Ενότητα 1: Εισαγωγή
Ασκήσεις και Λύσεις

Άσκηση 1

Αποδείξτε τη μεταβατική και τη συμμετρική ιδιότητα του Θ.

Λύση

Μεταβατική Ιδιότητα (ορισμός): Αν $f(n) = \Theta(g(n))$ και $g(n) = \Theta(h(n))$ τότε $f(n)=\Theta(h(n))$.

Για να ισχύει $f(n)= \Theta(h(n))$ πρέπει να δείξουμε ότι $f(n)= O(h(n))$ και ότι $f(n)= \Omega(h(n))$.

- **Αποδεικνύουμε πρώτα ότι f(n)= O(h(n)).**

  Αφού $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$, άρα:

  $$\exists\, c_1 \in R^+ \text{ και } n_1 \geq 0 \text{ τ.ω. } f(n) \leq c_1 * g(n), \forall\ n \geq n_1 \quad (1)$$

  Αφού $g(n) = \Theta(h(n)) \Rightarrow g(n)= O(h(n))$, άρα:

  $$\exists\, c_2 \in R^+ \text{ και } n_2 \geq 0 \text{ τ.ω. } g(n) \leq c_2 * h(n), \forall\ n \geq n_2 \quad (2)$$

  Επιλέγουμε $n_0 = \max\{n_1,n_2\}$ και τότε από (1) και (2) $\Rightarrow$

  $$f(n) \leq c_1 * g(n) \leq c_1 * c_2 * h(n), \forall\ n \geq n_0$$

  Άρα, αν επιλέξουμε $c = c_1 * c_2$, ισχύει πως $\exists\, c = c_1 * c_2$ και $n_0 = \max\{n_1,n_2\}$ τ.ω.

  $$f(n) \leq c * h(n), \forall\ n \geq n_0 .$$

  Επομένως, ισχύει πως $f(n) = O(h(n))$.

- **Αποδεικνύουμε στη συνέχεια ότι f(n) = Ω(h(n)).**

  Αφού $f(n) = \Theta(g(n)) \Rightarrow f(n)= \Omega(g(n))$, άρα:

  $$\exists\, c_1 \in R^+ \text{ και } n_1 \geq 0 \text{ τ.ω. } f(n) \geq c_1 * g(n), \forall\ n \geq n_1 \quad (1)$$

  Αφού $g(n) = \Theta(h(n)) \Rightarrow g(n)= \Omega(h(n))$, άρα

  $$\exists\, c_2 \in R^+ \text{ και } n_2 \geq 0 \text{ τ.ω. } g(n) \geq c_2 * h(n), \forall\ n \geq n_2 \quad (2)$$

  Επιλέγουμε $n_0 = \max\{n_1,n_2\}$ και τότε από (1) και (2) $\Rightarrow$

  $$f(n) \geq c_1 * g(n) \geq c_1 * c_2 * h(n), \forall\ n \geq n_0$$

  Άρα, αν επιλέξουμε $c = c_1 * c_2$, ισχύει πως $\exists\, c = c_1 * c_2$ και $n_0 = \max\{n_1,n_2\}$ τ.ω.

  $$f(n) \geq c * h(n), \forall\ n \geq n_0 .$$

  Άρα $f(n) = \Omega(h(n))$.

Αφού ισχύει ότι $f(n) = O(h(n))$ και ότι $f(n) = \Omega(h(n))$, συμπεραίνουμε ότι $f(n) = \Theta(h(n))$, όπως απαιτείται.

Συμμετρική ιδιότητα (ορισμός): $f(n) = \Theta(g(n))$ αν και μόνο αν $g(n) = \Theta(f(n))$.

Θα αποδείξουμε πως αν $f(n) = \Theta(g(n))$ τότε ισχύει πως $g(n) = \Theta(f(n))$. Η απόδειξη του αντίστροφου, δηλαδή η απόδειξη πως αν $g(n) = \Theta(f(n))$ τότε ισχύει πως $f(n) = \Theta(g(n))$, είναι συμμετρική.

Για να δείξω ότι $g(n) = \Theta(f(n))$ πρέπει να δείξω ότι $g(n) = O(f(n))$ και $g(n) = \Omega(f(n))$ (1)

Αφού $f(n) = \Theta(g(n))$ τότε $f(n) = O(g(n))$. Άρα $\exists\ c_1 \in R^+$ και $n_1 \geq 0$ τ.ω.:

$$f(n) \leq c_1 \cdot g(n) ,\ \forall\ n \geq n_1 \Rightarrow g(n) \geq 1/c_1 * f(n) ,\ \forall\ n \geq n_1$$

Επομένως, αν επιλέξουμε $c = 1/c_1$ και $n_0 = n_1$ προκύπτει ότι $g(n) \geq c * f(n),\ \forall\ n \geq n_0$.

Άρα, $g(n) = \Omega(f(n))$. (2)

Επιπρόσθετα, αφού $f(n) = \Theta(g(n))$ τότε $f(n) = \Omega(g(n))$. Άρα $\exists\ c_2 \in R^+$ και $n_2 \geq 0$ τ.ω.:

$$f(n) \leq c_2 \cdot g(n),\ \forall\ n \geq n_2 \Rightarrow g(n) \leq 1/c_2 * f(n) ,\ \forall\ n \geq n_2$$

Επομένως, αν επιλέξουμε $c = 1/c_2$ και $n_0 = n_2$ προκύπτει ότι $g(n) \leq c * f(n),\ \forall\ n \geq n_0$.

Άρα, $g(n) = O(f(n))$. (3)

Από (2) και (3) $\Rightarrow$ $g(n) = \Theta(f(n))$, όπως απαιτείται.

Άσκηση 2

1. Ισχύει ότι $\sqrt{n^5}$ $\log(\sqrt{n^5}) = O(n^3)$;
2. Ισχύει ότι $\log(\sqrt{n}) = \Theta(\log(n))$;

Λύση

1. Εξετάζουμε εάν $\sqrt{n^5}$ $\log(\sqrt{n^5}) \in O(n^3)$.

   Αναζητούμε $c \in R^+$ και ακέραιο $n_0 \geq 0$ τ.ω.

   $$\sqrt{n^5} \log(\sqrt{n^5}) \leq c * n^3,\ \forall\ n \geq n_0$$

   $$\Leftrightarrow\ n^{5/2} * \log n^{5/2} \leq c * n^3 \Rightarrow n^{5/2} * (5/2) \log n \leq c * n^3$$

   Ισχύει ότι:

   $$2.5 * n^{2.5} * \log n \leq 2.5 * n^{2.5} * \sqrt{n} = 2.5 * n^3 ,\ \forall\ n \geq 4$$

   Επομένως, αν επιλέξουμε $c = 2.5$ και $n_0 = 4$ προκύπτει ότι

   $$\sqrt{n^5} \log(\sqrt{n^5}) \leq c * n^3 ,\ \forall\ n \geq n_0.$$

   Άρα, ισχύει ότι $\sqrt{n^5}$ $\log(\sqrt{n^5}) \in O(n^3)$.

2. Εξετάζουμε εάν $\log(\sqrt{n}) \in O(\log(n))$.

   Αναζητούμε $c \in R^+$ και ακέραιο $n_0 \geq 0$ τ.ω.:

   $$\log(\sqrt{n}) \leq c * \log(n),\ \forall\ n \geq n_0$$

   $$\Leftrightarrow \log n^{1/2} \leq c * \log(n)$$

   $$\Leftrightarrow ½ \log(n) \leq c * \log(n)$$

   Επομένως, αν επιλέξουμε $c = 1/2$ και $n_0 = 1$ προκύπτει ότι

   $$\log(\sqrt{n}) \leq c * \log(n),\ \forall\ n \geq n_0$$

   Άρα, ισχύει ότι $\log(\sqrt{n}) \in O(\log n)$.

   Ομοίως, εξετάζουμε και αν ισχύει ότι $\log(\sqrt{n}) \in \Omega(\log n))$.

<u>Άσκηση 3</u>

1. Αποδείξτε επαγωγικά ότι αν $T(0) = 0$ και $T(n) = 2* T(n-1) + 1$, $n > 0$, τότε $T(n) = 2^n – 1$.
2. Θεωρήστε τη συνάρτηση $f : N \rightarrow N$ που ορίζεται ως εξής :

   $$f(0) = 1,$$

   $$f(1) = 2,$$

   $$f(n) = 4 * f(n-2) + 2^n \text{ αν } n > 1.$$

   Αποδείξτε επαγωγικά ότι για κάθε ακέραιο $n \geq 3$ ισχύει ότι

   $$f(n) \leq 3 * n * 2^{n-2}$$

Λύση:

1. Με επαγωγή ως προς n.

   Βάση επαγωγής (n=1): Η αναδρομική σχέση, για n=1 μας δίνει

   $$T(1) = 2*T(1-1) + 1 = 2*T(0) +1 = 0 + 1 = 1.$$

   Επιπρόσθετα, ισχύει πως $2^1 -1 = 2-1 = 1 = T(1)$, όπως απαιτείται.

   Επαγωγική Υπόθεση: Θεωρούμε οποιονδήποτε ακέραιο $n > 1$. Έστω ότι ο ισχυρισμός ισχύει για $n -1$, δηλαδή υποθέτουμε πως ισχύει ότι $T(n-1) = 2^{n-1} - 1$.

   Επαγωγικό Βήμα: Θα δείξουμε πως ο ισχυρισμός ισχύει και για την τιμή n, δηλαδή θα δείξουμε πως $T(n) = 2^n - 1$. Από την αναδρομική σχέση, συμπεραίνουμε ότι

$$T(n) = 2T(n-1) +1.$$

Επομένως, από την επαγωγική υπόθεση προκύπτει:

$$T(n) = 2T(n-1) +1 = 2(2^{n-1} – 1) +1 = 2^n -2 +1 = 2^n-1,$$

όπως απαιτείται.

2. Με επαγωγή ως προς n.

Βάση επαγωγής (n=3): Από την αναδρομική σχέση προκύπτει ότι:

$$f(3) = 4 * f(1) +2^3 = 4 * 2 + 8 = 16 \qquad (1)$$

Επιπρόσθετα, ισχύει ότι $3 * 3 * 2^1 = 18 \geq 16 = f(3)$ (από (1)).

Άρα, ο ισχυρισμός ισχύει για n = 3.

Επαγωγική Υπόθεση: Θεωρούμε οποιοδήποτε ακέραιο n > 3 και υποθέτουμε ότι ο ισχυρισμός ισχύει για κάθε τιμή n’ τ.ω., $3 \leq n' < n$, δηλαδή υποθέτουμε ότι ισχύει

$$f(n') \leq 3 * (n') * 2^{n'-2}, \qquad \forall\, n' \text{ τ.ω. } 3 \leq n' < n \qquad (2)$$

Επαγωγικό Βήμα: Θα αποδείξουμε ότι ο ισχυρισμός ισχύει για την τιμή n.

Από την αναδρομική σχέση προκύπτει ότι $f(n) = 4 * f(n-2) +2^n$. (3)

Διακρίνουμε περιπτώσεις.

<u>Περίπτωση 1</u>: n = 4. Στην περίπτωση αυτή, ισχύει ότι $f(4) = 4 * f(2) + 2^4 = 4 * f(2) + 16$.

Από την αναδρομική σχέση προκύπτει ότι $f(2) = 4*f(1) + 2^2 = 4 * 1 + 4 = 8$.

Επομένως, $f(4) = 4 * 8 + 16 = 48$.

Επιπρόσθετα, ισχύει ότι $3 * 4 * 2^2 = 12 * 4 = 48 \geq f(4)$. Επομένως, ο ισχυρισμός ισχύει σε αυτή την περίπτωση. (Είναι αξιοσημείωτο ότι σε αυτή την περίπτωση, δεν μπορώ να εφαρμόσω την επαγωγική υπόθεση για $n' = 2$, αφού η επαγωγική υπόθεση ισχύει μόνο για κάθε $n' \geq 3$.

<u>Περίπτωση 2</u>: n > 4.

Από (1) $\Rightarrow$

$$\begin{aligned}
f(n) &= 4 * f(n-2) +2^n \\
&\leq 4 * (3 * (n-2) * 2^{n-4}) + 2^n \qquad \text{(από επαγωγική υπόθεση, όπου } n' = n-2) \\
&= 2^2 (3 * (n-2) * 2^{n-4}) + 2^2 * 2^{n-2} \\
&\leq 3 * (n-2) * 2^{n-2} + 4 * 2^{n-2} \\
&= 3 * n * 2^{n-2} – 6 * 2^{n-2} + 4 * 2^{n-2} \\
&= 3 * n * 2^{n-2} - 2 * 2^{n-2} \\
&< 3 * n * 2^{n-2},
\end{aligned}$$

όπως απαιτείται. (Είναι αξιοσημείωτο ότι, αφού $n > 4$ σε αυτή την περίπτωση, ισχύει ότι $n-2 \geq 3$ και άρα μπορώ να εφαρμόσω την επαγωγική υπόθεση).

Άσκηση 4

Βρείτε την τάξη (βάσει των συμβολισμών Ο,Ω,Θ) της χρονικής πολυπλοκότητας T(n) του ακόλουθου αλγόριθμου

```
        Procedure f (integer n){
                for (i=1; i ≤  n; i++)
                        for(k = n; k ≤ n+5; k++)
                                x = x+1;
}
```

Λύση

Το i θα πάρει n διαφορετικές τιμές (i = 1, 2, …, n). Για κάθε μια από αυτές τις τιμές, θα εκτελεστεί ο εσωτερικός for βρόγχος. Άρα, ο εσωτερικός for βρόγχος θα εκτελεστεί συνολικά n φορές. Κάθε φορά που εκτελείται ο εσωτερικός for βρόγχος, η μεταβλητή k παίρνει 6 διαφορετικές τιμές (k = n. n+1, n+2, n+3, n+4, n+5). Επομένως, κάθε φορά που εκτελείται ο εσωτερικός for βρόγχος, η εντολή x = x+1 εκτελείται 6 φορές. Συμπεραίνουμε πως ο συνολικός αριθμός φορών που θα εκτελεστεί η εντολή x = x+1 είναι 6*n. Άρα, η χρονική πολυπλοκότητα T(n) της f() είναι T(n) = O(n).

Συνοπτική ιχνηλάτιση της εκτέλεσης της f() παρουσιάζεται στη συνέχεια.

Εξωτερικό for loop (ανακύκλωση i=1):
    (Εσωτερικό for loop:)
    k =n
    k=n+1
    k=n+2
    k=n+3
    k=n+4
    k=n+5
    (τέλος εκτέλεσης εσωτερικού for loop)
Εξωτερικό for loop (ανακύκλωση i=2):
    (Εσωτερικό for loop:)
    k =n
    k=n+1
    k=n+2
    k=n+3
    k=n+4
    k=n+5
    (τέλος εκτέλεσης εσωτερικού for loop)
.
.
.
Εξωτερικό for loop (ανακύκλωση i=n):

(Εσωτερικό for loop:)
k =n
k=n+1
k=n+2
k=n+3
k=n+4
k=n+5
(τέλος εκτέλεσης εσωτερικού for loop)
(Τέλος εκτέλεσης εξωτερικού for loop).

Άσκηση 5

Δίνεται ο αλγόριθμος Binary Search(), ο οποίος χρησιμοποιείται για την αναζήτηση ενός στοιχείου σε έναν ήδη ταξινομημένο πίνακα.

```
Index BinarySearch(Type A[0…N-1], Type value, Index low, Index high) {
    if (high < low)
        return -1;                      //not found
    mid = low + (high - low) / 2;
    if (A[mid] > value)
        return BinarySearch(A, value, low, mid-1);
    else if (A[mid] < value)
        return BinarySearch(A, value, mid+1, high);
    else
        return mid;                     //found
}
```

1. Παρουσιάστε σύντομη περιγραφή του τρόπου λειτουργίας του αλγορίθμου.
2. Ιχνηλατήστε την BinarySearch (A, 14, 0, 9) για την περίπτωση που A= [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]. Πρέπει να παρουσιαστούν όλες οι αναδρομικές κλήσεις της BinarySearch με τη σειρά που καλούνται καθώς και οι τιμές των παραμέτρων A, value, low, high σε κάθε κλήση. Πρέπει επίσης να παρουσιαστεί ο χώρος στη μνήμη που κατανέμεται για την εκτέλεση των αναδρομικών κλήσεων της BinarySearch.
3. Παρουσιάστε αναδρομική σχέση που να περιγράφει τη χρονική πολυπλοκότητα T(n) της BinarySearch για την περίπτωση που $n = 2^k$, για κάποιο k (δηλαδή για την περίπτωση που το n είναι μια δύναμη του 2).
4. Τι τάξης είναι η πολυπλοκότητα της BinarySearch, αποδείξτε τον ισχυρισμό σας.

Λύση

1. **Περιγραφή**

   Η BinarySearch() βασίζεται στην τεχνική του διαίρει και κυρίευε, η οποία περιλαμβάνει τρία βήματα:

   a. Διαίρεση του προβλήματος σε διάφορα υποπροβλήματα που είναι παρόμοια με το αρχικό πρόβλημα αλλά μικρότερου μεγέθους.

b. Κυριαρχία επί των υποπροβλημάτων, επιλύοντας τα αναδρομικά μέχρι αυτά να γίνουν αρκετά μικρού μεγέθους οπότε και τα επιλύουμε απευθείας.
c. Συνδυασμός των επιμέρους λύσεων των υποπροβλημάτων ώστε να συνθέσουμε μια λύση του αρχικού προβλήματος.

Η συνάρτηση παίρνει ως ορίσματα έναν ταξινομημένο πίνακα A, μία τιμή προς αναζήτηση value, και δύο ακεραίους low και high οι οποίοι υποδηλώνουν τα όρια του πίνακα μέσα στα οποία θα γίνει η αναζήτηση (δηλαδή η αναζήτηση για την τιμή value θα πραγματοποιηθεί στο μέρος A[low…high] του πίνακα) .
Η συνάρτηση βρίσκει το μεσαίο στοιχείο mid του προς εξέταση πίνακα A[low…high] και ελέγχει αν το στοιχείο στη θέση A[mid] είναι μεγαλύτερο ή μικρότερο από την προς αναζήτηση τιμή value. Στην περίπτωση που είναι μικρότερο, κάνει αναδρομική κλήση της BinarySearch(A, value, mid+1, high), δηλαδή αναζητά την τιμή value στο άνω μισό του πίνακα A (αφού το κάτω μισό περιέχει στοιχεία μικρότερα του A[mid] και άρα, αφού ο πίνακας είναι ταξινομημένος, μικρότερα και του προς αναζήτηση στοιχείου value). Αντίχτοιχα, αν το A[mid] είναι μεγαλύτερο κάνει αναδρομική κλήση της BinarySearch(A, value, low, mid-1) δηλαδή αναζητά την τιμή value στο κάτω μισό του πίνακα A (αφού το άνω μισό περιέχει στοιχεία μεγαλύτερα του A[mid] και άρα, αφού ο πίνακας είναι ταξινομημένος, μεγαλύτερα και του προς αναζήτηση στοιχείου value). Αν το στοιχείο δε βρεθεί στον πίνακα (δηλαδή φτάσουμε στο σημείο όπου high < low τότε η συνάρτηση επιστρέφει -1. Αν βρεθεί στοιχείο με τιμή value, τότε επιστρέφεται η θέση (mid) του πίνακα στην οποία βρέθηκε.

2. Ιχνηλάτιση

Ταξινομημένος πίνακας: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

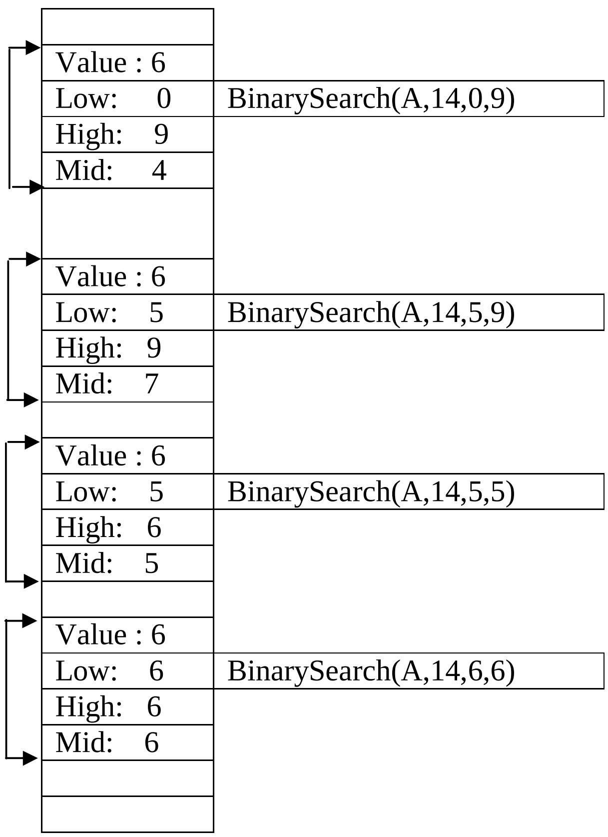
```
BinarySearch ([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 14, 0, 9):
---------------------------------------------------------------
if (9 < 0)                                  --> αποτιμάται σε false
mid = 0+(9-0)/2 = 4
if (10 > 14)                                --> αποτιμάται σε false
else if (10 < 14)                           --> αποτιμάται σε true
    BinarySearch ([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 14, 5, 9):
    ---------------------------------------------------------
    if (9 < 5)                              --> αποτιμάται σε false
    mid = 5+(9-5)/2 = 7
    if (16 > 14)                            --> αποτιμάται σε true
            BinarySearch ([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 14, 5, 6):
            ----------------------------------------------
                    if (6 < 5)              --> αποτιμάται σε false
                    mid = 5+(6-5)/2 = 5
                    if (12 > 14)            --> αποτιμάται σε false
                    else if (12 < 14)       --> αποτιμάται σε true
                            BinarySearch ([2, 4, 6, 8, 10, 12, 14, 16, 18, 20], 14, 6, 6):
                            ---------------------------------------------
                                    if (6 < 6)      --> αποτιμάται σε false
                                    mid = 6 + (6 - 6) / 2 = 6
```

```
if (14 > 14)          --> αποτιμάται σε false
else if (14 < 14)     --> αποτιμάται σε false
else
        return 6      //Found
```

Το στοιχείο βρίσκεται στη θέση 6 του πίνακα A[2, 4, 6, 8, 10, 12, 14, 16, 18, 20] -->A[6]=14.

Μνήμη

Ο πίνακας A = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20] σε όλες τις κλήσεις της BinarySearch().



3. Αναδρομική Σχέση

Η αναδρομική σχέση είναι η εξής:

$$T(0) = c_1 \qquad (1)$$

$$T(n) = T(n/2) + c_2 \qquad (2)$$

Η αναδρομική σχέση προκύπτει ως εξής. Για να προκύψει το μέρος (1) της αναδρομικής σχέσης, μετράμε πόσες στοιχειώδεις εντολές θα εκτελέσει ο αλγόριθμος αν n = 0, δηλαδή με παράμετρο ένα μέρος του πίνακα μηδενικού μεγέθους (δηλαδή αν high < low). Σε αυτή την περίπτωση θα εκτελεστούν οι γραμμές 1 και 2 και άρα σε αυτή την περίπτωση δεν θα εκτελεστούν περισσότερες από 3 στοιχειώδεις εντολές (μια για την αρχική κλήση της BinarySearch(), μια για τον έλεγχο της if της γραμμής 1 και μια για την εκτέλεση της return της γραμμής 2). Είναι αξιοσημείωτο πως, αφού το 3 είναι μια σταθερά, δεν χρειάζεται να είμαστε ακριβείς στην μέτρηση των στοιχειωδών εντολών σε αυτή την περίπτωση, δηλαδή μπορούμε να γράψουμε πως $T(0) = c_1$, όπως παραπάνω (παρότι τώρα γνωρίζουμε πως $c_1 = 3$ για τη BinarySearch() που μελετάμε).

Για να εξάγουμε το μέρος (2) της αναδρομικής σχέσης, μετράμε πόσες στοιχειώδεις εντολές θα εκτελέσει ο αλγόριθμος σε μια οποιαδήποτε κλήση της BinarySearch() χωρίς να υπολογίσουμε το κόστος για τις αναδρομικές κλήσεις. Στην περίπτωση της BinarySearch() το κόστος αυτός είναι το κόστος εκτέλεσης των γραμμών 1, 3 και 4, ή 1, 3, 4 και 6, ή 1, 3, 4, 6 και 9 (ανάλογα με την περίπτωση κάθε φορά). Αν υπολογίσουμε το κόστος εκτέλεσης αυτών των στοιχειωδών εντολών, συμπεραίνουμε ότι αυτό είναι ίσο με κάποια σταθερά c_2. Σημειώνουμε ότι δεν είναι σημαντικό να προσδιορίσουμε την σταθερά αυτή επ' ακριβώς, αφού η τιμή της δεν επηρεάζει τη λύση της αναδρομικής εξίσωσης (και άρα η τάξη της πολυπλοκότητας του αλγορίθμου θα προκύψει να είναι η ίδια, όποια και αν είναι η πραγματική τιμή της σταθεράς c_2).

4. Λύση Αναδρομικής Σχέσης – Τάξη Χρονικής Πολυπλοκότητας

$$
\begin{aligned}
T(n) &= T(n/2) + c_2 \\
&= (T(n/2^2) + c_2) + c_2 = T(n/2^2) + 2 * c_2 \\
&= (T(n/2^3) + c_2) + 2 * c2 = T(n/2^3) + 3 * c_2 \\
&= .... \\
&= T(n/2^k) + k * c_2
\end{aligned}
$$

Η επαναληπτική αντικατάσταση σταματάει όταν $n/2^k < 1 \Rightarrow k > \log n$. Τότε:

$$
\begin{aligned}
T(n) &\le T(0) + (\log n + 1) * c_2 \\
&= c_1 + (\log n + 1) * c_2 \\
&= \log n + (c_1 + c_2) \\
&= O(\log n).
\end{aligned}
$$

Ευχαριστίες

Ευχαριστούμε θερμά την, επί τρία χρόνια, βοηθό του μαθήματος Χρυσή Μπιρλιράκη για την παραγωγή της ηλεκτρονικής έκδοσης του παραπάνω υλικού.