
Θεμελιώδη Θέματα

Επιστήμης Υπολογιστών

ΣΗΜΜΥ – ΣΕΜΦΕ ΕΜΠ

Ενότητα 4η:

Αριθμητικοί υπολογισμοί

Επιμέλεια διαφανειών: [Στάθης Ζάχος](#), [Άρης Παγουρτζής](#)

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 $(a \bmod z \neq 0)$ **and** $(b \bmod z \neq 0)$ **do** $z := z-1$

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

$i := a ; j := b$

while $(i \neq j)$ **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** $i := i \bmod j$ **else** $j := j \bmod i$

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)
```

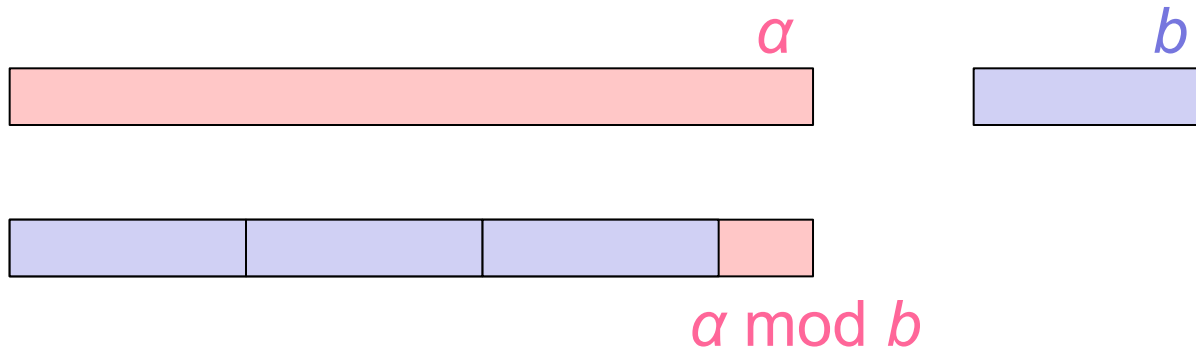
Αλγόριθμος του Ευκλείδη: $O(\log(a+b))$

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

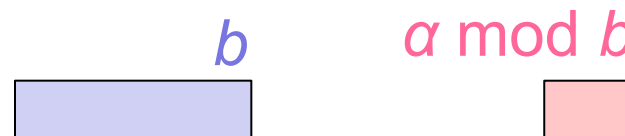
Πολυπλοκότητα Ευκλείδειου

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

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



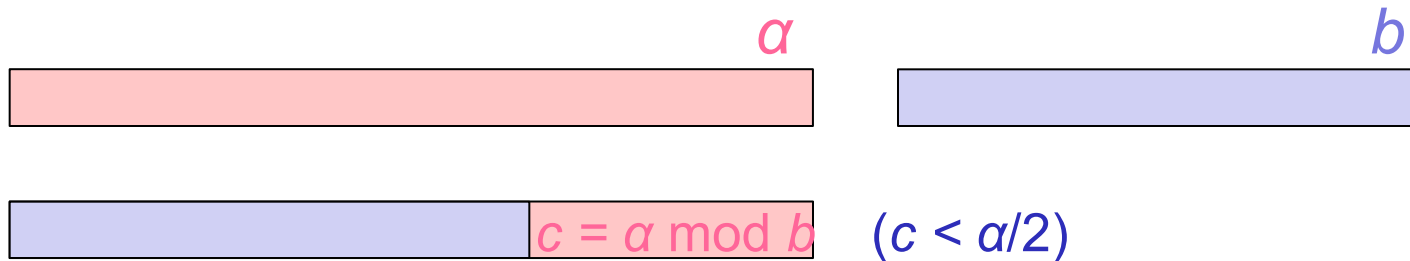
- σε 1 επανάληψη: $(b, a \bmod b)$



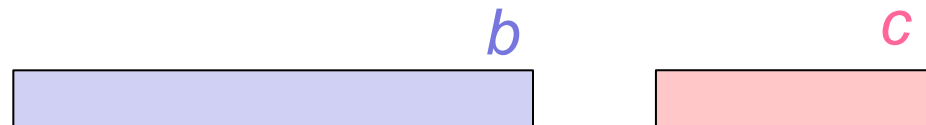
Πολυπλοκότητα Ευκλείδειου

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

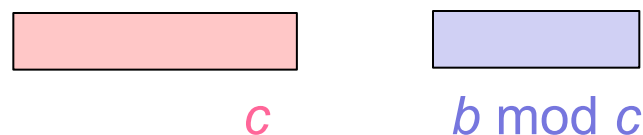
- Περίπτ. 2. αρχικά: (a, b) , $b > a/2$



- σε 1 επανάληψη: (b, c)



- σε 2 επαναλήψεις: $(c, b \bmod c)$



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 $F_k \approx \varphi^k / \sqrt{5}$,
(ισχύει $F_k = [\varphi^k - (1-\varphi)^k] / \sqrt{5}$, $\varphi = (1+\sqrt{5})/2$: χρυσή τομή)
- Άρα η πολυπλοκότητα του Ευκλείδειου είναι $\Theta(\log \max(a,b)) = \Theta(\log (a+b))$
- Bit complexity: $O(\log^3(\max(a,b)))$

Υψωση σε δύναμη

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

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

Υψωση 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: $O(2^{\log n}) = O(2^{\text{len}(n)})$*

$$\log n < \text{len}(n) \leq \log n + 1$$

... 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: $a^{13} = a^{8+4+1} = a^8 a^4 a^1 = a^{1.23+1.22+0.21+1.20}$

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

ως προς το μήκος της εισόδου: $O(\text{len}(n))$

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

```
a^n computation: a=15, n=507
```

```
-----  
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
```

```
4674626290798187255859375
```

Παράδειγμα σε Python: εκτέλεση

.

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
```

*Παρατήρηση: a^n έχει τεράστιο πλήθος ψηφίων: $O(n \log a)$.
Bit complexity αναγκαστικά εκθετική! (βλ. και παρακάτω)*

Modular exponentiation: $a^n \bmod p$

Ένας *αποδοτικός* και *πρακτικά χρήσιμος* αλγόριθμος
(έλεγχος πρώτων κατά Fermat: $a^{n-1} \bmod n =? 1$)

```
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: $O(\log n) = O(\text{len}(n))$

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

ως προς το μήκος της εισόδου: $O(\text{len}(n) \cdot \text{len}(p)^2)$

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

```
Fermat Primality Test
```

```
a^(n-1) mod n computation: a=15, n=127
```

```
126 15 1
```

```
63 98 98
```

```
31 79 122
```

```
15 18 37
```

```
7 70 50
```

```
3 74 17
```

```
1 15 1
```

```
1
```

Bit complexity για υπολογισμό a^n ; [*]

- ‘Αφελής’ αλγόριθμος, i -οστή επανάληψη: πολ/μός a με a^{i-1}
 $O((i-1) \text{len}(a)^2)$ Συνολικά: $O(n^2 \text{len}(a)^2) = O(4^{\text{len}(n)} \text{len}(a)^2)$
μη αποδοτικός!
- Αλγόριθμος τετραγωνισμού: $O(n^2 \text{len}(a)^2)$ επίσης! (γιατί;)
- Τετραγωνισμός με πολ/σμό Gauss-Karatsuba:
 $O(n^{\log 3} \log a^{\log 3}) = O(3^{\text{len}(n)} \text{len}(a)^{1.59})$ [προσεχώς!]
- Περαιτέρω βελτιώσεις [Harvey, van der Hoeven]:
 $O(n \log a (\log n + \log \log a)) = O(2^{\text{len}(n)} \text{len}(a) (\text{len}(n) + \log(\text{len}(a)))) \dots$
καλύτερος, αλλά επίσης *μη αποδοτικός*
- *Εγγενώς δύσκολο* πρόβλημα: a^n έχει *εκθετικό* πλήθος bits

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 ουσιαστικά όπως η $F(n)$ (συν μια σταθερά), οπότε αποδεικνύεται επαγωγικά ότι:

$$T(n) \geq c F(n) = \Omega(1.618^n) = \Omega(1.618^{2\text{len}(n)})$$

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!** : $O(2^{\text{len}(n)})$

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 $O(\log n) = O(\text{len}(n))$

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.
- τι ισχύει αν θέλουμε το αποτέλεσμα *mod t*, για κάποιο δοσμένο ακέραιο t; Ποιος αλγόριθμος υπερτερεί;

Σημαντική αλγοριθμική τεχνική

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

Divide-and-Conquer !

(Διαίρει-και-Κυρίευσε)

Divide-and-Conquer

- Χρονική πολυπλοκότητα αλγόριθμων «διαίρει-και-κυρίευε» με διατύπωση και λύση αναδρομικής εξίσωσης χρόνου εκτέλεσης.
- **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

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

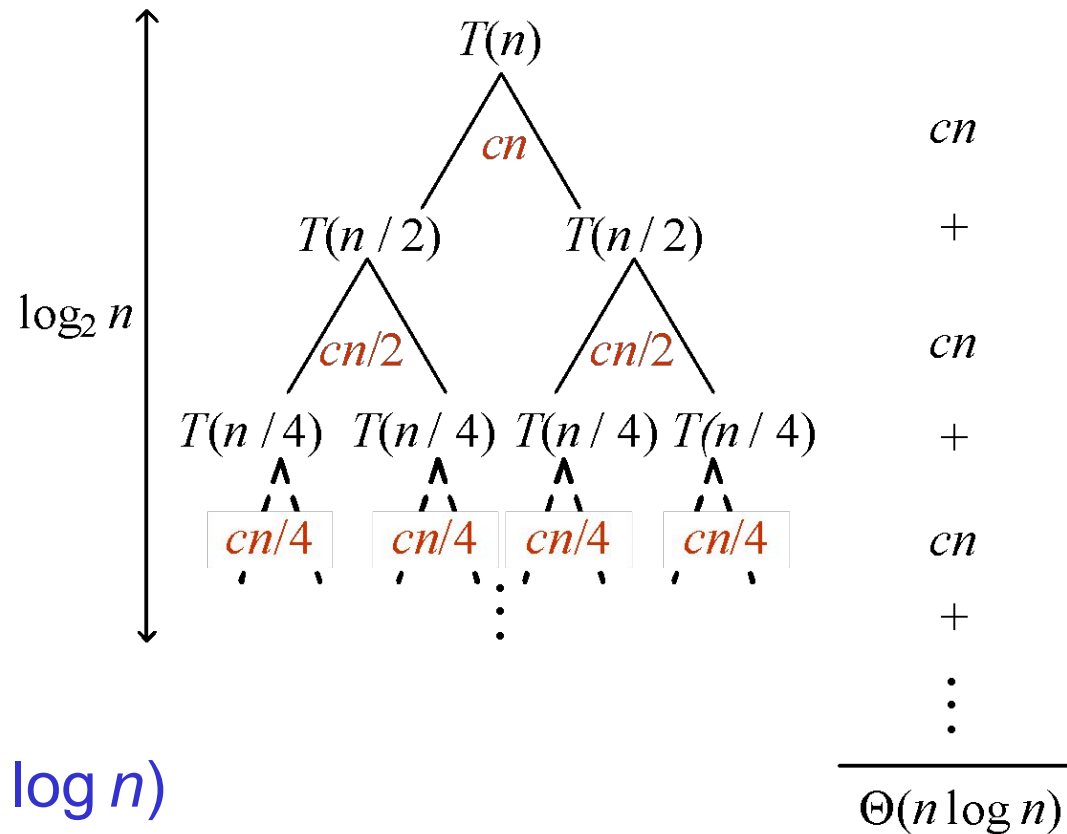
Δέντρο αναδρομής :

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

$< (4/2)^k cn$

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

$= O(n^2)^{cn}$

Συνολικά: $O(n^2)$

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

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:

$T(n)$

$T(n/2)$

$T(n/2)$

$T(n/2)$

~~$T(n/2)$~~

$T(n/4)$

$T(n/4)$

.....

.....

.....

.....

.....

$T(2)$

.....

.....

.....

.....

.....

$T(1)$

$T(1)$

.....

.....

.....

.....

.....

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

$+ cn$

$+ 3 c n/2$

$+ 9 c n/4$

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

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:

$T(n)$

$T(n/b)$

$T(n/b)$

$T(n/b)$

$T(n/b^2)$

$T(n/b^2)$

$T(1)$

$T(1)$

$+ cn$

$+ a c n/b$

$+ a^2 c n/b^2$

$+ a^{(k-1)} c n/b^{(k-1)}$

Χρον.
πολ/τα

$$\leq cn \sum_0^{k-1} (a/b)^i$$

=

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

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

$O(\log^3(\max(a, b)))$ GCD(a, b) := GCD($b, a \bmod b$) εύρεση ΜΚΔ

$O(\log n \log^2 a)$ modpow(a, n, p) := modpow($a^2, n/2, p$)
modular exponentiation

$O(\log n)^*$ αλγόριθμος πίνακα υπολογισμός n -οστού
[*αριθμ. πολυπλ/τα] fast doubling αριθμού Fibonacci

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

$O(n^{\log 3})$ Gauss-Karatsuba πολλ/σμός n -bit αριθμών

$O(n^{\log 7})$ Strassen πολλ/σμός πινάκων $n \times n$