ΦΡΟΝΤΙΣΤΗΡΙΟ 1

Πολυπλοκότητες – Επαναληπτικές Διαδικασίες

Ακολουθούν ασκήσεις σε πολυπλοκότητες αλγορίθμων.

Άσκηση 1

Να αποδείξετε ότι: $f(n) = \Theta(g(n)) \Leftrightarrow (f(n))^k = \Theta((g(n))^k), k \in \mathbb{R}^+$

Άσκηση 1

Να αποδείξετε ότι: $f(n) = \Theta\big(g(n)\big) \Leftrightarrow \big(f(n)\big)^k = \Theta((g(n))^k), k \in \mathbb{R}^+$

Λύση 1

Έχουμε: $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ για κάποια c_1, $c_2 > 0$ και $\forall n \geq n_0$.

Υψώνουμε εις την k και παίρνουμε: $0 \leq c_1^k \quad (g(n))^k \leq (f(n))^k \leq c_2^k \quad (g(n))^k$

Αφού $c_1^k \quad , c_2^k \quad > 0$, η αρχική υπόθεση ισχύει.

Άσκηση 2

Για δύο ασυμπτωτικά θετικές συναρτήσεις f(n), g(n), να αποδείξετε ότι ισχύει η ακόλουθη σχέση:

$$\max(f(n),g(n))=\Theta(f(n)+g(n))$$

Άσκηση 2

Για δύο ασυμπτωτικά θετικές συναρτήσεις f(n), g(n), να αποδείξετε ότι ισχύει η ακόλουθη σχέση:

$$\max(f(n),g(n))=\Theta(f(n)+g(n))$$

Λύση 2

Αρκεί να αποδείξουμε ότι: : $0 \leq c_1(f(n)+g(n)) \leq \max\left(f(n), g(n) \leq c_2(f(n)+g(n))\right.$

Αφού οι δύο συναρτήσεις είναι ασυμπτωτικά θετικές σημαίνει πως

$$\exists n_0', \forall n \geq n_0' \qquad f(n) > 0$$
$$\exists n_0'', \forall n \geq n_0'' \qquad g(n) > 0$$

Παίρνουμε, $n_0''' = \max\{n_0', n_0''\}$ ώστε $\forall n \geq n_0''' \qquad f(n), g(n) > 0$ και συνεπώς:

$0 \leq c_1(f(n)+g(n))$ με $c_1>0$.

Επίσης είναι προφανές ότι:

$$f(n)+g(n) \leq 2\max\{f(n), g(n)\} \implies \frac{1}{2}f(n)+\frac{1}{2}g(n) \leq \max\{f(n), g(n)\}$$

και

$$f(n)+g(n) \geq \max\{f(n), g(n)\}$$

Άρα, μπορούμε να πάρουμε $c_1=1/2$ & $c_2=1$ ώστε να ισχύει η αρχική υπόθεση.

Άσκηση 3

Δείξτε ποιες από τις παρακάτω σχέσεις ισχύουν:

A. $2^{n+1} = \Theta(2^n)$

B. $2^{2n} = \Theta(2^n)$

Γ. $n^2 = \Theta(4^{\log n})$

Άσκηση 3

Δείξτε ποιες από τις παρακάτω σχέσεις ισχύουν:

A. $2^{n+1} = \Theta(2^n)$

B. $2^{2n} = \Theta(2^n)$

Γ. $n^2 = \Theta(4^{logn})$

Λύση 3

A. Έχουμε: $2^{n+1} = 2\ 2^n = c\ 2^n$. Άρα ισχύει

B. Από την αρχική υπόθεση παίρνουμε:

$c_1 2^n \leq 2^{2n} \leq c_2 2^n \implies c_1 \leq 2^n \leq c_2$

Άρα, δεν ισχύει

Γ. Ισχύει διότι $4^{logn} = 2^{2logn} = 2^{logn^2} = n^2$

Άσκηση 4

Δείξτε αν η σχέση $f(n) = O(g(n))$ οδηγεί και στη $g(n) = O(f(n))$

Άσκηση 4
Δείξτε αν η σχέση $f(n) = O(g(n))$ οδηγεί και στη $g(n) = O(f(n))$

Λύση 4

Δεν ισχύει. Παράδειγμα: $n = O(n^2)$ αλλά n^2 <> $O(n)$

Άσκηση 5

Έστω k μια σταθερά με $0<k<1$. Να αποδείξετε ότι: $1+k+k^2+k^3+\ldots+k^n = \Theta(1)$

Άσκηση 5
Έστω k μια σταθερά με 0<k<1. Να αποδείξετε ότι: $1+k+k^2+k^3+\ldots+k^n = \Theta(1)$

Λύση 5
Ισχύει ότι $1+k+k^2+k^3+\ldots+k^n \leq \sum_{i=0}^{\infty} k^i$
Όμως από τη γεωμετρική πρόοδο έχουμε ότι (Το άθροισμα των άπειρων όρων μιας γεωμετρικής προόδου που έχει πρώτο όρο $\alpha_1=1$ και λόγο k, με $|k| < 1$, είναι):

$$\sum_{i=0}^{\infty} k^i = \frac{1}{1-k}$$

το οποίο είναι ένας σταθερός αριθμός.
Άρα η αρχική υπόθεση ισχύει.

Άσκηση 6

Έστω ότι ένας αλγόριθμος πολυπλοκότητας O(nlogn) χρειάζεται ακριβώς 1ms για να επεξεργαστεί 1,000 αντικείμενα. Υποθέτοντας ότι ο χρόνος T(n) για την επεξεργασία n αντικειμένων είναι ανάλογος της πολυπλοκότητας, δηλαδή T(n) = c nlogn, να εξάγετε μια παράσταση για το T(n) δεδομένου του χρόνου T(N) για την επεξεργασία των N αντικειμένων. Πόσος χρόνος θα χρειαστεί για την επεξεργασία 1,000,000 αντικειμένων;

Άσκηση 6

Έστω ότι ένας αλγόριθμος πολυπλοκότητας O(nlogn) χρειάζεται ακριβώς 1ms για να επεξεργαστεί 1,000 αντικείμενα. Υποθέτοντας ότι ο χρόνος T(n) για την επεξεργασία n αντικειμένων είναι ανάλογος της πολυπλοκότητας, δηλαδή T(n) = c nlogn, να εξάγετε μια παράσταση για το T(n) δεδομένου του χρόνου T(N) για την επεξεργασία των N αντικειμένων. Πόσος χρόνος θα χρειαστεί για την επεξεργασία 1,000,000 αντικειμένων;

Λύση 6

Αφού έχουμε δεδομένη την πολυπλοκότητα, παίρνουμε ότι: $T(n) = c\ nlogn$

Με βάση την εξίσωση μπορούμε εύκολα να δούμε την τιμή του c που είναι:

$c = \frac{T(N)}{NlogN}$ και συνεπώς η αρχική εξίσωση γίνεται: $T(n) = \frac{T(N)}{NlogN}\ nlogn$

Επομένως, για 1,000,000 αντικείμενα θα έχουμε:

$$T(1{,}000{,}000) = \frac{T(1{,}000)}{1{,}000\ log1{,}000}\ 1{,}000{,}000\ log1{,}000{,}000 = 1 \cdot \frac{1{,}000{,}000\ \cdot 6}{1{,}000\ \cdot 3} = 2{,}000\ ms$$

Άσκηση 7

Ένας αλγόριθμος έχει χρόνο επεξεργασίας $T(n)=cn^2$ και ξοδεύει χρόνο T(N) δευτερόλεπτα για την επεξεργασία N αντικειμένων. Πόσο χρόνο θα ξοδέψει ο αλγόριθμος για την επεξεργασία 5,000 αντικειμένων υποθέτοντας ότι N = 100 και T(N) = 1ms.

Άσκηση 7

Ένας αλγόριθμος έχει χρόνο επεξεργασίας T(n)=cn^2 και ξοδεύει χρόνο T(N) δευτερόλεπτα για την επεξεργασία N αντικειμένων. Πόσο χρόνο θα ξοδέψει ο αλγόριθμος για την επεξεργασία 5,000 αντικειμένων υποθέτοντας ότι N = 100 και T(N) = 1ms.

Λύση 7

Από την αρχική εξίσωση παίρνουμε ότι: $T(n) = c\, n^2$

Με βάση την εξίσωση μπορούμε εύκολα να δούμε την τιμή του c που είναι:

$c = \frac{T(N)}{N^2}$ και συνεπώς η αρχική εξίσωση γίνεται: $T(n) = \frac{T(N)}{N^2}\, n^2 = \frac{1}{10{,}000}\, n^2$

Συνεπώς:

$$T(5{,}000) = \frac{1}{10{,}000}\, 5{,}000^2 = 2{,}500\, ms$$

Άσκηση 8

Έστω αλγόριθμος με πολυπλοκότητα O(f(n)) και χρόνο επεξεργασίας T(n)=cf(n). Ο αλγόριθμος χρειάζεται 10 ms για να επεξεργαστεί 1,000 αντικείμενα. Πόσο χρόνο θα χρειαστεί για την επεξεργασία 100,000 αντικειμένων όταν f(n)=n & $f(n)=n^3$;

Άσκηση 8

Έστω αλγόριθμος με πολυπλοκότητα O(f(n)) και χρόνο επεξεργασίας T(n)=cf(n). Ο αλγόριθμος χρειάζεται 10 ms για να επεξεργαστεί 1,000 αντικείμενα. Πόσο χρόνο θα χρειαστεί για την επεξεργασία 100,000 αντικειμένων όταν f(n)=n & f(n)=n^3 ;

Λύση 8

Χρησιμοποιώντας το προηγούμενο σκεπτικό θα έχουμε πως $c = \frac{10}{f(1{,}000)}$

Άρα για κάθε συνάρτηση έχουμε:

$$T(n) = \frac{10}{1{,}000}\ 100{,}000 \quad = 1{,}000\ ms$$

&

$$T(n) = \frac{10}{1{,}000^3}\ 100{,}000^3 = 10^7\ ms$$

Άσκηση 9

Έστω αλγόριθμοι A και B που έχουν πολυπλοκότητες $T_A(n)=0.1n^2\log_{10}n$ & $T_B(n)=2.5n^2$. Ποιος από τους δύο είναι καλύτερος και για ποιο n_0 ο αλγόριθμος που επιλέξατε υπερτερεί του άλλου; Αν το πρόβλημά μας έχει μέγεθος $n<= 10^9$ ποιος αλγόριθμος θα πρέπει να επιλεγεί;

Άσκηση 9

Έστω αλγόριθμοι A και B που έχουν πολυπλοκότητες $T_A(n)=0.1n^2\log_{10}n$ & $T_B(n)=2.5n^2$. Ποιος από τους δύο είναι καλύτερος και για ποιο n_0 ο αλγόριθμος που επιλέξατε υπερτερεί του άλλου; Αν το πρόβλημά μας έχει μέγεθος $n<= 10^9$ ποιος αλγόριθμος θα πρέπει να επιλεγεί;

Λύση 9

Από πλευράς πολυπλοκότητας, ο B είναι καλύτερος. Υπερτερεί εφόσον: $T_B(n) <= T_A(n)$

Αυτό σημαίνει ότι: $2.5n^2 <= 0.1n^2\log_{10}n \rightarrow 2.5 <= 0.1\log_{10}n \rightarrow 25 <= \log_{10}n$

Άρα, θα πρέπει $n>n_0=10^{25}$.

Όταν $n<= 10^9$, τότε μας συμφέρει ο πρώτος αλγόριθμος.

Άσκηση 10

Έστω αλγόριθμοι A και B που έχουν πολυπλοκότητες $T_A(n)=5n\log_{10}n$ & $T_B(n)=25n$. Ποιος από τους δύο είναι καλύτερος και για ποιο n_0 ο αλγόριθμος που επιλέξατε υπερτερεί του άλλου;

Άσκηση 10

Έστω αλγόριθμοι A και B που έχουν πολυπλοκότητες $T_A(n)=5n\log_{10}n$ & $T_B(n)=25n$. Ποιος από τους δύο είναι καλύτερος και για ποιο n_0 ο αλγόριθμος που επιλέξατε υπερτερεί του άλλου;

Λύση 10

O αλγόριθμος B είναι καλύτερος. O B υπερτερεί όταν:

$25n <= 5n\log_{10}n \rightarrow 5 <= \log_{10}n \rightarrow 100{,}000 <= n$

Άσκηση 11

Έστω αλγόριθμοι A και B που χρειάζονται $T_A(n)=c_A n\log_2 n$ & $T_B(n)=c_B n^2$ ms για την επεξεργασία n στοιχείων. Ποιος από τους δύο είναι καλύτερος για την επεξεργασία $n= 2^{20}$ στοιχείων αν ο A ξοδεύει 10 ms για 1024 στοιχεία και ο B 1 ms για τον ίδιο αριθμό στοιχείων;

Άσκηση 11

Έστω αλγόριθμοι A και B που χρειάζονται $T_A(n)=c_A n\log_2 n$ & $T_B(n)=c_B n^2$ ms για την επεξεργασία n στοιχείων. Ποιος από τους δύο είναι καλύτερος για την επεξεργασία $n= 2^{20}$ στοιχείων αν ο A ξοδεύει 10 ms για 1024 στοιχεία και ο B 1 ms για τον ίδιο αριθμό στοιχείων;

Λύση 11

Αρχικά υπολογίζουμε τους σταθερούς συντελεστές για κάθε αλγόριθμο. Έτσι έχουμε:

$$c_A = \frac{10}{1024\,log1024} = \frac{1}{1024}$$

$$c_B = \frac{1}{1024^{\,2}}$$

Επομένως για $2^{20} = 1024^2$ στοιχεία θα χρειαστούν:

$$T_A(1024^2) = \frac{1}{1024} 2^{20} log 2^{20} = 20 \cdot 1024 = 20480\, ms$$

$$T_B(1024^2) = \frac{1}{1024^2} 2^{40} = 2^{20}\, ms$$

Άσκηση 12

Δύο αλγόριθμοι A και B έχουν χρόνο επεξεργασίας $3n^{1.5}$ και $0.03n^{1.75}$, αντίστοιχα. Αν ενδιαφερόμαστε για τη γρήγορη επεξεργασία δεδομένων μεγέθους $n = 10^6$ στοιχείων, ποιον αλγόριθμο θα πρέπει να επιλέξουμε;

Άσκηση 12

Δύο αλγόριθμοι Α και Β έχουν χρόνο επεξεργασίας $3n^{1.5}$ και $0.03n^{1.75}$, αντίστοιχα. Αν ενδιαφερόμαστε για τη γρήγορη επεξεργασία δεδομένων μεγέθους $n = 10^6$ στοιχείων, ποιον αλγόριθμο θα πρέπει να επιλέξουμε;

Λύση 12

Βλέποντας την πολυπλοκότητά τους, παρατηρούμε ότι ο Α είναι καλύτερος. Θα διαπιστώσουμε για ποιο μέγεθος εισόδου ο Α είναι καλύτερος. Πρέπει: $3n^{1.5} <= 0.03n^{1.75}$. Συνεχίζοντας τους υπολογισμούς παίρνουμε: $n^{0.25} >= 3/0.03 = 100$. Άρα, πρέπει $n >= 10^8$. Συνεπώς, για 10^6 στοιχεία, θα πρέπει να προτιμήσουμε τον αλγόριθμο Β.

Άσκηση 13

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        for( j = 0; j < 2n; j++)
                sum++;
```

Άσκηση 13

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        for( j = 0; j < 2n; j++)
                sum++;
```

Λύση 13

Εύκολα μπορούμε να εξάγουμε πως η πολυπλοκότητα είναι $O(n^2)$

Άσκηση 14

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        for( j = 0; j < n * n; j++)
                sum++;
```

Άσκηση 14

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
	for( j = 0; j < n * n; j++)
		sum++;
```

Λύση 14

Το πρώτο for έχει πολυπλοκότητα O(n). Το δεύτερο έχει πολυπλοκότητα $O(n^2)$. Άρα, η τελική πολυπλοκότητα θα είναι $O(n^3)$.

Άσκηση 15

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        for( j = 0; j < i; j++)
                sum++;
```

Άσκηση 15

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
	for( j = 0; j < i; j++)
sum++;
```

Λύση 15

Το πρώτο for έχει πολυπλοκότητα O(n). Το δεύτερο έχει πολυπλοκότητα O(n). Άρα, η τελική πολυπλοκότητα θα είναι $O(n^2)$.

Άσκηση 16

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        sum++;

val = 1;
for( j = 0; j < n*n; j++)
        val = val * j;
```

Άσκηση 16

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        sum++;

val = 1;
for( j = 0; j < n*n; j++)
        val = val * j;
```

Λύση 16

Το πρώτο for έχει πολυπλοκότητα $O(n)$. Το δεύτερο έχει πολυπλοκότητα $O(n^2)$. Άρα, η τελική πολυπλοκότητα θα είναι $O(n^2)$.

Άσκηση 17

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        sum++;
for( j = 0; j < n*n; j++)
                compute_val(sum,j);
```

Δοσμένης της πολυπλοκότητας της *compute_val(x,y)* που είναι O(nlog*n*)

Άσκηση 17

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
sum = 0;
for( i = 0; i < n; i++)
        sum++;
for( j = 0; j < n*n; j++)
                compute_val(sum,j);
```

Δοσμένης της πολυπλοκότητας της *compute_val(x,y)* που είναι O(nlog*n*)

Λύση 17

Το πρώτο for έχει πολυπλοκότητα O(n). Το δεύτερο έχει πολυπλοκότητα $O(n^3 \log n)$. Άρα, η τελική πολυπλοκότητα θα είναι $O(n^3 \log n)$.

Άσκηση 18

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
s=0
for i=1; i*i <= n; i++
        for j=1; j<=i; j++
                s+=n+i-j
```

Άσκηση 18

Να βρείτε την πολυπλοκότητα του επόμενου τμήματος αλγορίθμου:

```
s=0
for i=1; i*i <= n; i++
        for j=1; j<=i; j++
                s+=n+i-j
```

Λύση 18

Ο αλγόριθμος μπορεί να γραφεί ως εξής:

s=0

for i=1; i <= $\lfloor\sqrt{n}\rfloor$; i++

        for j=1; j<=i; j++

                s+=n+i-j

Άρα:

$$\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}\sum_{j=1}^{i}1=\sum_{i=1}^{\lfloor\sqrt{n}\rfloor}i=\frac{\lfloor\sqrt{n}\rfloor(\lfloor\sqrt{n}\rfloor+1)}{2}=\Theta\left(\lfloor\sqrt{n}\rfloor^{2}\right)=\Theta(n)$$

Άσκηση 19

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
for (int i=1; i<=n; i++)
{
   for (int j=1; j<=n; j++)
   {
      printf("*");
      break;
   }
}
```

Άσκηση 19

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
for (int i=1; i<=n; i++)
{
   for (int j=1; j<=n; j++)
   {
      printf("*");
      break;
   }
}
```

Λύση 19

Το πρώτο for έχει πολυπλοκότητα O(n). Το δεύτερο έχει πολυπλοκότητα O(1) λόγω του break (θα εκτελεστεί μόνο μια φορά). Άρα, η τελική πολυπλοκότητα θα είναι O(n).

Άσκηση 20

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
for (int i=n/2; i<=n; i++)
    for (int j=1; j<=n; j = 2 * j)
        for (int k=1; k<=n; k = k * 2)
            count++;
```

Άσκηση 20

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
for (int i=n/2; i<=n; i++)
     for (int j=1; j<=n; j = 2 * j)
        for (int k=1; k<=n; k = k * 2)
            count++;
```

Λύση 20

Το πρώτο for έχει πολυπλοκότητα O(n). Το δεύτερο έχει πολυπλοκότητα O(logn) όπως επίσης και το 3ο for. Άρα, η τελική πολυπλοκότητα θα είναι $O(n\log^2 n)$.

Άσκηση 21

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
for (int i=n/2; i<=n; i++)
    for (int j=1; j+n/2<=n; j = j++)
        for (int k=1; k<=n; k = k * 2)
            count++;
```

Άσκηση 21

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
for (int i=n/2; i<=n; i++)
    for (int j=1; j+n/2<=n; j = j++)
        for (int k=1; k<=n; k = k * 2)
            count++;
```

Λύση 21

Το πρώτο for θα εκτελεστεί n/2 φορές, το 2ο n/2 και το 3ο for logn φορές. Άρα, η τελική πολυπλοκότητα θα είναι $O(n^2 logn)$.

Άσκηση 22

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
int i = 1, s =1;
while (s <= n) {
   i++;
   s += i;
   printf("*");
}
```

Άσκηση 22

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος:

```
int i = 1, s =1;
while (s <= n) {
   i++;
   s += i;
   printf("*");
}
```

Λύση 22

Παρατηρούμε ότι το s αυξάνει με την τιμή του i το οποίο σε κάθε επανάληψη αυξάνει επίσης κατά 1. Αν k είναι η τελική τιμή του i για την οποία θα σταματήσουμε, το s θα έχει την τιμή: 1+2+3+4+…+k = $\frac{k(k+1)}{2} > n$. Άρα, η τελική πολυπλοκότητα είναι O($\sqrt{n}$).

Άσκηση 23

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος στη χειρότερη περίπτωση:

```
for (int i=0; i<n; i++)
    for (int j=i; j< i*i; j++)
        if (j%i == 0)
        {
            for (int k=0; k<j; k++)
                printf("*");
        }
```

Άσκηση 23

Να υπολογίσετε την πολυπλοκότητα του ακόλουθου προγράμματος στη χειρότερη περίπτωση:

```
for (int i=0; i<n; i++)
    for (int j=i; j< i*i; j++)
        if (j%i == 0)
        {
            for (int k=0; k<j; k++)
                printf("*");
        }
```

Λύση 23

Το πρώτο for θα εκτελεστεί n φορές, το 2ο n*n και το 3ο for j φορές (άρα n*n φορές). Άρα, η τελική πολυπλοκότητα θα είναι $O(n^5)$.

Άσκηση 24

Να δώσετε αλγόριθμο με πολυπλοκότητα O(nlogn), ο οποίος θα εξάγει αν υπάρχουν δύο στοιχεία σε ένα σύνολο τα οποία έχουν άθροισμα κάποια τιμή x.

Άσκηση 24

Να δώσετε αλγόριθμο με πολυπλοκότητα O(nlogn), ο οποίος θα εξάγει αν υπάρχουν δύο στοιχεία σε ένα σύνολο τα οποία έχουν άθροισμα κάποια τιμή x.

Λύση 24

```
A = SORT(A)
n = length[A]
for i = 1  to n do
        if A[i] > 0 and BINARY-SEARCH(A;A[i] - x; 1; n) then
                return true
        end if
end for
return false
```