Κεφάλαιο 4

Αναδρομή

Η αναδρομή αποτελεί μία από τις βασικότερες έννοιες στην επιστήμη των υπολογιστών. Μπορεί να θεωρηθεί ως το υπολογιστικό ανάλογο της αποδεικτικής διαδικασίας της μαθηματικής επαγωγής. Η αναδρομή είναι κατάλληλη για την επίλυση προβλημάτων τα οποία διέπονται από την αναδρομική ιδιότητα: η επίλυση ενός στιγμιότυπου μπορεί να υπολογιστεί από την λύση ενός ή περισσοτέρων στιγμιότυπων του ιδίου προβλήματος αλλά μικρότερου μεγέθους. Όταν η επίλυση ενός στιγμιότυπου μεγέθους αρκετά μικρού μπορεί να προκύψει εύκολα, τότε η παραπάνω ιδιότητα ευνοεί την ανάπτυξη αλγοριθμικής διαδικασίας για την επίλυση του προβλήματος. Ένας αλγόριθμος τέτοιου είδους ονομάζεται *αναδρομικός*. Στο παρόν κεφάλαιο θα περιγράψουμε διάφορους αναδρομικούς αλγόριθμους καταδεικνύοντας έτσι την πληθώρα των εφαρμογών που μπορούν να επιλυθούν με την τεχνική αυτή.

4.1 Εισαγωγή

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

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

Παρατηρούμε ότι η (4.1) είναι ισοδύναμη με

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

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

[1] Ισοδύναμα, λέμε ότι «η συνάρτηση ορίζεται αναδρομικά».

να μην ορίζεται αναδρομικά και η κλήση της συνάρτησης με τη συγκεκριμένη τιμή ορίσματος να προκύπτει (αναπόφευκτα) από τον αναδρομικό ορισμό. Έτσι διασφαλίζεται ότι το *βάθος* της αναδρομής είναι πεπερασμένο. Στην περίπτωση της (4.2) αυτό συμβαίνει αφού η $(n+1)-$ιοστή κλήση της συνάρτησης «παραγοντικό» έχει όρισμα 0 και η τιμή της συνάρτησης για αυτό το όρισμα είναι 1 (δηλαδή, δεν υπολογίζεται πια αναδρομικά). Η συνθήκη που πρέπει να ικανοποιεί το όρισμα προκειμένου η τιμή της συνάρτησης να μην υπολογίζεται αναδρομικά ονομάζεται *οριακή*. Για παράδειγμα, στην περίπτωση της (4.2), η οριακή συνθήκη είναι $n = 0$. Η (4.2) κωδικοποιείται από τον Αλγόριθμο 23.

Αλγόριθμος 23 Αναδρομικός υπολογισμός του $n!$.

Απαιτείται: Ακέραιος $n \geq 0$.
Επιστρέφεται: n παραγοντικό.

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

Οι περισσότερες γλώσσες προγραμματισμού υψηλού επιπέδου υποστηρίζουν αναδρομική κλήση προγραμμάτων με τη μορφή συναρτήσεων. Αυτό πιστοποιεί τη δύναμη και την ευελιξία την οποία παρέχει η αναδρομή η οποία συνοψίζεται στα λόγια του Niklaus Wirth [19]:

« Η δύναμη της αναδρομής βρίσκεται στη δυνατότητα να οριστούν άπειρα αντικείμενα μέσω μίας πεπερασμένης σειράς εντολών. Ισοδύναμα, μία άπειρη σειρά υπολογιστικών βημάτων μπορεί να περιγραφεί από ένα πεπερασμένο αριθμό εντολών ακόμα και αν η σειρά αυτή δεν περιέχει εντολή επανάληψης. »

Σε πρακτικές εφαρμογές οι οποίες αφορούν υλοποιήσεις αναδρομικών αλγορίθμων και εκτέλεσης τους σε υπολογιστικά περιβάλλοντα θα πρέπει το βάθος της αναδρομής να είναι αποδεδειγμένα όχι μόνο πεπερασμένο αλλά και αρκετά μικρό. Ο λόγος είναι ότι η αναδρομική κλήση απαιτεί χώρο στη μνήμη προκειμένου να αποθηκεύσει τόσο τις τιμές των τοπικών μεταβλητών όσο και την *κατάσταση του υπολογισμού (state of the computation)* η οποία πρέπει να ανακτηθεί όταν τερματιστεί η αναδρομική κλήση.

4.2 Παραδείγματα αναδρομικών αλγορίθμων

Στην ενότητα αυτή θα παραθέσουμε παραδείγματα αναδρομικών αλγορίθμων. Οι αλγόριθμοι αυτοί αποτελούν μέρος μίας ευρύτερης βιβλιοθήκης που χρησιμοποιούν την αναδρομή σαν το βασικό υπολογιστικό εργαλείο.

4.2.1 Η σειρά Fibonacci

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

αυτή αποτελείται από τους αριθμούς

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

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

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

Η (4.3) μπορεί να μετασχηματιστεί άμεσα σε υπολογιστική αναδρομική διαδικασία· ο Αλγόριθμος 24 την περιγράφει.

Αλγόριθμος 24 Υπολογισμός αριθμών Fibonacci.

Απαιτείται: Ακέραιος $n \geq 0$.
Επιστρέφεται: $n-$ιοστός αριθμός Fibonacci.

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

4.2.2 Αναζήτηση

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

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

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

Αλγόριθμος 25 Εύρεση στοιχείου με τη μέθοδο των ομάδων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ώς hi, στοιχείο αναζήτησης a.
Επιστρέφεται: Επιστροφή θέσης στον πίνακα A που βρίσκεται το μικρότερο στοιχείο που είναι μεγαλύτερο-ίσο του a.

```
1:  function Search_in_Group(int a, int A[], int lo, int hi)
2:      if lo > hi then
3:          return lo
4:      else
5:          n ← hi − lo + 1;
6:          q ← ⌊√n⌋;
7:          r ← (hi − lo) mod q;
8:          j ← hi;
9:          q1 ← r;
10:         if A[lo] > a then
11:             return lo;
12:         end if
13:         if A[hi] < a then
14:             return hi + 1;
15:         end if
16:         while j ≥ lo + q1 and A[j] > a do
17:             j ← j − q1;
18:             q1 ← q;
19:         end while
20:         if A[j] = a then
21:             return j;
22:         end if
23:         ghi ← min(hi, j + q1) − 1;
24:         glo ← j + 1;
25:         return Search_in_Group(a, A, glo, ghi);
26:     end if
27: end function
```

Μπορούμε να «μετατρέψουμε» τον Αλγόριθμο 25 σε αλγόριθμο δυαδικής αναζήτησης αν οι Γραμμές 6,7 αλλάξουν σε

$$q \leftarrow \lfloor \frac{hi - lo}{2} \rfloor;$$
$$r \leftarrow (hi - lo) \bmod 2;$$

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

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

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

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

Κάνοντας πράξεις το παραπάνω σύνολο γράφεται ως

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

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

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

ή στην ομάδα

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

ανάλογα με το αν $a > lo + \lfloor \frac{hi-lo}{2} \rfloor$ ή $a < lo + \lfloor \frac{hi-lo}{2} \rfloor$. Οπότε ο αλγόριθμος καλείται αναδρομικά με παραμέτρους που δεικτοδοτούν το πρώτο (μικρότερο) και το τελευταίο

(μεγαλύτερο) στοιχείο της αντίστοιχης ομάδας. Παρατηρούμε ότι αμφότερες οι ομάδες έχουν το ίδιο αριθμό στοιχείων ο οποίος είναι μικρότερος από το μισό της αρχικής, ήτοι

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

4.2.3 Ολλανδική σημαία

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

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

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

4.2.4 Διατεταγμένα στατιστικά

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

Ο Αλγόριθμος 27 επιλύει το παραπάνω πρόβλημα καλώντας τη συνάρτηση Pivot_Partition (Αλγόριθμος 10) η οποία, έχοντας αναδιατάξει τα στοιχεία του πίνακα A, επιστρέφει τη *σωστή θέση, έστω j, (Ιδιότητα 1)* που έχει τοποθετήσει το στοιχείο που βρισκόταν πριν σε μία τυχαία θέση pvt του πίνακα A. Η επιλογή της pvt γίνεται από τη συνάρτηση

Αλγόριθμος 26 Αλγόριθμος για το πρόβλημα της Ολλανδικής Σημαίας

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

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

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

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

Αλγόριθμος 27 $k-$ιοστο μικρότερο στοιχείο

Απαιτείται: πίνακας A με στοιχεία στις θέσεις από lo έως hi, ακέραιος $k \in \{1, \ldots, hi - lo + 1\}$.
Επιστρέφεται: $k-$ιοστό μικρότερο στοιχείο του πίνακα A.

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

4.2.5 Η δομή του σωρού

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

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