Διαίρει και Βασίλευε
Ασκήσεις

Δημήτριος Μάγος

28 Νοεμβρίου 2017

Άσκηση 1 *(Πύργοι του Ανόι) Δίνονται τρεις στήλες έστω Α, Β,Γ. Στην πρώτη στήλη υπάρχουν n δίσκοι ο ένας πάνω στον άλλου με αύξουσα διάμετρο από την κορυφή προς τη βάση. Να εκπονηθεί αλγόριθμος ο οποίος να μετακινεί τους δίσκους στην στήλη Γ με του περιορισμό να μετακινείται ένας δίσκος τη φορά και να μην μπορεί σε καμία στιγμή να βρεθεί ένας δίσκος με μεγαλύτερη διάμετρο πάνω από έναν δίσκο με μικρότερη διάμετρο. Ποια η πολυπλοκότητα του αλγόριθμου·*

Λύση Για λόγους περιγραφικότητας ονομάζουμε την στήλη A source, τη στήλη B aux(iliary), τη στήλη Γ dest(ination). Η κεντρική ιδέα του ζητούμενου αλγορίθμου είναι πως το πρόβλημα μπορεί να αποσυνδεθεί σε δύο υποπροβλήματα τα οποία των εκτελούν αναδρομικά. Το πρώτο αφορά τη μεταφορά $n-1$ δίσκων από την στήλη source στη βοηθητική στήλη aux χρησιμοποιώντας σαν βοηθητική τη στήλη dest. Μετά μεταφέρουμε (με μία κίνηση) το $n-$ιοστό δίσκο από τη στήλη source στη στήλη dest. Τέλος έχουμε το δεύτερο υποπρόβλημα που αφορά τη μεταφορά των $n-1$ δίσκων από τη στήλη auxστη στήλη dest χρησιμοποιώντας σαν βοηθητική τη στήλη source .

Ο αλγόριθμος 1 κωδικοποιεί την παραπάνω διαδικασία- η αρχική κλήση γίνεται ως anoi(n,Ά,Β',Γ'). Σε κάθε κλήση γίνεται μία σύγκριση (γραμμή 2). Ο αριθμός των στοιχειωδών πράξεων είναι

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

Algorithm 1 Πύργοι του Ανόι

```
1: void anoi(int n, char source, char aux, char dest)
2: if n ≥ 1 then
3:     anoi(n − 1, source, dest, aux);
4:     printf("Μετεφερε το δίσκο από τη στήλη source στη στήλη dest");
5:     anoi(n − 1, aux, source, dest);
6: end if
```

Ο αλγόριθμος ολοκληρώνεται όταν $n = 0$ με $T(0) = 1$. Επομένως,

$$T(n) = 2^n \cdot 1 + \sum_{j=0}^{n-1} 2^j = 2^{n+1} - 1 = 2 \cdot 2^n - 1 = \Theta(2^n).$$

Άσκηση 2 *1. Να διατυπώσετε αλγόριθμο fastmultbin, ανάλογο του fastmult, ο οποίος να επιστρέφει το γινόμενο δύο ακεραίων των n-bit.*

2. Να διατυπώσετε αναδρομικό αλγόριθμο ο οποίος να μετατρέπει το δεκαδικό ακέραιο 10^m σε δυαδικό χρησιμοποιώντας τον αλγόριθμο fastmultbin.Θεωρήστε ότι το m είναι δύναμη του 2.

Λύση

1. Η ρουτινα fastmultbin είναι αντίστοιχη της fastmult και υλοποιείται στον Αλγόριθμο 2. Η πολυπλοκότητα του παραπάνω αλγόριθμου είναι

Algorithm 2 Γινόμενο δύο n-bit ακεραίων.

```
1: bin_int fastmultbin(bin_int x, bin_int y)
2: if n == 1 then
3:     return x · y;
4: end if
5: x^L ← [x]^{n/2}; x^R ← [x]_{n/2};
6: y^L ← [y]^{n/2}; y^R ← [y]_{n/2};
7: p_1 ← fastmultbin(x^L, x^R);
8: p_2 ← fastmultbin(y^L, y^R);
9: p_3 ← fastmultbin(x^L + x^R, y^L + y^R);
10: return 2^n p_1 + p_2 + 2^{n/2}(p_3 − p_1 − p_2);
```

ίδια με αυτή του fastmult, δηλαδή ο αλγόριθμος είναι τάξης $O(n^{1,59})$, όπου n είναι ο αριθμός των bits των δύο ακεραίων που δέχεται σαν είσοδο.

2. Η συνάρτηση που επιστρέφει τη δυαδική αναπαράσταση του αριθμού απεικονίζεται στον Αλγόριθμο 3. Ο αριθμός των bits που χρειαζόμαστε

Algorithm 3 Μετατροπή του 10^m σε δυαδικό.

```
1: bin_int pwr2bin(int m)
2: if (m = 1) then
3:     return 1010_2;
4: else
5:     z ←pwr2bin(m/2);
6:     return fastmultbin(z, z);
7: end if
```

για την αποθήκευση του 10^m είναι

$$n = \lfloor \lg 10^m \rfloor + 1 = \lfloor m \lg 10 \rfloor + 1 \tag{1}$$

Ο Αλγόριθμος 3 κάνει μία σύγκριση σε κάθε αναδρομική κλήση. Καλεί τον αλγόριθμο fastmultbinμε ακέραιους των $\lfloor \frac{m \lg 10}{2} \rfloor + 1$ bits. Επομένως, ο αριθμός των πράξεων που εκτελεί ο αλγόριθμος φράζεται από τα επάνω ως εξής:

$$T(m) \leq T(\frac{m}{2}) + 1 + c\left(\lfloor \frac{m \lg 10}{2} \rfloor + 1\right)^{\lg 3}, \tag{2}$$

όπου ο τελευταίος όρος στο αριστερό μέλος αφορά την πολυπλοκότητα του αλγόριθμου fastmultbin όταν αυτός εφαρμόζεται σε έναν ακέραιο των $\lfloor \frac{m \lg 10}{2} \rfloor + 1$ bits όπως ο $10^{\frac{m}{2}}$. Επειδή

$$\lfloor \lg 10 \rfloor + 1 = 4 \Rightarrow \lfloor \frac{\lg 10}{2} \rfloor + 1 = 2 \Rightarrow m \cdot \left(\lfloor \frac{\lg 10}{2} \rfloor + 1\right) = 2m \Rightarrow \lfloor \frac{m \lg 10}{2} \rfloor + 1 \leq 2m.$$

Αντικαθιστώντας στην (2),

$$\begin{aligned} T(m) &\leq T(\frac{m}{2}) + 1 + c(2m)^{\lg 3} \\ &= T(\frac{m}{2^k}) + k + 2^{\lg 3} c m^{\lg 3} \sum_{j=0}^{k-1} \left(\frac{1}{2^j}\right)^{\lg 3} \\ &= T(\frac{m}{2^k}) + k + 3^{\lg 2} c m^{\lg 3} \sum_{j=0}^{k-1} \left(\frac{1}{2^{\lg 3}}\right)^{j} \\ &\leq T(\frac{m}{2^k}) + k + 3 c m^{\lg 3} \sum_{j=0}^{\infty} \left(\frac{1}{3}\right)^{j} \end{aligned}$$

Για $k = \lg m$ έχουμε $\frac{m}{2^k} = 1$. Για αυτή την τιμή $T(1) = 1$ αφού τότε γίνεται μία σύγκριση από τον Αλγόριθμο 3. Αντικαθιστώντας αυτή την τιμή του k στην παραπάνω έκφραση έχουμε

$$\begin{aligned} T(m) &\leq 1 + \lg m + 3cm^{\lg 3} \cdot \frac{1}{1 - \frac{1}{3}} \\ &= 1 + \lg m + \frac{9}{2} cm^{\lg 3} \end{aligned}$$

και άρα

$$T(m) = O(m^{\lg 3}).$$

Άσκηση 3 *Να εκπονήσετε έναν αλγόριθμο ο οποίος να μετατρέπει έναν δεκαδικό ακέραιο με n ψηφία στο δυαδικό σύστημα χρησιμοποιώντας έναν πίνακα $binary[0, \ldots, 9]$ όπου σε κάθε θέση του έχει τη δυαδική αναπαράσταση του αντίστοιχου δεκαδικού ψηφίου, δλδ., $binary[0] = 0_2, binary[1] = 1_2, binary[2] = 10_2, \ldots, binary[9] = 101_2$. Ποια η πολυπλοκότητα του αλγόριθμου· Για απλοποίηση θεωρήστε ότι το n είναι δύναμη του 2.*

Λύση Πριν παρουσιάσουμε τον αλγόριθμο θα προβούμε στην εξής κρίσιμη παρατήρηση. Ένας ακέραιος a με d ψηφία ικανοποιεί τη σχέση

$$10^{d-1} \leq a \leq 10^d - 1. \tag{3}$$

Λογαριθμώντας την αριστερή ανισότητα έχουμε ένα φράγμα για τη δυαδική αναπαράσταση του ακέραιου a σε σχέση με την παράμετρο d, ήτοι

$$\begin{aligned} \lg a \leq \lg(10^d - 1) \Rightarrow & \\ \lfloor \lg a \rfloor + 1 &\leq \lfloor \lg(10^d - 1) \rfloor + 1 && (4) \\ &= \lfloor \lg(10^d) \rfloor + 1 && (5) \\ &= \lceil \lg(10^d) \rceil \Rightarrow && (6) \\ \lfloor \lg a \rfloor + 1 &\leq \lceil d \lg(10) \rceil && (7) \end{aligned}$$

Το δεξί μέλος της (4), ως γνωστόν, είναι ο αριθμός των bits που χρειάζονται για την δυαδική απεικόνιση του ακεραίου a. Η ισότητα (5) προκύπτει από το γεγονός ότι μία δύναμη του δέκα και η δύναμη αυτή μείον ένα είναι στην ίδια "περιοχή" δύναμης του δύο - η δύναμη του δέκα μείον ένα είναι περιττός αριθμός. Για τον ίδιο λόγο ισχύει η (6).

Με αντίστοιχο τρόπο μπορούμε να δείξουμε ότι

$$\lfloor \lg a \rfloor + 1 \geq \lfloor (d-1)\lg 10 \rfloor + 1 = \lceil (d-1)\lg 10 \rceil \tag{8}$$

Η τελευταία ισότητα ισχύει αφού δεν υπάρχουν θετικές δυνάμεις του 10 που να είναι ταυτόχρονα και δυνάμεις του 2.

Η κεντρική ιδέα του αλγόριθμου είναι να χωρίσουμε τον δοθέν αριθμό σε δύο μέρη έστω a_L, a_R με $n/2$ ψηφία το καθένα (το $n/2$ είναι ακέραιος αφού υποθέτουμε ότι n είναι δύναμη του δύο). Μπορούμε λοιπόν να γράψουμε τον a σαν

$$a = a_L * 10^{\frac{n}{2}} + a_R \tag{9}$$

Προφανώς μπορούμε να εφαρμόσουμε την ίδια ιδέα αναδρομικά προκειμένου να υπολογίσουμε σε κάθε κλίση τα a_L, a_R, μέχρι αυτά να καταλήξουν μονοψήφιοι αριθμοί.

Η ιδέα αυτή υλοποιείται στον Αλγόριθμο 4 μόνο που υπολογίζουμε την παράσταση (9) στο δυαδικό σύστημα. Για να γίνει αυτό χρησιμοποιούμε τις συναρτήσεις fastmultbin και pwr2bin. Ο αριθμός των ψηφίων του a υπολογίζεται από την (3), όπου $d = n$. Συγκεκριμένα,

$$10^{n-1} \leq a \Rightarrow n \leq \lfloor \lg_{10} a + 1 \rfloor,$$
$$a \leq 10^n - 1 < 10^n \Rightarrow \lg_{10} a < n \Rightarrow \lfloor \lg_{10} a + 1 \rfloor \geq n.$$

Επομένως ο αλγόριθμος θέτει

$$n = \lfloor \lg_{10} a + 1 \rfloor = \lfloor \lg_{10} a \rfloor + 1.$$

Algorithm 4 Μετατροπή του a σε δυαδικό.

```
1: bin_int dec2bin(int a)
2: if (a < 10) then
3:    return binary[a];
4: else
5:    n ← ⌊lg10 a⌋ + 1;
6:    a_L ← ⌊a / 10^(n/2)⌋;
7:    a_R ← a − a_L · 10^(n/2);
8:    return fastmultbin(dec2bin(a_L),pwr2bin(n/2))+dec2bin(a_R);
9: end if
```

Ένα κρίσιμο σημείο στην υλοποίηση του Αλγόριθμου 4 είναι η χρήση της συνάρτησης fastmultbin η οποία απαιτεί ορίσματα με τον ίδιο αριθμό bits. Όπως είδαμε σε προηγούμενη άσκηση, η συνάρτηση pwr2bin($n/2$) επιστρέφει έναν δυαδικό ακέραιο αποθηκευμένο σε $\lfloor \frac{n}{2} \lg 10 \rfloor + 1$ bits (θεώρησε (1) με $m = n/2$). Αντίστοιχα η συνάρτηση dec2bin(a_L) επιστρέφει μία δυαδική παράσταση η οποία δεν μπορεί να ξεπερνάει τον ίδιο αριθμό bits (θεώρησε (7) με $d = n/2$) - σε περίπτωση που η δυαδική παράσταση έχει μικρότερο αριθμό bits προσθέτουμε αντίστοιχα μηδενικά στην αρχή της παράστασης[1].

Ο αλγόριθμος εκτελεί $\Theta(n/2)$ βήματα προκειμένου να υπολογίσει τα a_L, a_R. Η κλίση στη συνάρτηση pwr2bin($n/2$) ολοκληρώνεται σε $O(\left(\frac{n}{2}\right)^{\lg 3})$ βήματα (δες προηγούμενη άσκηση). Επίσης η συνάρτησης fastmultbin είναι τάξης $O(\lceil \frac{n \lg 10}{2} \rceil^{\lg 3})$.

Συνολικά, ο αριθμός των στοιχειωδών πράξεων που εκτελεί ο Αλγόριθμος 4 είναι

$$T(n) \leq 2T(\frac{n}{2}) + c_1 \cdot \lceil \frac{n \lg 10}{2} \rceil^{\lg 3} + c_2 \cdot \left(\frac{n}{2}\right)^{\lg 3} + c_3 \cdot \frac{n}{2} \tag{10}$$

$$\leq 2T(\frac{n}{2}) + c_1 \cdot \lceil \frac{n \cdot 4}{2} \rceil^{\lg 3} + c_2 \cdot \left(\frac{n}{2}\right)^{\lg 3} + c_3 \cdot \frac{n}{2} \tag{11}$$

$$\leq 2T(\frac{n}{2}) + c_1 \cdot \left(\frac{n \cdot 4}{2}\right)^{\lg 3} + c_2 \cdot \left(\frac{n}{2}\right)^{\lg 3} + c_3 \cdot \left(\frac{n}{2}\right)^{\lg 3} \tag{12}$$

$$= 2T(\frac{n}{2}) + c_1 \cdot 4^{\lg 3} \cdot \left(\frac{n}{2}\right)^{\lg 3} + c_2 \cdot \left(\frac{n}{2}\right)^{\lg 3} + c_3 \cdot \left(\frac{n}{2}\right)^{\lg 3} \tag{13}$$

$$= 2T(\frac{n}{2}) + d \cdot \left(\frac{n}{2}\right)^{\lg 3} \tag{14}$$

όπου[2]

$$d = 3^{\lg 4} c_1 + c_2 + c_3 = 9c_1 + c_2 + c_3.$$

Επειδή θεωρούμε αρχικά ότι ο a είναι δύναμη του δύο, ο αριθμός για τον οποίο θα ενεργοποιηθεί η συνθήκη 'if (a<10) ' θα είναι το οκτώ ο οποίος χρειάζεται για να αποθηκευτεί τέσσερα bits. Επομένως για $n \leq 4$, έχουμε μία σύγκριση και σταματάει η αναδρομική κλήση: δηλαδή $T(4) = 1$. Τώρα

[1] Ο αριθμός των επιπλέον bits δεν μπορεί να ξεπερνάει τη διαφορά των αριστερών μελών των (7) και (8).

[2] Η (11) προκύπτει από την (10) από την ανισότητα $\lg 10 < 4$. Η (12) προέρχεται από την (11) επειδή ο τελευταίος όρος έχει υψωθεί στην $\lg 3$. Επίσης το ταβάνι στον δεύτερο όρο έχει απλοποιηθεί αφού το κλάσμα είναι ακέραιος.

είμαστε έτοιμοι για την επίλυση της (14). Ήτοι

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

Από την οριακή συνθήκη έχουμε

$$\frac{n}{2^k} = 4 \Rightarrow k = \lg n - 2.$$

Αντικαθιστώντας την τιμή του k στην παραπάνω σχέση, έχουμε

$$\begin{aligned}
&T(n) \leq 2^{\lg n - 2} \cdot 1 + dn^{\lg 3} \leq n + dn^{\lg 3} \Rightarrow \\
&T(n) = O(n^{\lg 3}).
\end{aligned}$$

Ένας πιο "εύκολος" τρόπος απόδειξης είναι να χρησιμοποιήσουμε το κεντρικό θεώρημα. Συγκεκριμένα, γράφουμε την (14) σαν

$$\begin{aligned}
&T(n) \leq B(n), \\
&B(n) = 2B(\frac{n}{2}) + d \cdot \left(\frac{n}{2}\right)^{\lg 3}.
\end{aligned}$$

Εφαρμόζοντας το κεντρικό θεώρημα στην $B(n)$, έχουμε $a = b = 2$, και $f(n) = d \cdot \left(\frac{n}{2}\right)^{\lg 3}$. Προφανώς, $f(n) = \Omega(n^{\lg 2+\epsilon}) = \Omega(n^{\epsilon})$ για $\lg 3 > \epsilon > 0$. Επίσης,

$$2d\left(\frac{n}{2^2})\right)^{\lg 3} = \frac{2}{3}dn^{\lg 3} \leq cdn^{\lg 3}$$

για $\frac{2}{3} \leq c \leq 1$. Είμαστε στην τρίτη περίπτωση του κεντρικού θεωρήματος και επομένως

$$B(n) = \Theta(n^{\lg 3}) \Rightarrow T(n) = O(n^{\lg 3}).$$

Άσκηση 4 *Να περιγράψετε έναν αλγόριθμο συγχώνευσης δύο ταξινομημένων πινάκων που περιέχουν ακέραιους αε έναν τρίτο πίνακα που να είναι και αυτός ταξινομημένος.*

Λύση Θεωρούμε ότι οι προς συγχώνευση πίνακες έστω A, B έχουν στη θέση 0 έναν ακέραιο που υποδηλώνει το πλήθος των στοιχείων που περιέχουν. Το ίδιο ισχύει για τον πίνακα που επιστρέφουν (τελευταία εντολή εκχώρησης). Η υλοποίηση απεικονίζεται στον Αλγόριθμο 5. Ο αριθμός των πράξεων είναι

Algorithm 5 Συγχώνευση δύο ταξινομημένων πινάκων A, B σε έναν πίνακα C.

```
 1: void merge(int A[], int B[], int C[])
 2: {
 3: i ← 1; j ← 1; k ← 0;
 4: n ← A[0]; m ← B[0];
 5: A[n + 1] ← B[m] + 1; B[m + 1] ← A[n] + 1;
 6: while i ≤ n or j ≤ m do
 7:    k + +;
 8:    if A[i] ≤ B[j] then
 9:       C[k] ← A[i]; i + +;
10:    else
11:       C[k] ← B[j]; j + +;
12:    end if
13: end while
14: C[0] ← k;
15: }
```

τάξης $\Theta(m + n)$.

Άσκηση 5 *Να περιγράψετε του αλγόριθμο ταξινόμησης με συγχώνευση και να αναλύσετε την πολυπλοκότητά του.*

Λύση Έστω ότι την αταξινόμητη συστοιχία είναι ο πίνακας A οπού περιέχει ακεραίους από την θέση 1 έως τη θέση n. Ακολουθούμε τη σύμβαση ότι η τιμή n είναι το στοιχείο $A[0]$. Ο αλγόριθμος 6 αναδρομικά διαιρεί τον πίνακα A (περίπου) στη μέση μέχρι να φτάσουμε σε συστοιχίες πληθυκότητας 1 οι οποίες είναι ταξινομημένες και μετά τις συγχωνεύει χρησιμοποιώντας τη διαδικασία της προηγούμενης άσκησης. Η αρχική κλήση της διαδικασίας είναι $lb = 1, ub = n$. Παρατηρούμε ότι η διαδικασία συγχώνευσης εκτελείται

Algorithm 6 Ταξινόμηση με βάση τη συγχώνευση.

```
1: void mergesort(int A[], int lb, int ub)
2: if lb < ub then
3:    int A1[], A2[], A3[]
4:    k ← ⌊(ub)/2⌋;
5:    A1[0] ← k − lb + 1;
6:    for i ← lb; i ≤ k; i++ do
7:       A1[i − lb + 1] ← A[i];
8:    end for
9:    A2[0] ← ub − k;
10:   for i ← k + 1; i ≤ ub; i++ do
11:      A2[i − k] ← A[i];
12:   end for
13:   mergesort(A1, 1, A1[0]);
14:   mergesort(A2, 1, A2[0]);
15:   merge(A1, A2, A3);
16:   for i ← lb; i ≤ ub; i++ do
17:      A[i] ← A3[i − lb + 1];
18:   end for
19: end if
```

σε $\Theta(n+n) = \Theta(n)$ βήματα. Το ίδιο ισχύει και για τον αριθμό των υπόλοιπων στοιχειωδών πράξεων σε κάθε κλήση της διαδικασίας. Επομένως, ο αριθμός των στοιχειωδών πράξεων δίνεται από τη σχέση

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n).$$

Χωρίς μεγάλη βλάβη της γενικότητας, μπορούμε να αγνοήσουμε το "πάτωμα" και "ταβάνι" στην παραπάνω σχέση. Έτσι, η σχέση γίνεται

$$T(n) = 2T(\frac{n}{2}) + cn,$$

η οποία εμπίπτει στην δεύτερη περίπτωση του κεντρικού θεωρήματος και επομένως

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

Άσκηση 6 *Δίνεται μία ακολουθία n ακεραίων. Να περιγράψετε έναν αποτελεσματικό αλγόριθμο ο οποίος να εξετάζει αν υπάρχουν δύο ακέραιοι στην ακολουθία αυτή, έστω x, y, που να έχουν άθροισμα έναν δοθέν ακέραιο m.*

Λύση Ταξινομούμε τους ακέραιους σε αύξουσα σειρά. Στη συνέχεια για κάθε ακέραιο $a_i, i \in \{1, \ldots, n\}$ εκτελώντας δυαδική αναζήτηση στην υπόλοιπη σειρά προσπαθούμε να εντοπίσουμε αν υπάρχει ακέραιος $m - a_i$. Για την ταξινόμηση χρειαζόμαστε $\Theta(n \lg n)$ βήματα. Για κάθε αναζήτηση $\lg n$ βήματα, και επειδή γίνονται το πολύ n τέτοιες αναζητήσεις, ο συνολικός αριθμός βημάτων είναι τάξης $O(n \lg n)$.

Άσκηση 7 *Ένας πίνακας $A[1, \ldots, n]$ περιέχει ταξινομημένους ακέραιους. Το πρόβλημα της αναζήτησης είναι "Δεδομένου ενός ακέραιου k να βρεθεί αν ο ακέραιος υπάρχει στον πίνακα A". Ο αλγόριθμος της αναζήτησης σε ομάδες χωρίζει τον πίνακα σε περιοχές μεγέθους $\sqrt{n}$ εντοπίζει την περιοχή (ομάδα) μέσα στην οποία πιθανόν να βρίσκεται το k (υπάρχουν $\frac{n}{\sqrt{n}} = \sqrt{n}$ τέτοιες ομάδες) και εκτελεί αναδρομικά την αναζήτηση στην περιοχή αυτή. Να διατυπώσετε αλγοριθμικά το παραπάνω σχήμα και να το συγκρίνεται με τη δυαδική αναζήτηση.*

Λύση Η διαδικασία υλοποιείται στον Αλγόριθμο 7: αν δεν υπάρχει το k επιστρέφεται η τιμή -1, αλλιώς η θέση στην οποία εμφανίζεται στο στοιχείο k. Στην αρχική κλήση οι τιμές των παραμέτρων της διαδικασίας είναι $lb = 1, ub = n$.

Σε κάθε κλήση ο αλγόριθμος εκτελεί $\Theta(\sqrt{n})$ πράξεις (δες γραμμές 4-6) όπου $n = ub - lb + 1$. Επομένως, η πολυπλοκότητα του αλγόριθμου δίνεται από τη σχέση

$$T(n) = T(\sqrt{n}) + p\sqrt{n}. \tag{15}$$

Algorithm 7 Αναζήτηση σε ομάδες.

```
1: int group_search(int A[], int lb, int ub, int k)
2: size ← √(ub − lb + 1);
3: index ← lb;
4: while index ≤ ub and A[index] < k do
5:    index ← index + size;
6: end while
7: if size = 1 then
8:    if A[index] = k then
9:       return k;
10:   else
11:      return -1;
12:   end if
13: end if
14: if index > ub then
15:   index ← ub;
16: end if
17: if A[index] = k then
18:   return k;
19: else
20:   ub ← index;
21:   lb ← max{lb, index − size + 1};
22:   return group_search(A, lb, ub, k);
23: end if
```

Για να επιλύσουμε την (15) χρησιμοποιούμε τη μέθοδο αντικατάστασης των μεταβλητών. Θέτουμε

$$m = \lg n \Rightarrow n = 2^m. \tag{16}$$

Η (15) γίνεται

$$T(2^m) = T(2^{\frac{m}{2}}) + p \cdot 2^{\frac{m}{2}}. \tag{17}$$

Στη συνέχεια θέτοντας

$$S(m) = T(2^m) \tag{18}$$

η (17) γίνεται

$$S(m) = S(\frac{m}{2}) + p \cdot 2^{\frac{m}{2}}. \tag{19}$$

Εξετάζοντας την (19) 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 $cf(m) = c \cdot p \cdot 2^{\frac{m}{2}}$ για κάποιο $c < 1$, π.χ. θεωρήστε $c = 1/2$. Άρα από την τρίτη περίπτωση του κεντρικού θεωρήματος έχουμε ότι $S(m) = \Theta(f(m)) = \Theta(2^{\frac{m}{2}})$. Από τις (15), ..., (19), έχουμε,

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

Η δυαδική αναζήτηση έχει συνάρτηση πολυπλοκότητας $\Theta(\lg n)$. Επομένως έχει ασυμπτωτικά καλύτερη συμπεριφορά από την αναζήτηση σε ομάδες αφού $\lg n < \sqrt{n}$ για $n \geq 17$.

Άσκηση 8 *Ένας πίνακας A περιέχει n ταξινομημένους ακεραίους (θέσεις $1, \ldots, n$) διαφορετικούς μεταξύ τους. Να περιγράψετε έναν αποτελεσματικό αλγόριθμο που να βρίσκει αν υπάρχει στον πίνακα θέση k τέτοια ώστε $A[k] = k$.*

Λύση Υποθέτουμε ότι οι αριθμοί είναι ταξινομημένοι σε αύξουσα σειρά, δηλαδή,

$$A[1] < A[2] < \cdots < A[n].$$

Η αυστηρή ανισότητα προκύπτει από το γεγονός ότι οι αριθμοί είναι διαφορετικοί μεταξύ τους.

Παρατηρούμε ότι αν $A[k] < k$, για κάποιο $k \in \{1, \ldots, n\}$, τότε δεν υπάρχει $k' < k$ για το οποίο $A[k'] = k'$. Αυτό συμβαίνει γιατί

$$A[k] < k \Rightarrow A[k] - 1 < k - 1 \Rightarrow A[k-1] < k - 1,$$

αφού $A[k-1] \leq A[k]-1$. Επομένως αν $A[k] < k$, ο αλγόριθμος πρέπει να αναζητά το στοιχείο στις θέσεις από $k+1$ ως n. Με αντίστοιχο επιχείρημα αν $k > A[k]$ θα πρέπει να αναζητείται το στοιχείο στις θέσεις από 1 μέχρι $k-1$.

Όπως στη δυαδική αναζήτηση, εκτελούμε την ίδια διαδικασία αναδρομικά μειώνοντας σε κάθε κλήση το πλήθος των στοιχείων που εξετάζουμε κατά το ήμιση (περίπου). Η διαδικασία υλοποιείται στον Αλγόριθμο 8 σαν συνάρτηση (αρχική κλήση: find_thesi_k($A, 1, n$)). Στην περίπτωση που δεν υπάρχει το ζητούμενο στοιχείο η συνάρτηση επιστρέφει τιμή -1.

Algorithm 8 Εύρεση θέσης $k = A[k]$.

```
1: int find_thesi_k(int A[], int lb, int ub)
2: if lb ≤ ub then
3:    k ← ⌊(lb+ub)/2⌋;
4:    if A[k]<k then
5:       lb ← k + 1;
6:       return find_thesi_k(A, lb, ub);
7:    else if A[k]>k then
8:       ub ← k − 1;
9:       return find_thesi_k(A, lb, ub);
10:   else
11:      return k;
12:   end if
13: else
14:   return -1;
15: end if
```

Ο αριθμός των στοιχειωδών πράξεων δίνεται από την αναδρομική σχέση

$$T(n) = T(\frac{n}{2}) + 4.$$

Όπως και στην περίπτωση της δυαδικής αναζήτησης,

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

Άσκηση 9 *Έστω πίνακας A ο οποίος περιέχει n ακεραίους στις θέσεις από 1 ως n. Να εκπονήσετε αναδρομικό αλγόριθμο ο οποίος να υπολογίζει το $k-$ιοστό μικρότερο στοιχείο του πίνακα ($k \leq n$).*

Λύση Θεωρούμε ότι αναζητούμε το στοιχείο από τη θέση lb μέχρι τη θέση ub του πίνακα (αρχικά $lb = 1, ub = n$). Η βασική ιδέα είναι αυτή που χρησιμοποιείται στη μέθοδο quicksort: επιλέγουμε τυχαία ένα στοιχείο το οποίο τοποθετείται στη "σωστή" θέση στον πίνακα. Ο όρος αυτός υποδηλώνει ότι το στοιχείο τοποθετείται στη θέση, έστω r, με την ιδιότητα

$$A[i] \leq A[r] \leq A[j], lb \leq i < r, r < j \leq ub.$$

Αν $r = k$ ο αλγόριθμος τερματίζει επιστρέφοντας το στοιχείο $A[r]$. Αν $k < r$, το στοιχείο βρίσκεται στις θέσεις από lb έως $r-1$: θέτουμε $ub = r-1$. Αν $r < k$, το στοιχείο βρίσκεται στις θέσεις από $r+1$ έως ub : θέτουμε $lb = r+1$. Στις δύο τελευταίες περιπτώσεις καλούμε αναδρομικά τη διαδικασία.

Algorithm 9 Εύρεση του k−ιοστου μικρότερου στοιχείου.

```
1: int kth_element(int A[], int lb, int ub, int k)
2: cp ← position(A, lb, ub);
3: if cp = k then
4:    return A[cp];
5: else
6:    if k < cp then
7:       ub ← cp − 1;
8:    else
9:       lb ← cp + 1;
10:   end if
11:   return kth_element(A, lb, ub, k);
12: end if
```

Ο Αλγόριθμος 9 που υλοποιεί τη διαδικασία χρησιμοποιεί τον αλγόριθμο 10 ο οποίος τοποθετεί στη σωστή θέση το δεξιότερο στοιχείο του πίνακα A.

Παρατηρούμε ότι ο Αλγόριθμος 10 σαρώνει όλα τα στοιχεία του πίνακα A. Στη χειρότερη δε περίπτωση μπορεί να τοποθετεί το δεξιότερα στοιχείο στην ίδια θέση. Επομένως ο αριθμός των στοιχειωδών πράξεων για το συγκεκριμένο αλγόριθμο είναι

$$c(ub - lb + 1) \leq cn.$$

Ο αλγόριθμος 9, εκτός της κλήσης στον Αλγόριθμο 10, εκτελεί δύο συγκρίσεις και μία αριθμητική πράξη (ανάλογα το αποτέλεσμα της σύγκρισης). Επειδή στη χειρότερη περίπτωση ο αριθμός των στοιχείων που εξετάζονται σε κάθε

Algorithm 10 Τοποθέτηση στοιχείου $A[ub]$ στη σωστή θέση.

```
 1: int position(int A[], int lb, int ub)
 2: i ← lb; j ← ub; pivot ← ub;
 3: if j = i then
 4:    intersect ← true;
 5: else
 6:    intersect ← false;
 7: end if
 8: while not intersect do
 9:    while i < ub do
10:       if j = i then
11:          intersect ← true; break;
12:       end if
13:       if A[i] > A[pivot] then
14:          break;
15:       end if
16:       i ← i + 1;
17:    end while
18:    if not intersect then
19:       while j > lb do
20:          if j = i then
21:             intersect ← true; break;
22:          end if
23:          if A[j] < A[pivot] then
24:             break;
25:          end if
26:          j ← j − 1;
27:       end while
28:    end if
29:    if not intersect then
30:       swap(A[i], A[j]);
31:    end if
32: end while
33: r ← i;
34: swap(A[i], A[pivot]);
35: return i;
```

κλήση μειώνεται κατά ένα, ο αριθμός των στοιχειωδών πράξεων δίνεται από τη σχέση

$$\begin{aligned} T(n) &\leq T(n-1) + cn + 3 \\ &= T(n-k) + c\sum_{i=k+1}^{n} i + 3k \\ &= c\sum_{i=1}^{n} i + 3n \Rightarrow \\ T(n) &= O(n^2). \end{aligned}$$

Άσκηση 10 *Δίνονται δύο ταξινομημένες ακολουθίες ακεραίων μεγέθους m, n αντίστοιχα. Να εκπονήσετε αλγόριθμο ο οποίος να υπολογίζει το $k-$ιοστό μικρότερο στοιχείο $(k \leq m+n)$ σε $O(\lg(m+n))$ βήματα.*

Λύση Θεωρούμε ότι τα στοιχεία βρίσκονται στις θέσεις από $lb1$ έως $ub1$ στον πίνακα A και από $lb2$ έως $ub2$ στον B - στην αρχική κλίση της αλγοριθμικής διαδικασίας θέτουμε

$$lb1 = 1, ub1 = m, lb2 = 1, ub2 = n.$$

Αρχικά θεωρούμε ότι από άποψη μεγέθους οι αριθμοί είναι μοιρασμένοι ομοιόμορφα στους δύο πίνακες. Συγκρίνουμε λοιπόν το $k/2$ στοιχείο του πίνακα A (στοιχείο $A[lb1 + \lfloor k/2 \rfloor - 1]$) με το στοιχείο $B[lb2 + (k - \lfloor k/2 \rfloor) - 1]$[3]. Αν το στοιχείο του A είναι μικρότερο τα πρώτα $k/2$ στοιχεία του A είναι μικρότερα-ίσα με το στοιχείο του B και επομένως με το προς αναζήτηση στοιχείο. Το ίδιο συμβαίνει συμμετρικά στη συμπληρωματική περίπτωση για το B. Επομένως τώρα γνωρίζουμε τα $k/2$ μικρότερα στοιχεία και μένει να εντοπίσουμε τα υπόλοιπα $k/2$ για να φτάσουμε στο ζητούμενο στοιχείο. Αναπροσαρμόζουμε το $lb1$ ή το $lb2$ ανάλογα αν τα μικρότερα $k/2$ στοιχεία βρίσκονται στον πίνακα A ή B, αντίστοιχα. Θέτουμε k ίσο με $k/2$ και κάνουμε αναδρομική κλίση του αλγόριθμου αν $k > 1$. Το σχήμα υλοποιείται στον Αλγόριθμο 11.

Ο αλγόριθμος δεν καλείται αναδρομικά όταν όλα τα στοιχεία του ενός από τους δύο πίνακες έχουν εξαντληθεί (γραμμές 2-15) ή όταν $k = 1$ (γραμμές 26-30).

[3] Γιατί·

Algorithm 11 $k-$ιοστό στοιχείο δύο ταξινομημένων πινάκων.

```
1: int k-element(int k, int A[], int lb1, int ub1, int B[], int lb2, int ub2)
2: if lb1 > ub1 then
3:    if (k ≤ ub2 − lb2 + 1) then
4:       return B[lb2 + k − 1];
5:    else
6:       return -1;
7:    end if
8: end if
9: if lb2 > ub2 then
10:   if (k ≤ ub1 − lb1 + 1) then
11:      return A[lb1 + k − 1];
12:   else
13:      return -1;
14:   end if
15: end if
16: if k > 1 then
17:   k1 ← ⌊k/2⌋;      pos_elem_A ← min{lb1 + k1 − 1, ub1};
18:   k2 ← k − k1;   pos_elem_B ← min{lb2 + k2 − 1, ub2};
19:   if A[pos_elem_A] ≤ B[pos_elem_B] then
20:      lb1 ← pos_elem_A + 1;   k ← k − k1;
21:   else
22:      lb2 ← pos_elem_B + 1;   k ← k − k2;
23:   end if
24:   return k-element(k, A, lb1, ub1, B, lb2, ub2);
25: else
26:   if A[lb1] ≤ B[lb2] then
27:      return A[lb1];
28:   else
29:      return B[lb2];
30:   end if
31: end if
```

Σε κάθε κλήση γίνεται σταθερός αριθμός πράξεων, έστω c. Επίσης, κάθε φορά μένουμε με ένα πρόβλημα μεγέθους $k/2$. Ο αριθμός των στοιχειωδών πράξεων δίνεται από την αναδρομική σχέση

$$T(k) = T(\frac{k}{2}) + c$$

Η παραπάνω σχέση εμπίπτει στη δεύτερη περίπτωση του κεντρικού θεωρήματος αφού $a = 1, b = 2$ και $f(k) = c = \Theta(k^{\lg 1}) = \Theta(1)$. Επομένως, $T(k) = \Theta(k^{\lg 1} \lg k) = \Theta(\lg k)$. Επειδή, δε $k \leq m + n$, έχουμε

$$T(k) = O(\lg(m + n)).$$

Άσκηση 11 *Η διάμεσος μίας λίστας αριθμών είναι ο αριθμός από την λίστα με την ιδιότητα οι μισοί από τους αριθμούς (της λίστας) να είναι μικρότεροι-ίσοι από αυτόν και οι υπόλοιποι μεγαλύτεροι-ίσοι. Να εκπονήσετε αποτελεσματικό αλγόριθμο που να υπολογίζει τη διάμεσο.*

Άσκηση 12 *Ένας πίνακας $A[1, \ldots, n]$ 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 αποθηκευμένα 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 $A[i] > A[j]$.*

Λύση 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 δυο 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.

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 ← ⌊(ub−lb+1)/2⌋;
 6:    k ← ⌊(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
```

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

$$T(n) \leq B(n)$$

$$B(n) = 2B(\frac{n}{2}) + cn.$$

Σύμφωνα με το κεντρικό θεώρημα (δεύτερη περίπτωση) $B(n) = \Theta(n \lg n)$ και επομένως

$$T(n) = O(n \lg n)$$

Algorithm 13 Πλήθος 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;
```