Διαίρει-και-Βασίλευε

Διδάσκοντες: **Αρ. Παγουρτζής, Δ. Φωτάκης, Δ. Σούλιου, Παν. Γροντάς**

Επιμέλεια διαφανειών: **Δ. Φωτάκης**

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο

Διαίρει-και-Βασίλευε

- Γενική μέθοδος σχεδιασμού (αναδρομικών) αλγορίθμων:
  - **Διαίρεση** σε (≥ 2) υποπροβλήματα (σημαντικά) μικρότερου μεγέθους.
  - **Ανεξάρτητη επίλυση** υπο-προβλημάτων (αναδρομικά) (για μικρά υποπροβλήματα εφαρμόζονται στοιχειώδεις αλγόριθμοι).
  - **Σύνθεση** λύσης αρχικού προβλήματος από λύσεις υποπροβλημάτων.
- Ισχυρή μέθοδος, με πολλές σημαντικές εφαρμογές !
  - Ταξινόμηση: MergeSort, QuickSort.
  - Πολλαπλασιασμός αριθμών, πινάκων, FFT, closest pair of points.
  - «Εξειδίκευση»: Δυαδική αναζήτηση, QuickSelect, ύψωση σε δύναμη.
- (Εύκολη) ανάλυση με **αναδρομικές σχέσεις.**
  - Μη γραμμικές, συγκεκριμένης μορφής.

Πρόβλημα Ταξινόμησης

- Πρόβλημα Ταξινόμησης:
  - **Είσοδος** : ακολουθία n αριθμών $(a_1, a_2, ..., a_n)$.
  - **Έξοδος** : αναδιάταξη $(a'_1, a'_2, ..., a'_n)$ με αριθμούς σε **αύξουσα** σειρά $(\forall i \;\; a'_i \leq a'_{i+1})$.
- Στατική συλλογή δεδομένων (όχι εισαγωγή και διαγραφή).
- **Θεμελιώδες** αλγοριθμικό πρόβλημα.
  - Πολλές εφαρμογές (περ. 20%-25% υπολογιστικού χρόνου).
  - Ταχύτατη αναζήτηση σε ταξινομημένα δεδομένα.
  - Σημαντικές αλγοριθμικές ιδέες και τεχνικές.

Συγκριτικές Μέθοδοι Ταξινόμησης

- **Αντιμετάθεση** κάθε ζεύγους στοιχείων εκτός διάταξης (bubble sort).
- **Εισαγωγή** στοιχείου σε κατάλληλη θέση ταξινομημένου πίνακα (insertion sort).
- **Επιλογή** μεγαλύτερου στοιχείου και τοποθέτηση στο τέλος (selection sort, heapsort).
- **Συγχώνευση** ταξινομημένων πινάκων :
  Διαίρεση στη μέση, ταξινόμηση, συγχώνευση (mergesort).
- **Διαίρεση** σε μικρότερα και μεγαλύτερα από στοιχείο-διαχωρισμού και ταξινόμηση (quicksort).

Συγκριτικοί Αλγόριθμοι

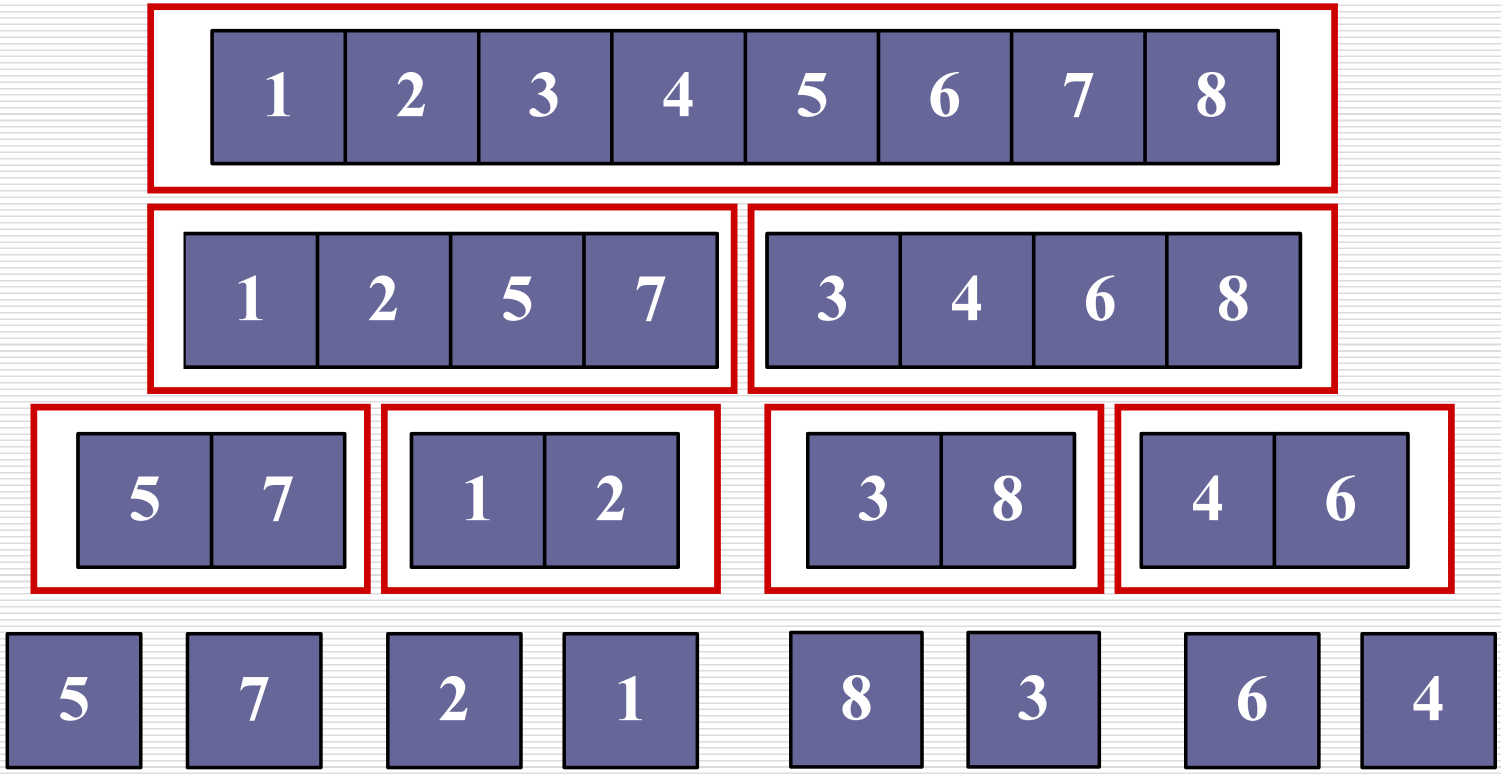
- Ταξινόμηση **αποκλειστικά** με συγκρίσεις και αντιμεταθέσεις στοιχείων.
  - Δεν εκμεταλλεύονται τύπο δεδομένων (π.χ. αριθμούς, συμβολοσειρές) : γενική εφαρμογή.
  - **Ανάλυση** : πλήθος συγκρίσεων.
πλήθος αντιμεταθέσεων.
  - Κάτω φράγμα #συγκρίσεων : $\Omega(n \log n)$.
- Ταξινόμηση (με αλγόριθμους που δεν είναι συγκριτικοί) σε γραμμικό χρόνο για n φυσικούς αριθμούς (π.χ. counting sort, radix sort, bucket sort).

MergeSort

- MergeSort (ταξινόμηση με συγχώνευση):
  - **Διαίρεση** ακολουθίας εισόδου (n στοιχεία) σε δύο υπο-ακολουθίες ίδιου μήκους (n / 2 στοιχεία).
  - **Ταξινόμηση** υπο-ακολουθιών **αναδρομικά**.
  - **Συγχώνευση** (merge) δύο ταξινομημένων υπο-ακολουθιών σε μία ταξινομημένη ακολουθία.

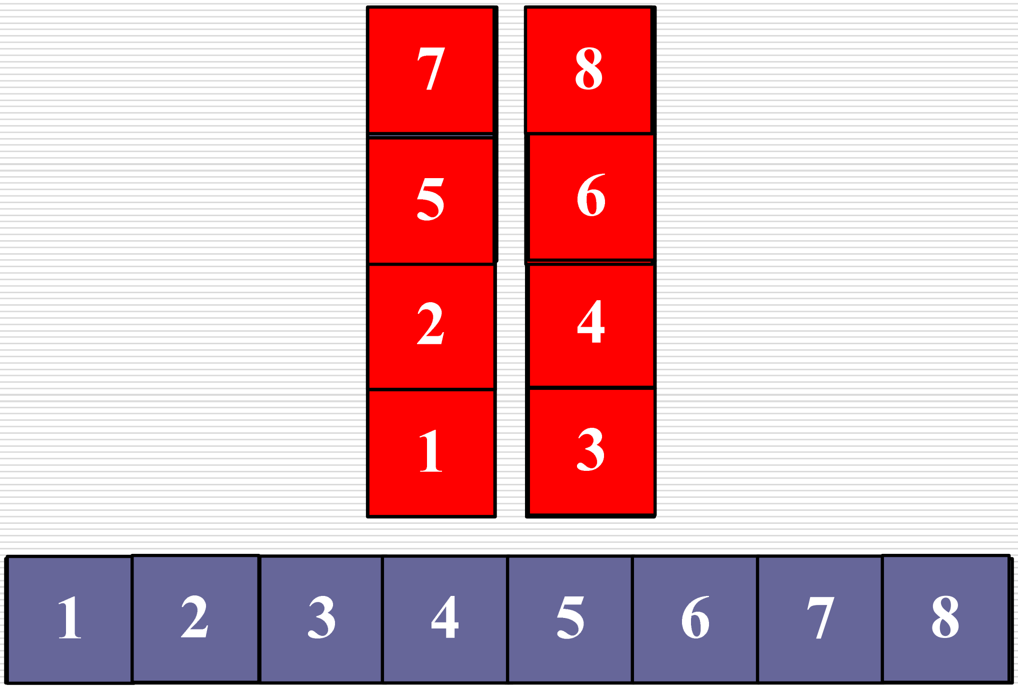
```
mergeSort(int A[], int left, int right) {
        if (left >= right) return;
        mid = (left + right) / 2;
        mergeSort(A, left, mid);
        mergeSort(A, mid+1, right);
        merge(A, left, mid, right); }
```

MergeSort



Συγχώνευση

- **Συγχώνευση** ταξινομημένων **`A[left...mid]`** και **`A[mid+1...right]`** σε ταξινομημένο **`A[left...right]`**.



Συγχώνευση

- **Συγχώνευση** ταξινομημένων `A[low...mid]` και `A[mid+1...up]` σε ταξινομημένο `A[low...up]`.
- ```
  X[up−low+1] ← A[low...up]; // προσωρινή αποθήκευση
  i : 0 <= i <= xmid  // δείκτης αριστερό τμήμα
  j : xmid+1 <= j <= xup // δείκτης δεξιό τμήμα
  k : low <= k <= up     // δείκτης στο συγχωνευμένο πίνακα
  ```
- `X[i]` : μικρότερο διαθέσιμο στοιχείο στο αριστερό τμήμα.
  `X[j]` : μικρότερο διαθέσιμο στοιχείο στο δεξιό τμήμα.
- ```
  while ((i <= xmid) && (j <= xup))
     if (X[i] <= X[j]) A[k++] = X[i++];
     else A[k++] =X[j++];
  ```
- Όταν ένα τμήμα **εξαντληθεί**, αντιγράφουμε όλα τα στοιχεία του άλλου στο A[ ].

Συγχώνευση

```
merge(int A[], int low, int mid, int up) {
    int xmid = mid-low, xup = up-low;
    int i = 0,  // δείκτης στο αριστερό τμήμα
    j = xmid+1, // δείκτης στο δεξιό τμήμα
    k = low;    // δείκτης στο αποτέλεσμα
    X[up-low+1] ← A[low...up];
    // συγχώνευση μέχρι ένα τμήμα να εξαντληθεί
    while ((i <= xmid) && (j <= xup))
        if (X[i] <= X[j]) A[k++] = X[i++];
        else A[k++] = X[j++];
    // αντέγραψε υπόλοιπα στοιχεία άλλου τμήματος
    if (i > xmid)
        for (int q = j; q <= xup; q++)
            A[k++] = X[q];
    else
        for (int q = i; q <= xmid; q++)
            A[k++] = X[q];
```

MergeSort: Ορθότητα

- Ορθότητα **merge** επειδή τα τμήματα είναι ταξινομημένα.
  - Όταν ένα στοιχείο μεταφέρεται στον A[ ], δεν υπάρχει μικρότερο διαθέσιμο στοιχείο στα δύο τμήματα.
- Ορθότητα **mergeSort** αποδεικνύεται επαγωγικά :
  - Βάση (ένα στοιχείο) τετριμμένη.
  - Δύο τμήματα σωστά ταξινομημένα (επαγωγική υποθ.) και συγχωνεύονται σωστά (ορθότητα merge) $\Rightarrow$

    Σωστά ταξινομημένος πίνακας A[ ].

```
mergeSort(int A[], int left, int right) {
        if (left >= right) return;
        mid = (left + right) / 2;
        mergeSort(A, left, mid);
        mergeSort(A, mid+1, right);
        merge(A, left, mid, right); }
```

Χρόνος Εκτέλεσης

- Χρόνος εκτέλεσης merge (για n στοιχεία) : $\Theta(n)$ (γραμμικός)
  - $\Theta(1)$ λειτουργίες για κάθε στοιχείο.
- Χρόνος εκτέλεσης αλγόριθμων «διαίρει-και-βασίλευε» με διατύπωση και λύση αναδρομικής εξίσωσης λειτουργίας.
- $\boldsymbol{T(n)}$: χρόνος (χ.π.) για ταξινόμηση n στοιχείων.
  - $\boldsymbol{T(n / 2)}$: ταξινόμηση αριστερού τμήματος (n / 2 στοιχεία).
  - $\boldsymbol{T(n / 2)}$: ταξινόμηση δεξιού τμήματος (n / 2 στοιχεία).
  - $\boldsymbol{\Theta(n)}$: συγχώνευση ταξινομημένων τμημάτων.

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

- Χρόνος εκτέλεσης MergeSort: $\boldsymbol{T(n) = ???}$

Δέντρο Αναδρομής

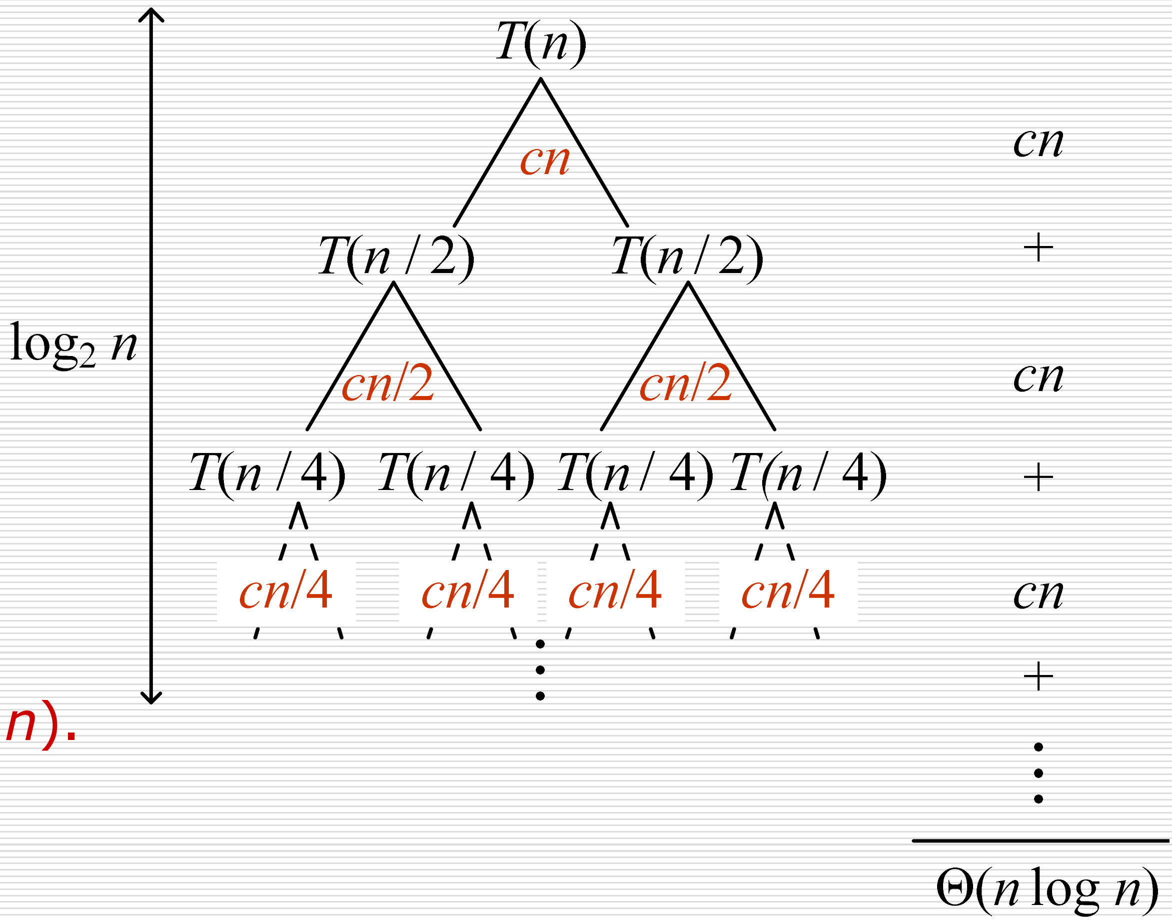
$T(n) = 2\,T(n/2) + \Theta(n)$,
$T(1) = \Theta(1)$

Δέντρο αναδρομής :

Ύψος : Θ(log n)
#κορυφών : Θ(n)

Χρόνος / επίπεδο : $\Theta(n)$

Συνολικός χρόνος : $\Theta(n \log n)$.



Master Theorem

- Ανάλυση χρόνου εκτέλεσης αλγορίθμων «διαίρει-και-βασίλευε» με αναδρομικές σχέσεις της μορφής

$$T(n) = a\,T(n/b) + f(n), T(1) = \Theta(1)$$

  όπου *a*, *b* σταθερές και *f*(*n*) θετική συνάρτηση.
- Επίλυση με Θεώρημα Κυρίαρχου Όρου (Master Theorem)
  1. Αν $f(n) = O(n^{\log_b a-\varepsilon}), \varepsilon > 0$, τότε $T(n) = \Theta(n^{\log_b a})$
  2. Αν $f(n) = \Theta(n^{\log_b a})$, τότε $T(n) = \Theta(n^{\log_b a}\log n)$
  3. Αν $f(n) = \Omega(n^{\log_b a+\varepsilon}), \varepsilon > 0$, και $af(n/b) < f(n)$, τότε $T(n) = \Theta(f(n))$
  - Ασυμπτωτικά μεγαλύτερος από *f*(*n*) και $n^{\log_b a}$ καθορίζει λύση.

Master Theorem: Ειδικές Μορφές

- Όταν $f(n) = \Theta(n)$, δηλ. $T(n) = a\,T(n/b) + \Theta(n), T(1) = \Theta(1)$
  1. Αν $a > b$, τότε $T(n) = \Theta(n^{\log_b a})$
  2. Αν $a = b$, τότε $T(n) = \Theta(n \log n)$
  3. Αν $a < b$, τότε $T(n) = \Theta(n)$

  - Αν $T(n) = T(\gamma_1 n) + T(\gamma_2 n) + \Theta(n), T(1) = \Theta(1)$
    με $\gamma_1 + \gamma_2 < 1 - \varepsilon$, τότε $T(n) = \Theta(n)$

- Όταν $f(n) = \Theta(n^d)$, δηλ. $T(n) = a\,T(n/b) + \Theta(n^d), T(1) = \Theta(1)$
  1. Αν $d < \log_b a$, τότε $T(n) = \Theta(n^{\log_b a})$
  2. Αν $d = \log_b a$, τότε $T(n) = \Theta(n^d \log n)$
  3. Αν $d > \log_b a$, τότε $T(n) = \Theta(n^d)$

Παραδείγματα

- T(n) = 9 T(n/3) + n. $T(n) = \Theta(n^2)$ (περ. 1)
- T(n) = T(2n/3) + 1. $T(n) = \Theta(\log n)$ (περ. 2)
- T(n) = 3 T(n/4) + n log n. $T(n) = \Theta(n \log n)$ (περ. 3)
- T(n) = 2 T(n/2) + n. $T(n) = \Theta(n \log n)$ (περ. 2)
- T(n) = 2 T(n/2) + n log n.
  - Δεν εμπίπτει! Με δέντρο αναδρομής βρίσκουμε ότι T(n) = Θ(n $\log^2$ n).

Πολλαπλασιασμός Αριθμών

- Υπολογισμός αθροίσματος x + y , x και y αριθμοί *n*-bits.
  - Κλασσικός αλγόριθμος πρόσθεσης, χρόνος Θ(n).
- Υπολογισμός γινομένου x × y , x και y αριθμοί με *n*-bits.
  - Κλασσικός αλγόριθμος πολ/μού, χρόνος $\Theta(n^2)$.
  - Καλύτερος αλγόριθμος;
- Διαίρει-και-Βασίλευε:
  - Διαίρεση: $x = 2^{n/2}x_h + x_l\,,\ y = 2^{n/2}y_h + y_l$

$$x \times y = 2^n \overbrace{x_h y_h}^{z_h} + 2^{n/2} \overbrace{(x_h y_l + x_l y_h)}^{z_m} + \overbrace{x_l y_l}^{z_l} = 2^n z_h + 2^{n/2} z_m + z_l$$

  - 4 πολλαπλασιασμοί (n / 2)-bits, 2 ολισθήσεις, 3 προσθέσεις.
  - Χρόνος: $T_1(n) = 4T_1(n/2) + \Theta(n) \Rightarrow T_1(n) = \Theta(n^2)$

Πολλαπλασιασμός Αριθμών

$$x \times y = 2^n \overbrace{x_h y_h}^{z_h} + 2^{n/2} \overbrace{(x_h y_l + x_l y_h)}^{z_m} + \overbrace{x_l y_l}^{z_l} = 2^n z_h + 2^{n/2} z_m + z_l$$

- Όμως z_m υπολογίζεται με 1 μόνο πολ/μο (n/2+1)-bits.

$$z_m = (x_h + x_l)(y_h + y_l) - x_h y_h - x_l y_l$$

  - 3 πολλαπλασιασμοί (n / 2)-bits, 2 ολισθήσεις, 6 προσθέσεις.
  - Χρόνος: $T(n) = 3T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n^{\log_2 3})$
- Παράδειγμα: 2576 × 7935 = 20440560

$x_h = 25\,,\ x_l = 76\,, \qquad y_h = 79\,,\ y_l = 35$

$z_h = 25 \times 79 = 1975\,, \quad z_l = 76 \times 35 = 2660$

$z_m = (25 + 76)(79 + 35) - 1975 - 2660 =$

$\quad = 101 \times 114 - 1975 - 2660 = 11514 - 1975 - 2660 = 6879$

$x \times y = 1975 \cdot 10^4 + 6879 \cdot 10^2 + 2660 = 20404560$

Πολλαπλασιασμός Πινάκων

- Υπολογισμός γινομένου C = A × B.
  A, B τετραγωνικοί πίνακες n × n.
- Εφαρμογή ορισμού: $C[i,j] = \sum_{k=1}^{n} A[i,k]B[k,j]$
  - Χρόνος Θ(n^3) (n^2 στοιχεία, χρόνος Θ(n) για καθένα).
- Διαίρει-και-Βασίλευε:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

  - 8 πολ/μοι και 4 προσθέσεις πινάκων $\frac{n}{2} \times \frac{n}{2}$
  - Χρόνος: $T_1(n) = 8\,T_1(n/2) + \Theta(n^2) \Rightarrow T_1(n) = \Theta(n^3)$

Αλγόριθμος Strassen (1969)

$$D_1 = (A_{21} + A_{22} - A_{11})(B_{22} - B_{12} + B_{11})$$
$$D_2 = A_{11}B_{11}$$
$$D_3 = A_{12}B_{21}$$
$$D_4 = (A_{11} - A_{21})(B_{22} - B_{12})$$
$$D_5 = (A_{21} + A_{22})(B_{12} - B_{11})$$
$$D_6 = (A_{12} - A_{21} + A_{11} - A_{22})B_{22}$$
$$D_7 = A_{22}(B_{11} + B_{22} - B_{12} - B_{21})$$

$$C_{11} = D_2 + D_3$$
$$C_{12} = D_1 + D_2 + D_5 + D_6$$
$$C_{21} = D_1 + D_2 + D_4 - D_7$$
$$C_{22} = D_1 + D_2 + D_4 + D_5$$

- 7 πολ/μοι και 24 προσθέσεις πινάκων $\frac{n}{2} \times \frac{n}{2}$
  - Χρόνος: $T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7})$

Υπολογισμός Δύναμης (Diffie-Hellman)

- Συμφωνία Αλίκης και Βασίλη σε κρυπτογραφικό κλειδί. Εύα παρακολουθεί για να «κλέψει» το κλειδί.
- A, B συμφωνούν δημόσια σε πρώτο p και ακέραιο $q < p$. E γνωρίζει p, q.
  - Εμπλεκόμενοι αριθμοί είναι πολυψήφιοι (π.χ. 512 ψηφία).
- A διαλέγει τυχαία $a < p$ και υπολογίζει $q_a = q^a \bmod p$
  B διαλέγει τυχαία $b < p$ και υπολογίζει $q_b = q^b \bmod p$
  A, B ανταλλάσσουν q_a, q_b και τα μαθαίνει E.
- A, B υπολογίζουν K (μόνοι τους). E δεν ξέρει K.
$$K = q_b^a \bmod p = (q^b \bmod p)^a \bmod p = q^{ab} \bmod p$$
- Για K, E χρειάζεται a, b (δεν μεταδόθηκαν). Επίλυση διακριτού λογαρίθμου (πολύ δύσκολο).

Υπολογισμός Δύναμης

- Εφαρμογή υποθέτει **αποδοτικό** αλγόριθμο υπολογισμού $\exp(x, n, p) = x^n \bmod p$, *x*, *n*, *p* πολυψήφιοι ακέραιοι.
  - Υπολογισμός δυνάμεων με τη σειρά (1, 2, 3, …): αν μήκος 512 bits, χρειάζεται περίπου 2^{512} πολ/μους!!!
- Διαίρει-και-Βασίλευε (έστω *n* δύναμη του 2):
  - Υπολογίζουμε αναδρομικά $\exp(x, n/2, p) = x^{n/2} \bmod p$
  - … και $\exp(x, n, p) = \exp(x, n/2, p) \times \exp(x, n/2, p)$
- #πολλαπλασιασμών:

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

$$\Rightarrow T(n) = \mathrm{O}(\log n)$$

  - p με μήκος 512 bits: περίπου 2^{10} πολ/μους.

$\mathrm{ExponRec}(x, n, p)$

- **if** $n = 1$ **then return**$(x \bmod p)$;
- $t \leftarrow \mathrm{ExponRec}(x, \lfloor n/2 \rfloor, p)$;
- $t \leftarrow t^2 \bmod p$;
- **if** n is odd **then return**$(t \times x \bmod p)$;
- **else return**(t);

Προϋποθέσεις Εφαρμογής

- Διαίρεση σημαντικά ευκολότερη από επίλυση αρχικού.
- Σύνθεση σημαντικά ευκολότερη από επίλυση αρχικού.
- Υπο-στιγμιότυπα σημαντικά μικρότερα από αρχικό
  (π.χ. αρχικό μέγεθος n, υπο-στιγμ. μεγέθους n / c, $c > 1$).
- Ανεξάρτητα υπο-στιγμιότυπα που λύνονται από ανεξάρτητες αναδρομικές κλήσεις.
  - Ίδια ή επικαλυπτόμενα υπο-στιγμιότυπα :
    σημαντική και αδικαιολόγητη αύξηση χρόνου εκτέλεσης.
  - Επικαλυπτόμενα υπο-στιγμιότυπα :
    Δυναμικός Προγραμματισμός

(Αντι)παράδειγμα

- Υπολογισμός *n*-οστού όρου ακολουθίας Fibonacci.

$$f_n = f_{n-1} + f_{n-2}, n \geq 2$$
$$f_0 = 0, f_1 = 1$$

```
long fibRec(long n) {
   if (n <= 1) return(n);
   return(fibRec(n-1) + fibRec(n-2)); }
```

- Χρόνος εκτέλεσης:

$$T(n) = \Theta(1) + T(n-1) + T(n-2)\,,\;\; T(1) = \Theta(1)$$

- Λύση: $T(n) = \Theta(\varphi^n)\,,\;\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$
- Επικαλυπτόμενα στιγμ.: Εκθετικός χρόνος!
- Αλγόριθμος «γραμμικού» χρόνου;
- Καλύτερος αλγόριθμος;

```
long fib(long n) {
  long cur = 1, prev = 0;
  for (i = 2; i <= n; i++) {
      cur = cur + prev;
      prev = cur – prev; }
  return(cur);  }
```

Ακολουθία Fibonacci

- Ακολουθία Fibonacci: $f_n = f_{n-1} + f_{n-2}, n \geq 2$
  $f_0 = 0, f_1 = 1$
- Θεωρούμε πίνακα $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ και $F_n = [f_n, f_{n-1}]$
  - Παρατηρούμε ότι $A \times F_n = [f_n + f_{n-1}, f_n] = F_{n+1}$
  - Με επαγωγή αποδεικνύουμε ότι $F_n = A^{n-1} \times F_1\,, F_1 = [1, 0]$
- Διαίρει-και-Βασίλευε:
  - Υπολογισμός A^n σε χρόνο O(log *n*) (όπως με αριθμούς).
  - Υπολογίζω αναδρομικά το $A^{n/2}$ και $A^n = A^{n/2} \times A^{n/2}$
  - Χρόνος: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\log n)$