Αλγόριθμοι Ταξινόμησης

Περιεχόμενα Κεφαλαίου

Η **ταξινόμηση** (sorting) τοποθετεί ένα σύνολο κόμβων ή εγγραφών σε μία ιδιαίτερη σειρά (αύξουσα ή φθίνουσα) με βάση την τιμή του (πρωτεύοντος) κλειδιού της εγγραφής. Σκοπός της ταξινόμησης είναι, στη συνέχεια, η διευκόλυνση της αναζήτησης των στοιχείων του αντίστοιχου συνόλου. Για παράδειγμα, είναι γνωστό ότι η αναζήτηση ενός κλειδιού σε μη ταξινομημένο πίνακα με n εγγραφές γίνεται σειριακά με συγκρίσεις της τάξης O(n), ενώ σε ταξινομημένο πίνακα η δυαδική αναζήτηση απαιτεί κόστος της τάξης O($\log n$). Η χρησιμότητα της ταξινόμησης αποδεικνύεται στην πράξη σε περιπτώσεις αναζήτησης αριθμητικών ή αλφαβητικών δεδομένων, όπως σε βιβλιοθηκονομικά συστήματα, λεξικά, τηλεφωνικούς καταλόγους, καταλόγους φόρου εισοδήματος και γενικά παντού, όπου γίνεται αναζήτηση αποθηκευμένων αντικείμενων. Στη συνέχεια, δίνεται ένας τυπικός ορισμός της ταξινόμησης.

Ορισμός.
Δοθέντων των στοιχείων $a_1, a_2, \ldots, , a_n$ η ταξινόμηση συνίσταται στη διάταξη (permutation) της θέσης των στοιχείων, ώστε να τοποθετηθούν σε μία σειρά $a_{k_1}, a_{k_2}, \ldots, a_{k_n}$, έτσι ώστε δοθείσης μίας **συνάρτησης διάταξης** (ordering function), f, να ισχύει:

$$f(a_{k_1}) \leq f(a_{k_2}) \leq \ldots \leq f(a_{k_n})$$

□

Αξίζει να σημειωθεί ότι η προηγούμενη συνάρτηση διάταξης μπορεί να τροποποιηθεί, ώστε να καλύπτει και την περίπτωση, όπου η ταξινόμηση γίνεται με φθίνουσα τάξη (descending sequence) μεγέθους του κλειδιού. Στη γενικότερη περίπτωση μπορεί να θεωρηθεί ότι η ταξινόμηση στηρίζεται σε δύο ή περισσότερα κλειδιά της εγγραφής. Για παράδειγμα, σε πολλά παιχνίδια της τράπουλας οι παίκτες ταξινομούν τα χαρτιά τους με πρώτο κλειδί το χρώμα του φύλλου (πχ. μπαστούνια, σπαθιά, καρό και κούπες) και με δεύτερο κλειδί την αξία του φύλλου (πχ. Άσσος, Ρήγας, Ντάμα, Βαλές, 10, ..., 2).

Όπως γνωρίζουμε από το μάθημα των Δομών Δεδομένων, το αντικείμενο της ταξινόμησης είναι πολύ πλούσιο, καθώς μπορεί να εφαρμοσθεί σε πολλά περιβάλλοντα (όπως **εσωτερική** (internal) ταξινόμηση στην κύρια μνήμη ή **εξωτερική** (external) ταξινόμηση στη δευτερεύουσα μνήμη) και υπό πολλές συνθήκες (όπως **επιτοπίως** (in situ, in place) ή με επιπλέον χώρο, ως προς τη στατιστική κατανομή των κλειδιών προς ταξινόμηση κλπ.). Στο κεφάλαιο αυτό θα εξετάσουμε εκ νέου τους γνωστούς αλγορίθμους εσωτερικής ταξινόμησης, ώστε να αποδείξουμε και τυπικά τις (γνωστές εξάλλου) εκφράσεις των πολυπλοκοτήτων τους. Αρχικά, θα εξετάσουμε μεθόδους ταξινόμησης που βασίζονται στη σύγκριση, ενώ στη συνέχεια θα μελετήσουμε άλλες μεθόδους. Επίσης,

θα απαντήσουμε στο ενδιαφέρον ερώτημα: «Πόσο γρήγορα μπορεί να γίνει η ταξινόμηση», με σκοπό να αποδείξουμε το θεωρητικό όριο της πολυπλοκότητας των αλγορίθμων ταξινόμησης.

8.1 Ταξινόμηση με Εισαγωγή

Αρχικά, θα εξετάσουμε τον αλγόριθμο της ταξινόμησης με εισαγωγή και θα καταλάβουμε τη συμπεριφορά του σε δύο περιπτώσεις: δίνοντας στην είσοδο τους πίνακες A1=[1,2,3,4,5,6] και A2=[6,5,4,3,2,1], με στοιχεία που είναι ήδη ταξινομημένα ή αντίστροφη σειρά ταξινομημένα. Παρατηρούμε ότι οι περιπτώσεις αυτές είναι ακραίες και δεν μπορούν να χαρακτηρισθούν ως η μέση περίπτωση, η περίπτωση δηλαδή όπου τα στοιχεία του πίνακα εμφανίζονται με μία τυχαία σειρά. Με την ίδια προσέγγιση θα ασχοληθούμε με την ταξινόμηση με επιλογή στο αμέσως επόμενο Κεφάλαιο 8.2. Ακολουθεί ο ψευδοκώδικας για τον αλγόριθμό μας.

```
     procedure insert
1.   for i <-- 2 to n do
2.      x <-- A[i];
3.      j <-- i-1;
4.      while j>0 and x<A[j] do
5.         A[j+1] <-- A[j]
6.         j <-- j-1
7.      A[j+1] <-- x
```

Η μέθοδος αυτή διακρίνει τον πίνακα σε δύο τμήματα: την ακολουθία πηγής (source sequence) και την ακολουθία προορισμού (destimation sequence). Δηλαδή, λαμβάνει στοιχεία από την ακολουθία πηγής και τα κατευθύνει στη σωστή θέση στην ακολουθία προορισμού. Έτσι, κάθε φορά το μέγεθος της ακολουθίας πηγής μειώνεται κατά ένα, ενώ το μέγεθος της ακολουθίας προορισμού αυξάνεται κατά ένα. Το κάθε φορά λαμβανόμενο στοιχείο είναι πρώτο της ακολουθίας πηγής (δες την εντολή 3). Το στοιχείο αυτό συγκρίνεται με τα στοιχεία της ακολουθίας προορισμού αρχίζοντας από το τέλος και συνεχίζοντας προς την αρχή μέχρι να εντοπίσει την κατάλληλη θέση. Ένα παράδειγμα του αλγορίθμου απεικονίζεται στο Σχήμα 8.1.

Πρόταση.
Η πολυπλοκότητα της ταξινόμησης με εισαγωγή για την καλύτερη, τη μέση και

PLACEHOLDER FOR ch8_insertionsort

Σχήμα 8.1: Παράδειγμα Ταξινόμησης με Εισαγωγή

τη χειρότερη περίπτωση είναι $\Theta(n)$, $\Theta(n^2)$ και $\Theta(n^2)$, αντιστοίχως.

Απόδειξη
Αν στην είσοδο θεωρηθεί ο πίνακας Α1, τότε η ταξινόμηση συμπεριφέρεται πολύ αποτελεσματικά και δεν εισέρχεται μέσα στο σώμα της εντολής 4. Έτσι, ουσιαστικά το κάθε φορά λαμβανόμενο στοιχείο από την ακολουθία πηγής συγκρίνεται με ένα μόνο στοιχείο (όπως προκύπτει από τη συνθήκη ελέγχου 4). Θεωρώντας αυτή τη σύγκριση ως την πράξη βαρόμετρο, καταλήγουμε ότι για ένα πίνακα μεγέθους n στοιχείων, ο αντίστοιχος αριθμός συγκρίσεων είναι $n-1$. Επομένως, προκύπτει ότι στην καλύτερη περίπτωση η πολυπλοκότητα είναι γραμμική $\Theta(n)$.

Τώρα ας θεωρήσουμε την περίπτωση όπου στην είσοδο δίνεται ο πίνακας Α2. Το κάθε φορά επιλεγόμενο στοιχείο είναι μικρότερο από τα ήδη τοποθετημένα στην ακολουθία προορισμού. Έτσι, εκτελείται ένας συγκεκριμένος αριθμός συγκρίσεων μέχρι να τοποθετηθεί το νέο στοιχείο στη σωστή θέση, δηλαδή στην πρώτη θέση της ακολουθίας προορισμού. Έτσι, η εντολή 5 εκτελείται i φορές, όπου το i μεταβάλλεται από 1 μέχρι $n-1$. Συνεπώς και πάλι θεωρώντας την πράξη αυτή ως βαρόμετρο, καταλήγουμε ότι για ένα πίνακα μεγέθους n στοιχείων, ο αντίστοιχος αριθμός συγκρίσεων είναι $\sum_{i=1}^{n-1} i = n(n-1)/2$. Επομένως, η πολυπλοκότητα στη χειρότερη περίπτωση είναι τετραγωνική $\Theta(n^2)$.

Το σημαντικότερο ερώτημα προκύπτει στην περίπτωση ταξινόμησης τυχαίων δεδομένων εισόδου, δηλαδή στην περίπτωση, όπου οποιαδήποτε από τις $n!$ διατάξεις των δεδομένων εισόδου μπορεί να εμφανισθεί με ίση πιθανότητα $1/n!$. Η επίδοση αυτή μπορεί να προκύψει θεωρώντας ότι το στοιχείο που κάθε φορά λαμβάνεται από την ακολουθία πηγής θα διανύσει το μισό δρόμο μέχρι την αρχή της ακολουθίας προορισμού. Όπως προηγουμένως, και πάλι θα προκύψει ότι στη μέση αυτή περίπτωση η επίδοση περιγράφεται από μία τετραγωνική συνάρτηση, επομένως και στην περίπτωση αυτή η πολυπλοκότητα είναι $\Theta(n^2)$. □

8.2 Ταξινόμηση με Επιλογή

Από το Κεφάλαιο 1.5 αντιγράφουμε τον ψευδοκώδικα της ταξινόμησης με επιλογή.

```
     procedure select
1.   for i <-- 1 to n-1 do
2.      minj <-- i; minx <-- A[i];
3.      for j <-- i+1 to n do
4.         if A[j]<minx then
5.            minj <-- j
6.            minx <-- A[j]
7.      A[minj] <-- A[i]
8.      A[i] <-- minx
```

Πρόταση.
Η πολυπλοκότητα της ταξινόμησης με επιλογή είναι $\Theta(n^2)$ σε κάθε περίπτωση.

Απόδειξη
Όπως φαίνεται στον ψευδοκώδικα, ουσιαστικά ο αλγόριθμος της ταξινόμησης με επιλογή σαρώνει τον πίνακα, ώστε να εντοπίσει το μικρότερο στοιχείο και να το τοποθετήσει στην πρώτη θέση. Στη συνέχεια, περιορίζει την αναζήτηση στα υπόλοιπα $n-1$ στοιχεία, ώστε να εντοπίσει το μικρότερο μεταξύ αυτών και να το τοποθετήσει στη δεύτερη θέση κοκ. Επομένως, εύκολα προκύπτει ότι η εντολή 4 (δηλαδή, `if A[j]<minx`) θα εκτελεσθεί τον ίδιο αριθμό επαναλήψεων ανεξαρτήτως του περιεχομένου του πίνακα εισόδου. Είτε, λοιπόν, στην είσοδο δοθεί ο πίνακας A1 είτε ο A2, το αντίστοιχο πλήθος συγκρίσεων θα είναι: $\sum_{i=1}^{n-1} i = n(n-1)/2$. Επομένως, σε κάθε περίπτωση η πολυπλοκότητα είναι τετραγωνική $\Theta(n^2)$. □

Από το παράδειγμα αυτό εξάγεται το συμπέρασμα ότι η ταξινόμηση με επιλογή έχει την ίδια επίδοση (δηλαδή, τετραγωνική πολυπλοκότητα) ανεξαρτήτως των δεδομένων εισόδου. Έτσι, επιβεβαιώνουμε κάτι που αναφέραμε στο Κεφάλαιο 2, δηλαδή ότι ο αλγόριθμος αυτός είναι **σταθερός** (robust), επειδή έχει ταυτόσημες την καλύτερη, χειρότερη και μέση περίπτωση. Αντίθετα, δεν συμβαίνει το ίδιο με τον αλγόριθμο ταξινόμησης με εισαγωγή.

Καθώς η εξέταση του πλήθους των συγκρίσεων ήταν εύκολη υπόθεση, στη συνέχεια θα εξετάσουμε τον αλγόριθμο αυτό, ώστε να προσδιορίσουμε το πλήθος των καταχωρήσεων. Καθώς υπάρχουν αρκετές καταχωρήσεις στον αλγόριθμο, επιλέγουμε ως σημαντικότερη την καταχώρηση `minx <-- A[j]` της εντολής 6 εντός του εσωτερικού βρόχου, παρά τις καταχωρήσεις που εκτελούνται εκτός του εσωτερικού αλλά εντός του εξωτερικού βρόχου. Εξάλλου είναι εύκολο να διαπιστωθεί ότι οι καταχωρήσεις στις εντολές `minj <-- i; minx <-- A[i]; A[minj] <-- A[i]` και `A[i] <-- minx` (εντολές 2,

7, 8) θα εκτελεσθούν συνολικά $4(n-1)$ φορές.

Πρόταση.
Η μέση τιμή του πλήθους των καταχωρήσεων κατά την ταξινόμηση με επιλογή είναι $\Theta(n \log n)$.

Απόδειξη
Ας υποθέσουμε ότι τα στοιχεία του πίνακα είναι διακριτά και ότι κάθε μία από τις $n!$ διατάξεις είναι ισοπίθανες να συμβούν. Στην i-οστή επανάληψη ο αλγόριθμος αναζητά το μικρότερο μεταξύ των $n-i+1$ στοιχείων του πίνακα στα δεξιά της θέσης i (συμπεριλαμβανομένης της θέσης αυτής). Έστω ότι με $R(m)$ συμβολίζουμε το πλήθος των καταχωρήσεων που απαιτούνται για την επιλογή του ελάχιστου μεταξύ των m στοιχείων, οπότε για τη μέση τιμή του $R(m)$ ισχύει:

$$R(m) \;=\; S(m)/m!$$

όπου $S(m)$ είναι το πλήθος των καταχωρήσεων που αθροίζονται κατά την εκτέλεση των $m!$ διαφορετικών διατάξεων των m στοιχείων.

Ας θεωρήσουμε τις $(m-1)!$ περιπτώσεις, όπου το ελάχιστο στοιχείο εντοπίζεται στην τελευταία θέση του πίνακα. Στην περίπτωση αυτή το πλήθος των καταχωρήσεων είναι

$$S(m-1) + (m-1)!$$

καθώς ο αλγόριθμος συμπεριφέρεται σαν να υπήρχαν αρχικά $(m-1)$ μη επαναλαμβανόμενα στοιχεία, οπότε στη συνέχεια θα εκτελούνταν άλλη μία καταχώρηση για κάθε μία από τις $(m-1)!$ διαφορετικές διατάξεις. Αν εξαιρεθεί η τελευταία περίπτωση (δηλαδή, το ελάχιστο βρίσκεται στην τελευταία θέση), τότε απομένουν προς εξέταση $(m-1)(m-1)!$ περιπτώσεις. Στις περιπτώσεις αυτές το ελάχιστο στοιχείο βρίσκεται από την πρώτη μέχρι την $(m-1)$-οστή θέση, οπότε για κάθε ομάδα $(m-1)!$ διατάξεων απαιτούνται $S(m-1)$ καταχωρήσεις. Αυτό συνολικά δίνει άλλες

$$(m-1)S(m-1)$$

καταχωρήσεις. Έτσι, προκύπτει η αναδρομική εξίσωση:

$$S(m) \;=\; mS(m-1) + (m-1)!$$

με αρχική συνθήκη $S(1) = 0$. Άρα, για τη μέση τιμή του $R(m)$ ισχύει:

$$R(m) \;=\; R(m-1) + 1/m \qquad\qquad m > 1$$

που επιλύεται εύκολα και δίνει τη λύση:

$$R(m) \;=\; \sum_{k=2}^{m} \frac{1}{k} \;=\; H_m - 1$$

Λαμβάνοντας υπ'όψιν το σύνολο των $n-1$ επαναλήψεων του εξωτερικού βρόχου (εντολή 1) προκύπτει ότι η μέση τιμή του πλήθους των καταχωρήσεων είναι:

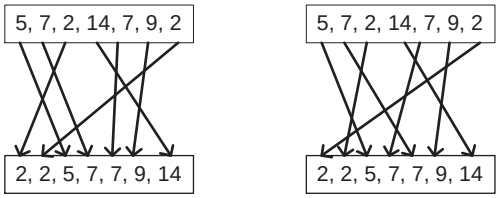
$$\begin{aligned} T(n) &= \sum_{i=1}^{n-1} (H_{n-i+1} - 1) \;=\; \sum_{i=2}^{n} (H_i - 1) \\ &= \left(\frac{1}{2}\right) + \left(\frac{1}{2} + \frac{1}{3}\right) + \ldots + \left(\frac{1}{2} + \frac{1}{2} + \ldots + \frac{1}{n}\right) \\ &= \sum_{k=2}^{n} (n-k+1)\frac{1}{k} \;=\; \ldots \;=\; (n+1)H_n - 2n \end{aligned}$$

Άρα, προκύπτει ότι στη μέση περίπτωση το πλήθος των καταχωρήσεων είναι της τάξης $\Theta(n \log n)$. Ακόμη και αν προσθέταμε τις $4(n-1)$ καταχωρήσεις που εκτελούνται εκτός του εσωτερικού βρόχου, και πάλι το αποτέλεσμα δε θα άλλαζε. □

Ένα τελικό σχόλιο. Μία μέθοδος ταξινόμησης λέγεται **ευσταθής** (stable), αν διατηρεί τη σχετική θέση πριν και μετά τη διάταξη για τα στοιχεία με την ίδια τιμή. Σε αντίθετη περίπτωση λέγεται **ασταθής**. Στο Σχήμα 8.2 παρουσιάζεται μία περίπτωση ευσταθούς και μία περίπτωση ασταθούς αλγορίθμου. Η ταξινόμηση με εισαγωγή είναι ευσταθής, αν και η μέθοδος αυτή έχει κακή σταθερή πολυπλοκότητα $\Theta(n^2)$. Για την περίπτωση της γρήγορης ταξινόμησης που θα μελετήσουμε στη συνέχεια, θα παρατηρήσουμε ότι η τελευταία αυτή μέθοδος δεν είναι ευσταθής, καθώς εκτελεί περιττές μετακινήσεις ίσων κλειδιών. Παρ' όλ' αυτά δεν παύει να θεωρείται μία πολύ αποτελεσματική μέθοδος.

8.3 Γρήγορη Ταξινόμηση

Η **γρήγορη ταξινόμηση** (quicksort), που αλλιώς ονομάζεται και ταξινόμηση με **διαμερισμό και ανταλλαγή** (partition exchange sort), θεωρείται ένας από τους top-10 αλγορίθμους ως προς την πρακτική χρησιμότητά και το ενδιαφέρον που προκάλεσε από τη θεωρητική σκοπιά. Η γρήγορη ταξινόμηση αποτελεί ένα κλασικό παράδειγμα αλγορίθμου της οικογενείας *Διαίρει και Βασίλευε*.



Σχήμα 8.2: (i) Ευσταθής και (ii) Ασταθής Ταξινόμηση

Όπως δηλώνει το πρώτο όνομα του αλγορίθμου, πρόκειται για μία πολύ αποτελεσματική μέθοδο ταξινόμησης (πλην ελαχίστων εξαιρέσεων). Όπως δηλώνει το δεύτερο όνομά του, τα βασικά χαρακτηριστικά του είναι δύο: η ανταλλαγή και ο διαμερισμός. Πρώτον, δηλαδή, εκτελούνται ανταλλαγές μεταξύ των στοιχείων του πίνακα, έτσι ώστε τα στοιχεία με μικρότερες τιμές να μετακινηθούν προς τη μία πλευρά, ενώ τα στοιχεία με μεγαλύτερες τιμές να μετακινηθούν προς την άλλη πλευρά του πίνακα. Έτσι, επακολουθεί η εφαρμογή του δεύτερου χαρακτηριστικού, δηλαδή του διαμερισμού του πίνακα σε δύο μικρότερους που ταξινομούνται ανεξάρτητα μεταξύ τους. Είναι προφανές ότι στους δύο υποπίνακες επιφυλάσσεται η ίδια αντιμετώπιση: ανταλλαγή και διαμερισμός.

```
     procedure quicksort(left,right);
1.   if left<right then
2.      i <-- left; j <-- right+1; pivot <-- A[left];
3.      repeat
4.         repeat i <-- i+1 until A[i]>=pivot;
5.         repeat j <-- j-1 until A[j]<=pivot;
6.         if i<j then swap(A[i],A[j]);
7.      until j<=i;
8.      swap(A[left],A[j]);
9.      quicksort(left,j-1);
10.     quicksort(j+1,right)
```

Η προηγούμενη διαδικασία `quicksort` υποθέτει ότι δίνεται προς ταξινόμηση ένας πίνακας `A[1..n]`, που περιέχει ακεραίους αριθμούς. Ο `pivot` είναι ένα στοιχείο του πίνακα, το οποίο παίρνει την τελική του θέση στον πίνακα, ώστε κατόπιν να γίνει ο διαμερισμός (δηλαδή, να γίνουν οι κλήσεις

της `partition`) σε δύο υποπίνακες με μικρότερα και μεγαλύτερα στοιχεία από τον `pivot` αντίστοιχα. Η επιλογή του `pivot` μπορεί να γίνει κατά πολλούς τρόπους. Στο σημείο αυτό δεχόμαστε ότι ως `pivot` λαμβάνεται το πρώτο στοιχείο του πίνακα. Έτσι, ο πίνακας σαρώνεται από τα αριστερά προς τα δεξιά αναζητώντας το πρώτο στοιχείο με τιμή μεγαλύτερη ή ίση με την τιμή του `pivot`. Ύστερα, ο πίνακας σαρώνεται από τα δεξιά προς τα αριστερά αναζητώντας το πρώτο στοιχείο με τιμή μικρότερη ή ίση με την τιμή του `pivot`. Όταν εντοπισθούν δύο τέτοια στοιχεία, τότε αυτά ανταλλάσσονται, ώστε κατά το δυνατόν να τείνουν να πλησιάσουν προς την τελική τους θέση. Κατόπιν, συνεχίζουμε τις σαρώσεις προσπαθώντας να εντοπίσουμε άλλα παρόμοια ζεύγη και να τα ανταλλάξουμε. Κάποια στιγμή οι δύο δείκτες σάρωσης διασταυρώνονται. Τότε η σάρωση σταματά και ο `pivot` λαμβάνει την τελική του θέση εκτελώντας άλλη μία ανταλλαγή μεταξύ του `pivot` και του στοιχείου όπου σταμάτησε ο δείκτης της σάρωσης από δεξιά. Έτσι, η διαδικασία μπορεί να συνεχίσει στους δύο υποπίνακες κατά τα γνωστά. Το επόμενο σχήμα δείχνει τα βήματα του κώδικα μέχρι τον πρώτο διαμερισμό.

Αρχικά κλειδιά	52	12	71 $i\uparrow$	56	5	10	19	90	45 $j\uparrow$
1η ανταλλαγή	52	12	45	56 $i\uparrow$	5	10	19 $j\uparrow$	90	71
2η ανταλλαγή Διασταύρωση δεικτών	52	12	45	19	5	10 $j\uparrow$	56 $i\uparrow$	90	71
Ανταλλαγή και διαμερισμός	[10	12	45	19	5]	52	[56	90	71]

Σχήμα 8.3: Διαδικασία διαμερισμού.

Τώρα ας προχωρήσουμε στην εύρεση της πολυπλοκότητας του αλγορίθμου. Κατ'αρχάς θα εξετάσουμε τη χειρότερη περίπτωση που είναι ευκολότερη, και στη συνέχεια την καλύτερη και τη μέση περιπτωση.

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n^2)$ στη χειρότερη περίπτωση.

Απόδειξη

Η περίπτωση αυτή συμβαίνει όταν σε πίνακα μεγέθους n, το επιλεγόμενο στοιχείο ως `pivot` είναι το μικρότερο ή το μεγαλύτερο, ώστε τα μεγέθη των δύο υποπινάκων που θα προκύψουν να είναι 0 και $n-1$ (αφού η μία θέση του συγκεκριμένου πίνακα θα καταληφθεί από τον ίδιο τον `pivot`). Έτσι, εύκολα προκύπτει η ακόλουθη αναδρομική εξίσωση:

$$T(n) \;=\; T(n-1) + n$$

Ο όρος n του δεξιού σκέλους αντιστοιχεί στις συγκρίσεις που θα εκτελεσθούν κατά τη σάρωση του πίνακα (εντολές 4-5)[1]. Η ανωτέρω αναδρομική εξίσωση είναι εύκολη υπόθεση, καθώς διαδοχικά ισχύει:

$$\begin{aligned} T(n-1) \;&=\; T(n-2) + (n-1) \\ T(n-2) \;&=\; T(n-3) + (n-2) \\ &\vdots \\ T(2) \;&=\; T(1) + 2 \end{aligned}$$

Αθροίζοντας τα αριστερά και δεξιά σκέλη αυτών των εξισώσεων και θεωρώντας ότι $(0) = (1) = 0$, προκύπτει ότι:

$$T(n) \;=\; T(1) + \sum_{i=2}^{n} i \;=\; n(n+1)/2 - 1 \;=\; \Theta(n^2)$$

□

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$ στην καλύτερη περίπτωση.

Απόδειξη
Για να αναλύσουμε την καλύτερη περίπτωση αρκεί να διαπιστώσουμε ότι αυτή συμβαίνει, όταν κάθε φορά ως `pivot` επιλέγεται ένα στοιχείο που λαμβάνοντας την τελική του θέση υποδιαιρεί τον πίνακα σε δύο υποπίνακες ίσου μεγέθους. Επομένως, ισχύει η αναδρομική εξίσωση:

$$T(n) \;=\; T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

[1] Στην πραγματικότητα είναι δυνατό να εκτελεσθούν περισσότερες από n συγκρίσεις, αλλά σε κάθε περίπτωση ο αριθμός των συγκρίσεων είναι μικρότερος από $2n$, γεγονός που δεν αλλάζει το τελικό συμπέρασμα.

με αρχικές συνθήκες $T(0) = T(1) = 0$. Για την ευκολία της ανάλυσης υποθέτουμε ότι $n = 2^k$, οπότε:

$$T(n) \;=\; 2T(n/2) + n \;\;\Rightarrow\;\; \frac{T(n)}{n} \;=\; \frac{T(n/2)}{n/2} + 1$$

Η τελευταία αναδρομική εξίσωση είναι πλέον εύκολη υπόθεση, καθώς ισχύει διαδοχικά:

$$\begin{aligned} \frac{T(n/2)}{n/2} \;&=\; \frac{T(n/4)}{n/4} + 1 \\ &\vdots \\ \frac{T(2)}{2} \;&=\; \frac{T(1)}{1} + 1 \end{aligned}$$

Αθροίζοντας τα αριστερά και δεξιά σκέλη αυτών των εξισώσεων προκύπτει ότι:

$$\frac{T(n)}{n} \;=\; \frac{T(1)}{1} + \log n \;\Rightarrow\; T(n) \;=\; n + n\log n \;=\; \Theta(n\log n)$$

□

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$ στη μέση περίπτωση.

Απόδειξη
Για να εξετάσουμε τη μέση περίπτωση θα υποθέσουμε ότι για n στοιχεία, κάθε διάταξη των στοιχείων αυτών (όπου $n!$ είναι το πλήθος των διατάξεων) είναι ισοπίθανη. Επομένως, ο κάθε φορά επιλεγόμενος `pivot` τελικά επιλέγεται με τυχαίο τρόπο. Θα χειρισθούμε τη μέση περίπτωση με τη βοήθεια της επόμενης αναδρομικής εξίσωσης για $n \geq 2$:

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

με αρχικές συνθήκες $T(0) \;=\; T(1) \;=\; 0$. Λόγω συμμετρίας και των αρχικών συνθηκών, η προηγούμενη αναδρομική εξίσωση μετασχηματίζεται σε:

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

Τώρα, τη σχέση αυτή πολλαπλασιάζουμε επί n και προκύπτει:

$$n\,T(n) \;=\; n^2 + 2\sum_{k=2}^{n-1} T(k)$$

Επίσης, στην ίδια σχέση αντικαθιστούμε το n με $n-1$ και πολλαπλασιάζουμε με $n-1$, οπότε προκύπτει

$$(n-1)\,T(n-1) \;=\; (n-1)^2 + 2\sum_{k=2}^{n-2} T(k)$$

Αφαιρώντας τα αντίστοιχα σκέλη των δύο τελευταίων εκφράσεων και εκτελώντας απλή άλγεβρα, προκύπτει:

$$n\,T(n) - (n-1)\,T(n-1) \;=\; 2n-1+2T(n-1) \;\Rightarrow$$

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

Με διαδοχικές αντικαταστάσεις του n με $n-1$ προκύπτει:

$$\begin{aligned}
\frac{T(n-1)}{n} &= \frac{T(n-2)}{n-1} + \frac{2n-3}{(n-1)n} \\
\frac{T(n-2)}{n-1} &= \frac{T(n-3)}{n-2} + \frac{2n-5}{(n-2)(n-1)} \\
&\vdots \\
\frac{T(2)}{3} &= \frac{T(1)}{2} + \frac{3}{2\times 3}
\end{aligned}$$

Αθροίζοντας αντίστοιχα τα αριστερά και τα δεξιά σκέλη και απλοποιώντας προκύπτει:

$$\frac{T(n)}{n+1} \;=\; \frac{2n-1}{n(n+1)} + \frac{2n-3}{(n-1)n} + \ldots + \frac{3}{2\times 3} \;=\; \ldots \;=\; \sum_{i=2}^{n}\left(\frac{3}{i+1} - \frac{1}{i}\right)$$

Επομένως, πρέπει να υπολογίσουμε το άθροισμα του δεξιού σκέλους:

$$\sum_{i=2}^{n}\left(\frac{3}{i+1} - \frac{1}{i}\right) \;=\; 3\left(H_{n+1} - 1 - \frac{1}{2}\right) - (H_n - 1) \;=\; \ldots \;=\; 2H_n - \frac{7}{2} + \frac{2}{n+1}$$

Τελικώς, έχουμε:

$$\frac{T(n)}{n+1} \;=\; 2H_n - \frac{7}{2} + \frac{2}{n+1} \;\Rightarrow$$

$$T(n) \;=\; 2(n+1)H_n - \frac{7}{2}(n+1) + 2 = \Theta(n \log n)$$

□

Η γρήγορη ταξινόμηση δεν είναι επιτόπιος αλγόριθμος, δηλαδή δεν αρκείται στο χώρο του συγκεκριμένου πίνακα, αλλά χρειάζεται επιπλέον χώρο. Ο χώρος αυτός δεν φαίνεται καθαρά στον ανωτέρω κώδικα αλλά απαιτείται από το μεταγλωττιστή κατά την αναδρομή. Εύλογα προκύπτει η ερώτηση σχετικά με το μέγεθος αυτού του απαιτούμενου χώρου. Στη χειρότερη περίπτωση το μέγιστο βάθος της αναδρομής θα είναι $n-1$ κλήσεις, αριθμός υπερβολικός. Για το λόγο αυτό έχει σχεδιασθεί μία επαναληπτική εκδοχή της γρήγορης ταξινόμησης, δηλαδή μία εκδοχή που υλοποιεί την απαραίτητη στοίβα, όπου τοποθετούνται τα όρια του μικρότερου υποπίνακα, όπως φαίνεται στην επόμενη διαδικασία `iterative_quicksort`.

```
     procedure iterative_quicksort(left,right);
1.   repeat
2.      while (left<right) do
3.         i <-- left; j <-- r+1; pivot <-- A[left];
4.         repeat
5.            repeat i <-- i+1 until A[i]>=pivot;
6.            repeat j <-- j-1 until A[j]<=pivot;
7.            if i<j then swap(A[i],A[j]);
8.         until j<=i;
9.         swap(A[left],A[j]);
10.        if ((j-p)<(q-j)) then
11.           push(j+1); push(q); q <-- j-1
12.        else
13.           push(p); push(j-1); p <-- j+1
14.     if stack is empry then return
15.     pop(q); pop(p)
16.  until (false)
```

Πρόταση.
Η χωρική πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$ στη μέση περίπτωση.

Απόδειξη

Ο απαιτούμενος χώρος εκφράζεται από την αναδρομική εξίσωση:

$$S(n) \leq \begin{cases} 2 + S(\lfloor n-1 \rfloor/2) & n > 1 \\ 0 & n \leq 1 \end{cases}$$

Επομένως, υποθέτοντας ότι $n = 2^k$, αρκεί να επιλύσουμε την εξίσωση:

$$\begin{aligned} S(n) &= S(n/2) + 2 = (S(n/4) + 2) + 2 = S(n/4) + 4 = \ldots = \\ &= S(1) + 2k = 2\log n = \Theta(\log n) \end{aligned}$$

□

8.4 Στατιστικά Διάταξης

Με τον όρο **στατιστικά διάταξης** (order statistics) εννοούμε την περίπτωση, όπου δεδομένου ενός πίνακα A με n αταξινόμητα στοιχεία, πρέπει να αναζητήσουμε το μικρότερο, το μεγαλύτερο, το μεσαίο ή γενικά το k-οστό στοιχείο του πίνακα με βάση την τιμή του κλειδιού του. Παρά το γεγονός ότι το αντικείμενο είναι πρόβλημα αναζήτησης και όχι ταξινόμησης, θα το εξετάσουμε στο σημείο αυτό για λόγους που θα φανούν στη συνέχεια.

Ας αρχίσουμε από τα εύκολα, δηλαδή έστω ότι θέλουμε να βρούμε ταυτόχρονα το μικρότερο και το μεγαλύτερο στοιχείο ενός πίνακα. Η προφανής λύση είναι να εκτελέσουμε δύο σαρώσεις, μία σάρωση για την εύρεση του μικρότερου και στη συνέχεια μία σάρωση για την εύρεση του μεγαλύτερου. Είναι ευνόητο ότι η πολυπλοκότητα αυτής της πρώτης προσέγγισης είναι $\Theta(n)$. Με τη διαδικασία που ακολουθεί προσπαθούμε με μία μόνο σάρωση να επιτύχουμε την ταυτόχρονη εύρεση των δύο στοιχείων που μας ενδιαφέρουν.

```
     procedure maxmin1
1.   max <-- A[1]; min <-- A[1];
2.   for i <-- 2 to n do
3.      if A[i]>max then max <-- A[i]
4.      if A[i]<min then min <-- A[i]
5.   return max, min
```

Η διαδικασία αυτή δεν παύει να είναι μία απλοϊκή προσέγγιση. Αν και γίνεται μία σάρωση του πίνακα, κάθε στοιχείο υποβάλλεται σε δύο συγκρίσεις, που όπως είπαμε είναι ένα σύνηθες βαρόμετρο. Η κατάσταση θα μπορούσε να βελτιωθεί, αν αντικαθιστούσαμε τις εντολές 3-4 με τις εντολές:

```
3.       if A[i]>max then max <-- A[i]
4.       else if A[i]<min then min <-- A[i]
```

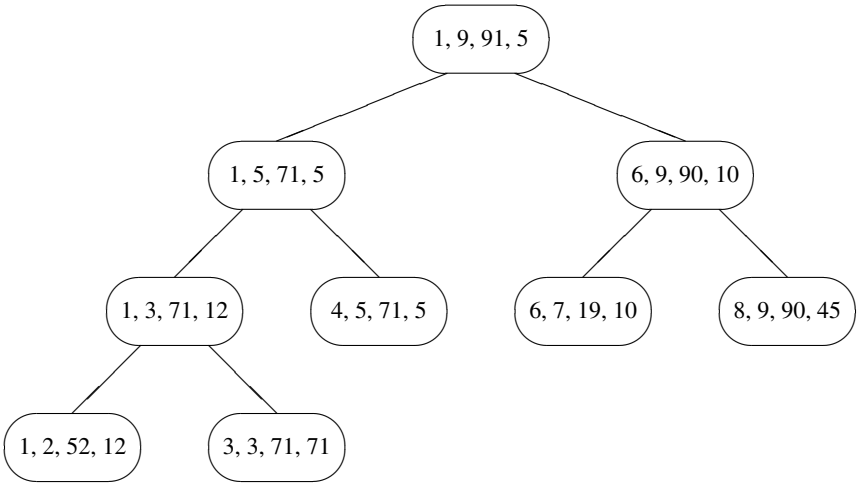
Ωστόσο, και αυτή η εκδοχή μπορεί να αποδειχθεί ότι δεν βοηθά στη χειρότερη περίπτωση. Η επόμενη διαδικασία, που προσπαθεί να εκμεταλλευθεί την τεχνική του *Διαίρει και Βασίλευε*, υποθέτει ότι με τα ορίσματα `i, j` δίνουμε τα όρια του πίνακα και με τα ορίσματα `fmax, fmin` μας επιστρέφονται οι τιμές που μας ενδιαφέρουν. Επίσης, υποθέτουμε ότι η συνάρτηση `max` επιστρέφει το μέγιστο μεταξύ δύο στοιχείων, ενώ η συνάρτηση `min` επιστρέφει το ελάχιστο μεταξύ των δύο στοιχείων.

```
     procedure maxmin2(i,j,fmax,fmin)
1.   if i=j then
2.      max <-- A[i]; min <-- A[i];
3.   else if i=j-1 then
4.      if A[i]<A[j] then
5.         fmax <-- A[j]; fmin <-- A[i];
6.      else
7.         fmax <-- A[i]; fmin <-- A[j];
8.   else
9.      middle <-- (i+j)/2;
10.     maxmin2(i,mid,gmax,gmin);
11.     maxmin2(mid+1,j,hmax,hmin);
11.     fmax <-- max(gmax,hmax);
12.     fmin <-- min(gmin,hmin);
```

Έστω ότι το περιεχόμενο του πίνακα `A` είναι τα 9 στοιχεία 52, 12, 71, 56, 5, 10, 19, 90, 45. Στο επόμενο σχήμα απεικονίζεται ένα δένδρο με κόμβους που περιέχουν τα ορίσματα των κλήσεων. Διασχίζοντας το δένδρο με προτεραιότητα κατά βάθος (dfs) μπορούμε να αποτυπώσουμε την επακριβή σειρά των διαφόρων κλήσεων.

Πρόταση.
Η πολυπλοκότητα της `maxmin2` είναι $\Theta(n)$ στη χειρότερη περίπτωση.

Απόδειξη
Η πολυπλοκότητα της διαδικασίας αυτής μπορεί να ευρεθεί επιλύοντας την



Σχήμα 8.4: Κλήσεις για τον υπολογισμό του ελάχιστου και του μέγιστου.

αναδρομική εξίσωση:

$$T(n) = \begin{cases} 0 & \text{αν } n = 1 \\ 1 & \text{αν } n = 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{αν } n > 2 \end{cases}$$

όπου οι σταθεροί όροι εκφράζουν το κόστος των συγκρίσεων στις εντολές 4 και 11-12. Θεωρώντας τη χειρότερη περίπτωση και ότι $n = 2^k$ διαδοχικά έχουμε:

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 \;=\; 4T(n/4) + 4 + 2 \\ &\vdots \\ &= 2^k + 2^{k-1} + \ldots + 2^1 \\ &= 2n - 2 \end{aligned}$$

Επομένως, παρότι ο τελευταίος αλγόριθμος είναι βελτιωμένος δεν μπορεί να υπερβεί το όριο της γραμμικής πολυπλοκότητας $\Theta(n)$. □

8.4.1 Στατιστικά σε Μέσο Γραμμικό Χρόνο

Έστω ότι επιθυμούμε να βρούμε το k-οστό στοιχείο ενός αταξινόμητου πίνακα. Αυτό μπορεί να επιτευχθεί ταξινομώντας τον πίνακα και λαμβάνοντας το πε-

ριεχόμενο της αντίστοιχης θέσης του πίνακα. Γνωρίζουμε ήδη από το αντικείμενο των Δομών Δεδομένων ότι το κάτω όριο (που θα αποδειχθεί στη συνέχεια) των αλγορίθμων ταξινόμησης που στηρίζονται σε συγκρίσεις είναι $O(n \log n)$. Επομένως, το όριο αυτό προσδιορίζει και την πολυπλοκότητα της μεθόδου που ανάγει το πρόβλημα της επιλογής σε πρόβλημα ταξινόμησης. Στη συνέχεια, θα επιλύσουμε το πρόβλημα της επιλογής του k-οστού στοιχείου με μία μέθοδο που στηρίζεται στη γρήγορη ταξινόμηση, όπως φαίνεται από τον επόμενο αλγόριθμο. Ο αλγόριθμος αυτός σχεδιάστηκε από τον C.A.R. Hoare [22]. Σημειώνεται ότι πρέπει να ισχύει: $left \leq k \leq right$.

```
     procedure find(left,right,k);
1.   if left=right then return A[left]
2.   else
3.      i <-- left; j <-- right+1; pivot <-- A[left];
4.      repeat
5.         repeat i <-- i+1 until A[i]>=pivot;
6.         repeat j <-- j-1 until A[j]<=pivot;
7.         if i<j then swap(A[i],A[j]);
8.      until j<=i;
9.      swap(A[left],A[j]);
10.     if k=j then return A[j]
11.     else if (k<j) then find(left,j-1,k)
12.     else find(j+1,right,k-j);
```

Παρατηρούμε ότι η διαδικασία `find` είναι σχεδόν ταυτόσημη με τη διαδικασία `quicksort`. Πιο συγκεκριμένα, οι εντολές 3-9 ταυτίζονται με αντίστοιχες εντολές της `quicksort`, καθώς αφορούν στις διαδικασίες επιλογής του `pivot` και του διαμερισμού. Στη συνέχεια, διαφοροποιούνται οι εντολές 10-12. Η ουσία είναι ότι καθώς ο `pivot` τοποθετείται οριστικά στη θέση `j` του πίνακα, αποφασίζουμε αν θα τερματίσουμε ή αν θα συνεχίσουμε την αναζήτηση του k-οστού στοιχείου στον αριστερό ή στο δεξιό υποπίνακα με αναδρομικό τρόπο. Η ανάλυση του αλγορίθμου είναι πλέον αρκετά προφανής με βάση όλα τα προηγούμενα.

Πρόταση.
Η πολυπλοκότητα της `find` είναι $\Theta(n^2)$, $\Theta(n)$ και $\Theta(n)$ στη χειρότερη, τη μέση και την καλύτερη περίπτωση, αντιστοίχως.

Απόδειξη

Κατ'αρχάς η χειρότερη περίπτωση είναι $\Theta(n^2)$ και αυτό θα συμβεί, όταν συνεχώς ο επιλεγόμενος `pivot` είναι το μικρότερο ή το μεγαλύτερο στοιχείο. Θα μελετήσουμε τη μέση περίπτωση για την οποία ισχύει:

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

με αρχικές συνθήκες $T(0) = T(1) = 0$. Επομένως:

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

Τώρα, τη σχέση αυτή πολλαπλασιάζουμε επί n και προκύπτει:

$$n\, T(n) \;=\; n^2 + \sum_{k=2}^{n-1} T(k)$$

Επίσης, στην ίδια σχέση αντικαθιστούμε το n με $n-1$ και πολλαπλασιάζουμε με $n-1$, οπότε προκύπτει

$$(n-1)\, T(n-1) \;=\; (n-1)^2 + \sum_{k=2}^{n-2} T(k)$$

Αφαιρώντας τα αντίστοιχα σκέλη των δύο τελευταίων εκφράσεων και εκτελώντας απλή άλγεβρα, προκύπτει:

$$\begin{aligned}
n\, T(n) - (n-1)\, T(n-1) &= 2n - 1 + T(n-1) \;\Rightarrow \\
T(n) - T(n-1) &= 2 - \frac{1}{n} \;\Rightarrow \\
T(n-1) - T(n-2) &= 2 - \frac{1}{n-1} \;\Rightarrow \\
&\vdots \\
T(2) - T(1) &= 2 - \frac{1}{2}
\end{aligned}$$

Αθροίζοντας αντίστοιχα τα αριστερά και τα δεξιά σκέλη και απλοποιώντας, προκύπτει:

$$\begin{aligned}
T(n) &= 2(n-1) - \left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \\
&= 2(n-1) + 1 - \left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \\
&= 2n - 1 - H_n
\end{aligned}$$

Καθώς για τον αρμονικό αριθμό ισχύει $H_n \approx \ln n$, έπεται ότι η πολυπλοκότητα της μέσης περίπτωσης της διαδικασίας `find` είναι $\Theta(n)$. Ομοίως, γραμμική είναι και η πολυπλοκότητα της καλύτερης περίπτωσης. □

8.4.2 Στατιστικά σε Γραμμικό Χρόνο Χειρότερης Περίπτωσης

Σε αυτή την ενότητα θα μελετήσουμε τον γραμμικό αλγόριθμο `Select` για την εύρεση του $k-$οστού μεγαλύτερου στοιχείου, ο οποίος αναπτύχθηκε από τους Blum et. al. [1]. Το πρόβλημα που είχε ο αλγόριθμος `find` ήταν ότι ο διαχωρισμός της αρχικής ακολουθίας δεν ήταν κατ' ανάγκην καλός και μπορούσε να οδηγήσει σε μεγάλα μεγέθη υποπροβλημάτων.

Ο αλγόριθμος `Select` βασίζεται στην εξής ιδέα για τον καλύτερο διαχωρισμό:

> Αυτό που χρειάζεται είναι να επιλέξουμε το στοιχείο διαχωρισμού βάσει ενός καλύτερου δείγματος. Αυτό το δείγμα μπορούμε να το χτίσουμε χωρίζοντας το M σε ομάδες π.χ. των 5 στοιχείων, να λάβουμε τους μέσους για κάθε ομάδα και με τους μέσους να κατασκευάσουμε μια μεγαλύτερη ομάδα, η οποία αποτελεί δειγματισμό των στοιχείων του M. Ο μέσος αυτής της ομάδας (μέσος των μέσων) είναι εγγυημένα καλό στοιχείο διαχωρισμού.

Η ιδέα αυτή παράγει σωστό διαχωρισμό, πέραν αυτού του βήματος όμως, ο αλγόριθμος λειτουργεί ακριβώς όπως ο `find`. Ο ψευδοκώδικας που παρουσιάζεται στη συνέχεια, υλοποιεί τον αλγόριθμο `Select`.

Η διαδικασία που εκτελείται στις γραμμές 5-8 μπορεί να φανεί στο Σχήμα 8.5. Στη γραμμή 5, χωρίζουμε το M σε ομάδες των 5 στοιχείων τις $M_1, M_2, \ldots, M_{\lceil \frac{n}{5} \rceil}$. Στη συνέχεια ταξινομούμε τις ομάδες (γραμμή 6) και βρίσκουμε τον μέσο κάθε ομάδας (έστω m_j ο μέσος της ομάδας M_j). Κάθε ομάδα απεικονίζεται με μια στήλη και η τοποθέτηση των στοιχείων στη στήλη είναι τέτοια, ώστε οι τιμές να αυξάνουν καθώς κινούμαστε προς τα πάνω σε κάθε στήλη. Στη γραμμή 8, βρίσκουμε το στοιχείο $\overline{m}$, το οποίο είναι ο μέσος όλων των m_j, $1 \leq j \leq \lceil \frac{n}{5} \rceil$, δηλαδή *ο μέσος των μέσων* $\overline{m}$ το στοιχείο που είναι εγγυημένα καλός διαχωριστής. Από το σημείο αυτό και έπειτα ο αλγόριθμος προχωρά, όπως ακριβώς και στο `find`.

Τα στοιχεία που περικλείονται από το γκρί ορθογώνιο στο Σχήμα 8.5 είναι αυτά τα οποία σίγουρα έχουν τιμή μικρότερη ίση από το $\overline{m}$ και άρα σίγουρα ανήκουν στο M_1. Αυτό μπορεί να φανεί, δεδομένου ότι τα στοιχεία του γκρίζου ορθογωνίου είναι μικρότερα από τον μέσο της στήλης στην οποία ανήκουν και

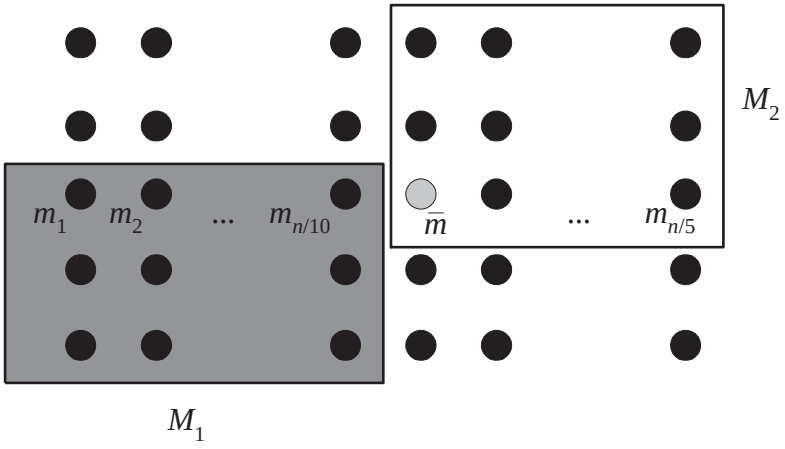
```
Select(M, i)
(* Εύρεση του i−οστού μεγαλύτερου στοιχείου του M *)
1.  n ← |M|;
2.  if n ≤ 100 then
3.      Ταξινόμηση του M και απευθείας εύρεση του i−οστού μεγαλύτερου;
4.  else
5.      Διαχωρισμός του M σε υποσύνολα M_1, M_2, ..., M_⌈n/5⌉ 5 στοιχείων το καθένα
        (το M_⌈n/5⌉, ενδέχεται να περιέχει ≤ 5 στοιχεία);
6.      Ταξινόμηση M_j, 1 ≤ j ≤ ⌈n/5⌉;
7.      Απευθείας εύρεση του μέσου m_j σε καθένα M_j;
8.      m̄ ←Select((m_1, m_2, ..., m_⌈n/5⌉), ⌈n/10⌉);
(* Εύρεση του μέσου των μέσων, έστω m̄ *)
9.      M_1 ← {m ∈ M; m̄ < s}; (* Τα μικρότερα του m̄ στοιχεία *)
10.     M_2 ← {m ∈ M; m̄ > s}; (* Τα μεγαλύτερα του m̄ στοιχεία *)
11.     if i ≤ |M_1| then
12.         Select(M_1, i);
13.     else
14.         Select(M_2, i − |M_1|);
15.     fi
16. fi
```

ο μέσος καθεμίας από της στήλες είναι μικρότερος από τον μέσο των μέσων, $\overline{m}$. Με εντελώς όμοιο συλλογισμό, τα στοιχεία του διαφανούς ορθογωνίου, είναι όλα μεγαλύτερα ή ίσα του $\overline{m}$ και ανήκουν στο M_2. Τα δε M_1, M_2, περιέχουν πιθανώς και άλλα στοιχεία, αλλά αυτά που περιβάλλονται από ορθογώνια, περιέχονται σίγουρα σε αυτά τα σύνολα.

Φράσσοντας τον μέγιστο αριθμό στοιχείων που μπορεί να έχει ένα τέτοιο σύνολο, δίνει την δυνατότητα φραγής του μέγιστου κόστους μιας αναδρομικής κλήσης της `Select`, κάτι που θα βοηθήσει στην απόδειξη της γραμμικότητάς της. Το παρακάτω Λήμμα προκύπτει εύκολα.

Λήμμα 8.1. *Το ελάχιστο μέγεθος του M_1 ή του M_2 είναι $\frac{3n}{10} - 6$*

Απόδειξη. Από την παρατήρηση του Σχήματος 8.5, προκύπτει ότι καθένα από τα ορθογώνια περιέχει τουλάχιστον $3\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$ στοιχεία. Αυτό προκύπτει αφού δεν μετράμε την τελευταία πεντάδα που μπορεί να μην είναι



Σχήμα 8.5: Διαίρεση σε πεντάδες. Τα στοιχεία του M_1 είναι μικρότερα του $\overline{m}$ και τα στοιχεία M_2 μεγαλύτερα ή ίσα του $\overline{m}$

γεμάτη καθώς και την πεντάδα που περιέχει το $\overline{m}$. ∎

Από το Λήμμα 8.1 προκύπτει ότι το μέγιστο υποπρόβλημα αν λάβουμε σαν στοιχείο διαχωρισμού το $\overline{m}$, περιέχει $\frac{7n}{10} + 6$ στοιχεία. Έστω $T(n)$, ο μέγιστος χρόνος εκτέλεσης του αλγορίθμου για κάθε σύνολο M που περιέχει n στοιχεία και κάθε i.

Λήμμα 8.2. *Η παρακάτω αναδρομική σχέση περιγράφει την πολυπλοκότητα του αλγορίθμου* `Select`.

$$T(n) \leq \begin{cases} \Theta(1) & \text{, για } n \leq 80 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + g(n) & \text{, για } n > 80 \end{cases}$$

Απόδειξη. Ο αλγόριθμος θα εκτελεστεί μια φορά για τον μέσο των μέσων (γραμμή 8), δηλαδή για $\lceil n/5 \rceil$ στοιχεία, οπότε προκύπτει ο όρος $(\lceil n/5 \rceil)$ και μια φορά αναδρομικά, είτε για το M_1 (γραμμή 12) είτε για το M_2 (γραμμή 14), όπου ο χρόνος που ξοδεύεται είναι το πολύ $(7n/10 + 6)$, λόγω του Λήμματος 8.1. Επίσης, θα πρέπει $7n/10+6 < n \Rightarrow n > 20$. Ο όρος $g(n) = O(n)$, αφού οι υπόλοιπες γραμμές του προγράμματος απαιτούν γραμμικό πλήθος βημάτων. Έστω λοιπόν σταθερά b, έτσι ώστε $g(n) \leq bn$, για κάθε n μεγαλύτερο από μία σταθερά. ∎

Θεώρημα 8.1. *Ο αλγόριθμος* `Select` *τρέχει σε χρόνο γραμμικό ως προς το πλήθος της εισόδου*

Απόδειξη. Θα αποδείξουμε με επαγωγή ότι $(n) \leq cn$, όπου c μία σταθερά εξαρτώμενη από τη σταθερά στον όρο $O(n)$ της αναδρομικής σχέσης.

Για n $\leq$ 80: Προφανές.

Για n $>$ 80: Υποθέτουμε ότι ισχύει $(\ell) \leq c\ell$, $\ell < n$. Θα δείξουμε ότι ισχύει και για $\ell = n$. Έχουμε

$$\begin{aligned} T(n) &\leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + bn \\ &\leq c\lceil n/5 \rceil + \frac{7cn}{10} + 6c + bn \\ &\leq \frac{cn}{5} + c + \frac{7cn}{10} + 6c + bn \\ &= \frac{9cn}{10} + 7c + bn \leq cn \end{aligned} \tag{8.1}$$

όπου η 8.1 ισχύει για

$$\frac{bn}{n + 7 + \frac{9n}{10}} \leq c$$

Αυτή η ανισότητα ισχύει για κάθε $n > 80$ αν καθορίζουμε επακριβώς το b.

Συνολικά έχουμε

$$T(n) \leq \begin{cases} \Theta(1), & n \leq 80 \\ cn & n > 80 \end{cases}$$

∎

8.5 Ταξινόμηση με Συγχώνευση

Με τον όρο συγχώνευση (merging) εννοούμε την πράξη της δημιουργίας ενός νέου ταξινομημένου πίνακα που περιέχει όλα τα στοιχεία δύο (ή και περισσοτέρων) πινάκων, που είναι ήδη ταξινομημένοι. Η διαδικασία `Merge` που ακολουθεί υλοποιεί τη συγχώνευση δύο ταξινομημένων πινάκων `A` και `B` (με n και m ακεραίους αντίστοιχα) στον πίνακα `C`. Θεωρείται ότι οι δύο πίνακες έχουν και μία επιπλέον θέση που χρησιμεύει για την αποθήκευση ενός φρουρού (sentinel).

```
     procedure merge;
1.   i <-- 1; j <-- 1; A[n+1] <-- maxint; B[m+1] <-- maxint;
2.   for k <-- 1 to n+m do
3.      if A[i]<B[j] then
4.         C[k] <-- A[i]; i <-- i+1
5.      else
6.         C[k] <-- B[j]; j <-- j+1
```

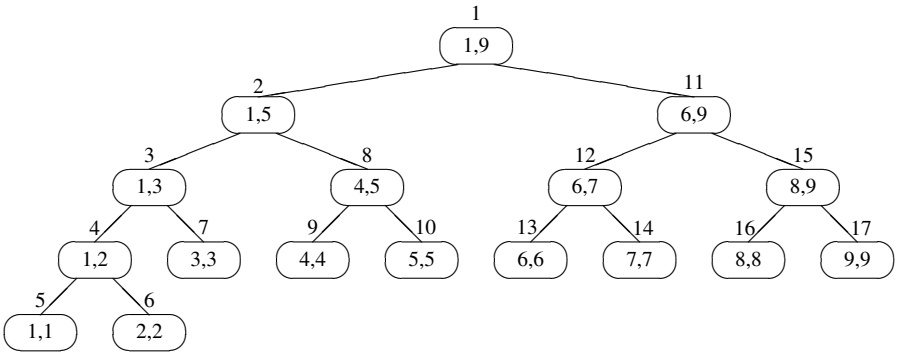
Στην ουσία η συγχώνευση γίνεται με τη βοήθεια δύο δεικτών `i` και `j` που σαρώνουν τους δύο πίνακες και διαδοχικά συγκρίνουν στοιχεία από τους πίνακες `A` και `B`. Έτσι, κάθε φορά το μικρότερο στοιχείο μεταφέρεται στον πίνακα εξόδου `C`. Ο αναγνώστης μπορεί πολύ εύκολα να καταλάβει τη λειτουργία της διαδικασίας `Merge` και δεν χρειάζονται περισσότερες επεξηγήσεις. Από το βρόχο της εντολής 2, είναι προφανές ότι η πολυπλοκότητα της συγχώνευσης είναι $\Theta(n+m)$, θεωρώντας φραγμένο από επάνω το κόστος της ερώτησης της εντολής 3-6.

Στη φιλοσοφία της διαδικασίας αυτής βασίζεται μία σημαντικότατη μέθοδος ταξινόμησης, η ονομαζόμενη ταξινόμηση με ευθεία συγχώνευση (straight merge sort). Κατά καιρούς η μέθοδος αυτή έχει υλοποιηθεί με πλήθος τρόπων, όπως επαναληπτικά ή αναδρομικά, για λίστες ή πίνακες κοκ. Στη συνέχεια, ακολουθεί μία εκδοχή της για πίνακες, η οποία αποτελείται από δύο τμήματα: την αναδρομική `merge_sort` και τη `merge`. Στις εντολές 16-18 της `merge_sort` βρίσκεται το μεσαίο στοιχείο του πίνακα και εκτελούνται 2 κλήσεις στους δύο υποπίνακες που προκύπτουν. Από εδώ προκύπτει ότι η μέθοδος ανήκει στην οικογένεια των αλγορίθμων *Διαίρει και Βασίλευε*. Η διαδικασία `merge` που καλείται όταν προκύψουν στοιχειώδεις υπο-...-υπο-πίνακες αναλαμβάνει τις συγκρίσεις των στοιχείων και τα τακτοποιεί με τη βοήθεια ενός βοηθητικού πίνακα `B`.

```
     procedure merge(left,middle,right);
1.   first <-- left; second <-- middle+1; temp <-- left;
2.   while (first<=middle) and (second<=right) do
3.      if A[first]<=A[second] then
4.         B[temp] <-- A[first]; first <-- first+1;
5.      else
6.         B[temp] <-- A[second]; second <-- second+1;
7.      temp <-- temp+1
8.   if first<=middle then
```

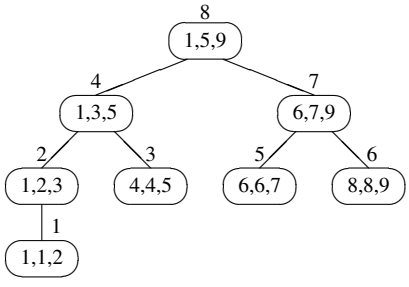
```
9.       for k <-- first to middle do
10.         B[temp] <-- A[k]; temp <-- temp+1
11.   else
12.      for k <-- second to right do
13.         B[temp] <-- A[k]; temp <-- temp+1
14.   for k <-- left to right do A[k] <-- B[k]

      procedure merge_sort(left,right);
15.   if left<right then
16.      middle <-- (left+right) div 2;
17.      merge_sort(left,middle);
18.      merge_sort(middle+1,right);
19.      merge(left,middle,right)
```



Σχήμα 8.6: Σειρά κλήσεων της merge_sort.

Αν εφαρμόσουμε τις ανωτέρω διαδικασίες σε πίνακα με τα 9 γνωστά κλειδιά, τότε η mergesort θα κληθεί 17 φορές με τη σειρά που απεικονίζεται επάνω



Σχήμα 8.7: Σειρά κλήσεων της merge.

από κάθε κόμβο του επομένου σχήματος, ενώ μέσα σε κάθε κόμβο φαίνονται τα ορίσματα της αντίστοιχης κλήσης. Στο Σχήμα 8.7 φαίνονται 8 κλήσεις της merge, με την αντίστοιχη τάξη εκτός του κόμβου και τα ορίσματα εντός του κόμβου. Και στις δύο περιπτώσεις η διάσχιση του δένδρου ακολουθεί τη λογική της αναζήτησης με προτεραιότητα βάθους (dfs).

Πρόταση.
Η πολυπλοκότητα της `merge_sort` είναι $\Theta(n \log n)$ στη χειρότερη περίπτωση.

Απόδειξη
Παρατηρώντας τη διαδικασία `merge_sort` εξάγουμε την επόμενη αναδρομική εξίσωση:

$$T(n) \;=\; T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

με αρχική συνθήκη $T(1) = 1$, ενώ οι τρεις όροι του δεξιού σκέλους αντιστοιχούν στις εντολές 17-19. Συγκεκριμένα, το κόστος σε συγκρίσεις της εντολής 19 προκύπτει από όσα αναφέρθηκαν προηγουμένως για τη διαδικασία συγχώνευσης 2 πινάκων. Για την ευκολία της ανάλυσης υποθέτουμε ότι $n = 2^k$, οπότε η ανωτέρω αναδρομική εξίσωση είναι πλέον εύκολη υπόθεση καθώς ισχύει διαδοχικά:

$$\begin{aligned} T(n) &= 2T(n/2) + n \;=\; 2(2T(n/4) + n/2) + n \;=\; 4T(n/4) + 2n \;=\; \ldots \\ &= 8T(n/8) + 3n \;=\; 2^k T(n/2^k) + kn \;=\; n + n\log n \;=\; \Theta(n \log n) \end{aligned}$$

□

Η μέθοδος ταξινόμησης με ευθεία συγχώνευση αξίζει την προσοχή μας γιατί είναι πολύ ευσταθής (stable) μέθοδος, δηλαδή ανεξάρτητα από τη φύση των δεδομένων θα εκτελέσει τον ίδιο αριθμό συγκρίσεων στη μέση και τη χειρότερη περίπτωση. Το χαρακτηριστικό αυτό έρχεται σε αντίθεση με το μειονέκτημα της γρήγορης ταξινόμησης, που διακρίνεται από πολυπλοκότητα $O(n^2)$ για τη χειρότερη περίπτωση. Ωστόσο, το συμπέρασμα αυτό προκύπτει γιατί στην ανάλυση λάβαμε υπ'όψιν μας μόνο το πλήθος των εκτελούμενων συγκρίσεων. Σε μία πραγματική υλοποίηση, η χρήση του βοηθητικού πίνακα `B` και οι εκτελούμενες καταχωρήσεις είναι, επίσης, πράξεις μη αμελητέου χρονικού κόστους, οι οποίες επιβαρύνουν τη συνολική επίδοση. Τελικώς, αν και η ταξινόμηση με συγχώνευση δεν είναι ίσως η καλύτερη μέθοδος ταξινόμησης στην κύρια μνήμη, εν τούτοις αποτελεί τη βάση για μεθόδους εξωτερικής ταξινόμησης.

8.6 Ταξινόμηση του Shell

Η μέθοδος που προτάθηκε από τον Shell έχει βασικό χαρακτηριστικό ότι χρησιμοποιεί μία ακολουθία ακεραίων $h_1, h_2, \ldots, h_k$, όπου ισχύει: $h_1 > h_2 > \ldots > h_{k-1} > h_k = 1$ και γι'αυτό φέρει και την ονομασία ταξινόμηση με **μειούμενες αυξήσεις** (diminishing increment). Στην πράξη η μέθοδος λειτουργεί για οποιεσδήποτε τιμές της ακολουθίας $h_1, h_2, \ldots, h_{k-1}$, αλλά θα πρέπει να ισχύει $h_k = 1$.

Θα μπορούσε να υποστηριχθεί ότι η ταξινόμηση του Shell είναι μία παραλλαγή της ταξινόμησης με εισαγωγή. Ουσιαστικά, η μέθοδος αποτελείται από k φάσεις, όπου στην i-οστή φάση (για $1 \leq i \leq k$) θεωρείται το βήμα h_i, οπότε ακολουθώντας τη λογική της ταξινόμησης με εισαγωγή τίθενται σε σωστή διάταξη μεταξύ τους τα στοιχεία του πίνακα που απέχουν h_i θέσεις. Προφανώς η k-οστή και τελευταία φάση είναι ένα κλασικό πέρασμα της ταξινόμησης με εισαγωγή που ολοκληρώνει τη διαδικασία. Πρέπει να τονισθεί ότι αποδεδειγμένα, αν ο πίνακας είναι ταξινομημένος με βάση κάποιο βήμα, τότε παραμένει ταξινομημένος με βάση το ίδιο βήμα ακόμη και αν εφαρμοσθούν ένα η περισσότερα επόμενα βήματα στη συνέχεια. Επίσης, καθώς η μέθοδος αυτή στηρίζεται στην ταξινόμηση με εισαγωγή, είναι και αυτή ευσταθής, δηλαδή δεν ανταλλάσσει αμοιβαία στοιχεία με ίσα κλειδιά.

Το παράδειγμα του Σχήματος 8.8 δείχνει το περιεχόμενο του πίνακα με τα γνωστά 9 κλειδιά και τον τρόπο ανταλλαγής τους, καθώς ο πίνακας ταξινομείται σταδιακά με 3 βήματα μεγέθους 4, 2 και 1. Μεταξύ των γραμμών που δείχνουν το πέρας κάθε φάσης, παρουσιάζονται οι εκτελούμενες ανταλλαγές στοιχείων του πίνακα. Στο παράδειγμα αυτό, επίσης, παρατηρούμε χαρακτηριστικά πως ο πίνακας παραμένει ταξινομημένος με βήμα 4 ακόμη και μετά το πέρας της φάσης με βήμα 2.

Ο ψευδοκώδικας που ακολουθεί παρουσιάζει αλγοριθμικά το προηγούμενο σκεπτικό. Ο ψευδοκώδικας δεν είναι αρκετά γενικός, καθώς θεωρεί μία απλή ακολουθία με 3 βήματα μεγέθους 4, 2 και 1, ώστε να συμφωνεί με το παράδειγμα. Η δεύτερη παρατήρηση είναι ότι για επιπλέον λόγους ευκολίας θεωρεί ότι στα αριστερά του πίνακα υπάρχουν τρεις επιπλέον θέσεις, όπου τοποθετούνται κόμβοι φρουροί με τιμές ίσες προς τα βήματα, αντίστοιχα.

```
     procedure shellsort;
1.   h[1] <-- 4; h[2] <-- 2; h[3] <-- 1;
2.   for m <--  1 to 3 do
3.      k <-- h[m]; s <-- -k
4.      for i <-- k+1 to n do
```

Αρχικά Κλειδιά	52	12	71	56	5	10	19	90	45
	5				52				
					45				52
		10				12			
			19				71		
Βήμα 4	5	10	19	56	45	12	71	90	52
							52		71
				12		56			
Βήμα 2	5	10	19	12	45	56	52	90	71
			12	19					
						52	56		
								71	90
Βήμα 1	5	10	12	19	45	52	56	71	90

Σχήμα 8.8: Ταξινόμηση με μειούμενες αυξήσεις.

```
5.          x <-- A[i]; j <-- i-k;
6.          if s=0 then s <-- -k;
7.          s <-- s+1; A[s] <-- x;
8.          while x<A[j] do
9.             A[j+k] <-- A[j]; j <-- j-k
10.         A[j+k] <-- x
```

Η απλή αυτή λογική της μεθόδου του Shell πάσχει κατά το ότι η ακολουθία των ακεραίων παραμένει μία παράμετρος προς βελτιστοποίηση. Δηλαδή, έχουν προταθεί και έχουν αναλυθεί με περισσότερη ή λιγότερη δυσκολία αρκετές ακολουθίες, αλλά θα μπορούσε να προταθεί μία νέα ακολουθία και να αποδειχθεί ότι αυτή είναι καλύτερη από κάθε προηγούμενη και πιθανώς η βέλτιστη. Στη συνέχεια, θα παρουσιάσουμε την ανάλυση της χειρότερης περίπτωσης για την ακολουθία που προτάθηκε από τον ίδιο τον Shell και δίνεται από τις σχέσεις: $h_1 = \lfloor n/2 \rfloor$ και $h_{i+1} = \lfloor h_i/2 \rfloor$.

Πρόταση.
Η πολυπλοκότητα της `shellsort` είναι $\Theta(n^2)$ στη χειρότερη περίπτωση για την ακολουθία του Shell, όπου $h_1 = \lfloor n/2 \rfloor$ και $h_{i+1} = \lfloor h_i/2 \rfloor$.

Απόδειξη
Για να αποδείξουμε ότι η πολυπλοκότητα της χειρότερης περίπτωσης για την ακολουθία του Shell είναι $\Theta(n^2)$, πρώτα θα αποδείξουμε ότι ισχύει το $\Omega(n^2)$ και μετά το $O(n^2)$. Έστω ότι $n = 2^k$, οπότε κάθε βήμα είναι άρτιος αριθμός εκτός

από το τελευταίο βήμα που ισούται με τη μονάδα. Έστω, επίσης, ότι αρχικά τα $n/2$ μεγαλύτερα στοιχεία τοποθετούνται στις άρτιες θέσεις, ενώ τα $n/2$ μικρότερα στοιχεία τοποθετούνται στις περιττές θέσεις. Μέχρι να φθάσουμε στο τελευταίο βήμα, όλα τα $n/2$ μεγαλύτερα στοιχεία είναι ακόμη στις άρτιες θέσεις και όλα τα $n/2$ στοιχεία είναι στις μικρότερες θέσεις. Επομένως, το i-οστό μικρότερο στοιχείο (για $i \leq n/2$) βρίσκεται στην $(2i-1)$-οστή θέση πριν αρχίσει το τελικό πέρασμα και πρέπει να εκτελεσθούν $i-1$ ανταλλαγές μέχρι να φθάσει στην τελική του θέση. Άρα, για όλα τα $n/2$ μικρότερα στοιχεία θα απαιτηθούν $\sum_{i=1}^{n/2}(i-1) = \frac{n^2-n}{8} = \Omega(n^2)$ ανταλλαγές. Το Σχήμα 8.9 απεικονίζει το παράδειγμα που περιγράφηκε πριν.

Αρχικά	1	5	2	6	3	7	4	8
Βήμα 4	1	5	2	6	3	7	4	8
Βήμα 2	1	5	2	6	3	7	4	8
Βήμα 1	1	2	3	4	5	6	7	8

Σχήμα 8.9: Ταξινόμηση με μειούμενες αυξήσεις.

Για να αποδείξουμε το δεύτερο σκέλος, αρκεί να θεωρήσουμε ότι κατά το πέρασμα με βήμα h_k, ουσιαστικά εκτελείται μία ταξινόμηση με εισαγωγή (τετραγωνικής πολυπλοκότητας) για ένα σύνολο n/h_k στοιχείων. Επομένως, για το σύνολο των περασμάτων με αυτό το βήμα η πολυπλοκότητα είναι $O(h_k(n/h_k)^2)$ = $O(n^2/h_k)$. Επομένως για το σύνολο των βημάτων προκύπτει η πολυπλοκότητα $O(\sum_{i+1}^{t} n^2/h_t) = O(n^2 \sum_{i+1}^{t} 1/h_t)$. Καθώς η ακολουθία των βημάτων είναι γεωμετρική, έπεται ότι $\sum_{i+1}^{t} 1/h_t < 2$. Έτσι προκύπτει το ζητούμενο, δηλαδή ότι η πολυπλοκότητα της ταξινόμησης του Shell για τη δεδομένη ακολουθία είναι $O(n^2)$. □

Στη βιβλιογραφία έχουν παρουσιασθεί πολλές εναλλακτικές προτάσεις για την ακολουθία των βημάτων και έχουν αναλυθεί. Για παράδειγμα, για την ακολουθία $1, 3, 7, \ldots, 2^k-1$ του Hibbard, αν και εικάζεται από πειραματικά αποτελέσματα ότι η πολυπλοκότητα είναι $O(n^{5/4})$, το όριο που έχει αποδειχθεί είναι $\Theta(n^{3/2})$.

8.7 Όρια Αλγορίθμων Ταξινόμησης

Μέχρι στιγμής εξετάσθηκαν μέθοδοι ταξινόμησης με πολυπλοκότητα της τάξης $O(n^2)$ ή $O(n \log n)$. Τι εκφράζει η κάθε μία από αυτές τις τάξεις πολυπλοκότητας; Ο μέθοδος της ταξινόμησης με εισαγωγή, της ταξινόμησης με

επιλογή και της ταξινόμησης με ανταλλαγή (bubblesort) είναι οι λεγόμενες **ευθείες** (straight) μέθοδοι που είναι απλές στην κατανόηση, την κωδικοποίηση και την ανάλυση, αλλά αποτελούν μία οικογένεια αργών αλγορίθμων ταξινόμησης. Για τις μεθόδους αυτές ισχύει η επόμενη θεώρηση.

Δοθέντος ενός πίνακα Α με n διακριτά στοιχεία λέγεται ότι υπάρχει μία **αντιστροφή** (inversion), αν ισχύει $A[i] > A[j]$ για κάποια $i < j$. Με απλά λόγια, στην ακολουθία $S = 3, 14, 1, 5, 9$ οι αντιστροφές είναι (3,1), (14,1), (14,5) και(14,9).

Πρόταση.
Κάθε αλγόριθμος που σε κάθε βήμα απομακρύνει το πολύ μία αντιστροφή απαιτεί $n(n-1)/2$ βήματα στη χειρότερη περίπτωση και $n(n-1)/4$ βήματα στη μέση περίπτωση.

Απόδειξη.
Αν υποθέσουμε ότι ο πίνακας Α περιέχει μία τυχαία διάταξη των τιμών $1, 2, \ldots, n$ από το σύνολο των $n!$ διατάξεων. Ορίζουμε ως **ανεστραμμένο** (reverse) του πίνακα Α, τον πίνακα $\mathrm{A}^R = (\mathrm{A}[n], \mathrm{A}[n-1], \ldots, \mathrm{A}[1])$. Σημειώνεται ότι $(\mathrm{A}^R)^R = \mathrm{A}$, ενώ για $n > 1$ δεν υπάρχει περίπτωση κάποιος πίνακας να αποτελεί αντίστροφο του εαυτού του. Έτσι οι $n!$ διατάξεις μπορούν να χωρισθούν σε $n!/2$ ζεύγη, όπου κάθε ζεύγος αποτελείται από μία διάταξη και την αντεστραμμένη της. Υπάρχουν $n(n-1)/2$ ζεύγη τιμών i, j, όπου $1 \leq j < i \leq n$. Για κάθε τέτοιο ζεύγος τιμών (i, j) και για κάθε ζεύγος διάταξης και της αντεστραμμένης της $(\mathrm{A}, \mathrm{A}^R)$, θα υπάρχει μόνο μία αντιστροφή στη μία διάταξη από τις δύο του κάθε ζεύγους. Επομένως ο συνολικός αριθμός αντιστροφών σε όλες τις διατάξεις είναι $n(n-1)/2$, ενώ η μέση τιμή των αντιστροφών είναι $n(n-1)/4$. □

Πρόταση.
Δεδομένου ενός πίνακα που περιέχει μία τυχαία διάταξη με n στοιχεία, το άθροισμα των αποστάσεων που διανύονται από τα στοιχεία κατά την ταξινόμησή τους είναι $(n^2-1)/3$.

Απόδειξη.
Ας υποθέσουμε ότι ο πίνακας Α περιέχει μία τυχαία διάταξη των τιμών $1, 2, \ldots, n$. Δοθέντος ενός τυχαίου στοιχείου $\mathrm{A}[j]$, η απόσταση που θα διανύσει είναι $\mathrm{A}[j] - j$. Επομένως θεωρώντας το σύνολο των στοιχείων ισχύει ότι η μέση τιμή της

απόστασης ισούται με:

$$\begin{aligned} \mathrm{E}[\mathrm{A}[j]-j] &= \frac{|1-j|+|2-j|+\ldots+|j-j|+\ldots+|n-j|}{n} \\ &= \frac{1}{n}\left(\sum_{i=1}^{j-1} i+\sum_{i=1}^{n-j} i\right) = \frac{1}{n}\left(\frac{j(j-1)}{2}+\frac{(n-j)(n-j+1)}{2}\right) \end{aligned}$$

Συνεπώς, η μέση διανυόμενη απόσταση είναι:

$$\sum_{j=1}^{n}(\mathrm{E}[\mathrm{A}[j]-j]) = \frac{1}{n}\sum_{j=1}^{n}\left(\frac{j(j-1)}{2}+\frac{(n-j)(n-j+1)}{2}\right)$$

Οι όροι εντός του αθροίσματος είναι ίδιοι για συμπληρωματικές τιμές του j και επομένως ισχύει:

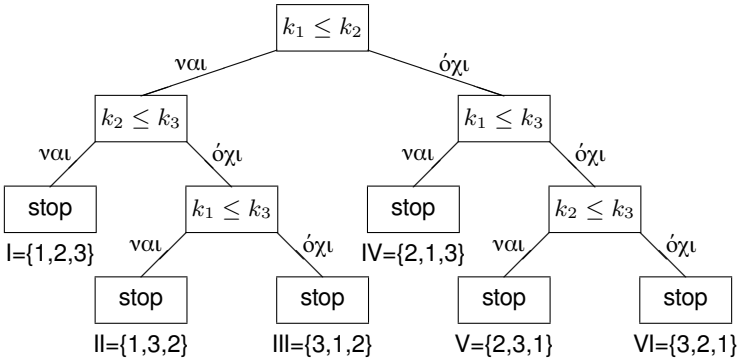
$$\sum_{j=1}^{n}(\mathrm{E}[\mathrm{A}[j]-j]) = \frac{1}{n}\sum_{j=1}^{n} j(j-1) = \ldots = \frac{n^2-1}{3}$$

Συνεπώς, ένας αλγόριθμος ταξινόμησης που μετακινεί τα στοιχεία κατά ένα σταθερό αριθμό θέσεων σε κάθε βήμα απαιτεί $\mathrm{O}(n^2)$ βήματα. □

Τελικά τίθεται το ερώτημα «Πόσο γρήγορα μπορεί μία μέθοδος να ταξινομήσει έναν πίνακα;». Θα αποδειχθεί ότι, όταν μία μέθοδος ταξινόμησης στηρίζεται μόνο σε συγκρίσεις και ανταλλαγές κλειδιών, τότε στη χειρότερη περίπτωση δεν μπορεί να ταξινομήσει σε χρόνο κατώτερο του $\mathrm{O}(n \log n)$.

Πριν την απόδειξη της σχετική πρότασης είναι αναγκαίο να περιγραφεί η διαδικασία ταξινόμησης με ένα **δένδρο αποφάσεων** (decision tree). Ένα μονοπάτι του δένδρου δείχνει μία πιθανή διαδοχή από υπολογισμούς που ο αλγόριθμος πιθανώς να ακολουθήσει. Για παράδειγμα, ας υποτεθεί ότι πρέπει να ταξινομηθούν τρία κλειδιά: τα k_1, k_2 και k_3 αντίστοιχα. Η σειρά εισόδου των κλειδιών είναι k_1, k_2 και k_3 που φαίνεται στο δένδρο αποφάσεων ως (1,2,3). Αρχικά, συγκρίνονται τα κλειδιά k_1 και k_2. Αν $k_1 < k_2$, τότε η σειρά (1,2,3) δεν αλλάζει, αλλιώς γίνεται (2,1,3).

Στο επόμενο σχήμα φαίνονται όλα τα πιθανά ενδεχόμενα που μπορούν να συμβούν σε έναν αλγόριθμο ταξινόμησης ανάλογα με τη σειρά σύγκρισης των κλειδιών στο δένδρο. Από το σχήμα αυτό φαίνεται ότι το δένδρο αποφάσεων είναι ένα δυαδικό δένδρο. Τα φύλλα του δένδρου είναι αριθμημένα από I ως VI και αποτελούν τα μοναδικά σημεία τερματισμού κάθε αλγορίθμου ταξινόμησης. Το δένδρο αυτό έχει 3!=6 φύλλα που εξασφαλίζει ότι ο αλγόριθμος βρίσκει



Σχήμα 8.10: Δένδρο αποφάσεων.

πάντοτε τη διάταξη εκείνη που αντιστοιχεί στην ταξινομημένη σειρά.

Πρόταση.
Κάθε δένδρο αποφάσεων που ταξινομεί n διακεκριμένα κλειδιά έχει ύψος τουλάχιστον $\log n!$.

Απόδειξη.
Ένα δένδρο αποφάσεων έχει $n!$ φύλλα που αντιστοιχούν σε κάθε μία από τις $n!$ διατάξεις των n κλειδιών. Ας υποτεθεί ότι το δένδρο είναι πλήρες και ότι το ύψος του είναι h. Τότε ο αριθμός των φύλλων είναι 2^{h-1}, οπότε συνεπάγεται ότι $n! = 2^{h-1}$. Λογαριθμώντας και τα δύο μέλη της σχέσης προκύπτει:

$$\log n! \;=\; \log 2^{h-1} \;=\; h-1$$

Το δένδρο δεν είναι πλήρες και, άρα, η πρόταση ισχύει. □

Πρόταση.
Κάθε αλγόριθμος που ταξινομεί κάνοντας μόνο συγκρίσεις και ανταλλαγές κλειδιών έχει στη χειρότερη περίπτωση χρόνο τάξης $\mathrm{O}(n \log n)$.

Απόδειξη.
Πρέπει να αποδειχθεί ότι σε κάθε δένδρο αποφάσεων με $n!$ φύλλα υπάρχει ένα μονοπάτι μήκους $c \cdot n \log n$, όπου c είναι μία σταθερά. Από την προηγούμενη πρόταση προκύπτει ότι κάθε μονοπάτι είναι μήκους $\log n!$. Αντικαθιστώντας με βάση τον τύπο του Stirling προκύπτει:

$$h \;=\; \log n! \;=\; \log \sqrt{2\pi n} + n \log n - n \log e \;=\; \mathrm{O}(n \log n)$$

□

8.8 Ταξινόμηση με Μέτρημα

Ας υποθέσουμε ότι δίνεται ένας πίνακας A με n στοιχεία, που λαμβάνουν τιμές από 1 μέχρι k, όπου $k < n$. Θα εκτελέσουμε μία ταξινόμηση με τη βοήθεια ενός βοηθητικού πίνακα C μεγέθους k, ενώ το αποτέλεσμα θα αποθηκευθεί στον πίνακα B. Με απλά λόγια, η ταξινόμηση αυτή δεν είναι επιτόπια. Δίνεται ο επόμενος ψευδοκώδικας και θα σχολιασθεί στη συνέχεια με τη βοήθεια ενός παραδείγματος.

```
     procedure countsort
1.   for i <-- 1 to k do C[i] <-- 0;
2.   for i <-- 1 to n do C[A[i]] <-- C[A[i]]+1;
3.   for i <-- 2 to k do C[i] <-- C[i]+C[i-1]
4.   for i <-- n downto 1 do
5.      B[C[A[i]]] <-- A[i]; C[A[i]] <-- C[A[i]]-1
```

5	7	3	5	2	1	4	3

Πίνακας 8.1: Ταξινόμηση με μέτρημα (αρχικός πίνακας).

Έστω $n = 8, k = 7$ και ο πίνακας A με το περιεχόμενο του Πίνακα 8.1. Με την εντολή 2 καταμετρώνται οι εμφανίσεις της κάθε τιμής (από τις k) και τοποθετούνται σε αντίστοιχες θέσεις του πίνακα C. Με την εντολή 3 σαρώνεται ο πίνακας C και προκύπτει η νέα μορφή του C, όπου κάθε θέση προκύπτει ως το άθροισμα των τιμών των δύο προηγούμενων θέσεων. Το περιεχόμενο του πίνακα C μετά το πέρας των εντολών 2-3 δίνονται στον Πίνακα 8.2. Με το βρόχο της εντολής 4, στον πίνακα B τοποθετείται η ζητούμενη ταξινομημένη ακολουθία και τερματίζει ο αλγόριθμος.

1	1	2	1	2	0	1

1	2	4	5	7	7	8

Πίνακας 8.2: Ταξινόμηση με μέτρημα (βοηθητικός πίνακας).

Παρατηρούμε ότι ο ψευδοκώδικας αποτελείται από τέσσερις διαδοχικούς βρόχους. Εξ αυτών οι δύο εκτελούν n επαναλήψεις, ενώ άλλοι δύο εκτελούν k επαναλήψεις. Ευνόητο είναι ότι η πολυπλοκότητα του αλγορίθμου είναι $\Theta(n+k)$. Με την παραδοχή ότι $k < n$, έπεται ότι η πολυπλοκότητα είναι $\Theta(n)$. Επιτύχαμε δηλαδή ένα γραμμικό αλγόριθμο ταξινόμησης σε αντίθεση με τα προηγούμενα; Η ερώτηση είναι παγίδα γιατί στην παρούσα περίπτωση (α) ο αλγόριθμος δεν στηρίζεται στις συγκρίσεις και (β) αν $k >> n$, τότε προφανώς δεν ισχύει το τελικό συμπέρασμα.

Τώρα ας προσέξουμε τον επόμενο κώδικα. Οι πρώτες δύο εντολές είναι πανομοιότυπες με τις πρώτες δύο εντολές της διαδικασίας `countsort`, αλλά στη συνέχεια υπάρχει διαφορά. Πιο συγκεκριμένα, δεν υπάρχει ο βοηθητικός πίνακας B και, επίσης παρατηρούμε ένα διπλό βρόχο στις εντολές 4-6. Αν εκτελέσουμε τον αλγόριθμο, έστω και με χαρτί και μολύβι, θα διαπιστώσουμε ότι ουσιαστικά εκτελεί την ίδια λειτουργία με την προηγούμενη διαδικασία.

```
     procedure countsort2
1.   for i <-- 1 to k do C[i] <-- 0;
2.   for i <-- 1 to n do C[A[i]] <-- C[A[i]]+1;
3.   i <-- 0;
4.   for j <-- 1 to k do
5.      for m <-- C[j] down to 1 do
6.         i <-- i+1; A[i] <-- j
```

Πρόταση.
Η πολυπλοκότητα της `countsort2` είναι $\Theta(n+k)$ στη χειρότερη περίπτωση.

Απόδειξη.
Η περίπτωση θυμίζει το παράδειγμα που είχαμε επιλύσει στο Κεφάλαιο 1.5 (όπου το άνω όριο του βρόχου δεν ήταν ένας κλασικός δείκτης αλλά το περιεχόμενο μίας θέσης του πίνακα). Σύμφωνα, λοιπόν, με μία πρώτη προσέγγιση, με βάση τα όρια των βρόχων `for` μπορούμε να φθάσουμε στο συμπέρασμα ότι η πολυπλοκότητα είναι $O(kn)$, επειδή n είναι η τιμή του `C[i]` στην εντολή 5 στη χειρότερη περίπτωση. Ωστόσο, η ανάλυση αυτή δεν είναι σφικτή.

Ο εξωτερικός βρόχος `for` της εντολής 4 είναι συγκεκριμένος, δηλαδή έχουμε k επαναλήψεις. Η εντολή 6 θα εκτελεσθεί ακριβώς n φορές και όχι kn. Αυτό προκύπτει με βάση την εξής παρατήρηση: Ας θεωρήσουμε την j-οστή επανάληψη του εξωτερικού βρόχου. Ο έλεγχος του εσωτερικού βρόχου `for` θα εκτελεσθεί `C[j]+1` φορές. Επομένως, ο συνολικός αριθμός ελέγχων στο σημείο

αυτό είναι:

$$\sum_{i=1}^{k} (C[i]+1) \;=\; \sum_{i=1}^{k} C[i] \;+\; \sum_{i=1}^{k} 1 \;=\; n+k$$

Συνεπώς και πάλι καταλήγουμε στο ίδιο συμπέρασμα ως προς την πολυπλοκότητα της διαδικασίας. □

8.9 Ταξινόμηση με Βάση τη Ρίζα

Η **ταξινόμηση με βάση τη ρίζα** (radix sort) είναι μία γενίκευση της ταξινόμησης με μέτρημα. Με απλά λόγια, έστω ότι δίνεται προς ταξινόμηση ένας πίνακας με n ακεραίους μέγεθους d ψηφίων. Εκτελούμε μία ταξινόμηση με μέτρημα ως προς το τελευταίο, δηλαδή το λιγότερο σημαντικό ψηφίο (least significant digit). Έτσι, προκύπτει μία νέα μορφή του πίνακα, που και πάλι ταξινομούμε με μέτρημα αλλά αυτή τη φορά ως προς το δεύτερο ψηφίο από το τέλος. Η διαδικασία αυτή συνεχίζεται και τελειώνει, όταν ο προκύπτων πίνακας ταξινομείται ως προς το πρώτο, δηλαδή το περισσότερο σημαντικό ψηφίο (most significant digit).

Σημειώνεται ότι στην ταξινόμηση με βάση τη ρίζα δεν υπάρχει περιορισμός ως προς το εύρος τιμών των κλειδιών, όπως υπάρχει στην ταξινόμηση με μέτρημα (η γνωστή παράμετρος k). Όμως, όταν λαμβάνεται σε κάθε κλήση το αντίστοιχο ψηφίο, τότε εκεί προφανώς τα ψηφία λαμβάνουν τιμές εντός του διαστήματος 1-9, και έτσι συνιστάται η χρήση της ταξινόμησης με μέτρημα. Η μέθοδος θα μπορούσε να εφαρμοσθεί και σε συμβολοσειρές, όπου κάθε χαρακτήρας θα λάμβανε τιμές στο διάστημα a-z ή α-ω (δηλαδή, 26 ή 24 διαφορετικές τιμές αντίστοιχα). Από το χρησιμοποιούμενο, λοιπόν, αριθμητικό σύστημα (δηλαδή το δεκαδικό, δυαδικό κλπ. σε περίπτωση αριθμών ή το διάστημα 1-24/26 σε περίπτωση χαρακτήρων) προκύπτει η ονομασία της μεθόδου (radix). Η επόμενη συνοπτικότατη διαδικασία καταγράφει αυτήν την περιγραφή.

```
   procedure radixsort
1. for i <-- 1 to d do
2.    countsort(d)
```

Έστω ότι δίνονται προς ταξινόμηση ο πίνακας `A` με τα γνωστά 9 κλειδιά 52, 12, 71, 56, 5, 10, 19, 90 και 45, όπως φαίνεται στον Πίνακα 8.3. Αν καλέσουμε την `countsort` της εντολής 2 με όρισμα `d=1` (θεωρούμε ότι η αρίθμηση

των ψηφίων αρχίζει από το λιγότερο σημαντικό), τότε θα προκύπτει η δεύτερη μορφή του πίνακα, ενώ με τη δεύτερη κλήση της `countsort` και όρισμα `d=2` θα προκύψει η τελική μορφή του πίνακα.

52	12	71	56	5	10	19	90	45

10	90	71	52	12	5	45	56	19

5	10	12	19	45	52	56	71	90

Πίνακας 8.3: Ταξινόμηση με βάση τη ρίζα.

Η ανάλυση της πολυπλοκότητας της ταξινόμησης με βάση της ρίζα είναι εύκολη υπόθεση. Εφόσον η ταξινόμηση με μέτρημα έχει γραμμική πολυπλοκότητα $\Theta(n+k)$, έπεται ότι η πολυπλοκότητα της ταξινόμησης με βάση τη ρίζα είναι $\Theta(dn+dk)$. Θεωρώντας το d σταθερά, προκύπτει και πάλι γραμμική πολυπλοκότητα.

8.10 Ταξινόμηση με Κάδους

Η **ταξινόμηση με κάδους** (bucket sort) στηρίζεται στην υπόθεση ότι δίνονται προς ταξινόμηση n αριθμοί που ανήκουν στο διάστημα [0,1). Το διάστημα αυτό υποδιαιρείται σε n ίσα υποδιαστήματα (τους λεγόμενους κάδους), όπου κατανέμονται τα ισάριθμα στοιχεία προς ταξινόμηση. Επειδή, θεωρούμε ότι τα στοιχεία υπακούουν σε μία ομοιόμορφη κατανομή, δεν αναμένουμε να ανήκουν πάρα πολλά στοιχεία σε ένα συγκεκριμένο υποδιάστημα. Η τελική ταξινομημένη ακολουθία προκύπτει ταξινομώντας τα στοιχεία του κάθε κάδου και κατόπιν λαμβάνοντας τους κάδους στη σειρά. Η επόμενη διαδικασία περιγράφει όσα αναφέραμε. Εκτός του πίνακα A με τα n στοιχεία, χρησιμοποιείται και ένας βοηθητικός πίνακας B με επίσης n θέσεις για τη δημιουργία συνδεδεμένων λιστών.

```
     procedure bucketsort
1.   for i <-- 1 to n do insert(A[i],B[floor(n*A[i])])
2.   for i <-- 1 to n do insertsort(B[i])
3.   return (B[1], B[2], ..., B[n])
```

Για παράδειγμα, έστω ότι δίνεται ο πίνακας A με τα γνωστά $n = 9$ στοιχεία από τα προηγούμενα παραδείγματα, διαιρεμένα δια 100. Θεωρούμε έναν

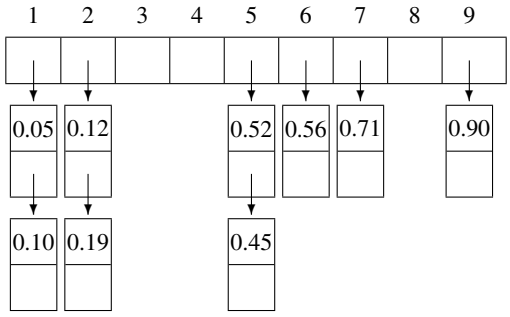
πίνακα B με 9 θέσεις, όποτε τα υποδιαστήματα που αντιστοιχούν στις θέσεις του πίνακα B είναι [0-0,11), [0,11-0,22), [0,22-0,33), ..., [0,88-1). Το Σχήμα 8.11 απεικονίζει την κατάσταση μετά το πέρας της εντολής 1, οπότε έχουν εισαχθεί όλα τα στοιχεία στις αντίστοιχες συνδεδεμένες λίστες. Εννοείται ότι στη συνέχεια τα στοιχεία κάθε λίστας πρέπει να ταξινομηθούν, ώστε να δοθούν στην έξοδο.

Πρόταση.
Η πολυπλοκότητα της `bucketsort` είναι γραμμική στη μέση περίπτωση.

Απόδειξη.
Προκειμένου να βρούμε την πολυπλοκότητα της μεθόδου πρέπει να εστιάσουμε στην εντολή 2, γιατί προφανώς η εντολή 1 έχει γραμμικό κόστος. Στην εντολή 2 ουσιαστικά εκτελούμε n ανεξάρτητες ταξινομήσεις με εισαγωγή με n_i στοιχεία η κάθε μία, όπου το n_i είναι μία τυχαία μεταβλητή. Επομένως, το συνολικό κόστος των εντολών 1-2 είναι:

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=1}^{n} \mathrm{O}\left(n_i^2\right)\right] \\ &= \Theta(n) + \sum_{i=1}^{n} \mathrm{E}\left[\mathrm{O}(n_i^2)\right] = \Theta(n) + \sum_{i=1}^{n} \mathrm{O}\left(\mathrm{E}[n_i^2]\right) \end{aligned}$$

Ορίζουμε μία τυχαία μεταβλητή X_{ij} που ισούται με 1, αν το στοιχείο $A[j]$ κατανέμεται στον i-οστό κάδο, ενώ αλλιώς ισούται με 0 (όπου $1 \leq i, j \leq n$). Άρα



Σχήμα 8.11: Ταξινόμηση με κάδους.

ισχύει $n_i = \sum_{j=1}^{n} X_{ij}$ και συνεπώς:

$$\begin{aligned} \mathrm{E}[n_i^2] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] = \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\ &= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n}\sum_{\substack{k=1 \\ k\neq j}}^{n} X_{ij}X_{ik}\right] \end{aligned}$$

Θα επιλύσουμε τα δύο αθροίσματα ξεχωριστά. Η τυχαία μεταβλητή X_{ij} ισούται με 1 με πιθανότητα $1/n$. Επομένως:

$$\mathrm{E}[X_{ij}^2] = 1 \times \frac{1}{n} + 0 \times (1 - \frac{1}{n}) = \frac{1}{n}$$

Σε σχέση με το δεύτερο άθροισμα ισχύει ότι $k \neq j$ και επομένως οι δύο τυχαίες μεταβλητές είναι ανεξάρτητες. Από το Κεφάλαιο 2.1 γνωρίζουμε ότι: $\mathrm{E}[XY] = \mathrm{E}[X] \times \mathrm{E}[Y]$. Συνεπώς:

$$\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\,\mathrm{E}[X_{ik}] = \frac{1}{n}\frac{1}{n} = \frac{1}{n^2}$$

Άρα, αντικαθιστώντας προκύπτει:

$$\begin{aligned} \mathrm{E}[n_i^2] &= \sum_{j=1}^{n} \mathrm{E}[X_{ij}^2] + \sum_{j=1}^{n}\sum_{\substack{k=1 \\ k\neq j}}^{n} \mathrm{E}[X_{ij}\,X_{ik}] \\ &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n}\sum_{\substack{k=1 \\ k\neq j}}^{n} \frac{1}{n^2} \\ &= n\frac{1}{n} + n(n-1)\frac{1}{n^2} = 2 - \frac{1}{n} \end{aligned}$$

Επιστρέφοντας στην αρχική σχέση έχουμε:

$$\mathrm{E}[T(n)] = \Theta(n) + \sum_{i=1}^{n}(2 - \frac{1}{n}) = \Theta(n) + \Theta(n) = \Theta(n)$$

□

Στη συνέχεια, ακολουθεί μία κομψότερη απόδειξη του προηγούμενου αποτελέσματος.

Πρόταση.
Η πολυπλοκότητα της `bucketsort` είναι γραμμική στη μέση περίπτωση.

Απόδειξη.
Η πιθανότητα ένα συγκεκριμένο στοιχείο να κατανεμηθεί στον i-οστό κάδο είναι $p = 1/n$. Η πιθανότητα υπακούει μία δυωνυμική κατανομή με προσδοκητή τιμή $\mathrm{E}[n_i] = np = 1$ και απόκλιση $\mathrm{Var}[\mathrm{n_i}] = np(1-p) = 1 - 1/n$. Από το Κεφάλαιο 2.1 γνωρίζουμε ότι για οποιαδήποτε τυχαία μεταβλητή ισχύει:

$$\mathrm{E}[n_i^2] = \mathrm{Var}[n_i] + \mathrm{E}^2[n_i]$$

Συνεπώς, ισχύει:

$$\mathrm{E}[n_i^2] \;=\; 1 - \frac{1}{n} + 1^2 \;=\; 2 - \frac{1}{n} \;=\; \Theta(1)$$

Άρα, αντικαθιστώντας την τελευταία έκφραση στον αρχικό τύπο του $\mathrm{E}[T(n)]$, τελικώς προκύπτει ότι η ταξινόμηση με κάδους έχει γραμμική πολυπλοκότητα $\Theta(n)$ στη μέση περίπτωση. Στη χειρότερη περίπτωση η πολυπλοκότητα είναι τετραγωνική $\Theta(n^2)$. □

8.11 Βιβλιογραφική Συζήτηση

Η ταξινόμηση είναι ένα εξαιρετικά τιμημένο αντικείμενο της Πληροφορικής, καθώς έχει απασχολήσει στο παρελθόν πλήθος ερευνητών λόγω της θεωρητικής και πρακτικής σπουδαιότητάς του. Οι αλγόριθμοι ταξινόμησης που παρουσιάσθηκαν προηγουμένως μπορούν να βρεθούν σε οποιοδήποτε διδακτικό εγχειρίδιο για δομές δεδομένων και αλγορίθμους. Ο 3ος τόμος του βιβλίου του Knuth είναι ίσως το πληρέστερο και βαθύτερο κείμενο [26]. Σύμφωνα με τον Knuth, κάθε αλγόριθμος ταξινόμησης κατατάσσεται σε μία από πέντε κατηγορίες αναλόγως με τη βασική λειτουργία του. Οι κατηγορίες είναι: με εισαγωγή, με ανταλλαγή, με επιλογή, με συγχώνευση και με κατανομή. Πλούσιο σε υλικό είναι επίσης και το βιβλίο των Gonnet-BaezaYates [17].

Καλές επισκοπήσεις μπορούν να βρεθούν στα βιβλία και άρθρα [12, 28, 30, 31, 32, 33]. Ιδιαιτέρως, στο άρθρο [33] οι αλγόριθμοι ταξινόμησης κατατάσσονται σε δύο κατηγορίες: αυτούς που έχουν εύκολο διαχωρισμό και δύσκολη

σύνδεση (όπως ταξινόμηση με εισαγωγή, ταξινόμηση με συγχώνευση, ταξινόμηση με μειούμενες αυξήσεις) και σε αυτούς που έχουν δύσκολο διαχωρισμό και εύκολη σύνδεση (όπως ταξινόμηση φυσαλίδας, ταξινόμηση επιλογής, γρήγορη ταξινόμηση).

Η λεγόμενη ταξινόμηση φυσαλίδας ή ταξινόμηση με αντικατάσταση έχει υπερ-παρουσιασθεί σε διδακτικά εγχειρίδια, αν και είναι η χειρότερη τετραγωνική μέθοδος. Παρότι δεν εξετάσθηκε στα πλαίσια του παρόντος βιβλίου παρά μόνο στις Ασκήσεις 8-9, σχετικό υλικό μπορεί να βρεθεί στα άρθρα [8, 9, 53]. Από τις τετραγωνικές μεθόδους ξεχωρίζει η μέθοδος της ταξινόμησης με εισαγωγή, η οποία έχει μελετηθεί περισσότερο από τις άλλες αντίστοιχες μεθόδους [29, 47], θεωρείται μάλιστα ότι είναι η προσφορότερη όταν ο πίνακας είναι περίπου ταξινομημένος [6]. Η ανάλυση για την πολυπλοκότητα των μετακινήσεων στην ταξινόμηση με επιλογή αναφέρεται στο άρθρο [14].

Η γρήγορη ταξινόμηση προτάθηκε από τον Hoare [23] και υπήρξε πολύ δημοφιλής, καθώς αποτέλεσε αφορμή για πλήθος παραλλαγών και αναλύσεων [5, 25, 27, 29, 34, 35, 36, 39, 41, 42, 43, 44, 49, 50, 51]. Βασική αναφορά για τη μέθοδο ταξινόμησης με συγχώνευση αποτελεί το άρθρο [4], ενώ παραλλαγές της μεθόδου προτείνονται στα άρθρα [37, 48]. Η ταξινόμηση shellsort προτάθηκε από τον Shell [46], ενώ ο Hibbard βελτίωσε σημαντικά την αρχική μέθοδο στο άρθρο [21]. Απόδειξη για την πολυπλοκότητα της μέθοδου χρησιμοποιώντας την ακολουθία Hibbard αναφέρεται στο βιβλίο [56]. Περαιτέρω, η μέθοδος μελετήθηκε εκτενώς, όπως στα άρθρα [2, 3, 10, 11, 16, 24, 38, 45, 52, 54, 55, 57].

Η χρήση του δέντρου απόφασης για το κάτω όριο ταξινόμησης βάσει συγκρίσεων οφείλεται στους Ford-Johnson [15]. Η παρουσίαση των ταξινομήσεων που δεν βασίζονται στις συγκρίσεις ακολουθεί την προσέγγιση του βιβλίου των Cormen-Leiserson-Rivest-Stein [7].

Η Άσκηση 8 βασίζεται στο άρθρο [13], η Άσκηση 14 στο βιβλίο [19], η Άσκηση 15 στα άρθρα [34, 35, 36], η Άσκηση 22 στο άρθρο [6], η Άσκηση 23 στο άρθρο [20], ενώ η Άσκηση 25 στο άρθρο [18].

Στα πλαίσια του βιβλίου αυτού δεν εξετάζονται αλγόριθμοι εξωτερικής ταξινόμησης (δηλαδή σε δευτερεύουσα μνήμη), καθώς περισσότερο ταιριάζουν στο αντικείμενο των Βάσεων Δεδομένων. Ωστόσο, ο ενδιαφερόμενος αναγνώστης παραπέμπεται στο βιβλίο της Salzberg [40]. Επίσης, ο αναγνώστης παραπέμπεται στο άρθρο [12], όπου εξετάζονται οι λεγόμενοι προσαρμοζόμενοι (adaptive) αλγόριθμοι ταξινόμησης, δηλαδή αλγόριθμοι των οποίων η πολυπλοκότητα εξαρτάται από την ποσότητα της αταξίας στον υπό ταξινόμηση πίνακα.

8.12 Ασκήσεις

1. Ποιά από τις επόμενες πράξεις μπορούν να υλοποιηθούν καλύτερα, αν προηγουμένως ο αντίστοιχος πίνακας έχει ταξινομηθεί;

   - εύρεση της ελάχιστης τιμής,
   - εύρεση της μέγιστης τιμής,
   - εύρεση της μέσης τιμής,
   - εύρεση της μεσαίας τιμής, και
   - εύρεση της συχνότερης τιμής (mode).

2. Ποιός είναι ο ελάχιστος αριθμός συγκρίσεων για την ταξινόμηση 5 διακριτών ακεραίων; Πόσες συγκρίσεις και μετακινήσεις απαιτούνται στην καλύτερη, μέση και χειρότερη περίπτωση. Να επιλυθεί η άσκηση θεωρώντας 8 διακριτούς αριθμούς.

3. Επί ευθείας γραμμής τοποθετούνται $2n$ δίσκοι, όπου εναλλάσσονται ένας μαύρος και ένας λευκός. Ας υποθέσουμε ότι οι δίσκοι περιττής (άρτιας) τάξης είναι μαύροι (λευκοί). Πρέπει να συγκεντρώσουμε όλους τους μαύρους στα αριστερά και όλους τους λευκούς στα δεξιά της ευθείας γραμμής. Η μοναδική επιτρεπόμενη κίνηση είναι να ανταλλάξουμε μεταξύ τους δύο γειτονικούς δίσκους. Ποιός είναι ο ελάχιστος αριθμός μετακινήσεων;

4. Επί ευθείας γραμμής τοποθετούνται $2n$ δίσκοι, όπου οι πρώτοι n είναι μαύροι και επόμενοι n είναι λευκοί. Ποιός είναι ο ελάχιστος αριθμός μετακινήσεων, ώστε στη γραμμή να εναλλάσσονται μαύροι και λευκοί (ο πρώτος δίσκος να είναι μαύρος);

5. Να σχεδιασθεί και να αναλυθεί η μέθοδος **εισαγωγής με δυαδική εισαγωγή** (binary insertion sort), όπου χρησιμοποιείται η δυαδική αναζήτηση για την εύρεση της σωστής θέσης όπου θα γίνει η κάθε φορά εισαγωγή του επόμενου στοιχείου μέσα στην ακολουθία προορισμού. Η ανάλυση να θεωρήσει το πλήθος των συγκρίσεων και το πλήθος των μετακινήσεων.

6. Να σχεδιασθεί και να αναλυθεί αλγόριθμος που να εντοπίζει τους 2 μεγαλύτερους αριθμούς σε ένα πίνακα με n διαφορετικούς ακεραίους.

7. Δίνεται ένας πίνακας με n κλειδιά και ζητείται να βρεθούν τα μικρότερα k κλειδιά, όπου $1 < k < n/2$, χωρίς όμως να ταξινομηθεί πρώτα ο πίνακας. Να σχεδιασθούν τουλάχιστον δύο τρόποι.

8. Δίνεται αταξινόμητος πίνακας με n στοιχεία και ζητείται το k-οστό μεγαλύτερο. Να σχεδιασθεί λεπτομερώς αλγόριθμος που θα ταξινομεί τα πρώτα k στοιχεία και κατόπιν θα εξετάζει τα επόμενα $n-k$ με τη βοήθεια της ταξινόμησης με εισαγωγή. Να βρεθεί η πολυπλοκότητα του προτεινόμενου αλγορίθμου.

9. Σε ένα πίνακα `A[0..n-1]` τα στοιχεία `A[0], A[2], A[4]` κλπ είναι τα μικρότερα στοιχεία, ενώ τα `A[1], A[3], A[5]` κλπ είναι τα μεγαλύτερα. Να βρεθεί ο συνολικός αριθμός των συγκρίσεων (ως συνάρτηση του n) που θα κάνει η μέθοδος ταξινόμησης με εισαγωγή και η μέθοδος του Shell με δύο μειούμενες αυξήσεις μεγέθους 2 και 1, αντιστοίχως.

10. Ο πίνακας `A[0..n-1]` περιέχει τις n διακριτές τιμές από 0 μέχρι $n-1$. Τι εκτελεί ο επόμενος κώδικας; Τι πολυπλοκότητα έχει; Ο πίνακας `B` ορίζεται ομοίως.

```
for (i=0; i<n; i++)
   B[A[i]]=A[i]
```

11. Δίνεται ο επόμενος κώδικας. Σε τι μοιάζει και σε τι διαφέρει με τον προηγούμενο κώδικα;

```
for (i=0; i<n; i++)
   while (A[i] != i)
      swap(i,A[i])
```

12. Σε έναν πίνακα υπάρχουν διπλά κλειδιά που πρέπει να απομακρυνθούν. Ποιά μέθοδος ταξινόμησης αν τροποποιηθεί εκτελεί αυτήν τη λειτουργία πιο αποτελεσματικά;

13. Έστω ότι πρέπει να ταξινομηθεί ένας πίνακας με n κλειδιά, όπου τα πρώτα n_1 κλειδιά είναι ήδη ταξινομημένα, ενώ τα τελευταία n_2 δεν είναι. (Ισχύει βέβαια $n = n_1 + n_2$.) Ας θεωρηθεί διαδοχικά ότι: $n_1 \gg n_2$ και ότι $n_1 \approx n_2$. Να σχεδιασθούν και να αναλυθούν εναλλακτικοί τρόποι για κάθε περίπτωση.

14. Δίνεται ταξινομημένος πίνακας `A[0..n-1]`. Με βάση τα στοιχεία του πίνακα αυτού υπολογίζονται τα στοιχεία:

$$b_1 = A[0] + A[1] \quad \text{και} \quad b_i = b_{i-1} + A[i] \ \text{ για } \ 1 < i \leq n-1$$

Να σχεδιασθεί γραμμικός αλγόριθμος δημιουργίας ενός πίνακα `B[0..2*n-2]` που να περιέχει τα στοιχεία του πίνακα `A` και τα στοιχεία b_i.

15. Η **ταξινόμηση με το μέσο όρο** (meansort) είναι μία παραλλαγή της γρήγορης ταξινόμησης σύμφωνα με την οποία κατά το πρώτο πέρασμα λαμβάνεται ως άξονας το κλειδί της πρώτης θέσης, όπως δηλαδή και στη γρήγορη ταξινόμηση. Όμως κατά τη διάρκεια του πρώτου περάσματος υπολογίζοντα οι μέσες τιμές των στοιχείων των δύο υποπινάκων και στη συνέχεια χρησιμοποιούνται ως άξονες για τις περαιτέρω υποδιαιρέσεις των υποπινάκων. Να σχεδιασθεί λεπτομερώς ο αλγόριθμος. Να βρεθούν τα πλεονεκτήματα και τα μειονεκτήματα της μεθόδου σε σχέση με τη γρήγορη ταξινόμηση, ως προς το πλήθος των περασμάτων και των ανταλλαγών. Να αποδειχθεί ότι και η παραλλαγή αυτή είναι O(n^2). (Hint: αρκεί να βρεθεί μία κατάλληλη διάταξη θεωρώντας ένα μικρό πίνακα).

16. Να σχεδιασθούν και να αναλυθούν επαναληπτικοί και αναδρομικοί αλγόριθμοι ταξινόμησης με **συγχώνευση 3 δρόμων** (3-way merge). Δηλαδή, ένας πίνακας n θέσεων θα πρέπει να διαιρεθεί σε τρία τμήματα μεγέθους $\lfloor n/3 \rfloor$, $\lfloor (n+1)/3 \rfloor$ και $\lfloor (n+2)/3 \rfloor$, αυτά με τη σειρά τους να ταξινομηθούν και στη συνέχεια να γίνει η συγχώνευση.

17. Δεδομένου πίνακα μεγέθους n, γενικεύοντας την προηγούμενη άσκηση να σχεδιασθούν και να αναλυθούν επαναληπτικοί και αναδρομικοί αλγόριθμοι ταξινόμησης με συγχώνευση $\sqrt{n}$ δρόμων.

18. Ως γνωστό η απλή συγχώνευση δύο ταξινομημένων πινάκων `A[0..n-1]` και `B[0..m-1]` έχει πολυπλοκότητα O($n+m$). Η πολυπλοκότητα αυτή μπορεί να βελτιωθεί αν $n \gg m$. Για παράδειγμα, αν m=1, τότε το στοιχείο αυτό μπορεί να καταλάβει τη σωστή θέση μεταξύ των n στοιχείων με δυαδική αναζήτηση. Για το λόγο αυτό, η επόμενη τεχνική ονομάζεται **δυαδική συγχώνευση** (binary merging).

    Σύμφωνα, λοιπόν, με τη μέθοδο αυτή ο πίνακας `A` υποδιαιρείται σε m+1 διαδοχικούς υποπίνακες. Αν το τελευταίο στοιχείο του πίνακα `B` είναι μικρότερο από το τελευταίο στοιχείο του προτελευταίου υποπίνακα του πίνακα `A`, τότε το στοιχείο αυτό μαζί με τα στοιχεία του τελευταίου υποπίνακα αποθηκεύονται στις τελευταίες θέσεις του πίνακα εξόδου. Σε αντίθετη περίπτωση βρίσκεται η θέση του m-οστού στοιχείου μεταξύ των στοιχείων του τελευταίου υποπίνακα, οπότε και πάλι το αποτέλεσμα αποθηκεύεται στον πίνακα εξόδου. Έτσι, η μέθοδος συνεχίζει αναδρομικά μέχρι την εξάντληση και των δύο πινάκων `A` και `B`. Να σχεδιασθεί και να αναλυθεί η μέθοδος.

19. Να σχεδιασθεί και να αναλυθεί η μέθοδος της **ευθείας εισαγωγής δύο δρόμων** (two way straight insertion), σύμφωνα με την οποία ένας πίνακας με n κλειδιά ταξινομείται τοποθετώντας τα κλειδιά σε έναν άλλον πίνακα μεγέθους $2n$+1 θέσεων. Η μέθοδος τοποθετεί το πρώτο κλειδί ως μεσαίο στοιχείο του πίνακα εξόδου. Τα υπόλοιπα κλειδιά τοποθετούνται αριστερά ή δεξιά του μεσαίου στοιχείου ανάλογα με το μέγεθος του κλειδιού. Κατά τη διάρκεια της τοποθέτησης γίνονται οι απαραίτητες μετακινήσεις. Για παράδειγμα, αν τα κλειδιά είναι στον πίνακα εισόδου με τη σειρά 52, 12, 71, 56, 5, 10, 19, 90 και 45, τότε τοποθετούνται στον πίνακα εξόδου με τη μορφή παρουσιάζεται στον Πίνακα 8.4.

4	5	6	7	8	9	10	11	12	13	14
						52				
					12	52				
					12	52	71			
					12	52	56	71		
				5	12	52	56	71		
			5	10	12	52	56	71		
		5	10	12	19	52	56	71		
		5	10	12	19	52	56	71	90	
	5	10	12	19	45	52	56	71	90	

Πίνακας 8.4: Το πρόβλημα της ευθείας εισαγωγής δύο δρόμων.

20. Η μέθοδος της **δυαδικής εισαγωγής δύο δρόμων** (two way binary insertion) είναι παρόμοια προς την ευθεία εισαγωγή δύο δρόμων της Άσκησης 19, αλλά η παρεμβολή γίνεται με δυαδικό τρόπο. Να σχεδιασθεί και να αναλυθεί η μέθοδος.

21. Σύμφωνα με τη μέθοδο της **ταξινόμησης περιττής-άρτιας μετάθεσης** (odd-even transposition sort) στα περάσματα περιττής (άρτιας) τάξης συγκρίνονται τα στοιχεία `A[i]` και `A[i+1]`, όπου το `i` είναι περιττός (αντίστοιχα, άρτιος) αριθμός. Αν ισχύει `A[i]>A[i+1]`, τότε τα στοιχεία ανταλλάσσονται. Τα περάσματα επαναλαμβάνονται μέχρις ότου δεν συμβούν άλλες ανταλλαγές. Να σχεδιασθεί και να αναλυθεί αντίστοιχος αλγόριθμος.

22. Η ιδανικότερη μέθοδος για την ταξινόμηση πινάκων με στοιχεία που είναι περίπου ταξινομημένα είναι η εξής: Αρχικά, ελέγχονται τα στοιχεία του πίνακα εισόδου για τον εντοπισμό ζευγών διαδοχικών κλειδιών που

δεν ακολουθούν τη σχέση διάταξης. Το ζεύγος αυτό διαγράφεται από τον πίνακα εισόδου και αποθηκεύεται σε έναν πίνακα εξόδου. Μετά την αφαίρεση κάποιου ζεύγους, η διαδικασία συνεχίζεται με τη σύγκριση του στοιχείου αμέσως πριν το πρώτο του ζεύγους με το στοιχείο αμέσως μετά το δεύτερο του ζεύγους. Μετά την απομάκρυνση κάποιων ζευγών ο πίνακας εισόδου είναι πλέον ταξινομημένος. Στη συνέχεια, ο πίνακας εξόδου ταξινομείται χρησιμοποιώντας τη γρήγορη ταξινόμηση, οπότε οι δύο ταξινομημένοι πίνακες συγχωνεύονται με μία απλή διαδικασία συγχώνευσης. Να σχεδιασθεί και να αναλυθεί η μέθοδος.

23. Η ακόλουθη διαδικασία της **κυκλικής ταξινόμησης** (cycle sort) είναι μία παραλλαγή που εντάσσεται στις ταξινομήσεις με μέτρημα (δες Κεφάλαιο 8.8). Υποτίθεται ότι ένα ιστόγραμμα (πίνακας `H`) διατηρεί τις συχνότητες εμφάνισης όλων των τιμών των κλειδιών που βρίσκονται στο διάστημα $1..m$. Ο αλγόριθμος να δοκιμασθεί για διαφορετικές τιμές του m και να υπολογισθεί η πολυπλοκότητα χρόνου και χώρου. Πότε η μέθοδος είναι αποτελεσματική;

```
     procedure cycle_sort;
1.   H[1] <-- 0;
2.   for k <-- 1 to m-1 do H[k+1] <-- H[k]+h[k];
3.   for i <-- 1 to n do
4.      k <-- A[i]; j <-- H[k]+1;
5.      if i>=j then
6.         if i<>j then
7.            temp <-- A[i];
8.            repeat
9.               swap(A[j],temp);
10.              H[k] <-- j; k <-- temp; j <-- H[k]+1;
11.           until i=j;
12.           A[i] <-- temp;
13.        H[k] <-- I;
```

24. Η μέθοδος **ταξινόμησης με ανταλλαγή ρίζας** (radix exchange sort) εξετάζει τα ψηφία από τα αριστερά προς τα δεξιά. Η μέθοδος θυμίζει τη γρήγορη ταξινόμηση, επειδή αρχικά τα κλειδιά που έχουν ως πρώτο ψηφίο το Ο τοποθετούνται πριν τα κλειδιά που έχουν ως πρώτο ψηφίο το 1. Κατόπιν, η επεξεργασία συνεχίζεται σε κάθε ένα από τα δύο αυτά υποσύνολα αναδρομικά για τα επόμενα ψηφία κατά τον ίδιο τρόπο.

Η επόμενη διαδικασία `radix_exchange` θεωρεί ότι τα κλειδιά είναι θετικοί ακέραιοι. Ο πίνακας σαρώνεται από αριστερά και από δεξιά με τη βοήθεια δύο δεικτών, `i` και `j`, αναζητώντας κλειδιά που να αρχίζουν από 1 και 0, αντιστοίχως. Όταν τέτοια κλειδιά εντοπισθούν, τότε ανταλλάσσονται αμοιβαία. Η διαδικασία σάρωσης συνεχίζεται μέχρι να συναντηθούν οι δύο δείκτες. Η ταξινόμηση εκτελείται με την κλήση `radix_exchange(1,` Ο τελεστής `shr` εκτελεί διολίσθηση προς δεξιά, ενώ το όρισμα `b` χρησιμεύει για τον έλεγχο του αντίστοιχου ψηφίου ξεκινώντας από την τιμή 15 (για το αριστερότερο ψηφίο) και φθάνοντας μέχρι και την τιμή 0 (για το δεξιότερο ψηφίο).

```
procedure radix_exchange(left,right,b);
if (right>left) and (b>=0) then
   i <-- left; j <-- right;
   repeat
      while (A[i] shr (b-1) mod 2=0) and (i<j) do i <-- i+1;
      while (A[j] shr (b-1) mod 2=1) and (i<j) do j <-- j-1;
      Swap(A[i],A[j]);
   until j <-- i;
   if (A[r] shr (b-1) mod 2=0) then j <-- j+1;
   radix_exchange(left,j-1,b-1); radix_exchange(j,right,b-1)
```

25. Η **ταξινόμηση γραμμικής εξέτασης** (linear probing sort) συνδυάζει στοιχεία της αναζήτησης παρεμβολής και του ταξινομημένου κατακερματισμού με γραμμική εξέταση. Εφαρμόζεται όταν είναι γνωστό το εύρος των τιμών `m<n` των κλειδιών του πίνακα `A[0..n-1]`, ενώ επίσης θα πρέπει να θεωρηθεί ένας συμπληρωματικός πίνακας `n+w` θέσεων, όπου οι w θέσεις χρησιμεύουν ως περιοχή υπερχείλισης. Κάθε στοιχείο του αρχικού πίνακα τοποθετείται σε κενή θέση του βοηθητικού πίνακα εφαρμόζοντας την αναζήτησης παρεμβολής. Αν η θέση του βοηθητικού πίνακα είναι κατειλημμένη από άλλο κλειδί, τότε η θέση αυτή καταλαμβάνεται τελικά από το μικρότερο κλειδί, ενώ το μεγαλύτερο τοποθετείται στην επόμενη θέση του βοηθητικού πίνακα. Μετά την εισαγωγή όλων των στοιχείων στο βοηθητικό πίνακα, με ένα απλό πέρασμα τα στοιχεία τοποθετούνται και πάλι στον αρχικό πίνακα. Να σχεδιασθεί η μέθοδος, να δοκιμασθεί πειραματικά και να αναλυθεί η επίδοσή της.

26. Αν στον αλγόριθμο `Select` το δείγμα αλλάξει από 5 σε 7 ή 3 τότε ο αλγόριθμος θα συνεχίσει να έχει γραμμική πολυπλοκότητα στη χειρότερη περίπτωση;

27. Να δείξετε πως ο αλγόριθμος ταξινόμησης `Quicksort` μπορεί να αλλάξει, ώστε η πολυπλοκότητά του να είναι $(n \log n)$ στην χειρότερη περίπτωση.

28. Έστω ότι $X[1 \ldots n]$ και $Y[1 \ldots n]$ δύο ταξινομημένοι πίνακες. Να δώσετε έναν αλγόριθμο με $O(\log n)$ πολυπλοκότητα για την εύρεση του μεσαίου στοιχείου των $2n$ στοιχείων των πινάκων X και Y.

Βιβλιογραφία

[1] M. Blum, R.W. Floyd, V. Pratt, R.L. Lewis, and R.E. Tarjan. Time bounds for selection. *journal of Computer and System Sciences*, 7:448–461, 1973.

[2] J. Boothroyd. Algorithm 201 - Shellsort. *Communications of the ACM*, 6(8):445, 1963.

[3] B. Brejova. Analyzing variants of Shellsort. *Information Processing Letters*, 79(5):223–227, 2001.

[4] C. Bron. Algorithm 426 - Merge sort algorithm. *Communications of the ACM*, 15(5):357–358, 1972.

[5] R. Chaudhuri and A.C. Dempster. A note on slowing quicksort. *ACM SIGCSE Bulletin*, 25(2):57–58, 1993.

[6] C.R. Cook and D.J. Kim. Best sorting algorithm for nearly sorted lists. *Communications of the ACM*, 23(11):620–624, 1980.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, McGraw-Hill, 2nd edition, 2001.

[8] E.E. Doberkat. Asymptotic estimates for the higher moments of the expected behavior of straight insertion sort. *Information Processing Letters*, 14(4):179–182, 1982.

[9] W. Dobosiewicz. An efficient variation of bubble sort. *Information Processing Letters*, 11(1):5–6, 1980.

[10] H. Erkio. A heuristic approximation of the worst case of Shellsort. *BIT*, 20(2):130–136, 1980.

[11] T.O. Espelid. Analysis of Shellsort algorithm. *BIT*, 13(4):394–400, 1973.

[12] V. Esteville-Castro and D. Woods. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[13] J. Fenwick, C. Norris, and J. Wilkes. Scientific experimentation via the matching game. *ACM SIGCSE Bulletin Inroads*, 34(1):326–330, 2002.

[14] F. Ferri and J. Albert. Average-case analysis in an elementary course on algorithms. *ACM SIGCSE Bulletin Inroads*, 30(1):202–206, 1998.

[15] Jr. Ford, R. Lestor, and S. Johnson. A tournament problem. *The American Mathematical Monthly*, 66:387–389, 1959.

[16] D. Ghoshdastidar and M.K. Roy. A study on the evaluation of Shell's sorting technique. *The Computer Journal*, 18(3):234–235, 1975.

[17] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 2nd edition, 1991.

[18] G.H. Gonnet and J.I. Munro. A linear probing sorting and its analysis. In *Proceedings 13th ACM Symposium on Theory of Computing (STOC)*, pages 90–95, 1981.

[19] D. Gries. *Science of Programming*. Springer-Verlag, 1981.

[20] B.K. Haddon. Cycle sort - a linear sorting method. *The Computer Journal*, 33(4):365–367, 1990.

[21] T.H. Hibbard. An empirical study of minimal storage sorting. *Communications of the ACM*, 6(5):206–213, 1963.

[22] C.A.R. Hoare. Algorithm 63 - Partition and algorithm 65 - Find. *Communications of the ACM*, 4:321–322, 1961.

[23] C.A.R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–15, 1962.

[24] J. Incerpi and R. Sedgewick. Practical variations of Shellsort. *Information Processing Letters*, 26(1):37–43, 1987.

[25] J. Jaja. A perspective on quicksort. *IEEE Computing in Science and Engineering*, 2(1):43–49, 2000.

[26] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1975.

[27] R. Loeser. Some performance tests of quicksort and descendants. *Communications of the ACM*, 17(3):143–152, 1974.

[28] H. Lorin. A guided bibliography to sorting. *IBM Systems Journal*, 10(3):244–254, 1971.

[29] Y. Manolopoulos. Variations of quicksort combined with insertion sort. *Wirtshafts Informatik*, 34(3):327–333, 1992.

[30] R.J. Maresh. Sorting out basic sorting algorithms. *ACM SIGCSE Bulletin*, 17(4):54–59, 1985.

[31] W.A. Martin. Sorting. *ACM Computing Surveys*, 3(4):148–174, 1971.

[32] K. Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer Verlag, 1984.

[33] S.M. Merritt. An inverted taxonomy of sorting algorithms. *Communications of the ACM*, 28(1):96–99, 1985.

[34] D. Motzkin. A stable quicksort. *Software – Practice and Experience*, 11(6):607–611, 1981.

[35] D. Motzkin. Meansort. *Communications of the ACM*, 26(4):250–251, 1983.

[36] D. Motzkin and J. Kapenga. More about meansort. *Communications of the ACM*, 27(7):719–722, 1984.

[37] E. Peltola and H. Erkio. Insertion merge sorting. *Information Processing Letters*, 7(2):92–99, 1978.

[38] C.G. Plaxton and T. Suel. Improved lower bounds for Shellsort. In *Proceedings 33rd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 226–235, 1992.

[39] J. Rohrich. A hybrid of quicksort with O($n \log n$) worst case complexity. *Information Processing Letters*, 14(3):119–123, 1982.

[40] B. Salzberg. *File Structures: an Analytic Approach*. Prentice Hall, 1988.

[41] R.S. Scowen. Algorithm 271 - Quicksort. *Communications of the ACM*, 8(11):669–670, 1965. Also: 9(5):354, 1966.

[42] R. Sedgewick. The analysis of quicksort programs. *Acta Informatica*, 7:327–355, 1977.

[43] R. Sedgewick. Quicksort with equal keys. *SIAM journal on Computing*, 6(2):240–267, 1977.

[44] R. Sedgewick. Implementing quicksort programs. *Communications of the ACM*, 21(10):847–857, 1978.

[45] R. Sedgewick. A new upper bound for Shellsort. *journal of Algorithms*, 7(2):159–173, 1986.

[46] D.L. Shell. A highspeed sorting procedure. *Communications of the ACM*, 2(7):30–32, 1959.

[47] J. Tillison and C.K. Shene. On generating worst-cases for the insertion sort. *ACM SIGCSE Bulletin*, 27(2):57–58, 1995.

[48] M. Van Der Nat. A fast sorting algorithm - a hybrid of distributive and merge sorting. *Information Processing Letters*, 10(3):163–167, 1980.

[49] M.H. Van Emden. Algorithm 402 - Increasing the efficiency of quicksort. *Communications of the ACM*, 13(9):563–566, 1970.

[50] M.H. Van Emden. Algorithm 402 - Qsort. *Communications of the ACM*, 13(11):693–694, 1970.

[51] R.L. Wainwright. A class of sorting algorithms based on quicksort. *Communications of the ACM*, 28(4):396–402, 1985. Also: 29(4):331–335, 1986.

[52] A. Weiss. Empirical results on the running time of Shellsort. *The Computer Journal*, 34(1):88–91, 1991.

[53] A. Weiss and R. Sedgewick. Bad cases for shaker-sort. *Information Processing Letters*, 28(3):13–16, 1988.

[54] A. Weiss and R. Sedgewick. More on Shellsort increment sequences. *Information Processing Letters*, 34(5):267–270, 1990.

[55] A. Weiss and R. Sedgewick. Tight lower bounds for Shellsort. *journal of Algorithms*, 11(2):242–251, 1990.

[56] M.A. Weiss. *Data Structures and Algorithms Analysis*. Benjamin Cummings, 1995.

[57] A.C. Yao. An analysis of (h,k,1)-Shellsort. *journal of Algorithms*, 1(1):14–50, 1980.