ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΑΛΓΟΡΙΘΜΟΙ

Ασυμπτωτικός Συμβολισμός (1/2)

Συμβολισμός ασυμπτωτικού χρόνου εκτέλεσης

Χρήση: σύγκριση ρυθμού αύξησης

Τρεις συμβολισμοί:

- **Big oh (O)**
- **Big omega (Ω)**
- **Big theta (Θ)**

Ασυμπτωτικός Συμβολισμός (2/2)

O(g(n)) είναι το **σύνολο όλων των συναρτήσεων** με μικρότερο ή ίσο ρυθμό αύξησης του g(n)

Παραδείγματα

- $n \in O(n^2)$
- $100n + 5 \in O(n^2)$
- $\frac{1}{2}\, n(n-1) \in O(n^2)$

O-notation

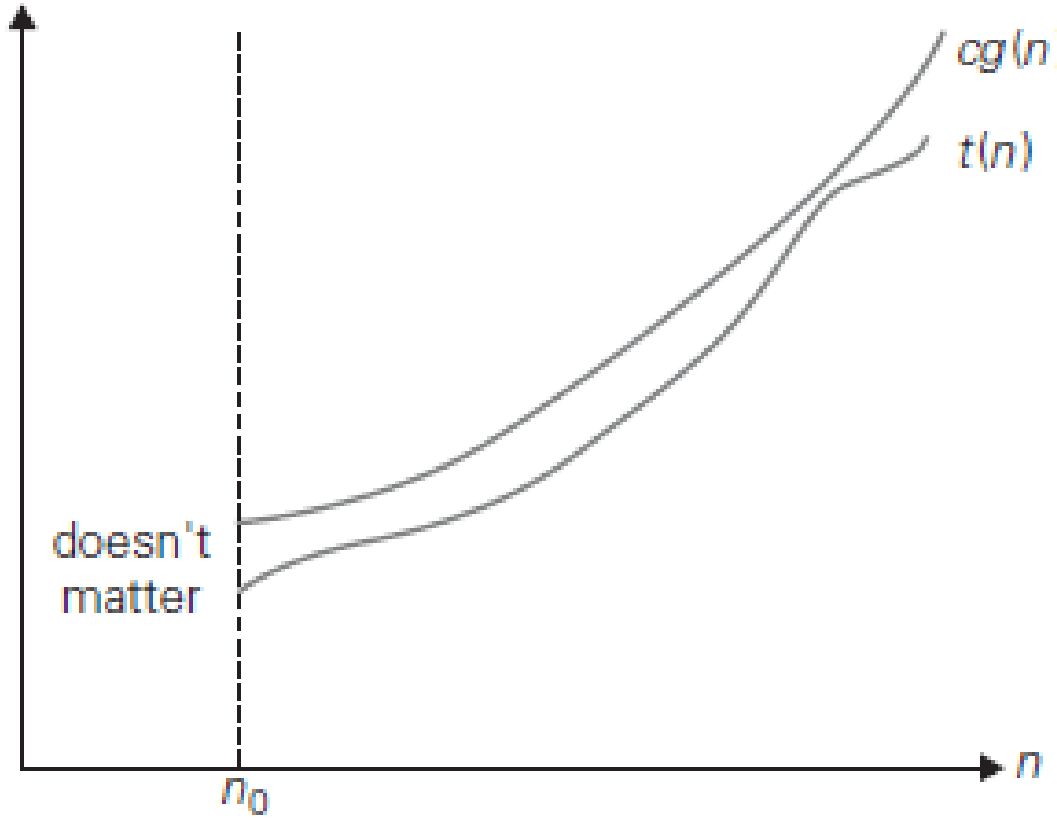
Μια συνάρτηση T(n) λέγεται ότι είναι O(g(n)) – T(n) ϵ O(g(n)) – αν η T(n) έχει **άνω φράγμα** την g(n) επί ένα σταθερό αριθμό c.

T(n) <= c g(n) για κάθε n >= n_0

Παραδείγματα:

$$100n + 5 \in O(n^2)$$

$$100n + 5 \leq 100n + n \text{ (for all } n \geq 5) = 101n \leq 101n^2$$



Ω-notation

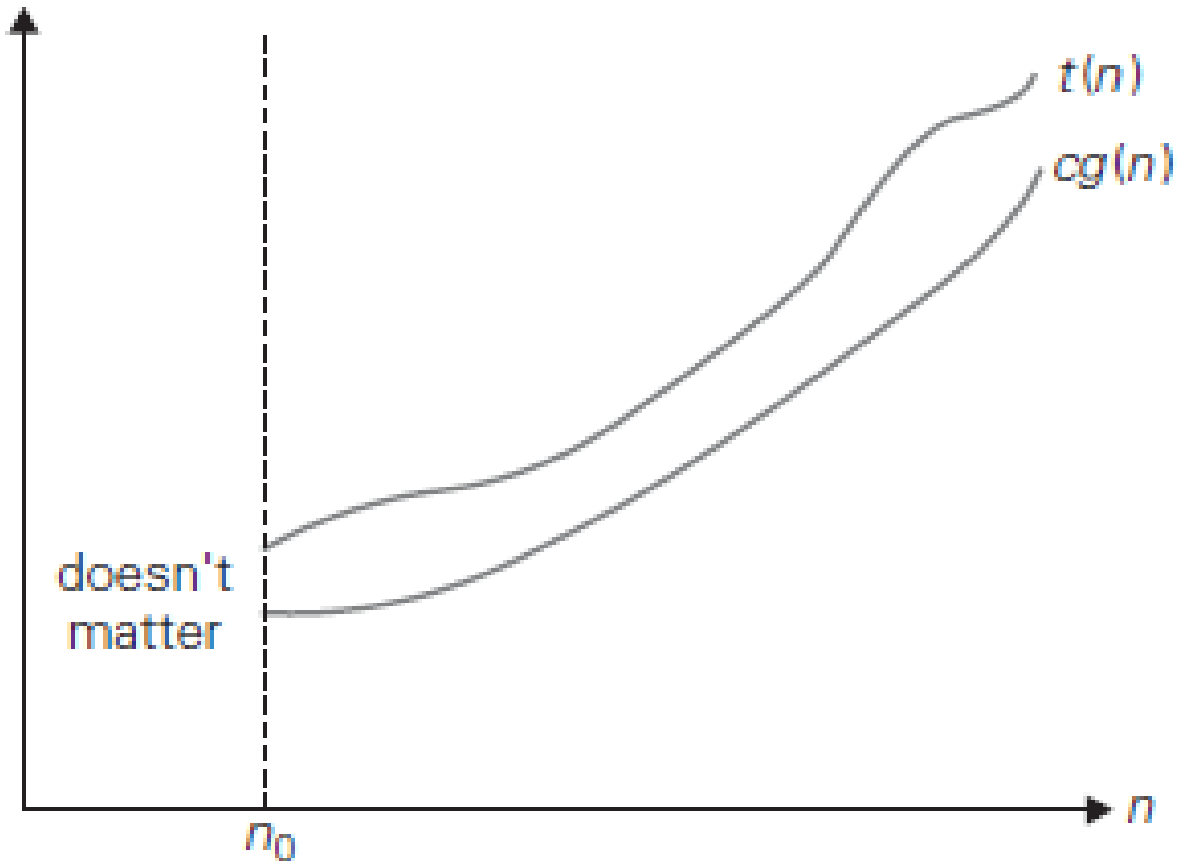
Μια συνάρτηση T(n) λέγεται ότι είναι Ω(g(n)) – T(n) ϵ Ω(g(n)) – αν η T(n) έχει **κάτω φράγμα** την g(n) επί ένα σταθερό αριθμό c.

T(n) >= c g(n) για κάθε n >= n_0

Παραδείγματα:

$$n^3 \in \Omega(n^2)$$

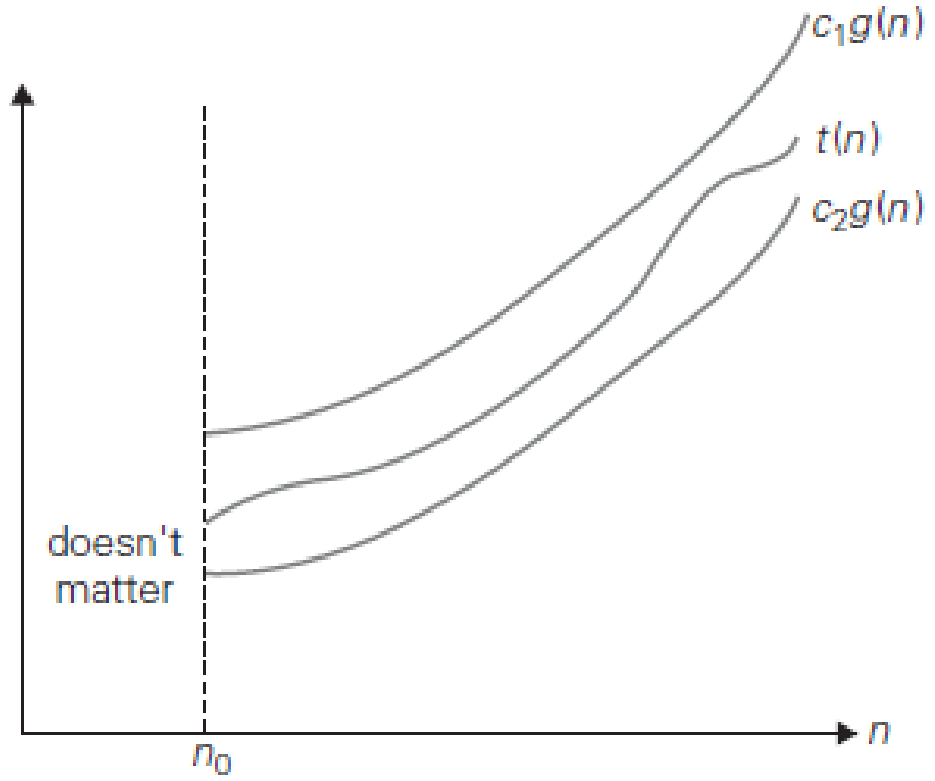
$$n^3 \geq n^2 \quad \text{for all } n \geq 0$$



Θ-notation (1/2)

Μια συνάρτηση T(n) λέγεται ότι είναι Θ(g(n)) – T(n) ϵ Θ(g(n)) – αν η T(n) έχει **κάτω και άνω φράγμα** πολλαπλάσια της g(n) για σταθερές c_1 και c_2.

c_2 g(n) <= T(n) <= c_1 g(n) για κάθε n >= n_0



Θ-notation (2/2)

Παραδείγματα:

$$\frac{1}{2}n(n-1) \in \Theta(n^2)$$

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \quad \text{for all } n \geq 0.$$

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \text{ (for all } n \geq 2) = \frac{1}{4}n^2.$$

o-notation

Μια συνάρτηση T(n) λέγεται ότι είναι o(g(n)) – T(n) ϵ o(g(n)) – αν η T(n) έχει **άνω φράγμα** την g(n) επί ένα σταθερό αριθμό c.

$T(n) < c\ g(n)$ για κάθε $n >= n_0$

Διαφορά με τον big oh notation

- Στο O(g(n)) η ανίσωση ισχύει για κάποιες σταθερές ενώ στο o(g(n)) η ανίσωση ισχύει για όλες τις σταθερές

Παράδειγμα

- $2n = o(n^2)$ αλλά $2n^2 \neq o(n^2)$

ω-notation

Μια συνάρτηση T(n) λέγεται ότι είναι o(g(n)) – T(n) ϵ o(g(n)) – αν η T(n) έχει **κάτω φράγμα** την g(n) επί ένα σταθερό αριθμό c.

$T(n) > c\ g(n)$ για κάθε $n >= n_0$

Διαφορά με τον big omega notation

- Στο Ω(g(n)) η ανίσωση ισχύει για κάποιες σταθερές ενώ στο ω(g(n)) η ανίσωση ισχύει για όλες τις σταθερές

Παράδειγμα

- $n^2/2 = \omega(n)$ αλλά $2n^2 \neq \omega(n^2)$

Θεωρήματα

Για οποιεσδήποτε δύο συναρτήσεις f(n) και g(n), έχουμε f(n) = Θ(g(n)) όταν και μόνο όταν f(n) = O(g(n)) και f(n) = Ω(g(n))

- Η απόδειξη εξάγεται εύκολα από τους ορισμούς

Χρήσιμο θεώρημα για ανάλυση αλγορίθμων με συνεχόμενα τμήματα

If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then

$$t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\}).$$

Πολυπλοκότητα κατά Μέρη - Απόδειξη

Αφού $t_1(n) \in O(g_1(n))$ και $t_2(n) \in O(g_2(n))$

τότε

$$t_1(n) \leq c_1 g_1(n) \quad \text{for all } n \geq n_1$$

$$t_2(n) \leq c_2 g_2(n) \quad \text{for all } n \geq n_2$$

Θέτω $c_3 = \max\{c_1, c_2\}$ και $n \geq \max\{n_1, n_2\}$

Προσθέτω κατά μέλη

$$\begin{aligned} t_1(n) + t_2(n) &\leq c_1 g_1(n) + c_2 g_2(n) \\ &\leq c_3 g_1(n) + c_3 g_2(n) = c_3[g_1(n) + g_2(n)] \\ &\leq c_3 2 \max\{g_1(n), g_2(n)\} \end{aligned}$$

Συνεπώς $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$

για $2c_3 = 2\max\{c_1, c_2\}$ και $\max\{n_1, n_2\}$.

Χαρακτηριστικές Πολυπλοκότητες

1	Σταθερή πολυπλοκότητα
$\log(n)$	Λογαριθμική πολυπλοκότητα
$\log^k(n)$	Πολυλογαριθμική πολυπλοκότητα όταν k μία σταθερά
n	Γραμμική πολυπλοκότητα (πχ, $f(n) = an + b$, με a, b σταθερές)
$n\log n$	
n^2	Τετραγωνική πολυπλοκότητα
n^3	Κυβική πολυπλοκότητα
n^k	Πολυωνυμικη πολυπλοκότητα, με k μία σταθερά, δηλ. $f(x) = a_k n^k + \ldots + a_0$
a^n	Εκθετική πολυπλοκότητα $1 < a < 2$
2^n	Εκθετική πολυπλοκότητα
a^n	Εκθετική πολυπλοκότητα $a > 2$
$n!$	Παραγοντική πολυπλοκότητα
n^n	Υπερεκθετική πολυπλοκότητα

Ασυμπτωτική Συμπεριφορά

Αν

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = a \neq 0$$

τότε $f(n) = \Theta(g(n))$

Αν

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$$

τότε $f(n) = O(g(n))$ και $g(n) = \Omega(f(n))$ (με $f(n) \neq \Theta(g(n))$)

Αν

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = \infty$$

τότε $g(n) = O(f(n))$ και $f(n) = \Omega(g(n))$ (με $g(n) \neq \Theta(f(n))$)

Συναρτήσεις Σύγκρισης (1/3)

Μεταβατικότητα (transitivity)

$$
\begin{aligned}
f(n) &= \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \text{ imply } f(n) = \Theta(h(n)) , \\
f(n) &= O(g(n)) \text{ and } g(n) = O(h(n)) \text{ imply } f(n) = O(h(n)) , \\
f(n) &= \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \text{ imply } f(n) = \Omega(h(n)) , \\
f(n) &= o(g(n)) \text{ and } g(n) = o(h(n)) \text{ imply } f(n) = o(h(n)) , \\
f(n) &= \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \text{ imply } f(n) = \omega(h(n)) .
\end{aligned}
$$

Συναρτήσεις Σύγκρισης (2/3)

Ανακλαστικότητα (reflexivity)

$$f(n) = \Theta(f(n)),$$
$$f(n) = O(f(n)),$$
$$f(n) = \Omega(f(n)).$$

Συναρτήσεις Σύγκρισης (3/3)

Συμμετρία (symmetry)

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) .$$

Ανάστροφη συμμετρία (transpose symmetry)

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)) ,$$
$$f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)) .$$

Πράξεις

$$cO(f(n)) = O(f(n)) \text{ με } c > 0$$

$$O(g(n)) + O(g(n)) = O(g(n))$$

$$O(g_1(n)) + O(g_2(n)) = O(\max\{g_1(n), g_2(n)\})$$

$$O(g_1(n)) * O(g_2(n)) = O(g_1(n) * g_2(n))$$

Ασκήσεις

Για τις ακόλουθες συναρτήσεις να βρείτε τον **επικρατέστερο όρο (dominant term)** καθώς και τον big oh notation συμβολισμό.

		$O(\ldots)$
$5 + 0.001n^3 + 0.025n$		
$500n + 100n^{1.5} + 50n \log_{10} n$		
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$		
$n^2 \log_2 n + n(\log_2 n)^2$		
$n \log_3 n + n \log_2 n$		
$3 \log_8 n + \log_2 \log_2 \log_2 n$		
$100n + 0.01n^2$		
$0.01n + 100n^2$		
$2n + n^{0.5} + 0.5n^{1.25}$		
$0.01n \log_2 n + n(\log_2 n)^2$		
$100n \log_3 n + n^3 + 100n$		
$0.003 \log_4 n + \log_2 \log_2 n$		

Λύσεις

		$O(\dots)$
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50n \log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5 \cdot n^{1.75}$	$2.5n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log n)$
$n \log_3 n + n \log_2 n$	$n \log_3 n,\ n \log_2 n$	$O(n \log n)$
$3 \log_8 n + \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log n)^2)$
$100n \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003 \log_4 n + \log_2 \log_2 n$	$0.003 \log_4 n$	$O(\log n)$

Υπολογισμός Πολυπλοκότητας (1/4)

Εξαρτάται από τις εντολές που υπάρχουν στον αλγόριθμο και το ‘είδος’ τους

Παραδείγματα:

A. Sequence of statements

Statement 1

Statement 2

…

Statement k

Ο συνολικός χρόνος προκύπτει ως το άθροισμα των χρόνων για κάθε εντολή

total time = time(statement 1) + time(statement 2) + ... + time(statement k)

Λύνεται με το θεώρημα που δείξαμε πιο πριν

Υπολογισμός Πολυπλοκότητας (2/4)

B. If – Then – Else

if (cond) then

block 1 (sequence of statements)

else

block 2 (sequence of statements)

end if;

Εδώ θα εκτελεστεί / εκτελείται είτε το block 1 ή το block 2

Η χειρότερη περίπτωση του αλγορίθμου εξαρτάται από τη χειρότερη περίπτωση από τα δύο blocks

Υπολογισμός Πολυπλοκότητας (3/4)

Γ. Loops

```
for I in 1 .. N
    sequence of statements
end;
```

Ελέγχουμε το πλήθος των εκτελέσεων του loop καθώς και την πολυπλοκότητα των εντολών μέσα στο loop

Παράδειγμα: αν το loop είναι O(N) και οι εντολές μέσα στο loop είναι O(1), τότε η πολυπλοκότητα είναι O(N)

Υπολογισμός Πολυπλοκότητας (4/4)

Γ. Nested Loops

```
for I in 1 .. N
  for J in 1 .. M
     sequence of statements
  end;
end;
```

Εδώ ελέγχουμε την πολυπλοκότητα του καθενός και υπολογίζουμε το γινόμενο

Προσοχή χρειάζεται όταν το πλήθος επαναλήψεων ενός loop εξαρτάται από το πλήθος ενός άλλου

Παράδειγμα: αν το εξωτερικό loop είναι O(N) και το εσωτερικό loop είναι O(M), τότε η πολυπλοκότητα είναι O(M * N)

Ασκήσεις

Υπολογίστε την πολυπλοκότητα του επόμενου αλγορίθμου

```
List<Integer> getBiggersList( int[] intArray )
{
   List<Integer> biggers = new LinkedList<Integer>();
    int sum = 0;
   for ( int item : intArray )
    {
            if ( item > sum )
               biggers.add( item );
        sum += item;
   }
   return biggers;
}
```

Λύσεις

Ο(n) διότι εξαρτάται μόνο από την επαναληπτική διαδικασία

Ασκήσεις

Υπολογίστε την πολυπλοκότητα των ακόλουθων αλγορίθμων

a.
```
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        sequence of statements of O(1)
    }
}
```

b.
```
for (i = 0; i < N; i++) {
    for (j = i+1; j < N; j++) {
        sequence of statements of O(1)
    }
}
```

Λύσεις

(a) $O(N^2)$

(b) $O(N^2)$, διότι το πλήθος των επαναλήψεων είναι $(N-1)+(N-2)+....+0$

Χρήσιμοι Μαθηματικοί Τύποι (1/3)

Λογάριθμοι

$$\log_a 1 = 0$$

$$\log_a a = 1$$

$$\log_a x^y = y \log_a x$$

$$\log_a xy = \log_a x + \log_a y$$

$$\log_a \frac{x}{y} = \log_a x - \log_a y$$

$$a^{\log_b x} = x^{\log_b a}$$

$$\log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$$

Χρήσιμοι Μαθηματικοί Τύποι (2/3)

Αθροίσματα

$$\sum_{i=l}^{u} 1 = \underbrace{1+1+\cdots+1}_{u-l+1 \text{ times}} = u-l+1 \ (l, u \text{ are integer limits}, l \le u); \quad \sum_{i=1}^{n} 1 = n$$

$$\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1}-1}{a-1} \ (a \neq 1); \quad \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$$

$$\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$$

$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma, \text{ where } \gamma \approx 0.5772\ldots \text{ (Euler's constant)}$$

$$\sum_{i=1}^{n} i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1}n^{k+1}$$

$$\sum_{i=1}^{n} \lg i \approx n \lg n$$

Χρήσιμοι Μαθηματικοί Τύποι (3/3)

Ιδιότητες

$$\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$$

$$\sum_{i=l}^{u}(a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

$$\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i, \text{ where } l \le m < u$$

$$\sum_{i=l}^{u}(a_i - a_{i-1}) = a_u - a_{l-1}$$

$$\int_{l-1}^{u} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l}^{u+1} f(x)dx \quad \text{for a nondecreasing } f(x)$$

$$\int_{l}^{u+1} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l-1}^{u} f(x)dx \quad \text{for a nonincreasing } f(x)$$

$$x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x + 1$$

$$\lfloor x + n \rfloor = \lfloor x \rfloor + n \text{ and } \lceil x + n \rceil = \lceil x \rceil + n \quad \text{for real } x \text{ and integer } n$$

$$\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$$

$$\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$$

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \quad \text{as } n \to \infty \text{ (Stirling's formula)}$$

Ασκήσεις (1/7)

Να αποδειχθεί ο κανόνας της συμμετρίας: f(n) = Θ(g(n)) είναι ισοδύναμο με g(n) = Θ(f(n))

Απόδειξη:

Από τον ορισμό έχουμε $\exists c_1, c_2, n_0 > 0$ ώστε $0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)$

Αφού c1, c2 > 0, διαιρούμε κατά μέλη. Συνεπώς:

$$0 \leq \frac{1}{c_2} f(n) \leq g(n) \text{ and } g(n) \leq \frac{1}{c_1} f(n)$$

και

$$0 \leq \frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n)$$

με

$$k_1 = \frac{1}{c_2} \quad k_2 = \frac{1}{c_1}$$

Συνεπώς $0 \leq k_2.f(n) \leq g(n) \leq k_1.f(n)$ for all $n \geq n_0$

Ασκήσεις (2/7)

Έστω $f(n) = ½\ n^2 – 3n$. Να αποδειχθεί ότι $f(n) = \Theta(n^2)$

Απόδειξη:

Αρκεί να αποδείξουμε ότι υπάρχουν $c_1, c_2 > 0$ τέτοια ώστε

$$0 \leq c_1.n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2.n^2$$

Διαιρούμε με n^2 και έχουμε $$0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

Παίρνουμε τρεις ανισώσεις

$$0 \leq c_1$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n}$$

$$\frac{1}{2} - \frac{3}{n} \leq c_2$$

Ασκήσεις (3/7)

Η 1η ανίσωση είναι προφανής για οποιοδήποτε c_1

Η δεύτερη ικανοποιείται για το μικρότερο n που την κάνει θετική. Αυτό είναι το 7. Το δεξιό μέρος γίνεται 1/14 για $c_1 = 1/14$ και $n_0' = 7$

Η τρίτη ικανοποιείται για $n_0'' = 1$ και $c_2 = 1$

Συνεπώς, $c_1 = 1/14$, $c_2 = 1$ και $n_0 = \max\{n_0', n_0''\} = 7$

$$0 \leq c_1$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n}$$

$$\frac{1}{2} - \frac{3}{n} \leq c_2$$

Ασκήσεις (4/7)

Να διαπιστώσετε αν $\frac{1}{1000}n^3 = O(1000n^2)$

Απόδειξη

Έστω ότι ισχύει. Σε αυτή την περίπτωση έχουμε:

$$\frac{1}{1000}n^3 \leq c.1000n^2$$

Οπότε $\forall n \geq n_0$:

είναι $$\frac{1}{1000}n \leq 1000.c \Leftrightarrow n \leq 1000000.c$$

Το οποίο είναι ψευδές

Ασκήσεις (5/7)

Έστω πολυώνυμο p(n) βαθμού k τέτοιο ώστε ο συντελεστής του μεγιστοβάθμιου όρου να είναι θετικός. Να δείξετε ότι $p(n) = \Theta(n^k)$

Απόδειξη

Αρκεί να αποδείξουμε ότι για κάθε $n \geq n_0,\ 0 \leq c_1 n^k \leq p(n) \leq c_2 n^k$

Δηλαδή αρκεί να αποδείξουμε ότι

$$c_1 n^k \leq a_k n^k + a_{k-1} n^{k-1} + a_{k-2} n^{k-2} \ldots + a_1 n + a_0 \leq c_2 n^k$$

Διαιρώντας με n^k παίρνουμε

$$c_1 \leq a_k + \underbrace{\frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \ldots + \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k}}_{T} \leq c_2$$ με $\lim_{n\to\infty} T = 0$. Άρα $0 < c_1 < a_k \qquad c_2 > a_k$

Ασκήσεις (6/7)

Αν $a \in \mathbb{R}$ και $b \in \mathbb{R}^+$. Να αποδείξετε ότι $(n+a)^b \approx n^b$

Παρατηρούμε ότι

$$n + a \leq n + |a| \leq 2n, \quad \text{με } n \geq |a|$$

$$n + a \geq n - |a| \geq \frac{n}{2} \quad \text{με } \frac{n}{2} \geq |a| \quad n \geq 2|a|$$

Έχουμε

$$\frac{1}{2}n \leq n + a \leq 2n, \text{ if } n \geq 2|a|$$

Υψώνοντας στη b^{th} παίρνουμε

$$\left(\frac{1}{2}\right)^b n^b \leq (n+a)^b \leq 2^b n^b$$

Έτσι

$$c_1 = \left(\tfrac{1}{2}\right)^b,\ c_2 = 2^b,\ n_0 = \lceil 2|a| \rceil$$

Ασκήσεις (7/7)

Να αποδειχθεί ότι $\sqrt[n]{n} = \Theta(1)$

Απόδειξη

Ισχύει

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

και $\sqrt[n]{n} = e^{\ln \sqrt[n]{n}} = e^{\left(\frac{\ln n}{n}\right)}$

Οπότε $e^{\left(\frac{\ln n}{n}\right)} = 1 + \underbrace{\frac{\ln n}{n} + \frac{\left(\frac{\ln n}{n}\right)^2}{2!} + \frac{\left(\frac{\ln n}{n}\right)^3}{3!} + \dots}_{T(n)}$

Όμως $S(x) = \frac{\ln x}{x} + \frac{\ln^2 x}{x^2} + \frac{\ln^3 x}{x^3} + \dots, \qquad \lim_{x\to\infty} S(x) = 0$

και συνεπώς $\lim_{n\to\infty} T(n) = 0$

Άρα $\lim_{n\to\infty} \sqrt[n]{n} = 1$

Μη Αναδρομικά Προβλήματα

Εύρεση Μέγιστης Τιμής (1/2)

Προσπαθούμε να βρούμε το μέγιστο ανάμεσα σε n αριθμούς

Η προφανής μετρική είναι το μέγεθος / πλήθος των αριθμών n

Δύο λειτουργίες είναι αυτές που θα εκτελεστούν μέσα στο for:

- Η σύγκριση
- Η ανάθεση τιμής στο μέγιστο

Η βασική λειτουργία είναι η **σύγκριση**

Αφού η σύγκριση θα εκτελεστεί n-1 φορές δεν υπάρχει ανάγκη για διάκριση καλύτερης, μέσης, χειρότερης περίπτωσης

ALGORITHM *MaxElement*$(A[0..n-1])$
//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$

Εύρεση Μέγιστης Τιμής (2/2)

Έστω C(n) το πλήθος των συγκρίσεων ως συνάρτηση του n

Έχουμε:

$$C(n) = \sum_{i=1}^{n-1} 1$$

Πολύ εύκολη παράσταση με την οποία καταλήγουμε ότι:

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n)$$

Μεθοδολογία

Η γενική μεθοδολογία για την ανάλυση μη αναδρομικών αλγορίθμων έχει ως εξής:

- **Αποφασίζουμε** την παράμετρο του μεγέθους της εισόδου
- **Αναγνωρίζουμε** τη βασική λειτουργία του αλγορίθμου (π.χ. σύγκριση, ανάθεση τιμής)
- **Ελέγχουμε** αν το πλήθος εκτέλεσης της βασικής λειτουργίας εξαρτάται μόνο από το μέγεθος εισόδου. Αν εξαρτάται από κάποια επιπλέον παράμετρο τότε μελετούμε ξεχωριστά την καλύτερη, τη χειρότερη και τη μέση περίπτωση.
- **Δημιουργούμε** ένα άθροισμα που απεικονίζει το πλήθος των εκτελέσεων των βασικών λειτουργιών.
- **Χρησιμοποιούμε** μαθηματικές μεθόδους και τύπους / θεωρήματα ώστε να καταλήξουμε στον τελικό ρυθμό αύξησης του αλγορίθμου.

Εύρεση Μοναδικότητας Στοιχείων (1/2)

Μέγεθος εισόδου: n (πλήθος στοιχείων του πίνακα)

Το loop περιέχει μόνο μια σύγκριση στοιχείων που είναι και η βασική λειτουργία

Το πλήθος συγκρίσεων εξαρτάται επίσης από το αν υπάρχουν ίσα στοιχεία

Μελετάμε τη χειρότερη περίπτωση

- **Πίνακες με όλα τα στοιχεία διαφορετικά**
- **Πίνακες με τα δύο τελευταία στοιχεία ίσα**

$C_{worst} = n$

ALGORITHM *UniqueElements*$(A[0..n-1])$

```
//Determines whether all the elements in a given array are distinct
//Input: An array A[0..n − 1]
//Output: Returns "true" if all the elements in A are distinct
//          and "false" otherwise
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] = A[j] return false
return true
```

Έχουμε:

$$\sum_{i=0}^{n-2}(n-1-i) = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2} \in \Theta(n^2)$$