

01_ΕΙΣΑΓΩΓΗ

Δυνάμεις

Για $x \neq 0$ και x, y πραγματικούς ισχύουν οι βασικές ιδιότητες

$$\begin{aligned}x^0 &= 1, & x^1 &= x, & x^{-1} &= \frac{1}{x}, & x^y x^z &= x^{y+z}, \\(x^y)^z &= x^{yz} = (x^z)^y, & \sqrt[z]{x^y} &= x^{\frac{y}{z}}.\end{aligned}$$

Για πραγματικές σταθερές $x > 1, y$ ισχύει ότι

$$\lim_{n \rightarrow \infty} \frac{n^y}{x^n} = 0.$$

Λογάριθμοι

Λογάριθμος ενός πραγματικού αριθμού $x > 0$ με βάση τον πραγματικό αριθμό $b > 0, b \neq 1$, συμβολικά $\log_b x$, είναι ο πραγματικός αριθμός y τέτοιος ώστε $b^y = x$.

Όταν εμφανίζεται ο λογάριθμος μέσα σε μία παράσταση τότε έχει σαν όρισμα τον πρώτο όρο που εμφανίζεται αμέσως μετά. Για παράδειγμα, ο συμβολισμός $\log_b x + z$ υποδηλώνει $(\log_b x) + z$. Περαιτέρω συμβάσεις που ακολουθούνται στο συμβολισμό δηλώνονται από τις ακόλουθες ισότητες

$$\begin{aligned}\lg n &= \log_2 n \quad (\text{δυαδικός λογάριθμος}), \\ \ln n &= \log_e n \quad (\text{φυσικός λογάριθμος}), \\ \log n &= \log_{10} n \quad (\text{δεκαδικός λογάριθμος}), \\ \lg^k n &= (\lg n)^k \quad (\text{υψωση σε δύναμη}), \\ \lg \lg n &= \lg(\lg n) \quad (\text{σύνθεση})\end{aligned}$$

Βασικές ιδιότητες των λογαρίθμων απεικονίζονται στη συνέχεια.

$$\begin{aligned}a &= b^{\log_b a}, & \log_b ac &= \log_b a + \log_b c, & \log_b a^c &= c \log_b a, \\ \log_b a &= \frac{\log_c a}{\log_c b}, & \log_b a &= \frac{1}{\log_b a}, & a^{\log_b c} &= c^{\log_b a},\end{aligned}$$

όπου a, b, c θετικοί πραγματικοί αλλά και μεγαλύτεροι της μονάδας όταν χρησιμοποιούνται σαν βάσεις λογαρίθμων στις παραπάνω εκφράσεις. Επιπλέον για το νεπέριο

$$\log_b \left\{ \frac{x}{y} \right\} = \log_b x - \log_b y$$

$$\log(\log n) \neq \log n \cdot \log n$$

Ταβάνια και Πατώματα

Δεδομένου ενός πραγματικού αριθμού x , το *πάτωμα* του x , ονομαστικά $\lfloor x \rfloor$, ορίζεται σαν ο μεγαλύτερος ακέραιος που είναι μικρότερος-ίσος με τον x . Αντίστοιχα, το *ταβάνι* του x , ονομαστικά $\lceil x \rceil$, ορίζεται σαν ο μικρότερος ακέραιος που είναι μεγαλύτερος-ίσος με τον x . Για τις συναρτήσεις αυτές ισχύουν οι εξής σχέσεις (a, b θετικοί ακέραιοι):

$$x - 1 \leq \lfloor x \rfloor \leq x \leq \lceil x \rceil \leq x + 1, \quad (13.8)$$

$$\frac{a-b+1}{b} \leq \lfloor \frac{a}{b} \rfloor \leq \frac{a}{b} \leq \lceil \frac{a}{b} \rceil \leq \frac{a+b-1}{b}, \quad (13.9)$$

$$\lfloor \lfloor n/a \rfloor / b \rfloor = \lfloor n/ab \rfloor, \quad (13.10)$$

$$\lceil \lceil n/a \rceil / b \rceil = \lceil n/ab \rceil, \quad (13.11)$$

$$\lfloor n/2 \rfloor + \lceil n/2 \rceil = n, \quad (13.12)$$

$$a \bmod n = a - n \lfloor \frac{a}{n} \rfloor. \quad (13.13)$$

Αθροίσματα

Βασικές ισότητες αθροισμάτων είναι οι ακόλουθες

$$\sum_{i=0}^n i = \frac{n(n+1)}{2}, \quad (13.16)$$

$$\sum_{i=0}^n i^2 = \frac{1}{6}n(n+1)(2n+1) = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}. \quad (13.17)$$

$$\sum_{i=0}^{n-1} x^i = \frac{1-x^n}{1-x}, \quad n > 0, \quad (13.18)$$

$$\sum_{i=m}^{n-1} x^i = \frac{x^m - x^n}{1-x}, \quad 0 \leq m < n, \quad (13.19)$$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}, \quad |x| < 1, \quad (13.20)$$

$$\sum_{i=0}^{\infty} ix^i = \frac{x}{(1-x)^2}, \quad |x| < 1. \quad (13.21)$$

Δεδομένου ενός αθροίσματος $\sum_{i=0}^n a_i$, αν ισχύει, για κάποιο $r < 1$, ότι

$$a_{k+1}/a_k \leq r, \quad \text{για } k \in \{0, \dots, n-1\},$$

τότε επίσης ισχύει

$$\sum_{i=0}^n a_i \leq \sum_{i=0}^{\infty} a_0 r^i = a_0 \sum_{i=0}^{\infty} r^i = a_0 \frac{1}{1-r}. \quad (13.26)$$

$$\sum_{i=A}^B c = c \sum_{i=A}^B 1 \quad c : \text{συστερι}$$

$$\sum_{i=A}^B 1 = B - A + 1$$

$$\sum_{i=1}^n (A+B) = \sum_{i=1}^n A + \sum_{i=1}^n B$$

$$\sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1}$$

Επαγωγή

Μία αποδεικτική διαδικασία που χρησιμοποιείται αρκετά συχνά στην ανάλυση των αλγορίθμων είναι η μέθοδος της επαγωγής. Η μέθοδος αυτή χρησιμοποιείται όταν θέλουμε να αποδείξουμε ότι μία πρόταση $P(n)$ ισχύει για κάθε φυσικό αριθμό n . Η απόδειξη με τη χρήση επαγωγής μπορεί να γίνει ώς εξής.

- *Βάση Επαγωγής:* Αποδεικνύουμε ότι η πρόταση ισχύει για μία ‘μικρή’ τιμή, έστω a , της n (π.χ. $n = 0$ ή $n = 1$ ή $n = 2, \dots$)
- *Επαγωγική υπόθεση:* Θεωρούμε ότι ισχύει για μία τιμή $n = k$ ($k > a$). Δηλαδή θεωρούμε ότι ισχύει η $P(k)$.
- *Επαγωγικό Βήμα:* Θα πρέπει να αποδείξουμε ότι η πρόταση ισχύει P για μία τιμή $n = k + 1$. Δηλαδή θα πρέπει να αποδείξουμε ότι ισχύει $P(k + 1)$ χρησιμοποιώντας την επαγωγική υπόθεση και τη Βάση της επαγωγής.

Αριθμητικές Σχέσεις

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1, x \in \mathbb{R}$$

$$\lceil \frac{x}{y} \rceil \leq \frac{x+y-1}{y}$$

$$\lfloor \frac{x}{y} \rfloor \geq \frac{x-y+1}{y}$$

Ταυτότητες

$$(\alpha + \beta)^2 = \alpha^2 + 2\alpha\beta + \beta^2$$

$$(\alpha - \beta)^2 = \alpha^2 - 2\alpha\beta + \beta^2$$

$$(\alpha + \beta)^3 = \alpha^3 + 3\alpha^2\beta + 3\alpha\beta^2 + \beta^3$$

$$(\alpha - \beta)^3 = \alpha^3 - 3\alpha^2\beta + 3\alpha\beta^2 - \beta^3$$

$$\alpha^2 - \beta^2 = (\alpha + \beta)(\alpha - \beta)$$

Υπολογισμός ΣΥΒ Αλγορίθμου

Αλγόριθμος 2 Διψήφιοι ακέραιοι με κανένα ίδιο ψηφίο

Απαιτείται:

Επιστρέφεται: Ακέραιοι από 10 ως 99 με διαφορετικά ψηφία

```
1: function NeqDigits
2:   for i  $\leftarrow$  1; i  $\leq$  9; i  $\leftarrow\rightleftharpoons$  do
3:     for j  $\leftarrow$  0; j  $\leq$  9; j  $\leftarrow\rightleftharpoons$  do
4:       if i  $\neq$  j then
5:         num  $\leftarrow$  10  $\cdot$  i + j;
6:         Output num;
7:       end if
8:     end for
9:   end for
10: end function
```

9: for $i \leftarrow 1; i \leq 9; i++$ do

Πόσες loops κάνει;
 $(b - a + 1 = \text{Αριθμός των loops})$

στον b: η τελεταιά τιμή του i, στην
απόστρα στην εκτέλεση της loop για τελευταία
ταινία χρόνου ($\text{για } i = 9$)

a: η αρχή της τιμής του i, στην απόστρα
εκτέλεση της loop για αρχή χρόνου
($\text{για } i = 1$)

Άρα $\# \text{loops} = b - a + 1 = 9 - 1 + 1 = 9$

Πόσα ΣΥΒ εκτελώνται συνολικά στο πλαίσιο
for;

$$3 \cdot 9 + 2 = 29 \text{ ΣΥΒ}$$

Πόσα ΣΥΒ εκτελώνται σε κάθε loop μέσα στη for

1 = loop: $i \leq 1$; $i \leq 9$; $i \leftarrow i+1$ (i $i++$)

$$\textcircled{1} + \textcircled{1 + 2} = 4 \Sigma y_B$$

2 = loop: $i \leq 9$; $i \leftarrow i+1$ (i $i++$)

$$\textcircled{1 + 2} = 3 \Sigma y_B$$

⋮

8 = loop: $i \leq 9$; $i \leftarrow i+1$ (i $i++$)

$$\textcircled{1 + 2} = 3 \Sigma y_B$$

9 = loop: $i \leq 9$

$$\textcircled{1} \Sigma y_B$$

Kan over 9 loop skeletoontrek
3 Σy_B , dan

$$3 \cdot 9 + 2 = 29 \Sigma y_B$$

(1)

OTO omga
Tys for

3: for $j \leq 0; j \leq 9; j++ \text{ do}$

Mε το ίσω σκελετό σινε και
στην 1η for-loop ...

$$\# \text{loops} = 9 - 0 + 1 = 10 \text{ loops}$$

αριθμός σινε της for εκτέλεσης
νταν (

$$3 \cdot 10 + 2 = 32 \Sigma YB$$

Επειδή, είναι nested-loop, οι
εκτελέσεις $\times 9$ ηρόεις for-loop

O γύρωτικός αριθμός των ΣYB ήσαν
Οι εκτελέσεις στην for είναι:

$$(3 \cdot 10 + 2) \cdot 9 = 288 \Sigma YB$$

(2)

4: if $i \neq j$ then

$i \neq j$ given $\Delta \Sigma YB$ (εν δικλήσι) }
είναι κωντός loop.

O γενικός αριθμός των loops
given # loops • # loops = $9 \cdot 10 = 90$
 $i=1$ for $i=10$ for loops

Άρα συνολικά 4 εκτελίσεις

$$90 \cdot 1 = 90 \Sigma YB \quad (3)$$

5: num $\leftarrow 10 \cdot i + j ;$

num $\in 10 \cdot i + j \quad 3 \Sigma YB$

ΠΡΟΣΟΧΗ: δεν εκτελίστε την κωντός loop

To i αρχική την δέκατη

To j αρχική την μονάδα

H συνθήκη στην γραφή 4 είναι false
στην $i=j$, 1812656

$I = 1$ for
loop

$J = 1$ for
loop

	<u>i</u>	<u>j</u>
1	1	0
1	1	1
1	1	2
1	1	3
:	:	
1	1	9

2	0	
2	1	
2	2	$\rightarrow i = j$
2	3	
:	:	
2	9	

Δημήτριος

loop T_{15}

$I = 1$ for,

1 copy and T_0

10 loops $T_{15} \cdot 2^9$

for loop, T_0

$i = j$ και επομ.

vws, or entries

T_{15} γραμμής 5

ΣΚΤΕΛΩΝΤΑΙ

9/10

loops ήν T0 έχει γ

2^9 for loop

Άρα $O T_{15}$

γραμμής 5 ΣΚΤΕΛΩΝΤΑΙ

(4)

$$9 \cdot (10 - 1) \cdot 3 = 243 \text{ ΣΥΒ}$$

Overall ΣΥΒ : ① + ② + ③ + ④ \Leftrightarrow

$$29 + 288 + 90 + 243 = 650 \text{ ΣΥΒ}$$

Υπολογισμός ΣΥΒ Αλγορίθμου με Αθροίσματα

Αλγόριθμος 4

Απαιτείται: ακέραιος n

Επιστρέφεται: ακέραιος r

```
1: function Guess01(int n)
2:   r ← 0;
3:   for i ← 1; i ≤ n - 1; i ++ do
4:     for j ← i + 1; j ≤ n; j ++ do
5:       for k ← 1; k ≤ j; k ++ do
6:         r++;
7:       end for
8:     end for
9:   end for
10:  return r;
11: end function
```

Λύση. Η τιμή της μεταβλητής r στον Αλγόριθμο 4 δίνεται από την έκφραση

$$\begin{aligned}
 r &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^j 1 \\
 &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n j \\
 &= \sum_{i=1}^{n-1} \left(\sum_{j=1}^n j - \sum_{j=1}^i j \right) = \sum_{i=1}^{n-1} \left(\frac{n(n+1)}{2} - \frac{i(i+1)}{2} \right) \\
 &= \sum_{i=1}^{n-1} \frac{n(n+1)}{2} - \sum_{i=1}^{n-1} \frac{i(i+1)}{2} = \frac{(n+1)n(n-1)}{2} - \frac{1}{2} \left(\sum_{i=1}^{n-1} i^2 + \sum_{i=1}^{n-1} i \right) \\
 &= \frac{(n+1)n(n-1)}{2} - \frac{1}{2} \left(\frac{(n-1)n(2(n-1)+1)}{6} + \frac{n(n-1)}{2} \right) \\
 &= \frac{(n+1)n(n-1)}{2} - \frac{n(n-1)}{2} \left(\frac{2n-1}{6} + \frac{1}{2} \right) \\
 &= \frac{(n+1)n(n-1)}{2} - \frac{n(n-1)(n+1)}{6} = \frac{1}{3} n(n-1)(n+1).
 \end{aligned}$$

Λύση. Στη Γραμμή 6 του αλγόριθμου εκτελούνται 2 ΣΥΒ. Ο αριθμός των επαναλήψεων του βρόχου των γραμμών 5-7 είναι ίσος με j - επομένως ο συνολικός αριθμός των ΣΥΒ που εκτελείται εσωτερικά του βρόχου είναι $2j$ (δες Άσκηση 1). Στον έλεγχο του βρόχου, σε σχέση με τον δείκτη k , εκτελούνται $3j + 2$ ΣΥΒ (δες Άσκηση 1). Επομένως ο συνολικός αριθμός των ΣΥΒ του εσωτερικού βρόχου είναι

$$A(j) = 2j + 3j + 2 = 5j + 2.$$

Ο αριθμός των επαναλήψεων που εκτελούνται από το βρόχο των γραμμών 4-8 είναι ίσος με $n - i$. Ο συνολικός αριθμός των ΣΥΒ που εκτελούνται εσωτερικά του βρόχου είναι ίσος με $\sum_{j=1}^{n-i} A(j)$. Επίσης σε σχέση με τον δείκτη j στον έλεγχο του βρόχου εκτελούνται $3(n - i) + 2$ ΣΥΒ. Ο συνολικός αριθμός των ΣΥΒ του βρόχου είναι

$$B(i) = \sum_{j=1}^{n-i} A(j) + 3(n - i) + 2 = \sum_{j=1}^{n-i} (5j + 2) + 3(n - i) + 2.$$

Ο αριθμός των επαναλήψεων του εξωτερικού βρόχου είναι ίσος με $n - 1$. Άρα ο συνολικός αριθμός των ΣΥΒ που εκτελούνται εσωτερικά του βρόχου είναι ίσος με $\sum_{i=1}^{n-1} B(i)$. Σε σχέση με τον δείκτη i στον έλεγχο του βρόχου εκτελούνται $3(n - 1) + 2$ ΣΥΒ. Επίσης εκτελείται μία εικρύρηση στη γραμμή 2. Ο συνολικός

αριθμός των ΣΥΒ του αλγόριθμου είναι ίσος με

$$\begin{aligned}
& \sum_{i=1}^{n-1} B(i) + 3(n-1) + 2 + 1 = \\
& \sum_{i=1}^{n-1} \left(\sum_{j=1}^{n-i} (5j+2) + 3(n-i)+2 \right) + 3n = \\
& \sum_{i=1}^{n-1} \left(5 \sum_{j=1}^{n-i} j + 2(n-i) + 3(n-i)+2 \right) + 3n = \\
& \sum_{i=1}^{n-1} \left(5 \frac{(n-i+1)(n-i)}{2} + 5(n-i)+2 \right) + 3n = \\
& \sum_{i=1}^{n-1} \left(\frac{5(n-i)^2}{2} + \frac{5(n-i)}{2} + 5(n-i)+2 \right) + 3n = \\
& \frac{5}{2} \sum_{i=1}^{n-1} (n-i)^2 + \frac{15}{2} \sum_{i=1}^{n-1} (n-i) + 2(n-1) + 3n = \\
& \frac{5}{2} \sum_{i=1}^{n-1} (n^2 - 2ni + i^2) + \frac{15}{2} (n(n-1) - \sum_{i=1}^{n-1} i) + 5n - 2 = \\
& \frac{5}{2} (n^2(n-1) - 2n \frac{n(n-1)}{2} + \frac{(n-1)n(2n-1)}{6}) + \frac{15n(n-1)}{4} + 5n - 2 = \\
& \frac{5(n-1)n(2n-1)}{12} + \frac{15n(n-1)}{4} + 5n - 2 = \\
& 5n(n-1) \left(\frac{2n-1}{12} + \frac{3}{4} \right) + 5n - 2 = \frac{5n(n-1)(n+4)}{6} + 5n - 2
\end{aligned}$$

■

02_ΚΛΑΣΕΙΣ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

Κλάσεις O, Ω, Θ

Ορισμός 2. Έστω συνάρτηση $g : \mathbb{N} \rightarrow \mathbb{R}_+$. Οι (ασυμπτωτικές) τάξεις συναρτήσεων ορίζονται ως

$$O(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+, \exists a, n_0 \in \mathbb{R}_+, f(n) \leq a \cdot g(n), \forall n \geq n_0\}, \quad (2.1)$$

$$\Omega(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+, \exists b, n_0 \in \mathbb{R}_+, f(n) \geq b \cdot g(n), \forall n \geq n_0\}, \quad (2.2)$$

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)). \quad (2.3)$$

Η (2.1) ορίζει την κλάση $O(g(n))$ ως το σύνολο που περιέχει όλες τις συναρτήσεις $f(n)$ οι οποίες φράζονται από τα πάνω από ένα θετικό πολλαπλάσιο της $g(n)$,

με δεδομένο ότι η τιμή του n είναι επαρκώς μεγάλη (τουλάχιστον ίση με μία κριτική τιμή n_0). Αν $f(n)$ είναι μία από τις συναρτήσεις του συνόλου $O(g(n))$ θα γράφουμε $f(n) = O(g(n))$ αντί του ορθότερου $f(n) \in O(g(n))$ και θα διαβάζουμε «η $f(n)$ είναι τάξης (μεγέθους) $O(g(n))$ » ή «η $f(n)$ φράζεται ασυμπτωματικά εκ' των άνω από την $g(n)$ ».³ Ο όρος «ασυμπτωματικά» υποδηλώνει ότι το φράγμα ισχύει όταν η τιμή της παραμέτρου n είναι επαρκώς μεγάλη.

Αντίστοιχα, η (2.2) ορίζει την κλάση $\Omega(g(n))$ που περιέχει τις συναρτήσεις οι οποίες φράζονται ασυμπτωματικά εκ' των κάτω από την $g(n)$. Ενδιαφέρον παρουσιάζει η κλάση που ορίζεται από την (2.3): περιέχει τις συναρτήσεις οι οποίες φράζονται ασυμπτωματικά και εκ' των άνω και εκ' των κάτω από την $g(n)$. Δηλαδή ένας ισοδύναμος ορισμός της κλάσης που ορίζει η (2.3) είναι

$$\Theta(g(n)) = \{f : \mathbb{N} \rightarrow \mathbb{R}_+, \exists a, b, n_0 \in \mathbb{R}_+, a \cdot g(n) \geq f(n) \geq b \cdot g(n), \forall n \geq n_0\}.$$

Άσκηση 1. Έστω $f(n) = 3n^2 - 5n + 4$. Χωρίς τη χρήση ορίων να δείξετε ότι $f(n) = \Theta(n^2)$.

Λύση. Θα δείξουμε ότι $f(n) = O(n^2)$ και $f(n) = \Omega(n^2)$. Για το πρώτο θα πρέπει να υπολογίσουμε $\alpha, n_0 > 0$ τέτοιο ώστε

$$f(n) \leq \alpha \cdot n^2 \Rightarrow 3n^2 - 5n + 4 \leq \alpha \cdot n^2 \quad (1)$$

για κάθε $n \geq n_0$.

Παρατηρούμε ότι $3n^2 - 5n + 4 \leq 3n^2 + 4$, για κάθε $n \geq 0$. Επιπλέον για $n \geq 1$, έχουμε

$$3n^2 + 4 \leq 3n^2 + 4n^2 \leq 7n^2.$$

Άρα για $\alpha = 7, n_0 = 1$ ισχύει η (1).

Αντίστοιχα, θα πρέπει να υπάρχει $\beta, n_0 > 0$ τέτοιο ώστε

$$f(n) \geq \beta \cdot n^2 \Rightarrow 3n^2 - 5n + 4 \geq \beta \cdot n^2, \quad (2)$$

για κάθε $n \geq n_0$. Επομένως θέλουμε να προσδιορίσουμε τιμή για το β τέτοια ώστε το πολυώνυμο

$$(3 - \beta)n^2 - 5n + 4$$

να παίρνει θετικές τιμές. Θα πρέπει

$$3 - \beta > 0 \Rightarrow 3 > \beta$$

και επιπλέον η διακρίνουσα του να είναι μικρότερη του μηδενός, δηλαδή

$$(-5)^2 - 4 \cdot (3 - \beta) \cdot 4 < 0 \Rightarrow \beta < \frac{23}{16}.$$

Επομένως για οποιαδήποτε τιμή $\beta < \min\{3, \frac{23}{16}\} = \frac{23}{16}$ ισχύει η (1) ανεξαρτήτως της τιμής του n - για παράδειγμα μπορούμε να θέσουμε $\beta = 1$.

Συνολικά για $\alpha = 7, \beta = 1, n_0 = 1$ ισχύουν η (1) και η (2) και άρα και το ζητούμενο. ■

Παράδειγμα

Ιεραρχία Συναρτήσεων

$f(n)$	Περιγραφή τύπου
1	Σταθερή
$\lg n$	Λογαριθμική
$\lg^b n, b > 1$	Πολυλογαριθμική
$n^\epsilon, 1 > \epsilon > 0$	Υπογραμμική
n	Γραμμική
$n \lg n$	Υπεργραμμική
n^k	Πολυωνυμική
$n^k \lg^b n, k, b > 1$	Υπερπολυωνυμική
c^n με $c > 1$	Εκθετική
$n!$	Παραγοντική
n^n	Υπερεκθετική

ΣΤΑΘΕΡΕΣ < ΛΟΓΑΡΙΘΜΙΚΕΣ < ΡΟΔΥΩΝΥΜΙΚΕΣ < ΕΚΘΕΤΙΚΕΣ < ΥΠΕΡΕΚΘΕΤΙΚΕΣ
 $T(n) = \Theta(1) < T(n) = \Theta(\log n) < T(n) = \Theta(n^k) < T(n) = \Theta(a^n) < T(n) = \Theta(n!)$
 $T(n) = \Theta(n^n)$

Για Λογαριθμικές = $\log \log n < \log n < \log^k n$

Για Ροδυωνυμικές = $n < n^2 < n^3 < \dots < n^k$

Για Εκθετικές = $a^n < 2^n < 3^n < \dots < b^n$

Για υπερεκθετικές = $n! < n^n$

Άσκηση 8. Καθώς $n \rightarrow \infty$, να διατάξετε σε αύξουσα σειρά τις συναρτήσεις

1. $f_1 = 2^{\lg \sqrt{n^n}}$
2. $f_2 = 3^{n^2}$
3. $f_3 = n^2 \cdot 2^{\lg \lg n^3}$
4. $f_4 = n^{\sqrt{n}}$
5. $f_5 = 2^{\sqrt{2 \lg n}}$

Λύση.

$$f_5 = 2^{\sqrt{2 \lg n}} = 2^{\sqrt{\lg n^2}} = 2^{(\lg n^2)^{1/2}}.$$

και

$$f_3 = n^2 \cdot 2^{\lg \lg n^3} = n^2 \cdot \lg n^3 > n^2 = 2^{\lg n^2}.$$

Επομένως

$$f_3 > f_5. \quad (10)$$

Επίσης,

$$f_1 = 2^{\lg \sqrt{n^n}} = \sqrt{n^n} = n^{\frac{n}{2}} \quad (11)$$

και επειδή $\frac{n}{2} > \sqrt{n}$, για κάθε $n > 4$,

$$f_1 = n^{\frac{n}{2}} > n^{\sqrt{n}} = f_4. \quad (12)$$

Επειδή, για $n > 10$, $\sqrt{n} > 3$,

$$f_4 = n^{\sqrt{n}} > n^3,$$

και

$$n^3 \geq n^2 \cdot \lg n^3 = f_3,$$

συνεπάγεται ότι

$$f_4 > f_3. \quad (13)$$

Τέλος, από την (11) έχουμε

$$f_1 = n^{\frac{n}{2}} = 3^{\lg_3 n^{\frac{n}{2}}} = 3^{\frac{n}{2} \lg_3 n}.$$

Γνωρίζοντας ότι $n > n/2$ και $n > \lg_3 n$, για $n \geq 3$, έχουμε ότι $n^2 > \frac{n}{2} \lg_3 n$.
Επομένως,

$$f_2 = 3^{n^2} > 3^{\frac{n}{2} \lg_3 n} = f_1. \quad (14)$$

Συνολικά από (10), (12), (13), (14) έχουμε

$$f_2 > f_1 > f_4 > f_3 > f_5.$$

Παράδειγμα

Ιδιότητα μεταξύ O και Ω

2.1: «Κλάσεις O , Ω , Θ »

2.2: «Κλάσεις O , Ω , Θ »

Λήμμα 1. $f(n) = O(g(n))$ αν και μόνο αν $g(n) = \Omega(f(n))$.

Λήμμα

Απόδειξη. Αν $f(n) = O(g(n))$ τότε σύμφωνα με την (2.1) υπάρχουν θετικές σταθερές a, n_0 για τις οποίες ισχύει

$$f(n) \leq a \cdot g(n), \forall n \geq n_0.$$

Από την παραπάνω πρόταση έχουμε ότι

$$\frac{1}{a} \cdot f(n) \leq g(n), \forall n \geq n_0$$

και επομένως $g(n) = \Omega(f(n))$ αφού θεωρούμε την (2.2) όπου οι συναρτήσεις $f(n), g(n)$ είναι σε εναλλασσόμενους ρόλους και $b = \frac{1}{a}$.

Η απόδειξη της αντίστροφης περίπτωσης γίνεται με αντίστοιχο τρόπο. \square

Απόδειξη

Είναι προφανές ότι για τις κλάσεις O και Ω ισχύει η αντισυμμετρική ιδιότητα: αν $f(n) = O(g(n))$ ($f(n) = \Omega(g(n))$) τότε $g(n) \neq O(f(n))$ ($g(n) \neq \Omega(f(n))$). Στην

Ισοδυναμία της κλάσης Θ

Ορισμός 2: «Κλάσεις O , Ω , Θ »

Πρότυπη 1. Η κλάση Θ ορίζει στο σύνολο των συναρτήσεων μία σχέση ισοδυναμίας.

Λήμμα

Απόδειξη. Αρκεί να δείξουμε ότι ισχύει η ανακλαστική, η συμμετρική και η μεταβατική ιδιότητα.

Ανακλαστική: Θα πρέπει να δείξουμε ότι $g(n) = \Theta(g(n))$. Αυτό ισχύει τετριψμένα από τον Ορισμό 2: θεωρούμε $a = b = 1$ και g στη θέση της f .

Συμμετρική: Θα πρέπει να δείξουμε ότι αν $f(n) = \Theta(g(n))$ τότε $g(n) = \Theta(f(n))$.
Από την υπόθεση ισχύει ότι

$$b \cdot g(n) \leq f(n) \leq a \cdot g(n), \quad a, b, n_0 > 0.$$

Επομένως,

$$\frac{1}{a} \cdot f(n) \leq g(n) \leq \frac{1}{b} \cdot f(n)$$

και άρα ισχύει το ζητούμενο.

Απόδειξη

Μεταβατική: Έστω συναρτήσεις f, g, h τέτοιες ώστε $f(n) = \Theta(g(n))$ και $g(n) = \Theta(h(n))$. Άρα υπάρχουν σταθερές a_1, a_2 τέτοιες ώστε, για κάποιο $n_0 > 0$, ισχύει ότι

$$\begin{aligned} f(n) &\leq a_1 \cdot g(n), \quad \forall n \geq n_0, \\ g(n) &\leq a_2 \cdot h(n), \quad \forall n \geq n_0. \end{aligned}$$

Συνολικά, έχουμε

$$f(n) \leq a_1 \cdot h(n), \quad \forall n \geq n_0,$$

όπου $a = a_1 \cdot a_2$ και άρα $a > 0$. Επομένως $f(n) = O(h(n))$. Με όμοιο τρόπο δείχνουμε ότι $f(n) = \Omega(h(n))$. Δηλαδή ισχύει το ζητούμενο:

$$f(n) = \Theta(g(n)) \text{ και } g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n)).$$

Ιδιότητες της Θ

Ερώτημα 1.1: «Ισοδυναμία της κλάσης Θ (Μεταβατική Ιδιότητα)»

Πόρισμα 1. Αν $f(n) = \Theta(g(n))$ τότε $\Theta(f(n)) = \Theta(g(n))$ και αντίστροφα.

Λήμμα

2. (\Rightarrow) Εφόσον $f(n) = \Theta(g(n))$ ισχύει ότι υπάρχουν σταθερές $a_1, a_2, n_1 > 0$, τέτοιες ώστε

$$a_1 \cdot g(n) \leq f(n) \leq a_2 \cdot g(n), \quad \forall n \geq n_1. \quad (1)$$

Εστω $v(n)$ μία συνάρτηση κλάσης $\Theta(f(n))$. Τότε υπάρχουν σταθερές $k_1, k_2, n_2 > 0$, τέτοιες ώστε

$$k_1 \cdot f(n) \leq v(n) \leq k_2 \cdot f(n), \quad \forall n \geq n_2.$$

Αντικαθιστώντας στην παραπάνω ανισότητα το κάτω και το πάνω φράγμα της $f(n)$ από την (1) έχουμε

$$\begin{aligned} k_1 \cdot a_1 \cdot g(n) &\leq v(n) \leq k_2 \cdot a_2 \cdot g(n), \quad \forall n \geq \max\{n_1, n_2\}, \\ \Rightarrow v(n) &= \Theta(g(n)) \\ \Rightarrow \Theta(f(n)) &\subseteq \Theta(g(n)). \end{aligned} \quad (2)$$

Παρατηρήστε ότι (1) συνεπάγεται

$$\lambda_1 \cdot f(n) \leq g(n) \leq \lambda_2 \cdot f(n), \quad \forall n \geq n_1, \quad (3)$$

όπου $\lambda_1 = \frac{1}{a_2} > 0$, $\lambda_2 = \frac{1}{a_1} > 0$. Θεωρούμε μία οποιαδήποτε συνάρτηση $u(n) = \Theta(g(n))$. Μέσω της (3) μπορούμε να δείξουμε, με αντίστοιχο τρόπο όπως στην απόδειξη της (2), ότι

$$u(n) = \Theta(f(n)) \Rightarrow \Theta(g(n)) \subseteq \Theta(f(n)). \quad (4)$$

Από (2), (4) έχουμε

$$\Theta(g(n)) = \Theta(f(n)).$$

(\Leftarrow) Στο Ερώτημα 1.1 δείξαμε ότι $f(n) = \Theta(f(n))$. Άρα από την υπόθεση που μας λέει ότι οι κλάσεις $\Theta(f(n))$ και $\Theta(g(n))$ ταυτίζονται ($\Theta(f(n)) = \Theta(g(n))$) έχουμε ότι $f(n) = \Theta(g(n))$.

Απόδειξη

Πόρισμα 2. Ισχύει αποκλειστικά ένα από τα παρακάτω.

1. $\Theta(f(n)) = \Theta(g(n))$.
2. $\Theta(f(n)) \cap \Theta(g(n)) = \emptyset$.

Η παρακάτω πρόταση προσφέρει ένα κριτήριο με βάση το οποίο μπορούμε να καθορίσουμε ποια από τις δύο περιπτώσεις του Πορίσματος 2 ισχύει.⁵

Λήμμα 2. Έστω συναρτήσεις f, g και $\lim \frac{f(n)}{g(n)} = c$. Αν $0 < c < \infty$ τότε ισχύει η Περίπτωση 1 του Πορίσματος 2. Αν $c = 0$ ή $c = \infty$ τότε ισχύει η Περίπτωση 2.

Απόδειξη. Από τον ορισμό του ορίου (Ορισμός 7, Ενότητα 13.10) έχουμε ότι για κάθε $\epsilon > 0$ υπάρχει n_0 τέτοιο ώστε για κάθε $n \geq n_0$, ισχύει

$$\begin{aligned} \left| \frac{f(n)}{g(n)} - c \right| &< \epsilon \Rightarrow \\ c - \epsilon &< \frac{f(n)}{g(n)} < c + \epsilon. \end{aligned} \tag{2.8}$$

Αν $c = 0$ τότε από αριστερό μέλος της (2.8) θα πρέπει να ισχύει για κάθε $\epsilon > 0$ ότι

$$\frac{f(n)}{g(n)} > -\epsilon \Rightarrow f(n) > -\epsilon \cdot g(n).$$

Άρα δεν υπάρχει θετική σταθερά που πολλαπλασιαζόμενη με την $g(n)$ να φράζει την $f(n)$ από τα κάτω δηλαδή $f(n) \neq \Omega(g(n))$.

Αν $c = \infty$ τότε από το δεξί μέλος της (2.8) προκύπτει με αντίστοιχο τρόπο ότι $f(n) \neq O(g(n))$.

Στην περίπτωση όπου $0 < c < \infty$ μπορούμε να επιλέξουμε ϵ τέτοιο ώστε $c - \epsilon > 0$ και θέτοντας

$$a = \max\{c + \epsilon, \max\{\frac{f(n)}{g(n)}, 1 \leq n \leq n_0\}\}, \tag{2.9}$$

$$b = \min\{c - \epsilon, \min\{\frac{f(n)}{g(n)}, 1 \leq n \leq n_0\}\}, \tag{2.10}$$

έχουμε $a, b > 0$. Από την (2.9), (2.10) και (2.8)

$$b \cdot g(n) \leq f(n) \leq a \cdot g(n)$$

η οποία ολοκληρώνει το ζητούμενο. □

Λήμμα

Απόδειξη

Ιδιότητα του ασυμπτωτικά θετικού πολυωνύμου

Ένα πολυώνυμο $p(n)$ ονομάζεται ασυμπτωματικά θετικό αν παίρνει θετικές τιμές για μεγάλες τιμές του n .

Πόρισμα 3. Έστω $n \in \mathbb{N}$ και $p(n)$ ασυμπτωματικά θετικό πολυώνυμο βαθμού d . Τότε, $p(n) = \Theta(n^d)$.

Απόδειξη. Το $p(n)$ γράφεται ως

$$p(n) = a_d n^d + \sum_{k \in K} a_k n^k,$$

όπου $K = \{0, \dots, d-1\}$. Συνεπώς,

$$\lim \frac{p(n)}{n^d} = \lim \frac{a_d n^d}{n^d} + \sum_{k \in K} \lim \frac{a_k n^k}{n^d} = a_d.$$

Εξ' ορισμού $\infty > a_d > 0$, και επομένως σύμφωνα με το Λήμμα 2 έχουμε ότι $\Theta(p(n)) = \Theta(n^d)$. Οπότε άμεσα από το Πόρισμα 1, προκύπτει ότι $p(n) = \Theta(n^d)$. \square

Λήμμα

Απόδειξη

Κλάσεις O , ω

Λήμμα 1: «Ιδιότητα μεταξύ O και Ω »

$$o(g(n)) = \{f : \forall a \in \mathbb{R}_+, \exists n_0 > 0, a \cdot g(n) > f(n) \geq 0, \forall n \geq n_0\}, \quad (2.11)$$

$$\omega(g(n)) = \{f : \forall b \in \mathbb{R}_+, \exists n_0 > 0, f(n) > b \cdot g(n) \geq 0, \forall n \geq n_0\}. \quad (2.12)$$

Παρατηρούμε ότι η συνθήκη που συνδέει τις $f(n), g(n)$ στην (2.11) γράφεται ισοδύναμα ως

$$\forall a \in \mathbb{R}_+, \exists n_0 > 0, a > \frac{f(n)}{g(n)}, \forall n \geq n_0.$$

Όμως αυτός είναι ο Ορισμός 7 (Ενότητα 13.10) όταν το όριο του λόγου $\frac{f(n)}{g(n)}$ είναι ίσο με μηδέν του n τείνοντος στο άπειρο.⁶ Αντίστοιχα η συνθήκη που συνδέει τις $f(n), g(n)$ στην (2.11) ισοδυναμεί με τον Ορισμό 9 (Ενότητα 13.10) όταν το όριο του λόγου $\frac{f(n)}{g(n)}$ είναι ίσο με άπειρο για n τείνοντος στο άπειρο. Επομένως οι (2.11), (2.12) είναι ισοδύναμες με την

$$f(n) = \begin{cases} o(g(n)) & \text{αν και μόνο αν } \lim \frac{f(n)}{g(n)} = 0, \\ \omega(g(n)) & \text{αν και μόνο αν } \lim \frac{f(n)}{g(n)} = \infty. \end{cases} \quad (2.13)$$

Είναι εύκολο να διαπιστώσει κάποιος ότι το Λήμμα 1 ισχύει με την κλάση o στη θέση της O και την ω στη θέση της Ω .

Ασυμπτωτική κατάταξη συναρτήσεων με χρήση ορίων

Η χρήση ορίων μπορεί να υιοθετηθεί για την ασυμπτωτική κατάταξη συναρτήσεων, δηλαδή για την αύξουσα διάταξη συναρτήσεων για μεγάλες τιμές της ανεξάρτητης μεταβλητής n . Συγκεκριμένα, μπορούμε να συγκρίνουμε ασυμπτωματικά συναρτήσεις $f(n), g(n)$ με το ορίου του λόγου της f με τη g ως εξής

$$\lim \frac{f(n)}{g(n)} \begin{cases} = 1, & \text{tότε } f(n) = g(n), \\ > 1, & \text{tότε } f(n) > g(n), \\ < 1, & \text{tότε } f(n) < g(n). \end{cases} \quad (2.15)$$

Υπολογισμός ΣΥΒ και Πολυπλοκότητας Αλγορίθμου

Πόρισμα 3: «Ιδιότητα των ασυμπτωτικά θετικού πολυωνύμου»

Αλγόριθμος 9 Υπολογισμός αθροίσματος $\sum_{j=0}^{n-1} (a + j \cdot t)$

Απαιτείται: θετικοί ακέραιοι n, a, t ;

Επιστρέφεται: άθροισμα n όρων αριθμητικής προόδου με πρώτο όρο το a και βήμα t

```
1: function Sum_of_Terms02(int n, int a, int t)
2:   sum  $\leftarrow$  0;
3:   for j  $\leftarrow$  0; j  $<$  n; j  $\leftarrow\leftarrow$  do
4:     oros  $\leftarrow$  0;
5:     multiple_t  $\leftarrow$  0;
6:     for i  $\leftarrow$  1; i  $\leq$  j; i  $\leftarrow\leftarrow$  do
7:       multiple_t  $\leftarrow$  multiple_t + t;
8:     end for
9:     oros  $\leftarrow$  a + multiple_t;
10:    sum  $\leftarrow$  sum + oros;
11:   end for
12:   return sum;
13: end function
```

των ΣΥΒ που εκτελείται στις Γραμμές 6-8 είναι ίσος με

$$A(j) = 2j + 3j + 2 = 5j + 2, \quad (2.20)$$

όπου τα δύο επιπλέον ΣΥΒ προκύπτουν από την αρχικοποίηση του δείκτη i και τη σύγκριση που θα οδηγήσει σε τερματισμό του εσωτερικού βρόχου.

Με αντίστοιχο τρόπο υπολογίζουμε τον αριθμό των ΣΥΒ του εξωτερικού βρόχου: σε κάθε επανάληψη έχουμε τα έξι ΣΥΒ που εκτελούνται στις Γραμμές 4-5, 9-10, τα ΣΥΒ που δίνονται από την (2.20) και τρία ΣΥΒ που αφορούν τον δείκτη j στη Γραμμή 3. Ο αριθμός των επαναλήψεων του εξωτερικού βρόχου είναι n και εκτελούνται επιπλέον τρία ΣΥΒ: μία εκχώρηση στη Γραμμή 2, η αρχικοποίηση του δείκτη j και η σύγκριση που οδηγεί στον τερματισμό του εξωτερικού βρόχου. Επομένως ο συνολικός αριθμός των ΣΥΒ είναι

$$\begin{aligned} T(n) &= \sum_{j=0}^{n-1} A(j) + 6n + 3n + 3 = \sum_{j=0}^{n-1} (5j + 2) + 9n + 3 \\ &\Rightarrow T(n) = \frac{1}{2}(5n^2 + 17n + 6). \end{aligned} \quad (2.21)$$

Από το Πόρισμα 3 προκύπτει άμεσα ότι ο Αλγόριθμος 9 είναι πολυπλοκότητας $\Theta(n^2)$.

Βρόχοι και Πολυπλοκότητα

Άλγοριθμος 9: «Υπολογισμός SYB και Πολυπλοκότητας Αλγορίθμου»

Το παραπάνω παράδειγμα είναι αποκαλυπτικό: η ασυμπτωτική συμπεριφορά ενός αλγόριθμου δεν επηρεάζεται από τον αριθμό των SYB εφόσον ο αριθμός αυτός είναι σταθερός, δηλαδή, δεν εξαρτάται από την παράμετρο n που αντικατοπτρίζει το μέγεθος του στιγμιότυπου. Εξαρτάται από τον αριθμό των SYB όταν αυτός αποτελεί συνάρτηση του n . Στον Άλγοριθμο 9 σε κάθε επανάληψη του εσωτερικού βρόχου ο αριθμός των SYB είναι σταθερός. Όμως επειδή ο αριθμός των επαναλήψεων είναι συνάρτηση της τιμής του δείκτη j , ο συνολικός αριθμός των SYB του εσωτερικού βρόχου, που δίνεται από το γινόμενο των επαναλήψεων επί το σταθερό αριθμό των πράξεων που εκτελείται σε κάθε επανάληψη, αποτελεί επίσης συνάρτηση του j . Ως προς των εξωτερικό

βρόχο παρατηρούμε ότι κάθε επανάληψη αντιστοιχεί σε μία τιμή του δείκτη j και για αυτή την τιμή εκτελείται σταθερός αριθμός SYB εκτός από τα SYB που αφορούν τον εσωτερικό βρόχο τα οποία αυξάνουν ευθέως ανάλογα με την τιμή του δείκτη j . Έτσι διαμορφώνεται η τάξη μεγέθους των SYB του αλγόριθμου: ο δείκτης j παίρνει n διαφορετικές τιμές και για κάθε τέτοια τιμή εκτελούνται $\Theta(j)$ SYB από τον εσωτερικό βρόχο. Άρα ο αριθμός των SYB που εκτελούνται από τον εσωτερικό βρόχο συνολικά (κατά την εκτέλεση του αλγόριθμου) δίνεται από το άθροισμα των SYB αυτών για κάθε τιμή $j \in \{0, \dots, n - 1\}$ · με μαθηματικούς όρους, έχουμε

$$\sum_{j=0}^{n-1} \Theta(j) = \sum_{j=0}^{n-1} c_1 \cdot j = c_1 \sum_{j=0}^{n-1} j = c_1 \frac{n(n-1)}{2} = \Theta(n^2). \quad (2.23)$$

Η παραπάνω ανάλυση οδηγεί στο συμπέρασμα ότι ο μεγαλύτερος αριθμός των βρόχων που είναι διαδοχικά φωλιασμένοι καθορίζει την ασυμπτωτική συμπεριφορά ενός αλγόριθμου. Θα πρέπει να είμαστε ιδιαίτερα προσεκτικοί με το συμπέρασμα αυτό αφού δεν ισχύει ως γενικός κανόνας. Για να ισχύει θα πρέπει ο αριθμός των επαναλήψεων του κάθε βρόχου να είναι συνάρτηση - έστω και έμμεσα - της παραμέτρου n που αντανακλά το μέγεθος του στιγμιότυπου και επιπλέον ο αριθμός των SYB σε κάθε επανάληψη - εκτός των SYB που εκτελούνται από τους εμπειρεχόμενους βρόχους - να είναι σταθερός, δηλαδή ανεξάρτητος της παραμέτρου n . Επιπλέον θα πρέπει να διασαφηνίσουμε ότι οι παραπάνω παρατηρήσεις δεν αφορούν αναδρομικούς αλγόριθμους.

Λειτουργίες σε Πίνακα

Στην Ενότητα 1.2 έγινε αναφορά στις κυριότητες λειτουργίες που αφορούν δομές δεδομένων. Σε σχέση με έναν πίνακα οι περισσότερες από αυτές υλοποιούνται αρκετά εύκολα αλγορίθμικά. Για το λόγο αυτό θα μας απασχολήσουν μόνο αλγορίθμικά σχήματα που αφορούν λειτουργίες ταξινόμησης, συγχώνευσης και αναζήτησης. Χωρίς βλάβη της γενικότητας, οι λειτουργίες αυτές θα περιγραφούν σε δομή πίνακα ακεραίων. Επίσης για λόγους παραμετροποίησης θα υποθέσουμε ότι οι ακέραιοι αυτοί θα βρίσκονται αποθηκευμένοι από τη θέση lo έως τη θέση hi του πίνακα. Επομένως το πλήθος των ακεραίων (παράμετρος n) θα δίνεται από τη σχέση

$$n = hi - lo + 1. \quad (3.1)$$

Ταξινόμηση

3.1: «Λειτουργίες σε Πίνακα

Το πρόβλημα της ταξινόμησης στοιχείων πίνακα (σε αύξουσα σειρά) συνίσταται στην ανατοποθέτηση των στοιχείων του A κατά τρόπο ώστε

$$A[lo] \leq A[lo + 1] \leq \cdots \leq A[hi - 1] \leq A[hi].$$

Προφανώς το πλήθος των προς ταξινόμηση ακεραίων n - όπως δίνεται από την (3.1) - εκφράζει το μέγεθος του κάθε στιγμιότυπου του συγκεκριμένου προβλήματος.

Θεωρώντας ότι $lo = 1$, $hi = n$, το πρόβλημα μπορεί να διατυπωθεί με ισοδύναμο τρόπο ως εξής: δεδομένης μίας σειράς στοιχείων a_1, \dots, a_n , που βρίσκεται αποθηκευμένη στις n πρώτες θέσεις ενός πίνακα A , να βρεθεί μία μετάθεση των δεικτών $1, \dots, n$, έστω $p(1), \dots, p(n)$ τέτοια ώστε $a_{p(1)} \leq \cdots \leq a_{p(n)}$.

Παράδειγμα 8. Θεωρούμε $lo = 1$, $hi = 5$ και έστω $A[1] = 2, A[2] = 7, A[3] = 5, A[4] = 7, A[5] = 1$. Θέλουμε να αναδιαταχτούν τα στοιχεία ώστε ο πίνακας να έχει την εικόνα

$$A = [1, 2, 5, 7, 7].$$

Η ζητούμενη μετάθεση είναι $p(1) = 5, p(2) = 1, p(3) = 3, p(4) = 2, p(5) = 4$. Παρατηρούμε ότι αν υπάρχουν στοιχεία που είναι μεταξύ τους ίσα, η μετάθεση δεν είναι μοναδική. Επίσης, παρατηρούμε ότι στην παραπάνω σειρά το στοιχείο 5 δεν άλλαξε θέση καταλαμβάνει την ίδια θέση - θέση 3 - στον πίνακα A και όταν αυτός είναι ταξινομημένος.

Ιδιότητα της σωστής θέσης ενός στοιχείου στον Πίνακα

Η ιδιότητα υλοποιείται στον αλγόριθμο «ΑΛΓΟΡΙΘΜΟΙ/03_ΠΙΝΑΚΕΣ/Pivot_Partition»

Ιδιότητα 1 (Σωστή Θέση). Ένα στοιχείο έχει την ιδιότητα της σωστής θέσης (ισοδύναμα, βρίσκεται στη σωστή θέση) όταν η θέση του στην ακολουθία είναι ίδια με τη θέση που θα έχει στην ακολουθία όταν αυτή ταξινομηθεί.

Συνεπώς, αν ένα στοιχείο σε μία ακολουθία πληροί την παραπάνω ιδιότητα τότε όλα τα στοιχεία που βρίσκονται αριστερότερα από αυτό είναι μικρότερα-ίσα από αυτό και όλα τα στοιχεία που βρίσκονται δεξιότερα είναι μεγαλύτερα-ίσα (από αυτό).

Η λειτουργία της αναζήτησης συνίσταται στην διαπίστωση της ύπαρξης(ή όχι) ενός ακεραίου, ονομαστικά a , σε ένα σύνολο ακεραίων αποθηκευμένων σε ένα πίνακα A . Για την επίλυση του προβλήματος αυτού, θα παρουσιάσουμε διαφορετικούς αλγόριθμους, με τη μορφή συναρτήσεων, με το γενικευμένο όνομα Search. Κάθε τέτοια συνάρτηση θα δέχεται σαν είσοδο τον ακέραιο a , τον πίνακα A , και τις θέσεις lo και hi ($lo \leq hi$) του πίνακα ανάμεσα στις οποίες αναζητείται ο a . Δηλαδή, θα αναζητείται η θέση i για την οποία να ισχύει ότι $a = A[i]$ με $lo \leq i \leq hi$. Αν υπάρχει τέτοια θέση, η συνάρτηση θα την επιστρέψει. Αν δεν υπάρχει τέτοια θέση, η συνάρτηση θα επιστρέψει

1. την θέση (τιμή) $hi + 1$ αν $a > \max\{A[i] : lo \leq i \leq hi\}$,
2. την θέση στην οποία βρίσκεται το μικρότερο στοιχείο του πίνακα A που είναι μεγαλύτερο από το a , αν ο A είναι ταξινομημένος.

Διαμερισμός διατεταγμένων ακεραίων ενός Πίνακα

3.1: «Λειτουργίες σε Πίνακα

3.3: «ΑΛΓΟΡΙΘΜΟΙ/03_ΠΙΝΑΚΕΣ/Search_in_Group»

Λήμμα 5. Το σύνολο των n διατεταγμένων ακεραίων μπορεί να διαμεριστεί σε ένα σύνολο από r στοιχεία⁴, d σύνολα από q στοιχεία το καθένα και ένα σύνολο από ένα στοιχείο.

Απόδειξη. Από την (3.3) έχουμε,

$$d \cdot q + r + 1 = \lfloor \frac{hi - lo}{q} \rfloor \cdot q + (hi - lo) \bmod q + 1 = hi - lo + 1 = n,$$

όπου η τελευταία ισότητα προκύπτει από την (3.1). □

Λήμμα

Απόδειξη

Στοίβα

Η στοίβα είναι ένας περιέκτης (δοχείο) αντικειμένων τα οποία εισάγονται σε αυτό και εξάγονται από αυτό σύμφωνα με τον κανόνα LIFO (last-in, first-out): το τελευταίο αντικείμενο που θα εισαχθεί είναι το πρώτο αντικείμενο που θα εξαχθεί. Μπορούμε να εισάγουμε αντικείμενο οποιαδήποτε στιγμή άλλα για να εξαχθεί ένα συγκεκριμένο αντικείμενο πρέπει να εξαχθούν όλα τα αντικείμενα που έχουν εισαχθεί μετά από αυτό στη στοίβα. Οι βασικές λειτουργίες σε μία στοίβα είναι η pop και η push. Η πρώτη εισάγει ένα αντικείμενο στη στοίβα ενώ η δεύτερη εξάγει το αντικείμενο που εισήλθε χρονικά πιο πρόσφατα σε αυτή.

Η ουρά είναι ένα αντικείμενο αντίστοιχο της στοίβας με τη διαφορά ότι η διαχείριση σε σχέση με την εισαγωγή και εξαγωγή στοιχείων γίνεται σύμφωνα με τον κανόνα FIFO (first-in, first-out). Δηλαδή ένα στοιχείο μπορεί να εισαχθεί οποιαδήποτε στιγμή αλλά το στοιχείο που εξέρχεται πρώτο είναι αυτό που έχει παραμείνει στην ουρά περισσότερο. Άρα κάθε νέο στοιχείο προστίθεται στο τέλος της ουράς ενώ το παλαιότερο στοιχείο βρίσκεται στην κορυφή της. Η λειτουργία εισαγωγής ονομάζεται enqueue ενώ dequeue η διαδικασία εξαγωγής.

04_ΑΝΑΔΡΟΜΗ

Αναδρομή

Ένα απλό πρόβλημα είναι ο υπολογισμός του παραγοντικού ενός φυσικού αριθμού n . Η ποσότητα αυτή συμβολίζεται με $n!$ και υπολογίζεται από τον τύπο

$$n! = \begin{cases} 1 \cdot 2 \cdot \dots \cdot (n-1) \cdot n, & n > 0, \\ 1, & n = 0. \end{cases} \quad (4.1)$$

Παρατηρούμε ότι η (4.1) είναι ισοδύναμη με

$$n! = \begin{cases} (n-1)! \cdot n, & n > 0, \\ 1, & n = 0. \end{cases} \quad (4.2)$$

Οι σχέσεις (4.1), (4.2) είναι ισοδύναμες: η ποσότητα $n!$ μπορεί να οριστεί είτε από την πρώτη είτε από τη δεύτερη. Με μία πρώτη ματιά ο ορισμός μέσω της (4.2) φαντάζει περίεργος: ορίζεται η έννοια «παραγοντικό» με βάση τον εαυτό της. Όμως εστιάζοντας καλύτερα παρατηρούμε ότι είναι η τιμή της συνάρτησης «παραγοντικό» που ορίζεται με βάση την τιμή της ίδιας συνάρτησης όπως υπολογίζεται για μικρότερη τιμή του ορίσματος της. Σε μία τέτοια περίπτωση λέμε ότι η συνάρτηση είναι αναδρομική.¹ Όμως προκειμένου να μπορεί να χρησιμοποιηθεί υπολογιστικά ο ορισμός μίας αναδρομικής συνάρτησης, θα πρέπει να υπάρχει μία τιμή του ορίσματος για την οποία η συνάρτηση

να μην ορίζεται αναδρομικά και η κλήση της συνάρτησης με τη συγκεκριμένη τιμή ορίσματος να προκύπτει (αναπόφευκτα) από τον αναδρομικό ορισμό. Ετσι διασφαλίζεται ότι το βάθος της αναδρομής είναι πεπερασμένο. Στην περίπτωση της (4.2) αυτό συμβαίνει αφού η $(n+1)$ -ιοστή κλήση της συνάρτησης «παραγοντικό» έχει όρισμα 0 και η τιμή της συνάρτησης για αυτό το όρισμα είναι 1 (δηλαδή, δεν υπολογίζεται πια αναδρομικά). Η συνθήκη που πρέπει να ικανοποιεί το όρισμα προκειμένου η τιμή της συνάρτησης να μην υπολογίζεται αναδρομικά ονομάζεται οριακή. Για παράδειγμα,

Σωρός

Μία από τις πιο δεδομένες μεθόδους ταξινόμησης είναι μέσω της χρήσης *σωρού* (*heap*). Η δομή αυτή είναι πολύ χρήσιμη αφού πέραν της ταξινόμησης μπορεί και να αξιοποιηθεί για την υλοποίηση ουράς προτεραιότητας. Ο σωρός είναι μία (*ιδεατή*) δυαδική δενδρική δομή η οποία όμως υλοποιείται με πίνακα. Έστω λοιπόν πίνακας A με στοιχεία στις n πρώτες θέσεις. Τα στοιχεία αυτά περιγράφουν ένα σωρό αν κάθε στοιχείο του πίνακα αντιστοιχεί στην κορυφή ενός δυαδικού δένδρου με τις ακόλουθες ιδιότητες:

- ρίζα του δένδρου αποτελεί το στοιχείο $A[1]$
- για $i = 1, \dots, \lfloor \frac{n}{2} \rfloor$, κάθε στοιχείο $A[i]$ έχει σαν άμεσο αριστερό απόγονο το στοιχείο $A[2i]$ και σαν άμεσο δεξί το $A[2i + 1]$,
- για $i = 2, \dots, n$, κάθε στοιχείο $A[i]$ έχει σαν άμεσο πρόγονο το στοιχείο $A[\lfloor \frac{i}{2} \rfloor]$.

Μέθοδος της Εικασίας

Στη μέθοδο αυτή κάνουμε μία ασυμπτωτική εκτίμηση για την αναδρομική συνάρτηση και στη συνέχεια προσπαθούμε να αποδείξουμε ότι η εκτίμηση αυτή ισχύει με τη μέθοδο της επαγωγής. Για παράδειγμα, δοθείσης της συνάρτησης

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

εικάζουμε ότι $T(n) = O(\lg n)$.

Με τη χρήση επαγωγής, θα πρέπει να δείξουμε ότι υπάρχουν $a > 0, n_0 \in \mathbb{N}$ τέτοια ώστε

$$T(n) \leq a \cdot \lg n, \forall n \geq n_0. \quad (5.3)$$

Ως βάση της επαγωγής χρησιμοποιούμε την τιμή $n = 2$. Στην περίπτωση αυτή έχουμε

$$T(2) = 1 \leq a \cdot \lg 2 \Rightarrow a \geq 1.$$

Η επαγωγική υπόθεση είναι ότι η (5.3) ισχύει για $T\left(\frac{n}{2}\right)$. Δηλαδή,

$$\begin{aligned} T\left(\frac{n}{2}\right) &\leq a \cdot \lg \frac{n}{2} \Rightarrow \\ T\left(\frac{n}{2}\right) + 1 &\leq a \cdot \lg \frac{n}{2} + 1 \Rightarrow \\ T(n) &\leq a \cdot \lg \frac{n}{2} + 1 = a \cdot (\lg n - \lg 2) + 1 = a \lg n - a + 1. \end{aligned}$$

Από την τελευταία σχέση προκύπτει ότι

$$T(n) \leq a \cdot \lg n$$

για $a \geq 1$

Επομένως δείξαμε ότι η (5.3) ισχύει για $a \geq 1$ και για κάθε $n \geq n_0 = 2$.

Στη συνέχεια εικάζουμε ότι $T(n) = \Omega(\lg n)$. Οπως και στην προηγούμενη περίπτωση θα δείξουμε ότι υπάρχουν $b > 0, n_0 \in \mathbb{N}$ τέτοια ώστε

$$T(n) \geq b \cdot \lg n, \forall n \geq n_0. \quad (5.4)$$

Και πάλι ως βάση της επαγωγής χρησιμοποιούμε την τιμή $n = 2$ από την οποία προκύπτει ότι $1 \geq b$. Η επαγωγική υπόθεση είναι ότι η (5.4) ισχύει για $T\left(\frac{n}{2}\right)$. Δηλαδή,

$$\begin{aligned} T\left(\frac{n}{2}\right) &\geq b \cdot \lg \frac{n}{2} \Rightarrow \\ T\left(\frac{n}{2}\right) + 1 &\geq b \cdot \lg \frac{n}{2} + 1 \Rightarrow \\ T(n) &\geq b \cdot \lg \frac{n}{2} + 1 > b \cdot \lg n \end{aligned}$$

Η παραπάνω ανισότητα ισχύει για $1 \geq b \geq 0$ και για κάθε $n \geq n_0 = 2$.

Συνολικά, δείξαμε ότι $T(n) = \Theta(\lg n)$.

Μέθοδος

Παράδειγμα

Μέθοδος της Εικασίας με ταβάνια και πατώματα

Οι αναδρομικές σχέσεις που εκφράζουν τον αριθμό των ΣΥΒ $T(n)$ έχουν συνήθως νόημα μόνο στην περίπτωση που σε κάθε κλήση της συνάρτησης η τιμή της παραμέτρου n είναι ακέραια. Συνήθως σε αυτές τις περιπτώσεις, το μέγεθος του στιγμιότυπου για το οποίο καλείται αναδρομικά ο αλγόριθμος (δηλαδή η τιμή της παραμέτρου με την οποία καλείται η αναδρομική συνάρτηση) είναι ίσο με το «πάτωμα» ή το «ταβάνι» ενός λόγου του μεγέθους του αρχικού στιγμιότυπου (δηλαδή της παραμέτρου n). Στην περίπτωση αυτή χρειάζεται προσοχή στη μορφή του φράγματος που χρησιμοποιούμε στην επαγωγή. Για παράδειγμα, έστω ότι θέλουμε να αποδείξουμε ότι η συνάρτηση

$$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + 1, & n > 1, \\ 1, & n = 1. \end{cases} \quad (5.5)$$

είναι τάξης $O(n)$. Ακολουθώντας τα βήματα του προηγούμενου παραδείγματος, αρκεί να δείξουμε ότι υπάρχει $a > 0$ και $n_0 \in \mathbb{N}$ τέτοια ώστε $T(n) \leq a \cdot n$, για κάθε $n \geq n_0$. Θα επιχειρήσουμε να το αποδείξουμε κάνοντας την επαγωγική υπόθεση ότι αυτό ισχύει για $T(\lfloor n/2 \rfloor)$. Δηλαδή έστω ότι ισχύει

$$T(\lfloor n/2 \rfloor) \leq a \cdot \lfloor n/2 \rfloor.$$

Αντικαθιστώντας στην (5.5) έχουμε

$$T(n) \leq 2 \cdot a \cdot \lfloor n/2 \rfloor + 1$$

Επειδή $\lfloor n/2 \rfloor \leq n/2$, η παραπάνω ανισότητα συνεπάγεται ότι

$$\begin{aligned} T(n) &\leq 2 \cdot a \cdot \frac{n}{2} + 1 \\ &= a \cdot n + 1. \end{aligned}$$

Όμως η τελευταία τιμή είναι μεγαλύτερη του $a \cdot n$ και όχι μικρότερη-ίση με αυτή (που είναι το ζητούμενο στο επαγωγικό βήμα).

Για να άρουμε το αδιέξοδο, αλλάζουμε λίγο το ζητούμενο: αντί να προσπαθούμε να δείξουμε ότι $T(n) \leq a \cdot n$, θα δείξουμε ότι

$$T(n) \leq a \cdot n - b, \quad (5.6)$$

για $b > 0$. Προσέξτε ότι η (5.6) συνεπάγεται το ζητούμενο, δηλαδή ότι $T(n) = O(n)$ αφού για $b > 0$,

$$T(n) \leq a \cdot n - b \leq a \cdot n.$$

Κάνουμε την επαγωγική υπόθεση ότι η (5.6) ισχύει για $T(\lfloor n/2 \rfloor)$. Δηλαδή έστω ότι ισχύει

$$T(\lfloor n/2 \rfloor) \leq a \cdot \lfloor n/2 \rfloor - b.$$

Αντικαθιστώντας στην (5.5) έχουμε

$$\begin{aligned} T(n) &\leq 2 \cdot (a \cdot \lfloor n/2 \rfloor - b) + 1 \\ &\leq 2 \cdot a \cdot \frac{n}{2} - 2b + 1 \\ &= a \cdot n - b - b + 1 \Rightarrow \\ T(n) &\leq a \cdot n - b, \end{aligned}$$

για $b \geq 1$.

Επίσης επειδή $T(n=1) = 1$, από την παραπάνω σχέση έχουμε ότι

$$T(1) = 1 \leq a \cdot 1 - b \Rightarrow a \geq b + 1.$$

Παράδειγμα

Διαδοχική Αντικατάσταση

Η μέθοδος που αναπτύχθηκε στο προηγούμενο κεφάλαιο έχει το μειονέκτημα ότι χρειάζεται εμπειρία προκειμένου να διατυπωθεί μία «σωστή» εικασία. Με τη μέθοδο που θα παρουσιάσουμε στη συνέχεια κάτι τέτοιο δεν είναι απαραίτητο.

Η μέθοδος της Διαδοχικής Αντικατάστασης συνίσταται στην εκτέλεση των παρακάτω βημάτων.

- Αντικαθιστούμε το αριστερό μέλος της αναδρομικής έκφρασης διαδοχικά μέχρι να μπορούμε να γράψουμε την έκφραση σε γενική μορφή (σε σχέση με μία παράμετρο έστω k) που δηλώνει τον αριθμό των αντικαταστάσεων.
- Υπολογίζουμε την τιμή του k για την οποία η έκφραση παύει να είναι αναδρομική.
- Αντικαθιστούμε την τιμή αυτή στη γενική έκφραση και υπολογίζουμε την τελική (κλειστή) της μορφή.

Θα χρησιμοποιήσουμε αυτή τη μέθοδο για τη μελέτη της συνάρτησης

$$T(n) = \begin{cases} T(n-1) + 2, & \text{για } n > 0, \\ 1, & \text{για } n = 0. \end{cases}$$

Εφαρμόζοντας διαδοχική αντικατάσταση έχουμε

$$\begin{aligned} T(n) &= T(n-1) + 2 \\ &= [T((n-1)-1) + 2] + 2 = T(n-2) + 2 \cdot 2 \\ &= [T((n-2)-1) + 2] + 2 \cdot 2 = T(n-3) + 3 \cdot 2 \\ &= \dots \\ &= T(n-k) + k \cdot 2. \end{aligned}$$

Θα πρέπει να υπολογίσουμε για ποια τιμή του k η σχέση παύει να είναι αναδρομική. Αυτό γίνεται όταν

$$n - k = 0 \Rightarrow k = n.$$

Αντικαθιστώντας στην παραπάνω σχέση, έχουμε την κλειστή μορφή

$$T(n) = (0) + 2n = 2n + 1 \Rightarrow T(n) = \Theta(n).$$

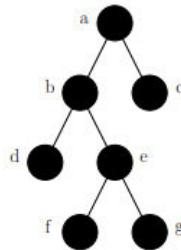
Από την ανάλυση που κάνομε είναι φανερό ότι ο υπολογισμός του ακριβούς αριθμού των στοιχειωδών πράξεων σε κάθε κλήση του αναδρομικού αλγόριθμου δεν είναι απαραίτητος για την κατάταξη του αλγορίθμου στην ιεραρχία πολυπλοκότητας. Ακόμα και αν θεωρούσαμε ότι σε κάθε κλήση εκτελούνται ένας **σταθερός** αριθμός στοιχειωδών πράξεων, έστω c (χωρίς να προσδιορίζαμε την ακριβή τιμή του c), θα προέκυπτε ότι $T(n) = c \cdot n + 1$ και άρα πάλι $T(n) = \Theta(n)$. Με αντίστοιχο τρόπο παρατηρούμε ότι δεν είναι απαραίτητο να υπολογίσουμε επακριβώς τον αριθμό των στοιχειωδών πράξεων που εκτελεί ο αλγόριθμος όταν παύει να είναι αναδρομικός.

Μέθοδος

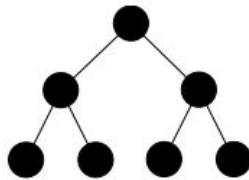
Παράδειγμα

Δένδρο Αναδρομής

Ένα δένδρο είναι ένα άκυκλο συνδεδεμένο γράφημα. Για παράδειγμα, στο Σχήμα 5.1 απεικονίζεται ένα δένδρο με 7 κορυφές (κόμβους). Στην απεικόνιση του σχήματος, πα-



Σχήμα 5.1: Δυαδικό δένδρο με επτά κορυφές



Σχήμα 5.2: Πλήρες δυαδικό δένδρο με επτά κορυφές

ρατηρούμε ότι οι κορυφές είναι διατεταγμένες σε επίπεδα. Η κορυφή που εμφανίζεται πάνω-πάνω (κορυφή *a*) ονομάζεται *ρίζα*. Οι κορυφές που δεν συνδέονται με κάποια κορυφή σε χαμηλότερο επίπεδο ονομάζονται φύλλα ενώ οι υπόλοιπες εσωτερικές. Για το δένδρο του Σχήματος 5.1 φύλλα αποτελούν οι κορυφές *c,d,f,g* ενώ εσωτερικές είναι οι κορυφές *b,e*. Κάθε εσωτερική κορυφή έχει κορυφές απόγονους ενώ κάθε φύλλο όχι. Πρόγονος μίας κορυφής *u* είναι η κορυφή *w* που βρίσκεται σε αμέσως προηγούμενο επίπεδο από την *u* και συνδέεται με αυτή με ακμή. Κάθε κορυφή έχει έναν ακριβώς πρόγονο εκτός από την *ρίζα* που δεν έχει κανένα.

Υψος μίας κορυφής είναι ο αριθμός των ακμών στο μακρύτερο μονοπάτι που συνδέει την κορυφή με κάποιο φύλλο. Το κάθε φύλλο έχει ύψος ίσο με μηδέν. Βάθος μίας κορυφής είναι ο αριθμός των ακμών στο μονοπάτι που συνδέει την κορυφή με τη *ρίζα*. Η *ρίζα* έχει βάθος ίσο με μηδέν. Ύψος ενός δένδρου είναι το ύψος της *ρίζας* του.

Ένα δένδρο ονομάζεται *δυαδικό* αν κάθε εσωτερικός κόμβος έχει το πολύ δύο απογόνους. Το δένδρο του Σχήματος 5.1 είναι δυαδικό. Ένα δένδρο ονομάζεται *πλήρες* δυαδικό αν κάθε εσωτερικός κόμβος έχει ακριβώς δύο απογόνους. Το πλήρες δυαδικό δένδρο με επτά κορυφές παρουσιάζεται στο Σχήμα 5.2.

Ορισμοί για τα δένδρα

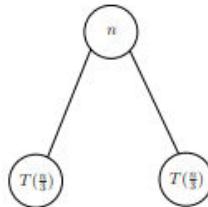
Έστω d ο αριθμός των κορυφών ενός δυαδικού δένδρου και h το ύψος του. Τότε, σε ένα δυαδικό δένδρο σε βάθος m υπάρχουν από 1 έως 2^m κορυφές. Συνεπώς,

$$h + 1 \leq d \leq 2^{h+1} - 1 \quad (5.7)$$

Το κάτω φράγμα στο d προκύπτει στην περίπτωση που το ύψος κάθε κορυφής είναι μοναδικό· δηλαδή, κάθε εσωτερική κορυφή έχει μόνο έναν πρόγονο και έναν απόγονο, η ρίζα έχει μόνο έναν απόγονο ενώ υπάρχει μόνο ένα φύλλο. Το άνω φράγμα στο d προκύπτει για ένα πλήρες δυαδικό δένδρο. Συνεπώς ένα πλήρες δυαδικό δένδρο έχει $2^{h+1} - 1$ κορυφές, 2^h φύλλα και $2^h - 1$ εσωτερικές κορυφές.

Πόρισμα 4. Ο μεγαλύτερος αριθμός κορυφών που μπορεί να έχει ένα δυαδικό δένδρο ύψους h είναι $2^{h+1} - 1$.

Από την (5.7) μπορούμε να εξάγουμε φράγματα για το ύψος του δένδρου σε συνάρ-



Σχήμα 5.3: Δένδρο αναδρομής - ανάπτυξη της ρίζας

τηση με τον αριθμό των κορυφών που περιέχει. Παίρνοντας τους λογαριθμούς, έχουμε

$$\lg(d + 1) - 1 \leq h \leq d - 1 \quad (5.8)$$

Επειδή το h είναι ακέραιος από το αριστερό μέλος της (5.8) έχουμε

$$\lceil \lg(d + 1) - 1 \rceil \leq h \Rightarrow \lceil \lg(d + 1) \rceil - 1 \leq h. \quad (5.9)$$

Η (5.9) ισχύει σαν ισότητα όταν το δένδρο είναι πλήρες δυαδικό.

Πόρισμα 5. Ανάμεσα στα δυαδικά δένδρα με d κορυφές, όπου $d = 2^k$, $k \geq 2$, αυτό που έχει το μικρότερο ύψος είναι το πλήρες δυαδικό.

Ένα πιο «απλό» κάτω φράγμα για το ύψος του δένδρου προκύπτει ως εξής:

$$2^{h+1} - 1 \geq d \Rightarrow 2^{h+1} > d \Rightarrow h + 1 > \lg d \geq \lfloor \lg d \rfloor.$$

Επειδή το ύψος h είναι ακέραιος η παραπάνω σχέση συνεπάγεται

$$h \geq \lfloor \lg d \rfloor. \quad (5.10)$$

Ας δούμε τώρα πως μπορούμε να χρησιμοποιήσουμε τη δενδρική δομή προκειμένου να επιλύσουμε μία (αναδρομική) σχέση που περιγράφει τον αριθμό των ΣΥΒ ενός αναδρομικού αλγόριθμου. Παρατηρούμε ότι οι αναδρομικές κλήσεις μπορούν να αναπαρασταθούν από ένα δένδρο υπολογισμού: κάθε κορυφή του δένδρου αντιστοιχεί σε μία αναδρομική κλήση η οποία εκτελεί μία σειρά από ΣΥΒ. Αθροίζοντας τον αριθμό των ΣΥΒ για τις κορυφές που βρίσκονται στο ίδιο ύψος, βρίσκουμε τη συνεισφορά του ύψους αυτό στο συνολικό αριθμό των ΣΥΒ. Στη συνέχεια αθροίζουμε ως προς κάθε ύψος για να βρούμε τον συνολικό αριθμό ΣΥΒ.

Μέθοδος

Μπορούμε να γενικεύσουμε την ανάλυση που πραγματοποιήθηκε ως εξής. Έστω

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + cn, & \text{για } n > 1, \\ 1, & \text{για } n \leq 1, \end{cases} \quad (5.13)$$

όπου $a, b > 1$. Το ύψος του δένδρου αναδρομής (μήκος μονοπατιού από τη ρίζα μέχρι κάποιο φύλλο του δένδρου) είναι $\lceil \lg_b n \rceil$. οι κορυφές διακρίνονται σε επίπεδα τα οποία αριθμούνται από 0 (επίπεδο στο οποίο βρίσκεται η ρίζα του δένδρου) μέχρι $\lceil \lg n \rceil$ (επίπεδο στο οποίο βρίσκονται τα φύλλα). Επομένως, αριθμός των ΣΥΒ σε κάθε επίπεδο j είναι

$$\begin{aligned} \left(\frac{a}{b}\right)^j \cdot n, & \quad j = 0, \dots, \lceil \lg_b n \rceil - 1, \\ a^j, & \quad j = \lceil \lg_b n \rceil. \end{aligned}$$

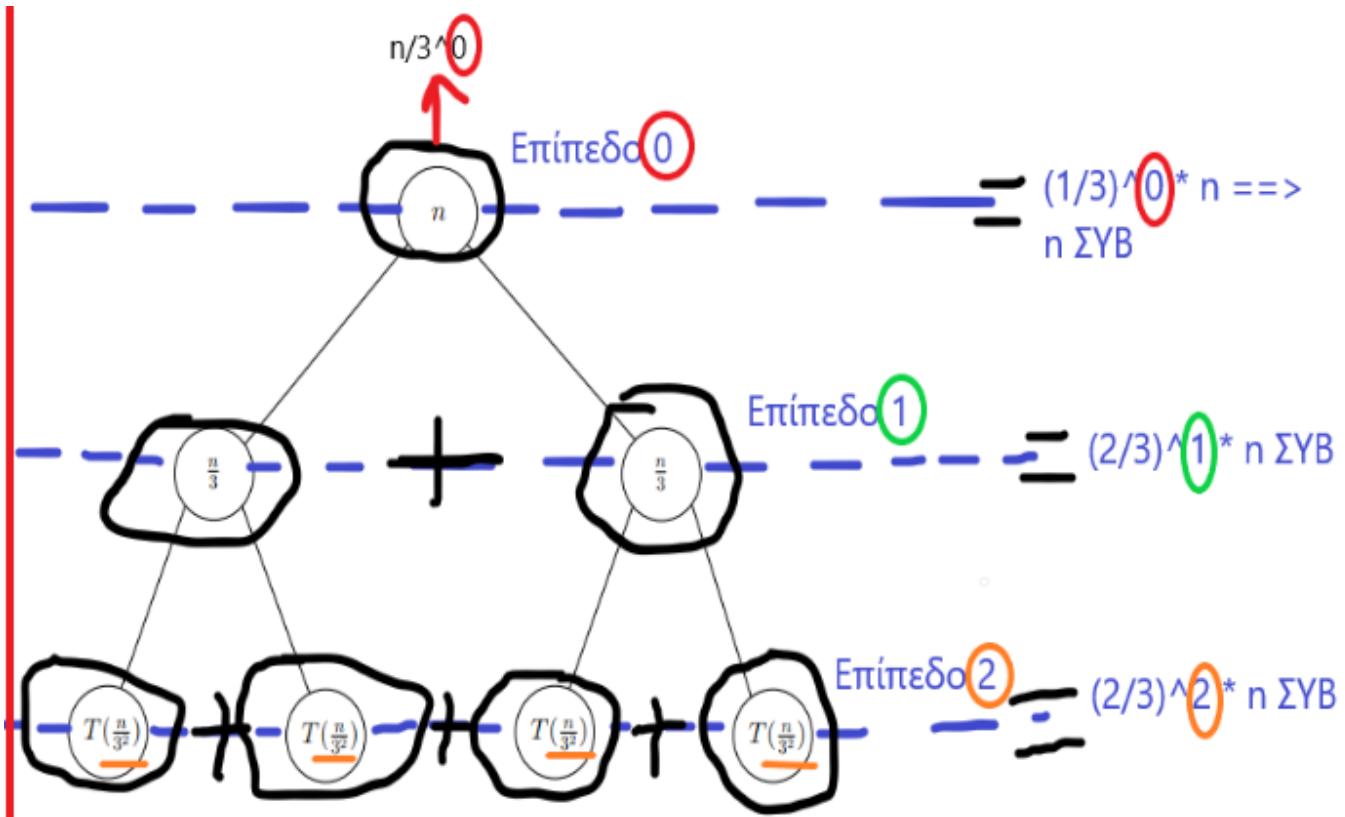
Το δένδρο αναδρομής είναι ιδιαίτερα χρήσιμο για την μελέτη αναδρομικών συναρτήσεων στις οποίες περιλαμβάνονται περισσότεροι του ενός αναδρομικοί όροι στο δεξί μέλος. Ένα παράδειγμα αποτελεί η ακόλουθη πρόταση.

Μέθοδος

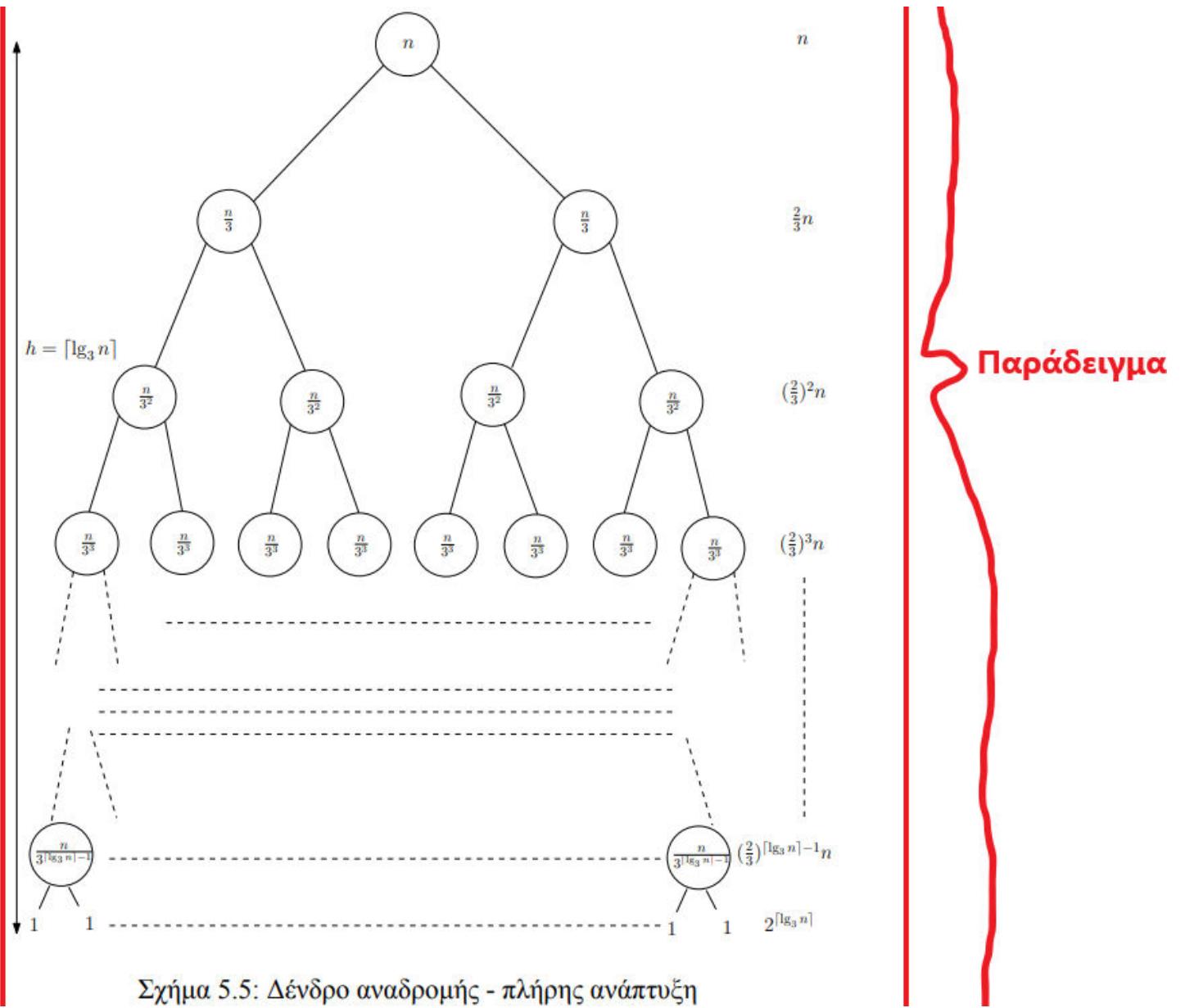
Παράδειγμα 16. Έστω ότι ο αριθμός των ΣΥΒ κάποιου αλγόριθμου δίνεται από την αναδρομική συνάρτηση

$$T(n) = \begin{cases} 2T\left(\frac{n}{3}\right) + n, & \text{για } n > 1, \\ 1, & \text{για } n \leq 1. \end{cases} \quad (5.11)$$

Το δένδρο υπολογισμού αρχικά έχει μόνο μία κορυφή η οποία αντιστοιχεί στην κλήση της συνάρτησης με παράμετρο $n > 1$. Αυτή η κορυφή θα αποτελέσει τη ρίζα του δένδρου. Αν αντικαταστήσουμε με τον αριθμό των ΣΥΒ από την (5.11) έχουμε, σε όρους της αναδρομικής συνάρτησης, την εικόνα του Σχήματος 5.3. Αναπτύσσοντας πάλι από την (5.11), τις δύο μη-ριζικές κορυφές έχουμε το δένδρο του Σχήματος 5.4. Συνεχίζουμε με τον ίδιο τρόπο μέχρι ο αναδρομικός αλγόριθμος να κληθεί με τιμή παραμέτρου μικρότερη-ίση με ένα. Το δένδρο σε πλήρη ανάπτυξη απεικονίζεται



Σχήμα 5.4: Δένδρο αναδρομής - επίπεδα 0-2



στο Σχήμα 5.5. Το δένδρο είναι δυαδικό και πλήρες. Άρα ο αριθμός των ακμών σε οποιοδήποτε μονοπάτια από τη ρίζα σε φύλλο αποτελεί το ύψος του δένδρου. Στο μονοπάτι αυτό η κάθε ακμή αντιστοιχεί σε μία διαίρεση του μεγέθους του στιγμιότυπου με το 3. Άρα το ύψος του δένδρου, έστω h , μας δίνει μικρότερο αριθμό διαιρέσεων του n με το 3 προκειμένου να φτάσουμε σε τιμή μικρότερη-ίση με 1. Δηλαδή,

$$\frac{n}{3^h} \leq 1 \Rightarrow n \leq 3^h \Rightarrow h \geq \lg_3 n \Rightarrow h = \lceil \lg_3 n \rceil. \quad (5.12)$$

Παρατηρούμε ότι η (5.12) δίνει το ύψος του δένδρου όπως αυτό προκύπτει από την (5.9), η οποία ισχύει σαν ισότητα (αφού το δένδρο είναι πλήρες δυαδικό), με αριθμό κορυφών $d = 2^{\lceil \lg_3 n \rceil + 1} - 1$. Στο Σχήμα 5.5 σημειώνεται το ύψος του δένδρου καθώς και ο αριθμός των ΣΥΒ που εκτελούνται σε κάθε ύψος (ποσότητες στο αριστερό περιθώριο του σχήματος).

Μετρώντας από επάνω προς τα κάτω, θεωρούμε τον δείκτη βάθους $j \in \{0, \dots, \lceil \lg_3 n \rceil\}$ - σε βάθος $j = 0$ βρίσκεται η ρίζα του δένδρου που συνεισφέρει n στο συνολικό αριθμό των, σε βάθος $j = 1$ οι δύο κορυφές που συνεισφέρουν η καθεμία $\frac{n}{3}$, κλπ. Δηλαδή, ο αριθμός των ΣΥΒ σε κάθε βάθος είναι

$$\begin{aligned} & \left(\frac{2}{3}\right)^j \cdot n, \quad j = 0, \dots, \lceil \lg_3 n \rceil - 1, \\ & 2^j \cdot 1, \quad j = \lceil \lg_3 n \rceil. \end{aligned}$$

Αθροίζοντας, για όλες τις τιμές του δείκτη j , έχουμε

$$\begin{aligned} T(n) &= n \sum_{j=0}^{\lceil \lg_3 n \rceil - 1} \left(\frac{2}{3}\right)^j + 2^{\lceil \lg_3 n \rceil} \\ &\leq n \sum_{j=0}^{(\lg_3 n+1)-1} \left(\frac{2}{3}\right)^j + 2^{(\lg_3 n+1)} \\ &= n \sum_{j=0}^{\lg_3 n} \left(\frac{2}{3}\right)^j + 2 \cdot 2^{\lg_3 n} \end{aligned}$$

Επομένως,

$$\begin{aligned} T(n) &\leq n \sum_{j=0}^{\infty} \left(\frac{2}{3}\right)^j + 2 \cdot n^{\lg_3 2} \\ &= \frac{1}{1 - \frac{2}{3}} n + 2n^{\lg_3 2} = 3n + 2n^{\lg_3 2} \\ &\leq 3n + 2n \\ T(n) &\leq 5n. \end{aligned}$$

Από την τελευταία ανισότητα έχουμε ότι $T(n) = O(n)$.

Πρόταση 2. Έστω

$$T(n) = \begin{cases} T(n/a) + T(n/b) + \Theta(n), & n > 1, \\ 0 & n \leq 1, \end{cases}$$

όπου $a, b > 1$. Τότε

$$T(n) = \begin{cases} \Theta(n), & \text{av } a^{-1} + b^{-1} < 1, \\ \Theta(n \lg n), & \text{av } a^{-1} + b^{-1} = 1. \end{cases}$$

Απόδειξη. Το δένδρο αναδρομής απεικονίζεται στο Σχήμα 5.6. Παρατηρούμε ότι το δένδρο δεν είναι ισοζυγισμένο: ο αριθμός των ακμών σε κάθε μονοπάτι από την ρίζα σε ένα φύλο δεν είναι ίδιος. Αν υποθέσουμε (χωρίς βλάβη της γενικότητας) ότι $a < b$ ο αριθμός αυτός κυμαίνεται από $\lceil \lg_b n \rceil$ μέχρι $\lceil \lg_a n \rceil$.³ Επομένως το ύψος του δένδρου είναι $\lceil \lg_a n \rceil$.

Οπως φαίνεται και από το Σχήμα 5.6 η συνεισφορά των εσωτερικών κορυφών σε βάθος k στην συνάρτηση $T(n)$ είναι⁴

$$cn \left(\frac{1}{a} + \frac{1}{b} \right)^k. \quad (5.14)$$

Αφού το μεγαλύτερο βάθος δεν ξεπερνά το $\lceil \lg_a n \rceil$ αν αθροίσουμε τις παραπάνω ποσότητες για $k = 0, \dots, \lceil \lg_a n \rceil$ έχουμε ένα άνω φράγμα στον αριθμό των ΣΥΒ

$$T(n) \leq cn \sum_{j=0}^{\lceil \lg_a n \rceil - 1} \left(\frac{1}{a} + \frac{1}{b} \right)^j + 2^{\lceil \lg_a n \rceil} \cdot 0 = cn \sum_{j=0}^{\lceil \lg_a n \rceil - 1} \left(\frac{1}{a} + \frac{1}{b} \right)^j. \quad (5.15)$$

Συνεπώς,

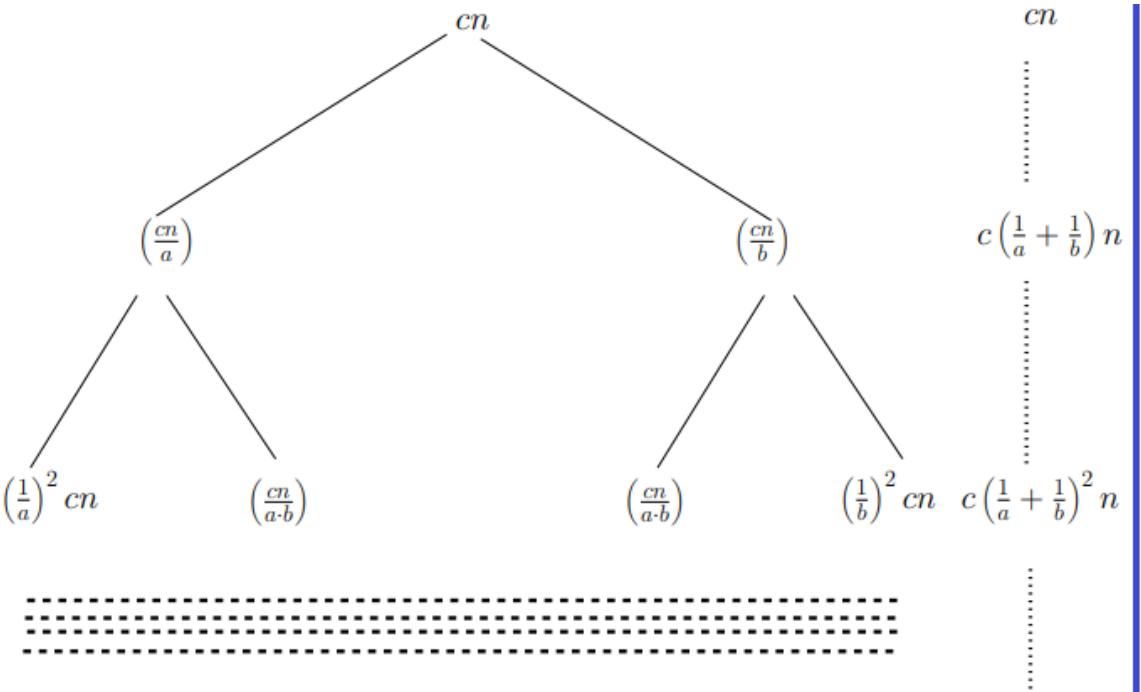
$$T(n) \leq \sum_{j=0}^{\infty} \left(\frac{1}{a} + \frac{1}{b} \right)^j cn = \frac{c}{1 - \left(\frac{1}{a} + \frac{1}{b} \right)} n. \quad (5.16)$$

Αν $\frac{1}{a} + \frac{1}{b} < 1$, ο παρανομαστής του δεξιού μέλους της (5.16) είναι θετικός και άρα το ίδιο ισχύει και για το κλάσμα με το οποίο πολλαπλασιάζεται η παράμετρος n . Άρα $T(n) = O(n)$. Περαιτέρω επειδή $T(n) \geq n \Rightarrow T(n) = \Omega(n)$, έχουμε ότι

$$T(n) = \Theta(n).$$

³Θέτοντας $x = \lg_a n$, $y = \lg_b n$ έχουμε $a^x = n = b^y$. Επειδή $a < b$, για να ισχύει η ισότητα, θα πρέπει $x < y$ και επομένως $\lg_b n < \lg_a n$. Οπότε ισχύει ότι $\lceil \lg_b n \rceil \geq \lceil \lg_a n \rceil$.

⁴Δες παράρτημα για μια αναστροφή απόδειξη της (5.14)



Σχήμα 5.6: Δένδρο αναδρομής, Πρόταση 2

Αν $\frac{1}{a} + \frac{1}{b} = 1$, η (5.15) συνεπάγεται ότι

$$T(n) \leq \sum_{j=0}^{\lceil \lg_a n \rceil - 1} \left(\frac{1}{a} + \frac{1}{b} \right)^j cn = cn \sum_{j=0}^{\lceil \lg_a n \rceil - 1} 1 = cn \cdot \lceil \lg_a n \rceil \leq cn(\lg_a n + 1). \quad (5.17)$$

Επειδή για οποιοδήποτε ακέραιο $a > 1$ ισχύει ότι $\lg_a n = \Theta(\lg n)$, η (5.17) συνεπάγεται $T(n) = O(n \lg n)$.

Αν θεωρήσουμε το άθροισμα της (5.14) για $k = 0, \dots, \lceil \lg_b n \rceil - 1$, τότε παίρνουμε ένα κάτω φράγμα για την $T(n)$. Ακολουθώντας αντίστοιχη αποδεικτική διαδικασία με την παραπάνω, βρίσκουμε ότι $T(n) = \Omega(n \lg n)$. \square

Θεώρημα Κυριαρχικού Όρου

Θεώρημα 1 (Θεώρημα κυριαρχου όρου - Master theorem). Έστω συνάρτηση $T(n)$ ορισμένη στους μη-αρνητικούς πραγματικούς που να ικανοποιεί την αναδρομική σχέση

$$T(n) = a \cdot T(n/b) + f(n)$$

όπου $a > 0, b > 1$ και $f(n)$ μη-αρνητική συνάρτηση ορισμένη στους θετικούς πραγματικούς. Τότε

$$T(n) = \begin{cases} \Theta(n^{\lg_b a}), & \text{αν } f(n) = O(n^{\lg_b a - \epsilon}), \epsilon > 0, \\ \Theta(n^{\lg_b a} \lg^{k+1} n), & \text{αν } f(n) = \Theta(n^{\lg_b a} \lg^k n), k \geq 0, \\ \Theta(f(n)), & \text{αν } f(n) = \Omega(n^{\lg_b a + \epsilon}), \epsilon > 0, \text{ και} \\ & af(n/b) \leq cf(n), c < 1 \end{cases}$$

Μέθοδος

Θεώρημα 2. Έστω σταθερές $a > 0, b > 1$. Θεωρούμε μη-αρνητική συνάρτηση $f(n)$ ορισμένη στο \mathbb{R}_+ και $T(n)$ ορισμένη στο \mathbb{N} τέτοια ώστε να ικανοποιεί την αναδρομική σχέση

$$T(n) = a_1 T(\lfloor \frac{n}{b} \rfloor) + a_2 T(\lceil \frac{n}{b} \rceil) + f(n)$$

όπου $a_1, a_2 \geq 0 \geq 1$, και $a = a_1 + a_2$. Τότε

$$T(n) = \begin{cases} \Theta(n^{\lg_b a}), & \text{av } f(n) = O(n^{\lg_b a - \epsilon}), \epsilon > 0, \\ \Theta(n^{\lg_b a} \lg^{k+1} n), & \text{av } f(n) = \Theta(n^{\lg_b a} \lg^k n), k \geq 0, \\ \Theta(f(n)), & \text{av } f(n) = \Omega(n^{\lg_b a + \epsilon}), \epsilon > 0, \text{ και} \\ & af(n/b) \leq cf(n), c < 1 \end{cases}$$

Μέθοδος

Ως παράδειγμα, θα χρησιμοποιήσουμε το Θεώρημα 1 για να δώσουμε ασυμπτωτικές εκτιμήσεις για τις συναρτήσεις

- a) $T(n) = 8T(n/2) + n^2$,
- b) $T(n) = 2T(n/4) + n^{1/2}$,
- c) $T(n) = 2T(n/3) + n$.

Έχουμε:

- a) $a = 8 = 2^3, b = 2, f(n) = n^2$. Είναι τετριψμένο ότι $n^2 = O(n^{3-\epsilon}), 1 \geq \epsilon \geq 0$ και $3 = \lg 2^3$. Επομένως,

$$f(n) = n^2 = O(n^{\lg 2^3 - \epsilon})$$

και άρα σύμφωνα με την πρώτη περίπτωση του κεντρικού θεωρήματος

$$T(n) = \Theta(n^3).$$

- b) $a = 2, b = 4, f(n) = n^{1/2} = \Theta(n^{\lg_4 2} \lg^0 n)$. Άρα σύμφωνα με τη δεύτερη περίπτωση του κεντρικού θεωρήματος για $k = 0$, έχουμε

$$T(n) = \Theta(n^{1/2} \lg n)$$

- c) $a = 2, b = 3, f(n) = n$. Επειδή

$$n \geq n^{\lg_3 2 + \epsilon}$$

όταν

$$1 \geq \lg_3 2 + \epsilon \Rightarrow 1 - \lg_3 2 \geq \epsilon \Rightarrow \lg_3 3 - \lg_3 2 \geq \epsilon \Rightarrow \lg_3 \frac{3}{2} \geq \epsilon.$$

Άρα

$$f(n) = \Omega(n^{\lg_3 2 + \epsilon}) \text{ οταν } \lg_3 \frac{3}{2} \geq \epsilon \geq 0.$$

Επίσης

$$2 \cdot n/3 \leq c \cdot n \text{ για } 1 > c \geq 2/3.$$

Από τα παραπάνω και σύμφωνα με την τρίτη περίπτωση του κεντρικού θεωρήματος έχουμε

$$T(n) = \Theta(n).$$

Παράδειγμα

Θα πρέπει να σημειώσουμε ότι το Θεώρημα 1 όπως διατυπώθηκε δεν καλύπτει κάθε περίπτωση αναδρομικής σχέσης. Για παράδειγμα η ασυμπτωτική συμπεριφορά της συνάρτησης

$$T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\lg n}$$

δεν μπορεί να εκτιμηθεί με βάση αυτό το θεώρημα αφού

$$\frac{\frac{n}{\lg n}}{n^{\lg 2}} = \frac{1}{\lg n} < n^\epsilon,$$

για κάθε $\epsilon > 0$.

Περιπτώσεις που
δεν ισχύει το
Θεώρημα
κυριαρχικού όρου

Μέθοδος των φραγμάτων

Πρόταση 2: «Δένδρο Αναδρομής με δύο αναδρομές»

Ενότητα 5.5: «Θεώρημα Κυριαρχικού Όρου»

Στην περίπτωση που η $T(n)$ είναι αρκετά σύνθετη για να τη χειριστούμε μέσω κάποιας από τις μεθόδους που αναπτύχθηκαν μέχρι τώρα μπορούμε να επιχειρήσουμε να την φράξουμε από επάνω και από τα κάτω από δύο έτερες συναρτήσεις οι οποίες μπορούν να μελετηθούν πιο εύκολα. Οι ασυμπτωτικές εκτιμήσεις των συναρτήσεων αυτών μπορούν να χρησιμοποιηθούν προκειμένου να εξαχθεί μία ασυμπτωτική εκτίμηση για την συνάρτηση την οποία φράζουν. Για παράδειγμα έστω ότι θέλουμε να υπολογίσουμε μία ασυμπτωτική εκτίμηση για τη συνάρτηση

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{n}{5}\right) + n. \quad (5.18)$$

Μέθοδος

Από την Πρόταση 2 γνωρίζουμε ότι $T(n) = \Theta(n)$. Θα δείξουμε πως μπορούμε να καταλήξουμε στο ίδιο αποτέλεσμα φράζοντας την $T(n)$ από επάνω και από κάτω. Παρατηρούμε ότι

$$2T\left(\frac{n}{5}\right) + n \leq T(n) \leq 2T\left(\frac{n}{3}\right) + n. \quad (5.19)$$

Ισοδύναμα,

$$\begin{aligned} T'(n) &\leq T(n) \leq T''(n), \text{ όπου,} \\ T'(n) &= 2T'\left(\frac{n}{5}\right) + n, T''(n) = 2T''\left(\frac{n}{3}\right) + n. \end{aligned}$$

Παράδειγμα

Στην Ενότητα 5.5 δείξαμε ότι $T''(n) = \Theta(n)$ (Παράδειγμα (γ)). Για την περίπτωση της $T'(n)$ έχουμε $a = 2, b = 5, f(n) = n$. Επειδή $n > n^{\lg_5 2 + \epsilon}$ για $\epsilon > 0$ και αρκετά μικρό ($\epsilon \leq \lg_5 \frac{5}{2}$), έχουμε ότι $n = \Omega(n^{\lg_5 2 + \epsilon})$. Επιπλέον

$$2f(n/5) = 2n/5 \leq cn,$$

με $1 > c \geq 2/5$. Επομένως σύμφωνα με την τρίτη περίπτωση του κεντρικού θεωρήματος έχουμε

$$T'(n) = \Theta(n). \quad (5.20)$$

Για παραπάνω σε συνδυασμό με την (5.19) συνεπάγονται

$$T(n) = \Theta(n).$$

Το θεώρημα αυτό μπορεί να θεωρηθεί σαν μία γενίκευση του κεντρικού θεωρήματος. Αφορά αναδρομικές σχέσεις της μορφής

$$T(n) = \begin{cases} \sum_{i=1}^k a_i T(n/b_i) + g(n), & n > n_0, \\ \Theta(1), & 1 \leq n \leq n_0, \end{cases} \quad (5.21)$$

όπου

- $k \geq 1$,
- $b_i > 1$, και $n_0 > \max\{b_i, b_i/(b_i - 1)\}$, για κάθε $i \in \{1, \dots, k\}$,
- $a_i > 0$, για κάθε $i \in \{1, \dots, k\}$,
- $g(n)$ είναι μη-αρνητική συνάρτηση για την οποία υπάρχουν σταθερές α, β τέτοιες ώστε $g(n) = O(n^\alpha)$ και $g(n) = \Omega(n^\beta)$, με $0 < \beta \leq \alpha$.

Θεώρημα 3 (Akra - Bazzi). Η λύση της αναδρομικής εξίσωσης (5.21) είναι

$$T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right) \quad (5.22)$$

όπου η παράμετρος p αποτελεί τη μοναδική λύση της εξίσωσης

$$\sum_{i=1}^k a_i \left(\frac{1}{b_i}\right)^p = 1. \quad (5.23)$$

Το Θεώρημα 3 έχει το πλεονέκτημα ότι δεν απαιτεί οι παράμετροι a_i, b_i να είναι ακέραιοι (ούτε καν ρητοί). Από την άλλη η χαρακτηριστική εξίσωση (5.23) μπορεί να μην έχει αναλυτική λύση· στη γενική περίπτωση χρειάζονται μέθοδοι αριθμητικής ανάλυσης για τον προσδιορισμό της τιμής της παραμέτρου p . Όμως σε πολλές περιπτώσεις κάτι τέτοιο δεν είναι απαραίτητο.

Μέθοδος

5.18: «Μέθοδος των φραγμάτων»

Παράδειγμα 17. Δίνεται η συνάρτηση

$$T(n) = T(3n/4) + T(n/4) + n.$$

Παρατηρούμε ότι $a_1 = 1, b_1 = 4/3, a_2 = 1, b_2 = 4$ και επομένως η εξίσωση (5.23) γίνεται

$$\left(\frac{3}{4}\right)^p + \left(\frac{1}{4}\right)^p = 1.$$

Η προφανής λύση της παραπάνω εξίσωσης είναι $p = 1$ και επομένως

$$T(n) = \Theta\left(n\left(1 + \int_1^n \frac{u}{u^{1+1}} du\right)\right) = \Theta\left(n\left(1 + \int_1^n \frac{1}{u} du\right)\right) = \Theta(n \log n).$$

Ένα πιο σύνθετο παράδειγμα αποτελεί η επίλυση της (5.18) με τη χρήση του Θεωρήματος 3. Στην περίπτωση αυτή η εξίσωση (5.23) είναι

$$\left(\frac{1}{3}\right)^p + \left(\frac{1}{5}\right)^p = 1$$

η οποία δεν φαίνεται να έχει αναλυτική λύση. Όμως παρατηρούμε ότι η συνάρτηση αυτή είναι φθίνουσα. Άρα $0 < p < 1$ και

$$\int_1^n \frac{g(u)}{u^{p+1}} du = \int_1^n u^{-p} du = \frac{u^{1-p}}{1-p}|_{u=1}^n = \frac{n^{1-p} - 1}{1-p} = \Theta(n^{1-p}).$$

Επομένως

$$T(n) = \Theta(n^p \cdot (1 + \Theta(n^{1-p}))) = \Theta(n).$$

Παράδειγμα

Μετατροπές

Αρκετές φορές οι αναδρομικές σχέσεις που καλούμαστε να μελετήσουμε δεν ταιριάζουν στα πρότυπα που έχουμε αναλύσει παραπάνω. Στην περίπτωση αυτή επιχειρούμε να μετατρέψουμε τη δοθείσα σχέση σε μια μορφή που ταιριάζει σε κάποιο πρότυπο ορίζοντας μια νέα συνάρτηση με βάση αυτή που προσπαθούμε να μελετήσουμε. Οι βασικοί τύποι μετατροπών είναι

- Μετατροπή του πεδίου ορισμού: ορίζουμε μία νέα συνάρτηση $S(n) = T(g(n))$ την οποία διέπει με μία απλούστερη αναδρομική σχέση,
- Μετατροπή του πεδίου τιμών: ορίζουμε μία νέα συνάρτηση $S(n) = g(T(n))$ την οποία διέπει με μία απλούστερη αναδρομική σχέση,
- Μετατροπή διαφοράς: απλοποιούμε την αναδρομική συνάρτηση $T(n)$ μελετώντας τη συνάρτηση διαφοράς $\Delta T(n) = T(n) - T(n-1)$.

Μέθοδος

Η ασυμπτωτική μελέτη της συνάρτησης

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + 1 \quad (5.24)$$

μπορεί να γίνει με τη χρήση της μετατροπής του πεδίου ορισμού. Συγκεκριμένα, αν θέσουμε $g(n) = 2^n$, μπορούμε να μελετήσουμε στη θέση της (5.24) την

$$T(g(n)) = T(2^n) = T(2^{n-1}) + T(2^{n-2}) + 1.$$

Θέτοντας $t(n) = T(2^n)$, η παραπάνω εξίσωση γίνεται

$$t(n) = t(n-1) + t(n-2) + 1 \quad (5.25)$$

η οποία μας θυμίζει την αναδρομική εξίσωση που παράγει τους αριθμούς Fibonacci. Επομένως $t(n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ και επειδή $t(n) = T(2^n)$ συνεπάγεται $T(n) = t(\lg n)$, έχουμε

$$T(n) = t(\lg n) = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^{\lg n}\right) = \Theta\left(n^{\lg\left(\frac{1+\sqrt{5}}{2}\right)}\right) = \Theta(n^{0.69424}).$$

Παράδειγμα της (α) περίπτωσης

Μία διαφορετική εφαρμογή της περίπτωσης (α) παρουσιάζεται στη συνέχεια. Έστω ότι θέλουμε να μελετήσουμε ασυμπτωτικά τη συνάρτηση

$$T(n) = \begin{cases} 2T\left(\frac{n}{3}\right) - 1, & n > 1, \\ 1, & n \leq 1. \end{cases} \quad (5.26)$$

Ορίζουμε μία νέα συνάρτηση

$$S(n) = T(n+a) \quad (5.27)$$

όπου a μία σταθερά τέτοια ώστε να ικανοποιείται η αναδρομική σχέση

$$S(n) = 2S\left(\frac{n}{3}\right) + \Omega(n). \quad (5.28)$$

Από την τρίτη περίπτωση του Θεωρήματος 1 έχουμε ότι

$$S(n) = \Theta(n). \quad (5.29)$$

Αν (μετα-)γράψουμε την (5.28) χρησιμοποιώντας την (5.27) έχουμε τη σχέση

$$T(n+a) = 2T\left(\frac{n}{3} + a\right) + \Omega(n). \quad (5.30)$$

Αμεσα από την (5.26) αν αντικαταστήσουμε το n από το $n+a$ παίρνουμε

$$T(n+a) = 2T\left(\frac{n+a}{3} - 1\right) + n + a. \quad (5.31)$$

Για να είναι (5.30) ισοδύναμη με την (5.31) θα πρέπει

$$\frac{n}{3} + a = \frac{n+a}{3} - 1 \Rightarrow a = -\frac{3}{2}.$$

Άρα

$$T(n) = S(n-a) = S\left(n + \frac{3}{2}\right) = \Theta\left(n + \frac{3}{2}\right) = \Theta(n).$$

Παράδειγμα της (α) περίπτωσης

$$T(n) = T(\sqrt{n}) + p\sqrt{n}, \quad (5.32)$$

όπου p μία θετική σταθερά. Θα αναλύσουμε την ασυμπτωτική συμπεριφορά της συνάρτησης αυτής. Θέτουμε

$$m = \lg n \Rightarrow n = 2^m \Rightarrow \sqrt{n} = 2^{\frac{m}{2}}. \quad (5.33)$$

H (5.32) γίνεται

$$T(2^m) = T(2^{\frac{m}{2}}) + p \cdot 2^{\frac{m}{2}}. \quad (5.34)$$

Στη συνέχεια θεωρούμε τη συνάρτηση

$$S(m) = T(g(m)), \quad (5.35)$$

όπου $g(m) = 2^m$. Επομένως,

$$S(m) = T(2^m) \Rightarrow S(m/2) = T(2^{m/2}).$$

H (5.34) γίνεται

$$S(m) = S\left(\frac{m}{2}\right) + p \cdot 2^{\frac{m}{2}}. \quad (5.36)$$

Εξετάζοντας την (5.36) υπό το πρίσμα του κεντρικού θεωρήματος έχουμε ότι $a = 1$, $b = 2$, $f(m) = p \cdot 2^{\frac{m}{2}}$. Παρατηρούμε ότι $f(m) = p \cdot 2^{\frac{m}{2}} \geq m^{\lg 1+\epsilon} = m^\epsilon$ για $\epsilon > 0$ (π.χ. $\epsilon = 1$) και επομένως $f(m) = \Omega(m^{\lg 1+\epsilon})$. Επιπλέον, $1 \cdot f(m/2) = p \cdot 2^{\frac{m}{4}}$ είναι μικρότερο -ίσο από $c f(m) = c \cdot p \cdot 2^{\frac{m}{2}}$ για κάποιο $c < 1$, π.χ. θεωρήστε $c = 1/2$. Άρα από την τρίτη περίπτωση του κεντρικού θεωρήματος έχουμε ότι $S(m) = \Theta(f(m)) = \Theta(p \cdot 2^{\frac{m}{2}}) = \Theta(2^{\frac{m}{2}})$. Από τις (5.32), ..., (5.36), έχουμε,

$$T(n) = T(2^m) = S(m) = \Theta(2^{\frac{m}{2}}) = \Theta(2^{\frac{\lg n}{2}}) = \Theta(\sqrt{n}).$$

**Παράδειγμα
της (α)
περίπτωσης**

Θεώρημα 3: «Θεώρημα Akra-Bazzi»

Πρόταση 2: «Δένδρο Αναδρομής με δύο αναδρομές»

Για ένα παράδειγμα της περίπτωσης (β) θεωρούμε την αναδρομική σχέση

$$T(n) = \frac{1}{4}T(n/4) + \frac{3}{4}T(3n/4) + 1.$$

Τη σχέση αυτή μπορούμε να την επιλύσουμε εφαρμόζοντας το Θεώρημα 3. Εναλλακτικά ορίζουμε τη συνάρτηση $U(n) = nT(n)$. Πολλαπλασιάζοντας τα δύο μέλη της παραπάνω σχέσης με n έχουμε

$$U(n) = U(n/4) + U(3n/4) + n$$

η οποία με τη χρήση του δένδρου αναδρομής (Πρόταση 2) μας δίνει ότι

$$U(n) = \Theta(n \lg n) \Rightarrow T(n) = \Theta(\lg n)$$

**Παράδειγμα
της (β)
περίπτωσης**

Για ένα παράδειγμα της περίπτωσης (γ), θεωρούμε τη συνάρτηση (προκύπτει από τον αλγόριθμο της τυχαιοκρατικής ταχυταξινόμησης - randomized quicksort)

$$T(n) = \frac{2}{n} \sum_{k=0}^{n-1} T(k) + n.$$

Η δοθείσα αναδρομική συνάρτηση είναι **πλήρους-ιστορίας** αφού η τιμή της $T(n)$ εξαρτάται από τις τιμές που είχε πάρει η συνάρτηση αυτή για κάθε $k < n$. Πολλαπλασιάζοντας με n έχουμε

$$nT(n) = 2 \sum_{k=0}^{n-1} T(k) + n^2 \Rightarrow \quad (5.37)$$

$$(n-1)T(n-1) = 2 \sum_{k=0}^{n-2} T(k) + (n-1)^2. \quad (5.38)$$

Η διαφορά (5.37)-(5.38) δίνει τη σχέση

$$T(n) = \frac{n+1}{n} T(n-1) + 2 - \frac{1}{n}.$$

Ουσιαστικά οι παραπάνω χειρισμοί κατάφεραν να εκφράσουν την $T(n)$ σε σχέση μόνο με την $T(n-1)$. Στη συνέχεια θέτουμε $t(n) = T(n)/(n+1)$ και οι παραπάνω σχέση γίνεται

$$t(n) = t(n-1) + \frac{2}{n+1} - \frac{1}{n(n+1)} = t(n-1) + \frac{3}{n+1} - \frac{1}{n}.$$

Με διαδοχική αντικατάσταση και θεωρώντας ότι $t(0) = 1$ η παραπάνω σχέση συνεπάγεται

$$\begin{aligned} t(n) &= 1 + \sum_{j=2}^{n+1} \frac{3}{j} - \sum_{j=1}^n \frac{1}{j} \\ &= \frac{3}{n+1} - 2 + \sum_{j=1}^n \frac{3}{j} - \sum_{j=1}^n \frac{1}{j} \\ &= \frac{3}{n+1} - 2 + 2 \sum_{j=1}^n \frac{1}{j}. \end{aligned}$$

Επειδή $\frac{3}{n+1} - 2 = \Theta(1)$ και $\sum_{j=1}^n \frac{1}{j}$ είναι ο αρμονικός μέσος ο οποίος είναι τάξης $\Theta(\lg n)$, έχουμε ότι $t(n) = \Theta(\lg n)$. Αντικαθιστώντας, από τη σχέση $t(n) = T(n)/(n+1)$ προκύπτει ότι

$$T(n) = \Theta(n \lg n).$$

**Παράδειγμα
της (γ)
περίπτωσης**

Βασικές Αρχές

Η αναδρομή αποτέλεσε το κύριο θέμα προηγούμενου κεφαλαίου. Συνοπτικά, ένας αναδρομικός αλγόριθμος επιλύει το στιγμιότυπο ενός προβλήματος λύνοντας αναδρομικά μικρότερα στιγμιότυπα του ίδιου προβλήματος. Όταν σε κάθε αναδρομική κλήση συνδυάζονται λύσεις μικρότερων στιγμιότυπων (*υποπροβλημάτων*) τότε ο αλγόριθμος εμπίπτει στην κατηγορία «*Διαιρεί και Κυρίευε*». Συγκεκριμένα, το υπόδειγμα «*Διαιρεί και Κυρίευε*» περιλαμβάνει τρεις φάσεις σε κάθε επίπεδο αναδρομής (αναδρομικής κλήσης):

Διαιρεί: το παρόν στιγμιότυπο διαιρείται (διασπάται) σε στιγμιότυπα μικρότερου μεγέθους του ιδίου προβλήματος,

Κυρίευε: τα μικρότερα στιγμιότυπα (*υποπροβλήματα*) επιλύονται αναδρομικά - αν είναι αρκετά μικρά επιλύονται απευθείας,

Συνδύασε: οι λύσεις των μικρότερων στιγμιοτύπων συνδυάζονται προκειμένου να προκύψει η λύση στο παρόν στιγμιότυπο.

Για να είναι επιτυχημένη η εφαρμογή της στρατηγικής αυτής θα πρέπει τα στιγμιότυπα που δημιουργούνται σε κάθε αναδρομική κλήση να είναι περίπου ιδίου μεγέθους.

Στις επόμενες ενότητες θα παρουσιαστούν αλγόριθμοι που ακολουθούν το παραπάνω πρότυπο.

01_ΕΙΣΑΓΩΓΗ

Swap

Μπορεί να χρησιμοποιηθεί σαν υπο-ρουτίνα σε αλγόριθμο Ταξινόμησης ή και Αναζήτησης. Ο κώδικας δεν χρειάζεται παρά μόνο περιγραφικά να γράψεις τι κάνει ο αλγόριθμος.

Αλγόριθμος 2 Ανταλλαγή τιμών δύο μεταβλητών

Απαιτείται: Ακέραιες μεταβλητές n_1, n_2 .

Επιστρέφεται: n_1, n_2 με ανταλλαγμένες τιμές.

```
1: function Swap(int n1, int n2)
2:   n ← n1;
3:   n1 ← n2;
4:   n2 ← n;
5: end function
```

Sum_of_Terms

Παράδειγμα 1. Για δεδομένους θετικούς ακέραιους n, a, t , θέλουμε να υπολογίσουμε το άθροισμα $z = \sum_{j=0}^{n-1} (a + j \cdot t)$. Δηλαδή, μας ζητείται να υπολογίσουμε το άθροισμα n όρων αριθμητικής προόδου με πρώτο όρο το a και βήμα το t , ήτοι

$$z = \sum_{j=0}^{n-1} (a + j \cdot t) = a + (a + t) + (a + 2t) + \cdots + (a + (n - 1)t). \quad (1.1)$$

Αλγόριθμος 3 Υπολογισμός αθροίσματος $\sum_{j=0}^{n-1} (a + j \cdot t)$

Απαιτείται: θετικοί ακέραιοι n, a, t ;

Επιστρέφεται: άθροισμα n όρων αριθμητικής προόδου με πρώτο όρο το a και βήμα t

```
1: function Sum_of_Terms(int n, int a, int t)
2:   sum ← 0;
3:   j ← 1;
4:   while j ≤ n do
5:     sum ← sum + a;
6:     a ← a + t;
7:     j++;
8:   end while
9:   return sum;
10: end function
```

$$z \leftarrow \text{Sum_of_Terms}(3, 2, 1).$$

Από την εκχώρηση αυτή, η μεταβλητή z παίρνει την τιμή 9. Οι τιμές των μεταβλητών σε κάθε επανάληψη απεικονίζονται στον Πίνακα 1.2.

j	sum	a (Γραμμή 6)
1	2	3
2	5	4
3	9	7

Πίνακας 1.2: Εκτέλεση Αλγόριθμου 3 με $n = 3, a = 2, t = 1$

Russian_Multi

Παράδειγμα 2. Θέλουμε να υπολογίσουμε το γινόμενο δύο φυσικών $a, b > 0$. Ο ρωσικός πολλαπλασιασμός είναι μία διαδικασία η οποία υπολογίζει το γινόμενο αυτό χρησιμοποιώντας πολλαπλασιασμό και ακέραια διαίρεση με το 2 (συμπεριλαμβανομένου και του υπόλοιπου) και πρόσθεση. Η διαδικασία είναι επαναληπτική: σε κάθε επανάληψη η τιμή του b γίνεται ίση με το πάτωμα του μισού του και η τιμή του a διπλασιάζεται. Η διαδικασία ολοκληρώνεται όταν η τιμή του b γίνει ίση με μηδέν.³ Το γινόμενο των δύο αριθμών ισούται με το άθροισμα των τιμών του a για τις περιττές τιμές του b . Ο Αλγόριθμος 4 κωδικοποιεί τη διαδικασία. Στον Πίνακα 1.3 απεικονίζονται οι τιμές των

Αλγόριθμος 4 Υπολογισμός γινομένου $a \cdot b$ με ρωσικό πολλαπλασιασμό

Απαιτείται: $a, b \in \mathbb{N}, a, b > 0$

Επιστρέφεται: $a \cdot b$

```

1: function Russian_Mult(int a, int b)
2:   prod  $\leftarrow$  0;
3:   while b  $>$  0 do
4:     if b mod 2 = 1 then
5:       prod  $\leftarrow$  prod + a;
6:     end if
7:     b  $\leftarrow$   $\lfloor \frac{b}{2} \rfloor$ ;
8:     a  $\leftarrow$  2  $\cdot$  a;
9:   end while
10:  return prod;
11: end function

```

μεταβλητών σε κάθε επανάληψη από την κλήση του Αλγόριθμου 4 με τιμές παραμέτρων $a = 35, b = 18$. Το αποτέλεσμα της κλήσης είναι η τιμή 630 όπως εμφανίζεται στην τρίτη στήλη της τελευταίας γραμμής του πίνακα.

b	a	$prod$	$\lfloor \frac{b}{2} \rfloor$	$2 \cdot a$
18	35	0	9	70
9	70	70	4	140
4	140	70	2	280
2	280	70	1	560
1	560	630	0	1120

Πίνακας 1.3: Ρωσικός πολλαπλασιασμός για τον υπολογισμό του γινομένου $18 \cdot 35$

Αλγόριθμος 5 Διαιρέτες n .**Απαιτείται:** Ακέραιος $n > 1$ **Επιστρέφεται:** Εκτύπωση διαιρετών του n .

```

1: function Divisors(int n)
2:   bound  $\leftarrow \lfloor \sqrt{n} \rfloor$ ;
3:   for i  $\leftarrow 1$ ; i  $\leq \text{bound}$ ; i  $\text{++}$  do
4:     if n mod i = 0 then
5:       a  $\leftarrow i$ ;
6:       b  $\leftarrow \frac{n}{i}$ ;
7:       print a, b;
8:     end if
9:   end for
10: end function

```

Παράδειγμα 3. Θέλουμε να βρούμε τους διαιρέτες ενός ακέραιου n μεγαλύτερου της μονάδας. Ο Αλγόριθμος 5 επιλύει το πρόβλημα αυτό. Ο αλγόριθμος εξετάζει αν η μεταβλητή i αποτελεί διαιρέτη του n (Γραμμή 4), περίπτωση κατά την οποία τόσο η μεταβλητή a (Γραμμή 5) όσο και η μεταβλητή b (Γραμμή 6) αποτελούν διαιρέτες του n και ισχύει ότι $a \cdot b = n$. Ο αλγόριθμος υπολογίζει όλους τους διαιρέτες αφού για κάθε ζευγάρι a, b με την ιδιότητα το γινόμενο τους να είναι ίσο n , ένας τουλάχιστον από τους δύο πρέπει να είναι μικρότερος-ίσος του $\lfloor \sqrt{n} \rfloor$. στην αντίθετη περίπτωση έχουμε $a, b \geq \lfloor \sqrt{n} \rfloor + 1 > \sqrt{n}$ και επομένως $a \cdot b > \sqrt{n} \cdot \sqrt{n} = n$. Τέλος παρατηρούμε ότι στην περίπτωση που $\lfloor \sqrt{n} \rfloor = \sqrt{n}$ (δηλαδή \sqrt{n} είναι ακέραιος) έχουμε ότι $a = b = \sqrt{n}$. Στον Πίνακα 1.4 απεικονίζονται οι τιμές των μεταβλητών του Αλγόριθμου 5 για $n = 12$.

<i>i</i>	<i>a</i>	<i>b</i>
1	1	12
2	2	6
3	3	4

Πίνακας 1.4: Διαιρέτες του 12

Small_k

Άσκηση 2. Να εκπονήσετε έναν αλγόριθμο που να δέχεται σαν είσοδο έναν μη-αρνητικό ακέραιο n και υπολογίζει τον μικρότερο ακέραιο k για τον οποίο ισχύει ότι $n \leq 2^k$. Να υπολογίσετε τον αριθμό των επαναλήψεων που εκτελεί ο αλγόριθμος.

Αλγόριθμος 1 Μικρότερο k που ικανοποιεί τη σχέση $n \leq 2^k$.

Απαιτείται: Μη-αρνητικός ακέραιος n

Επιστρέφεται: Μικρότερος ακέραιος k για τον οποίο $n \leq 2^k$.

```
1: function Small_k(int n)
2:    $k \leftarrow 0$ ;
3:    $r \leftarrow 1$ ;
4:   while  $r < n$  do
5:      $r \leftarrow 2r$ ;
6:      $k++$ ;
7:   end while
8:   return  $k$ ;
9: end function
```

Λύση. Ο Αλγόριθμος 1 είναι ο ζητούμενος. Παρατηρούμε ότι ο βρόχος εκτελείται μέχρι $r = 2^k \geq n$. Δηλαδή ο k είναι ο μικρότερος ακέραιος για τον οποίο πρέπει να αληθεύει αυτή η ανισότητα. Έχουμε,

$$r = 2^k \geq n \Rightarrow \begin{cases} k \geq \lg n \Rightarrow k = \lceil \lg n \rceil & \text{αν } n > 1, \\ 0, & \text{αν } n \leq 1. \end{cases}$$

Η τελευταία ισότητα προκύπτει από το γεγονός ότι το k είναι ακέραιος. ■

NegDigits

Άσκηση 3. Να εκπονήσετε έναν αλγόριθμο που να τυπώνει τους διψήφιους ακέραιους με κανένα ίδιο ψηφίο. Να υπολογίσετε τον αριθμό των ΣΥΒ που εκτελεί ο αλγόριθμος.

Αλγόριθμος 2 Διψήφιοι ακέραιοι με κανένα ίδιο ψηφίο

Απαιτείται:

Επιστρέφεται: Ακέραιοι από 10 ως 99 με διαφορετικά ψηφία

```
1: function NegDigits
2:   for  $i \leftarrow 1; i \leq 9; i++$  do
3:     for  $j \leftarrow 0; j \leq 9; j++$  do
4:       if  $i \neq j$  then
5:          $num \leftarrow 10 \cdot i + j$ ;
6:         Output num;
7:       end if
8:     end for
9:   end for
10: end function
```

Λύση. Ο Αλγόριθμος 2 είναι ο ζητούμενος. Σε κάθε επανάληψη των γραμμών 4-6 εκτελούνται 4 SYB αν τα ψηφία διαφέρουν και 1 SYB αν τα ψηφία είναι ίδια. Ο αριθμός των επαναλήψεων για τον εσωτερικό βρόχο είναι ίσος με 10. Από τις επαναλήψεις αυτές σε μία μόνο έχουμε ίδια ψηφία. Επομένως ο συνολικός αριθμός των SYB για τις 10 επαναλήψεις που εκτελούνται στις γραμμές 4 ως 6 είναι ίσος με $9 \cdot 4 + 1 = 37$.

Ο αριθμός των SYB της δομής **for** που αφορούν τον δείκτη j είναι $3 \cdot 10 + 2 = 32$. Επομένως ο αριθμός των SYB που εκτελούνται από τη γραμμή 3 έως την 7 είναι $32 + 37 = 69$.

Επομένως 69 είναι ο αριθμός των SYB που εκτελούνται για κάθε επανάληψη του

εξωτερικού βρόχου. Ο αριθμός των επαναλήψεων του εξωτερικού βρόχου είναι ίσος με $9 - 1 + 1 = 9$ (Άσκηση 1). Επίσης αριθμός των SYB της δομής **for** που αφορούν τον δείκτη i είναι $3 \cdot 9 + 2 = 29$. Ο συνολικός αριθμός των SYB του Αλγόριθμου 2 είναι $9 \cdot 69 + 29 = 650$. ■

Division_n_m

Άσκηση 4. Να εκπονήσετε έναν αλγόριθμο οποίος να δέχεται σαν είσοδο μη αρνητικούς ακέραιους n, m , όπου $n \geq m > 0$, και να υπολογίζει το πηλίκο και το υπόλοιπο της διαίρεσης του n με το m εκτελώντας μόνο προσθέσεις και αφαιρέσεις. Πόσα SYB εκτελεί ο αλγόριθμος σας;

Αλγόριθμος 3 $n \text{ div } m, n \text{ mod } m$ μόνο με προθέσεις και αφαιρέσεις

Απαιτείται: Μη-αρνητικοί ακέραιοι n, m με $n \geq m > 0$.

Επιστρέφεται: $k = n \text{ div } m, d = n \text{ mod } m$ μόνο με προθέσεις και αφαιρέσεις.

```
1: function Division_n_m(int n, int m)
2:    $d \leftarrow n$ ;
3:    $k \leftarrow 0$ ;
4:   while  $d \geq m$  do
5:      $k++$ ;
6:      $d \leftarrow d - m$ ;
7:   end while
8:   Output k, d;
9: end function
```

Όπως είναι γνωστό αν $k, d \in \mathbb{Z}$ είναι το πηλίκο και το υπόλοιπο, αντίστοιχα, της διαιρεσης του n με τον m τότε

$$n = k \cdot m + d \Rightarrow \frac{n}{m} = k + \frac{d}{m}.$$

Επειδή $m - 1 \geq d \geq 0 \Rightarrow 1 > \frac{d}{m} \geq 0$ και k ακέραιος, η παραπάνω σχέση συνεπάγεται ότι $k = \lfloor \frac{n}{m} \rfloor$. Επομένως αυτός είναι ο αριθμός των επαναλήψεων του βρόχου στον Αλγόριθμο 3. Σε κάθε επανάληψη εκτελούνται 4 ΣΥΒ. Επίσης στον έλεγχο του βρόχου εκτελούνται συνολικά $k + 1$ συγκρίσεις. Προσθέτοντας και τις δύο εκχωρήσεις που γίνεται για την αρχικοποίηση των μεταβλητών στις γραμμές 2 και 3, έχουμε ότι ο συνολικός αριθμός των ΣΥΒ του αλγόριθμου είναι ίσος με

$$4\lfloor \frac{n}{m} \rfloor + \lfloor \frac{n}{m} \rfloor + 1 + 2 = 5\lfloor \frac{n}{m} \rfloor + 3.$$

■

MKD

Άσκηση 8. Έστω φυσικοί n, m με $n \geq m > 1$. Να υλοποιήσετε σε ψευδοκώδικα τον αλγόριθμο του Ευκλείδη ο οποίος υπολογίζει τον μέγιστο κοινό διαιρέτη των αριθμών αυτών. Να βρείτε ένα άνω φράγμα στον αριθμό των επαναλήψεων που εκτελεί ο αλγόριθμος.

Αλγόριθμος 6 $n \text{ div } m, n \text{ mod } m$ μόνο με προθέσεις και αφαιρέσεις

Απαιτείται: Μη-αρνητικοί ακέραιοι n, m με $n \geq m > 1$.

Επιστρέφεται: Μέγιστος κοινός διαιρέτης των n, m .

```
1: function MKD(int n, int m)
2:   a ← n;
3:   b ← m;
4:   while b > 0 do
5:     r ← a mod b;
6:     a ← b;
7:     b ← r;
8:   end while
9:   return a;
10: end function
```

$$r < \frac{a}{2}. \quad (1)$$

Για να το δείξουμε αυτό θεωρούμε τις περιπτώσεις (α) $b \leq \frac{a}{2}$ και (β) $b > \frac{a}{2}$. Στην περίπτωση (α) επειδή $0 < r \leq b - 1$ το ζητούμενο ισχύει. Στην περίπτωση (β) $r = a - b < \frac{a}{2}$ άρα πάλι το ζητούμενο ισχύει.

Παρατηρούμε ότι η η συνθήκη του βρόχου στη Γραμμή 4 μπορεί να γραφτεί λιγότερο αυστηρά ως $a \cdot b > 0$. Δηλαδή αντί να εξετάσουμε πως διαμορφώνεται η τιμή της μεταβλητής b σε κάθε επανάληψη - κάτι που δεν είναι τόσο προφανές - θα διερευνήσουμε το πως διαμορφώνεται η τιμή του γινομένου $a \cdot b$.

Παρατήρηση 1. Από τον Αλγόριθμο 6 παρατηρούμε ότι σε κάθε επανάληψη η τιμή της μεταβλητής b (της προηγούμενης επανάληψης) εικωρείται στην a και η τιμή της r (που υπολογίστηκε από τις τιμές των a, b της προηγούμενης επανάληψης) στη b .

Πολλαπλασιάζοντας τα δύο μέλη της (1) με b έχουμε

$$b \cdot r < \frac{1}{2}(a \cdot b).$$

Η σχέση αυτή σε συνδυασμό με την Παρατήρηση 1 μας οδηγεί στο συμπέρασμα ότι το γινόμενο $a \cdot b$ μειώνεται σε λιγότερο από το μισό σε κάθε επανάληψη. Μας ενδιαφέρει να προσδιορίσουμε την επανάληψη, έστω k , κατά την οποία το γινόμενο ab γίνεται ίσο με μηδέν - αυτή θα είναι η τελευταία επανάληψη του βρόχου της Γραμμής 4. Επειδή οι μεταβλητές a, b αρχικοποιούνται στις τιμές των n, m , αντίστοιχα, η τιμή k είναι ο μικρότερος ακέραιος για τον οποίο ισχύει

$$\begin{aligned} \lfloor \frac{n \cdot m}{2^k} \rfloor = 0 &\Rightarrow \frac{n \cdot m}{2^k} < 1 \Rightarrow n \cdot m < 2^k \Rightarrow \\ \lg n + \lg m < k &\Rightarrow k = \lfloor \lg n + \lg m \rfloor + 1. \end{aligned}$$

Επομένως η τιμή αυτή μας δίνει ένα άνω φράγμα στον αριθμό των επαναλήψεων του βρόχου της Γραμμής 4. Αυτή η τιμή του k αποτελεί φράγμα γιατί

- η τιμή του γινομένου $a \cdot b$ αποτελεί προσέγγιση (από τα επάνω) της τιμής της μεταβλητής b βάσει της οποίας γίνεται ο έλεγχος του βρόχου στη Γραμμή 4,
- η τιμή του γινομένου $a \cdot b$ σε κάποια επανάληψη μπορεί να μειώνεται (από την τιμή του γινομένου στην προηγούμενη επανάληψη) αρκετά περισσότερο από το μισό.

■

02_ΚΛΑΣΕΙΣ ΠΟΛΥΠΛΟΚΟΤΗΤΑΣ

Sum_of_Terms02

Pivot_Partition

Μπορεί να χρησιμοποιηθεί σαν υπο-ρουτίνα σε αλγόριθμο Ταξινόμησης ή και Αναζήτησης. Ο κώδικας δεν χρειάζεται παρά μόνο περιγραφικά να γράψεις τι κάνει ο αλγόριθμος.

Αλγόριθμος 10 Τοποθέτηση οδηγού - στοιχείο $A[put]$ - στη σωστή θέση

Απαιτείται: πίνακας A με στοιχεία στις θέσεις από lo έως hi , όπου $lo \leq hi \leq \text{SizeOf}(A)$, στοιχείο οδηγός στη θέση put , όπου $lo \leq put \leq hi$.

Επιστρέφεται: η σωστή θέση του οδηγού (επανατοποθέτηση του) αφού έχει αναδιαρθρωθεί ο πίνακας A

```

1: function Pivot_Partition(int A[], int lo, int hi, int pvt)
2:   Swap(A[put], A[hi]);
3:   i  $\leftarrow$  lo; j  $\leftarrow$  hi;
4:   while i < j do
5:     while i < j and A[i] ≤ A[hi] do
6:       i  $\leftarrow$  i + +;
7:     end while
8:     while i < j and A[j] ≥ A[hi] do
9:       j  $\leftarrow$  j - -;
10:    end while
11:    Swap(A[i], A[j]);
12:   end while
13:   Swap(A[i], A[hi]);
14:   return i;
15: end function

```

Ο Αλγόριθμος 10 εκτελεί την λειτουργία τοποθέτησης ενός στοιχείου μίας ακολουθίας ακεραίων στη σωστή του θέση. Συγκεκριμένα, ο αλγόριθμος δέχεται σαν είσοδο έναν πίνακα A , στον οποίο βρίσκεται αποθηκευμένη η ακολουθία - στις θέσεις από lo έως hi - και μία θέση put , ($lo \leq put \leq hi$). Αναδιατάσσει τα στοιχεία της ακολουθίας ώστε το στοιχείο που βρισκόταν αρχικά στη θέση put να τοποθετηθεί στη σωστή θέση η οποία επιστρέφεται. Η αναδιάταξη γίνεται ως εξής. Αρχικώς, το στοιχείο στη θέση put , ονομαστικά οδηγός, τοποθετείται στην τελευταία θέση - ανταλλάσσεται με το στοιχείο στη θέση hi (Γραμμή 2).¹ Στη συνέχεια ο πίνακας σαρώνεται από δύο δείκτες: ο δείκτης i εκκινεί από τη θέση lo και αυξάνεται ενώ ο δείκτης j από τη θέση hi και μειώνεται. Όσο ισχύει ότι $i < j$ οι δείκτες κινούνται με αντίθετη φορά (όπως περιγράφηκε προηγουμένως) ακολουθώντας τον εξής κανόνα: ο δείκτης i σταματά (να αυξάνεται) αν το στοιχείο $A[i]$ είναι μεγαλύτερο από το οδηγό ενώ ο δείκτης j σταματά (να μειώνεται) αν το στοιχείο $A[j]$ είναι μικρότερο από το οδηγό. Όταν σταματήσουν οι δείκτες τότε τα στοιχεία $A[i], A[j]$ ανταλλάσσονται. Μετά την ανταλλαγή η σάρωση συνεχίζεται. Όταν οι δείκτες συναντηθούν τότε το στοιχείο στο οποίο δείχνουν ανταλλάσσεται με το οδηγό (ο οποίος βρίσκεται στη θέση hi). Η θέση στην οποία διασταυρώθηκαν οι δείκτες είναι η σωστή θέση στην οποία έχει τοποθετηθεί ο οδηγός και είναι αυτή που επιστρέφεται από τη συνάρτηση.

Περιγραφή

Παράδειγμα 8 (συνέχεια). Έστω ότι θέλουμε να τοποθετήσουμε το στοιχείο $A[3] = 5$ στη σωστή θέση· αυτό θα αποτελέσει και το στοιχείο οδηγό. Εφόσον το στοιχείο αυτό είναι στη θέση 3 αυτή είναι και η τιμή που αντιστοιχεί στην παράμετρο pvt . Ο Αλγόριθμος 10 καλείται ως

$$k \leftarrow \text{Pivot_Partition}(A, 1, 5, 3)$$

Αρχικώς το στοιχείο αυτό τοποθετείται στην τελευταία θέση του πίνακα - ανταλλάσσεται με το στοιχείο της τελευταίας θέσης. Οπότε αμέσως πριν την εκτέλεση του βρόχου της Γραμμής 4, ο πίνακας A έχει την εικόνα που παρουσιάζεται στον Πίνακα 3.1.

$\downarrow i$				
2	7	1	7	5
				$\uparrow j$

Πίνακας 3.1: Τοποθέτηση στοιχείου στη σωστή θέση - Παράδειγμα 8

Στη συνέχεια ο δείκτης i αυξάνεται μέχρι να βρει κάποιο στοιχείο με τιμή μεγαλύτερη από το 5. Ο δείκτης j μειώνεται μέχρι να υποδειξεί κάποιο στοιχείο που είναι μικρότερο από το 5. Ο δείκτης j σταματάει στη τρίτη θέση αφού $A[3] = 1 < 5$. Η κατάσταση απεικονίζεται στον Πίνακα 3.2. Στη συνέχεια τα στοιχεία $A[2], A[3]$ ανταλλάσσουν θέσεις -

	$\downarrow i$			
2	7	1	7	5
		$\uparrow j$		

Πίνακας 3.2: Τοποθέτηση στοιχείου στη σωστή θέση - Παράδειγμα 8

Πίνακας 3.3 - και η σάρωση συνεχίζεται. Στο επόμενο βήμα οι δείκτες συναντιούνται

	$\downarrow i$			
2	1	7	7	5
		$\uparrow j$		

Πίνακας 3.3: Τοποθέτηση στοιχείου στη σωστή θέση - Παράδειγμα 8

(δείχνουν στο ίδιο στοιχείο) το οποίο ανταλλάσσεται με τον οδηγό (που βρίσκεται στη θέση hi). Στο σημείο αυτό η διαδικασία ολοκληρώνεται και το στοιχείο 5 βρίσκεται στη σωστή θέση. Ο πίνακας A έχει την εικόνα

$$A = [2, 1, 5, 7, 7].$$

Παράδειγμα

Λήμμα 3. Ο Αλγόριθμος 10 εκτελεί $\Theta(n)$ ΣΥΒ.

Απόδειξη. Ο αριθμός των διαδοχικών φωλιασμένων βρόχων είναι δύο: ο εξωτερικός βρόχος - Γραμμές 4-12 - περιλαμβάνει τους δυο εσωτερικούς που περιγράφονται στις Γραμμές 5-7 και 8-10 οι οποίοι είναι «ξένοι» μεταξύ τους. Παρατηρούμε ότι ο αριθμός των επαναλήψεων που εκτελούνται συνολικά από τους βρόχους δεν ξεπερνά τον αριθμό των στοιχείων· το κάθε στοιχείο του πίνακα A σαρώνεται μία φορά - είτε από τον δείκτη i ή/και από τον j - ενώ για κάθε στοιχείο εκτελείται σταθερός αριθμός ΣΥΒ.² Επομένως η πολυπλοκότητα του αλγόριθμου είναι γραμμική στον αριθμό των στοιχείων (παράμετρος n). \square

Πολυπλοκότητα

H μέθοδος της ένθετης ταξινόμησης

Αλγόριθμος 11 Ενθετική Ταξινόμηση

Απαιτείται: πίνακας A πίνακας A με στοιχεία στις θέσεις από lo έως hi , όπου $lo \leq hi \leq \text{SizeOf}(A)$.

Επιστρέφεται: Ταξινομημένος πίνακας A

```

1: function Insert_Sort(int A[], int lo, int hi)
2:   for  $i \leftarrow lo + 1; i \leq hi; i++ \text{ do}$ 
3:      $j \leftarrow i;$ 
4:     while  $j \geq lo + 1 \text{ and } A[j] < A[j - 1] \text{ do}$ 
5:       Swap(A[j], A[j - 1]);
6:        $j --;$ 
7:     end while
8:   end for
9: end function

```

Σύμφωνα με τη μέθοδο αυτή ο πίνακας A , που περιέχει τα προς-ταξινόμηση στοιχεία, χωρίζεται σε δύο τμήματα: στο ταξινομημένο και στο αταξινόμητο. Το ταξινομημένο τμήμα περιέχει τα στοιχεία που βρίσκονται στις πρώτες θέσεις του πίνακα ενώ τα υπόλοιπα βρίσκονται στο αταξινόμητο. Σε κάθε επανάληψη ένα στοιχείο από το αταξινόμητο τμήμα τοποθετείται στο ταξινομημένο στη σωστή θέση. Το στοιχείο που επιλέγεται για να τοποθετηθεί στη σωστή - στοιχείο οδηγός - είναι το πρώτο από το αταξινόμητο τμήμα. Μετά το πέρας της επανάληψης το ταξινομημένο τμήμα έχει αυξηθεί κατά ένα στοιχείο ενώ το αταξινόμητο έχει μειωθεί κατά ένα. Προφανώς μετά από $n - 1$ επαναλήψεις όλα τα στοιχεία βρίσκονται στη σωστή θέση και ο πίνακας είναι ταξινομημένος. Στην αρχή της εκτέλεσης της διαδικασίας το ταξινομημένο τμήμα αποτελείται από το στοιχείο $A[lo]$ ενώ το αταξινόμητο από τα υπόλοιπα. Η διαδικασία απεικονίζεται στον Αλγόριθμο 11.

Περιγραφή

Παράδειγμα 8 (συνέχεια). Θα χρησιμοποιήσουμε τον Αλγόριθμο 11 προκειμένου να ταξινομήσουμε τα στοιχεία του πίνακα $A = [2, 7, 5, 7, 1]$. Στον Πίνακα 3.4 παρουσιάζεται η διάταξη των στοιχείων για κάθε τιμή του δείκτη j πριν τη Γραμμή 4 (δηλαδή πριν να εκτελεστεί ο εσωτερικός βρόχος) και μετά τη Γραμμή 7 (δηλαδή αφού ολοκληρωθεί ο εσωτερικός βρόχος). Ο οδηγός σημειώνεται με υπογράμμιση.

j	Γραμμή 4	Γραμμή 7
2	[2, <u>7</u> , 5, 7, 1]	[2, <u>7</u> , 5, 7, 1]
3	[2, 7, <u>5</u> , 7, 1]	[2, <u>5</u> , 7, 7, 1]
4	[2, 5, 7, <u>7</u> , 1]	[2, 5, 7, <u>7</u> , 1]
5	[2, 5, 7, 7, <u>1</u>]	[<u>1</u> , 2, 5, 7, 7]

Παράδειγμα

Πίνακας 3.4: Παράδειγμα ενθετικής ταξινόμησης

Λήμμα 4. Ο Αλγόριθμος 11 εκτελεί $O(n^2)$ ΣΥΒ.

Απόδειξη. Ο αλγόριθμος χρησιμοποιεί δύο φωλιασμένους βρόχους. Στον εσωτερικό (Γραμμές 4-7), σε κάθε επανάληψη, εκτελεί σταθερό αριθμό ΣΥΒ, έστω c_1 . Ο αριθμός των επαναλήψεων αποτελεί συνάρτηση του δείκτη i - δεν ξεπερνά το $i - (lo + 1) + 1 = i - lo$. Σε κάθε επανάληψη του εξωτερικού βρόχου (Γραμμές 2-8) εκτελείται σταθερός αριθμός από ΣΥΒ, έστω c_2 συν τα ΣΥΒ που εκτελούνται από τον εσωτερικό βρόχο. Ο αριθμός των επαναλήψεων του εξωτερικού βρόχου - μία επανάληψη για κάθε τιμή του δείκτη i - είναι $hi - (lo + 1) + 1$. Επιτλέον εκτελείται σταθερός αριθμός ΣΥΒ, έστω c_3 που αφορά την αρχικοποίηση του δείκτη i και τη σύγκριση τερματισμού. Ο συνολικός αριθμός ΣΥΒ, έστω $T(n)$ ικανοποιεί τη σχέση

$$\begin{aligned} T(n) &\leq c_3 + \sum_{i=lo+1}^{hi} \left(c_2 + \sum_{j=lo+1}^i c_1 \right) \\ &= \sum_{i=lo+1}^{hi} \sum_{j=lo+1}^i c_1 + \sum_{i=lo+1}^{hi} c_2 + c_3 \\ &= \sum_{i=lo+1}^{hi} c_1(i - lo) + \sum_{i=lo+1}^{hi} c_2 + c_3. \end{aligned}$$

Από την (3.1) έχουμε ότι $hi - lo = n - 1$ και επομένως η παραπάνω σχέση γίνεται

$$\begin{aligned} T(n) &\leq \sum_{i=1}^{n-1} i \cdot c_1 + \sum_{i=1}^{n-1} c_2 + c_3 = c_1 \frac{n(n-1)}{2} + c_2(n-1) + c_3 \Rightarrow \\ T(n) &= O(n^2). \end{aligned}$$

Πολυπλοκότητα

□

Μπορεί να χρησιμοποιηθεί σαν υπο-ρουτίνα σε αλγόριθμο Ταξινόμησης ή και Αναζήτησης. Ο κώδικας δεν χρειάζεται παρά μόνο περιγραφικά να γράψεις τι κάνει ο αλγόριθμος.

Αλγόριθμος 12 Συγχώνευση δύο ταξινομημένων πινάκων A, B σε έναν πίνακα C .

Απαιτείται: Ταξινομημένος πίνακας με ακέραιους στις θέσεις από loA μέχρι hiA , όπου $loA \leq hiA < \text{SizeOf}(A)$, ταξινομημένος πίνακας B με ακέραιους στις θέσεις από loB μέχρι hiB , όπου $loB \leq hiB < \text{SizeOf}(B)$

Επιστρέφεται: Ταξινομημένος πίνακας C με τα στοιχεία των πινάκων A, B στις θέσεις από lo μέχρι hi , όπου $lo \leq hi \leq \text{SizeOf}(C)$ και $hi - lo + 1 = (hiA - loA + 1) + (hiB - loB + 1)$

```

1: function Merge(int A[], int loA, int hiA, int B[], int loB, int hiB, int C[], int lo, int hi)
2:   i ← loA;
3:   j ← loB;
4:   k ← lo - 1;
5:   A[hiA + 1] ← B[hiB] + 1;
6:   B[hiB + 1] ← A[hiA] + 1;
7:   while i ≤ hiA or j ≤ hiB do
8:     k++;
9:     if A[i] ≤ B[j] then
10:      C[k] ← A[i];
11:      i++;
12:    else
13:      C[k] ← B[j];
14:      j++;
15:    end if
16:   end while
17: end function

```

Η συγχώνευση (των στοιχείων) δύο πινάκων A, B αφορά ταξινομημένους πίνακες. Ισοδυναμεί με την παραγωγή ενός τρίτου πινάκα C ο οποίος περιέχει τα στοιχεία των πινάκων ταξινομημένα. Η ιδέα είναι αρκετά διαισθητική: εκκινώντας από το πρώτο στοιχείο κάθε πίνακα επιλέγουμε το μικρότερο για να εκχωρηθεί στην παρούσα θέση στον πίνακα C . Στη συνέχεια επαναλαμβάνουμε τη διαδικασία συγκρίνοντας το στοιχείο που βρίσκεται στην επόμενη θέση του πίνακα από τον οποίο έγινε η εκχώρηση. Στην περίπτωση που ολοκληρωθεί η εκχώρηση των στοιχείων του ενός εκ' των A, B , αντιγράφονται τα υπόλοιπα στοιχεία του έτερου πίνακα στον πίνακα C . Ο Αλγόριθμος 12 υλοποιεί την παραπάνω περιγραφή.

Είσοδος του αλγόριθμου αποτελούν οι ταξινομημένοι πίνακες A, B με ακέραιους από τις θέσεις loA μέχρι hiA και loB μέχρι hiB , αντίστοιχα. Ο πίνακας C αποτελεί έξοδο του αλγόριθμου ο οποίος περιέχει ταξινομημένα τα στοιχεία των δύο πινάκων A, B στη θέση από lo μέχρι hi . Το πλήθος των στοιχείων της εξόδου δίνεται από την (3.1) η οποία εξειδικεύεται σε

$$n = hi - lo + 1 = (hiA - loA + 1) + (hiB - loB + 1).$$

Οι εκχωρήσεις στις Γραμμές 5,6 γίνονται προκειμένου να μην χρειάζεται να ελέγχουμε αν κάποιος από τους δείκτες i και j παίρνει τιμή μεγαλύτερη από hiA και hiB , αντίστοιχα.

Περιγραφή

Ενότητα 2.5.1: «Βρόχοι και Πολυπλοκότητα»

Η πολυπλοκότητα του Αλγόριθμου 12 καθορίζεται από τον αριθμό των επαναλήψεων του βρόχου **while** εφόσον σε κάθε επανάληψη εκτελείται σταθερός αριθμός ΣΥΒ (Ενότητα 2.5.1). Ο αριθμός των επαναλήψεων είναι ίσος με την τιμή της μεταβλητής k όταν ολοκληρώθει ο βρόχος, δηλαδή εκτελούνται n επαναλήψεις. Άρα ο Αλγόριθμος 12 είναι πολυπλοκότητας $\Theta(n)$.

} Πολυπλοκότητα

Search_Seq (Αταξινόμητος Πίνακας)

Αλγόριθμος 14 Εύρεση στοιχείου σε έναν πίνακα A .

Απαιτείται: Πίνακας ακέραιών A , ακέραιοι lo, hi με $lo \leq hi$, ακέραιος a .

Επιστρέφεται: i αν $A[i] = a$, $hi + 1$ διαφορετικά.

```
1: function Search_Seq(int a, int A[], int lo, int hi)
2:   i ← lo;
3:   while i ≤ hi do
4:     if A[i] = a then
5:       return i;
6:     end if
7:     i++;
8:   end while
9:   return i;
10: end function
```

Αν ο πίνακας A περιέχει αταξινόμητα δεδομένα δεν υπάρχουν πολλές επιλογές για τη στρατηγική αναζήτησης: σαρώνεται ο πίνακας από τη θέση lo ως τη θέση hi προκειμένου να βρεθεί το στοιχείο. Η ιδέα υλοποιείται από τον Αλγόριθμο 14. Τετριμμένα, ο αριθμός των ΣΥΒ είναι τάξης $\Theta(n)$ στη χειρότερη περίπτωση (το στοιχείο που αναζητείται δεν υπάρχει ή βρίσκεται στη θέση hi) και $\Theta(1)$ στην καλύτερη περίπτωση (το στοιχείο που αναζητείται βρίσκεται στη θέση lo).

Search_Seq (Ταξινομημένος Πίνακας)

Αλγόριθμος 14 Εύρεση στοιχείου σε έναν πίνακα A .

Απαιτείται: Πίνακας ακέραιών A , ακέραιοι lo, hi με $lo \leq hi$, ακέραιος a .

Επιστρέφεται: i αν $A[i] = a$, $hi + 1$ διαφορετικά.

```
1: function Search_Seq(int a, int A[], int lo, int hi)
2:   i ← lo;
3:   while i ≤ hi and A[i] ≤ a do
4:     if A[i] = a then
5:       return i;
6:     end if
7:     i++;
8:   end while
9:   return i;
10: end function
```

Αν ο πίνακας A είναι ταξινομημένος στις θέσεις από lo ως hi , μπορούμε να χρησιμοποιήσουμε τον Αλγόριθμο 14 όπως παρουσιάστηκε προηγουμένως. Όμως μπορούμε και να τον βελτιώσουμε περαιτέρω εμπλουτίζοντας τη συνθήκη του βρόχου με την απαίτηση το στοιχείο $A[i]$ να είναι μικρότερο-ίσο από τον προς-αναζήτηση ακέραιο a . Δηλαδή, η Γραμμή 3 του αλγόριθμου θα πρέπει να γίνει³

while $i \leq hi$ **and** $A[i] \leq a$ **do**

Με τον τρόπο αυτό δεν θα συνεχιστεί η σάρωση του πίνακα A αν η θέση που θα έπρεπε να βρίσκεται το προς-αναζήτηση στοιχείο a έχει προσπεραστεί· ισοδύναμα ο δείκτης i δείχνει στοιχείο με τιμή μεγαλύτερη του a .

Η προτεινόμενη αλλαγή μειώνει τον αριθμό των επαναλήψεων για ορισμένα στιγμότυπα αλλά δεν βελτιώνει την πολυπλοκότητα στη χειρότερη περίπτωση· για παράδειγμα, όταν το στοιχείο που αναζητείται βρίσκεται στη θέση hi ή δεν υπάρχει στον πίνακα και είναι μεγαλύτερο από τον ακέραιο $A[hi]$.

Exists_in_A

Search: «*Search_Seq (Αταξινόμητος Πίνακας)*» ή «*Search_Seq (Ταξινομημένος Πίνακας)*»

3.1: «*Λειτουργίες του Πίνακα*»

Αλγόριθμος 13 Χρήση συνάρτησης για την εύρεση στοιχείου σε έναν πίνακα A .

Απαιτείται: Πίνακας A με στοιχεία στις πρώτες n θέσεις και στοιχείο αναζήτησης a .

Επιστρέφεται: True αν υπάρχει το στοιχείο a στον πίνακα A και False διαφορετικά.

```
1: function Exists_in_A(int a, int A[])
2:   index  $\leftarrow$  Search(a, A, 1, n);
3:   if index  $> hi$  or A[index]  $\neq a$  then
4:     return False;
5:   else
6:     return True;
7:   end if
8: end function
```

Ένα παράδειγμα κλήσης της *Search* για την αναζήτηση του στοιχείου a ανάμεσα σε στοιχεία που καταλαμβάνουν τις n πρώτες θέσεις του πίνακα A απεικονίζεται στον Αλγόριθμο 13. Προφανώς η πολυπλοκότητα του Αλγόριθμου 13 ταυτίζεται με την πολυπλοκότητα του αλγόριθμου που υλοποιεί τη συνάρτηση *Search*. Το πλήθος των ακέραιων ανάμεσα στους οποίους αναζητείται ο ακέραιος a αποτελεί την παράμετρο n η οποία αντιπροσωπεύει το μέγεθος του στιγμιότυπου και δίνεται από τη (3.1).

Σε σχέση με την υλοποίηση της συνάρτησης *Search*, θα διακρίνουμε την περίπτωση κατά την οποία τα στοιχεία του πίνακα A είναι αταξινόμητα και αυτή που τα στοιχεία είναι ταξινομημένα.

Search_in_Group

Αλγόριθμος 15 Εύρεση στοιχείου με τη μέθοδο των ομάδων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi , στοιχείο αναζήτησης a .

Επιστρέφεται: Θέση που βρίσκεται το μικρότερο στοιχείο που είναι μεγαλύτερο-ίσο του A .

```

1: function Search_in_Group(int  $a$ , int  $A[]$ , int  $lo$ , int  $hi$ )
2:    $n \leftarrow hi - lo + 1$ ;
3:    $q \leftarrow \lfloor \sqrt{n} \rfloor$ ;
4:    $r \leftarrow (hi - lo) \bmod q$ ;
5:    $j \leftarrow hi$ ;
6:    $q_1 \leftarrow r$ ;
7:   if  $A[lo] > a$  then
8:     return  $lo$ ;
9:   end if
10:  if  $A[hi] < a$  then
11:    return  $hi + 1$ ;
12:  end if
13:  while  $j \geq lo + q_1$  and  $A[j] > a$  do
14:     $j \leftarrow j - q_1$ ;
15:     $q_1 \leftarrow q$ ;
16:  end while
17:  if  $A[j] = a$  then
18:    return  $j$ ;
19:  end if
20:   $ghi \leftarrow \min(hi, j + q_1) - 1$ ;
21:   $glo \leftarrow j + 1$ ;
22:  return Search_Seq( $a, A, glo, ghi$ );
23: end function
```

Για να εκμεταλλευτούμε καλύτερα τη διάταξη $A[lo] \leq A[lo + 1] \leq \dots \leq A[hi]$ θα εφαρμόσουμε μία διαφορετική στρατηγική: την αναζήτηση με τη χρήση ομάδων. Η βασική ιδέα συνίσταται στον διαμερισμό των n ταξινομημένων στοιχείων, σε ισομεγέθεις ομάδες από συνεχόμενα στοιχεία και στον εντοπισμό της ομάδας στην οποία πιθανώς να βρίσκεται το προς-αναζήτηση στοιχείο. Στη συνέχεια εκτελείται σειριακή αναζήτηση στην ομάδα αυτή. Η ιδέα υλοποιείται ως εξής. Αν η παράμετρος q συμβολίζει το πλήθος των στοιχείων της ομάδας, τότε θέτοντας

$$r = (hi - lo) \bmod q, \quad d = \lfloor \frac{hi - lo}{q} \rfloor.$$

μπορούμε να γράψουμε τη διαφορά $hi - lo$ ως

$$hi - lo = q \cdot d + r \Rightarrow hi - r = lo + q \cdot d \quad (3.2)$$

Η (3.2) λέει ότι αν από τη διαφορά του αριστερού μέλους αφαιρούμε το q, d φορές, θα «χτυπήσουμε» τη θέση lo . Αυτό σημαίνει ότι μπορούμε να διαμερίσουμε το σύνολο $\{lo, \dots, hi\}$ ως εξής

$$\begin{aligned} \{lo, \dots, hi\} &= \{hi - r + 1, \dots, hi\} \\ &\cup \bigcup_{j=1}^d \{hi - r - j * q + 1, \dots, hi - r - (j - 1) * q\} \\ &\cup \{lo\}. \end{aligned} \quad (3.3)$$

Το καθένα από τα σύνολα που αναφέρονται στο Λήμμα 5 αποτελεί μία ομάδα. Με δεδομένο ότι $A[lo] \leq a \leq A[hi]$, μπορούμε να εντοπίσουμε την ομάδα μέσα στην οποία πιθανώς να βρίσκεται το a αποκλείοντας τις υπόλοιπες. Για να δούμε το πως, έστω δύο από τις αναφερόμενες ομάδες τέτοιες ώστε να είναι διαδοχικές. Έστω ότι η ομάδα με τα μικρότερα στοιχεία δεικτοδοτείται από το σύνολο $\{lo_1, \dots, hi_1\}$ ενώ η άλλη από το $\{lo_2, \dots, hi_2\}$, όπου $lo_i \leq hi_i, i = 1, 2$. Λόγω του ότι ο πίνακας είναι ταξινομημένος και οι ομάδες είναι διαδοχικές, ισχύει ότι

$$A[lo_1] \leq \dots \leq A[hi_1] \leq A[lo_2] \leq \dots \leq A[hi_2], \quad (3.4)$$

με $lo_2 = hi_1 + 1$.

Αν το στοιχείο a ανήκει στην ομάδα που δεικτοδοτείται από το σύνολο $\{lo_2, \dots, hi_2\}$. Θα πρέπει

$$A[hi_2] \geq a \geq A[lo_2] = A[hi_1 + 1] \Rightarrow A[hi_2] \geq a \geq A[hi_1].$$

Η παραπάνω σχέση οδηγεί στην ακόλουθη στρατηγική αναζήτησης. Εκκινώντας από τη θέση $j = hi$ και μειώνοντας το δείκτη j αρχικά κατά r και σε κάθε επόμενη φορά κατά q , αναζητούμε την μεγαλύτερη τιμή j^* για την οποία $A[j^*] \leq a$. Δηλαδή

$$j^* = \operatorname{argmax}\{A[j] : A[j] \leq a, j \in \{hi, hi - r, hi - r - q, \dots, lo\}\}. \quad (3.5)$$

Προφανώς, αν $A[j^*] = a$ τότε ο ακέραιος a βρίσκεται στη θέση j^* και η διαδικασία τερματίζει. Αν $A[j^*] < a$ τότε ο ακέραιος a , αν υπάρχει στον πίνακα, θα βρίσκεται ανάμεσα στη θέση που ορίζει η αμέσως προηγούμενη τιμή του δείκτη j (από την j^*) μειωμένη κατά ένα και στη θέση $j^* + 1$. Ακολουθεί σειριακή αναζήτηση ανάμεσα στα στοιχεία της ομάδας αυτής και η διαδικασία τερματίζει. Παρατηρούμε ότι η σειριακή αναζήτηση εκτελείται σε ομάδα με το πολύ $q - 1$ στοιχεία.

Αν ο δείκτης j μειωθεί μέχρι την τιμή lo τότε η παραπάνω διαδικασία τερματίζει: επιστρέφει την τιμή αυτή ανεξάρτητα αν $A[lo] = a$ ή $A[lo] < a$, αφού και στις δύο περιπτώσεις ο ακέραιος $A[lo]$ είναι το μικρότερο στοιχείο του A για το οποίο ισχύει ότι $A[lo] \leq a$.

Ο Αλγόριθμος 15 υλοποιεί την διαδικασία που περιγράφαμε παραπάνω θέτοντας $q = \lfloor \sqrt{n} \rfloor$. Αυτή η επιλογή δεν επηρεάζει την πολυπλοκότητα του αλγόριθμου αφού το πάνω φράγμα που προκύπτει από την $f(q)$ και σε αυτή την περίπτωση είναι πολυώνυμο με μεγιστοβάθμιο όρο τον \sqrt{n} .

Η ομάδα μέσα στην οποία πιθανώς να βρίσκεται ο ακέραιος a δεικτοδοτείται από το σύνολο $\{glo, \dots, ghi\}$ (Γραμμές 20, 21). Παρατηρούμε ότι το σύνολο αυτό έχει το πολύ $q - 1$ στοιχεία αφού το στοιχείο $A[ghi + 1]$, έχει διαπιστωθεί ότι είναι μεγαλύτερο από το a στην προτελευταία επανάληψη του βρόχου της Γραμμής 13. Επίσης η χρήση της συνάρτησης \min στη Γραμμή 20 είναι απαραίτητη για την περίπτωση που ο βρόχος

Περιγραφή

Παράδειγμα 9. Θεωρούμε τον ταξινομημένο πίνακα

$$A = [1, 4, 4, 5, 7, 12, 15, 20].$$

Η εντολή

$$k \leftarrow \text{Search_in_Group}(6, A, 1, 8);$$

Θα εκχωρήσει στην μεταβλητή k την τιμή 5 : ο ακέραιος 6 δεν υπάρχει στη σειρά ενώ ο μικρότερος ακέριος που είναι μεγαλύτερος από αυτόν είναι ο 7 και βρίσκεται στη θέση 5. Αναλυτικά,

Γραμμή 2: $n = 8 - 1 + 1 = 8$

Γραμμή 3: $q = \lfloor \sqrt{8} \rfloor = 2$

Γραμμή 4: $r = (8 - 1) \bmod 2 = 1$

Γραμμή 5: $j = 8$

Γραμμή 6: $q1 = r = 1$

Γραμμή 8: $A[1] = 1 < a = 6$

Γραμμή 10: $A[8] = 20 > a = 6$

Γραμμή 13: $j = 8 \geq 1 + 2$ και $A[8] = 20 > a = 6$

Γραμμή 14: $j = 8 - 1 = 7$

Γραμμή 15: $q1 = 2$

Γραμμή 13: $j = 7 \geq 1 + 2$ και $A[7] = 15 > a = 6$

Γραμμή 14: $j = 7 - 2 = 5$

Γραμμή 15: $q1 = 2$

Γραμμή 13: $j = 5 \geq 1 + 2$ και $A[5] = 7 > a = 6$

Γραμμή 14: $j = 5 - 2 = 3$

Γραμμή 15: $q1 = 2$

Γραμμή 13: $j = 3 \geq 1 + 2$ και $A[3] = 4 < a = 6$

Γραμμή 17: $A[3] = 7 \neq a = 6$

Γραμμή 20: $ghi = \min\{8, 3 + 2\} - 1 = 4$

Γραμμή 21: $glo = 3 + 1 = 4$

Γραμμή 22: **return** Search_Seq(6, A, 4, 4)

Η συνάρτηση Search_Seq(6, A, 4, 4) επιστρέφει την θέση $hi + 1 = 4 + 1 = 5$ αφού $A[4] = 5 < a = 6$.

Παράδειγμα

Το μόνο που μένει είναι να καθοριστεί η τιμή της παραμέτρου q . Από την (3.5) προκύπτει άμεσα ότι ο αριθμός των επαναλήψεων προκειμένου να καθοριστεί η ομάδα μέσα στην οποία μπορεί να βρίσκεται το a δεν μπορεί να ξεπερνά το $d + 2$. Αν θεωρήσουμε ότι σε κάθε επανάληψη εκτελείται σταθερός αριθμός ΣΥΒ, ο συνολικός αριθμός των ΣΥΒ είναι $O(\lfloor \frac{d}{q} \rfloor)$. Επειδή $d = hi - lo \leq n - 1 < n$, μπορούμε να θεωρήσουμε ότι για τον καθορισμό της ομάδας εκτελούνται $O(\frac{n}{q})$ ΣΥΒ.

Όταν καθοριστεί η ομάδα, θα αναζητηθεί σειριακά το a ανάμεσα σε q το πολύ στοιχεία. Επομένως θα εκτελεστούν $O(q)$ ΣΥΒ (δες πολυπλοκότητα Αλγόριθμου 14).

Από την παραπάνω ανάλυση, προκύπτει ότι ο αριθμός των ΣΥΒ της αναζήτησης σε ομάδες $T(n)$ είναι τάξης $O(\frac{n}{q}) + O(q)$ και άρα υπάρχουν θετικές σταθερές c_1, c_2, n_0 τέτοιες ώστε

$$T(n) \leq f(q) = c_1 \cdot \frac{n}{q} + c_2 \cdot q, \quad (3.6)$$

για κάθε $n \geq n_0$.

Στόχος μας είναι να υπολογίσουμε την τιμή του q που να ελαχιστοποιεί την $f(q)$. Παραγωγίζοντας ως προς q , έχουμε

$$\frac{df(q)}{dq} = c_1 \frac{\frac{d}{q}}{dq} + c_2 \frac{dq}{dq} = \frac{-nc_1}{q^2} + c_2, \quad (3.7)$$

$$\frac{d^2f(q)}{dq^2} = \frac{2nc_1}{q^3}. \quad (3.8)$$

Εξισώνοντας την πρώτη παράγωγο με το μηδέν, έχουμε

$$\frac{df(q)}{dq} = 0 \Rightarrow c_1 \frac{-n}{q^2} + c_2 = 0 \Rightarrow c_1 \frac{n}{q^2} = c_2 \Rightarrow q^2 = n \frac{c_1}{c_2} \Rightarrow q = p \cdot \sqrt{n},$$

όπου $p = \sqrt{c_1/c_2}$. Επειδή ($n > 0, q > 0$), η δεύτερη παράγωγος (3.8) είναι μεγαλύτερη του μηδενός και άρα η τιμή

$$q = p \cdot \sqrt{n} \quad (3.9)$$

ελαχιστοποιεί την $f(q)$.

Αντικαθιστώντας την τιμή του q στην (3.6) έχουμε

$$T(n) \leq c_1 \cdot p \cdot \frac{n}{\sqrt{n}} + c_2 \cdot p \cdot \sqrt{n} = \left(\sqrt{\frac{c_1^3}{c_2}} + \sqrt{c_1 c_2} \right) \sqrt{n}$$

και επομένως

$$T(n) = O(\sqrt{n}).$$

Πολυπλοκότητα

k_seq_sum (Κυλιόμενο Παράθυρο)

Αλγόριθμος 16 Εύρεση μεγαλύτερου αθροίσματος k συνεχόμενων ακεραίων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi , ακέραιος .

Επιστρέφεται: Μεγαλύτερο άθροισμα k συνεχόμενων ακεραίων του πίνακα

```
1: function k_seq_sum(int  $k$ , int  $A[]$ , int  $lo$ , int  $hi$ )
2:    $max\_sum \leftarrow -\infty;$ 
3:   for  $i \leftarrow lo; i \leq hi - k + 1; i++ \text{ do}$ 
4:      $w \leftarrow 0;$ 
5:     for  $j \leftarrow 0; j \leq k - 1; j++ \text{ do}$ 
6:        $w \leftarrow w + A[i + j];$ 
7:     end for
8:     if  $max\_sum < w$  then
9:        $max\_sum \leftarrow w;$ 
10:    end if
11:   end for
12:   return  $max\_sum;$ 
13: end function
```

Η τεχνική αυτή χρησιμοποιείται προκειμένου να μειωθεί ο αριθμός των φωλιασμένων βρόχων. Απευθύνεται μόνο σε συγκεκριμένες περιπτώσεις όπου το παράθυρο υπολογισμού έχει σταθερό μέγεθος. Σαν υπόδειγμα θα θεωρήσουμε το εξής πρόβλημα: σε ένα πίνακα A που βρίσκονται αποθηκευμένοι ακέραιοι στις θέσεις από lo ως hi , θέλουμε να βρούμε το μεγαλύτερο άθροισμα k συνεχόμενων ακεραίων για δεδομένη τιμή της παραμέτρου αυτής η οποία αποτελεί το παράθυρο υπολογισμού. Ένας απλός τρόπος για να επιλυθεί το πρόβλημα αυτό παρουσιάζεται στον Αλγόριθμο 16.



Πολυπλοκότητα

Η πολυπλοκότητα του Αλγόριθμου 16. είναι ίση με $O(nk)$. νεγονός που προκύπτει από τη χρήση των δυο φωλιασμένων βρόχων.

k_seq_sum02 (Κυλιόμενο Παράθυρο με καλύτερη Πολυπλοκότητα)

Αλγόριθμος 17 Εύρεση μεγαλύτερου αθροίσματος k συνεχόμενων ακεραίων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi , ακέραιος .

Επιστρέφεται: Μεγαλύτερο άθροισμα k συνεχόμενων ακεραίων του πίνακα.

```
1: function k_seq_sum02(int  $k$ , int  $A[]$ , int  $lo$ , int  $hi$ )
2:    $w \leftarrow 0;$ 
3:   for  $i \leftarrow lo; i \leq lo + k - 1; i++ \text{ do}$ 
4:      $w \leftarrow w + A[i];$ 
5:   end for
6:    $max\_sum \leftarrow w;$ 
7:   for  $i \leftarrow lo + k; i \leq hi; i++ \text{ do}$ 
8:      $w \leftarrow w + A[i] - A[i - k];$ 
9:     if  $max\_sum < w$  then
10:       $max\_sum \leftarrow w;$ 
11:    end if
12:   end for
13:   return  $max\_sum;$ 
14: end function
```

Μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα ακολουθώντας την εξής στρατηγική. Αρχικώς υπολογίζουμε το άθροισμα των πρώτων k στοιχείων, ήτοι

$$w = \sum_{i=lo}^{lo+k-1} A[i].$$

Η τιμή αυτή αποτελεί το αποτέλεσμα του πρώτου παραθύρου. Αφαιρώντας από το w το στοιχείο $A[lo]$ και προσθέτοντας το στοιχείο $A[lo + k]$ έχουμε το αποτέλεσμα του δεύτερου παραθύρου, κοκ. Επαναλαμβάνοντας τη διαδικασία αυτή μπορούμε να υπολογίσουμε κάθε άθροισμα k συνεχόμενων στοιχείων χρησιμοποιώντας μόνο ένα βρόχο - διαδικασία κυλιόμενου παράθυρου. Ο Αλγόριθμος 17 κωδικοποιεί τη διαδικασία αυτή επιτυγχάνοντας πολυπλοκότητα $O(n)$.

Πολυπλοκότητα

Sum_Equals_a (Τεχνική δύο δεικτών)

Αλγόριθμος 10: «Pivot_Partition»

```

found ← False;
for  $i \leftarrow lo; i \leq hi - 1; i++$  do
    for  $j \leftarrow i + 1; j \leq hi; j++$  do
        if  $A[i] + A[j] = a$  then
            found ← True;
        end if
    end for
end for

```

Η τεχνική δύο δεικτών αφορά τη σάρωση ενός πίνακα με δύο δείκτες: ο ένας δείκτης εκκινεί από το πρώτο στοιχείο του πίνακα και αυξάνεται ενώ ο δεύτερος από το τελευταίο και μειώνεται. Η τεχνική αυτή χρησιμοποιήθηκε στον Αλγόριθμο 10 προκειμένου να τοποθετηθεί ένα στοιχείο στην σωστή θέση.

Η τεχνική αυτή όταν χρησιμοποιείται σε συνδυασμό με κάποιον πίνακα που είναι ταξινομημένος αποδεικνύεται αποτελεσματική σε όρους πολυπλοκότητας. Σαν εφαρμογή θεωρούμε το πρόβλημα εύρεσης ενός ζευγαριού ακεραίων σε έναν πίνακα A με άθροισμα μία συγκεκριμένη δεδομένη τιμή a . Ο πιο «αφελής» τρόπος είναι να χρησιμοποιηθούν δύο φωλιασμένοι βρόχοι:

Περιγραφή

Προφανώς η διαδικασία αυτή είναι πολυπλοκότητας $O(n^2)$. Σαρώνει τον πίνακα η στοιχείων 2 φορές, εξού και το n^2 .

Πολυπλοκότητα

Sum_Equals_a (Τεχνική δύο δεικτών με καλύτερη Πολυπλοκότητα)

Αλγόριθμος 18 Εύρεση δύο στοιχείων ενός πίνακα με άθροισμα δεδομένη τιμή a

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi , ακέραιος a .

Επιστρέφεται: True αν υπάρχουν i, j τέτοια ώστε $A[i] + A[j] = a$, False, διαφορετικά.

```
1: function Sum_Equals_a(int a, int A[], int lo, int hi)
2:   Ταξινόμηση  $A$ ;
3:    $i \leftarrow lo$ ;
4:    $j \leftarrow hi$ ;
5:   Found  $\leftarrow$  False;
6:   while  $i < j$  do
7:     temp  $\leftarrow A[i] + A[j] - a$ ;
8:     if temp  $= 0$  then
9:       Found  $\leftarrow$  True;
10:    else if temp  $< 0$  then
11:      i++;
12:    else
13:      j--;
14:    end if
15:   end while
16:   return Found
17: end function
```

Μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα ταξινομώντας πρώτα τον πίνακα και χρησιμοποιώντας στη συνέχεια την τεχνική των δύο δεικτών. Σαρώνεται ο ταξινομημένος πίνακας με έναν δείκτη i ο οποίος τοποθετείται στο πρώτο στοιχείο και βαίνει αυξανόμενος και ένα δείκτη j ο οποίος τοποθετείται στο τελευταίο στοιχείο και βαίνει μειούμενος. Αν το άθροισμα των δύο στοιχείων που δείχνουν οι δείκτες είναι μικρότερο του a τότε αυξάνεται ο δείκτης i ενώ αν είναι μικρότερο μειώνεται ο j . Όταν διασταυρωθούν οι δείκτες τότε η διαδικασία τερματίζει. Ο Αλγόριθμος 18 κωδικοποιεί τη διαδικασία.

Περιγραφή

Παρατηρούμε ότι η πολυπλοκότητα του Αλγόριθμου 18 είναι γραμμική από τη στιγμή που ο πίνακας A έιναι ταξινομημένος. Η διαδικασία ταξινόμησης - με αλγόριθμους που θα παρουσιαστούν σε επόμενο κεφάλαιο - μπορεί να πραγματοποιηθεί σε $O(n \lg n)$ ΣΥΒ και άρα αυτή είναι η πολυπλοκότητα του παραπάνω αλγόριθμου.

Πολυπλοκότητα

Push / Pop (Στοίβα)

Αλγόριθμος 19 Εισαγωγή στοιχείου στη στοίβα.

Απαιτείται: Πίνακας Q , ακέραιος $size_Q$, ακέραιος a .

Επιστρέφεται: Ο πίνακας Q ενημερωμένος με τον ακέραιο a .

```
1: function Push(int  $Q[]$ , int  $size\_Q$ , int  $a$ )
2:   if  $size\_Q = \text{SizeOf}(Q)$  then
3:     print Σφάλμα πλήρους στοίβας;
4:   else
5:      $size\_Q ++;$ 
6:      $Q[size\_Q] \leftarrow a;$ 
7:   end if
8: end function
```

Αλγόριθμος 20 Εξαγωγή στοιχείου από στοίβα.

Απαιτείται: Πίνακας Q , ακέραιος $size_Q$

Επιστρέφεται: -1 αν η στοίβα δεν περιέχει κανένα στοιχείο. Διαφορετικά, το στοιχείο που μπήκε πιο πρόσφατα στον Q .

```
1: function Pop(int  $Q[]$ , int  $size\_Q$ )
2:   if  $size\_Q = 0$  then
3:     print Σφάλμα κενής στοίβας;
4:     return  $-1$ ;
5:   else
6:      $a \leftarrow Q[size\_Q];$ 
7:      $size\_Q --;$ 
8:     return  $a$ ;
9:   end if
10: end function
```

Παραπάνω διαδικασίες: «ΤΥΠΟΙ-ΙΔΙΟΤΗΤΕΣ/03_ΠΙΝΑΚΕΣ/Στοίβα»

Για την υλοποίηση των παραπάνω διαδικασιών, θεωρούμε ότι με τη στοίβα Q σχετίζεται η μεταβλητή $size_Q$ η οποία περιέχει το πλήθος των στοιχείων της. Η στοίβα Q μπορεί να υλοποιηθεί σαν μοναδιάστατος πίνακας. Στην περίπτωση αυτή, η τιμή της $size_Q$ διαφέρει από την τιμή της $\text{SizeOf}(Q)$ αφού η τελευταία επιστρέφει τον αριθμό των θέσεων μνήμης του πίνακα. Οι λειτουργίες push και pop, στην περίπτωση που η στοίβα υλοποιείται από τον μονοδιάστατο πίνακα Q υλοποιούνται από τους Αλγόριθμους 19 και 20, αντίστοιχα. Η μεταβλητή $size_Q$ αρχικοποιείται στο μηδέν.



Περιγραφή

Παρατηρούμε ότι ο αριθμός των ΣΥΒ σε κάθε μία από τις λειτουργίες push και pop είναι τάξης $\Theta(1)$.

Πολυπλοκότητα

Διαδικασίες: «ΤΥΠΟΙ-ΙΔΙΟΤΗΤΕΣ/03_ΠΙΝΑΚΕΣ/Ουρά»

Όπως και στην περίπτωση της στοίβας, η ουρά Q μπορεί να υλοποιηθεί με τη χρήση μονοδιάστατου πίνακα: το πλήθος των στοιχείων στην ουρά δίνεται (πάλι) από την τιμή της μεταβλητής $size_Q$. Η τιμή της μεταβλητής αυτής μειώνεται (κατά ένα) από κάθε κλήση της διαδικασίας `dequeue` ενώ αυξάνεται (κατά ένα) από κάθε κλήση της διαδικασίας `enqueue`. Προφανώς αν η τιμή της μεταβλητής αυτής είναι ίση με μηδέν τότε

Περιγραφή

Left_Right_Sum (Τεχνική δύο δεικτών)

Άλγοριθμος 4 Υπολογισμός μεγαλύτερου αριστερού-δεξιού αθροίσματος

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi

Επιστρέφεται: Τιμή μεγαλύτερου αριστερού - δεξιού αθροίσματος a

```
1: function Left_Right_Sum(int A[], int lo, int hi)
2:    $i \leftarrow lo$ ;
3:    $j \leftarrow hi$ ;
4:    $a \leftarrow 0$ ;
5:    $left\_sum \leftarrow 0$ ;
6:    $right\_sum \leftarrow 0$ ;
7:   while  $i < j$  do
8:      $left\_sum \leftarrow left\_sum + A[i]$ ;
9:      $right\_sum \leftarrow right\_sum + A[j]$ ;
10:     $A[i] \leftarrow 0$ ;
11:     $A[j] \leftarrow 0$ ;
12:    if  $left\_sum > right\_sum$  then
13:       $j --$ ;
14:    else if  $left\_sum < right\_sum$  then
15:       $i ++$ ;
16:    else
17:       $a \leftarrow left\_sum$ ;
18:       $i ++$ ;
19:    end if
20:  end while
21:  return  $a$ ;
22: end function
```

Άσκηση 4. (Τεχνική δύο δεικτών) Δίνεται μια ακολουθία θετικών ακέραιών αριθμών οι οποίοι είναι αποθηκευμένοι σε έναν πίνακα A στις θέσεις από lo ως hi . Ζητείται αλγόριθμος γραμμικού χρόνου που να υπολογίζει το μεγαλύτερο δυνατό αριθμό a , για τον οποίο να υπάρχουν ακέραιοι p, q με την ιδιότητα

$$\sum_{i=lo}^{lo+p-1} A[i] = a = \sum_{j=hi-q+1}^{hi} A[j]$$

Πολυπλοκότητα

και επιπλέον $p + q \leq hi - lo + 1$.

Προσέξτε ότι το πρόβλημα έχει πάντα λύση (για $a = 0$ και $p = q = 0$).

Λύση. Θα χρησιμοποιήσουμε την τεχνική των δύο δεικτών: ο δείκτης i εκκινεί από θέση lo και βαίνει αυξανόμενος ενώ ο δείκτης j από θέση hi και βαίνει μειούμενος. Αν το άθροισμα από τα αριστερά είναι ίσο με το άθροισμα από τα δεξιά αυτή είναι μία πιθανή τιμή για το a - στην περίπτωση αυτή μεταβάλλουμε τον έναν από τους δύο δείκτες. Αν το άθροισμα από αριστερά είναι μεγαλύτερο από το άθροισμα από τα δεξιά τότε μεταβάλλουμε τον δείκτη j ενώ στην αντίθετη περίπτωση μεταβάλλουμε τον δείκτη i . Σε κάθε μία από τις δύο τελευταίες περιπτώσεις αυξάνουμε το αντίστοιχο άθροισμα. Ο Αλγόριθμος 4 κωδικοποιεί τη διαδικασία. ■

Περιγραφή

Triads

Αλγόριθμος 3 Υπολογισμός πλήθους τριάδων στοιχείων με άθροισμα μηδέν

Απαιτείται: ακέραιος n , πίνακας A με στοιχεία στις θέσεις από 1 ώς n

Επιστρέφεται: Πλήθος τριάδων στοιχείων με άθροισμα 0

```

1: function Triads(int A[], int n)
2:   Ταξινόμηση  $A$ ;
3:   count  $\leftarrow 0$ ;
4:   for  $k \leftarrow 1; k \leq n; k++ \text{ do}$ 
5:      $i \leftarrow k + 1$ ;
6:      $j \leftarrow n$ ;
7:     while  $i < j \text{ do}$ 
8:        $diff \leftarrow A[k] + A[i] + A[j]$ ;
9:       if  $diff = 0 \text{ then}$ 
10:         $count++$ ;
11:         $i++; j--$ ;
12:       else if  $diff < 0 \text{ then}$ 
13:         $i++$ ;
14:       else
15:         $j--$ ;
16:       end if
17:     end while
18:   end for
19:   return  $count$ ;
20: end function

```

Περιγραφή

Άσκηση 3. Δίνεται μια ακολουθία ακεραίων αριθμών οι οποίοι είναι αποθηκευμένοι σε έναν πίνακα A στις θέσεις από 1 ως n . Να εκπονηθεί αλγόριθμος ο οποίος να υπολογίζει το πλήθος των τριάδων των αριθμών που περιέχονται στον πίνακα και το άθροισμα τους είναι ίσο με μηδέν.

Λύση. Μία αφελης προσέγγιση είναι να πάρουμε το άθροισμα κάθε δυνατής τριάδας αριθμών και να το συγκρίνουμε με το μηδέν. Επειδή ο αριθμός των διακριτών τριάδων δίνεται από τον τύπο

$$\binom{n}{3} = \frac{n(n-1)(n-2)}{6},$$

μία τέτοια προσέγγιση οδηγεί σε αλγόριθμο πολυπλοκότητας $O(n^3)$.

Ταξινομώντας την ακολουθία και χρησιμοποιώντας την τεχνική των δύο δεικτών μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα. Συγκεκριμένα ο Αλγόριθμος 3 υπολογίζει το ζητούμενο πλήθος επιτυγχάνοντας πολυπλοκότητα $O(n^2)$. ■

Πολυπλοκότητα

04_ANAΔΡΟΜΗ

Factorial

Περιγραφή: «ΤΥΠΟΙ-ΙΔΙΟΤΗΤΕΣ/04_ANAΔΡΟΜΗ/Αναδρομή»

Αλγόριθμος 23 Αναδρομικός υπολογισμός του $n!$.

Απαιτείται: Ακέραιος $n \geq 0$.

Επιστρέφεται: n παραγοντικό.

```
1: function Factorial(int n)
2:   if n = 0 then
3:     return 1;
4:   else
5:     return n· Factorial(n - 1);
6:   end if
7: end function
```

Fibonacci

Αλγόριθμος 24 Υπολογισμός αριθμών Fibonacci.

Απαιτείται: Ακέραιος $n \geq 0$.

Επιστρέφεται: n -ιοστός αριθμός Fibonacci.

```
1: function Fibo(int n)
2:   if n ≤ 1 then
3:     return n;
4:   else
5:     return Fibo(n - 1) + Fibo(n - 2);
6:   end if
7: end function
```

Δεν είναι λίγες οι φορές όπου μία αναδρομική συνάρτηση καλεί τον εαυτό της παραπάνω από μία φορά με διαφορετική τιμή του ορίσματος της. Ένα τέτοιο παράδειγμα αποτελεί ο υπολογισμός της σειράς Fibonacci μέσω αναδρομικού αλγόριθμου. Η σειρά αυτή αποτελείται από τους αριθμούς

$$0, 1, 1, 2, 3, 5, 8, 13, \dots$$

Δηλαδή στη σειρά αυτή ο n -ιοστός αριθμός της σειράς, ονομαστικά $F(n)$, ισούται με το άθροισμα των δύο προηγουμένων αριθμών της σειράς με τους πρώτους δύο αριθμούς της σειράς να είναι το 0 και το 1. Ισοδύναμα,

$$F(n) = \begin{cases} F(n-1) + F(n-2), & n \geq 2, \\ 1, & n = 1, \\ 0, & n = 0. \end{cases} \quad (4.3)$$

Περιγραφή

Search_in_Group

Αλγόριθμος 25 Εύρεση στοιχείου με τη μέθοδο των ομάδων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi , στοιχείο αναζήτησης a .

Επιστρέφεται: Επιστροφή θέσης στον πίνακα A που βρίσκεται το μικρότερο στοιχείο που είναι μεγαλύτερο-ίσο του a .

```

1: function Search_in_Group(int  $a$ , int  $A[]$ , int  $lo$ , int  $hi$ )
2:   if  $lo > hi$  then
3:     return  $lo$ 
4:   else
5:      $n \leftarrow hi - lo + 1$ ;
6:      $q \leftarrow \lfloor \sqrt{n} \rfloor$ ;
7:      $r \leftarrow (hi - lo) \bmod q$ ;
8:      $j \leftarrow hi$ ;
9:      $q_1 \leftarrow r$ ;
10:    if  $A[lo] > a$  then
11:      return  $lo$ ;
12:    end if
13:    if  $A[hi] < a$  then
14:      return  $hi + 1$ ;
15:    end if
16:    while  $j \geq lo + q_1$  and  $A[j] > a$  do
17:       $j \leftarrow j - q_1$ ;
18:       $q_1 \leftarrow q$ ;
19:    end while
20:    if  $A[j] = a$  then
21:      return  $j$ ;
22:    end if
23:     $ghi \leftarrow \min(hi, j + q_1) - 1$ ;
24:     $glo \leftarrow j + 1$ ;
25:    return Search_in_Group( $a$ ,  $A$ ,  $glo$ ,  $ghi$ );
26:  end if
27: end function
```

Ενότητα 3.2.3: «03_ΠΙΝΑΚΕΣ/Search_in_Group»

Στην Ενότητα 3.2.3 περιγράφουμε την αναζήτηση ενός στοιχείου a σε μία ταξινομημένη σειρά n στοιχείων με τη χρήση ομάδων. Η ιδέα στην οποία βασίζεται ο συγκεκριμένος αλγόριθμος αφορά τον εντοπισμό μίας ομάδας με $\lfloor \sqrt{n} \rfloor$ στοιχεία στην οποία αναζητείται το στοιχείο a σειριακά (Αλγόριθμος 15).

Μία επέκταση της ιδέας αυτής είναι αντί να αναζητηθεί το στοιχείο στην ομάδα σειριακά, να θεωρήσουμε την ομάδα αυτή σαν την αρχική σειρά και να εφαρμόσουμε εκ' νέου αναζήτηση σε ομάδες. Η διαδικασία αυτή μπορεί να εφαρμοστεί αναδρομικά μέχρι να καταλήξουμε σε μέγεθος ομάδας μικρότερο-ίσο του 1. Ο Αλγόριθμος 25 υλοποιεί την ιδέα: όπως και στην περίπτωση του Αλγόριθμου 15, η ταξινομημένη σειρά των στοιχείων βρίσκεται στις θέσεις από lo ως hi ενός πίνακα A .

Η αναζήτηση σε ομάδες είναι ένα πολύ ευέλικτο σχήμα: σαν μέγεθος ομάδας μπορεί να χρησιμοποιηθεί οποιαδήποτε τιμή μεγαλύτερη-ίση της μονάδας και μικρότερη-ίση του πλήθους των στοιχείων της ταξινομημένης σειράς. Μία πολύ γνωστή μέθοδος προκύπτει αν θεωρήσουμε σαν μέγεθος ομάδας το ήμισυ του πλήθους αυτού: τότε ο αλγόριθμος που προκύπτει είναι γνωστός με το όνομα δυαδική αναζήτηση (*binary search*). Στον αλγόριθμο αυτό συγκρίνεται το στοιχείο a (προς-αναζήτηση στοιχείο) με το μεσαίο στοιχείο του πίνακα A . Αν είναι ίσο με αυτό τότε ο αλγόριθμος τερματίζει. Αν είναι μεγαλύτερο από αυτό τότε η διαδικασία επαναλαμβάνεται για το δεύτερο μισό της σειράς ενώ αν είναι μικρότερο από αυτό για το πρώτο μισό. Η διαδικασία είναι αναδρομική. Σε κάθε κλήση η σειρά χωρίζεται σε δύο ομάδες μισού μεγέθους και επιλέγεται η μία από τις δύο για να συνεχισθεί η αναζήτηση.

Περιγραφή

Binary_Search

Αλγόριθμος 25 Εύρεση στοιχείου με τη μέθοδο των ομάδων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi , στοιχείο αναζήτησης a .

Επιστρέφεται: Επιστροφή θέσης στον πίνακα A που βρίσκεται το μικρότερο στοιχείο που είναι μεναλύτερο-ίσο του a .

```
1: function Binary_Search (int  $a$ , int  $A[]$ , int  $lo$ , int  $hi$ )
2:   if  $lo > hi$  then
3:     return  $lo$ 
4:   else
5:      $n \leftarrow hi - lo + 1$ ;       $q \leftarrow \lfloor \frac{hi - lo}{2} \rfloor$ ;
6:      $r \leftarrow (hi - lo) \bmod 2$ ;
7:      $j \leftarrow hi$ ;
8:      $q_1 \leftarrow r$ ;
9:     if  $A[lo] > a$  then
10:       return  $lo$ ;
11:     end if
12:     if  $A[hi] < a$  then
13:       return  $hi + 1$ ;
14:     end if
15:     end if
16:     while  $j \geq lo + q_1$  and  $A[j] > a$  do
17:        $j \leftarrow j - q_1$ ;
18:        $q_1 \leftarrow q$ ;
19:     end while
20:     if  $A[j] = a$  then
21:       return  $j$ ;
22:     end if
23:      $ghi \leftarrow \min(hi, j + q_1) - 1$ ;
24:      $glo \leftarrow j + 1$ ;
25:     return Binary_Search ( $a, A, glo, ghi$ );
26:   end if
27: end function
```

Με αυτόν τον τρόπο επιτυγχάνεται διαμερισμός του συνόλου $\{lo, \dots, hi\}$ ως εξής:

$$\begin{aligned} \{lo, \dots, hi\} &= \{hi\} \\ &\cup \{hi - r\} \\ &\cup \{hi - r - 1, \dots, hi - r - \lfloor \frac{hi - lo}{2} \rfloor + 1\} \\ &\cup \{hi - r - \lfloor \frac{hi - lo}{2} \rfloor\} \\ &\cup \{hi - r - \lfloor \frac{hi - lo}{2} \rfloor - 1, \dots, hi - r - 2\lfloor \frac{hi - lo}{2} \rfloor + 1\} \\ &\cup \{hi - r - 2\lfloor \frac{hi - lo}{2} \rfloor\}. \end{aligned}$$

Η παραπάνω διαμέριση αποτελείται από τέσσερα - τρία αν $(hi - lo) \bmod 2 = r = 0$ - σύνολα που περιέχουν μία τιμή και από δύο σύνολα με περισσότερες τιμές. Η ένωση των μονότιμων συνόλων αποτελεί το πεδίο τιμών του δείκτη j . Δηλαδή, οι τιμές αυτές υποδεικνύουν τις θέσεις των στοιχείων του πίνακα A με τα οποία συγκρίνεται το στοιχείο a . Πιο συγκεκριμένα, οι τιμές που παίρνει ο δείκτης j όσο η συνθήκη του βρόχου της Γραμμής 16 είναι αληθής αποτελούν το σύνολο

$$\left\{ hi, hi - r, hi - r - \lfloor \frac{hi - lo}{2} \rfloor, hi - r - 2\lfloor \frac{hi - lo}{2} \rfloor \right\}$$

Κάνοντας πράξεις το παραπάνω σύνολο γράφεται ως

$$\left\{ hi, hi - r, lo + \lfloor \frac{hi - lo}{2} \rfloor, lo \right\}$$

Ο βρόχος δεν εκτελείται αν το στοιχείο a είναι μεγαλύτερο από το $A[hi]$ ή μικρότερο από το $A[lo]$ ενώ τερματίζει αν είναι μικρότερο-ίσο από κάποιο από τα στοιχεία που υποδεικνύουν οι τιμές του παραπάνω συνόλου - τα στοιχεία αυτά εξετάζονται με τη σειρά που εμφανίζονται οι τιμές στην απεικόνιση του παραπάνω συνόλου. Μετά τον τερματισμό, εξετάζεται αν το j δεικτοδοτεί στοιχείο ίσο με το a (Γραμμή 20). Αν δεν συμβαίνει αυτό τότε αν υπάρχει το στοιχείο a στον πίνακα A τότε θα πρέπει να αναζητηθεί είτε στην ομάδα που δεικτοδοτείται από το σύνολο

$$\{hi - r - 1, \dots, hi - r - \lfloor \frac{hi - lo}{2} \rfloor + 1\} = \{hi - r - 1, \dots, lo + \lfloor \frac{hi - lo}{2} \rfloor + 1\}$$

ή στην ομάδα

$$\{hi - r - \lfloor \frac{hi - lo}{2} \rfloor - 1, \dots, hi - r - 2\lfloor \frac{hi - lo}{2} \rfloor + 1\} = \{lo + \lfloor \frac{hi - lo}{2} \rfloor - 1, \dots, lo + 1\}$$

ανάλογα με το αν $a > lo + \lfloor \frac{hi - lo}{2} \rfloor$ ή $a < lo + \lfloor \frac{hi - lo}{2} \rfloor$. Οπότε ο αλγόριθμος καλείται αναδρομικά με παραμέτρους που δεικτοδοτούν το πρώτο (μικρότερο) και το τελευταίο (μεγαλύτερο) στοιχείο της αντίστοιχης ομάδας. Παρατηρούμε ότι αμφότερες οι ομάδες έχουν το ίδιο αριθμό στοιχείων ο οποίος είναι μικρότερος από το μισό της αρχικής, ήτοι

$$\lfloor \frac{n-1}{2} \rfloor - 1.$$

Περιγραφή

Dutch_Flag (Ολλανδική Σημαία)

Άλγοριθμος 26 Άλγοριθμος για το πρόβλημα της Ολλανδικής Σημαίας

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ως hi ($lo \leq hi$), κριτική τιμή p .

Επιστρέφεται: Αναδιάταξη των στοιχείων του A με τα στοιχεία που είναι μικρότερα του p (κόκκινα) να εμφανίζονται στις πρώτες θέσεις, τα στοιχεία που είναι μεγαλύτερα του p (μπλε) στις τελευταίες θέσεις και τα στοιχεία που είναι ίσα με p (λευκά) μετά από τα στοιχεία που είναι μικρότερα του p και πριν από τα στοιχεία που είναι μεγαλύτερα του p . Το πλήθος των κόκκινων, λευκών και μπλε στοιχείων επιστρέφονται στις μεταβλητές r, w, b , αντίστοιχα.

```
1: function Dutch_Flag(int p, int A[], int lo, int hi, int r, int w, int b)
2:   if lo > hi then
3:     r ← 0; w ← 0; b ← 0;
4:   else
5:     if A[hi] < p then
6:       Swap(A[lo], A[hi]);
7:       Dutch_Flag(p, A, lo + 1, hi, r, w, b);
8:       r++;
9:     else if A[hi] = p then
10:      Dutch_Flag(p, A, lo, hi - 1, r, w, b);
11:      Swap(A[lo + r + w], A[hi]);
12:      w++;
13:    else
14:      Dutch_Flag(p, A, lo, hi - 1, r, w, b);
15:      b++;
16:    end if
17:   end if
18: end function
```

Ενότητα 3.2.1: «03_ΠΙΝΑΚΕΣ/Pivot_Partition»

Σε πολλές περιπτώσεις ένας αναδρομικός αλγόριθμος μπορεί να περιέχει παραπάνω από μία αναδρομικές κλήσεις. Θα περιγράψουμε ένα τέτοιο αλγόριθμο για την επίλυση του προβλήματος της Ολλανδικής σημαίας το οποίο ορίζεται ως εξής. Θεωρούμε έναν πίνακα A του οποίου τα στοιχεία είναι χαρακτηρισμένα είτε ως κόκκινα, ή, ως άσπρα, ή ως μπλε. Θέλουμε να αναδιατάξουμε τα στοιχεία του πίνακα προκειμένου όλα τα κόκκινα στοιχεία να εμφανίζονται μαζί πριν από όλα τα άσπρα τα οποία πρέπει να εμφανίζονται μαζί πριν από όλα τα μπλε. Επίσης θέλουμε να γνωρίζουμε πόσα στοιχεία ανήκουν σε κάθε κατηγορία.

Μπορούμε να εξειδικεύσουμε το παραπάνω πρόβλημα περαιτέρω. Θεωρούμε ότι τα στοιχεία του πίνακα είναι ακέραιοι στις θέσεις από lo ως hi ($lo \leq hi$). Ο χαρακτηρισμός των στοιχείων μπορεί να γίνει με βάση μία «κριτική τιμή» p . Οι ακέραιοι που είναι μικρότεροι από αυτή την τιμή αποτελούν τα κόκκινα στοιχεία, οι ακέραιοι που είναι ίσοι με p τα λευκά στοιχεία και οι ακέραιοι που είναι μεγαλύτεροι από p τα μπλε στοιχεία. Ουσιαστικά το πρόβλημα αυτό αποτελεί μία γενίκευση του προβλήματος τοποθέτησης ενός στοιχείου με τιμή p στη σωστή θέση (Ενότητα 3.2.1): όμως στο πρόβλημα αυτό λύση μπορεί να αποτελεί και μία σειρά όπου τα στοιχεία με τιμή ίση με p δεν εμφανίζονται σε συνεχόμενες θέσεις.

Η επίλυση του προβλήματος βασίζεται σε ανταλλαγή των στοιχείων όταν αυτά είναι τοποθετημένα σε «λάθος» θέση· με το τρόπο αυτό μειώνεται το μέγεθος του στιγμιότυπου. Συγκεκριμένα, ξεκινώντας από τη θέση hi ελέγχουμε το χρώμα του στοιχείου της θέσης αυτής. Αν είναι μπλε το αγνοούμε και στη συνέχεια έχουμε να επιλύσουμε ένα στιγμιότυπο με ένα στοιχείο λιγότερο αφού το στοιχείο στη θέση lo είναι σωστά τοποθετημένο. Αν είναι λευκό δεν γνωρίζουμε που να το τοποθετήσουμε· μπορούμε προσωρινά να το αγνοήσουμε και να επιλύσουμε το υπόλοιπο πρόβλημα. Στη συνέχεια μπορούμε να το ανταλλάξουμε με το πρώτο από τα μπλε στοιχεία ή αν δεν υπάρχουν μπλε στοιχεία να το αφήσουμε στη θέση που βρίσκεται. Ο Αλγόριθμος 26 υλοποιεί την ιδέα. Με την ολοκλήρωση του αλγόριθμου, οι μεταβλητές r, w, b περιέχουν το πλήθος των στοιχείων που είναι μικρότερα, ίσα ή μεγαλύτερα του p . Παρατηρούμε ότι οι μεταβλητές αυτές αρχικοποιούνται στην τιμή μηδέν στο μη αναδρομικό τμήμα του αλγόριθμου.

Περιγραφή

k-Element (Διατεταγμένα στατιστικά)

Αλγόριθμος 27 k-ιοστό μικρότερο στοιχείο

Απαιτείται: πίνακας A με στοιχεία στις θέσεις από lo έως hi , ακέραιος $k \in \{1, \dots, hi - lo + 1\}$.

Επιστρέφεται: k -ιοστό μικρότερο στοιχείο του πίνακα A .

```

1: function k-Element(int A[], int lo, int hi, int k)
2:   if lo = hi then
3:     return lo
4:   end if
5:   n  $\leftarrow$  hi - lo + 1;
6:   pvt  $\leftarrow$  Rand(n) + lo;
7:   j  $\leftarrow$  Pivot_Partition(A, lo, hi, pvt);
8:   i  $\leftarrow$  j - lo + 1;
9:   if i = k then
10:    return A[j];
11:   else if i > k then
12:    return k-Element(A, lo, j - 1, k);
13:   else
14:    return k-Element(A, j + 1, hi, k - i);
15:   end if
16: end function

```

Αλγόριθμος 10: «03_ΠΙΝΑΚΕΣ/Pivot_Partition»

Iδιότητα 1: «ΤΥΠΟΙ-ΙΔΙΟΤΗΤΕΣ/03_ΠΙΝΑΚΕΣ/Ιδιότητα της σωστής θέσης ενός στοιχείου στον Πίνακα»

Ένα από τα πρώτα προβλήματα που περιγράφαμε (Κεφάλαιο 1) ήταν αυτό της εύρεσης του μικρότερου σε μία αταξινόμητη σειρά ακεραίων. Θα γενικεύσουμε το πρόβλημα αυτό στο πρόβλημα εύρεσης του k -ιοστού μικρότερου στοιχείου σε μία αταξινόμητη σειρά n στοιχείων, όπου $k \leq n$. Θα θεωρήσουμε ότι η σειρά των ακεραίων βρίσκεται αποθηκευμένη σε έναν πίνακα A στις θέσεις από lo έως hi .

Ο Αλγόριθμος 27 επιλύει το παραπάνω πρόβλημα καλώντας τη συνάρτηση Pivot_Partition (Αλγόριθμος 10) η οποία, έχοντας αναδιατάξει τα στοιχεία του πίνακα A , επιστρέφει τη σωστή θέση, έστω j , (Ιδιότητα 1) που έχει τοποθετήσει το στοιχείο που βρισκόταν ποιν σε μία τυγαία θέση nut του πίνακα A . Η επιλογή της nut νίνεται από τη συνάρτηση Rand η οποία επιστρέφει έναν «τυχαίο» ακέραιο από το σύνολο $\{lo, \dots, hi\}$.

Έστω ότι το στοιχείο $A[j]$ - που πλέον έχει τοποθετηθεί στη σωστή θέση (Γραμμή 5) - είναι το i -οστό μικρότερο στοιχείο του πίνακα: παρατηρούμε ότι η τιμή i υπολογίζεται από τη σχέση $i = j - lo + 1$. Αν $i = k$ τότε το στοιχείο αυτό είναι το ζητούμενο και ο υπολογισμός ολοκληρώνεται. Διαφορετικά, αν $i > k$ τότε η διαδικασία επαναλαμβάνεται θέτοντας $hi = j - 1$, ενώ αν $i < k$ θέτοντας $lo = j + 1$ και αναζητώντας το $k - i$ μικρότερο στοιχείο. Η διαδικασία επαναλαμβάνεται αναδρομικά μέχρι να βρεθεί μία θέση j για την οποία $i = k$. Αυτό, αν δεν προκύψει νωρίτερα, θα συμβεί αναγκαστικά όταν η σειρά αποτελείται από ένα στοιχείο ($lo = hi$). Επομένως η διαδικασία τερματίζει.

Περιγραφή

Palindrome

Αλγόριθμος 2 Αποφασίζεται αν μία σειρά είναι παλίνδρομη

Απαιτείται: Ακέραιοι lo, hi ($lo \leq hi$) πίνακας A που περιέχει στις θέσεις από lo ως hi στοιχεία από το σύνολο $\{0, 1\}$

Επιστρέφεται: True αν ο πίνακας A αποτελεί μία παλίνδρομη σειρά, False εδιαφρετικά.

```

1: function Palindrome(int A[], int lo, int hi )
2:   if lo < hi then
3:     if A[lo] = A[hi] then
4:       return Palindrome(A, lo + 1, hi - 1);
5:     else
6:       return False;
7:     end if
8:   else
9:     return True;
10:  end if
11: end function

```

Άσκηση 2. Μία συμβολοσειρά αποτελούμενη από ψηφία του συνόλου $\{0, 1\}$ ονομάζεται παλίνδρομη αν ταυτίζεται με την αντίστροφή της. Να εκπονήσετε αλγόριθμο ο οποίος να δέχεται σαν είσοδο μία σειρά από αριθμούς και να αποφασίζει αν η συμβολοσειρά αυτή είναι παλίνδρομη. Να διατυπώσετε την αναδρομική σχέση που περιγράφει τον αριθμό των ΣΥΒ.

Λύση. Ο Αλγόριθμος 2 υπολογίζει το ζητούμενο. Αν τα στοιχεία βρίσκονται στις n πρώτες θέσεις του πίνακα A τότε ο Αλγόριθμος 2 καλείται ως

$$decide \leftarrow \text{Palindrome}(A, 1, n);$$

Παρατηρούμε ότι σε κάθε κλήση της συνάρτησης εκτελείται σταθερός αριθμός ΣΥΒ. Στην αναδρομική κλήση που εκτελείται στη Γραμμή 5 του αλγόριθμου, εξετάζεται ένα στιγμιότυπο με

$$hi - 1 - (lo + 1) + 1 = hi - lo - 1$$

στοιχεία. Το πλήθος των στοιχείων του αρχικού στιγμιότυπου είναι $n = hi - lo + 1$ και επομένως στο στιγμιότυπο που εξετάζεται στην αναδρομική κλήση έχει $n - 2$ στοιχεία.

Αρα ο αριθμός των ΣΥΒ περιγραφεται από την αναδρομικη σχεση

$$T(n) \leq \begin{cases} T(n-2) + \Theta(1), & n \geq 2, \\ \Theta(1), & \text{διαφορετικά.} \end{cases}$$

Περιγραφή

Πολυπλοκότητα

Unimodal

Άλγοριθμος 3 Υπολογισμός σημείου καμπής σε μονότροπη σειρά

Απαιτείται: ακέραιοι lo, hi ($lo \leq hi$) πίνακας A με στοιχεία που σχηματίζουν μονότροπη σειρά

Επιστρέφεται: σημείο καμπής

```

1: function Unimodal(int A[], int lo, int hi)
2:   if lo = hi then
3:     return A[lo];
4:   end if
5:   j  $\leftarrow \lfloor \frac{lo+hi}{2} \rfloor$ ;
6:   if A[j] < A[j + 1] then
7:     return Unimodal(A, j + 1, hi);
8:   else
9:     return Unimodal(A, lo, j);
10:  end if
11: end function

```

Άσκηση 3. Μία σειρά από διαφορετικούς ακεραίους a_1, \dots, a_n ονομάζεται μονότροπη (unimodal) αν υπάρχει ένα στοιχείο a_k , $1 \leq k \leq n$, για το οποίο ισχύει

$$a_1 < a_2 < \dots < a_{k-1} < a_k > a_{k+1} > \dots > a_{n-1} > a_n.$$

Από τον παραπάνω ορισμό προκύπτει ότι το σημείο καμπής μπορεί να είναι το πρώτο ή το τελευταίο στοιχείο της σειράς. Το στοιχείο a_k ονομάζεται σημείο καμπής.

Δίνεται μία μονότροπη σειρά ακεραίων αποθηκευμένη στις θέσεις $1, \dots, n$ του πίνακα A . Να εκπονήσετε αναδρομικό αλγόριθμο που να επιστρέφει τη θέση του σημείου καμπής και να διατυπώσετε την αναδρομική σχέση που περιγράφει τον αριθμό των ΣΥΒ που εκτελούνται.

Λύση. Θα εξετάσουμε το στοιχείο σε μία τυχαία θέση j ($1 \leq j \leq n$) του πίνακα A . Αν $A[j] < A[j + 1]$ τότε το στοιχείο καμπής βρίσκεται από τη θέση $j + 1$ και μετά. Διαφορετικά, το ζητούμενο στοιχείο βρίσκεται από τη θέση j και πριν. Ο Αλγόριθμος 3 υλοποιεί την ιδέα.

Σε κάθε κλήση της συνάρτησης Unimodal εκτελείται ένας σταθερός αριθμός ΣΥΒ και μετά είτε έχουμε κλήση της συνάρτησης για την επόμενη ενός στιγμιότυπου με

$$hi - (\lfloor \frac{lo + hi}{2} \rfloor + 1) + 1 \quad (1)$$

στοιχεία ή ενός στιγμιότυπου με

$$\lfloor \frac{lo + hi}{2} \rfloor - lo + 1 \quad (2)$$

στοιχεία. Γνωρίζουμε ότι

$$hi - lo + 1 = n \Rightarrow hi = n - 1 + lo \Rightarrow hi + lo = n - 1 + 2lo \quad (3)$$

Συνδυάζοντας την (3) με την (2) έχουμε

$$\lfloor \frac{lo + hi}{2} \rfloor - lo + 1 = \lfloor \frac{n - 1 + 2lo}{2} \rfloor - lo + 1 \quad (4)$$

$$= \lfloor \frac{n - 1}{2} \rfloor + 1 = \lfloor \frac{n + 1}{2} \rfloor = \lceil \frac{n}{2} \rceil. \quad (5)$$

Συνεπώς η (1) συνεπάγεται

$$hi - (\lfloor \frac{lo + hi}{2} \rfloor + 1) + 1 = \lceil \frac{n}{2} \rceil \quad (6)$$

Περιγραφή

Από τις (5), (6) έχουμε ότι για τη συνάρτηση που δίνει τον αριθμό των ΣΥΒ ισχύει

$$T(n) \leq \begin{cases} T(\lceil \frac{n}{2} \rceil) + \Theta(1), & n \geq 2, \\ \Theta(1), & \text{διαφορετικά.} \end{cases} \quad (7)$$

■

Πολυπλοκότητα

QuickSort (Ταχυταξινόμηση)**Αλγόριθμος 44 Ταχυταξινόμηση**

Απαιτείται: πίνακας A με στοιχεία στις θέσεις από lo έως hi

Επιστρέφεται: Ταξινομημένος πίνακας A

```

1: function Quicksort(int A[], int lo, int hi)
2:   if lb < ub then
3:     i ← Pivot_Partition(A, lo, hi, hi);
4:     Quicksort(A, lb, i - 1);
5:     Quicksort(A, i + 1, ub);
6:   end if
7: end function

```

Αλγόριθμος 10: «03_ΠΙΝΑΚΕΣ/Pivot_Partition»

Στην Ενότητα 4.2.4 παρουσιάσαμε τον Αλγόριθμο 10 οποίος δεδομένης μίας θέσης pvt ($lo \leq pvt \leq hi$) τοποθετεί το στοιχείο $A[pvt]$ στη σωστή θέση - στη θέση που θα πρέπει να βρίσκεται όταν ο πίνακας ταξινομηθεί. Ο αλγόριθμος επιστρέφει τη θέση αυτή.

Μπορούμε να εκμεταλλευτούμε την παραπάνω αλγορίθμική διαδικασία ως εξής. Αφού τοποθετηθεί το στοιχείο στη σωστή θέση, έστω k , έχουμε αταξινόμητα δύο μέρη (τμήματα) του πίνακα: αυτό που περιέχει τα στοιχεία $A[lo], \dots, A[k-1]$ και $A[k+1], \dots, A[hi]$.

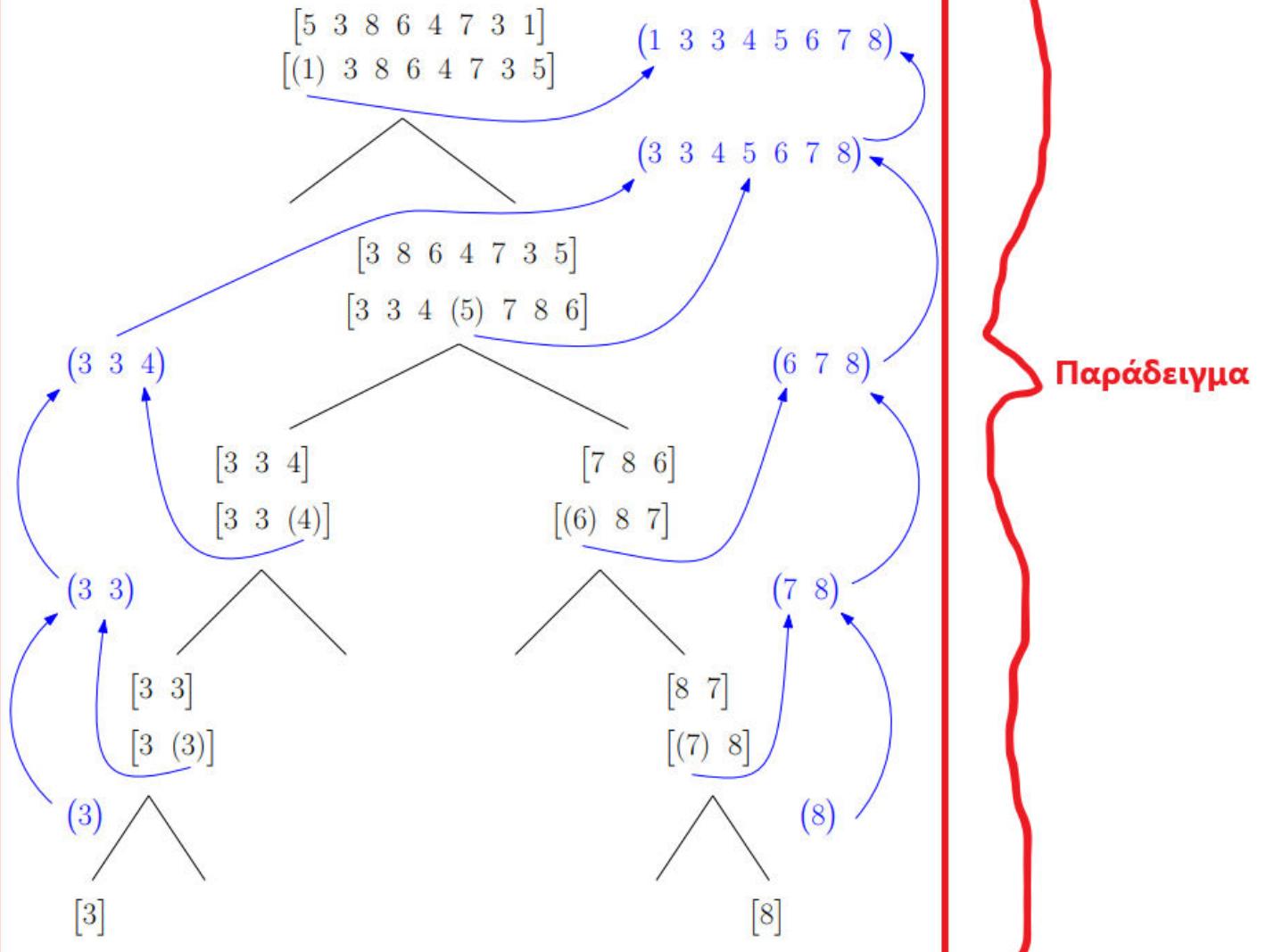
Προφανώς αν με κάποιο τρόπο μπορέσουμε να ταξινομήσουμε τα στοιχεία των δύο αυτών μερών θα έχουμε την αρχική ακολουθία ταξινομημένη. Πως όμως θα ταξινομηθούν τα δύο αυτά μέρη; Η ιδέα είναι να χρησιμοποιήσουμε αναδρομικά την ίδια διαδικασία για καθένα από τα δύο μέρη. Σε κάθε επίπεδο αναδρομής παρατηρούμε ότι το πλήθος των στοιχείων είναι κατά ένα μεγαλύτερο από το άθροισμα του πλήθους των στοιχείων των δύο μερών (αφού ένα από τα στοιχεία έχει ήδη μπει στη σωστή θέση από τον Αλγόριθμο 10). Επομένως σε κάθε επίπεδο έχουμε να λύσουμε δύο στιγμιότυπα προβλήματος ταξινόμησης (υποπροβλήματα) με συνολικά μικρότερο αριθμό στοιχείων από το παρόν. Η βάση της αναδρομής προκύπτει όταν έχουμε να επιλύσουμε στιγμιότυπο ταξινόμησης με ένα (ή κανένα) στοιχείο. Στην περίπτωση αυτή δεν εκτελείται καμία εντολή και ο έλεγχος επιστρέφει στο προηγούμενο επίπεδο της αναδρομικής διαδικασίας. Η κωδικοποίηση αποτυπώνεται στον Αλγόριθμο 44. Το στοιχείο που τοποθετείται στη σωστή θέση από τη διαδικασία Pivot_Partition (Αλγόριθμος 10) είναι αυτό που αρχικώς βρίσκεται στη μεγαλύτερη θέση - θέση hi .

Περιγραφή

Παράδειγμα 18. Για να ταξινομήσουμε τα στοιχεία του πίνακα A που βρίσκονται στις θέσεις από 1 ως 8, ήτοι

$$A = [5 \ 3 \ 8 \ 6 \ 4 \ 7 \ 3 \ 1],$$

με τον Αλγόριθμο 44 καλούμε τη συνάρτηση QuickSort($A, 1, 8$). Η διαδικασία υπολογισμού απεικονίζεται στο Σχήμα 7.1. Το σχήμα αναπαριστά το ιδεατό δένδρο (virtual tree) της αναδρομής. Σε κάθε κόμβο του δένδρου παρουσιάζεται η εικόνα του πίνακα



Σχήμα 7.1: Αλγόριθμος ταχυταξινόμησης, Παράδειγμα 18

Α πριν και μετά την εκτέλεση της διαδικασίας τοποθέτησης ενός στοιχείου στη σωστή θέση· το στοιχείο αυτό απεικονίζεται σε παρένθεση. Οι κλάδοι που δεν καταλήγουν σε κάποιο κόμβο αντιστοιχούν σε στιγμιότυπα όπου δεν υπάρχει κανένα στοιχείο προς ταξινόμηση (περίπτωση κατά την οποία $lo > hi$). Στην περίπτωση αυτή όπως και στην περίπτωση που το στιγμιότυπο έχει μόνο ένα στοιχείο ο αντίστοιχος κόμβος είναι τερματικός και δεν εκτελείται καμία πράξη - εκτός από την πράξη της σύγκρισης των παραμέτρων lo, hi . Η σειρά των αριθμών σε παρενθέσεις¹ είναι η εικόνα του διανύσματος A - κατά το τμήμα που αφορά το παρόν στιγμιότυπο - που επιστρέφει η παρούσα αναδρομική κλήση κατά την ολοκλήρωση της.

Αλγόριθμος 10: «03_ΠΙΝΑΚΕΣ/Pivot_Partition»

Υπολογιστικά η συμπεριφορά της ταχυταξινόμησης δεν εξαρτάται μόνο από τον αριθμό των στοιχείων αλλά και από τον τρόπο που είναι αυτά αρχικώς διατεταγμένα στον πίνακα A . Για να υπολογίσουμε την πολυπλοκότητα της ταχυταξινόμησης αρχικώς παρατηρούμε ότι η διαδικασία Pivot_Partition (Αλγόριθμος 10) είναι τάξης $\Theta(n)$. Εφόσον ένα στοιχείο τοποθετείται από τον αλγόριθμο αυτό στη σωστή θέση παραμένουν προς ταξινόμηση $n - 1$ στοιχεία εκ' των οποίων τα q ($0 \leq q \leq n - 1$) συνθέτουν το στιγμιότυπο της Γραμμής 5 και τα $n - q - 1$ της Γραμμής 6. Επομένως, ο αριθμός των ΣΥΒ της χειρότερης περίπτωσης, ονομαστικά $T(n)$ ικανοποιεί τη σχέση

$$T(n) = \max\{T(q) + T(n - q - 1) : 0 \leq q \leq n - 1\} + \Theta(n). \quad (7.1)$$

Εικάζουμε ότι $T(n)$ είναι $O(n^2)$. Εφαρμόζοντας την εικασία αυτή στην παραπάνω σχέση, στους όρους $T(q), T(n - q - 1)$, έχουμε, για κάποιο $c > 0$,

$$\begin{aligned} T(n) &\leq \max\{cq^2 + c(n - q - 1)^2 : 0 \leq q \leq n - 1\} + \Theta(n) \\ &= c \max\{q^2 + (n - q - 1)^2 : 0 \leq q \leq n - 1\} + \Theta(n) \end{aligned}$$

Η συνάρτηση

$$f(q) = q^2 + (n - q - 1)^2 \quad (7.2)$$

είναι κυρτή· η δεύτερη παράγωγος της είναι ίση με μία θετική σταθερά. Επομένως παίρνει τη μεγαλύτερη τιμή της στα άκρα του πεδίου ορισμού της, ήτοι για $q = 0$ ή $q = n - 1$. Για τις τιμές αυτές, έχουμε $f(0) = f(n - 1) = (n - 1)^2$. Αντικαθιστώντας στην προηγούμενη σχέση έχουμε

$$T(n) \leq c(n - 1)^2 + \Theta(n) = cn^2 - c(2n - 1) + \Theta(n) \leq cn^2$$

με την τελευταία ανισότητα να προκύπτει επιλέγοντας c αρκετά μεγάλο ώστε ο όρος $c(2n - 1)$ να κυριαρχεί πάνω στον όρο $\Theta(n)$.

Η παραπάνω ανάλυση δείχνει ότι η χειρότερη περίπτωση για τον αλγόριθμο της ταχυταξινόμησης προκύπτει όταν το ένα από τα δύο στιγμιότυπα είναι κενό· η αντίστοιχη σειρά δεν περιέχει καθόλου στοιχεία. Είναι εύκολο να δει κανείς ότι αυτό προκύπτει όταν η σειρά περιέχει διαφορετικά στοιχεία και είναι εξ' αρχής ταξινομημένη ή ταξινομημένη σε φθίνουσα διάταξη. Όμως σε αυτή την περίπτωση σε κάθε κλήση μετά την διαδικασία Pivot_Partition μένει να ταξινομήσουμε μία σειρά με ένα στοιχείο λιγότερο και άρα ο αριθμός των ΣΥΒ εκφράζεται από την αναδρομική σχέση

$$T(n) = T(n - 1) + \Theta(n).$$

Με τη μέθοδο της διαδοχικής αντικατάστασης δεν είναι δύσκολο να δείξει κάποιος ότι $T(n) = \Theta(n^2)$.

Οπως είδαμε προηγουμένως η χειρότερη περίπτωση στην ταχυταξινόμηση προκύπτει όταν το δένδρο αναδρομής είναι μη-ισορροπημένο. Διαισθητικά η καλύτερη περίπτωση πρέπει να προκύπτει όταν το δένδρο είναι πλήρως ισορροπημένο. Δηλαδή, όταν το στοιχείο που μπαίνει κάθε φορά στη σωστή θέση (από τη διαδικασία Pivot_Partition) διαμορφώνει δυο στιγμιότυπα με (περίπου) τον ίδιο αριθμό στοιχείων: στο ένα στιγμιότυπο που πρέπει να λυθεί αναδρομικά έχουμε $\lfloor \frac{n-1}{2} \rfloor$ στοιχεία ενώ στο άλλο $\lceil \frac{n-1}{2} \rceil$. Αν $T(n)$ είναι η συνάρτηση που δίνει τον αριθμό των ΣΥΒ στην περίπτωση αυτή, ισχύει ότι

$$T(n) = \begin{cases} 1, & n \leq 1, \\ T(\lfloor \frac{n-1}{2} \rfloor) + T(\lceil \frac{n-1}{2} \rceil) + \Theta(n), & n \geq 2. \end{cases} \quad (7.3)$$

Παρατηρούμε ότι $T(n) \geq \bar{T}(n)$ όπου

$$\bar{T}(n) = \begin{cases} 2\bar{T}(\lfloor \frac{n-1}{2} \rfloor) + \Theta(n), & n \geq 2, \\ 1, & n \leq 1. \end{cases}$$

Θα δείξουμε ότι $\bar{T}(n) = \Omega(n \lg n)$. Χρησιμοποιώντας τη μέθοδο της επαγωγής, Θα υποθέσουμε ότι αυτό ισχύει για $\bar{T}(\lfloor \frac{n-1}{2} \rfloor)$. Δηλαδή, έστω ότι υπάρχει $a > 0$ τέτοιο ώστε

$$\begin{aligned} \bar{T}(\lfloor \frac{n-1}{2} \rfloor) &\geq a \lfloor \frac{n-1}{2} \rfloor \lg \lfloor \frac{n-1}{2} \rfloor \Rightarrow \\ \bar{T}(n) &\geq 2a \lfloor \frac{n-1}{2} \rfloor \lg \lfloor \frac{n-1}{2} \rfloor + cn \\ &\geq 2a \left(\frac{n-1}{2} - 1 \right) \lg \left(\frac{n-1}{2} - 1 \right) + cn \\ &= 2a \frac{n-3}{2} \lg \frac{n-3}{2} + cn \Rightarrow \\ \bar{T}(\lfloor \frac{n-1}{2} \rfloor) &\geq a(n-3)(\lg(n-3) - 1) + cn. \end{aligned} \quad (7.4)$$

Παρατηρούμε ότι για $n \geq 4$, ισχύει²

$$\lg(n-3) - 1 \geq \lg n - 3. \quad (7.5)$$

Από (7.4), (7.5) έχουμε

$$\begin{aligned} T(n) &\geq a(n-3)(\lg n - 3) + cn = an \lg n - 3a \lg n - 3an + 9a + cn \\ &\geq an \lg n - 3a \lg n - 3an + cn \\ &\geq an \lg n - 3an - 3an + cn = an \lg n - (6a - c)n \Rightarrow \\ \bar{T}(n) &\geq an \lg n, \end{aligned}$$

$$\text{για } 6a - c \leq 0 \Rightarrow a \leq \frac{c}{6}.$$

Αρα $\bar{T}(n) = \Omega(n \lg n)$ και συνεπώς $T(n) = \Omega(n \lg n)$

$$\lg(n-3) - 1 \geq \lg n - 3 \Rightarrow \lg(n-3) \geq \lg n - 2 \Rightarrow \lg(n-3) \geq \lg \frac{n}{4} \Rightarrow n-3 \geq n/4 \Rightarrow n \geq 4.$$

Πολυπλοκότητα

Πόρισμα 6. Ο αριθμός των ΣΥΒ που εκτελεί ο αλγόριθμος της ταχυταξινόμησης, για οποιοδήποτε στιγμιότυπο μεγέθους n , είναι τάξης $O(n^2)$ και $\Omega(n \lg n)$.

Πολυπλοκότητα

Binary_Search (Δυαδική Αναζήτηση)

• Η ευάλωτη αρχή δέονται συνδίγεται να είναι η λέξη της ευάλωτης σειράς αριθμών, οπός οτι παραδεγματικός

Anατανάσι: A, lo, hi, στα lo ≤ hi, και d

Ενισχύεται: Έδειν τα γνωστά στοιχεία, αλλώς γνώση των τιμών (n.x.-1) (A, lo, hi (lo ≤ hi), d)

function BSearch (int a, int A[], int lo, int hi)

{

if (lo = hi) then τετριτικό (trivial) ουγκόστιο

if A[lo] = a then

return lo;

else return 0;

-else

mid ← $\left\lfloor \frac{lo+hi}{2} \right\rfloor$;

if a < A[mid] then

return BSearch(a, A, lo, mid);

else

return BSearch(a, A, mid+1, hi);

-end_if

-end_if

end_function

n = hi - lo + 1;

mid ← $\left\lceil \frac{n}{2} \right\rceil$

mid ← $\left\lceil \frac{hi+lo}{2} \right\rceil$

Δε συναντήσεται
κατία δυν., γιατί
διαφέρει είτε το
ένα είτε το άλλο
BSearch

• Απαγορεύεται να
ταυτίσει την έναν
και τον άλλον ουγκόστιον

MergeSort (Συγχωνευτική Ταξινόμηση)

Αλγόριθμος 45 Συγχωνευτική Ταξινόμηση

Απαιτείται: πίνακας A με στοιχεία στις θέσεις από lo έως hi

Επιστρέφεται: Ταξινομημένος πίνακας A

```
1: function MergeSort(int A[], int lo, int hi)
2:   if lo < hi then
3:     mid ← ⌊ $\frac{ub+lb}{2}$ ⌋;
4:     Mergesort(A, lo, mid);
5:     Mergesort(A, mid + 1, hi);
6:     for i ← lo; i ≤ mid; i ++ do
7:       A1[i] ← A[i];
8:     end for
9:     for i ← mid + 1; i ≤ hi; i ++ do
10:      A2[i] ← A[i];
11:    end for
12:    Merge(A1, lo, mid, A2, mid + 1, hi, A, lo, hi);
13:  end if
14: end function
```

Ενότητα 3.2.1: «ΤΥΠΟΙ-ΙΔΙΟΤΗΤΕΣ/03_ΠΙΝΑΚΕΣ/Ταξινόμηση»

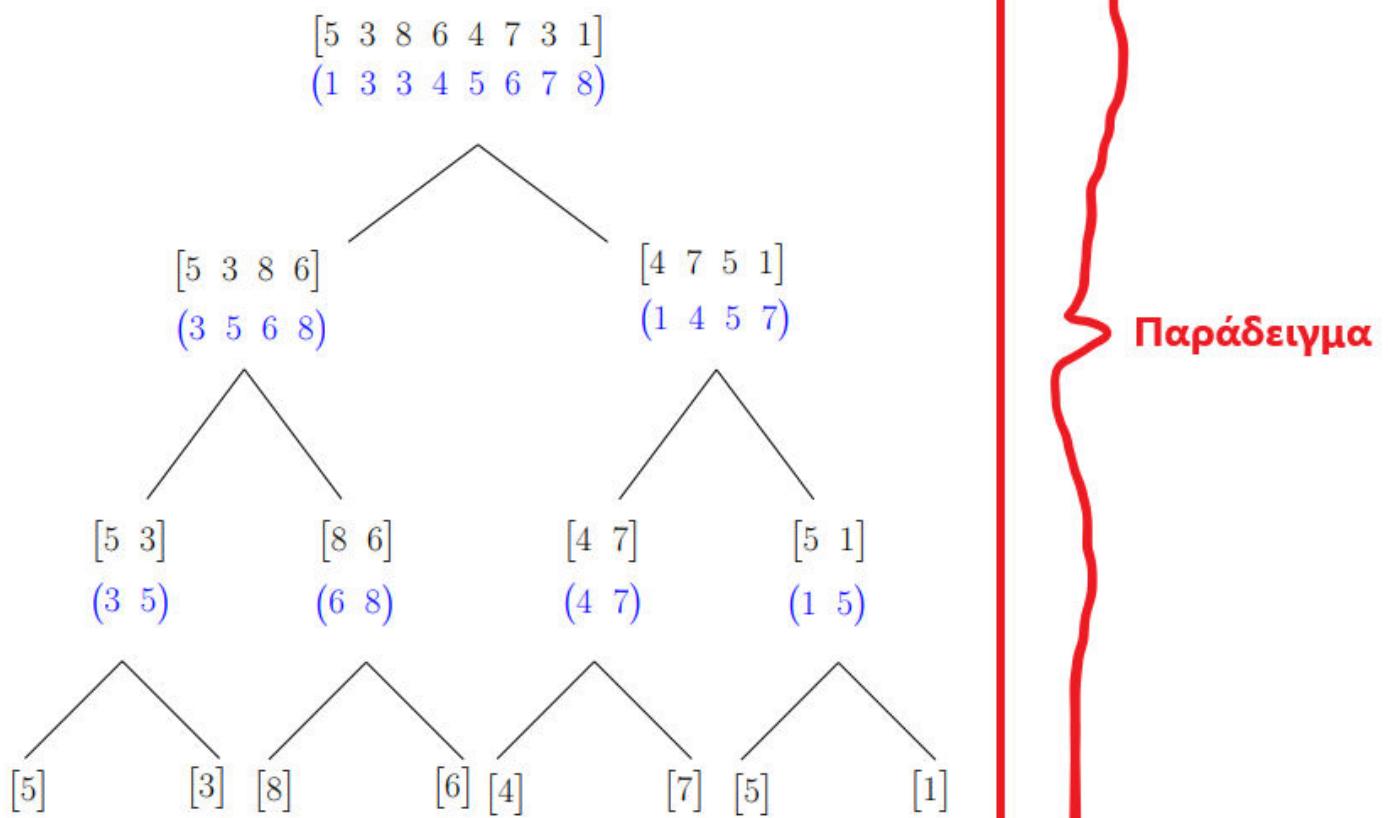
Αλγόριθμος 12: «03_ΠΙΝΑΚΕΣ/Merge»

Μία από τις πλέον χαρακτηριστικές περιπτώσεις αλγορίθμου που εκφράζει το πνεύμα της μεθοδολογίας «Διαίρει και Κυρίευε» είναι η συγχωνευτική ταξινόμηση (*merge sort*). Ο αλγόριθμος επιλύει το πρόβλημα της ταξινόμησης (Ενότητα 3.2.1) με τον εξής τρόπο. Αποσυνθέτει την αταξινόμητη σειρά στα δύο - επομένως από ένα στιγμιότυπο προβλήματος ταξινόμησης παράγονται δύο στιγμιότυπα με μισά στοιχεία στο καθένα (φάση «Διαίρειν»). Τα στιγμιότυπα αυτά αναδρομικά αποσυντίθενται με τον ίδιο τρόπο (φάση «Κυρίευε») μέχρι να προκύψουν στιγμιότυπα με ένα (ή κανένα) στοιχείο - τέτοια στιγμιότυπα είναι εξ' ορισμού ταξινομημένα. Όταν σε μία κλήση τα δύο στιγμιότυπα που αποτελούνται από τα μισά στοιχεία είναι ταξινομημένα τότε μέσω της διαδικασίας της συγχώνευσης (Αλγόριθμος 12) μπορεί να επιλυθεί το πρόβλημα της ταξινόμησης στο παρόν στιγμιότυπο (φάση «Συνδύασε»).

Ο Αλγόριθμος 45 υλοποιεί την ιδέα με τη χρήση του Αλγόριθμου 12 για τη λειτουργία της συγχώνευσης. Οι φάσεις «Διαίρειν» και «Κυρίευε» είναι σχετικά απλές και υλοποιούνται όπως περιγράφηκαν πρωτότερα. Η φάση «Συνδύασε» περιγράφεται αναλυτικότερα ως εξής. Μόλις επιλυθούν τα δύο μικρότερα στιγμιότυπα, δηλαδή ταξινομηθούν οι σειρές $A[lo], \dots, A[mid]$ και $A[mid + 1], \dots, A[hi]$ αντιγράφονται στους τοπικούς πίνακες $A1$ και $A2$ και στη συνέχεια με τη διαδικασία *merge* (Αλγόριθμος 12) τα στοιχεία τους συγχωνεύονται στον τοπικό πίνακα $A3$. Τα στοιχεία του πίνακα αυτού είναι τα στοιχεία του πίνακα A ταξινομημένα κατά αύξουσα σειρά: όμως τα στοιχεία αυτά καταλαμβάνουν τις θέσεις 1 ως $hi - lo + 1$ του πίνακα $A3$. Στη συνέχεια, τα στοιχεία αυτά αντιγράφονται στον πίνακα A στις θέσεις από lo ως hi .

Περιγραφή

Το δένδρο αναδρομής του Αλγόριθμου 45 για το στιγμιότυπο του Παραδείγματος 18 απεικονίζεται στο Σχήμα 7.2. Σε κάθε κόμβο του δένδρου σε τετράγωνες παρενθέσεις παρουσιάζεται η σειρά που πρέπει να ταξινομηθεί ενώ σε παρενθέσεις το ταξινομημένο διάνυσμα όπως επιστρέφεται από την συνάρτηση Merge (Αλγόριθμος 12)



Ενότητα 3.2.2: «03_ΠΙΝΑΚΕΣ/Merge»

Θεώρημα 2: «ΤΥΠΟΙ-ΙΔΙΟΤΗΤΕΣ/05_ΑΝΑΔΡΟΜΙΚΕΣ ΣΧΕΣΕΙΣ/Θεώρημα Κυριαρχικού Όρου»

Όπως είναι εμφανές (και από το Σχήμα 7.2), ο αριθμός των ΣΥΒ που εκτελεί ο Αλγόριθμος 45 εξαρτάται μόνο από τον αριθμό των προς ταξινόμηση στοιχείων και όχι από τη διάταξη τους στην αταξινόμητη σειρά. Συγκεκριμένα, αν ο αριθμός των στοιχείων σε κάποια κλήση είναι $n = hi - lo + 1$, ζητείται η επίλυση ενός στιγμιότυπου ταξινόμησης με

$$\lfloor \frac{hi + lo}{2} \rfloor - lo + 1 \quad (7.6)$$

στοιχεία (Γραμμή 4) και ενός στιγμιότυπου με

$$hi - (\lfloor \frac{hi + lo}{2} \rfloor + 1) + 1 \quad (7.7)$$

στοιχεία (Γραμμή 5). Στη συνέχεια και αφού επιλυθούν τα δύο αυτά στιγμιότυπα (δηλαδή ταξινομηθούν οι αντίστοιχες σειρές), εκτελούνται Θ(n) ΣΥΒ για τη φάση της συγχώνευσης (Ενότητα 3.2.2).

Θα δείξουμε ότι η (7.6) είναι ίση με $\lceil \frac{n}{2} \rceil$. Εξ' ορισμού,

$$n - 1 = hi - lo \Rightarrow n - 1 + 2lo = hi + lo.$$

Διαιρώντας με 2 τα δύο μέλη, θεωρώντας το πάτωμα τους, μεταφέροντας τον όρο lo στο δεξί μέλος και προσθέτοντας σε αμφότερα το μέλη τη μονάδα, παίρνουμε την (7.6) ως

$$\lfloor \frac{hi + lo}{2} \rfloor - lo + 1 = \lfloor \frac{n - 1}{2} \rfloor + 1 = \lfloor \frac{n + 1}{2} \rfloor = \lceil \frac{n}{2} \rceil. \quad (7.8)$$

Συνεπώς για την (7.7) ισχύει ότι

$$hi - (\lfloor \frac{hi + lo}{2} \rfloor + 1) + 1 = \lfloor \frac{n}{2} \rfloor \quad (7.9)$$

Από την παραπάνω ανάλυση προκύπτει ότι ο αριθμός των ΣΥΒ του Αλγόριθμου 45 ικανοποιεί την αναδρομική σχέση

$$T(n) = \begin{cases} 1, & n \leq 1, \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n), & n \geq 2. \end{cases}$$

Από το Θεώρημα 2 έχουμε ότι $T(n) = \Theta(n \lg n)$.

Πολυπλοκότητα

MatrixMult (Πολλαπλασιασμός Πινάκων)

Άλγοριθμος 46 Πολλαπλασιασμός πινάκων με τη μέθοδο Strassen

Απαιτείται: ακέραιος $n = 2^m$, για $m \geq 0$, πίνακες P, R διάστασης $n \times n$.

Επιστρέφεται: πίνακας $Q = PR$.

```
1: function MatrixMult(int n, int P[], int R[], int Q[])
2:   if n = 1 then
3:     Q[1, 1] = P[1, 1] · R[1, 1];
4:   else
5:     half_n = n/2;
6:     for i ← 1; i ≤ half_n; i ++ do
7:       for j ← 1; j ≤ half_n; j ++ do
8:         A[i, j] ← P[i, j]; E[i, j] ← R[i, j];
9:         B[i, j] ← P[i, j + half_n]; F[i, j] ← R[i, j + half_n];
10:        C[i, j] ← P[i + half_n, j]; G[i, j] ← R[i + half_n, j];
11:        D[i, j] ← P[i + half_n, j + half_n]; H[i, j] ← R[i + half_n, j +
half_n];
12:       end for
13:     end for
14:     for i ← 1; i ≤ half_n; i ++ do
15:       for j ← 1; j ≤ half_n; j ++ do
16:         BD[i, j] ← B[i, j] – D[i, j]; GH[i, j] ← G[i, j] + H[i, j];
17:         AD[i, j] ← A[i, j] + D[i, j]; EH[i, j] ← E[i, j] + H[i, j];
18:         AC[i, j] ← A[i, j] – C[i, j]; EH[i, j] ← E[i, j] + H[i, j];
19:         AB[i, j] ← A[i, j] + B[i, j]; FH[i, j] ← F[i, j] – H[i, j];
20:         GE[i, j] ← G[i, j] – E[i, j]; CD[i, j] ← C[i, j] + D[i, j];
21:       end for
22:     end for
23:     MatrixMult(half_n, BD, GH, Q1);
24:     MatrixMult(half_n, AD, EH, Q2);
25:     MatrixMult(half_n, AC, EF, Q3);
26:     MatrixMult(half_n, AB, H, Q4);
27:     MatrixMult(half_n, A, FH, Q5);
28:     MatrixMult(half_n, D, GE, Q6);
29:     MatrixMult(half_n, CD, E, Q7);
30:     for i ← 1; i ≤ half_n; i ++ do
31:       for j ← 1; j ≤ half_n; j ++ do
32:         Q[i, j] ← Q1[i, j] + Q2[i, j] – Q4[i, j] + Q6[i, j];
33:         Q[i, j + half_n] ← Q4[i, j] + Q5[i, j];
34:         Q[i + half_n, j] ← Q6[i, j] + Q7[i, j];
35:         Q[i + half_n, j + half_n] ← Q2[i, j] – Q3[i, j] + Q5[i, j] – Q7[i, j];
36:       end for
37:     end for
38:   end if
39: end function
```

Το πρόβλημα του πολλαπλασιασμού πινάκων μπορεί να διατυπωθεί ως εξής: δοθέντων δύο πινάκων P, R θέλουμε να υπολογίσουμε το γινόμενο $Q = PR$.³ Χωρίς βλάβη της γενικότητας θα θεωρήσουμε ότι οι πίνακες P, R είναι διάστασης $n \times n$, όπου $n = 2^k$.

Ο πλέον άμεσος τρόπος είναι να υπολογιστεί ο πίνακας Q είναι κάθε στοιχείο του $q_{ij}, i, j \in \{1, \dots, n\}$ να προκύψει από την παράσταση

$$q_{i,j} = \sum_{t=1}^n p_{it} r_{tj}, \quad (7.10)$$

όπου p_{it} είναι το στοιχείο της γραμμής i και στήλης t του πίνακα P και r_{tj} είναι το στοιχείο της γραμμής t και στήλης j του πίνακα R . Ο αριθμός των ΣΥΒ που εκτελούνται προκειμένου να υπολογιστεί η τιμή του q_{ij} είναι τάξης $\Theta(n)$ αφού απαιτούνται n πολλαπλασιασμοί και $n - 1$ προσθέσεις. Άρα ο υπολογισμός του πίνακα Q πραγματοποιείται σε $\Theta(n^3)$ ΣΥΒ. Θα δούμε στη συνέχεια ότι με την τεχνική «Διαίρει και Βασίλευε» μπορούμε να επιτύχουμε πολυπλοκότητα $o(n^3)$.

Παρατηρούμε ότι οι πίνακες P, R μπορούν να διαμεριστούν σε πίνακες διάστασης $\frac{n}{2} \times \frac{n}{2}$ με τον ακόλουθο τρόπο

$$P = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad R = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Χρησιμοποιώντας το παραπάνω σχήμα, μπορούμε να υπολογίσουμε τον πίνακα Q ως

$$Q = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}. \quad (7.11)$$

Η παραπάνω παράσταση παραπέμπει σε αλγορίθμική υλοποίηση τύπου «Διαίρει και Κυρίευε» αφού ο υπολογισμός του γινομένου δύο πινάκων διάστασης $n \times n$ μπορεί να γίνει με βάση τα γινόμενα οκτώ πινάκων διάστασης $\frac{n}{2} \times \frac{n}{2}$ τα οποία συνδυάζονται με τέσσερις πράξεις πρόσθεσης πινάκων αντίστοιχης διάστασης. Η αναδρομική σχέση που ικανοποιεί ο αριθμός των ΣΥΒ είναι

$$T(n) = \begin{cases} \Theta(1), & n = 1, \\ 8T(\frac{n}{2}) + cn^2, & n > 1 \end{cases}$$

Περιγραφή

Όμως και πάλι ο αριθμός των ΣΥΒ για τον υπολογισμό του πίνακα Q , όπως προκύπτει από το Θεώρημα 1 (πρώτη περίπτωση) είναι τάξης $\Theta(n^3)$. Στην εργασία [18] χρησιμοποιούνται οι υποπίνακες A, \dots, H που ορίστηκαν προηγουμένως προκειμένου να επιτευχθεί ο υπολογισμός σε ασυμπτωτικά καλύτερο χρόνο. Συγκεκριμένα, αποδεικνύεται με στοιχειώδη άλγεβρα πινάκων ότι ο πίνακας Q προκύπτει ως εξής:

$$Q = \begin{bmatrix} Q_1 + Q_2 - Q_4 + Q_6 & Q_4 + Q_5 \\ Q_6 + Q_7 & Q_2 - Q_3 + Q_5 - Q_7 \end{bmatrix} \quad (7.12)$$

όπου

$$\begin{aligned} Q_1 &= (B - D)(G + H), \\ Q_2 &= (A + D)(E + H), \\ Q_3 &= (A - C)(E + F), \\ Q_4 &= (A + B)H, \\ Q_5 &= A(F - H), \\ Q_6 &= D(G - E), \\ Q_7 &= (C + D)E. \end{aligned}$$

Η ορθότητα της διαδικασίας αποδεικνύεται εύκολα αντικαθιστώντας τους πίνακες $Q_t, t \in \{1, \dots, 7\}$, στην (7.12) από τις παραπάνω ισότητες και κάνοντας αλγεβρικές πράξεις· σαν αποτέλεσμα προκύπτουν οι υποπίνακες που συνθέτουν τον Q στην (7.11). Ως παράδειγμα θα δείξουμε ότι $Q_4 + Q_5 = AF + BH$. Πράγματι,

$$Q_4 + Q_5 = (A + B)H + A(F - H) = AH + BH + AF - AH = AF + BH.$$

Παρατηρούμε ότι ο υπολογισμός του πίνακα $Q_t, t \in \{1, \dots, 7\}$, προκύπτει από την επίλυση στιγμιότυπου του ίδιου προβλήματος (πολλαπλασιασμός πινάκων) με είσοδο πίνακες με τον μισό αριθμό γραμμών και στηλών από τους πίνακες P και R . Η παρατήρηση αυτή οδηγεί στον Αλγόριθμο 46 που χρησιμοποιεί τη μεθοδολογία «Διαιρεί και Κυρίευε».

Στη γενική περίπτωση που η τιμή του n δεν είναι δύναμη του δύο συμπληρώνουμε τους πίνακες με μηδενικές γραμμές και στήλες μέχρι να φτάσουμε στον μικρότερη δύναμη του 2 που είναι μεγαλύτερη του n .

Ο αριθμός των ΣΥΒ ικανοποιεί την αναδρομική σχέση

$$T(n) = \begin{cases} \Theta(1), & n = 1, \\ 7T\left(\frac{n}{2}\right) + cn^2, & n > 1 \end{cases}$$

Σύμφωνα με την πρώτη περίπτωση του Θεωρήματος 1, $T(n) = \Theta(n^{\lg_2 7})$ και άρα ο υπολογισμός του γινόμενου δύο πινάκων μέσω του Αλγόριθμου 46 γίνεται ασυμπτωτικά γρηγορότερα από τον υπολογισμό που γίνεται τετριμμένα μέσω της (7.10).

Πολυπλοκότητα

Median_of_Medians (Διάμεσος των διαμέσων)

Αλγόριθμος 47 k -ιοστό μικρότερο στοιχείο

Απαιτείται: πίνακας A με στοιχεία στις θέσεις από lo έως hi , ακέραιος $k \in \{1, \dots, hi - lo + 1\}$.

Επιστρέφεται: k -ιοστό μικρότερο στοιχείο του πίνακα A .

```
1: function Median_of_Medians(int A[], int lo, int hi, int k)
2:   if lo = hi then
3:     return lo
4:   end if
5:   n ← hi - lo + 1;
6:   y ← n mod 5;
7:   if y = 0 then
8:     y ← 5;
9:   end if
10:  groups ← 0;
11:  first_in_group ← lo; last_in_group ← first_in_group + y - 1;
12:  while last_in_group ≤ hi do
13:    group_median ← Median_of_Five(A, first_in_group, last_in_group);
14:    Swap(A[lo + groups], A[group_median]);
15:    groups++;
16:    first_in_group ← first_in_group + y;
17:    last_in_group ← first_in_group + 4;
18:    y ← 5;
19:  end while
20:  mid ← lo + ⌊ $\frac{groups}{2}$ ⌋ - 1 + (groups mod 2);  $\triangleright$  Αναδρομικός υπολογισμός
    θέσης της διαμέσου των διαμέσων
21:  pvt ← Median_of_Medians(A, lo, lo + groups - 1, mid);  $\triangleright$  Διαχωρισμός με
    βάση τη διάμεσο των διαμέσων
22:  j ← Pivot_Partition(A, lo, hi, pvt);
23:  i ← j - lo + 1;
24:  if i = k then
25:    return A[j];
26:  else if i > k then
27:    return Median_of_Medians(A, lo, j - 1, k);
28:  else
29:    return Median_of_Medians(A, j + 1, hi, k - i);
30:  end if
31: end function
```

Στην Ενότητα 4.2.4 παραθέσαμε έναν αλγόριθμο για την εύρεση του k -ιοστού μικρότερου αριθμού σε μία αταξινόμητη σειρά n ακεραίων. Ο αλγόριθμος αυτός (Αλγόριθμος 27) εντάσσεται στη μεθοδολογία «Διαίρει και Κυρίευε» αφού σε κάθε αναδρομική κλήση επιλύει ένα στιγμιότυπο που είναι μικρότερου μεγέθους από το προηγούμενο. Ο αριθμός των ΣΥΒ που εκτελεί ο αλγόριθμος είναι τάξης $O(n^2)$ αφού σε κάθε αναδρομική κλήση, στη χειρότερη περίπτωση, ο αριθμός των στοιχείων του πίνακα A ανάμεσα στα οποία γίνεται η αναζήτηση μειώνεται μόλις κατά ένα. Οπότε είναι πιο αποτελεσματικό να ταξινομήσουμε τα στοιχεία σε αύξουσα σειρά (πολυπλοκότητα $O(n \lg n)$ με τη μέθοδο της συγχωνευτικής ταξινόμησης) και στη συνέχεια να επιλέξουμε απευθείας το k -ιοστό μικρότερο. Όμως το γεγονός ότι για ορισμένες τιμές της παραμέτρου k μπορούμε να λύσουμε το πρόβλημα σε γραμμικό χρόνο (π.χ. $k = 1$ ή $k = n$) προκρίνει την ιδέα ότι θα μπορούσε να γίνει το ίδιο για οποιαδήποτε τιμή της παραμέτρου k . Όντως στην εργασία [4] παρουσιάζεται μία τέτοια μέθοδος.

Η μέθοδος χρησιμοποιεί τον αναδρομικό Αλγόριθμο 27 επιλέγοντας το στοιχείο οδηγό $A[put]$ - άρα τη θέση put - όχι τυχαία αλλά με συγκεκριμένο τρόπο. Θυμίζουμε ότι η επιλογή του οδηγού είναι κρίσιμη για τον αριθμό των στοιχείων ανάμεσα στα οποία θα πραγματοποιηθεί η αναζήτηση στην επόμενη κλήση (Γραμμές 9 και 11). Προκύπτει ότι αριθμός των στοιχείων στη χειρότερη περίπτωση είναι μικρότερος από ένα σημαντικό κλάσμα του n . Η διαδικασία είναι ένα σχήμα «Διαίρει και Κυρίευε» το οποίο βασίζεται στην έννοια της διάμεσου σε μία σειρά στοιχείων, διάμεσος είναι το στοιχείο που είναι μικρότερο-ίσο από τα μισά στοιχεία της σειράς και μεγαλύτερο-ίσο από

τα υπόλοιπα. Αρχικώς χωρίζεται η σειρά των αταξινόμητων στοιχείων σε ομάδες των πέντε στοιχείων (πεντάδες) και υπολογίζεται η διάμεσος της κάθε ομάδας. Οι διάμεσοι των ομάδων διαμορφώνουν μία νέα αταξινόμητη σειρά. Η διάμεσος της σειράς αυτής (διάμεσος των διαμέσων) είναι ένα στοιχείο που πληροί την εξής ιδιότητα: τουλάχιστον $\frac{3n}{10}$ στοιχεία της αρχικής σειράς είναι μικρότερα-ίσα από αυτή. Άρα αν επιλεγεί το στοιχείο αυτό σαν οδηγός στα πλαίσια του Αλγόριθμου 27, το πλήθος των στοιχείων ανάμεσα στα οποία καλείται να γίνει η αναζήτηση στην επόμενη κλήση του αλγόριθμου (κλήση της διαδικασίας k -Element) (είτε στη Γραμμή 9 ή στην 11) δεν μπορεί να ξεπερνάει τα $\frac{7n}{10}$. Τέλος παρατηρούμε ότι ο υπολογισμός της διάμεσου των διαμέσων μπορεί να γίνει με την αναδρομική κλήση της παραπάνω διαδικασίας αναζητώντας τη διάμεσο στην αταξινόμητη σειρά των διαμέσων των πεντάδων.

Γιατί όμως η διάμεσος των διαμέσων έχει την προαναφερθείσα ιδιότητα; Παρατηρούμε ότι ο αριθμός των διαμέσων είναι ίσος με τον αριθμό των πεντάδων, δηλαδή $\frac{n}{5}$.⁴ Η διάμεσος των διαμέσων είναι μεγαλύτερη-ίση από τις μισές από αυτές, δηλαδή από $\frac{n}{2} = \frac{n}{10}$ ενώ κάθε μία από αυτές είναι μεγαλύτερη-ίση από δύο στοιχεία (αφού κάθε ομάδα αποτελείται από πέντε στοιχεία). Επομένως η διάμεσος των διαμέσων είναι μεγαλύτερη-ίση τουλάχιστον από

$$\frac{n}{10} + 2 \frac{n}{10} = \frac{3n}{10}$$

στοιχεία της αρχικής σειράς

Η διαδικασία εξειδικεύεται στον Αλγόριθμο 47 ο οποίος υπολογίζει τη θέση του k -ιστού μικρότερου στοιχείου μίας αταξινόμητης σειράς ακεραίων. Ο αλγόριθμος αυτός μπορεί να θεωρηθεί μία εκτεταμένη έκδοση του Αλγόριθμου 27. Χρησιμοποιεί τη routīna Median_of_Five, ή οποία - δεν εξειδικεύεται και - αφορά τον υπολογισμό της διάμεσου σε μία ομάδα από πέντε ή λιγότερα στοιχεία. Μια άλλη σημαντική λεπτομέρεια της εξειδίκευσης είναι ότι προκειμένου να μην χρησιμοποιηθεί επιπλέον μνήμη, η σειρά των διαμέσων των πεντάδων αποθηκεύεται στις πρώτες θέσεις του πίνακα A ανταλλάσσοντας τα στοιχεία αυτά με τα στοιχεία που βρίσκονται στις αντίστοιχες θέσεις.

Θεωρούμε ότι ο υπολογισμός της διαμέσου μίας ομάδας που αποτελείται από πέντε (ή λιγότερα) στοιχεία πραγματοποιείται σε σταθερό χρόνο. Υπάρχουν $\lceil \frac{n}{5} \rceil$ τέτοιες ομάδες και επομένως ο αριθμός των ΣΥΒ για τον υπολογισμό των διαμέσων είναι $O(n)$. Για την επίλυση του αρχικού στιγμιότυπου απαιτείται

- η εύρεση της διαμέσου των διαμέσων, δηλαδή η επίλυση ενός στιγμιότυπου του ιδίου προβλήματος με $\lceil \frac{n}{5} \rceil$ στοιχεία,
- η τοποθέτηση της διαμέσου των διαμέσων στη σωστή θέση με την εκτέλεση της συνάρτησης Pivot_Partition ($(O(n))$ ΣΥΒ),
- η επίλυση ενός στιγμιότυπου - πάλι του ιδίου προβλήματος - με το πολύ $\frac{7n}{10}$ στοιχεία.

Συνολικά, ο αριθμός των ΣΥΒ που εκτελεί ο αλγόριθμος ικανοποιεί την αναδρομική σχέση⁵

$$T(n) = \begin{cases} \Theta(1), & n \leq 5, \\ T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn, & n \geq 6. \end{cases}$$

Το δεξί μέλος της παραπάνω σχέσης για $n \geq 6$ γράφεται ως

$$T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + cn = T\left(\frac{n}{2}\right) + T\left(\frac{n}{7}\right) + cn. \quad (7.13)$$

Για να υπολογίσουμε την τάξη της παράστασης αυτής με τη χρήση του Θεώρηματος 3 πρέπει να βρούμε το p για το οποίο

$$\left(\frac{2}{10}\right)^p + \left(\frac{7}{10}\right)^p = 1.$$

Δεν χρειάζεται να υπολογίσουμε το p ακριβώς: επειδή το άθροισμα των δύο λόγων είναι μικρότερο της μονάδας, έχουμε ότι $0 < p < 1$ και άρα

$$\int_1^n \frac{g(u)}{u^{p+1}} du = \int_1^n u^{-p} du = \frac{u^{1-p}}{1-p} \Big|_{u=1}^n = \frac{n^{1-p} - 1}{1-p} = \Theta(n^{1-p}).$$

Επομένως σύμφωνα με το Θεώρημα 3, η (7.13) είναι τάξης $\Theta(n^p \cdot (1 + \Theta(n^{1-p}))) = \Theta(n)$ και άρα $T(n) = O(n)$.

Πολυπλοκότητα

Closest_Pair (Εγγύτερο ζεύγος σημείων)

Αλγόριθμος 48 Εγγύτερο ζεύγος σημείων στο επίπεδο

Απαιτείται: Πίνακας P με σημεία αποθηκευμένα στις θέσεις από lo έως hi και ταξινομημένα κατά αύξουσα σειρά ως προς τις τετμημένες τους και Πίνακας Q με τα σημεία του πίνακα P αποθηκευμένα στις θέσεις από 1 έως n και ταξινομημένα κατά αύξουσα σειρά ως προς τις τεταγμένες τους.

Επιστρέφεται: Εγγύτερα σημεία p_1, p_2 που ανήκουν στον πίνακα P και η απόσταση τους.

```
1: function Closest_Pair(point P[], int lo,int hi, point Q[], int n, point
   p1, point p2)
2:   if n ≤ 1 then
3:     n ← hi – lo + 1;
4:     δ ← ∞; p1 ← (0, 0); p2 ← (0, 0);
5:   else if 2 ≤ n ≤ 3 then
6:     for (i ← lo; i ≤ hi – 1; i++) do
7:       for (j ← i + 1; i ≤ hi; j++) do
8:         if δ > dist(P[i], P[j]); then
9:           δ ← dist(P[i], P[j]); p1 ← P[i]; p2 ← P[j];
10:        end if
11:      end for
12:    end for
13:   else
14:     mid ← ⌊ $\frac{lo+hi}{2}$ ⌋;
15:     j ← 0; k ← 0;
16:     for i ← 1; i ≤ n; i++ do
17:       if Q[i].x ≤ P[mid].x and j < mid – lo + 1 then
18:         j++; Q_L[j] ← Q[i]
19:       else
20:         k++; Q_R[k] ← Q[i]
21:       end if
22:     end for
23:     d_L ← Closest_Pair(P, lo, mid, Q_L, j, q1, q2);
24:     d_R ← Closest_Pair(P, mid + 1, hi, Q_R, k, r1, r2);
25:     if d_L < d_R then
26:       δ ← d_L; p1 ← q1; p2 ← q2;
27:     else
28:       δ ← d_R; p1 ← r1; p2 ← r2;
29:     end if
30:     k ← 0;
31:     for (j ← 1; j ≤ n; j++) do
32:       if |Q[j].x – P[mid].x| < δ then
33:         k++; Q_S[k] ← Q[j];
34:       end if
35:     end for
36:     for (i ← 1; i ≤ k – 1; i++) do
```

```

37:      $j \leftarrow 1;$ 
38:     while  $i + j \leq k$  and  $j \leq 7$  do
39:         if  $\delta > \text{dist}(Q_S[i], Q_S[i + j])$  then
40:              $\delta \leftarrow \text{dist}(Q_S[i], Q_S[i + j]);$ 
41:              $p_1 \leftarrow Q_S[i]; p_2 \leftarrow Q_S[i + j];$ 
42:         end if
43:          $j \leftarrow j + 1;$ 
44:     end while
45: end for
46: end if
47: return  $\delta;$ 
48: end function

```

Το πρόβλημα αυτό είναι πολύ απλό στη διατύπωση του: δεδομένου ενός συνόλου σημείων $P \subset \mathbb{R}^2$, να βρεθεί το ζεύγος των σημείων που ελαχιστοποιεί την μεταξύ τους απόσταση.

Χωρίς βλάβη της γενικότητας μπορούμε να θεωρήσουμε ως μέτρο την Ευκλείδεια απόσταση- δεδομένων δύο σημείων $x^1 = (x_1, y_1)$ και $x^2 = (x_2, y_2)$,

$$d(x^1, x^2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}. \quad (7.14)$$

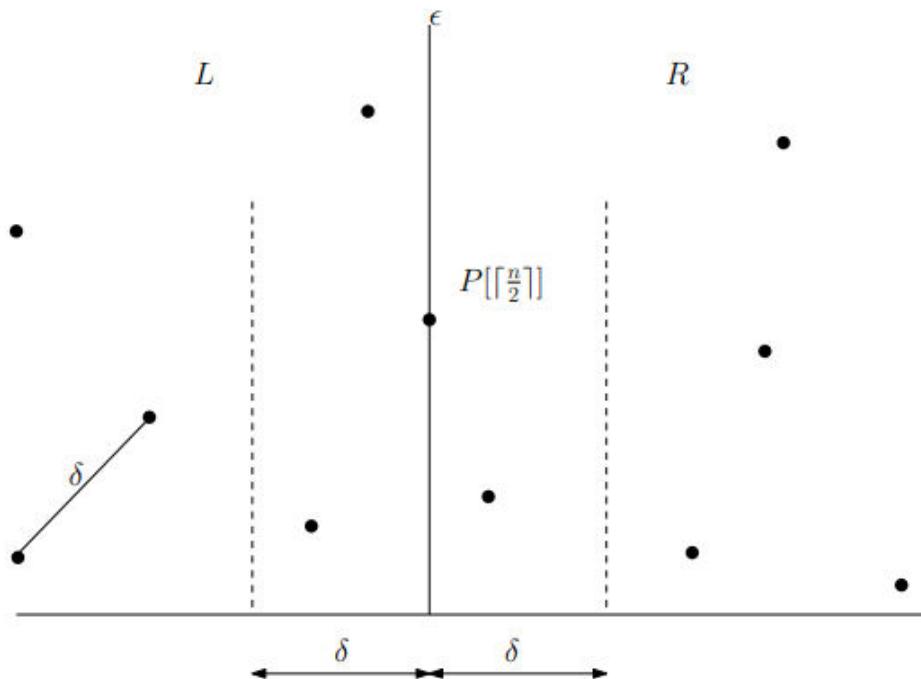
Έτσι το παραπάνω πρόβλημα έγκειται στην εύρεση σημείων $x^p, x^q \in P$ τέτοιων ώστε

$$d(x^p, x^q) = \min\{d(x^r, x^s) : x^r, x^s \in P\}.$$

Ένας απλός τρόπος να επιλύσουμε το πρόβλημα αυτό είναι να υπολογίσουμε την ποσότητα (7.14) για κάθε ζευγάρι σημείων και να επιστρέψουμε το ζευγάρι που την ελαχιστοποιεί. Κάτι τέτοιο απαιτεί $O(n^2)$ ΣΥΒ, όπου $|P| = n$. Όπως θα δούμε στη συνέχεια μέσω ενός αναδρομικού σχήματος «Διαιρεί και Κυρίευε» μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα.

Το αναδρομικό σχήμα λειτουργεί ως εξής. Αν το σύνολο P περιέχει το πολύ τρία σημεία ($n \leq 3$), τότε υπολογίζουμε την (7.14) για κάθε ζευγάρι και επιστρέφεται μαζί με την ελάχιστη τιμή της το αντίστοιχο ζευγάρι (συνθήκη τερματισμού αναδρομής). Διαφορετικά (αναδρομική κλήση), έστω $x^m = (x_m, y_m)$ το σημείο με την ιδιότητα⁶

$$|L| = |R|,$$



Σχήμα 7.3: Εγγύτερο ζεύγος σημείων

όπου

$$L = \{x^j = (x_j, y_j) \in P : x_j \leq x_m\},$$

$$R = \{x^j = (x_j, y_j) \in P \setminus L : x_j \geq x_m\}.$$

Δηλαδή, η τετμημένη του σημείου x^m είναι η διάμεσος των τετμημένων των σημείων του P .

Έχουμε τρεις περιπτώσεις για το που μπορεί να βρίσκεται το ζητούμενο ζευγάρι των εγγύτερων σημείων: είτε μπορεί να βρίσκεται στο σύνολο L , είτε στο R , ή το ένα σημείο του ζεύγους στο L και το άλλο στο R . Συνεπώς ορίζονται τρία στιγμιότυπα του ίδιου προβλήματος (φάση «Διαιρέι»). Τα δύο πρώτα στιγμιότυπα έχουν περίπου το μισό αριθμό σημείων από το αρχικό ($\frac{n}{2}$). Για το τρίτο στιγμιότυπο μας ενδιαφέρουν μόνο τα σημεία του συνόλου

$$Q = \{x^j = (x_j, y_j) : |x_j - x_m| < \delta\},$$

όπου $\delta = \min\{d_L, d_R\}$, όπου $d_L(d_R)$ η απόσταση του εγγύτερου ζεύγους σημείων του συνόλου $L(R)$.

Σχηματικά, φέρουμε μία κάθετη ευθεία (ϵ) από το σημείο x^m προς τον οριζόντιο άξονα και περιλαμβάνουμε στο Q όλα τα σημεία του P που απέχουν απόσταση μικρότερη από δ από την ευθεία αυτή - δες Σχήμα 7.3. Προφανώς $Q \subset P$ αφού τα σημεία που βρίσκονται σε απόσταση μεγαλύτερη-ίση από δ δεν ανήκουν στο Q .

Μας ενδιαφέρει το σύνολο Q αφού αν υπάρχει ένα ζευγάρι σημείων με απόσταση μικρότερη της δ τα σημεία αυτά θα ανήκουν στο σύνολο αυτό. Για κάθε σημείο $x^j \in Q$ ορίζουμε το σύνολο των σημείων

$$Q_j = \{x^k \in Q : d(x^j, x^k) < \delta\}$$

και

$$d_j = \min\{d(x^j, q) : q \in Q_j\},$$

Εστω q^j το σημείο του συνόλου Q_j για το οποίο ισχύει $d(x^j, q^j) = d_j$. Σαν λύση του τρίτου στιγμιότυπου επιστρέφεται το ζευγάρι των σημείων x^j, q^j που επιτυγχάνει την μικρότερη τιμή d_j . Η τιμή αυτή είναι

$$d_Q = \min\{d_j : x^j \in Q_j\}. \quad (7.15)$$

Τέλος - φάση «Συνδύασε» - επιστρέφεται η μικρότερη από τις τιμές δ, d_Q με το αντίστοιχο ζευγάρι σημείων.

Για να λειτουργήσει αποδοτικά το παραπάνω αναδρομικό σχήμα θα πρέπει να επιλύεται με υπολογιστικά αποτελεσματικό τρόπο το τρίτο στιγμιότυπο. Προς τούτο, θεωρούμε τα σημεία του Q είναι διατεταγμένα κατά αύξουσα σειρά ως προς τις τεταγμένες τους - δηλαδή

$$\bar{Q} = (x^{i_1} = (x_{i_1}, y_{i_1}), x^{i_2} = (x_{i_2}, y_{i_2}), \dots, x^{i_r} = (x_{i_r}, y_{i_r}))$$

όπου $1 \leq r \leq n$ και για κάθε $k \geq 1$ και $k \leq r-1$ ισχύει

$$y_{i_k} \leq y_{i_{k+1}}.$$

Θα επαναρίσουμε τα σύνολα Q_j με βάση το διατεταγμένο πια σύνολο \bar{Q} ως

$$\bar{Q}_j = \{x^k = (x_k, y_k) \in \bar{Q} : y^k > y_j, d(x^k, x^j) < \delta\},$$

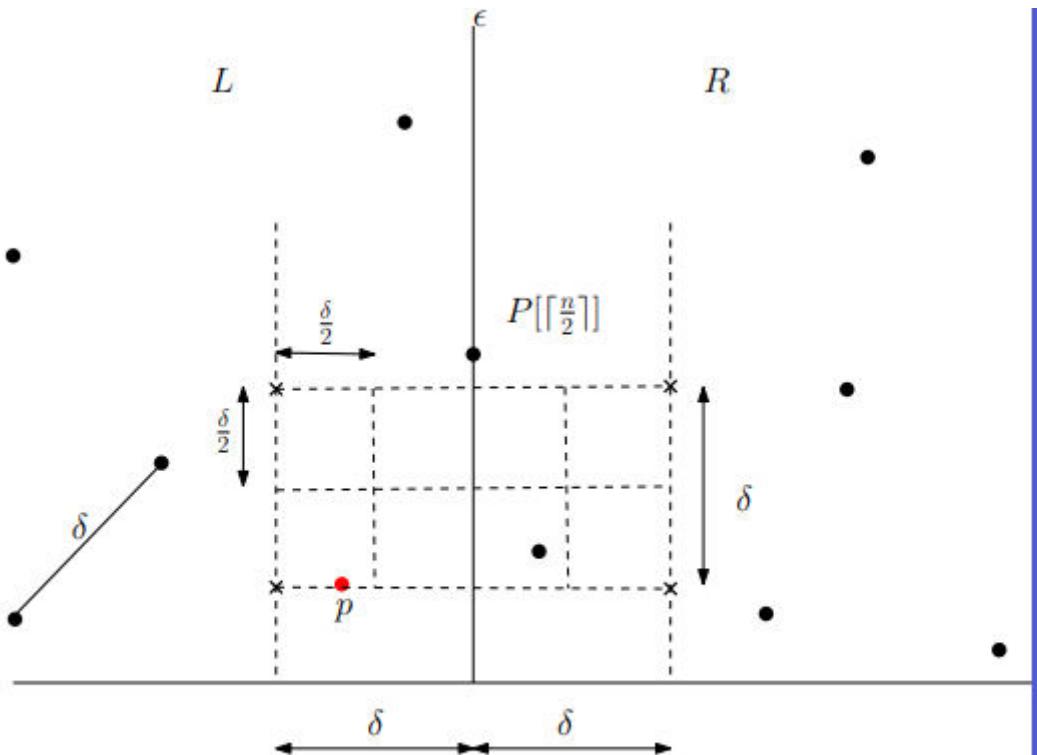
για κάθε $x^j = (x_j, y_j) \in \bar{Q}$.

Στο [2] αποδεικνύεται ένα εκπληκτικό αποτέλεσμα για κάθε \bar{Q}_j , ήτοι

$$|\bar{Q}_j| \leq 7.$$

Για να δούμε γιατί αυτό επαρκεί ας παρατηρήσουμε το Σχήμα 7.4. Έστω ότι το παρόν σημείο του \bar{Q} που εξετάζεται είναι το σημείο p . Αν υπάρχει κάποιο σημείο με μεγαλύτερη-ίση τεταγμένη με το σημείο αυτό που να έχει απόσταση μικρότερη από $\delta = \min\{d_L, d_R\}$ τότε το σημείο αυτό θα βρίσκεται αναγκαστικά μέσα στο παραλληλόγραμμο που ορίζουν τα σημεία \times . Μπορούμε να διαιρέσουμε το χώρο αυτό σε οκτώ τετράγωνα πλευράς ίσης με $\frac{\delta}{2}$, το καθένα όπως φαίνεται στο Σχήμα 7.4. Θα δείξουμε ότι σε καθένα από τα τετράγωνα αυτά μπορεί να υπάρχει μόνο ένα σημείο με τεταγμένη μεγαλύτερη-ίση με το p που να έχει απόσταση από αυτό μικρότερη από δ . Έστω ότι δεν ισχύει αυτό και υπάρχει σε κάποιο τετράγωνο παραπάνω από ένα τέτοιο σημείο. Εστω q_1, q_2 δύο τέτοια σημεία. Επειδή τα σημεία αυτά ανήκουν στο ίδιο τετράγωνο η απόσταση ανάμεσα τους πρέπει να είναι μεγαλύτερη-ίση από δ αφού το τετράγωνο ανήκει εξ' ολοκλήρου σε κάποιον από τους δύο ημιχώρους τους οποίους ορίζει η ευθεία που περνάει από το σημείο που αποτελεί διάμεσο και επομένως είτε $q_1, q_2 \in L$, ή $q_1, q_2 \in R$. Όμως αυτό αποτελεί αντίφαση με το γεγονός ότι η μεγαλύτερη απόσταση που μπορεί να χωρίζει δύο σημεία σε κάθε τετράγωνο είναι ίση με το μήκος της διαγωνίου του τετραγώνου το οποίο είναι ίσο με $\frac{\delta}{\sqrt{2}}$ το οποίο είναι μικρότερο από δ . Συνεπώς υπάρχει το πολύ ένα σημείο σε κάθε τετράγωνο και το ένα από αυτά είναι ήδη κατειλημμένο από το p . Άρα μένουν επτά τετράγωνα σε καθένα από τα οποία μπορεί να υπάρχει το πολύ ένα σημείο με τεταγμένη μεγαλύτερη-ίση με το p που να

Περιγραφή



Σχήμα 7.4: Εγγύτερο ζεύγος σημείων

έχει απόσταση από αυτό μικρότερη από δ . Επομένως επαρκεί για κάθε σημείο του \bar{Q} να υπολογιστούν αποστάσεις με τα επτά σημεία του (αν υπάρχουν) που έχουν αμέσως μεγαλύτερη-ίση τεταγμένη από αυτό.

Για να κωδικοποιηθεί η παραπάνω διαδικασία απομένει να διασαφηνιστούν κάποιες λεπτομέρειες. Ο αλγόριθμος δέχεται σαν είσοδο δύο πίνακες, ονομαστικά P, Q , που περιέχουν τα σημεία ανάμεσα στα οποία αναζητείται το εγγύτερο ζεύγος στις θέσεις από 1 έως n : στον πρώτο πίνακα τα σημεία είναι ταξινομημένα σε αύξουσα σειρά ως προς τις τετμημένες και στον δεύτερο ως προς τις τεταγμένες τους. Δηλαδή, σε όρους κωδικοποίησης, το σημείο που βρίσκεται αποθηκευμένο στη θέση j του πίνακα P έχει τετμημένη $P[j].x$, τεταγμένη $P[j].y$ και ισχύει ότι

$$P[i].x \leq P[i+1].x, i \in \{1, \dots, n-1\}.$$

Αντίστοιχα, ισχύει ότι

$$Q[j].x \leq Q[j+1].x, j \in \{1, \dots, n-1\}.$$

Σε κάθε κλήση, με πάνω από τρία σημεία, ο πίνακας Q «χωρίζεται στα δύο» τροφοδοτώντας τους πίνακες Q_L, Q_R με τα σημεία των συνόλων L και R , αντίστοιχα. Οι πίνακες αυτοί είναι ταξινομημένοι ως προς τις τεταγμένες και αποτελούν μαζί με τον πίνακα P την είσοδο στις κλήσεις που επιλέγουν τα στιγμιότυπα που αφορούν τα σημεία των συνόλων L, R . Για το τρίτο στιγμιότυπο, χρησιμοποιούνται μόνο τα σημεία του Q που έχουν απόσταση μικρότερη από δ από την ευθεία ϵ . Τα σημεία αυτά καταχωρούνται σε αύξουσα σειρά ως προς τις τεταγμένες τους σε έτερο πίνακα, ονομαστικά Q_S . Ο Αλγόριθμος 48 κωδικοποιεί την διαδικασία η οποία καλείται στα πλαίσια ενός «κύριου προγράμματος» ως

ταξινόμησε τα σημεία ως προς τις τετμημένες παράγοντας τον πίνακα P .
ταξινόμησε τα σημεία ως προς τις τετμημένες παράγοντας τον πίνακα Q .
εκτύπωσε απόσταση: $\text{Closest_Pair}(P, 1, n, Q, n, p_1, p_2)$ και ζεύγος σημείων (p_1, p_2) .

Η διαδικασία Closest_Pair παρουσιάζεται ως Αλγόριθμος 48. Δέχεται σαν είσοδο τον ταξινομημένο-ως-προς-τις-τετμημένες πίνακα P (τα σημεία καταλαμβάνουν τις θέσεις από lo ως hi .) τον ταξινομημένο-ως-προς-τις-τεταγμένες πίνακα Q (τα σημεία στον πίνακα αυτόν καταλαμβάνουν τις θέσεις από 1 ως n). Σαν έξοδο ο αλγόριθμος επιστρέφει στο όνομα της συνάρτησης την εγγύτερη απόσταση και τα σημεία για τα οποία επιτυγχάνεται η απόσταση αυτή στις παραμέτρους p_1, p_2 . Ο αλγόριθμος χρησιμοποιεί τη συνάρτηση dist η οποία δέχεται σαν είσοδο δύο σημεία και υπολογίζει τη μεταξύ τους απόσταση.

Μία τελευταία παρατήρηση για τον Αλγόριθμο 48 αφορά τον διαμοιρασμό των στοιχείων στους πίνακες Q_L, Q_R . Θα πρέπει το σύνολο των σημείων του πρώτου πίνακα να ταυτίζεται με το σύνολο $\{P[lo], \dots, P[mid]\}$ ενώ το δεύτερο με το σύνολο $\{P[mid+1], \dots, P[hi]\}$. Για να διασφαλιστεί αυτό στην περίπτωση που κάποια σημεία έχουν την ίδια τετμημένη (ή τεταγμένη), θα πρέπει ο πίνακας P να είναι δευτερευόντως ταξινομημένος ως προς τις τετμημένες. Συμπληρωματικά, θα πρέπει να ελέγχεται ότι ο αριθμός των στοιχείων του πίνακα Q_L είναι ίσος με

$$mid - lo + 1 = \lfloor \frac{lo + hi}{2} \rfloor - lo + 1$$

το οποίο από την (7.8) είναι ίσο με $\lceil \frac{n}{2} \rceil$.

Θεώρημα 2: «ΤΥΠΟΙ-ΙΛΙΟΤΗΤΕΣ/05_ΑΝΑΔΡΟΜΙΚΕΣ ΣΧΕΣΕΙΣ/Θεώρημα Κυριαρχικού Όρου»

Εφόσον μας δίνονται n στοιχεία, η ταξινόμηση των πινάκων P, Q είτε με την μέθοδο της συγχωνευτικής ταξινόμησης ή με τη χρήση σωρού πάρνει χρόνο $O(n \lg n)$.

Σε σχέση με τη διαδικασία Closest_Pair , όταν αληθεύει η συνθήκη τερματισμού της αναδρομής εκτελείται σταθερός αριθμός εκτελούνται - περίπτωση κατά την οποία $n \leq 3$. Για το δημιουργία των στιγμιότυπων που αφορούν τα σύνολα των σημείων L και R - τοποθέτηση σημείων στους πίνακες Q_L, Q_R , αντίστοιχα, εκτελούνται $\Theta(n)$ ΣΥΒ. Αν $T(n)$ είναι ο αριθμός των ΣΥΒ για την επίλυση του αρχικού προβλήματος, τότε $(\frac{n}{2})$ είναι ο αριθμός των ΣΥΒ που εκτελούνται για την επίλυση των δύο αυτών στιγμιότυπων.⁷ Για το τρίτο στιγμιότυπο, ο αριθμός των σημείων στον πίνακα Q_S δεν μπορεί να ξεπερνά το n και επειδή για κάθε στοιχείο ελέγχονται το πολύ τα επόμενα επτά (στον πίνακα Q_S), ο αριθμός των ΣΥΒ για την επίλυση του τρίτου στιγμιότυπου είναι $O(n)$.

Από τα παραπάνω προκύπτει ότι ο αριθμός των ΣΥΒ που εκτελεί ο Αλγόριθμος 48 (διαδικασία Closest_Pair) ικανοποιεί την αναδρομική σχέση

$$T(n) = \begin{cases} \Theta(1), & n \leq 3, \\ 2T(\frac{n}{2}) + O(n), & n \geq 4, \end{cases}$$

Αν στην παραπάνω αναδρομική σχέση ο όρος $O(n)$ αντικατασταθεί από τον $\Theta(n)$, μπορούμε, με τη χρήση του Θεωρήματος 2, να εκτιμήσουμε ότι η λόση της αναδρομικής σχέσης είναι τάξης $\Theta(n \lg n)$. Αποκαθιστώντας τον τελευταίο όρο της αναδρομικής σχέσης σε $O(n)$, παίρνουμε ότι η ασυμπτωτική εκτίμηση ισχύει ως προς το άνω φράγμα, δηλαδή $T(n) = O(n \lg n)$. Παρατηρούμε ότι ο αριθμός των ΣΥΒ που εκτελούνται αρχικώς (πριν την κλήση του Αλγόριθμου 48) για την ταξινόμηση των σημείων σε πίνακες P, Q , είναι ίδιας τάξης, επομένως συνολικός αριθμός ΣΥΒ για τον υπολογισμό εγγύτερου ζεύγους σημείων είναι τάξης $O(n \lg n)$.

Πολυπλοκότητα

Majority (Πλειοψηφούντος στοιχείο)

Algorithm 12 Εύρεση πλειοψηφούντος στοιχείου.

```
1: int majority(int A[], int lb, int ub)
2: if lb = ub then
3:   return A[lb];
4: else
5:   half_length ← ⌊ $\frac{ub-lb+1}{2}$ ⌋;
6:   k ← ⌊ $\frac{ub+lb}{2}$ ⌋;
7:   maj_el1 ← majority(A, lb, k);
8:   maj_el2 ← majority(A, k + 1, ub);
9:   if maj_el1 = maj_el2 then
10:    return maj_el1;
11:   else
12:     if count_el(A, lb, ub, maj_el1) > half_length then
13:       return maj_el1;
14:     else if count_el(A, lb, ub, maj_el2) > half_length then
15:       return maj_el2;
16:     else
17:       return -1;
18:     end if
19:   end if
20: end if
```

Algorithm 13 Πλήρθος εμφανίσεων του στοιχείου el στις θέσεις από $A[lb]$ έως $A[ub]$.

```
1: int count(int A[], int lb, int ub, int el)
2: k ← 0;
3: for i ← lb : i ≤ ub : i ← i + 1 do
4:   if A[i] = el then
5:     k ← k + 1;
6:   end if
7: end for
8: return k;
```

Άσκηση 12 Ένας πίνακας $A[1, \dots, n]$ έχει ένα πλειοψηφούν στοιχείο αν τα περισσότερα από τα μισά στοιχεία του είναι ίδια. Να εκπονηθεί αλγόριθμος πολυπλοκότητας $O(n \lg n)$ ο οποίος να προσδιορίζει αν ο πίνακας έχει πλειοψηφούν στοιχείο και αν ναι να βρίσκει το στοιχείο αυτό. Θεωρήστε ότι τα στοιχεία που βρίσκονται αποδημεύμενα στον πίνακα δεν είναι απαραίτητα αριθμοί και επομένως μπορεί να χρησιμοποιηθούν λογικές εκφράσεις τις μορφής $A[i] = A[j]$ ή $A[i] \neq A[j]$ και όχι $A[i] >= A[j]$ ή $A[i] > A[j]$.

Δύση Έστω ότι ο πίνακας έχει πλειοψηφούν στοιχείο. Αν τον χωρίσουμε στη μέση, τότε σε κάποιο από τα δύο μισά (ή και στα δύο) το στοιχείο αυτό θα παραμένει πλειοψηφούν στοιχείο. Επομένως μπορούμε αναδρομικά να διασπάμε το πρόβλημα σε δύο υποπροβλήματα μισού μεγέθους το καθένα και αν υπάρχει πλειοψηφούν στοιχείο σε κάποιο από τα δύο υποπροβλήματα να το αποθηκεύουμε. Αν και στα δύο υποπροβλήματα υπάρχει πλειοψηφούν στοιχείο και είναι το ίδιο τότε αυτό είναι το πλειοψηφούν στοιχείο του προβλήματος. Αν το κάθε υποπρόβλημα έχει διαφορετικό πλειοψηφούν στοιχείο τότε ελέγχουμε για το κάθε πλειοψηφούν στοιχείο αν εμφανίζεται σε περισσότερες από τις μισές θέσεις του αρχικού προβλήματος. Αν κανένα από τα δύο στοιχεία δεν ικανοποιούν την ιδιότητα αυτή τότε το πρόβλημα δεν έχει πλειοψηφούν στοιχείο. Διαφορετικά, το στοιχείο που εμφανίζεται σε περισσότερες από τις μισές θέσεις του αρχικού προβλήματος είναι το πλειοψηφούν στοιχείο. Ο Αλγόριθμος 12 υλοποιεί την αναδρομική διαδικασία θεωρώντας ότι ο πίνακας A περιέχει ακεραίους. Ο αλγόριθμος χρησιμοποιεί τον Αλγόριθμο 13 με τη μορφή συνάρτησης η οποία επιστρέφει το πλήθος των εμφανίσεων ενός συγκεκριμένου στοιχείου που δίνεται σαν είσοδος σε μία περιοχή του πίνακα.

Περιγραφή

Με δεδομένο ότι στην αρχική κλήση έχουμε n στοιχεία ($lb = 1$, $ub = n$), ο αλγόριθμος δημιουργεί δύο υποπροβλήματα το καθένα μισού μεγέθους $(n/2)$. Μετά συνδυάζει τις λύσεις εκτελώντας $O(n)$ συγκρίσεις (κλήση της διαδικασίας count). Ο αριθμός των στοιχειωδών πράξεων είναι

$$T(n) \leq B(n)$$

$$B(n) = 2B\left(\frac{n}{2}\right) + cn.$$

Σύμφωνα με το κεντρικό θεώρημα (δεύτερη περίπτωση) $B(n) = \Theta(n \lg n)$ και επομένως

$$T(n) = O(n \lg n)$$

Πολυπλοκότητα