε $k < n$, τότε ισχύει και η Π(n).

Η διαμέριση είναι ορθή, λόγω της υπόθεσης, και έχει σαν αποτέλεσμα:



QuickSort

Απόδειξη με επαγωγή

- Έστω k_1, k_2 τα μεγέθη του $1^{ου}$ και $2^{ου}$ τμήματος αντίστοιχα, με $k_1, k_2 < n$.
- Από την επαγωγική υπόθεση έχουμε ότι $\Pi(k_1)$ και $\Pi(k_2)$ ισχύουν, δηλαδή ότι το $1^{ο}$ και $2^{ο}$ τμήμα έχουν ταξινομηθεί σωστά.
- Συνεπώς, μετά τις αναδρομικές κλήσεις ολόκληρος ο πίνακας θα έχει ταξινομηθεί σωστά.

Medium of Three Partitioning (συν.)

Υποθέστε ότι όλα τα κλειδιά είναι ίσα με το οδηγό.

Ερώτηση:

- Ο i θα σταματήσει όταν συναντήσει ένα κλειδί ίσο με το οδηγό ;
- Ο j θα σταματήσει όταν συναντήσει ένα κλειδί ίσο με το οδηγό ;

Ανάλυση

$$T(n) = T(q-1) + T(n-q) + \Theta(n), \qquad q \in [1, n] \tag{31}$$

Χείριστη Περίπτωση

Αν $A[1..n]$ ταξινομημένος Για $q = 1$ η (31) δίνει:

$$T(n) = T(0) + T(n-1) + \Theta(n), \qquad n > 1$$

ή αφού $T(0) = 0$

$$T(n) = T(n-1) + \Theta(n)$$

Αναδρομή

QuickSort

Διαμέριση Χείριστης Περίπτωσης

χωρίζει τον πίνακα μεγέθους n σε 2 υποπίνακες μεγέθους $n-1$ και 0 στοιχείων.

Η διαμέριση κοστίζει $\Theta(n)$ χρόνο.

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

Επομένως $T(n) = \Theta(n^2)$.

Συμβαίνει μόνο όταν η είσοδος είναι ένας ήδη ταξινομημένος πίνακας.

Αναδρομή

QuickSort

Διαμέριση Βέλτιστης Περίπτωσης

Η διαδικασία Partition δημιουργεί δύο υποπροβλήματα, το ένα υποπρόβλημα θα είναι μεγέθους $\lfloor n/2 \rfloor$, το άλλο μεγέθους $\lceil n/2 \rceil - 1$.

$$T(n) \leq 2T(n/2) + \Theta(n)$$

$$T(n) = O(nlogn).$$

Μέση Περίπτωση

Υπόθεση: Η πιθανότητα να επιλεγεί ως οδηγό στοιχείο οποιοδήποτε συγκεκριμένο στοιχείο είναι $1/n$

$$C_{\mu\pi} = \sum_{d \in D} p(d)\text{κόστος}(d)$$

$$T(n) = \frac{1}{n}\sum_{q=1}^{n}[T(q-1) + T(n-q)] + cn$$

ή

$$T(n) = \frac{2}{n}\sum_{q=1}^{n-1} T(q) + cn \tag{1}$$

Πολλαπλασιάζοντας την (1) επί n έχουμε:

$$nT(n) = 2\sum_{q=1}^{n-1} T(q) + cn^2 \tag{32}$$

Μέση Περίπτωση (συν.)

για $n = n - 1$ η (32) δίνει

$$(n-1)T(n-1) = 2\sum_{q=1}^{n-2} T(q) + c(n-1)^2 \tag{33}$$

Αφαιρώντας την (32) από την (33)

$$nT(n) - (n-1)T(n-1) = 2T(n-1) + 2cn - c$$

ή (αγνοώντας το $-c$)

$$nT(n) = (n+1)T(n-1) + 2cn \tag{34}$$

Διαιρώντας την (34) δια $n(n+1)$ έχουμε:

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1} \tag{35}$$

Ανάλυση(συν.)

Μέση Περίπτωση (συν.)

Για διαφορετικές τιμές του $n = n-1, n-2, \cdots$ η (35) δίνει

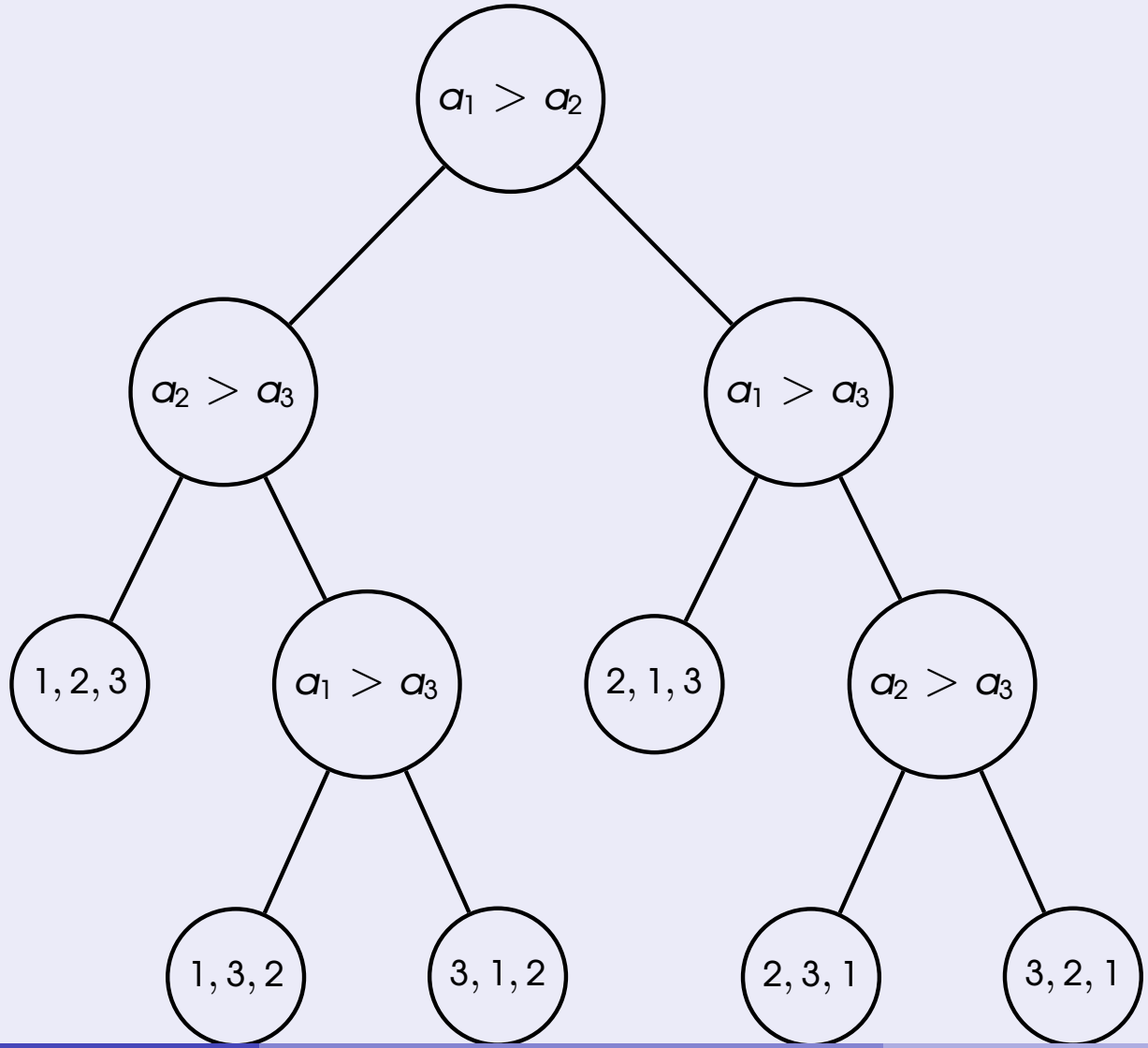
$$\frac{T(n-1)}{n} = \frac{T(n-2)}{n-1} + \frac{2c}{n}$$
$$\frac{T(n-2)}{n-1} = \frac{T(n-3)}{n-2} + \frac{2c}{n-1}$$
$$\vdots \qquad \vdots$$
$$\frac{T(2)}{3} = \frac{T(1)}{2} + \frac{2c}{3}$$

προσθέτοντας κατά μέλη προκύπτει:

$$\frac{T(n)}{n+1} = \frac{T(1)}{2} + 2c\sum_{i=3}^{n+1}\frac{1}{i}, \qquad T(1) = 1$$

Αναδρομή

Δένδρα Απόφασης



Αναδρομή

Δένδρα Απόφασης

Κάθε δένδρο απόφασης έχει $n!$ φύλλα, όσες οι δυνατές μεταθέσεις των n στοιχείων. Το ύψος h ενός τέτοιου δένδρου είναι $h \geq \lceil \log_2(n!) \rceil$. Από τον τύπο του Stirling

$$n! = \sqrt{(2\pi n)} \left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + o(\frac{1}{n})\right)$$

και επομένως

$$\log_2(n!) = \log_2 \left(\sqrt{(2\pi n)} \left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + o(\frac{1}{n})\right)\right) =$$

$$n \log_2 n - n \log_2 e + \frac{1}{2} \log_2 n + \frac{1}{2} \log_2 2\pi + \log_2 \left(1 + \frac{1}{12n} + o\left(\frac{1}{n}\right)\right)$$

τελικά

$$h = \Omega(n \log n).$$

Αναδρομή

Δένδρα Απόφασης

- Ακολουθώντας μια διαδρομή στο δένδρο μπορούμε να έχουμε μια ταξινόμηση των n στοιχείων.
- Επομένως η ταξινόμηση n στοιχείων βασισμένη σε συγκρίσεις των στοιχείων ανά δύο, απαιτεί $\Omega(n \log n)$ συγκρίσεις.
- Συμπεραίνουμε ότι οι αλγόριθμοι ταξινόμησης
  - με σωρό
  - με συγχώνευση
  - με διχοτομική εισαγωγή

  αγνοώντας τις καταχωρήσεις

  είναι βέλτιστοι αλγόριθμοι στην χείριστη περίπτωση.
- Αντίθετα ο αλγόριθμος QuickSort
  - δεν είναι βέλτιστος στην χείριστη περίπτωση
  - είναι βέλτιστος ως προς την κατά μέσο όρο πολυπλοκότητα.

Αναδρομικές

Μέθοδος Πρόβλεψης

1. Πρόβλεψη
2. Επαγωγή

Παράδειγμα

$$T(n) = 4T(n/2) + n$$
$$T(1) = \Theta(1)$$

Πρόβλεψη: $T(n) = O(n^3)$
Επαγωγή: $T(k) = O(k^3)$
$T(k) \leq ck^3$ για $k < n$

Αναδρομικές

Μέθοδος Πρόβλεψης (συν.)

Απόδειξη για $k = n$

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
T(n) &\leq 4[c(n/2)^3] + n \\
&= \frac{1}{2}cn^3 + n \\
&= cn^3 - \left(\frac{1}{2}cn^3 - n\right) \\
&\leq cn^3 \\
\text{αν } & \frac{1}{2}cn^3 - n \geq 0 \\
\text{ή } & \frac{1}{2}cn^2 - 1 \geq 0 \\
\text{ή } & c \geq \frac{2}{n^2} \text{ ή } c \geq 2 \quad \forall \quad n \geq n_0 = 1
\end{aligned}
$$

Μέθοδος Πρόβλεψης (συν.)

Προσοχή

$$
\begin{aligned}
\text{Αν} \qquad & T(n) = O(n^2) \qquad \text{τότε} \\
& T(k) \leq ck^2 \\
& T(n) = 4T(n/2) + n \\
& \qquad \leq 4[c(n/2)^2] + n \\
& \qquad = cn^2 + n \\
& \qquad = cn^2 - (-n) \\
& \qquad \leq cn^2 \quad ;;; \\
& T(k) \leq c_1 k^2 - c_2 k, k < n
\end{aligned}
$$

Αναδρομή

The Master Theorem

- Μία συνάρτηση $f(n)$ είναι πολυωνυμικά φραγμένη αν $f(n) = O(n^k)$ για κάποια σταθερά k.
- Κάθε εκθετική συνάρτηση με βάση μεγαλύτερη του 1 αυξάνεται γρηγορότερα από κάθε πολυωνυμική συνάρτηση. Για a και b σταθερές, με $a > 1$, ότι:

$$\lim_{n\to\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = o(a^n).$$

- Η $f(n)$ είναι πολυλογαριθμικά φραγμένη αν $f(n) = O(\log^k n)$ για κάποια σταθερά k.
- Κάθε θετική πολυωνυμική συνάρτηση αυξάνεται γρηγορότερα από κάθε πολυλογαριθμική συνάρτηση.

$$\lim_{n\to\infty} \frac{\log^k n}{n^b} = 0 \Rightarrow \log^k(n) = o(n^b)$$

για k και b θετικές σταθερές

Αναδρομή

Θεώρημα Κυρίαρχου Όρου The Master Theorem

Ας θεωρήσουμε την

$$T(n) = aT(\frac{n}{b}) + f(n)$$

όπου $a \geq 1$ και $b > 1$ και $f(n)$ μια ασυμπτωτική θετική συνάρτηση με το $\frac{n}{b}$ εννοούμε $\lfloor \frac{n}{b} \rfloor$ ή $\lceil \frac{n}{b} \rceil$.

Theorem

Η $T(n)$ φράσσεται ασυμπτωτικά ως εξής:

1. *Εάν $f(n) = O(n^{\log_b a - \epsilon})$ για κάποια σταθερά $\epsilon > 0$ τότε $T(n) = \Theta(n^{\log_b a})$.*
2. *Εάν $f(n) = \Theta(n^{\log_b a})$ τότε $T(n) = \Theta(n^{\log_b a} \lg n)$.*
3. *Εάν $f(n) = \Omega(n^{\log_b a + \epsilon})$ για κάποια σταθερά $\epsilon > 0$ και εάν $af(\frac{n}{b}) \leq cf(n)$ για κάποια σταθερά $c < 1$ και όλα τα αρκετά μεγάλα n τότε $T(n) = \Theta(f(n))$.*

Αναδρομή

The Master Theorem

- Συγκρίνουμε τη συνάρτηση $f(n)$ με τη συνάρτηση $n^{\log_b a}$. Διαισθητικά, η λύση της αναδρομικής εξίσωσης καθορίζεται από την μεγαλύτερη από τις δύο αυτές συναρτήσεις.
- Στην πρώτη περίπτωση του θεωρήματος, η συνάρτηση $f(n)$ δεν πρέπει απλά να είναι μικρότερη, πρέπει να είναι *πολυωνυμικά* μικρότερη από την $n^{\log_b a}$.
- Στην τρίτη περίπτωση η $f(n)$ πρέπει επίσης να είναι *πολυωνυμικά* μεγαλύτερη από την $n^{\log_b a}$
- πρέπει να ικανοποιεί και την συνθήκη κανονικότητας $af(\frac{n}{b}) \leq cf(n)$
- οι τρεις περιπτώσεις δεν καλύπτουν όλες τις πιθανές εκδοχές για τη συνάρτηση $f(n)$.

Η Απόδειξη για Δυνάμεις του b

Λήμμα 1

Έστω $a \geq 1$ και $b > 1$ σταθερές και $f(n)$ μια μη αρνητική συνάρτηση που ορίζεται για δυνάμεις του b. Αν η $T(n)$ ορίζεται για δυνάμεις του b από την αναδρομική σχέση

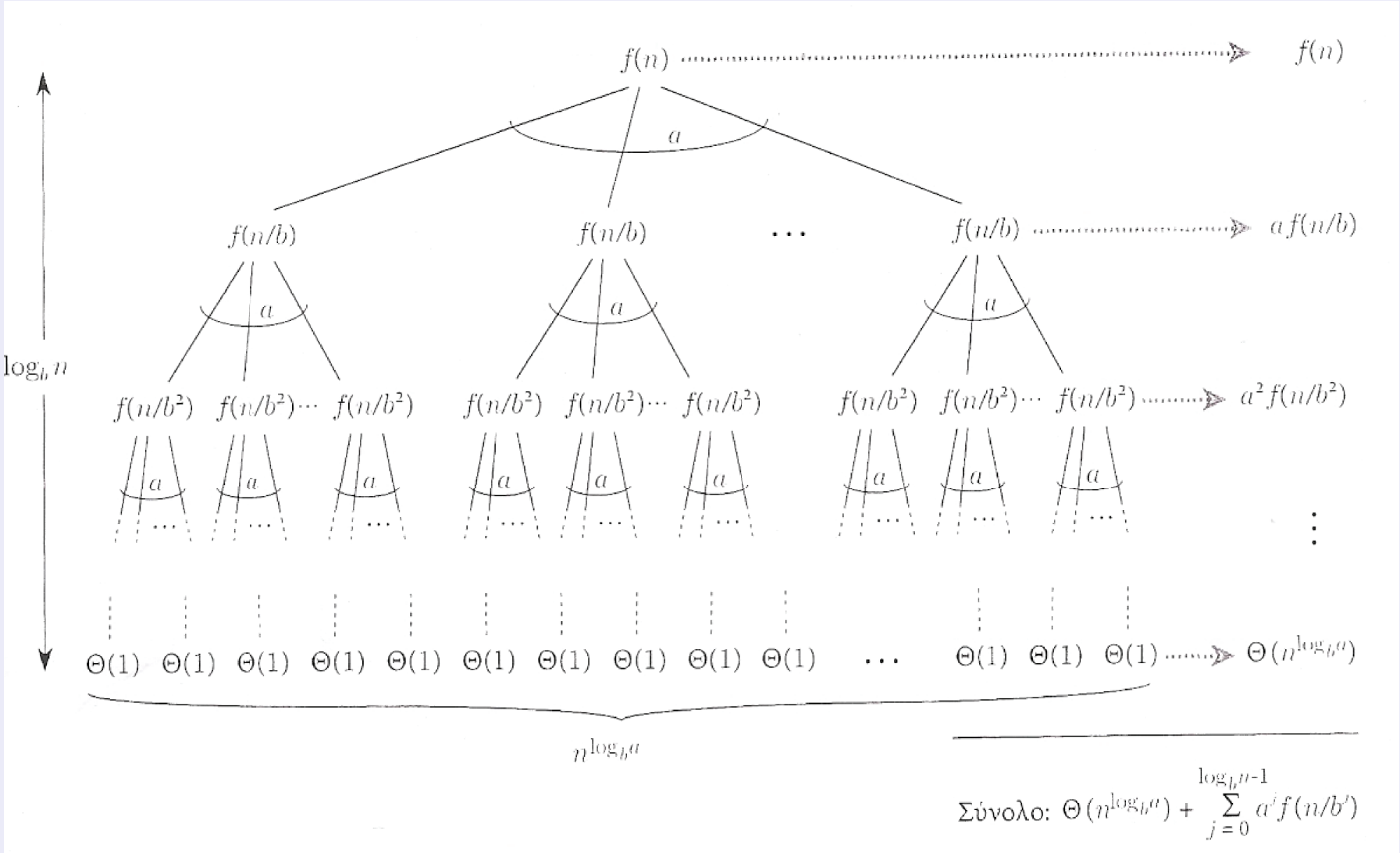
$$T(n) \begin{cases} \Theta(1), & \text{εάν } n = 1, \\ aT(n/b) + f(n), & \text{εάν } n = b^i, \end{cases}$$

όπου i θετικός ακέραιος, τότε

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα1, Απόδειξη



Η Απόδειξη για Δυνάμεις του b

Λήμμα 2

Έστω $a \geq 1$ και $b > 1$ σταθερές και $f(n)$ μια μη αρνητική συνάρτηση που ορίζεται για δυνάμεις του b. Στην περίπτωση αυτή, μια συνάρτηση $g(n)$ που ορίζεται επί των δυνάμεων του b μέσω του αθροίσματος

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$

φράσσεται ασυμπτωτικά για δυνάμεις του b ως εξής:

1. Αν $f(n) = O(n^{\log_b a - \epsilon})$ για κάποια σταθερά $\epsilon > 0$, τότε $g(n) = O(n^{\log_b a})$
2. Αν $f(n) = \Theta(n^{\log_b a})$, τότε $g(n) = \Theta(n^{\log_b a} lgn)$
3. Αν $af(n/b) \leq cf(n)$ για κάποια σταθερά $c < 1$ και για όλα τα $n \geq b$, τότε $g(n) = \Theta(f(n))$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη

Για την περίπτωση (1) έχουμε:

$$f(n) = O(n^{\log_b a - \epsilon})$$
$$f(n/b^j) = O((n/b^j)^{\log_b a - \epsilon})$$
$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right)$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη (συν.)

$$\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a-\epsilon} = n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a-\epsilon} \sum_{j=0}^{\log_b n-1} (b^\epsilon)^j$$

$$= n^{\log_b a-\epsilon} \left(\frac{b^{\epsilon \log_b n}-1}{b^\epsilon-1}\right)$$

$$= n^{\log_b a-\epsilon} \left(\frac{n^\epsilon-1}{b^\epsilon-1}\right)$$

Η τελευταία έκφραση μπορεί να γραφτεί: $n^{\log_b a-\epsilon} O(n^\epsilon) = O(n^{\log_b a})$

$$g(n) = O(n^{\log_b a})$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη

Για την περίπτωση (2) έχουμε:

$$f(n) = \Theta(n^{\log_b a})$$
$$f(n/b^j) = \Theta((n/b^j)^{\log_b a})$$
$$g(n) = \Theta\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right)$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη (συν.)

$$\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} = n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n-1} 1$$

$$= n^{\log_b a} \log_b n$$

Συνεπώς:

$$g(n) = \Theta(n^{\log_b a} \log_b n)$$

$$= \Theta(n^{\log_b a} lgn)$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη

Η περίπτωση (3) αποδεικνύεται με όμοιο τρόπο:

$$
\begin{aligned}
g(n) &= \Omega(f(n)) \\
af(n/b) &\leq cf(n), c < 1 \text{ και } n \geq b \\
f(n/b) &\leq c/af(n) \\
f(n/b^j) &\leq (c/a)^j f(n) \\
a^j f(n/b^j) &\leq c^j f(n)
\end{aligned}
$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη (συν.)

$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \\
&\leq \sum_{j=0}^{\log_b n - 1} c^j f(n) \\
&\leq f(n) \sum_{j=0}^{\infty} c^j \\
&\leq f(n) \left(\frac{1}{1-c} \right) \\
&= O(f(n))
\end{aligned}
$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 2, Απόδειξη (συν.)

$$\sum_{j=0}^{\log_b n-1} a^j f(n/b^j) = \Omega(f(n))$$

Επομένως:

$$g(n) = \Theta(f(n))$$

Η Απόδειξη για Δυνάμεις του b

Λήμμα 3

Έστω $a \geq 1$ και $b > 1$ σταθερές και $f(n)$ μια μη αρνητική συνάρτηση που ορίζεται για δυνάμεις του b. Αν η $T(n)$ ορίζεται για δυνάμεις του b από την αναδρομική σχέση

$$T(n) \begin{cases} \Theta(1), & \text{εάν } n = 1, \\ aT(n/b) + f(n), & \text{εάν } n = b^i, \end{cases}$$

όπου i θετικός ακέραιος, τότε η $T(n)$ φράσσεται ασυμπτωτικά για δυνάμεις του b ως εξής:

1. Αν $f(n) = O(n^{\log_b a - \epsilon})$ για κάποια σταθερά $\epsilon > 0$, τότε $T(n) = \Theta(n^{\log_b a})$.
2. Αν $f(n) = \Theta(n^{\log_b a})$, τότε $T(n) = \Theta(n^{\log_b a} lgn)$.
3. Αν $f(n) = \Omega(n^{\log_b a} + \epsilon)$ για κάποια σταθερά $\epsilon > 0$, και αν $af(n/b) \leq cf(n)$ για κάποια σταθερά $c < 1$ και για όλα τα n από κάποια τιμή και πάνω, τότε $T(n) = \Theta(f(n))$.

Η Απόδειξη για Δυνάμεις του b

Λήμμα 3, Απόδειξη

Για την περίπτωση (1) έχουμε:

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a}) + O(n^{\log_b a}) \\ &= \Theta(n^{\log_b a}) \end{aligned}$$

Για την περίπτωση (2) έχουμε:

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} lgn)) \\ &= \Theta(n^{\log_b a} lgn) \end{aligned}$$

Για την περίπτωση (3) έχουμε:

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a}) + \Theta(f(n)) \\ &= \Theta(f(n)), \\ \text{διότι } f(n) &= \Omega(n^{\log_b a+\epsilon}) \end{aligned}$$

Αναδρομικές

Master Theorem

$$T(n) = 4T(n/2) + n$$

$$a = 4 \quad b = 2 \quad f(n) = n$$

$$n^{\log_b a} = n^{\log_2 4} = n^2$$

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

$$\text{Περίπτωση } 1\text{η} \quad (\epsilon = 1)$$

$$T(n) = \Theta(n^2)$$

Αναδρομικές

Master Theorem

$$T(n) = 4T(n/2) + n^2$$

$$n^{\log_b a} = n^2 \quad f(n) = n^2$$

$$f(n) = n^2 = \Theta(n^2)$$

Περίπτωση 2η

$$T(n) = \Theta(n^2 \log_2 n)$$

Αναδρομικές

Master Theorem

$$T(n) = 4T(n/2) + n^3$$
$$f(n) = n^3 = \Omega(n^2)$$
$$af(n/b) \leq cf(n), \quad c < 1$$
$$4(n/2)^3 = \frac{1}{2}n^3$$
$$T(n) = \Theta(f(n)) = \Theta(n^3)$$

Αναδρομή

Παράδειγμα 1

$$T(n) = 9T(\frac{n}{3}) + n$$

- Έχουμε $a = 9$, $b = 3$ και $f(n) = n$.
- Συνεπώς $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$
- Αφού λοιπόν $f(n) = O(n^{\log_3 9 - \epsilon})$ όπου $\epsilon = 1$, μπορούμε να εφαρμόσουμε την περίπτωση 1
- Η λύση είναι $T(n) = \Theta(n^2)$

Αναδρομή

Παράδειγμα 2

$$T(n) = T(\frac{2n}{3}) + 1$$

- Έχουμε $a = 1, b = \frac{3}{2}$ και $f(n) = 1$
- Συνεπώς $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
- Αφού $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$, εφαρμόζουμε την περίπτωση 2
- Η λύση είναι $T(n) = \Theta(\log n)$

Αναδρομή

Παράδειγμα 3

$$T(n) = 3T(\frac{n}{4}) + n \log n$$

- Έχουμε $a = 3, b = 4, f(n) = n \log n$ και $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$.
- Δηλαδή $f(n) = \Omega(n^{\log_4 3 + \epsilon})$ με $\epsilon \approx 0.2$
- Επιπρόσθετα $af(\frac{n}{b}) = 3(\frac{n}{4}) \log(\frac{n}{4}) \leq \frac{3}{4} n \log n = cf(n)$ με $c = \frac{3}{4}$
- Συνεπώς θα εφαρμόσουμε την περίπτωση 3
- Η λύση είναι $T(n) = \Theta(n \log n)$

Αναδρομή

Παράδειγμα 4

$$T(n) = 2T(\frac{n}{2}) + n \log n$$

- Έχουμε $a = 2, b = 2$ και $f(n) = n \log n$
- Συνεπώς $n^{\log_b a} = n$
- Ενώ $n \log n > n$ ο λόγος $\frac{n \log n}{n} = \log n$ είναι ασυμπτωτικά μικρότερος από το n^{ϵ} για κάθε θετική σταθερά ϵ
- Κατά συνέπεια η $f(n)$ δεν είναι πολυωνυμικά μεγαλύτερη, κατά ένα παράγοντα n^{ϵ}, από την συνάρτηση $n^{\log_b a} = n$ και η περίπτωση 3 του θεωρήματος δεν μπορεί να εφαρμοστεί.

Αναδρομή

Αναδρομικές γραμμικές εξισώσεις

Οι αναδρομικές εξισώσεις χωρίζονται σε κατηγορίες ανάλογα με

1. τον τύπο της συνάρτησης f. Η συνάρτηση f μπορεί να είναι γραμμικός συνδυασμός των $T(p)$, με συντελεστές σταθερούς ή μεταβλητούς ή πωλυώνυμα $T(p)$ κ.τ.λ.
2. το σύνολο τιμών p που εμπλέκονται για τον υπολογισμό του $T(n)$. Πιο συγκεκριμένα έχουμε:
   - $T(n) = f(T(n-1))$ εξίσωση τάξης 1
   - $T(n) = f(T(n-1), \ldots, T(n-k))$ εξίσωση τάξης k για k σταθερό
   - $T(n) = f(\{T(p); \forall p < n\})$ πλήρης εξίσωση

Αναδρομή

Γραμμικές αναδρομές τάξης k

Οι γραμμικές αναδρομές τάξης k είναι της μορφής

$$T(n) = f(n, T(n-1), \ldots, T(n-k)) + g(n)$$

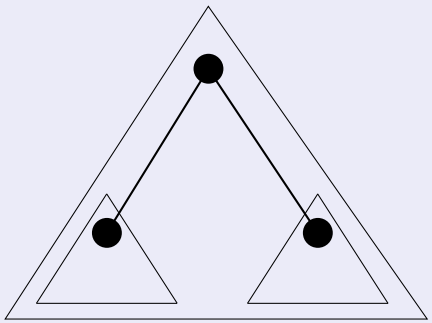
όπου $k \geq 1$ σταθερά, f γραμμικός συνδυασμός των $T(i)$ για $n-k \leq i \leq n-1$ και g οποιαδήποτε συνάρτηση του n.

Αναδρομή

Παράδειγμα

Πλήρες δυαδικό δένδρο. Ο αριθμός κόμβων a_n του πλήρους δυαδικού δένδρου ύψους n είναι ίσος με τον αριθμό των κόμβων των δύο υποδένδρων ύψους $n-1$ συν 1 κόμβο ο οποίος είναι η ρίζα.

$$a_n = 2a_{n-1} + 1 \ (n \geq 1, k = 1, a_0 = 1)$$



Σχήμα: Πλήρες δυαδικό δένδρο

Αναδρομή

Αναδρομές διαμερίσεων

Ο γενικός τύπος των αναδρομικών εξισώσεων αυτής της κατηγορίας είναι:

$$T(n) = aT(\frac{n}{b}) + d(n)$$

όπου a, b ακέραιες σταθερές και $d(n)$ οποιαδήποτε συνάρτηση.

Παράδειγμα

Ο αλγόριθμος Mergesort.

$$T(n) = 2T(\frac{n}{2}) + n - 1 \quad n \geq 2$$
$$T(1) = 0$$

όπου $n - 1$ είναι το κόστος της συγχώνευσης.

Αναδρομή

Αναδρομές πλήρεις, γραμμικές ή πολυωνυμικές

Ο γενικός τύπος των αναδρομικών εξισώσεων αυτής της κατηγορίας είναι:

$$T(n) = f(n, T(n-1), T(n-2), \ldots, T(0)) + g(n)$$

όπου f μια συνάρτηση γραμμική ή πολυωνυμική των $T(i)$ και $g(n)$ οποιαδήποτε συνάρτηση.

Αναδρομή

Παράδειγμα 1 Γραμμικών Αναδρομικών Εξισώσεων

Να βρεθεί το πλήθος των κόμβων ενός πλήρους δυαδικού δέντρου ύψους n.

$$a_n = a_{n-1} + 2^n, \quad n \geq 1$$
$$a_0 = 1$$

Λύση:

$$a_n = a_{n-1} + 2^n$$
$$a_{n-1} = a_{n-2} + 2^{n-1}$$
$$a_{n-2} = a_{n-3} + 2^{n-2}$$
$$\dots$$
$$a_2 = a_1 + 2^2$$
$$a_1 = a_0 + 2^1$$

Αναδρομή

... Παράδειγμα 1 Γραμμικών Αναδρομικών Εξισώσεων

Αθροίζοντας κατά μέλη λαμβάνουμε:

$$a_n = a_0 + \sum_{i=1}^{n} 2^i, \quad n \geq 1$$

$$a_n = 1 + \frac{2^{n+1} - 2}{2 - 1}$$

$$a_n = 2^{n+1} - 1$$

Αναδρομή

Παράδειγμα 2 Γραμμικών Αναδρομικών Εξισώσεων

Να επιλυθεί το προηγούμενο πρόβλημα με διαφορετική αναδρομική εξίσωση.

Λύση:

$$
\begin{aligned}
a_n &= 2a_{n-1} + 1 \quad (\times 2^0) \\
a_{n-1} &= 2a_{n-2} + 1 \quad (\times 2^1) \\
a_{n-2} &= 2a_{n-3} + 1 \quad (\times 2^2) \\
&\dots \\
a_2 &= 2a_1 + 1 \quad (\times 2^{n-2}) \\
a_1 &= 2a_0 + 1 \quad (\times 2^{n-1})
\end{aligned}
$$

Αναδρομή

... Παράδειγμα 2 Γραμμικών Αναδρομικών Εξισώσεων

Χρησιμοποιώντας τη μέθοδο των αθροιζομένων παραγόντων λαμβάνουμε:

$$a_n = 2^n a_0 + \sum_{i=0}^{n-1} 2^i$$

$$a_n = 2^n + \frac{2^n - 1}{2 - 1}$$

$$a_n = 2^{n+1} - 1, \quad n \geq 0$$

Αναδρομή

Γραμμικές Αναδρομικές Εξισώσεις με σταθερούς συντελεστές

Η γραμμική εξίσωση τάξης k με σταθερούς συντελεστές είναι της μορφής:

$$u_n + a_1 u_{n-1} + a_2 u_{n-2} + \ldots + a_k u_{n-k} = b(n)$$

με a_i σταθερές. Η εξίσωση

$$u_n + a_1 u_{n-1} + a_2 u_{n-2} + \ldots + a_k u_{n-k} = 0$$

καλείται αντίστοιχη ομογενής εξίσωση της αναδρομικής.
Το σύνολο των λύσεων της ομογενούς εξίσωσης σχηματίζει έναν διανυσματικό χώρο διάστασης μικρότερης είτε ίσης του k και κάποιες λύσεις είναι της μορφής:

$$u_n = r^n.$$

Αναδρομή

Γραμμικές Αναδρομικές Εξισώσεις με σταθερούς συντελεστές

Αν αντικαταστήσουμε στην εξίσωση παίρνουμε την χαρακτηριστική εξίσωση της ομογενούς εξίσωσης:

$$r^k + a_1 r^{k-1} + \ldots + a_k = 0$$

Αν οι k ρίζες της εξίσωσης είναι διαφορετικές $r_1, r_2, \ldots, r_k$ τότε η λύση της ομογενούς εξίσωσης είναι:

$$u_n = \lambda_1 r_1^n + \lambda_2 r_2^n + \ldots + \lambda_k r_k^n$$

με λ_i σταθερές.

Αναδρομή

Παράδειγμα

Να επιλυθεί η παρακάτω αναδρομική εξίσωση:

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2$$
$$F_0 = 0$$
$$F_1 = 1$$

Λύση: Έχουμε ότι:

$$F_n - F_{n-1} - F_{n-2} = 0.$$

Αν $u_n = r^n$ τότε:

$$r^n - r^{n-1} - r^{n-2} = 0 \Rightarrow$$
$$r^{n-2}(r^2 - r - 1) = 0 \Rightarrow$$
$$r^2 - r - 1 = 0.$$

Αναδρομή

... Παράδειγμα

Οι λύσεις είναι:

$$r_1 = \frac{1+\sqrt{5}}{2}$$
$$r_2 = \frac{1-\sqrt{5}}{2}$$

Η λύση της ομογενούς εξίσωσης:

$$F_n = \lambda_1 r_1^n + \lambda_2 r_2^n$$

$$F_0 = 0 \Leftrightarrow \lambda_1 = -\lambda_2$$

$$F_1 = 1 \Leftrightarrow \lambda_1 r_1 + \lambda_2 r_2 = 1 \Leftrightarrow \lambda_1 r_1 - \lambda_1 r_2 = 1$$

Αναδρομή

... Παράδειγμα

Τελικά $\lambda_1 = \frac{1}{r_1 - r_2}$ και $\lambda_2 = \frac{1}{r_2 - r_1}$ δηλαδή

$$\lambda_1 = \frac{1}{\frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2}} = \frac{1}{\sqrt{5}}$$

$$\lambda_2 = -\frac{1}{\sqrt{5}}$$

Τελικά, η λύση είναι:

$$F_n = \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n]$$

Αναδρομή

Εξισώσεις μετατρεπόμενες σε γραμμικές αναδρομές

- Ας υποθέσουμε την αναδρομική εξίσωση

$$T(n) = aT(\frac{n}{b}) + d(n)$$

με $n \geq 2$, a, b σταθερές και $T(1) = 1$.

- Επιπρόσθετα, ας υποθέσουμε ότι $n = b^k$ και άρα

$$T(b^k) = aT(b^{k-1}) + d(b^k).$$

- Εάν θέσουμε $t_k = T(b^k)$:

$$t_k = at_{k-1} + d(b^k), \quad t_0 = 1$$

- Ας θεωρήσουμε, επιπρόσθετα την εξίσωση:

$$a(n)u_n = b(n)u_{n-1} + c(n).$$

Αναδρομή

Εξισώσεις μετατρεπόμενες σε γραμμικές αναδρομές

Αν ισχύει

$$f(n) = \frac{\prod_{i=1}^{n-1} a(i)}{\prod_{i=1}^{n} b(i)}$$

τότε

$$v_n = v_{n-1} + f(n)c(n)$$

με $v_n = a(n)f(n)u_n$ και χρησιμοποιώντας τη μέθοδο των αθροιζομένων παραγόντων θα έχουμε

$$u_n = \frac{1}{a(n)f(n)}(a(0)f(0)u_0 + \sum_{i=1}^{n} f(i)c(i))$$

και άρα

$$t_k = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

Αναδρομή

Εξισώσεις μετατρεπόμενες σε γραμμικές αναδρομές

Τελικά, αφού $k = \log_b n$ και $a^{\log_b n} = n^{\log_b a}$ η αρχική εξίσωση γίνεται

$$T(n) = n^{\log_b a} + \sum_{j=0}^{(\log_b n)-1} a^j d(\frac{n}{b^j}).$$

Αναδρομή

Παράδειγμα

Να επιλυθεί η παρακάτω αναδρομική εξίσωση (πολυπλοκότητα Merge Sort):

$$T(n) = 2T(\frac{n}{2}) + n - 1 \quad n \geq 2$$
$$T(1) = 1$$

Λύση: Έχουμε $a = b = 2$ και $d(n) = n - 1$ άρα

$$T(n) = n + \sum_{j=0}^{k-1} 2^j((\frac{n}{2^j}) - 1)$$
$$= n + kn - (2^k - 1)$$

και επειδή $k = \log_2 n$ τελικά

$$T(n) = n \log_2 n + 1.$$