ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΝΑΛΥΣΗ ΑΛΓΟΡΙΘΜΩΝ

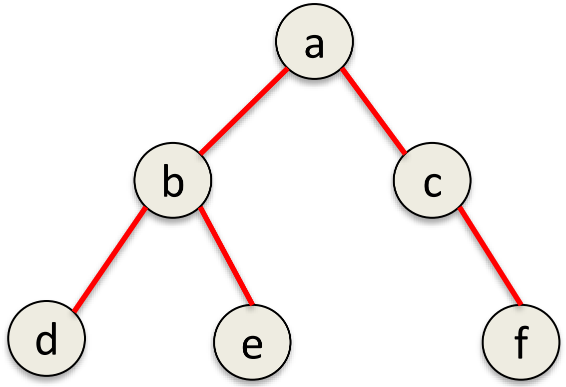
Διαίρει και Βασίλευε

Κωνσταντίνος Γιαννουτάκης
Επίκουρος Καθηγητής

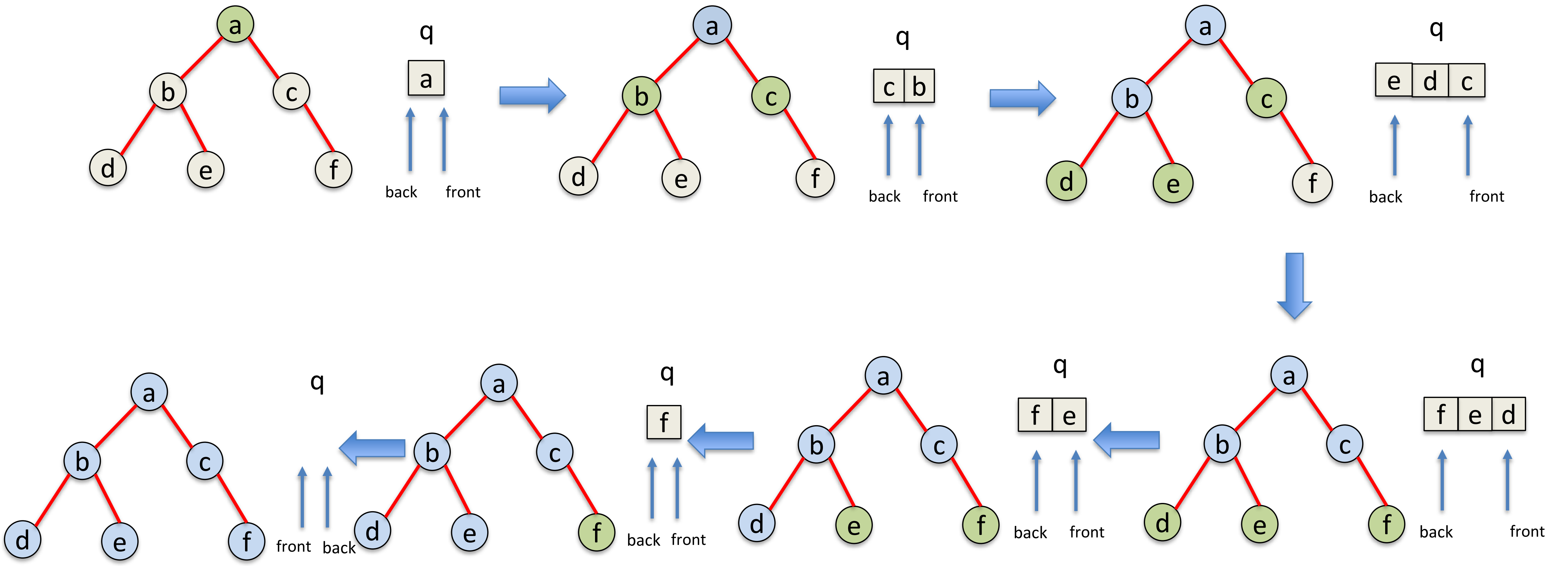
ΑΣΚΗΣΗ ΠΡΟΗΓΟΥΜΕΝΗΣ ΔΙΑΛΕΞΗΣ

- Περιγράψτε έναν αλγόριθμο διαπέρασης ενός δυαδικού δένδρου κατά επίπεδα, με τη χρήση μίας βοηθητικής ουράς.

1. Δημιούργησε μια κενή ουρά q
2. Τοποθέτησε τη ρίζα του δένδρου στην ουρά (push(root))
3. Ενώ η ουρά q δεν είναι κενή (!isEmpty())
    - Τύπωσε το 1^{ο} στοιχείο της ουράς
    - Τοποθέτησε το αριστερό και δεξί παιδί του στοιχείου στην ουρά
    - Εξήγαγε ένα στοιχείο από την ουρά



ΑΣΚΗΣΗ ΠΡΟΗΓΟΥΜΕΝΗΣ ΔΙΑΛΕΞΗΣ



ΣΥΝΟΨΗ ΔΙΑΛΕΞΗΣ

- Διαίρει και Βασίλευε

ΕΙΣΑΓΩΓΗ

ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ/ΚΥΡΙΕΥΕ (DIVIDE AND CONQUER)

Οι αποτελεσματικότεροι αναδρομικοί αλγόριθμοι υλοποιούν την αρχή του διαίρει και βασίλευε:

- **Διαιρούμε** το πρόβλημα σε υποπροβλήματα που είναι μικρότερα στιγμιότυπα του ίδιου προβλήματος.
- **Κυριεύουμε** τα υποπροβλήματα, επιλύοντάς τα αναδρομικά.
- **Συνδυάζουμε** τις λύσεις των υποπροβλημάτων για να συνθέσουμε μια λύση του αρχικού προβλήματος.

Όταν τα προβλήματα θα γίνουν **αρκετά μικρά ώστε να μην επιλύονται αναδρομικά**, τότε **η αναδρομή εξαντλείται** και έχουμε αναχθεί στη **στοιχειώδη περίπτωση**.

ΑΡΧΗ ΤΗΣ ΕΞΙΣΟΡΡΟΠΗΣΗΣ

Γενικός κανόνας: Η διατήρηση μιας ισορροπίας, ή και ισότητας, μεταξύ των μεγεθών $n_1, n_2, \ldots, n_k$ των υποπροβλημάτων $\Pi_1, \Pi_2, \ldots, \Pi_k$ που προκύπτουν από τη διάσπαση ενός προβλήματος Π μεγέθους *n*.

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

ΔΙΑΔΙΚΑΣΙΕΣ ΤΕΧΝΙΚΗΣ ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

1. **Διαίρει** (**divide**): Διαίρεσε το πρόβλημα Π σε k (συνήθως $k = 2$) υποπροβλήματα $\Pi_1, \Pi_2, \dots, \Pi_k$(στιγμιότυπα ίδιου τύπου με το Π, όσο δυνατόν ίδιου μεγέθους και δυσκολίας)
2. **Κατάκτησε** (**conquer**): Επίλυσε αναδρομικά όλα τα υποπροβλήματα $\Pi_1, \Pi_2, \dots, \Pi_k$. Εάν αυτά είναι μικρού μεγέθους, τότε επίλυσέ τα άμεσα.
3. **Συνδύασε** (**combine**): Συνδύασε τις επιμέρους λύσεις των $\Pi_1, \Pi_2, \dots, \Pi_k$ για τη λύση του αρχικού προβλήματος Π.

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ

- Έστω k το πλήθος των υποπροβλημάτων $\Pi_1, \Pi_2, \ldots, \Pi_k$ της αρχικής εισόδου μήκους n.

$D(n)$	το πλήθος βημάτων της	divide
$S(n)$	το πλήθος βημάτων της	conquer
$C(n)$	το πλήθος βημάτων της	combine

Τότε

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

ΑΝΑΔΡΟΜΙΚΟ ΣΧΗΜΑ

Διάσπαση προβλήματος

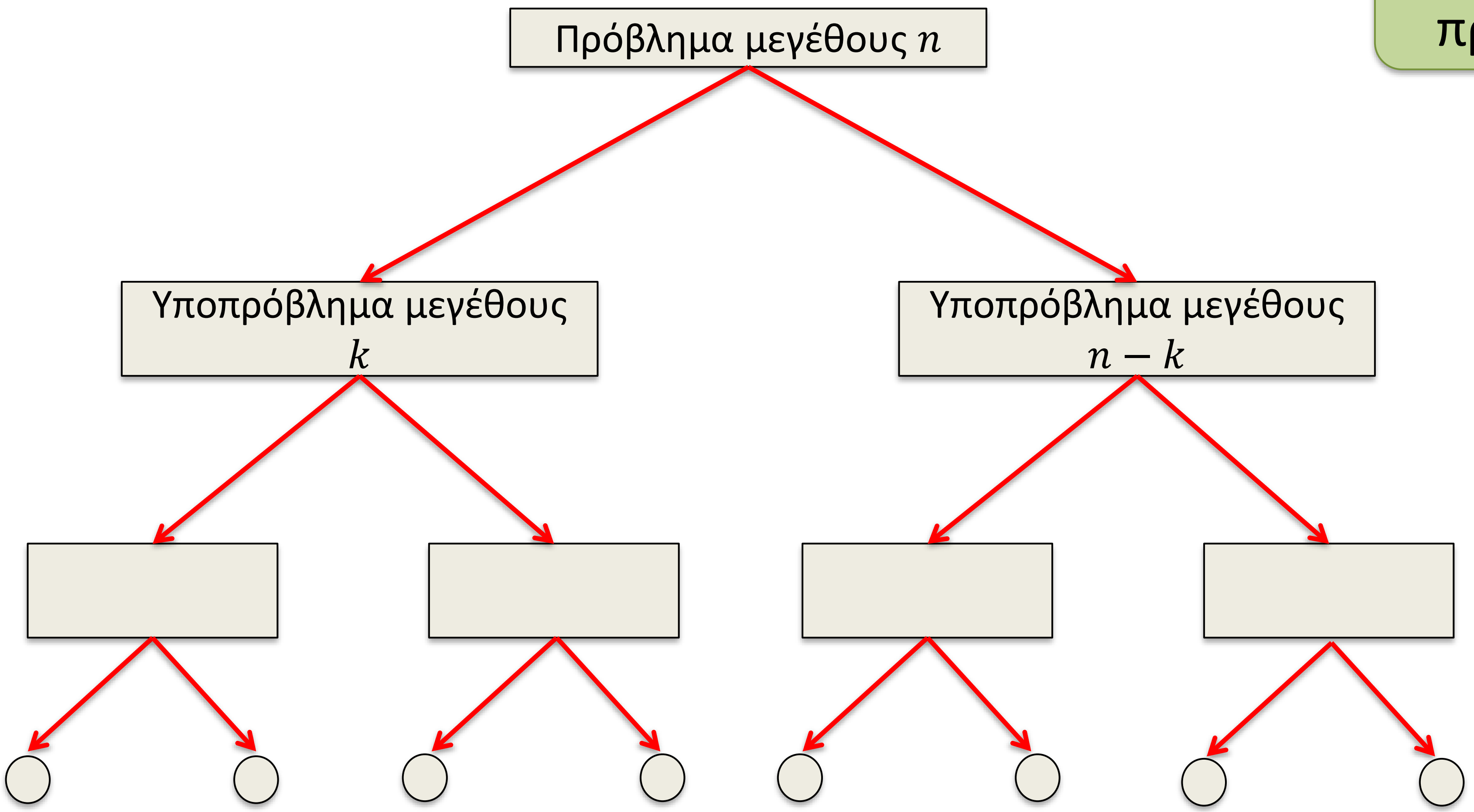
Πρόβλημα μεγέθους n

Υποπρόβλημα μεγέθους k

Υποπρόβλημα μεγέθους $n-k$

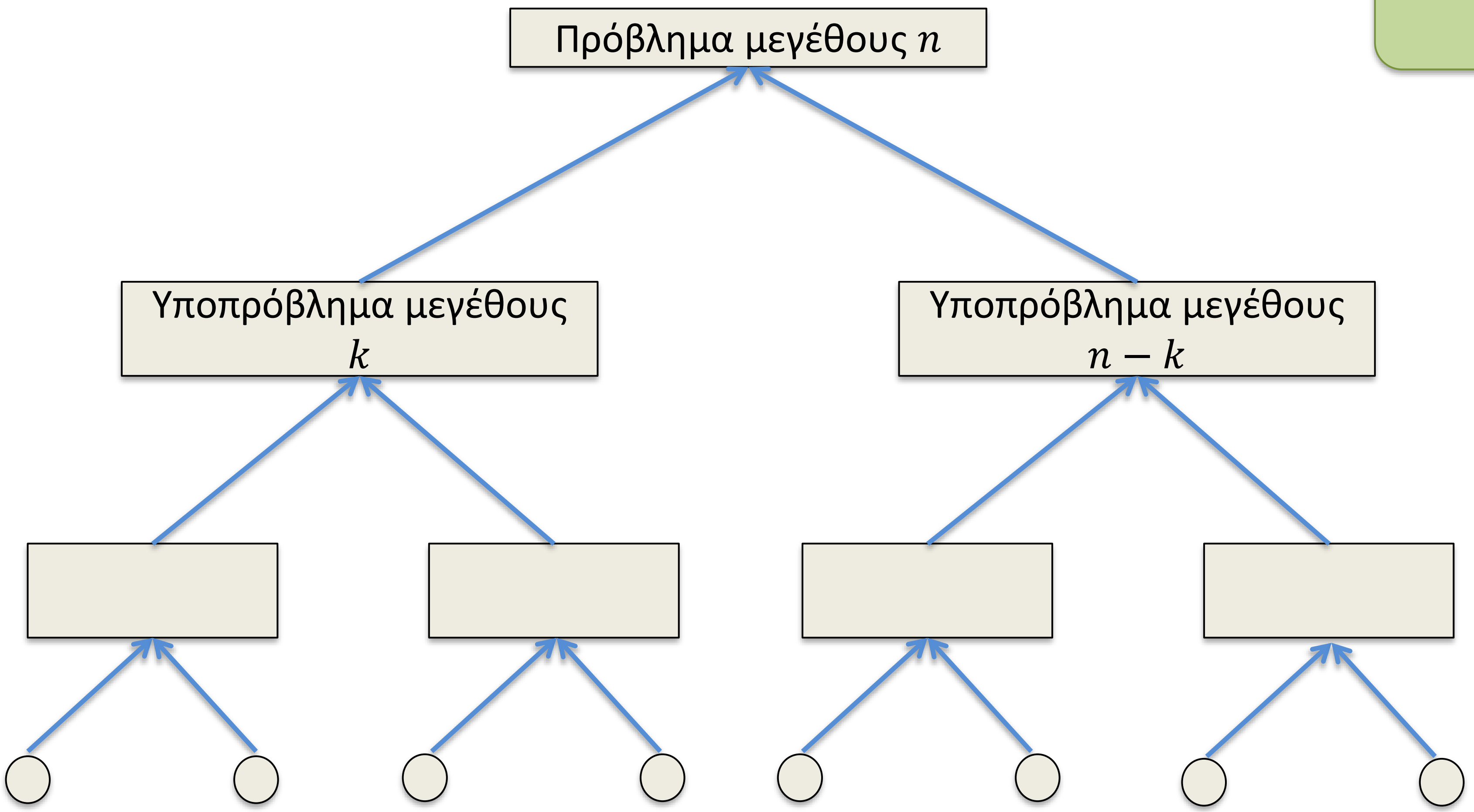
ΑΝΑΔΡΟΜΙΚΟ ΣΧΗΜΑ

Διάσπαση προβλήματος

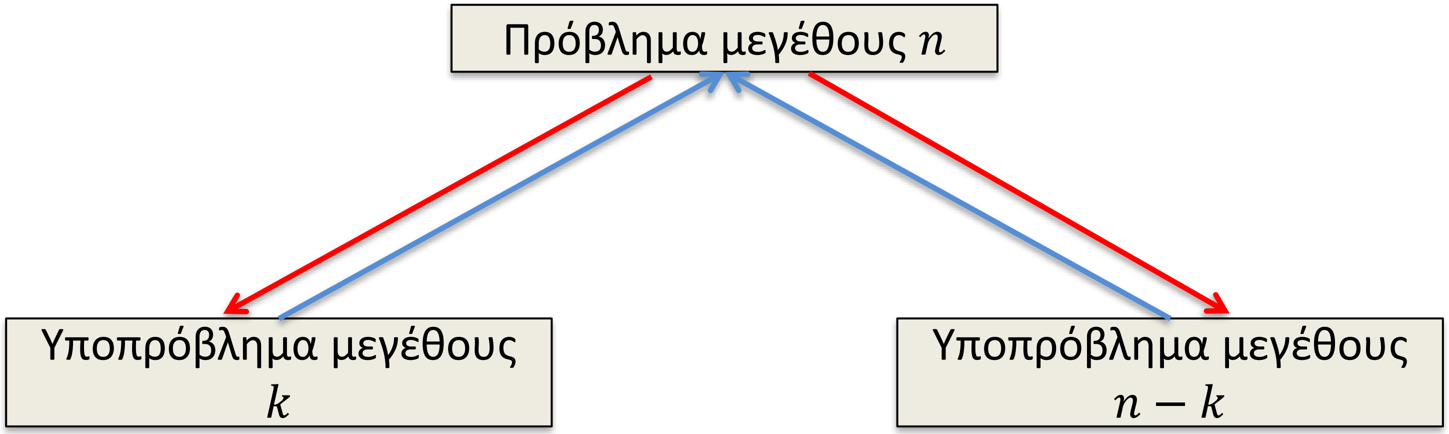


ΑΝΑΔΡΟΜΙΚΟ ΣΧΗΜΑ

Συνδυασμός λύσεων



ΑΝΑΔΡΟΜΙΚΟ ΣΧΗΜΑ



Χρόνος Εκτέλεσης

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

Χρόνος διάσπασης

Χρόνος σύνθεσης

ΥΠΟΛΟΓΙΣΜΟΣ ΔΥΝΑΜΗΣ

ΥΠΟΛΟΓΙΣΜΟΣ ΔΥΝΑΜΗΣ

- **Διαιρούμε** τον εκθέτη n σε $n/2$
- **Κυριεύουμε** τον υπολογισμό της δύναμης **αναδρομικά**
- **Συνδυάζουμε/Συγχωνεύουμε** τις **δύο υπολογισθείσες τιμές** ώστε να υπολογιστεί η σωστή δύναμη

ΚΛΑΣΙΚΟΣ ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ

```
int power(int base, int n) {
    if (n != 0)
        return (base * power(base, n - 1));
    else
        return 1;
}
```

Χρόνος Εκτέλεσης

$$T(n) = \begin{cases} \mathcal{O}(1), & \alpha\nu\ n = 0 \\ T(n-1) + \mathcal{O}(1), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Πολυπλοκότητα

$\mathcal{O}(n)$

ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

```
int power(int base, int n)
{
    if (n == 0)
        return 1;
    else if (n%2 == 0)
        return power(base, n/2)*power(base, n/2);
    else
        return base*power(base, n/2)*power(base, n/2);
}
```

Χρόνος Εκτέλεσης

$$T(n) = \begin{cases} \mathcal{O}(1), & \alpha\nu\ n = 0 \\ 2T\left(\frac{n}{2}\right) + \mathcal{O}(1), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Πολυπλοκότητα

$$\mathcal{O}(n)$$

ΒΕΛΤΙΩΜΕΝΟΣ ΚΩΔΙΚΑΣ

```
int power(int base, int n)
{
    int temp;
    if(n == 0)
        return 1;
    temp = power(base, n/2);
    if (n%2 == 0)
        return temp*temp;
    else
        return base*temp*temp;
}
```

Χρόνος Εκτέλεσης

$$T(n) = \begin{cases} \mathcal{O}(1), & \alpha\nu\ n = 0 \\ T\left(\frac{n}{2}\right) + \mathcal{O}(1), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Πολυπλοκότητα

$$\mathcal{O}(\lg n)$$

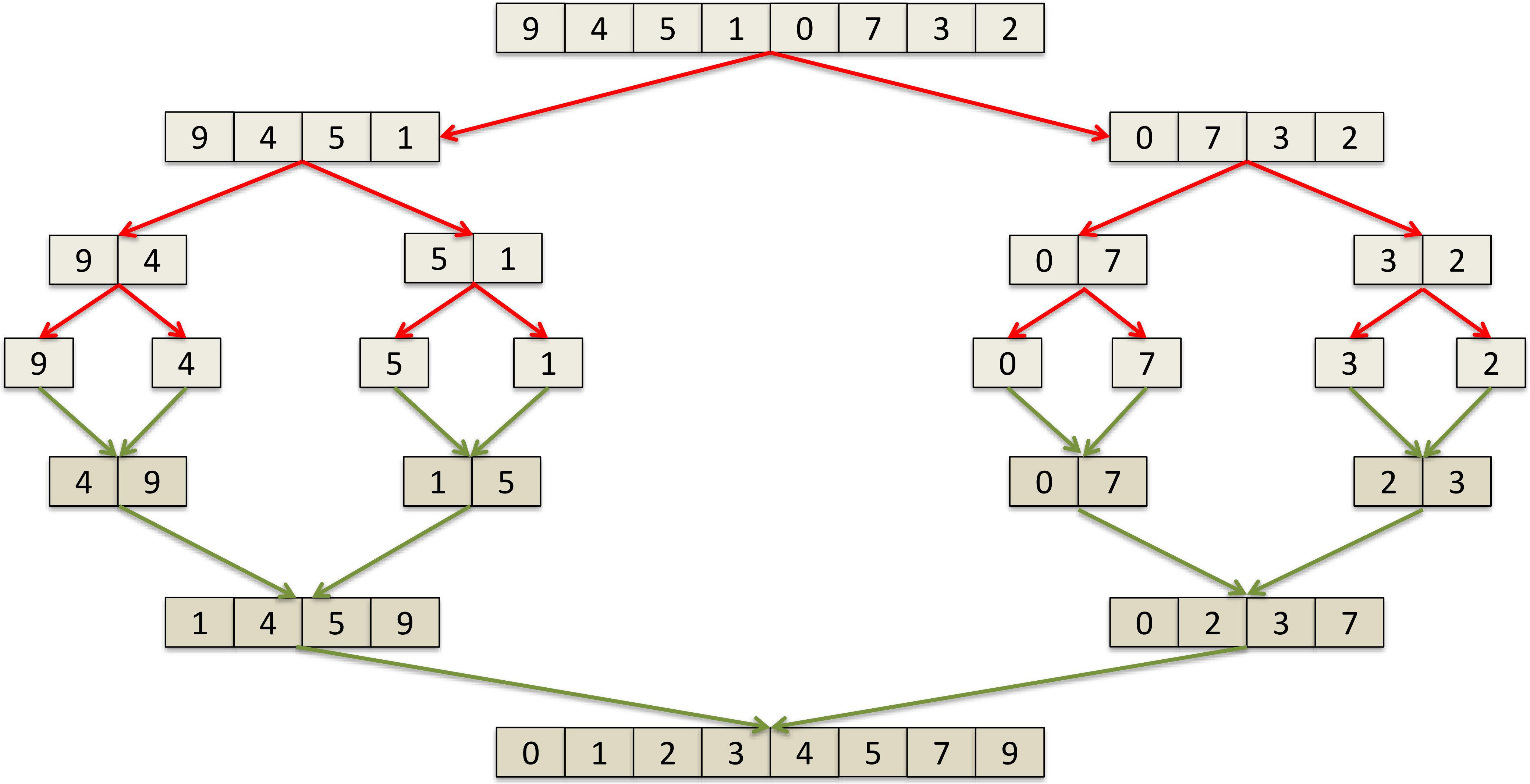
ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΣΥΓΧΩΝΕΥΣΗ

ΤΑΞΙΝΟΜΗΣΗ ΜΕ ΣΥΓΧΩΝΕΥΣΗ (MERGE SORT)

- **Διαιρούμε** την ακολουθία n στοιχείων σε **δύο υπακολουθίες $n/2$ στοιχείων**
- **Κυριεύουμε** τις δύο υπακολουθίες **αναδρομικά**
- **Συνδυάζουμε/Συγχωνεύουμε** τις **δύο ταξινομημένες υπακολουθίες** ώστε να σχηματιστεί η τελική ταξινομημένη ακολουθία

- Η αναδρομή τερματίζει όταν η ταξινομητέα ακολουθία έχει μήκος 1.

ΠΑΡΑΔΕΙΓΜΑ ΔΙΑΙΡΕΣΗΣ ΚΑΙ ΣΥΓΧΩΝΕΥΣΗΣ



ΔΙΑΔΙΚΑΣΙΑ ΣΥΓΧΩΝΕΥΣΗΣ

```
// Συγχώνευση δυο υποπινάκων του arr[]:
// arr[l..m] και arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* Βοηθητικοί πίνακες */
    int L[n1], R[n2];

    /* Αντιγραφή τιμών στους L[] και R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Συγχώνευση των L και M στον[l..r]*/
    i = 0; // Δείκτης για τον πίνακα L
    j = 0; // Δείκτης για τον πίνακα R
    k = l; // Δείκτης για τον πίνακα arr
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Αντιγραφή των στοιχείων του L[] που περίσσεψαν
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Αντιγραφή των στοιχείων του R[] που περίσσεψαν
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Πολυπλοκότητα

ΔΙΑΔΙΚΑΣΙΑ ΣΥΓΧΩΝΕΥΣΗΣ

```
// Συγχώνευση δυο υποπινάκων του arr[]:
// arr[l..m] και arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* Βοηθητικοί πίνακες */
    int L[n1], R[n2];

    /* Αντιγραφή τιμών στους L[] και R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Συγχώνευση των L και M στον[l..r]*/
    i = 0; // Δείκτης για τον πίνακα L
    j = 0; // Δείκτης για τον πίνακα R
    k = l; // Δείκτης για τον πίνακα arr
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Αντιγραφή των στοιχείων του L[] που περίσσεψαν
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Αντιγραφή των στοιχείων του R[] που περίσσεψαν
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

n1 + n2 επαναλήψεις

1 σύγκριση

n1 επαναλήψεις

n2 επαναλήψεις

Πολυπλοκότητα

$\mathcal{O}(\boldsymbol{n1} + \boldsymbol{n2}) = \mathcal{O}(\boldsymbol{n})$

ΔΙΑΔΙΚΑΣΙΑ ΔΙΑΙΡΕΣΗΣ

```
/* l είναι ο αριστερός δείκτης και r ο δεξιός δείκτης του υποπίνακα του arr */
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = (r + l) / 2;

        // Ταξινόμησε αναδρομικά τον πρώτο και δεύτερο υποπίνακα
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}
```

Πολυπλοκότητα

ΔΙΑΔΙΚΑΣΙΑ ΔΙΑΙΡΕΣΗΣ

```
/* l είναι ο αριστερός δείκτης και r ο δεξιός δείκτης του υποπίνακα του arr */
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        int m = (r + l) / 2;

        // Ταξινόμησε αναδρομικά τον πρώτο και δεύτερο υποπίνακα
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}
```

Χρόνος Εκτέλεσης Merge Sort

$$T(n) = \begin{cases} 1, & \alpha\nu\ n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Πολυπλοκότητα

$$O(n\lg n)$$

ΠΛΕΟΝΕΚΤΗΜΑΤΑ/ΜΕΙΟΝΕΚΤΗΜΑΤΑ

Πλεονεκτήματα

- Πολυπλοκότητα $\mathcal{O}(n\lg n)$ ανεξαρτήτως εισόδου

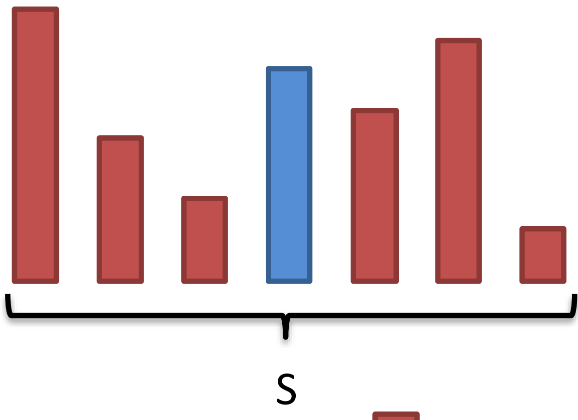
Μειονεκτήματα

- Απαιτείται πρόσθετος χώρος στη μνήμη $\mathcal{O}(n)$

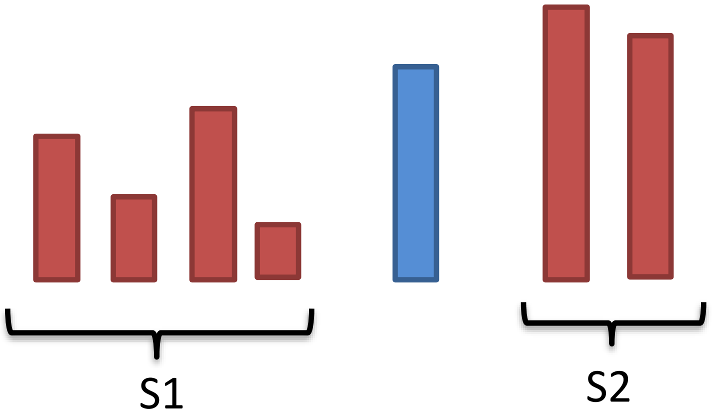
ΑΛΓΟΡΙΘΜΟΣ ΓΡΗΓΟΡΗΣ ΤΑΞΙΝΟΜΗΣΗΣ

ΓΡΗΓΟΡΗ ΤΑΞΙΝΟΜΗΣΗ (QUICK SORT)

1. **Διαίρει** (**divide**): Διάλεξε ένα στοιχείο pivot (άξονα) του πίνακα. Με βάση αυτό, χώρισε στα δύο τον πίνακα: στον αριστερό υποπίνακα βάλε όλα τα μικρότερα στοιχεία και στο δεξιό όλα τα μεγαλύτερα στοιχεία από το pivot.



2. **Κατάκτησε** (**conquer**): Αναδρομικά κάνε το ίδιο και στους δύο υποπίνακες.
3. **Συνδύασε** (**combine**): Βάλε στη σειρά τον αριστερό υποπίνακα, το pivot και τον δεξιό υποπίνακα.



ΚΩΔΙΚΑΣ C

```c
void quickSort(int a[], int left, int right)
{
   int j;
   if (left < right) {
     // divide and conquer
     j = partition(a, left, right);
     quickSort(a, left, j-1);
     quickSort(a, j+1, right);
   }
}
```

ΚΩΔΙΚΑΣ C

```c
int partition(int a[], int left, int right)
{
   int pivot, i, j, t;
   pivot = a[left];
   i = left; j = right+1;
   while (1){
     do ++i;
     while (a[i]<=pivot && i<j);
     do --j;
     while (a[j] > pivot);
     if(i >= j) break;
     t=a[i]; a[i]=a[j]; a[j]=t;
   }
   t=a[left]; a[left]=a[j]; a[j]=t;
   return j;
}
```

ΠΑΡΑΔΕΙΓΜΑ

```c
int partition(int a[], int left, int right)
{
   int pivot, i, j, t;
   pivot = a[left];
   i = left; j = right+1;
   while (1){
     do ++i;
     while (a[i]<=pivot && i<j);
     do --j;
     while (a[j] > pivot);
     if(i >= j) break;
     t=a[i]; a[i]=a[j]; a[j]=t;
   }
   t=a[left]; a[left]=a[j]; a[j]=t;
   return j;
}
```

23	42	10	100	8	4	12	56	57	50	
pivot i										j

23	42	10	100	8	4	12	56	57	50
pivot	i					j			

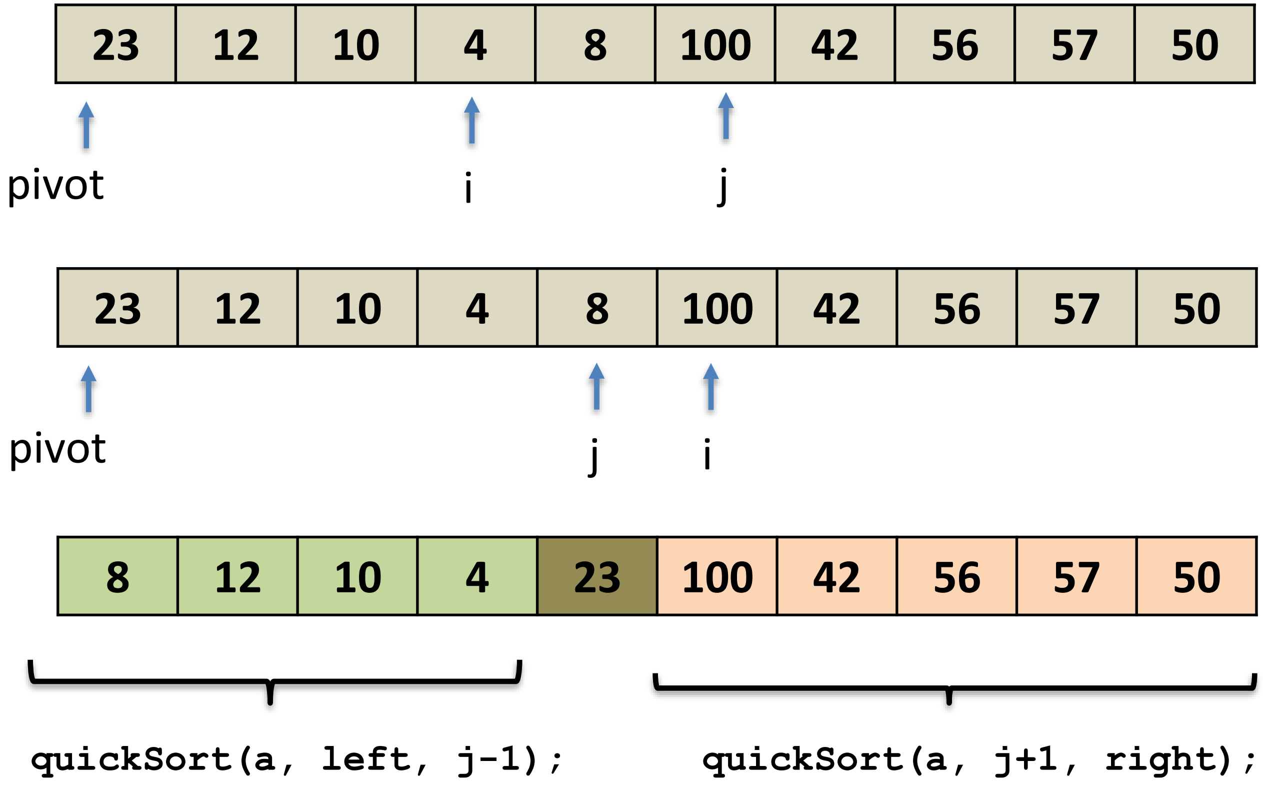
23	12	10	100	8	4	42	56	57	50
pivot	i					j			

23	12	10	100	8	4	42	56	57	50
pivot			i		j				

23	12	10	4	8	100	42	56	57	50
pivot			i		j				

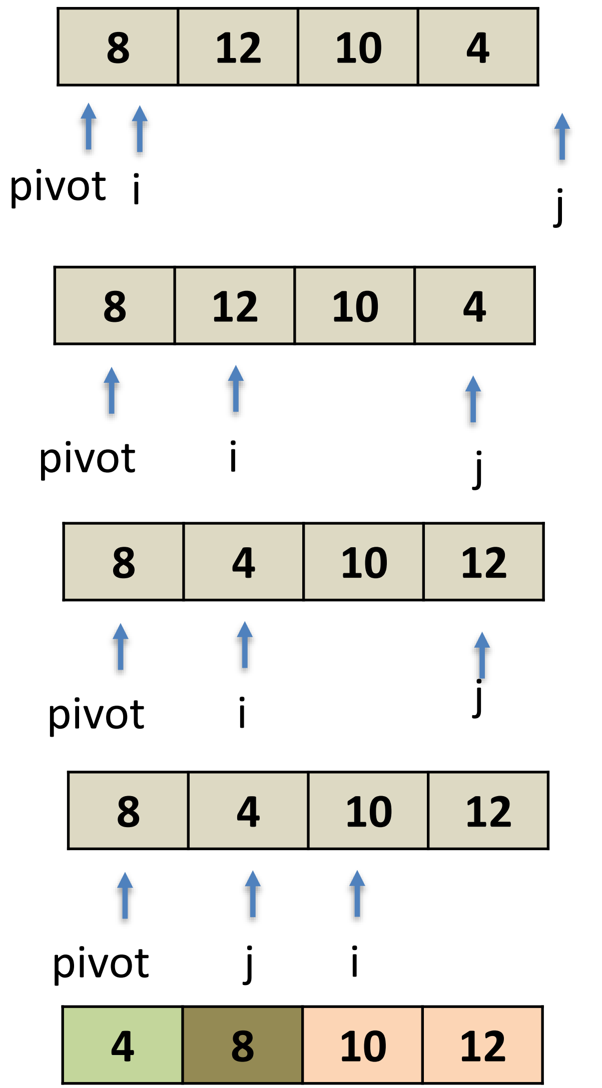
ΠΑΡΑΔΕΙΓΜΑ

```c
int partition(int a[], int left, int right)
{
    int pivot, i, j, t;
    pivot = a[left];
    i = left; j = right+1;
    while (1){
      do ++i;
      while (a[i]<=pivot && i<j);
      do --j;
      while (a[j] > pivot);
      if(i >= j) break;
      t=a[i]; a[i]=a[j]; a[j]=t;
    }
    t=a[left]; a[left]=a[j]; a[j]=t;
    return j;
}
```



ΠΑΡΑΔΕΙΓΜΑ

```
int partition(int a[], int left, int right)
{
   int pivot, i, j, t;
   pivot = a[left];
   i = left; j = right+1;
   while (1){
     do ++i;
     while (a[i]<=pivot && i<j);
     do --j;
     while (a[j] > pivot);
     if(i >= j) break;
     t=a[i]; a[i]=a[j]; a[j]=t;
   }
   t=a[left]; a[left]=a[j]; a[j]=t;
   return j;
}
```



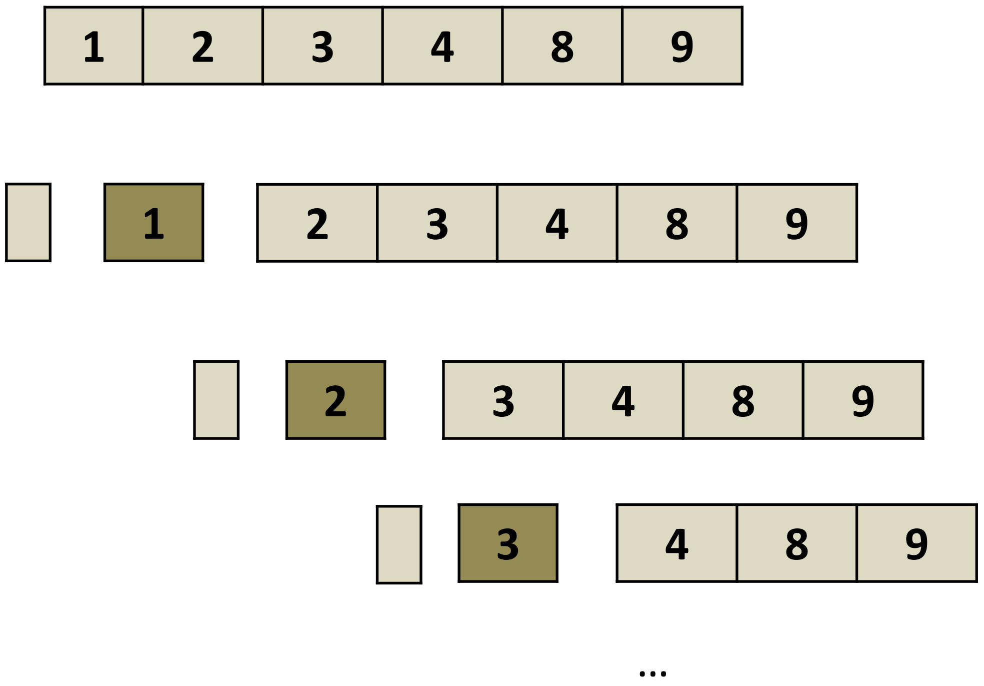
ΠΟΛΥΠΛΟΚΟΤΗΤΑ

- Στη διαδικασία **partition** το στοιχείο **pivot** συγκρίνεται με κάθε στοιχείο της λίστας
- Σύνολο $n + 1$ συγκρίσεις

```
int partition(int a[], int left, int right)
{
   int pivot, i, j, t;
   pivot = a[left];
   i = left; j = right+1;
   while (1){
     do ++i;
     while (a[i]<=pivot && i<j);
     do --j;
     while (a[j] > pivot);
     if(i >= j) break;
     t=a[i]; a[i]=a[j]; a[j]=t;
   }
   t=a[left]; a[left]=a[j]; a[j]=t;
   return j;
}
```

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

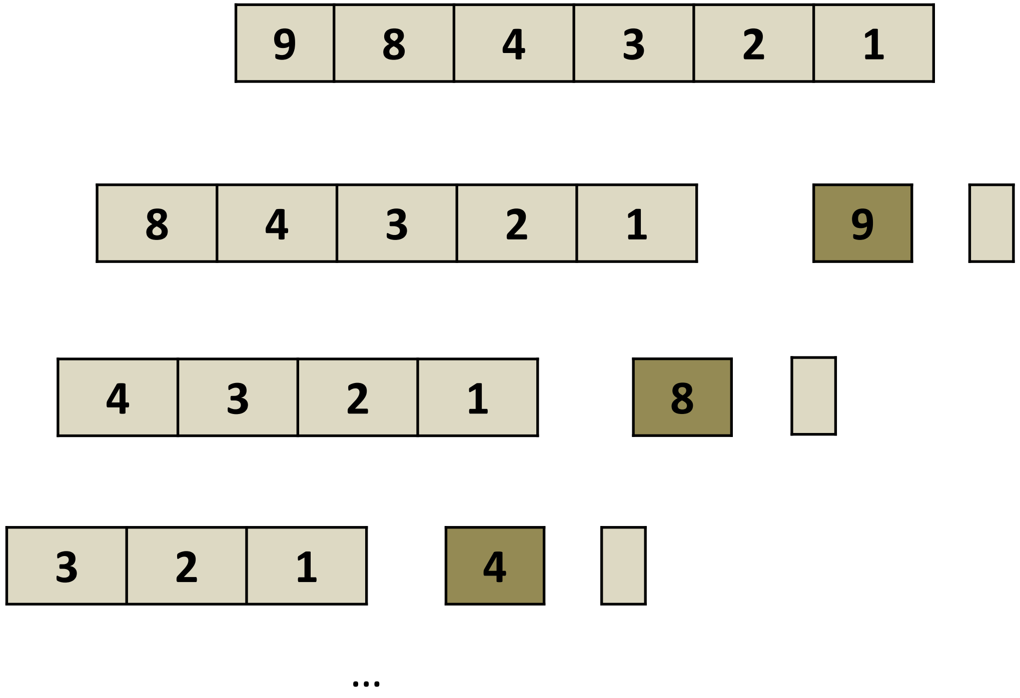
- Χειρότερη περίπτωση
  - Εάν το pivot=a[left] είναι το **μικρότερο** στοιχείο, τότε η λίστα θα διαχωριστεί στο $1^{ο}$ της στοιχείο (η $1^{η}$ αναδρομή δεν θα εκτελεστεί)




```
void quickSort(int a[], int left, int right)
{
    int j;
    if (left < right) {
      // divide and conquer
      j = partition(a, left, right);
      quickSort(a, left, j-1);
      quickSort(a, j+1, right);
    }
}
```

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

- Χειρότερη περίπτωση
  - Εάν το pivot=a[left] είναι το **μεγαλύτερο** στοιχείο, τότε η λίστα θα διαχωριστεί στο 1ο της στοιχείο (η 2η αναδρομή δεν θα εκτελεστεί)




```
void quickSort(int a[], int left, int right)
{
    int j;
    if (left < right) {
      // divide and conquer
      j = partition(a, left, right);
      quickSort(a, left, j-1);
      quickSort(a, j+1, right);
    }
}
```

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

- Ο χρόνος εκτέλεσης για τη χειρότερη περίπτωση είναι

$$T(n) = \begin{cases} 1, & \alpha\nu\ n = 2 \\ T(n-1) + (n+1), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

- Με τη μέθοδο της επανάληψης, έχουμε

$$T(n) = T(n-1) + (n+1) = T(n-2) + (n-1+1) + (n+1) = T(n-2) + n + (n+1)$$
$$= T(n-3) + (n-2+1) + n + (n+1) = T(n-3) + (n-1) + n + (n+1) = \cdots$$
$$= 1 + 4 + 5 + \cdots + (n-1) + n + (n+1) = \frac{(n+1)(n+2)}{2} - 5$$

- Άρα πολυπλοκότητα $\boldsymbol{O}(\boldsymbol{n}^2)$

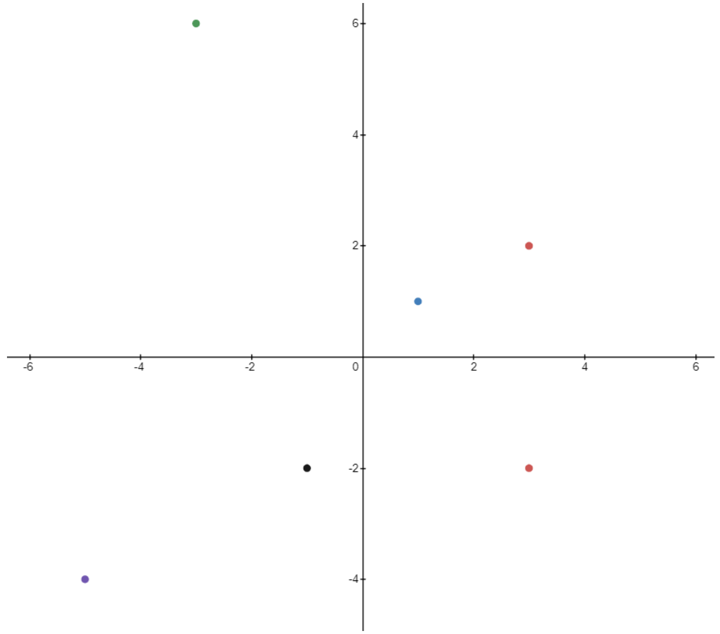
ΠΟΛΥΠΛΟΚΟΤΗΤΑ

- Καλύτερη περίπτωση
  - Συμβαίνει όταν ως pivot επιλέγεται το στοιχείο που υποδιαιρεί τη λίστα σε δύο λίστες μεγέθους $\frac{n}{2}$ (έστω η πρώτη λίστα έχει $\left\lfloor\frac{n}{2}\right\rfloor$ στοιχεία και η δεύτερη $\left\lceil\frac{n}{2}\right\rceil - 1$ στοιχεία).
  - Τότε
$$T(n) = \begin{cases} 1, & \alpha\nu\ n = 2 \\ 2T\left(\frac{n}{2}\right) + O(n), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$
  - Από κύριο θεώρημα, έχουμε $a = 2, b = 2$ και $f(n) = n$. Έχουμε $n^{log_b \alpha} = n^{log_2 2} = n$, άρα είμαστε στη 2η περίπτωση με $\boldsymbol{T(n) = \Theta(n \lg n)}$

ΕΛΑΧΙΣΤΗ ΑΠΟΣΤΑΣΗ ΣΗΜΕΙΩΝ

Κοντινότερο Ζεύγος Σημείων

Να βρεθούν τα δυο πλησιέστερα σημεία μεταξύ n σημείων στον k-διάστατο χώρο

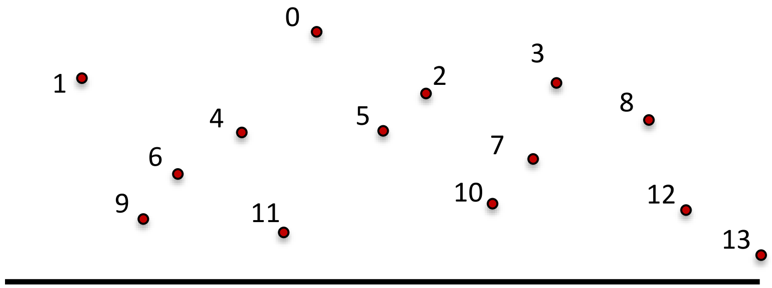


Ο αλγόριθμος **ωμής βίας** απαιτεί $\boldsymbol{O}(\boldsymbol{n}^2)$ συγκρίσεις

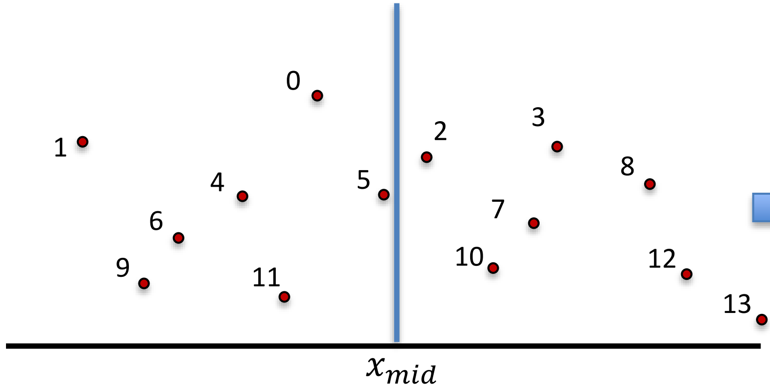
ΕΛΑΧΙΣΤΗ ΑΠΟΣΤΑΣΗ ΣΗΜΕΙΩΝ ΜΕ ΤΕΧΝΙΚΗ ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

1. Ταξινόμησε όλα τα σημεία ως προς την x συντεταγμένη.
2. Χώρισε το σύνολο των σημείων σε δύο υποσύνολα ίσου μεγέθους με μια κάθετη γραμμή x_{mid}.
3. Λύσε το πρόβλημα αναδρομικά στα αριστερά και δεξιά υποσύνολα. Αυτό αποδίδει τις ελάχιστες αποστάσεις στην αριστερή και δεξιά πλευρά d_{Lmin} και d_{Rmin} αντίστοιχα. Θεωρούμε $d = \min(d_{Lmin}, d_{Rmin})$.
4. Βρες τα σημεία που βρίσκονται αριστερά και δεξιά της κάθετης γραμμής x_{mid} και των οποίων η απόσταση από το x_{mid} είναι μικρότερη από το d.
5. Βρες την ελάχιστη απόσταση d_{LRmin} μεταξύ του συνόλου ζευγών σημείων στα οποία το ένα σημείο βρίσκεται στα αριστερά και το άλλο στα δεξιά της x_{mid}.
6. Βρες την ελάχιστη απόσταση μεταξύ των d_{Lmin}, d_{Rmin} και d_{LRmin}.

ΕΛΑΧΙΣΤΗ ΑΠΟΣΤΑΣΗ ΣΗΜΕΙΩΝ

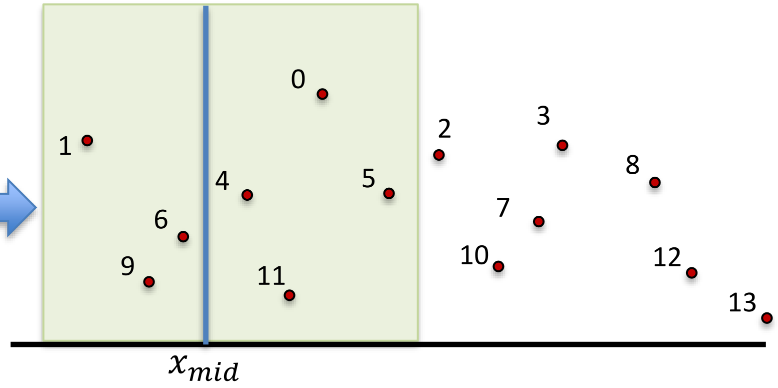


arr[]={1,9,6,4,11,0,5,2,10,7,3,8,12,13}



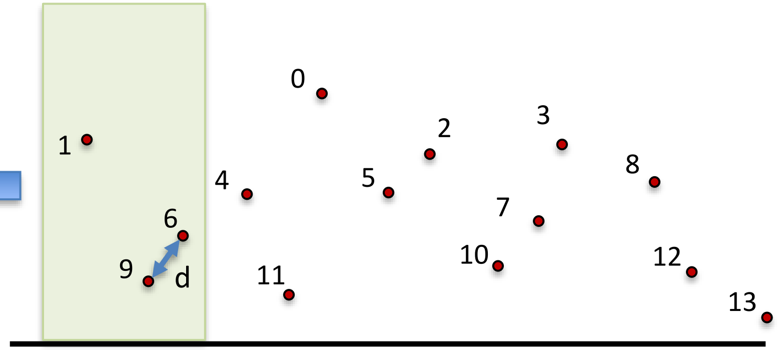
P_L[]={1,9,6,4,11,0,5}

P_R[]={2,10,7,3,8,12,13}

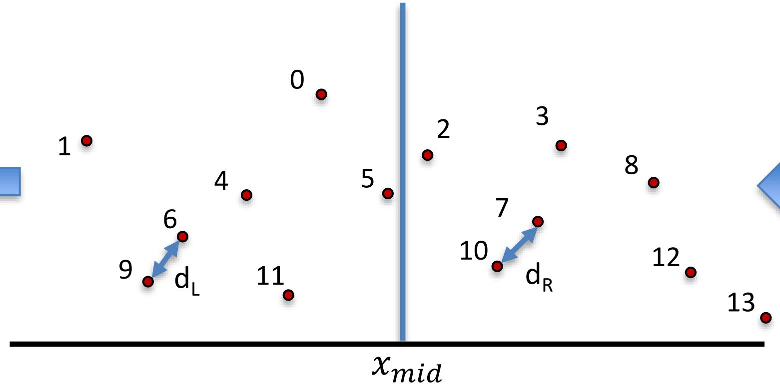


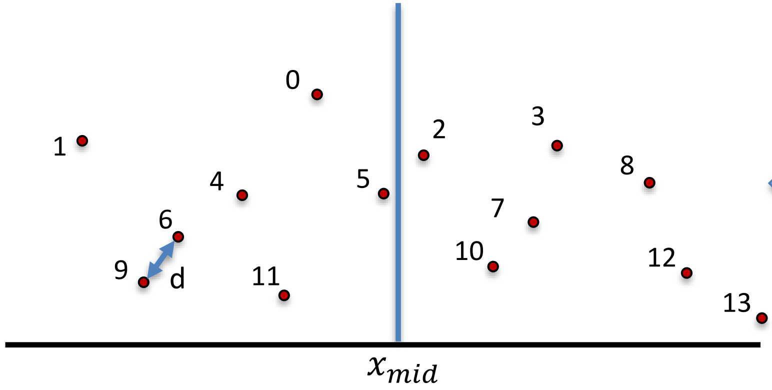
P_L[]={1,9,6}

P_R[]={4,11,0,5}

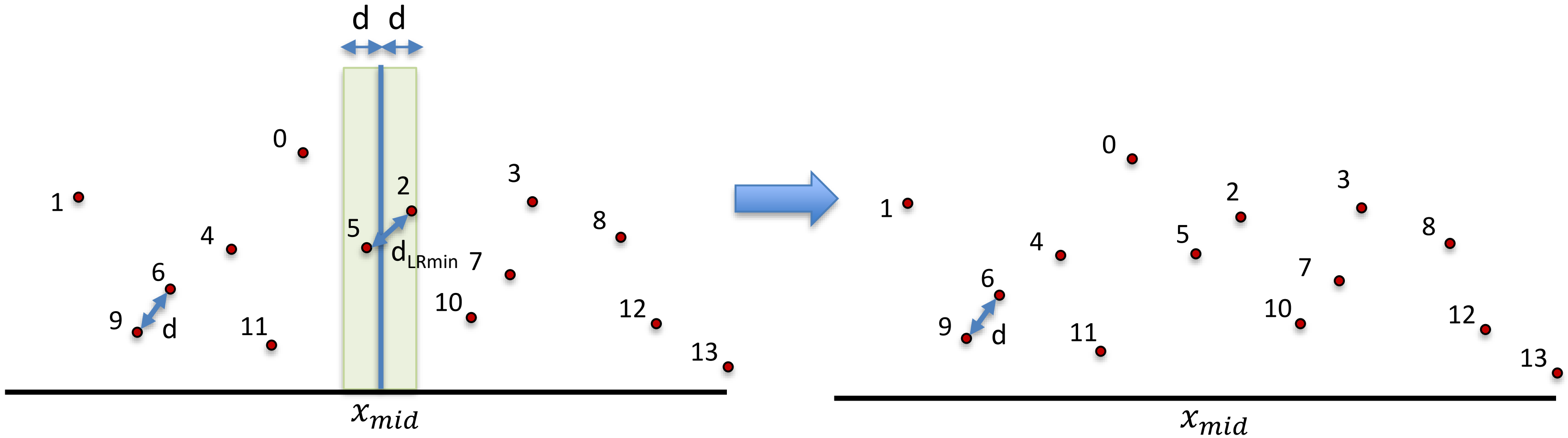


arr[]={1,9,6}





ΕΛΑΧΙΣΤΗ ΑΠΟΣΤΑΣΗ ΣΗΜΕΙΩΝ



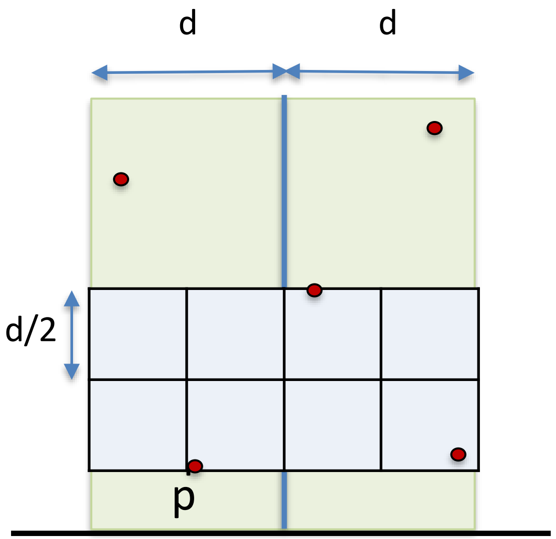
ΚΩΔΙΚΑΣ C

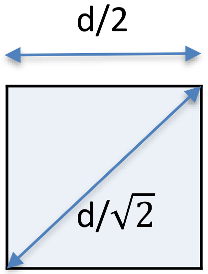
```c
float nearestFunc(Point arr[], int n)  {// Έστω ότι ο arr είναι ταξινομημένος ως προς x
    if (n <= 3) // Εάν έχουμε μόνο 2 ή 3 σημεία χρησιμοποίησε τη μέθοδο ωμής βίας
        return bruteForce(arr, n);
    int mid = n/2;  // Εύρεση μεσαίου σημείου Xmid
    Point midPoint = arr[mid];
    // Υπολογισμός του dl στην αριστερή πλευρά του μεσαίου σημείου και του dr στη δεξιά
    float dl = nearestFunc(arr, mid);   // Αναδρομική κλήση της nearestFunc
    float dr = nearestFunc(arr + mid, n - mid); // Αναδρομική κλήση της nearestFunc
    float d = min(dl, dr);  // Εύρεση της μικρότερης απόστασης μεταξύ των 2 (dl, dr)
    // Δημιουργία ενός πίνακα που περιέχει τα σημεία που απέχουν λιγότερο από d από τη γραμμή
    // που διέρχεται από το Xmid
    Point strip[n];  int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(arr[i].x - midPoint.x) < d) {
            strip[j] = arr[i]; j++;
        }
    return min(d, stripNearest(strip, j, d) );
}
```

Η **stripNearest** υπολογίζει την ελάχιστη απόσταση μεταξύ των στοιχείων των δύο λωρίδων πλάτους d. Με αλγόριθμο ωμής βίας αυτό θα απαιτούσε $\mathcal{O}(n^2)$ **συγκρίσεις.**

ΕΥΡΕΣΗ ΕΛΑΧΙΣΤΗΣ ΑΠΟΣΤΑΣΗΣ ΣΤΙΣ ΔΥΟ ΛΩΡΙΔΕΣ

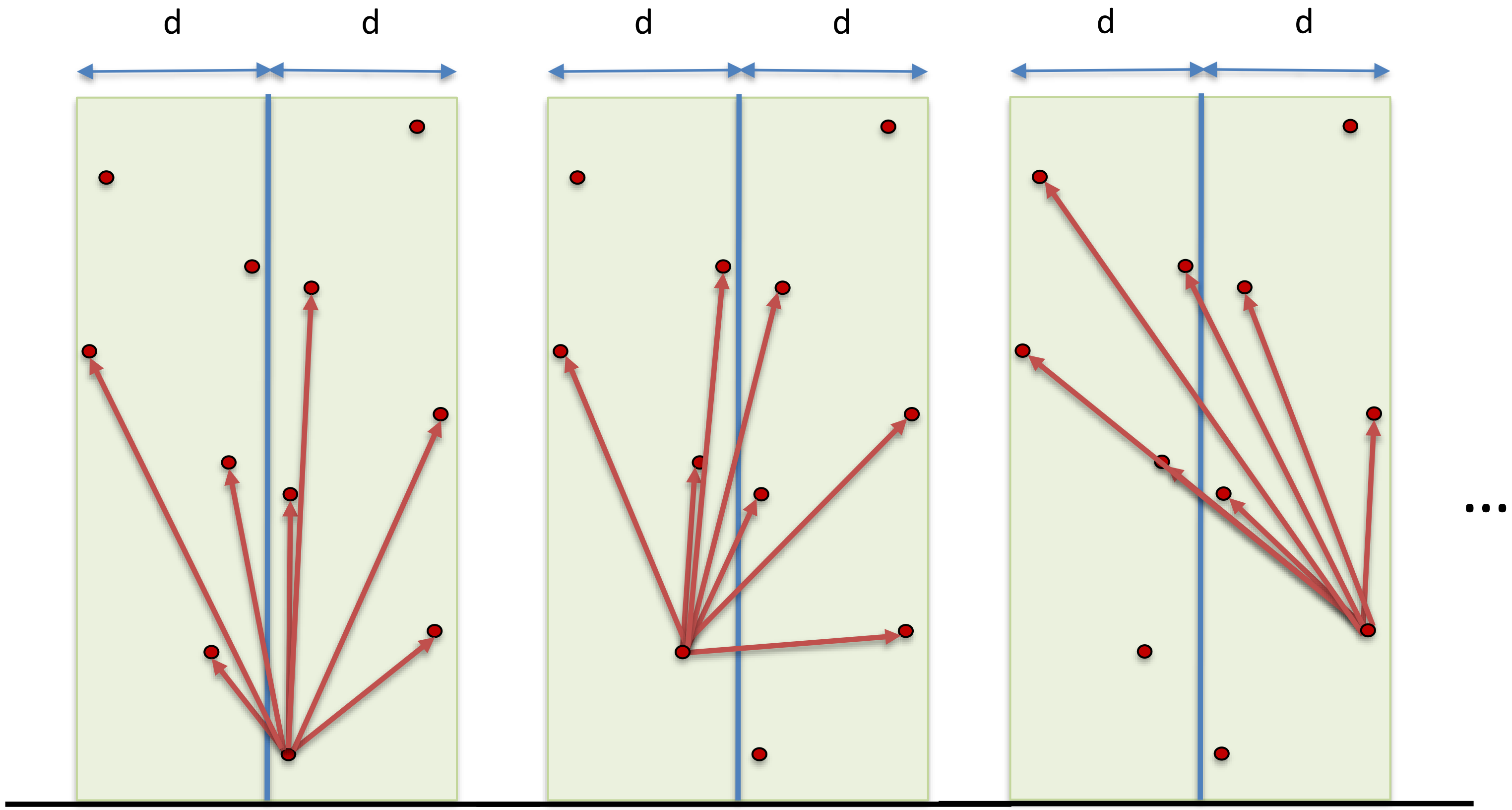
- **Ταξινομούμε** όλα τα στοιχεία ως προς y.
- Τα πιθανά σημεία που μπορούν να έχουν από το σημείο p μικρότερη απόσταση από d, βρίσκονται **μέσα στο ορθογώνιο**.
- Το κάθε τετράγωνο μπορεί να έχει **το πολύ 1 σημείο** (γιατί?).
- Επομένως, θα πρέπει να ελέγξουμε το πολύ **7 σημεία**.
- Συνεχίζουμε την ίδια διαδικασία για τα επόμενα σημεία.





Γιατί το κάθε τετράγωνο μπορεί να περιέχει το πολύ 1 σημείο?
Έστω ότι περιέχει 2 σημεία. Η μέγιστη απόσταση που μπορούν να έχουν αυτά τα σημεία, είναι η απόσταση της διαγωνίου του τετραγώνου, δηλαδή $\frac{d}{\sqrt{2}}$. Αυτό είναι άτοπο, διότι το κάθε τετράγωνο βρίσκεται σε μία μόνο λωρίδα, το οποίο σημαίνει ότι η μέγιστη απόσταση μεταξύ δύο σημείων είναι d και προφανώς $d > \frac{d}{\sqrt{2}}$.

ΕΥΡΕΣΗ ΕΛΑΧΙΣΤΗΣ ΑΠΟΣΤΑΣΗΣ ΣΤΙΣ ΔΥΟ ΛΩΡΙΔΕΣ



1ο Βήμα
7 υπολογισμοί απόστασης και 6 συγκρίσεις

2ο Βήμα
7 υπολογισμοί απόστασης και 6 συγκρίσεις

3ο Βήμα
7 υπολογισμοί απόστασης και 6 συγκρίσεις

Έστω ότι έχουμε $\boldsymbol{m}(\leq \boldsymbol{n})$ στοιχεία στις δύο λωρίδες. Τότε θα χρειαστούμε συνολικά το πολύ

$$\sum_{i=1}^{m} 6 = 6m = \boldsymbol{O}(\boldsymbol{n})$$

συγκρίσεις και

$$\sum_{i=1}^{m} 7 = 7m = \boldsymbol{O}(\boldsymbol{n})$$

υπολογισμούς απόστασης

ΑΡΙΘΜΟΣ ΣΥΓΚΡΙΣΕΩΝ

1 σύγκριση

```
float nearestFunc(Point arr[], int n)  {// Έστω ότι ο arr είναι ταξινομημένος ως προς x
    if (n <= 3) // Εάν έχουμε μόνο 2 ή 3 σημεία χρησιμοποίησε τη μέθοδο ωμής βίας
        return bruteForce(arr, n);
    int mid = n/2;  // Εύρεση μεσαίου σημείου Xmid
    Point midPoint = arr[mid];
    // Υπολογισμός του dl στην αριστερή πλευρά του μεσαίου σημείου και του dr στη δεξιά
    float dl = nearestFunc(arr, mid);   // Αναδρομική κλήση της nearestFunc
    float dr = nearestFunc(arr + mid, n - mid); // ... λήση της nearestFunc
    float d = min(dl, dr); // Εύρεση της μικρότερ... μεταξύ των 2 (dl, dr)
    // Δημιουργία ενός πίνακα που περιέχει τα σημεία που απέχουν λιγότερο από d από τη γραμμή
    // που διέρχεται από το Xmid
    Point strip[n];  int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(arr[i].x - midPoint.x) < d) {
            strip[j] = arr[i]; j++;
        }
    return min(d, stripNearest(strip, j, d) );
}
```

9 πολ/σμοί και 2 συγκρίσεις

1 σύγκριση

1 σύγκριση

$\mathcal{O}(n)$+1 συγκρίσεις

Χρόνος εκτέλεσης (συγκρίσεις)

$$T(n) = \begin{cases} 3, & \alpha\nu\ n \leq 3 \\ 2T\left(\frac{n}{2}\right) + \mathcal{O}(n), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Από το κύριο θεώρημα, προκύπτει εύκολα ότι η πολυπλοκότητα είναι

$\boldsymbol{\mathcal{O}(n \lg n)}$

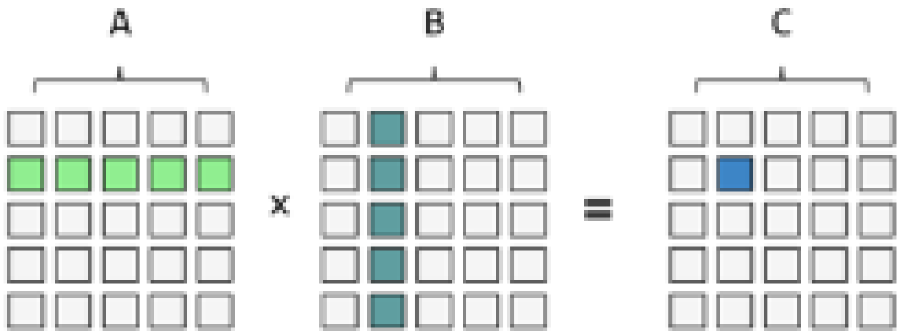
ΣΥΓΚΡΙΣΗ ΑΠΟΔΟΣΗΣ

n	$\lg n$	n^2	$n \lg n$
16	4	256	64
1,024	10	1,048,576	10,240
2,048	11	4,194,304	22,528
4,096	12	16,777,216	49,152
1,048,576	20	1,099,511,627,776	20,971,520
33,554,432	25	1,125,899,906,842,624	838,860,800

ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΣ ΠΙΝΑΚΩΝ

Εάν $A, B \in M_{n,n}$, τότε ορίζουμε το γινόμενό τους ως

$$C \in M_{n,n}\colon\ c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$



Κλασικός αλγόριθμος πολλαπλασιασμού πινάκων
$\boldsymbol{\mathcal{O}(n^3)}$ πολλαπλασιασμοί και $\boldsymbol{\mathcal{O}(n^3)}$ προσθέσεις

ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

- Διαμερίζουμε τους πίνακες A, B, C σε τέσσερις υπο-πίνακες (διάστασης $\frac{n}{2} \times \frac{n}{2}$):

$$A = \left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right] \qquad B = \left[\begin{array}{c|c} B_{1,1} & B_{1,2} \\ \hline B_{2,1} & B_{2,2} \end{array}\right] \qquad C = \left[\begin{array}{c|c} C_{1,1} & C_{1,2} \\ \hline C_{2,1} & C_{2,2} \end{array}\right]$$

- Υπολογίζουμε αναδρομικά τις τιμές

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix}$$

ή πιο απλά

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

ΔΙΑΙΡΕΙ ΚΑΙ ΒΑΣΙΛΕΥΕ

$$\begin{array}{cc} a & b \\ \multicolumn{2}{c}{\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} & a_{2,8} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & a_{3,8} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & a_{6,7} & a_{6,8} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} & a_{7,8} \\ a_{8,1} & a_{8,2} & a_{8,3} & a_{8,4} & a_{8,5} & a_{8,6} & a_{8,7} & a_{8,8} \end{bmatrix}} \\ c & d \end{array} \times \begin{array}{cc} e & f \\ \multicolumn{2}{c}{\begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} & b_{1,6} & b_{1,7} & b_{1,8} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} & b_{2,6} & b_{2,7} & b_{2,8} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} & b_{3,6} & b_{3,7} & b_{3,8} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} & b_{4,6} & b_{4,7} & b_{4,8} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} & b_{5,6} & b_{5,7} & b_{5,8} \\ b_{6,1} & b_{6,2} & b_{6,3} & b_{6,4} & b_{6,5} & b_{6,6} & b_{6,7} & b_{6,8} \\ b_{7,1} & b_{7,2} & b_{7,3} & b_{7,4} & b_{7,5} & b_{7,6} & b_{7,7} & b_{7,8} \\ b_{8,1} & b_{8,2} & b_{8,3} & b_{8,4} & b_{8,5} & b_{8,6} & b_{8,7} & b_{8,8} \end{bmatrix}} \\ g & h \end{array} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

$$\begin{array}{cc} a & b \\ \multicolumn{2}{c}{\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{3,2} & a_{4,3} & a_{4,4} \end{bmatrix}} \\ c & d \end{array} \times \begin{array}{cc} e & f \\ \multicolumn{2}{c}{\begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{bmatrix}} \\ g & h \end{array} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

$$\begin{array}{cc} a & b \\ \multicolumn{2}{c}{\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix}} \\ c & d \end{array} \times \begin{array}{cc} e & f \\ \multicolumn{2}{c}{\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}} \\ g & h \end{array} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

$$[a_{1,1}] \times [b_{1,1}] = a_{1,1} \cdot b_{1,1}$$

ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

- Παρατηρώ ότι χρειάζομαι 8 πολλαπλασιασμούς πινάκων διάστασης $\frac{n}{2} \times \frac{n}{2}$ και 4 προσθέσεις πινάκων διάστασης $\frac{n}{2} \times \frac{n}{2}$:

$$T(n) = \begin{cases} 1, & \alpha\nu\ n = 1 \\ 8T\left(\frac{n}{2}\right) + \mathcal{O}(n^2), & \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\alpha \end{cases}$$

Οι προσθέσεις είναι $4\left(\frac{n}{2}\right)^2 = n^2$

- Από κύριο θεώρημα, έχω $a = 8$, $b = 2$ και $f(n) = n^2$. Υπολογίζουμε $n^{\log_b \alpha} = n^{\log_2 8} = n^3$, άρα είμαστε στην Περίπτωση 1 (η $f(n)$ είναι πολυωνυμικά μικρότερη από την $n^{\log_b \alpha}$ κατά παράγοντα n^1), οπότε

$$\boldsymbol{T(n) = \mathcal{O}\left(n^3\right)}$$

ΑΛΓΟΡΙΘΜΟΣ ΤΟΥ STRASSEN

- Ο Strassen πρότεινε τους ακόλουθους υπολογισμούς:
  - $p_1 = a(f - h)$
  - $p_2 = (a + b)h$
  - $p_3 = (c + d)e$
  - $p_4 = d(g - e)$
  - $p_5 = (a + d)(e + h)$
  - $p_6 = (b - d)(g + h)$
  - $p_7 = (a - c)(e + f)$

7 πολλαπλασιασμοί
10 προσθέσεις

οπότε η σχέση

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

μετασχηματίζεται σε

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p_5 + p_4 - p_2 + p_6 & p_1 + p_2 \\ p_3 + p_4 & p_5 + p_1 - p_3 - p_7 \end{bmatrix}$$

8 προσθέσεις

ΟΡΘΟΤΗΤΑ ΑΛΓΟΡΙΘΜΟΥ

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} p_5 + p_4 - p_2 + p_6 & p_1 + p_2 \\ p_3 + p_4 & p_5 + p_1 - p_3 - p_7 \end{bmatrix}$$

$$\begin{aligned} p_1 &= a(f-h) \\ p_2 &= (a+b)h \\ p_3 &= (c+d)e \\ p_4 &= d(g-e) \\ p_5 &= (a+d)(e+h) \\ p_6 &= (b-d)(g+h) \\ p_7 &= (a-c)(e+f) \end{aligned}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

- $p_5 + p_4 - p_2 + p_6 = (a+d)(e+h) + d(g-e) - (a+b)h + (b-d)(g+h) = ae + ah + de + dh + dg - de - ah - bh + bg + bh - dg - dh = ae + bg$

- $p_1 + p_2 = a(f-h) + (a+b)h = af - ah + ah + bh = af + bh$

- $p_3 + p_4 = (c+d)e + d(g-e) = ce + de + dg - de = ce + dg$

- $p_5 + p_1 - p_3 - p_7 = (a+d)(e+h) + a(f-h) - (c+d)e - (a-c)(e+f) = ae + ah + de + dh + af - ah - ce - de - ae - af + ce + cf = cf + dh$

ΠΟΛΥΠΛΟΚΟΤΗΤΑ

- Έχουμε πλέον 7 πολλαπλασιασμούς πινάκων διάστασης $\frac{n}{2} \times \frac{n}{2}$ και 18 προσθέσεις

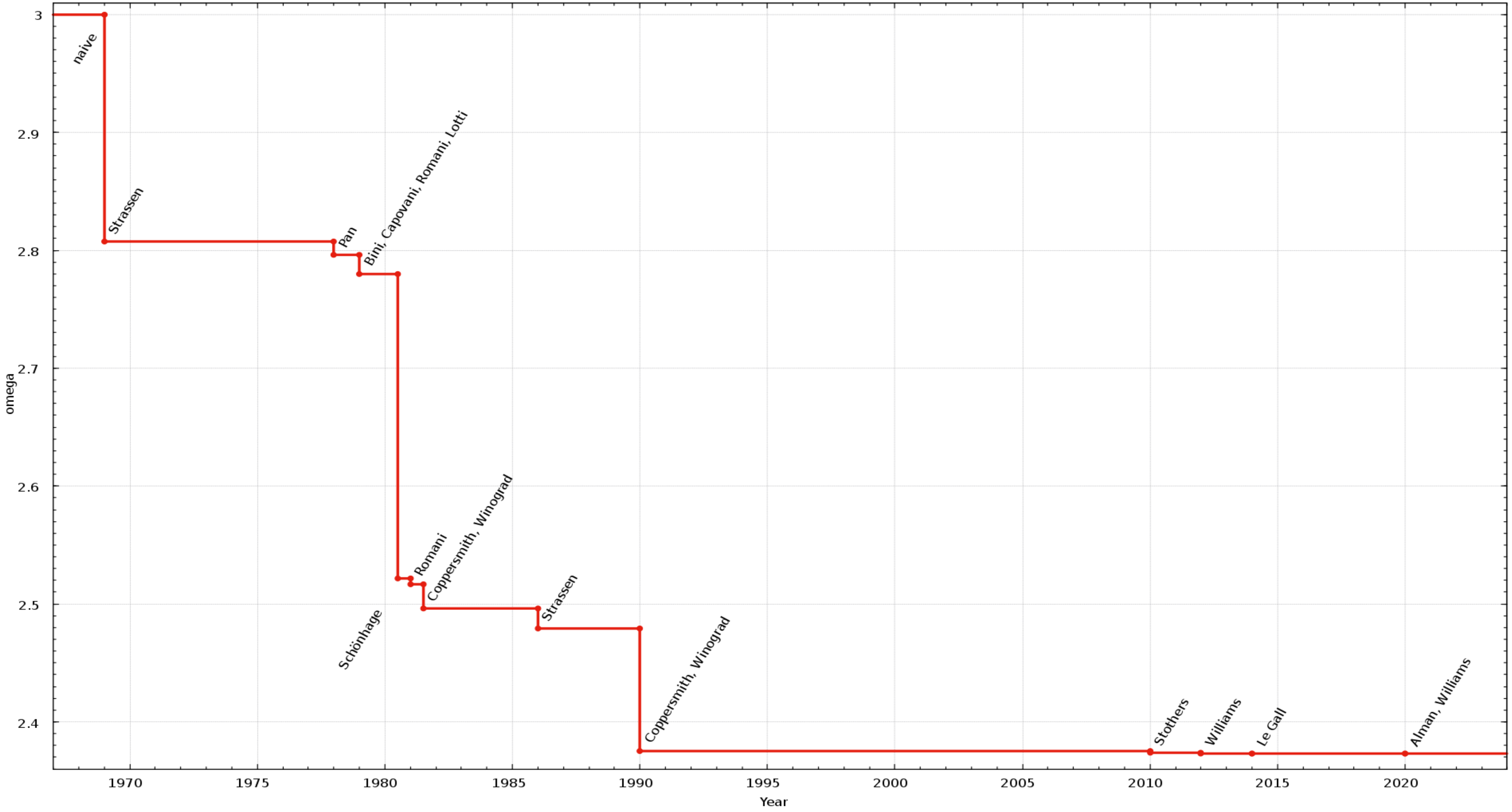
$$T(n) = \begin{cases} 1, & αν\ n = 1 \\ 7T\left(\frac{n}{2}\right) + \mathcal{O}(n^2), & διαφορετικα \end{cases}$$

Οι προσθέσεις είναι
$18\left(\frac{n}{2}\right)^2 = \frac{9}{2}n^2$

- Από κύριο θεώρημα, έχω $a = 7$, $b = 2$ και $f(n) = n^2$. Υπολογίζουμε $n^{\log_b \alpha} = n^{\log_2 7} \cong n^{2.81}$, άρα είμαστε στην Περίπτωση 1 (η $f(n)$ είναι πολυωνυμικά μικρότερη από την $n^{\log_b \alpha}$ κατά παράγοντα $n^{0.81}$), οπότε

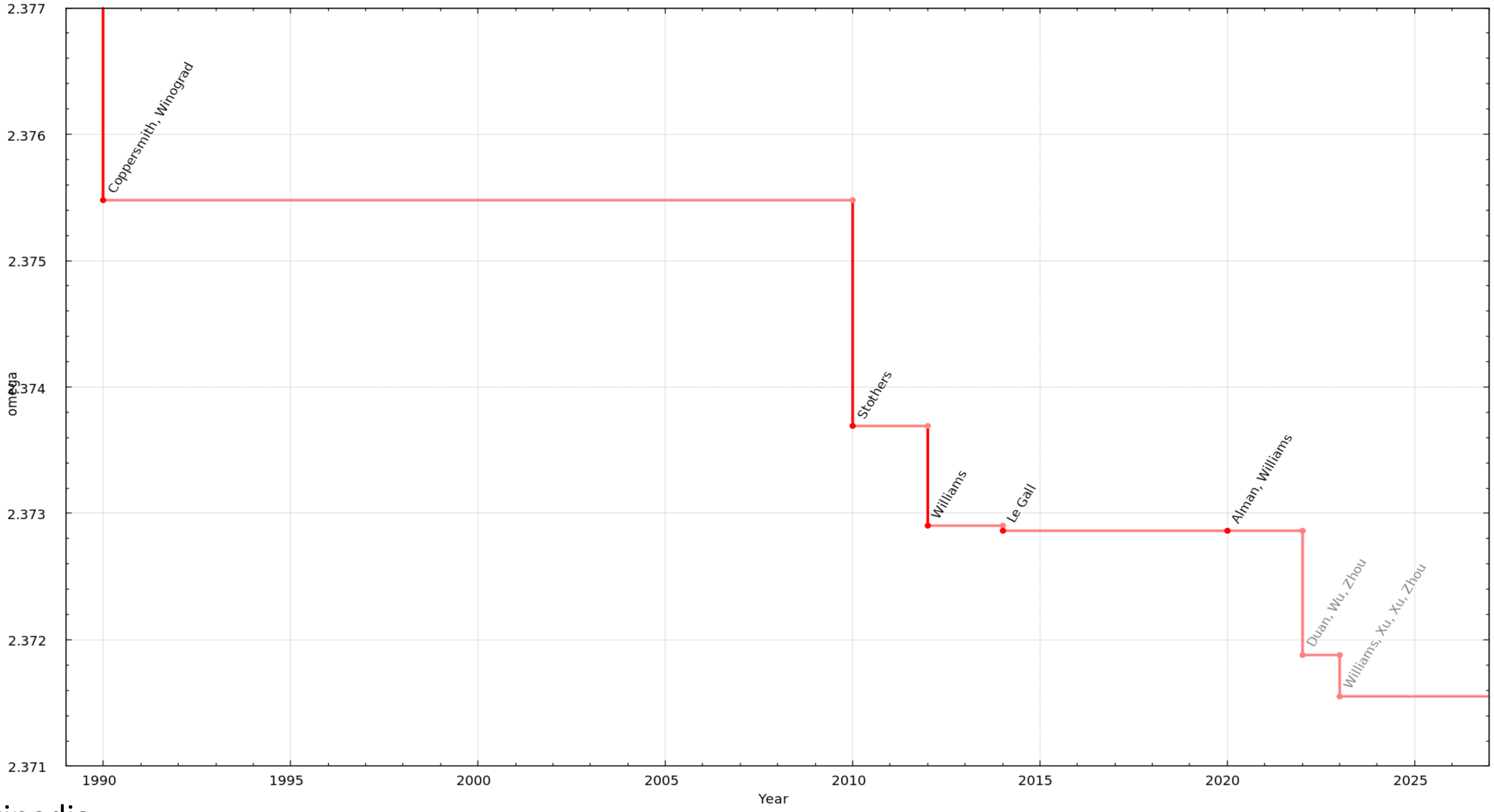
$$\boldsymbol{T(n) = \mathcal{O}\left(n^{2.81}\right)}$$

ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ ΠΙΝΑΚΩΝ



Πηγή: Wikipedia

ΑΛΓΟΡΙΘΜΟΙ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ ΠΙΝΑΚΩΝ



Πηγή: Wikipedia

Κωνσταντίνος Γιαννουτάκης

Επίκ. Καθηγητής

kgiannou@uom.edu.gr

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ