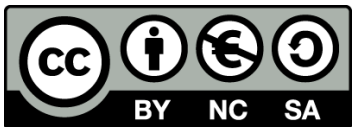
ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

ΑΛΓΟΡΙΘΜΟΙ

Ενότητα 4: Διαίρει και Βασίλευε

Ιωάννης Μανωλόπουλος, Καθηγητής
Αναστάσιος Γούναρης, Επίκουρος Καθηγητής
Τμήμα Πληροφορικής ΑΠΘ



Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

Διαίρει και Βασίλευε

Ταξινόμηση με Συγχώνευση, Γρήγορη Ταξινόμηση, Πολλαπλασιασμός κατά Strassen, Quickhull, Διάσχιση Δένδρων

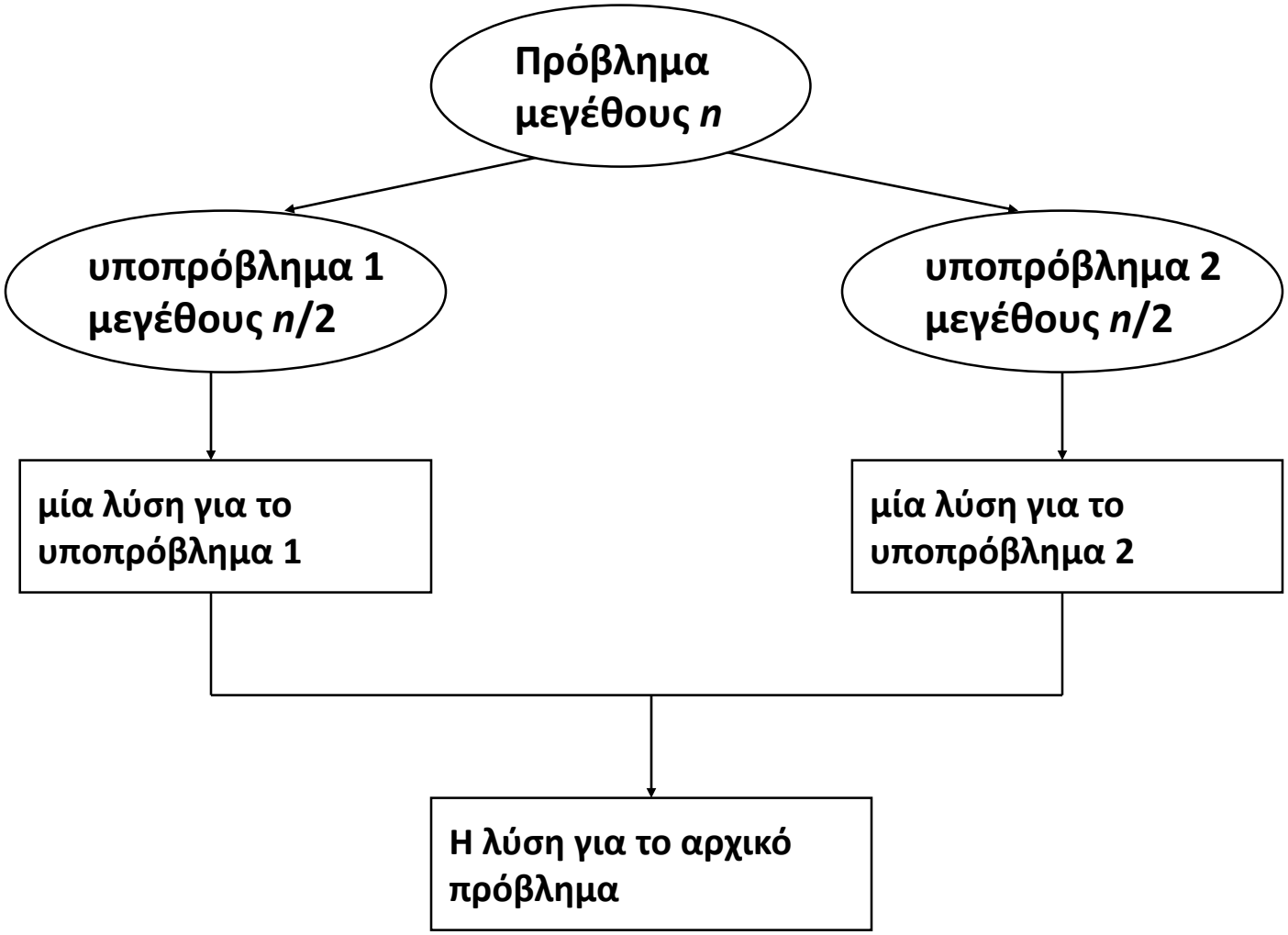
Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Διαίρει και Βασίλευε

Η γνωστότερη μέθοδος σχεδιασμού αλγορίθμων:

1. Διαιρούμε το στιγμιότυπο του προβλήματος σε δύο ή περισσότερα μικρότερα στιγμιότυπα
2. Επιλύουμε τα μικρότερα στιγμιότυπα αναδρομικά
3. Βρίσκουμε τη λύση στο αρχικό (μεγαλύτερο) στιγμιότυπο συνδυάζοντας αυτές τις λύσεις

Διαίρει και Βασίλευε



Παραδείγματα Διαίρει και Βασίλευε

- Ταξινόμηση με συγχώνευση και γρήγορη ταξινόμηση
- Διασχίσεις δένδρων
- Δυαδική αναζήτηση (;)
- Πολλαπλασιασμός πινάκων – Αλγόριθμος του Strassen
- Κυρτό περίβλημα - Αλγόριθμος QuickHull

Γενική αναδρομή του Διαίρει και Βασίλευε

$T(n) = aT(n/b) + f(n)$ όπου $f(n) \in \Theta(n^k)$, $k \geq 0$

1. $a < b^k$ $\quad T(n) \in \Theta(n^k)$
2. $a = b^k$ $\quad T(n) \in \Theta(n^k \lg n)$
3. $a > b^k$ $\quad T(n) \in \Theta(n^{\log_b a})$

Σημείωση: το ίδιο αποτέλεσμα ισχύει για το συμβολισμό Ο αντί για το Θ.

Ταξινόμηση με συγχώνευση (1)

Αλγόριθμος:

- Διαιρούμε τον πίνακα A[1..*n*] σε δύο τμήματα και αντιγράφουμε το καθένα στους πίνακες B[1.. *n*/2] και C[1.. *n*/2]
- Ταξινομούμε τους πίνακες B και C
- Συγχωνεύουμε τους πίνακες B και C στον πίνακα A ως εξής:
  - Επαναλαμβάνουμε μέχρι να μη μείνει κάποιο στοιχείο σε έναν από τους πίνακες:
    - Συγκρίνουμε τα πρώτα στοιχεία στα εναπομένοντα μη επεξεργασμένα τμήματα των πινάκων
    - Αντιγράφουμε το μικρότερο από τα δύο στοιχεία στον πίνακα A, ενώ αυξάνουνε το δείκτη προς το μη επεξεργασμένο τμήμα του
  - Όταν επεξεργασθούμε όλα τα στοιχεία του ενός πίνακα, τότε αντιγράφουμε στον πίνακα A τα υπόλοιπα μη επεξεργασθέντα στοιχεία από τον άλλο πίνακα

Ταξινόμηση με συγχώνευση (2)

Algorithm Mergesort(A[0..n-1])

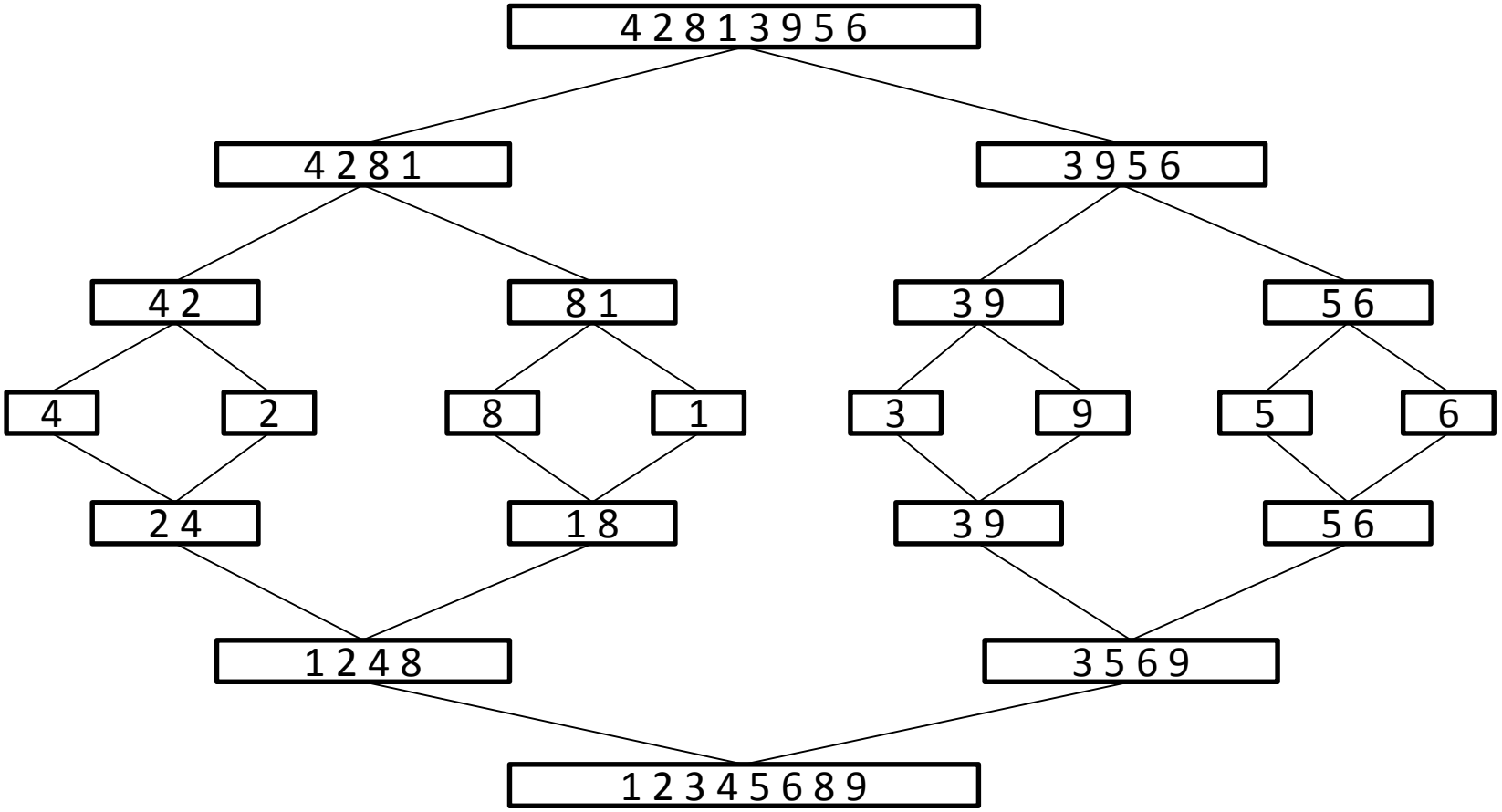
```
// Sorts array A[0..n-1] by recursive mergesort
// Input: An array A[0..n-1] of orderable elements
// Output: Array A[0..n-1] sorted in nondecreasing order
if n>1:
        copy A[0..⌊n/2⌋-1] to B[0..⌊n/2⌋-1]
        copy A[⌊n/2⌋..n-1] to C[0..⌈n/2⌉-1]
        Mergesort(B[0..⌊n/2⌋-1])
        Mergesort(C[0..⌈n/2⌉-1])
        Merge(B, C, A)
```

Ταξινόμηση με συγχώνευση (3)

Algorithm *Merge*(B[0..*p*-1], C[0..*q*-1], A[0..*p+q*-1])

```
// Merges two sorted arrays into one sorted array
// Input: Arrays B[0..p-1] and C[0..q-1] both sorted
// Output: Sorted array A[0..p+q-1] of the elements
of B and C
i ← 0; j ← 0; k ← 0
while i < p and j < q do
      if B[i] ≤ C[j]: A[k] ← B[i]; i++
      else A[k] ← C[j]; j++
      k++
if i = p: copy C[j..q-1] to A[k..p+q-1]
else copy B[i..p-1] to A[k..p+q-1]
```

Ταξινόμηση με συγχώνευση (4)

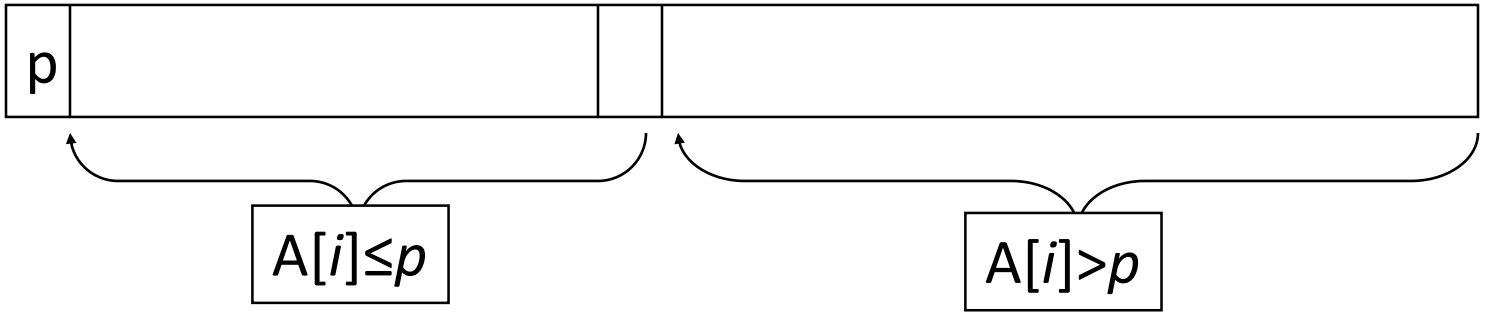


Αποδοτικότητα της ταξινόμησης με συγχώνευση

- $C(n) = 2\,C(n/2) + C_{merge}(n)$ for $n>1$, $C(1)=0$
- Όλες οι περιπτώσεις έχουν την ίδια αποδοτικότητα: $\Theta(n \log n)$
- Το πλήθος των συγκρίσεων προσεγγίζει το θεωρητικό ελάχιστο για τις ταξινομήσεις που βασίζονται στη σύγκριση:
  - $\log n ! \approx n \lg n - 1.44\, n$
- Απαιτήσεις χώρου: $\Theta(n)$ (δεν είναι in-place)
- Μπορεί να υλοποιηθεί χωρίς αναδρομή (bottom-up)

Γρήγορη Ταξινόμηση (1)

- Επιλέγουμε ένα *αξονικό στοιχείο* ως *άξονα* (το στοιχείο της διαμέρισης)
- Τακτοποιούμε τη λίστα έτσι ώστε τα στοιχεία σε θέσεις πριν από τον άξονα να είναι μικρότερα ή ίσα προς τον άξονα, ενώ εκείνα που είναι μετά τον άξονα είναι μεγαλύτερά του (δες τον αλγόριθμο *Partition*)
- Ανταλλάσουμε τον άξονα με το τελευταίο στοιχείο της πρώτης λίστας (δηλαδή, τη λίστα ≤), οπότε ο άξονας λαμβάνει την τελική του θέση
- Ταξινομούμε τις δύο λίστες



Γρήγορη Ταξινόμηση (2)

Algorithm Quicksort(A[l..r])

```
// Input: an array A[0..n-1] of orderable
   elements
// Output: Array A[0..n-1] sorted ascendingly

if l<r
   s ← partition(A[l..r])
   quicksort(A[l..s-1])
   quicksort(A[s+1..r])
```

Γρήγορη Ταξινόμηση (3)

Algorithm Partition(A[l..r])

```
// Partitions a subarray by using its first
   element as pivot
// Input: a subarray A[l..r], l<r
// Output: A partition of A[l..r] with the
   split position returned as this function's
   value

p←A[l]; i←l; j←r+1;
repeat
      repeat i←i+1 until A[i]≥p
      repeat j←j-1 until A[j]≤p
      swap(A[i],A[j])
until i≥j
swap(A[i],A[j])  //εξουδετέρωση αλλαγής όταν
   i≥j
swap(A[l],A[j])
return j
```

Γρήγορη Ταξινόμηση (4)

Παράδειγμα

5 3 1 9 8 2 4 7

Ποιά είναι η σειρά των κλήσεων συναρτήσεων?

Γρήγορη Ταξινόμηση (5)

- *Καλύτερη περίπτωση*: χωρισμός στη μέση — $\Theta(n \log n)$
- *Χειρότερη περίπτωση*: ταξινομημένος πίνακας ! — $\Theta(n^2)$
- *Μέση περίπτωση*: τυχαίος πίνακας — $\Theta(n \log n)$
- Βελτιώσεις:
  - Καλύτερη επιλογή του άξονα: το μεσαίο από 3 στοιχεία αποφεύγει τη χειρότερη περίπτωση σε ταξινομημένα αρχεία
  - Γυρνούμε σε ταξινόμηση με εισαγωγή για μικρά υποαρχεία
  - Αποφεύγουμε την αναδρομή

  Μπορούν να δώσουν βελτίωση μέχρι 20-25%
- Θεωρείται η καλύτερη μέθοδος για εσωτερική ταξινόμηση μεγάλων αρχείων ($n \geq 10000$)

Πολλαπλασιασμός μεγάλων ακεραίων (1)

- Για να πολλαπλασιάσουμε δύο ακεραίους με n_1 και n_2 ψηφία με το χέρι, θα εκτελέσουμε $n_1 n_2$ πράξεις πολλαπλασιασμού
- Πρόβλημα όταν έχουμε πολλά ψηφία:
  - A = 12345678901357986429 B = 87654321284820912836
- Ας δούμε ένα απλούστερο παράδειγμα: 23*14,

  - Όμως $23 = 2 \cdot 10^1 + 3 \cdot 10^0$, $14 = 1 \cdot 10^1 + 4 \cdot 10^0$

  - Άρα $23*14 = (2*1)10^2 + (2*4+3*1)10^1 + (3*4)\ 10^0$
- Αν A = 2135 , B = 4014:

  $A*B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$

  $= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$

Πολλαπλασιασμός μεγάλων ακεραίων (2)

- Βασική παρατήρηση:

  α) $23*14 = (2*1)10^2 +(2*4+3*1)10^1 + (3*4)\ 10^0$

  $= (2*1)10^2 +[(2+3)*(1+4)-2*1-3*4]10^1 + (3*4)\ 10^0$

- β) $2135 *4014 = 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$

  $= 21 * 40 \cdot 10^4 + [(21+35)* (40+14) – 21*40-35*14] \cdot 10^2 + 35 * 14$

- Γενικώς: $a=a_1a_0$, $b=b_1b_0$ και ζητείται $c=a*b$: $c = a*b = c_2 \cdot 10^n + c_1 \cdot 10^{n/2} + c_0 \cdot 10^0$ όπου $c_2=a_1*b_1$, $c_0=a_0*b_0$, $c_1=(a_1+a_0)*(b_1+b_0)-(c_2+c_0)$

- Αναδρομική εξίσωση: $M(n) = 3M(n/2)$, $M(1) = 1$

- Αποδοτικότητα; $M(n) = 3^{\log 2^n} = n^{\log 2^3} \approx n^{1.585}$

Πολλαπλασιασμός πινάκων αλά Strassen

- Ο Strassen παρατήρησε [1969] ότι το γινόμενο δύο πινάκων Α και Β (μεγέθους 2^nx2^n) μπορεί να υπολογισθεί ως εξής:

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$\begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

Υποπίνακες του Strassen

- $M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$
- $M_2 = (A_{10} + A_{11}) * B_{00}$
- $M_3 = A_{00} * (B_{01} - B_{11})$
- $M_4 = A_{11} * (B_{10} - B_{00})$
- $M_5 = (A_{00} + A_{01}) * B_{11}$
- $M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$
- $M_7 = (A_{01} - A_{11}) * (B_{10} + B_{11})$

Αποδοτικότητα του αλγορίθμου Strassen

- Αν το *n* δεν είναι δύναμη του 2, τότε οι πίνακες παραγεμίζονται με μηδενικά
- Πλήθος πολλαπλασιασμών:
- Πλήθος προσθέσεων:
- Αναδρομή:
- Άλλοι αλγόριθμοι έχουν βελτιώσει αυτό το αποτέλεσμα αλλά είναι ακόμη περισσότεροι σύνθετοι

Το πρόβλημα του πλησιέστερου ζεύγους

- Πρόβλημα: βρες τα πλησιέστερα σημεία μεταξύ *n* σημείων σε ένα *k*-διάστατο χώρο
- Υποθέτουμε ότι τα σημεία ταξινομούνται με βάση τις τιμές των τετμημένων *x*
- Διαιρούμε τα σημεία σε δύο υποσύνολα S_1 και S_2 με $n/2$ σημεία με βάση μία κατακόρυφη γραμμή στο $x=c$
- Επιλέγουμε το πλησιέστερο ζεύγος μεταξύ των ζευγών του αριστερού υποσυνόλου (απόσταση d_1), μεταξύ των ζευγών του δεξιού υποσυνόλου (απόσταση d_2), και το πλησιέστερο ζεύγος με σημεία και από τις δύο πλευρές
- Πρέπει να εξετάσουμε μία λωρίδα πλάτους 2d, όπου $d=\min[d_1,d_2]$

Το πρόβλημα του πλησιέστερου ζεύγους (2)

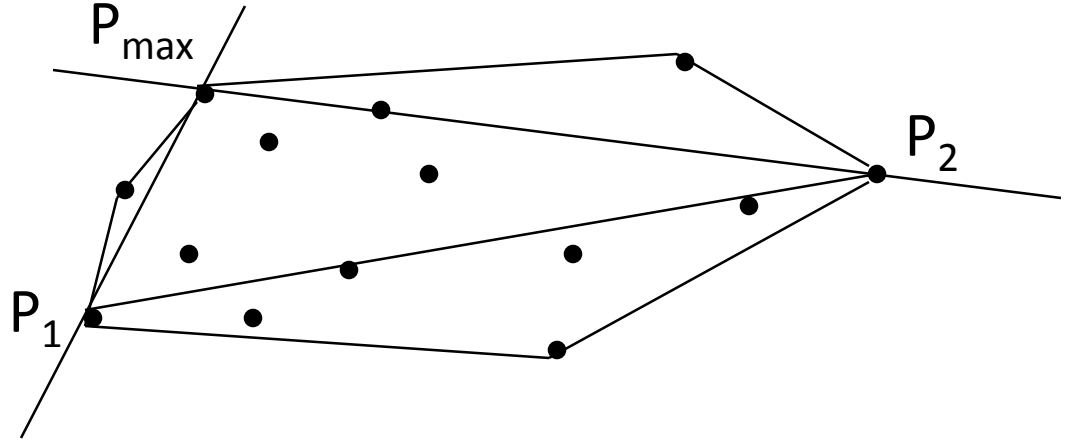
- Γεωμετρική εξήγηση

- Αναδρομή $T(n) = 2T(n/2) + M(n)$
- Αποδοτικότητα $O(n \log n)$
- Βελτιστότητα

Αλγόριθμος QuickHull

Υπολογίζει το Κυρτό Περίβλημα με έμπνευση από τη Γρήγορη Ταξινόμηση:

- Υποθέτουμε ότι τα σημεία είναι ταξινομημένα ως προς την συντεταγμένη *x*
- Προσδιορίζουμε δύο ακραία σημαία P_1 και P_2 (μέρη του περιβλήματος)
- Το σύνολο σημείων S διαιρείται σε δύο υποσύνολα S_1 και S_2
- Υπολογίζουμε το κυρτό περίβλημα για το S_1
- Ομοίως, υπολογίζουμε το κυρτό περίβλημα για το S_1



Αλγόριθμος QuickHull (2)

- Υπολογισμός του κυρτού (επάνω) περιβλήματος για το S_1
  - Βρίσκουμε το πιο απομακρυσμένο σημείο P_{max} από τη γραμμή P_1P_2
  - Υπολογίζουμε το περίβλημα για τα σημεία αριστερά της γραμμής P_1P_{max}
  - Υπολογίζουμε το περίβλημα για τα σημεία αριστερά της γραμμής $P_{max}P_2$
- Εύρεση του σημείου P_{max}
  - *Το* P_{max} μεγιστοποιεί την επιφάνεια του τριγώνου $P_{max}P_1P_2$
  - Σε περίπτωση ισοπαλίας, επιλέγεται το P_{max} που μεγιστοποιεί τη γωνία $P_{max}P_1P_2$
- Τα σημεία μέσα στο τρίγωνο $P_{max}P_1P_2$ αγνοούνται εφεξής
- Πως θα βρεθεί γεωμετρικά το σημείο P_{max} και τα σημεία στα αριστερά της γραμμής P_1P_{max} ;

Αλγόριθμος QuickHull (3)

- Πως θα βρεθεί γεωμετρικά το σημείο P_{max} και τα σημεία στα αριστερά της γραμμής $P_1P_{\max}$
  - Δίνονται τα σημεία $P_1(x_1,y_1)$, $P_2(x_2,y_2)$, $P_{max}(x_{max},y_{max})$
  - Το εμβαδόν του τριγώνου είναι μισό του μεγέθους της ορίζουσας

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_{max} & y_{max} & 1 \end{bmatrix} = x_1y_2+x_{max}y_1+x_2y_{max}-x_{max}y_2-x_2y_1-x_1y_{max}$$

  - Το πρόσημο της έκφρασης είναι θετικό αν και μόνο αν το σημείο P_{max} είναι στα αριστερά της γραμμής P_1P_2

Αποδοτικότητα Αλγορίθμου QuickHull (4)

- Η εύρεση του πλέον απομακρυσμένου σημείου από την ευθεία P_1P_2 μπορεί να βρεθεί σε γραμμικό χρόνο
- Έτσι προκύπτει η ίδια πολυπλοκότητα με τη γρήγορη ταξινόμηση:
  - *Χειρότερη περίπτωση*: $\Theta(n^2)$
  - *Μέση περίπτωση*: $\Theta(n \log n)$
- Αν τα σημεία δεν είναι αρχικά ταξινομημένα στον άξονα x, αυτό επιτυγχάνεται με κόστος $\Theta(n \log n)$

- Άλλοι αλγόριθμοι για το κυρτό περίβλημα:
  - Σάρωση του Graham, DCHull

  Επίσης σε $\Theta(n \log n)$

Δυαδική Αναζήτηση

Πολύ αποδοτικός αλγόριθμος για αναζήτηση σε <u>ταξινομημένο πίνακα</u>:

K

vs

A[0] . . . A[m] . . . A[n-1]

Αν K = A[m], σταμάτησε (επιτυχημένη αναζήτηση. Διαφορετικά, συνέχισε την αναζήτηση με όμοιο τρόπο στον πίνακα A[0..m-1] αν K < A[m] ή στον A[m+1..n-1] αν K > A[m]

```
l ← 0;    r ← n-1
while l ≤ r do
   m ← ⌊(l+r)/2⌋
      if  K = A[m]  return m
      else if K < A[m]  r ← m-1
      else l ← m+1
return -1
```

Ανάλυση Δυαδικής Αναζήτησης

- Χρονική αποδοτικότητα
  - Χειρότερη περίπτωση: $C_w(n) = 1 + C_w(\lfloor n/2 \rfloor)$, $C_w(1) = 1$
  - Λύση: $C_w(n) = \lceil \log_2(n+1) \rceil$

    Αυτό είναι πολύ γρήγορο: π.χ., $C_w(10^6) = 20$
- Βέλτιστη μέθοδος για αναζήτηση σε ταξινομημένο πίνακα.
- Περιορισμοί: η είσοδος πρέπει να είναι ταξινομημένη και στη μορφή πίνακα, όχι, πχ, συνδεδεμένη λίστα.
- ΟΧΙ καλό παράδειγμα τεχνικής «διαίρει και βασίλευε».
- Υπάρχει ανάλογη τεχνική για τη συνεχή περίπτωση.

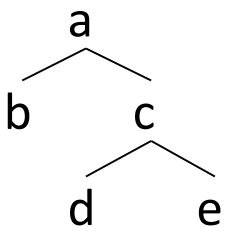
Αλγόριθμοι Δυαδικών Δένδρων

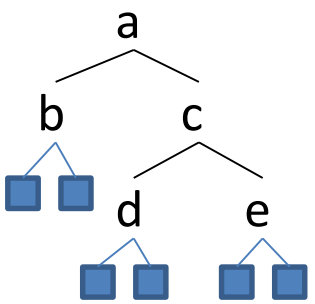
Τα δυαδικά δένδρα έχουν έτοιμη τη δομή για αλγορίθμους διαίρει και βασίλευε.

Π.χ., κλασικές διασχίσεις (προ/μετα/ενδο-διατεταγμένη)

Algorithm *Inorder*(*T*)

```
if T ≠ ∅
    Inorder(T_left)
    print(root of T)
    Inorder(T_right)
```

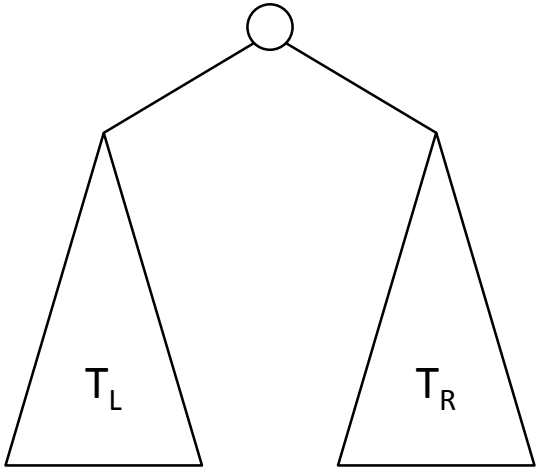




Αποδοτικότητα: Θ(*n*)

Αλγόριθμοι Δυαδικών Δένδρων (2)

Άλλο παράδειγμα: υπολογισμός ύψους



- $h(T) = \max\{h(T_L), h(T_R)\} + 1$ if $T \neq \varnothing$ and $h(\varnothing) = -1$
- Αποδοτικότητα: $\Theta(n)$

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Τέλος ενότητας

Επεξεργασία: Ανδρέας Κοσματόπουλος

Θεσσαλονίκη, Αύγουστος 2015

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.00.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.