

Θεμελιώδη Θέματα Επιστήμης Υπολογιστών

ΣΗΜΜΥ – ΣΕΜΦΕ ΕΜΠ

Ενότητα 4η:

Αλγοριθμικές τεχνικές, αριθμητικοί υπολογισμοί,
αποδοτικότητα αλγορίθμων

Επιμέλεια διαφανειών: Στάθης Ζάχος, Άρης Παγουρτζής

Αλγόριθμος

- **Webster's** 50 χρόνια πριν: ανύπαρκτος όρος
- **Oxford's**, 1971: «erroneous refashioning of *algorism*: calculation with Arabic numerals»
- **Abu Jaffar Mohammed Ibn Musa Al-Khowarizmi**, الخوارزمي موسى بن محمد, 9^{ος} αι. μ.Χ.
- Παραδείγματα:
 - Ευκλείδειος αλγόριθμος (**Ευκλείδης**, 3^{ος} αι. π.Χ.) για εύρεση ΜΚΔ
 - Αριθμοί Fibonacci (**Leonardo Pisano Filius Bonacci**, 13^{ος} αι. μ.Χ.)
 - Τρίγωνο Pascal (**Yang Hui**, 13^{ος} αι. μ.Χ.)

Αλγόριθμος (συν.)

- **Πρωταρχική έννοια**. Μέθοδος επίλυσης προβλήματος δοσμένη ως πεπερασμένο σύνολο κανόνων (ενεργειών, διεργασιών) που επενεργούν σε δεδομένα (data).
- **Πεπερασμένη εκτέλεση** (finiteness).
- Κάθε κανόνας ορίζεται επακριβώς και η αντίστοιχη διεργασία είναι συγκεκριμένη (definiteness).
- Δέχεται μηδέν ή περισσότερα **μεγέθη εισόδου (input)**.
- Δίνει τουλάχιστον ένα μέγεθος ως **αποτέλεσμα (output)**.
- Μηχανιστικά αποτελεσματικός, **εκτέλεση με “μολύβι και χαρτί”** (effectiveness).

Αλγόριθμος Ευκλείδη

- **if** $a > b$ **then** $\text{GCD}(a, b) := \text{GCD}(a \bmod b, b)$
else $\text{GCD}(a, b) := \text{GCD}(a, b \bmod a)$

$(a \bmod b = \text{το υπόλοιπο της διαίρεσης } a \text{ div } b)$

$$\text{ΜΚΔ}(172, 54) =$$

$$\text{ΜΚΔ}(10, 54) =$$

$$\text{ΜΚΔ}(10, 4) =$$

$$\text{ΜΚΔ}(2, 4) =$$

$$\text{ΜΚΔ}(2, 0) = \mathbf{2}$$

Αλγόριθμος Ευκλείδη

- **if** $a > b$ **then** $\text{GCD}(a, b) := \text{GCD}(a \bmod b, b)$
else $\text{GCD}(a, b) := \text{GCD}(a, b \bmod a)$

($a \bmod b =$ το υπόλοιπο της διαίρεσης $a \mathbf{div} b$)

- *Ο Ευκλείδειος αλγόριθμος είναι ο καλύτερος γνωστός αλγόριθμος για ΜΚΔ!*
- Ανοιχτό ερώτημα: είναι βέλτιστος;

Τρίγωνο Pascal (Yang Hui)

				1					
			1		1				
		1		2		1			
	1		3		3		1		
1		4		6		4		1	

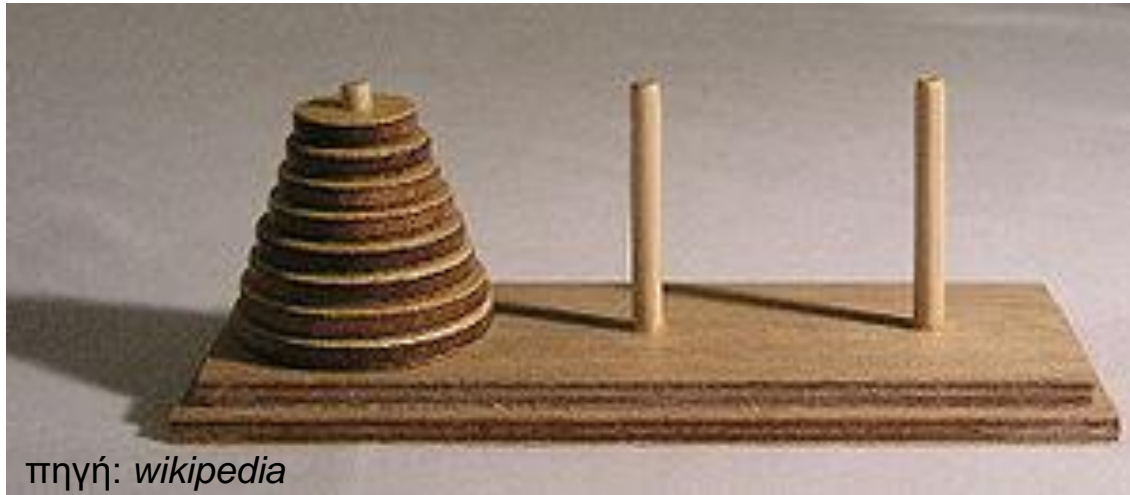
Διωνυμικοί συντελεστές / συνδυασμοί:

$$(a+b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

Αλγοριθμικές τεχνικές

- Επανάληψη (Iteration)
- Αναδρομή (Recursion)
- Επαγωγή (Induction)

Πύργοι Ανόι (Hanoi Towers)



πηγή: *wikipedia*

Πύργοι Ανόι (Hanoi Towers)



πηγή: *wikipedia*

Πύργοι Ανόι (Hanoi Towers): αναδρομή

```
procedure move_anoi(n from X to Y using Z)
begin
    if n = 1 then
        move top disk from X to Y
    else
        move_anoi(n-1 from X to Z using Y);
        move top disk from X to Y;
        move_anoi(n-1 from Z to Y using X)
    end
```

Πύργοι Ανόι (Hanoi Towers): αναδρομή σε Python

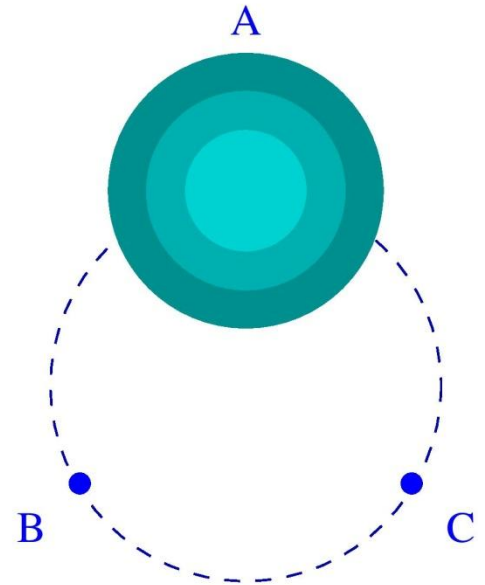
```
def hanoi(ndisks, startPeg, endPeg, usePeg):  
    if ndisks > 0:  
        hanoi(ndisks-1, startPeg, usePeg, endPeg)  
        print ("Move disk", ndisks, "from peg", startPeg,  
              "to peg", endPeg)  
        hanoi(ndisks-1, usePeg, endPeg, startPeg)
```

Δοκιμάστε: `hanoi(5, "X", "Y", "Z")`

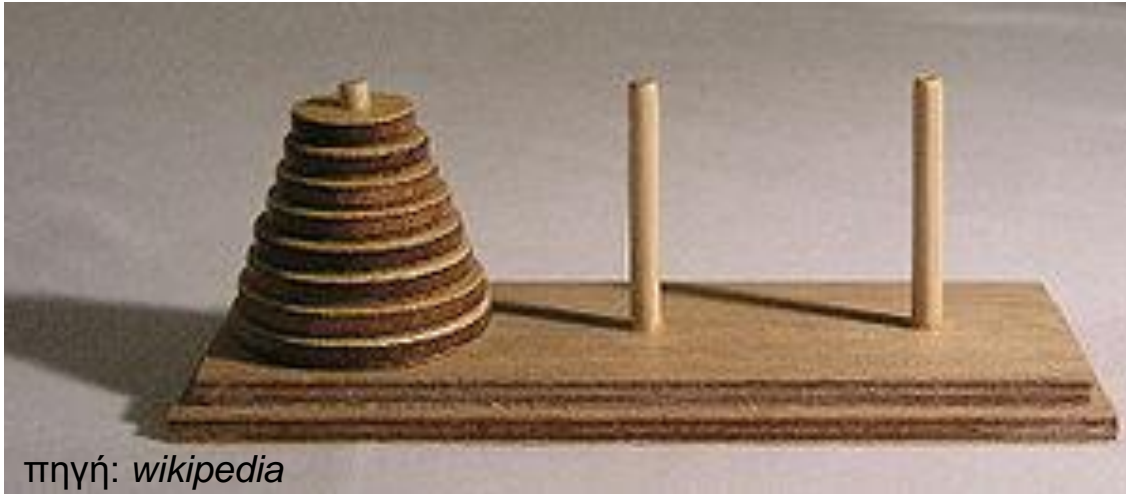
Πύργοι Ανόι (Hanoi Towers): επανάληψη

Επανάλαβε (μέχρι να επιτευχθεί η μετακίνηση):

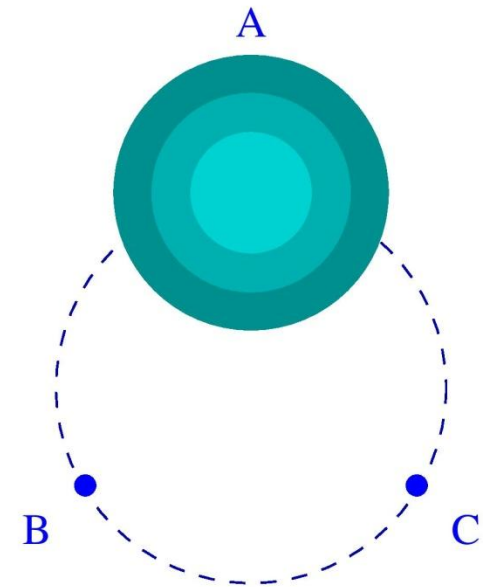
- Μετακίνησε κατά τη θετική φορά τον μικρότερο δίσκο
- Κάνε την μοναδική επιτρεπτή κίνηση που δεν αφορά τον μικρότερο δίσκο



Πύργοι Ανόι (Hanoi Towers)



- Ποιό είναι το πλήθος κινήσεων των δύο τρόπων για n δίσκους;
- Γίνεται καλύτερα;



Υπολογιστικά προβλήματα

- Δίνεται ακέραιος n : πόσο γρήγορα μπορούμε να υπολογίσουμε:
 - το άθροισμα των αριθμών $1 \dots n$
 - αν ο n είναι πρώτος
 - τον n -οστό αριθμό Fibonacci

Και ένα ακόμη...

- Πόσο γρήγορα μπορούμε να υπολογίσουμε:
το **ελάχιστο πλήθος δοκιμών** για να βρούμε σε
ποιο ύψος σπάει ένα γυάλινο αντικείμενο
 - αν μας ενδιαφέρει **ακρίβεια εκατοστού**
 - αν μας ενδιαφέρει ύψος μέχρι **n εκατοστά**
 - αν διαθέτουμε **1** δοκιμαστικό αντικείμενο;
 - αν διαθέτουμε **2** δοκιμαστικά αντικείμενα;
 - αν διαθέτουμε **k** δοκιμαστικά αντικείμενα;



Αποδοτικότητα αλγορίθμου

- Μετράμε το κόστος αλγορίθμου σαν συνάρτηση των υπολογιστικών πόρων που απαιτούνται σε σχέση με το μέγεθος της (αναπαράστασης της) εισόδου στην *χειρότερη περίπτωση*:

$$\text{cost}_A(n) = \max \{ \text{κόστος αλγορίθμου } A \text{ για είσοδο } x \}$$

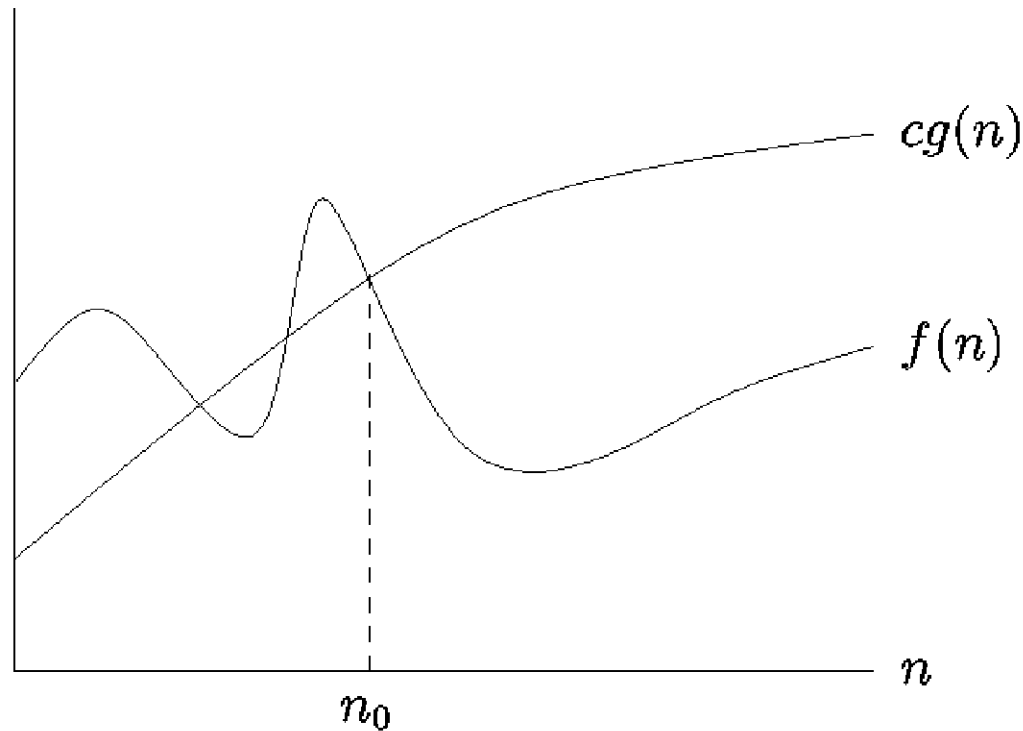
μεταξύ όλων
των εισόδων
x μήκους n

- Παράδειγμα: $\text{time-cost}_{\text{MS}}(n) \leq c n \log n$
(MS = MergeSort, c μία σταθερά)

Αποδοτικότητα αλγορίθμου

- Συνήθως μας ενδιαφέρει το κόστος σε χρόνο, ή αλλιώς η **χρονική πολυπλοκότητα**.
- Ενδιαφέρον παρουσιάζει και το κόστος σε χώρο, ή αλλιώς **χωρική πολυπλοκότητα**.
- Παράδειγμα: $\text{space-cost}_{\text{MS}}(n) \leq c' n$
(MS = MergeSort, c' κάποια σταθερά)

Ασυμπτωτικός συμβολισμός (i)



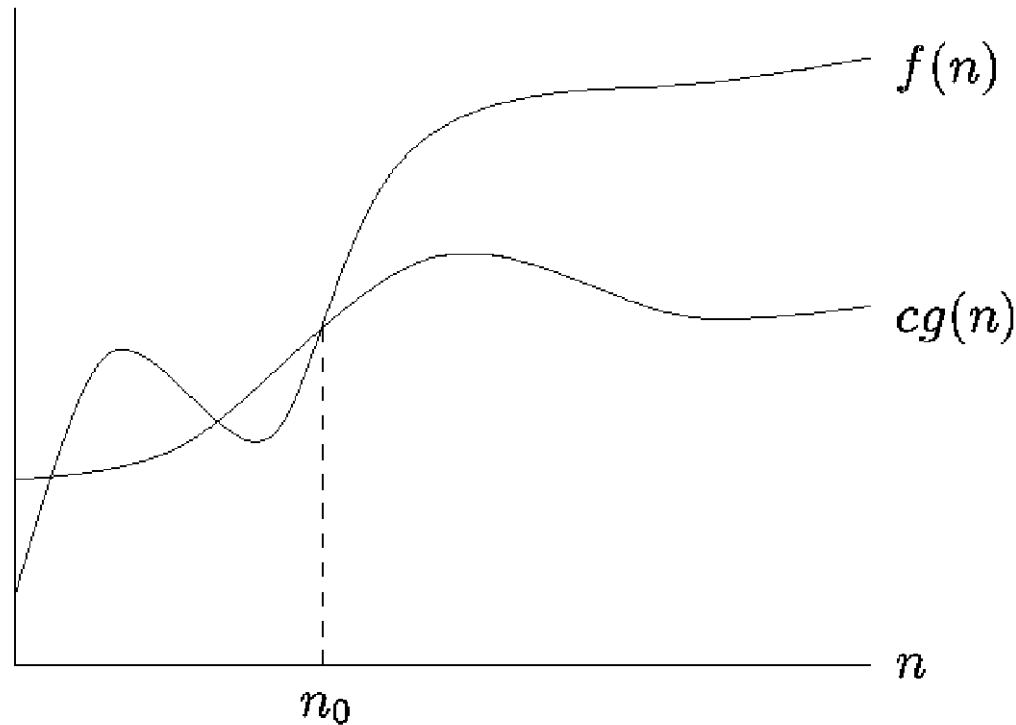
$$f = O(g)$$

$$O(g) = \{f \mid \exists c > 0, \exists n_0 : \forall n > n_0 \ f(n) \leq cg(n)\}$$

Συμβολισμός O : παραδείγματα

- BubbleSort: $T_{BS}(n) = O(n^2)$
- InsertionSort: $T_{IS}(n) = O(n^2)$
- MergeSort: $T_{MS}(n) = O(n \log n)$
- Προσοχή: πολυπλοκότητα χειρότερης περίπτωσης

Ασυμπτωτικός συμβολισμός (ii)



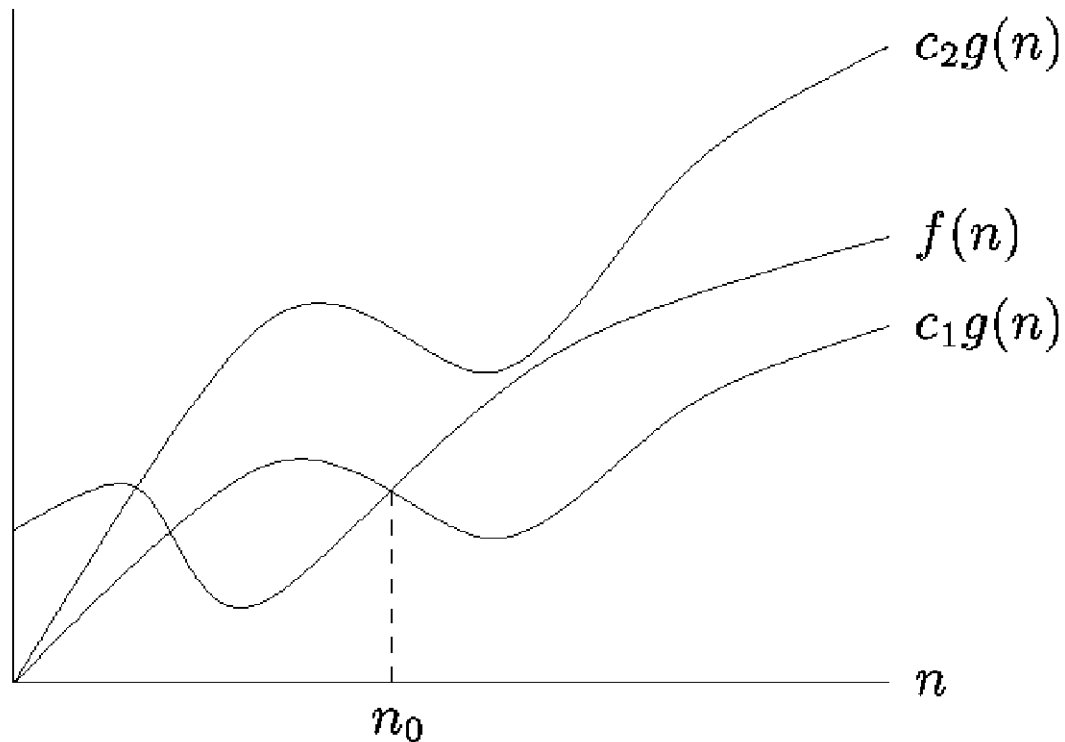
$$f = \Omega(g)$$

$$\Omega(g) = \{f \mid \exists c > 0, \exists n_0 : \forall n > n_0 \ f(n) \geq cg(n)\}$$

Συμβολισμός Ω : παραδείγματα

- BubbleSort: $T_{BS}(n) = \Omega(n^2)$
- InsertionSort: $T_{IS}(n) = \Omega(n^2)$
- MergeSort: $T_{MS}(n) = \Omega(n \log n)$
- Προσοχή: πολυπλοκότητα χειρότερης περίπτωσης

Ασυμπτωτικός συμβολισμός (iii)



$$f = \Theta(g)$$

$$\Theta(g) = \left\{ f \mid \exists c_1 > 0, \exists c_2 > 0, \exists n_0 : \forall n > n_0 \quad c_1 \leq \frac{f(n)}{g(n)} \leq c_2 \right\}$$

Συμβολισμός Θ : παραδείγματα

- BubbleSort: $T_{BS}(n) = \Theta(n^2)$
- InsertionSort: $T_{IS}(n) = \Theta(n^2)$
- MergeSort: $T_{MS}(n) = \Theta(n \log n)$
- Προσοχή: πολυπλοκότητα χειρότερης περίπτωσης

Ασυμπτωτικός συμβολισμός : συμβάσεις και ιδιότητες

- Γράφουμε: $g(n) = O(f(n))$ αντί για $g(n) \in O(f(n))$
- $\Theta(f) = O(f) \cap \Omega(f)$
- $p(n) = \Theta(n^k)$, για κάθε πολυώνυμο p
- $O(poly) = \bigcup O(n^k)$ (για όλα τα $k \in \mathbb{N}$)

Ασυμπτωτικός συμβολισμός : συμβάσεις και ιδιότητες

$$\begin{aligned} O(1) &< O(\alpha(n)) < O(\log^* n) \\ &< O(\log(n)) < O(\sqrt{n}) < O(n) \\ &< O(n \log(n)) < O(n^2) < \dots < O(\text{poly}) \\ &< O(2^n) < O(n!) < O(n^n) < O(A(n)) \end{aligned}$$

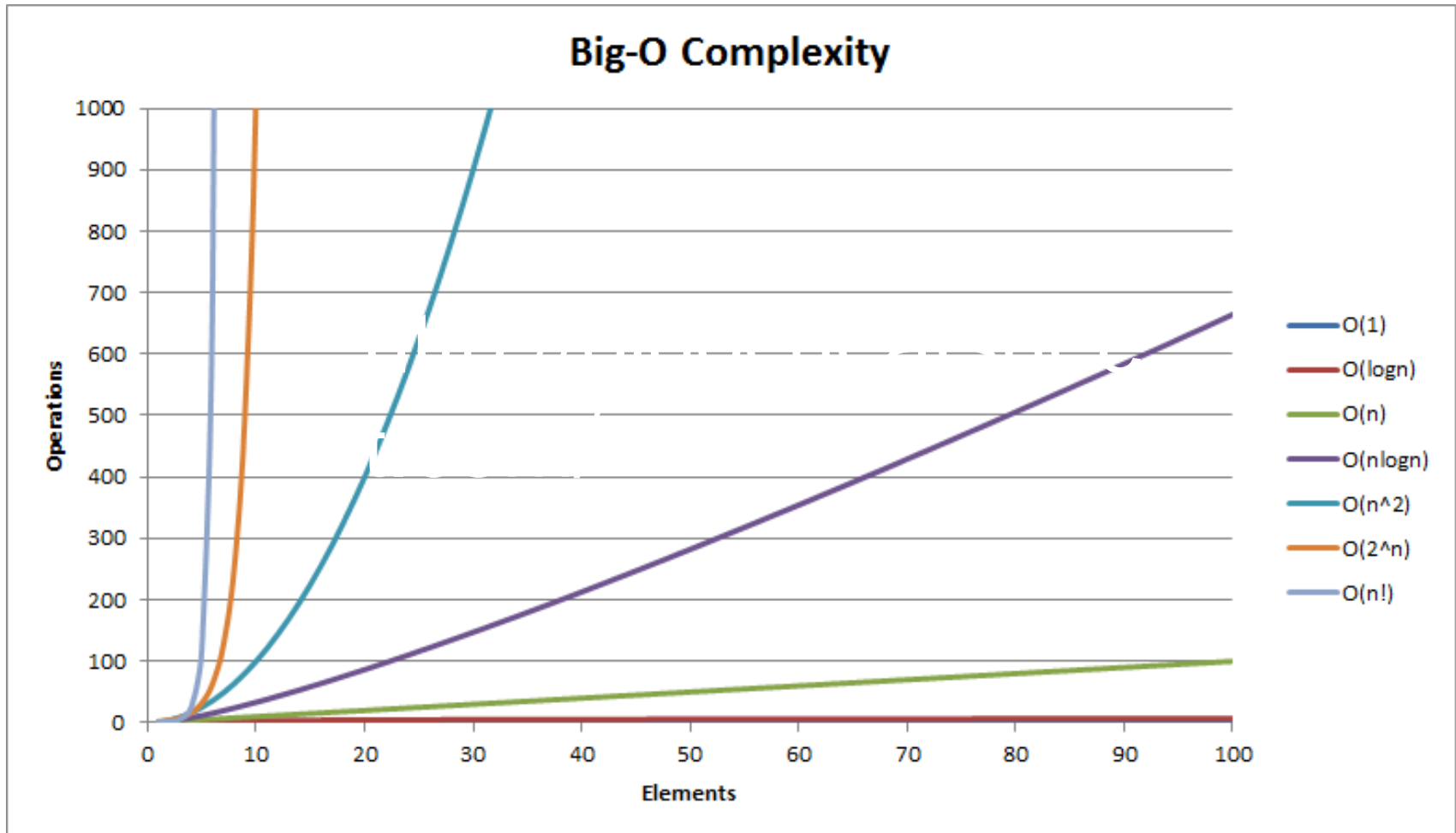
Σημείωση: γράφουμε “<” αντί “⊂”.

log* n: πόσες φορές πρέπει να λογαριθμήσουμε το n για να φτάσουμε κάτω από το 1 (αντίστροφη υπερεκθετικής)

A: Ackermann.

α: αντίστροφη της A.

Γιατί ασυμπτωτικός συμβολισμός;



Source: bigocheatsheet.com/

Ασυμπτωτικός συμβολισμός: απόδειξη φραγμάτων

Θεώρημα. $\log(n!) = \Theta(n \log n)$

Απόδειξη: **ασυμπτωτικά** (για $n > n_0$) ισχύει:

$$(n/2)^{n/2} < n! < n^n \Rightarrow$$

$$(1/2) n (\log n - 1) < \log(n!) < n \log n \Rightarrow$$

$$(1/3) n \log n < \log(n!) < n \log n$$

Προσοχή: μπορείτε να χρησιμοποιήσετε κανόνα de l'Hospital, αλλά συνήθως γίνεται απλούστερα!

Πολυπλοκότητα αλγορίθμων: απλοποιήσεις

- Συχνά θεωρούμε ως **μέγεθος της εισόδου** το **πλήθος των στοιχείων εισόδου** μόνο (αγνοώντας το μέγεθός τους σε bits):
- Ικανοποιητική εκτίμηση, αν τυχόν αριθμοί εισόδου είναι **«μικροί»** σε σχέση με υπόλοιπη είσοδο
- ή αν είναι **«μεγάλοι»** αλλά η τιμή τους δεν επηρεάζει ιδιαίτερα το πλήθος των στοιχειωδών πράξεων: π.χ. **ταξινόμηση με συγκρίσεις**

Πολυπλοκότητα αλγορίθμων: απλοποιήσεις

- Θεωρούμε ακόμη ότι κάθε **στοιχειώδης** αριθμητική πράξη (πρόσθεση, πολ/σμός, σύγκριση) έχει **μοναδιαίο κόστος** (1 βήμα):

αυτό λέγεται **αριθμητική πολυπλοκότητα** (**arithmetic complexity**) και είναι συνήθως ικανοποιητική εκτίμηση

- η εκτίμηση της **πολυπλοκότητας ψηφιοπράξεων** (**bit complexity**) είναι απαραίτητη όταν οι αριθμοί «μεγαλώνουν» πολύ κατά τη διάρκεια εκτέλεσης: π.χ. ύψωση σε δύναμη, n -οστός Fibonacci

Πολυπλοκότητα προβλήματος

- Είναι η πολυπλοκότητα του βέλτιστου αλγορίθμου που λύνει το πρόβλημα.

$$\text{cost}_{\Pi}(n) = \min \{ \text{cost}_A(n) \}$$

μεταξύ όλων των αλγορίθμων

A που επιλύουν το Π

- Παράδειγμα: **time-cost**_{SORT}(n) = $O(n \log n)$
(SORT = πρόβλημα ταξινόμησης)
- Για να δείξουμε *βελτιστότητα αλγορίθμου* χρειάζεται και *απόδειξη αντίστοιχου κάτω φράγματος*: $\Omega(n \log n)$

Ανάλυση χρονικής πολυπλοκότητας αλγορίθμων

Μέτρηση βημάτων που θα εκτελεστούν:

- είτε με απευθείας άθροιση πλήθους βημάτων (επαναληπτικοί αλγόριθμοι)
 - Π.χ.: $T_{BS}(n) \leq c n^2 = O(n^2)$
(BS = BubbleSort, c κάποια σταθερά)
- είτε με επίλυση αναδρομικών σχέσεων (αναδρομικοί αλγόριθμοι)
 - Π.χ.: $T_{MS}(n) \leq 2T_{MS}(n/2) + cn = \dots = O(n \log n)$
(MS = MergeSort, c κάποια σταθερά)

Πίνακας χρονικής πολυπλ/τας

$O(1)$	$a := b * c;$	απλές εντολές
$O(\log n)$	if $x < A[n/2]$ search($A[1, n/2]$)...	δυναδική αναζήτηση
$O(n)$	for $i := 1$ to n do $\langle O(1) \rangle$	απλός βρόχος
$O(n \log n)$	mergesort($A[1, n/2]$) mergesort($A[n/2+1, n]$) merge($A[1, n/2], A[n/2+1, n]$)	ταξινόμηση με συγχώνευση
$O(n^2)$	for $i := 1$ to n do for $j := 1$ to n do $\langle O(1) \rangle$	διπλός βρόχος
$O(2^n)$	for all $S \subseteq \{a_1, \dots, a_n\}$ do $\langle O(1) \rangle$	υποσύνολα
$O(n!)$	for all σ in $S[n]$ do $\langle O(1) \rangle$	μεταθέσεις

Αριθμητικοί υπολογισμοί

- Εύρεση ΜΚΔ (Ευκλείδειος αλγόριθμος)
- Ύψωση σε δύναμη
- Αριθμοί Fibonacci
- Πολλαπλασιασμός ακεραίων
- Divide-and-Conquer
- Επίλυση αναδρομών: **master theorem**

Εύρεση Μέγιστου Κοινού Διαιρέτη (gcd)

Δεν είναι λογικό να ανάγεται στο πρόβλημα εύρεσης πρώτων παραγόντων γιατί αυτό δεν λύνεται αποδοτικά.

Απλός αλγόριθμος: $O(\min(a,b))$

```
z := min(a,b)
```

```
while (a mod z ≠ 0) or (b mod z ≠ 0) do z := z-1
```

Αλγόριθμος με αφαιρέσεις: $O(\max(a,b))$

```
i := a ; j := b
```

```
while (i ≠ j) do if i > j then i := i - j else j := j - i
```

```
return i
```

Αλγόριθμος του Ευκλείδη: $O(\log(a+b))$

```
i := a ; j := b
```

```
while (i > 0) and (j > 0) do
```

```
    if i > j then i := i mod j else j := j mod i
```

```
return i + j
```

Εύρεση Μέγιστου Κοινού Διαιρέτη (gcd): υλοποίηση με αναδρομή

Αλγόριθμος με αφαιρέσεις: $O(\max(a,b))$

```
if a=b then GCD(a,b):=a
else if a>b then GCD(a,b):= GCD(a-b,b)
      else GCD(a,b):= GCD(a,b-a)
```

Αλγόριθμος του Ευκλείδη: $O(\log(a+b))$

```
if b=0 then GCD(a,b):= a
      else GCD(a,b):= GCD(b, a mod b)
```

Εύρεση Μέγιστου Κοινού Διαιρέτη (gcd): υλοποίηση με αναδρομή

Αλγόριθμος του Ευκλείδη: $O(\log(a+b))$

```
if b=0 then GCD(a,b) := a
    else GCD(a,b) := GCD(b, a mod b)
```

$$\text{ΜΚΔ}(6887, 6319) =$$

$$\text{ΜΚΔ}(6319, 568) =$$

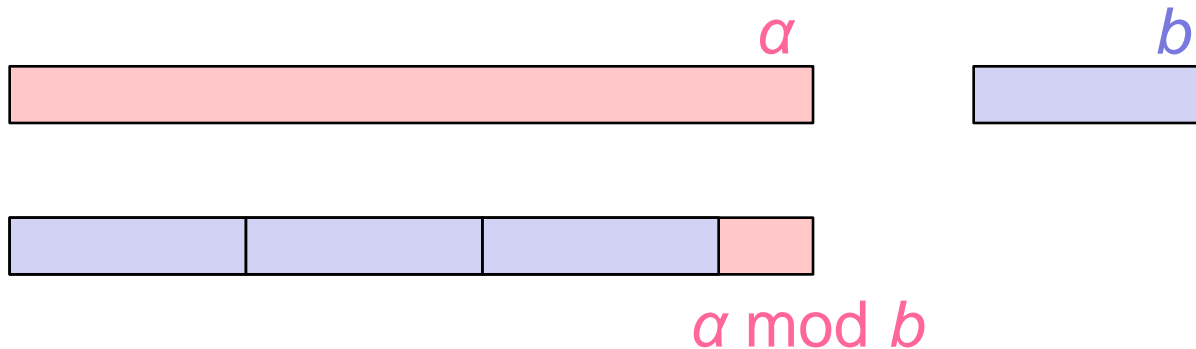
$$\text{ΜΚΔ}(568, 71) =$$

$$\text{ΜΚΔ}(71, 0) = \mathbf{71}$$

Πολυπλοκότητα Ευκλείδειου

- $O(\log \max(a, b))$: σε κάθε 2 επαναλήψεις το πολύ ο μεγαλύτερος αριθμός υποδιπλασιάζεται:

- Περίπτ. 1. αρχικά: (a, b) , $b \leq a/2$



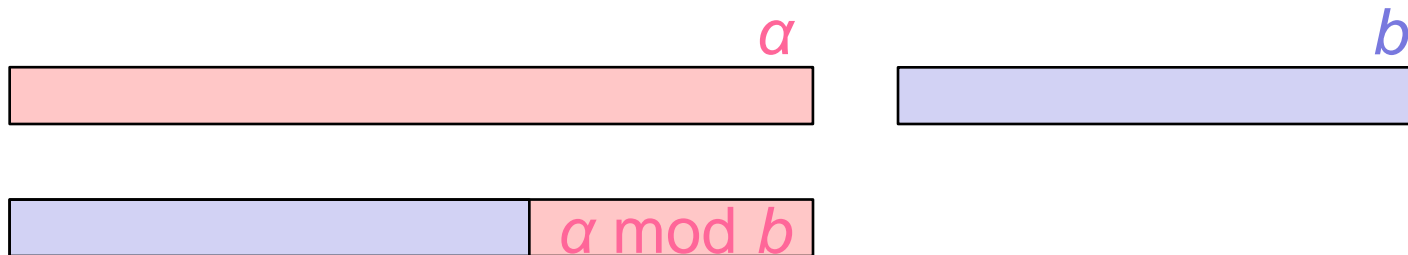
- σε 1 επανάληψη: $(b, a \bmod b)$



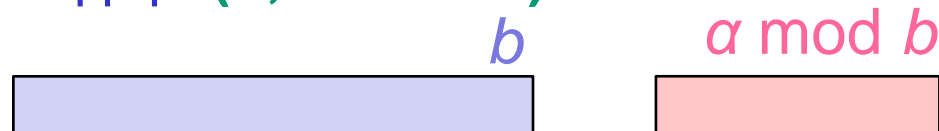
Πολυπλοκότητα Ευκλείδειου

- $O(\log \max(a, b))$: σε κάθε 2 επαναλήψεις το πολύ ο μεγαλύτερος αριθμός υποδιπλασιάζεται:

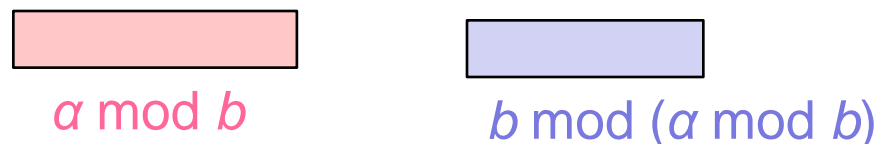
- Περίπτ. 2. αρχικά: (a, b) , $b > a/2$



- σε 1 επανάληψη: $(b, a \bmod b)$



- σε 2 επαναλήψεις: $(a \bmod b, b \bmod (a \bmod b))$



Πολυπλοκότητα Ευκλείδειου Αλγόριθμου

- $O(\log \max(a,b))$: σε κάθε 2 επαναλήψεις το πολύ ο μεγαλύτερος αριθμός υποδιπλασιάζεται
- $\Omega(\log \max(a,b))$: για ζεύγη διαδοχικών αριθμών Fibonacci F_{k-1}, F_k , ο αλγόριθμος κάνει k επαναλήψεις (γιατί;), και $k = \Theta(\log F_k)$, αφού $F_k \approx \varphi^k / \sqrt{5}$,
($\varphi = (1+\sqrt{5})/2$: χρυσή τομή)
- Επομένως: $\Theta(\log \max(a,b)) = \Theta(\log (a+b))$ [αριθμητική πολυπλοκότητα]
- Bit complexity: $O(\log^3(a+b))$
- Βελτι/νη ανάλυση: $O(\log^2(a+b))$ // βλ. & binary gcd [Stein]

Ύψωση σε δύναμη

```
power(a, n)
  result := 1;
  for i := 1 to n do
    result := result*a;
  return result
```

Πολυπλοκότητα: $O(n)$ – εκθετική! (γιατί;)

Ύψωση σε δύναμη

```
power(a, n)
  result := 1;
  for i := 1 to n do
    result := result*a;
  return result
```

Πολυπλοκότητα: $O(n)$ – εκθετική! (γιατί;)

*ως προς το μήκος της εισόδου: $O(2^{|n|})$
(δυναμική αναπαράσταση)*

... με επαναλαμβανόμενο τετραγωνισμό (Gauss)

```
fastpower(a, n)
  result := 1;
  while n > 0 do
    if odd(n) then result := result * a;
    n := n div 2;
    a := a * a
  return result
```

Ιδέα: $a^{13} = a^{8+4+1} = a^8 a^4 a^1 = a^{1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0}$

Πολυπλοκότητα: $O(\log n)$ - πολυωνυμική

ως προς το μήκος της εισόδου: $O(\|n\|)$

Παράδειγμα σε Python

```
def fastpower(a,n):  
    res=1  
    while (n>0):  
        if (n%2==1):  
            res=res*a  
        print (n, a, res)  
        n=n//2  
        a=a*a  
    return res
```

Εκτέλεση: `print (fastpower(15, 507))`

Παράδειγμα σε Python: εκτέλεση

```
Python 3.5.2 (default, Dec 2015, 13:05:11) [GCC 4.8.2] on
linux

15 ^ 507 computation
-----
n a res
-----
507 15 15
253 225 3375
126 50625 3375
63 2562890625 8649755859375
31 6568408355712890625 56815128661595284938812255859375
15 43143988327398919500410556793212890625
245123124779553476933979997334084321991554134001489728689193
7255859375
7
186140372879473421546741060475570282012336420507381262723356
4853668212890625
456273098478477754404871005449560512605808364897244527162041
372045715025563297070224521967231717463114783889244208126001
```

Παράδειγμα σε Python: εκτέλεση

```
4674626290798187255859375
1
120050042531184094299874716051889779577766242877999774926621
511666739054560393666116288901943297970025344754226257166624
937808359064757447892755473928062711328282864028330039243822
312184801024809026818185212475825302998589583459405014766541
044631750295336452974762075918135906249517574906349182128906
25
189787821718375110360762239350954129966731376201805406772602
442272623305647775089332670743514799079383484405610052572959
489905303006182040948649980490955066029181857035802223386098
036059919330482483657720145859824904754399181629709999530085
887946894416436785712070915319327998703108254211607359805831
177526935506324973606577448389957910588756698435943358452701
472306966001783648996571417819411171853449747320497842238598
203675133090389640160833787160530771936112655849181000704050
329471980267281185338842889154689834276189225020172717011238
228371563475037862855909764903117320500314235687255859375
```

Παράδειγμα σε Python: εκτέλεση

```
15 ^ 507 =  
189787821718375110360762239350954129966731376201805406772602  
442272623305647775089332670743514799079383484405610052572959  
489905303006182040948649980490955066029181857035802223386098  
036059919330482483657720145859824904754399181629709999530085  
887946894416436785712070915319327998703108254211607359805831  
177526935506324973606577448389957910588756698435943358452701  
472306966001783648996571417819411171853449747320497842238598  
203675133090389640160833787160530771936112655849181000704050  
329471980267281185338842889154689834276189225020172717011238  
228371563475037862855909764903117320500314235687255859375
```

Bit complexity για a^n ;

- ‘Αφελής’ αλγόριθμος: $O(n^2 \|a\|^2) = O(4^{\|n\|} \|a\|^2)$
 i -οστή επανάληψη: $O((i-1) \|a\|^2)$
 $\|a\|$ = μήκος του αριθμού a σε bits
- Αλγόριθμος τετραγωνισμού: $O(n^2 \|a\|^2)$ επίσης!
(γιατί;)
- Τετραγωνισμός με πολλαπλασιασμό Gauss-Karatsuba:
 $O(n^{\log 3} \|a\|^{\log 3}) = O(3^{\|n\|} \|a\|^{1.59})$ [προσεχώς!]
- Περαιτέρω βελτιώσεις (Schönhage–Strassen):
 $O(n \|a\| \log n \log \log n) = O(2^{\|n\|} \|a\| \|n\| \log(\|n\|))$

Modular exponentiation $a^n \bmod p$

```
fastmodpower(a,n,p)
  result := 1;
  while n>0 do
    if odd(n) then res:=res*a mod p;
    n := n div 2;
    a := a*a mod p
  return res
```

Arithmetic complexity: $O(\log n) = O(\|n\|)$

Bit complexity: $O(\log n \log^2 p)$ - πολυωνυμική

ως προς το μήκος της εισόδου: $O(\|n\| \cdot \|p\|^2)$

Παράδειγμα σε Python

```
def fastmodpower(a,n,p):  
    res=1  
    while (n>0):  
        if (n%2==1):  
            res=res*a % p  
        print (n, a, res)  
        n=n//2  
        a=a*a % p  
    return res
```

Εκτέλεση: `print (fastmodpower(15,126,127))`

(Fermat primality test: $a^{n-1} \bmod n =? 1$)

Παράδειγμα σε Python: εκτέλεση εκτέλεσης

```
Python 3.5.2 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
```

```
126 15 1
63 98 98
31 79 122
15 18 37
7 70 50
3 74 17
1 15 1
1
```

Αριθμοί Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

$$F_0 = 0, F_1 = 1$$

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

- Πρόβλημα: Δίνεται n , να υπολογιστεί το F_n
- Πόσο γρήγορο μπορεί να είναι το πρόγραμμά μας;

Αριθμοί Fibonacci – αναδρομικός αλγόριθμος

```
Fib1(n)  
  if (n<2) then return n  
  else return Fib1(n-1)+Fib1(n-2)
```

- Πολυπλοκότητα: $T(n) = T(n-1) + T(n-2) + c$,
δηλ. η $T(n)$ ορίζεται όπως η $F(n)$ (συν μια σταθερά),
οπότε:

$$T(n) = c' \cdot F(n) = \Omega(1.618^n) = \Omega(1.618^{2\lfloor n/2 \rfloor})$$

Αριθμοί Fibonacci – καλύτερος αλγόριθμος

```
Fib2(n)  
  a:=0; b:=1;  
  for i:=2 to n do  
    c:=b; b:=a+b; a:=c;  
  return b
```

- Πολυπλοκότητα: $O(n)$
- Είναι πολυωνυμική;

Αριθμοί Fibonacci – καλύτερος αλγόριθμος

```
Fib2(n)
  a:=0; b:=1;
  for i:=2 to n do
    c:=b; b:=a+b; a:=c;
  return b
```

- Πολυπλοκότητα: $O(n)$
- Είναι πολυωνυμική; **Όχι!** : $O(2^{\lfloor n/2 \rfloor})$

Αριθμοί Fibonacci – ακόμα καλύτερος αλγόριθμος

Μπορούμε να γράψουμε τον υπολογισμό σε μορφή πινάκων:

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

Από αυτό συμπεραίνουμε:

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

... και το πλήθος των αριθμητικών πράξεων (*αριθμητική πολυπλοκότητα*) μειώνεται σε $O(\log n) = O(\|n\|)$

Αριθμοί Fibonacci – ακόμα καλύτερος αλγόριθμος

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Άσκηση: βρείτε την *πολυπλοκότητα ψηφιοπράξεων (bit complexity)* του παραπάνω αλγορίθμου. Συγκρίνετε με τον επαναληπτικό αλγόριθμο.

Σημαντική αλγοριθμική τεχνική

Ποια ιδέα είναι κοινή και στους 3 προηγούμενους αλγόριθμους (Ευκλείδη, Repeated Squaring, Fibonacci με πίνακα);

Divide-and-Conquer!

(Διαίρει-και-Κυρίευε)

Divide-and-Conquer

- Χρόνος εκτέλεσης αλγόριθμων «διαίρει-και-κυρίευε» με διατύπωση και λύση αναδρομικής εξίσωσης λειτουργίας.
- **MergeSort**
 - $T(n)$: χρόνος για ταξινόμηση n στοιχείων.
 - $T(n/2)$: ταξινόμηση αριστερού τμήματος ($n/2$ στοιχεία).
 - $T(n/2)$: ταξινόμηση δεξιού τμήματος ($n/2$ στοιχεία).
 - $O(n)$: συγχώνευση ταξινομημένων τμημάτων.

$$T(n) = 2 T(n/2) + O(n), T(1) = O(1)$$

- Χρόνος εκτέλεσης MergeSort: $T(n) = ?$

Δέντρο Αναδρομής

$$T(n) = 2 T(n/2) + O(n),$$
$$T(1) = O(1)$$

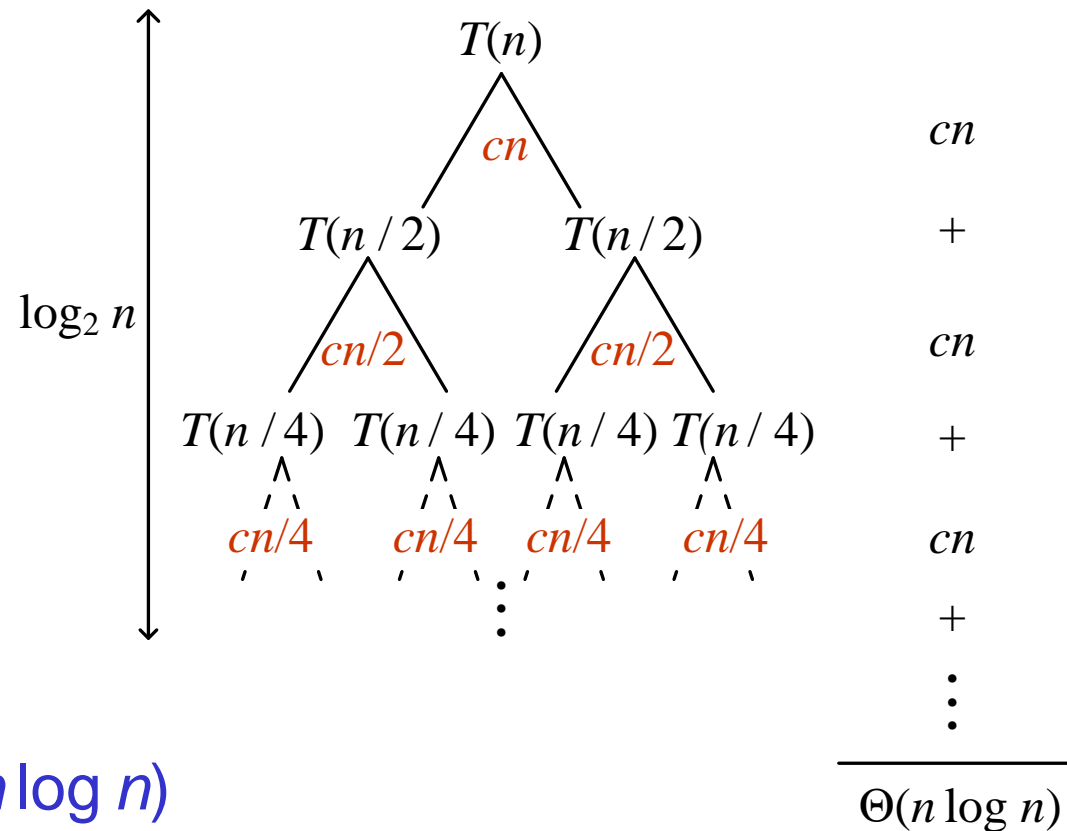
Δέντρο αναδρομής :

Ύψος : $O(\log n)$

#κορυφών : $O(n)$

Χρόνος / επίπεδο : $O(n)$

Συνολικός χρόνος : $O(n \log n)$



Πολλαπλασιασμός Ακεραίων

$$X : \begin{array}{|c|c|} \hline & \\ \hline a & b \\ \hline \end{array} = a \cdot 2^{\frac{n}{2}} + b$$

$$Y : \begin{array}{|c|c|} \hline & \\ \hline c & d \\ \hline \end{array} = c \cdot 2^{\frac{n}{2}} + d$$

$$X Y = ac \cdot 2^n + (ad + bc) \cdot 2^{\frac{n}{2}} + bd$$

Πολυπλοκότητα Πολλαπλασιασμού

$$T(n) = \begin{cases} a & , \text{για } n = 1 \\ 4T\left(\frac{n}{2}\right) + cn & , \text{για } n > 1 \end{cases}$$

Απόδειξη:

$T(n)$

$T(n/2)$

$T(n/2)$

$T(n/2)$

$T(n/2)$

$T(n/4)$

$T(n/4)$

.....

.....

.....

.....

.....

$T(2)$

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

$T(1)$

$T(1)$

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

$+ cn$

$+ 4 cn/2$

$+ 16 cn/4$

⋮

⋮

⋮

$+ 4^{(k-1)} cn/2^{(k-1)}$

Χρον.
πολ/τα

$< (4/2)^k cn$

$\leq 2^{(\log n + 1)} cn$

$= O(n^2)$

Συνολικά: $O(n^2)$

Ύψος
δένδρου

$k = \lceil \log n \rceil$

4^k 'φύλλα', χρονική πολυπλ/τα $\alpha \cdot 4^k = O(n^2)$

Πολυπλοκότητα Πολλαπλασιασμού

$$T(n) = \begin{cases} a & , \text{για } n = 1 \\ 4T\left(\frac{n}{2}\right) + cn & , \text{για } n > 1 \end{cases}$$

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

Βελτιωμένος Πολλαπλασιασμός (Gauss-Karatsuba)

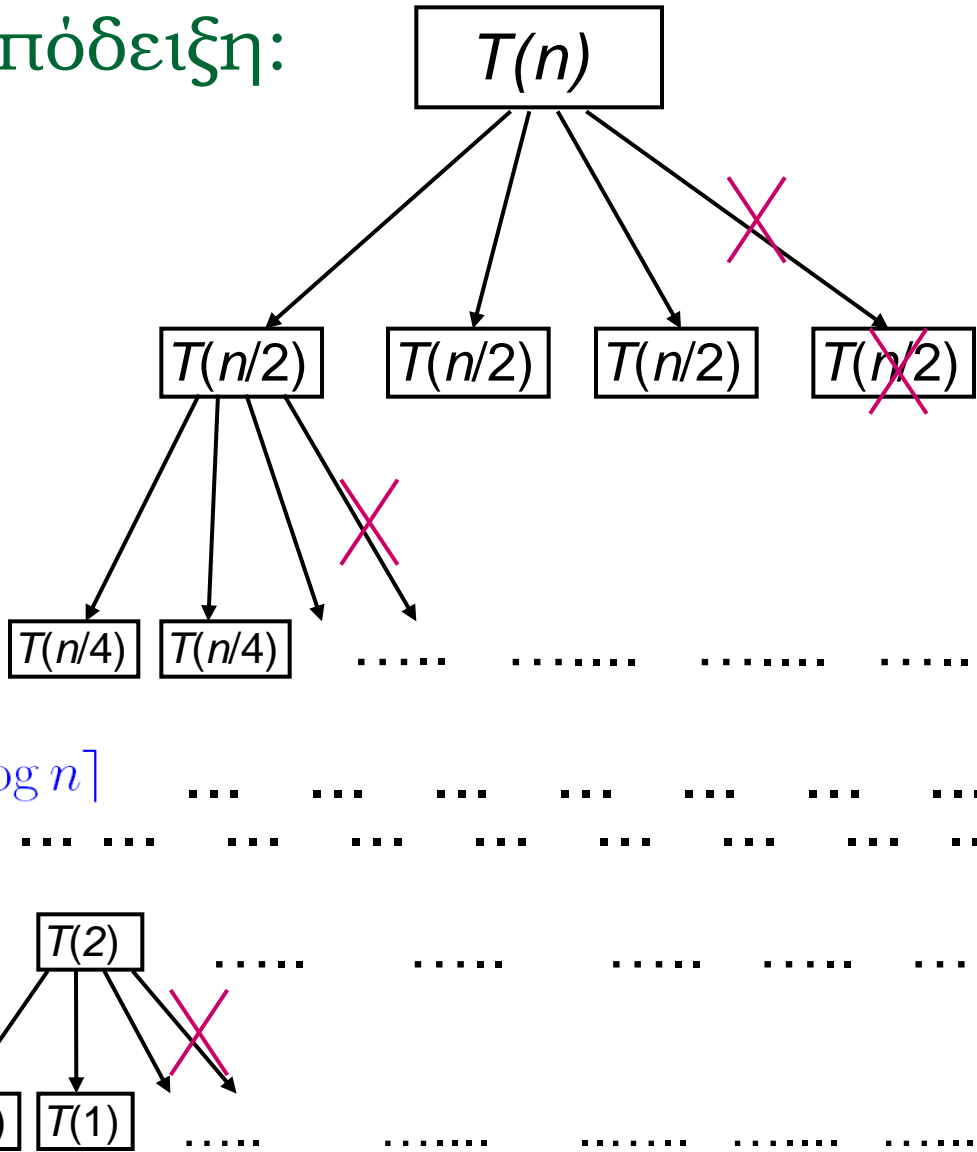
$$(ad + bc) = [(a - b)(d - c) + ac + bd]$$

$$X Y = ac \cdot 2^n + [(a - b)(d - c) + ac + bd] 2^{\frac{n}{2}} + bd$$

Πολυπλοκότητα Βελτίωσης

$$T(n) = \begin{cases} a & , \text{για } n = 1 \\ 3T\left(\frac{n}{2}\right) + cn & , \text{για } n > 1 \end{cases}$$

Απόδειξη:



$+ cn$

$+ 3 c n/2$

$+ 9 c n/4$

$+ 3^{(k-1)} c n/2^{(k-1)}$

Χρον.
πολ/τα

$< 2 \cdot (3/2)^k cn$

$\leq 3 \cdot (3/2)^{(\log n)} cn$

$= O(n^{\log 3})$

Συνολικά: $O(n^{\log 3})$

3^k 'φύλλα', χρονική πολυπλ/τα $\alpha \cdot 3^k = O(3^{\log n}) = O(n^{\log 3})$

Πολυπλοκότητα Βελτίωσης

$$T(n) = \begin{cases} a & , \text{για } n = 1 \\ 3T\left(\frac{n}{2}\right) + cn & , \text{για } n > 1 \end{cases}$$

$$T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

Master Theorem (απλή μορφή)

Αν $T(n) = aT(n/b) + O(n)$,

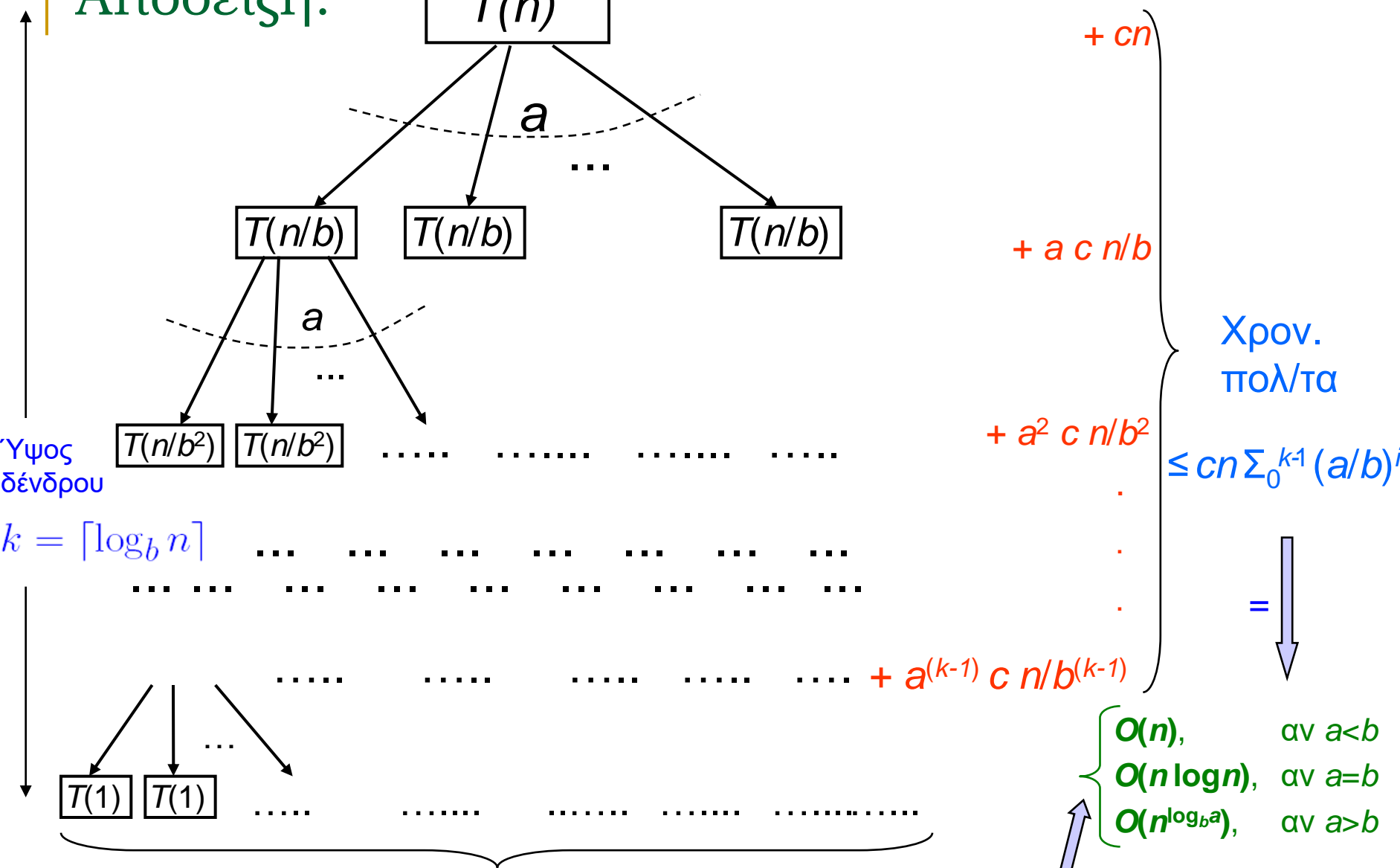
για θετικούς ακέραιους a, b

και $T(1) = O(1)$

τότε:

$$T(n) = \begin{cases} O(n), & \text{αν } a < b \\ O(n \log n), & \text{αν } a = b \\ O(n^{\log_b a}), & \text{αν } a > b \end{cases}$$

Απόδειξη:



Υψος δένδρου

$k = \lceil \log_b n \rceil$

a^k 'φύλλα', χρονική πολυπλ/τα $c \cdot a^k = O(a^{\log_b n}) = O(n^{\log_b a}) \leq$

Master Theorem (γενική μορφή)

Αν $T(n) = aT(n/b) + O(n^d)$,

για θετικούς ακέραιους a, b, d

και $T(1) = O(1)$

Τότε:

$$T(n) = \begin{cases} O(n^d), & \text{αν } a < b^d \\ O(n^d \log n), & \text{αν } a = b^d \\ O(n^{\log_b a}), & \text{αν } a > b^d \end{cases}$$

Master Theorem: εφαρμογή

Αν $T(n) = aT(n/b) + O(n^d)$, για θετικούς ακέραιους a, b, d

και $T(1) = O(1)$ τότε:

$$T(n) = \begin{cases} O(n^d), & \text{αν } a < b^d \\ O(n^d \log n), & \text{αν } a = b^d \\ O(n^{\log_b a}), & \text{αν } a > b^d \end{cases}$$

Matrix Multiplication

• 'Standard' divide-and-conquer: $T(n) = 8T(n/2) + O(n^2)$

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

• Strassen's algorithm:

$$T(n) = 7T(n/2) + O(n^2)$$

$$\Rightarrow T(n) = O(n^{\log_2 7})$$

Αλγόριθμοι divide & conquer

$O(\log n)$ if $x < A[n/2]$ search($A[1, n/2]$)... δυαδική αναζήτηση

$O(\|a\| + \|b\|)$ * $\text{GCD}(a, b) := \text{GCD}(b, a \bmod b)$ εύρεση ΜΚΔ

$O(\|n\|)$ * $\text{pow}(a, n) := \text{pow}(a^2, n/2)$ ύψωση σε δύναμη

$O(\|n\|)$ * αλγόριθμος πίνακα fast doubling υπολογισμός n -οστού αριθμού Fibonacci

$O(n \log n)$ mergesort($A[1, n/2]$) ταξινόμηση
mergesort($A[n/2+1, n]$) με συγχώνευση
merge($A[1, n/2], A[n/2+1, n]$)

$O(n^{\log 3})$ αλγόριθμος Gauss-Karatsuba πολλαπλασιασμός n -ψήφιων αριθμών

$O(n^{\log 7})$ αλγόριθμος Strassen πολλαπλασιασμός πινάκων $n \times n$

* αριθμητική πολυπλοκότητα, $\|x\| = \# \text{ψηφίων του } x = O(\log x)$