Κεφάλαιο 3

Βασικές δομές δεδομένων

Ένας *πίνακας (array)* αποτελεί τη βασική δομή δεδομένων της κύριας (εσωτερικής) μνήμης του μοντέλου ΜΑΠ. Το κεφάλαιο αυτό θα περιγραφούν αλγοριθμικές διαδικασίες που αφορούν λειτουργίες πάνω στη δομή αυτή.

3.1 Πίνακες

Ένας πίνακας (array) γενικεύει την έννοια της μεταβλητής. Αποτελεί συστοιχία (συνεχόμενων) θέσεων στη μνήμη της ΜΑΠ που «φιλοξενούν» τιμές ιδίου τύπου. Ο πίνακας αποτελεί μία στατική δομή - περιέχει ένα συγκεκριμένο αριθμό θέσεων ο οποίος δηλώνεται πριν τη χρήση τους και δεν αλλάζει μέχρι το τέλος του αλγόριθμου. Οι θέσεις (τα στοιχεία) που περιλαμβάνει αριθμούνται με βάση ένα συνεχόμενο διάστημα διακριτών τιμών *(ακτίνα)*. Η αναφορά σε έναν πίνακα γίνεται μέσω ενός μνημονικού ονόματος. Για την αναφορά σε ένα στοιχείο χρησιμοποιείται το όνομα του πίνακα μαζί με μία τιμή από την ακτίνα περιστοιχισμένη από αριστερή και δεξιά αγκύλη. Η τιμή αυτή μπορεί να είναι σταθερά ή τιμή μίας μεταβλητής *(δείκτης)*. Έτσι αλλάζοντας την τιμή του δείκτη αναφερόμαστε σε διαφορετικό στοιχείο του πίνακα. Για παράδειγμα, ο συμβολισμός $A[1..10]$ δηλώνει έναν πίνακα με το μνημονικό όνομα A με δέκα θέσεις στη μνήμη αριθμούμενες από 1 ως 10. Δηλαδή η ακτίνα του πίνακα είναι $1..10$. Η αναφορά στο τέταρτο στοιχείο του πίνακα γίνεται μέσω του συμβολισμού $A[4]$. Ισοδύναμα, η αναφορά μπορεί να γίνει σαν $A[i]$ όπου i είναι μία μεταβλητή (δείκτης) που προηγουμένως έχει πάρει την τιμή τέσσερα.

Όταν ένας πίνακας περνά σαν παράμετρος σε μία συνάρτηση δηλώνεται ο τύπος του (όπως και στην περίπτωση παραμέτρου μεταβλητής) και ένα μνημονικό όνομα ακολουθούμενο από αριστερή και δεξιά αγκύλη. Αν δεν δηλώνεται διαφορετικά, η ακτίνα ενός πίνακα ξεκινά από τη θέση 1, δηλαδή το πρώτο στοιχείο του είναι το $A[1]$. Ο αριθμός των θέσεων της μνήμης ενός πίνακα A συμβολίζεται με - δίνεται από τη συνάρτηση - `SizeOf`(A). Στις περισσότερες αλγοριθμικές διαδικασίες θα χρησιμοποιείται μόνο ένα υποσύνολο των θέσεων αυτών. Για την αποφυγή επανάληψης, οι συμβάσεις αυτές δεν θα δηλώνονται στην προδιαγραφή του αλγόριθμου.

Ένας πίνακας μπορεί να έχει παραπάνω από μία διάσταση. Για παράδειγμα η δήλωση $A[1..10][1..5]$ δηλώνει ένα δισδιάστατο πίνακα με δέκα γραμμές και πέντε στήλες. Στην περίπτωση αυτή η συνάρτηση `nRows`(A) θα επιστρέφει το πλήθος των γραμμών του πίνακα (στο παράδειγμα 10) και η `nCols`(A) θα επιστρέφει το πλήθος των στηλών του πίνακα (στο παράδειγμα 5). Θα πρέπει να παρατηρήσουμε ότι παρόλο που μας επι-

τρέπεται να χρησιμοποιούμε πίνακες δισδιάστατους, τρισδιάστατους, κλπ, στη κύρια μνήμη της ΜΑΠ οι πίνακες είναι μονοδιάστατες οντότητες που συνήθως δεσμεύουν συνεχόμενες θέσεις.

3.2 Λειτουργίες σε πίνακα

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

$$n = hi - lo + 1. \tag{3.1}$$

3.2.1 Ταξινόμηση

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

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

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

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

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

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

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

Η παρατήρηση που αφορά το στοιχείο 5 στο παραπάνω παράδειγμα οδηγεί στην ακόλουθη ιδιότητα.

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

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

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

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

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

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

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

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

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

[1] Η διαδικασία Swap (Αλγόριθμος 2) ανταλλάσσει τις τιμές των μεταβλητών που δέχεται σαν παραμέτρους (Κεφάλαιο 1).

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

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

Πίνακας 3.1: Τοποθέτηση στοιχείου στη σωστή θέση - Παράδειγμα 8

Στη συνέχεια ο δείκτης i αυξάνεται μέχρι να βρει κάποιο στοιχείο με τιμή μεγαλύτερη από το 5. Ο δείκτης σταματάει στη δεύτερη θέση αφού $A[2] = 7 > 5$. Αντίστοιχα, ο δείκτης j μειώνεται μέχρι να υποδείξει κάποιο στοιχείο που είναι μικρότερο από το 5. Ο δείκτης σταματάει στη τρίτη θέση αφού $A[3] = 1 < 5$. Η κατάσταση απεικονίζεται στον Πίνακα 3.2. Στη συνέχεια τα στοιχεία $A[2], A[3]$ ανταλλάσσουν θέσεις -

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

Πίνακας 3.2: Τοποθέτηση στοιχείου στη σωστή θέση - Παράδειγμα 8

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

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

Πίνακας 3.3: Τοποθέτηση στοιχείου στη σωστή θέση - Παράδειγμα 8

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

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

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

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

Η Ιδιότητα 1 είναι βρίσκεται στον πυρήνα των αλγόριθμων ταξινόμησης που βασίζονται σε συγκρίσεις στοιχείων· οι αλγόριθμοι αυτοί είναι διαδικασίες που τοποθετούν

[2] Ο αλγόριθμος αυτός αποτελεί κλασικό παράδειγμα της περίπτωσης όπου ο αριθμός των διαδοχικά φωλιασμένων βρόχων δεν αυξάνει αναλογικά την πολυπλοκότητα ενός αλγόριθμου (Ενότητα 2.5.1)

κάθε στοιχείο στη σωστή του θέση. Θα χρησιμοποιήσουμε την ιδιότητα αυτή στη μέθοδο της *ενθετικής ταξινόμησης* που θα περιγράψουμε στη συνέχεια.

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

Αλγόριθμος 11 Ενθετική Ταξινόμηση

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

```
1: function Insert_Sort(int A[], int lo, int hi)
2:     for i ← lo + 1; i ≤ hi; i++ do
3:         j ← i;
4:         while j ≥ lo + 1 and A[j] < A[j − 1] do
5:             Swap(A[j], A[j − 1]);
6:             j − −;
7:         end while
8:     end for
9: end function
```

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

j	Γραμμή 4	Γραμμή 7
2	$[2, \underline{7}, 5, 7, 1]$	$[2, \underline{7}, 5, 7, 1]$
3	$[2, 7, \underline{5}, 7, 1]$	$[2, \underline{5}, 7, 7, 1]$
4	$[2, 5, 7, \underline{7}, 1]$	$[2, 5, 7, \underline{7}, 1]$
5	$[2, 5, 7, 7, \underline{1}]$	$[\underline{1}, 2, 5, 7, 7]$

Πίνακας 3.4: Παράδειγμα ενθετικής ταξινόμησης

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

Απόδειξη. Ο αλγόριθμος χρησιμοποιεί δύο φωλιασμένους βρόχους. Στον εσωτερικό (Γραμμές 4-7), σε κάθε επανάληψη, εκτελεί σταθερό αριθμό ΣΥΒ, έστω c_1. Ο αριθμός των επαναλήψεων αποτελεί συνάρτηση του δείκτη i - δεν ξεπερνά το $i-(lo+1)+1 = i-lo$. Σε κάθε επανάληψη του εξωτερικού βρόχου (Γραμμές 2-8) εκτελείται σταθερός

αριθμός από ΣΥΒ, έστω c_2 συν τα ΣΥΒ που εκτελούνται από τον εσωτερικό βρόχο. Ο αριθμός των επαναλήψεων του εξωτερικούς βρόχου - μία επανάληψη για κάθε τιμή του δείκτη i -είναι $hi - (lo + 1) + 1$. Επιπλέον εκτελείται σταθερός αριθμός ΣΥΒ, έστω c_3 που αφορά την αρχικοποίηση του δείκτη i και τη σύγκριση τερματισμού. Ο συνολικός αριθμός ΣΥΒ, έστω $T(n)$ ικανοποιεί τη σχέση

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

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

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

□

Σε επόμενο κεφάλαιο θα περιγράψουμε αλγόριθμους ταξινόμησης χαμηλότερης πολυπλοκότητας.

3.2.2 Συγχώνευση

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

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

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

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

Η πολυπλοκότητα του Αλγόριθμου 12 καθορίζεται από τον αριθμό των επαναλήψεων του βρόχου **while** εφόσον σε κάθε επανάληψη εκτελείται σταθερός αριθμός ΣΥΒ

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

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

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

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

(Ενότητα 2.5.1). Ο αριθμός των επαναλήψεων είναι ίσος με την τιμή της μεταβλητής k όταν ολοκληρωθεί ο βρόχος, δηλαδή εκτελούνται n επαναλήψεις. Άρα ο Αλγόριθμος 12 είναι πολυπλοκότητας $\Theta(n)$.

3.2.3 Αναζήτηση

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

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

Ένα παράδειγμα κλήσης της Search για την αναζήτηση του στοιχείου a ανάμεσα σε στοιχεία που καταλαμβάνουν τις n πρώτες θέσεις του πίνακα A απεικονίζεται στον Αλγόριθμο 13. Προφανώς η πολυπλοκότητα του Αλγόριθμου 13 ταυτίζεται με την πο-

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

Απαιτείται: Πίνακας A με στοιχεία στις πρώτες n θέσεις και στοιχείο αναζήτησης a.
Επιστρέφεται: `True` αν υπάρχει το στοιχείο a στον πίνακα A και `False` διαφορετικά.

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

λυπλοκότητα του αλγόριθμου που υλοποιεί τη συνάρτηση Search. Το πλήθος των ακεραίων ανάμεσα στους οποίους αναζητείται ο ακέραιος a αποτελεί την παράμετρο n η οποία αντιπροσωπεύει το μέγεθος του στιγμιότυπου και δίνεται από τη (3.1).

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

Αταξινόμητα δεδομένα

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

Αλγόριθμος 14 Εύρεση στοιχείου σε έναν πίνακα A.

Απαιτείται: Πίνακας ακεραίων A, ακέραιοι lo, hi με $lo \leq hi$, ακέραιος a.
Επιστρέφεται: i αν $A[i] = a$, $hi + 1$ διαφορετικά.

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

Ταξινομημένα δεδομένα

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

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

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

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

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

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

μπορούμε να γράψουμε τη διαφορά $hi - lo$ ως

$$hi - lo = q \cdot d + r \Rightarrow hi - r = lo + q \cdot d \tag{3.2}$$

[3] Επιπλέον θα πρέπει να αλλάξει το δεύτερο κομμάτι της προδιαγραφής του αλγόριθμου σε:

Επιστρέφεται: argmin$\{i : A[i] \geq a, lo \leq i \leq hi\}$ αν $A[hi] \geq a$, $hi + 1$ διαφορετικά.

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

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

Λήμμα 5. Το σύνολο των n διατεταγμένων ακεραίων μπορεί να διαμεριστεί σε ένα σύνολο από r στοιχεία[4], d σύνολα από q στοιχεία το καθένα και ένα σύνολο από ένα στοιχείο.

Απόδειξη. Από την (3.3) έχουμε,

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

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

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

$$A[lo_1] \leq \cdots \leq A[hi_1] \leq A[lo_2] \leq \cdots \leq A[hi_2], \tag{3.4}$$

με $lo_2 = hi_1 + 1$.

Αν το στοιχείο a ανήκει στην ομάδα που δεικτοδοτείται από το σύνολο $\{lo_2, \ldots, hi_2\}$, θα πρέπει

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

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

$$j^\star = \text{argmax}\{A[j] : A[j] \leq a, j \in \{hi, hi - r, hi - r - q, \ldots lo\}\}. \tag{3.5}$$

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

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

[4] Το σύνολο αυτό δεν υφίσταται αν $r = 0$.

Το μόνο που μένει είναι να καθοριστεί η τιμή της παραμέτρου q. Από την (3.5) προκύπτει άμεσα ότι ο αριθμός των επαναλήψεων προκειμένου να καθοριστεί η ομάδα μέσα στην οποία μπορεί να βρίσκεται το a δεν μπορεί να ξεπερνά το $d+2$. Αν θεωρήσουμε ότι σε κάθε επανάληψη εκτελείται σταθερός αριθμός ΣΥΒ, ο συνολικός αριθμός των ΣΥΒ είναι $O(\lfloor \frac{d}{q} \rfloor)$. Επειδή $d = hi - lo \leq n-1 < n$, μπορούμε να θεωρήσουμε ότι για τον καθορισμό της ομάδας εκτελούνται $O(\frac{n}{q})$ ΣΥΒ.
Όταν καθοριστεί η ομάδα, θα αναζητηθεί σειριακά το a ανάμεσα σε q το πολύ στοιχεία. Επομένως θα εκτελεστούν $O(q)$ ΣΥΒ (δες πολυπλοκότητα Αλγόριθμου 14).

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

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

για κάθε $n \geq n_0$.

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

$$\frac{\mathrm{d}f(q)}{\mathrm{d}q} = c_1 \frac{\mathrm{d}\frac{n}{q}}{\mathrm{d}q} + c_2 \frac{\mathrm{d}q}{\mathrm{d}q} = \frac{-nc_1}{q^2} + c_2, \tag{3.7}$$

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

Εξισώνοντας την πρώτη παράγωγο με το μηδέν, έχουμε

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

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

$$q = p \cdot \sqrt{n} \tag{3.9}$$

ελαχιστοποιεί την $f(q)$.
Αντικαθιστώντας την τιμή του q στην (3.6) έχουμε

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

και επομένως

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

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

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

Αλγόριθμος 15 Εύρεση στοιχείου με τη μέθοδο των ομάδων.

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

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

της Γραμμής 13 εκτελείται μία φορά και $r \neq 0$. Στην περίπτωση αυτή η ομάδα στην οποία θα αναζητηθεί το a έχει r στοιχεία - είναι το σύνολο $\{hi - r + 1, \ldots, hi\}$ - και εφόσον $r < q$, θα έχουμε $j + q_1 = j + q > hi$.

Παράδειγμα 9. Θεωρούμε τον ταξινομημένο πίνακα

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

Η εντολή

$$k \leftarrow \text{Search\_in\_Group}(6, A, 1, 8);$$

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

Γραμμή 2: $n = 8 - 1 + 1 = 8$

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

Γραμμή 4: $r = (8 - 1) \bmod 2 = 1$

Γραμμή 5: $j = 8$

Γραμμή 6: $q1 = r = 1$

Γραμμή 8: $A[1] = 1 < a = 6$

Γραμμή 10: $A[8] = 20 > a = 6$

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

Γραμμή 14: $j = 8 - 1 = 7$

Γραμμή 15: $q1 = 2$

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

Γραμμή 14: $j = 7 - 2 = 5$

Γραμμή 15: $q1 = 2$

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

Γραμμή 14: $j = 5 - 2 = 3$

Γραμμή 15: $q1 = 2$

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

Γραμμή 17: $A[3] = 7 \neq a = 6$

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

Γραμμή 21: $glo = 3 + 1 = 4$

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

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

3.2.4 Τεχνικές επίλυσης προβλημάτων

Όπως είδαμε ένας πίνακας είναι ένα σύνολο συνεχόμενων θέσεων στη μνήμη με τη δυνατότητα άμεσης πρόσβασης σε οποιαδήποτε από τις θέσεις αυτές - και στο περιεχόμενο της - με τη χρήση μεταβλητών-δεικτών. Η δυνατότητα αυτή αποτελεί τη βάση αλγοριθμικών τεχνικών χαμηλής πολυπλοκότητας που επιλύουν προβλήματα που αφορούν πίνακες. Τέτοιες τεχνικές θα παρουσιαστούν στη συνέχεια.

Κυλιόμενο παράθυρο

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

Αλγόριθμος 16 Εύρεση μεγαλύτερου αθροίσματος k συνεχόμενων ακεραίων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ώς hi, ακέραιος .
Επιστρέφεται: Μεγαλύτερο άθροισμα k συνεχόμενων ακεραίων του πίνακα

```
1:  function k_seq_sum(int k, int A[], int lo, int hi)
2:      max_sum ← −∞;
3:      for i ← lo; i ≤ hi − k + 1; i + + do
4:          w ← 0;
5:          for j ← 0; j ≤ k − 1; j + + do
6:              w ← w + A[i + j];
7:          end for
8:          if max_sum < w then
9:              max_sum ← w;
10:         end if
11:     end for
12:     return max_sum;
13: end function
```

Η πολυπλοκότητα του Αλγόριθμου 16. είναι ίση με $O(nk)$, γεγονός που προκύπτει από τη χρήση των δυο φωλιασμένων βρόχων. Μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα ακολουθώντας την εξής στρατηγική. Αρχικώς υπολογίζουμε το άθροισμα των πρώτων k στοιχείων, ήτοι

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

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

Αλγόριθμος 17 Εύρεση μεγαλύτερου αθροίσματος k συνεχόμενων ακεραίων.

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ώς hi, ακέραιος .
Επιστρέφεται: Μεγαλύτερο άθροισμα k συνεχόμενων ακεραίων του πίνακα.

```
1:  function k_seq_sum02(int k, int A[], int lo, int hi)
2:      w ← 0;
3:      for i ← lo; i ≤ lo + k − 1; i + + do
4:          w ← w + A[i];
5:      end for
6:      max_sum ← w;
7:      for i ← lo + k; i ≤ hi; i + + do
8:          w ← w + A[i] − A[i − k];
9:          if max_sum < w then
10:             max_sum ← w;
11:         end if
12:     end for
13:     return max_sum;
14: end function
```

Τεχνική των δύο δεικτών

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

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

```
found ← False;
for i ← lo; i ≤ hi − 1; i + + do
  for j ← i + 1; j ≤ hi; j + + do
    if A[i] + A[j] = a then
      found ← True;
    end if
  end for
end for
```

Προφανώς η διαδικασία αυτή είναι πολυπλοκότητας $O(n^2)$. Μπορούμε να επιτύχουμε χαμηλότερη πολυπλοκότητα ταξινομώντας πρώτα τον πίνακα και χρησιμοποιώντας στη συνέχεια την τεχνική των δύο δεικτών.· σαρώνεται ο ταξινομημένος πίνακας με έναν δείκτη i ο οποίος τοποθετείται στο πρώτο στοιχείο και βαίνει αυξανόμενος και ένα δείκτη j ο οποίος τοποθετείται στο τελευταίο στοιχείο και βαίνει μειούμενος. Αν το άθροισμα των δύο στοιχείων που δείχνουν οι δείκτες είναι μικρότερο του a τότε αυξάνεται ο δείκτης i ενώ αν είναι μικρότερο μειώνεται ο j. όταν διασταυρωθούν οι δείκτες τότε η διαδικασία τερματίζει. Ο Αλγόριθμος 18 κωδικοποιεί τη διαδικασία.

Παρατηρούμε ότι η πολυπλοκότητα του Αλγόριθμου 18 είναι γραμμική από τη στιγμή που ο πίνακας A έιναι ταξινομημένος. Η διαδικασία ταξινόμησης - με αλγόριθμους που θα παρουσιαστούν σε επόμενο κεφάλαιο - μπορεί να πραγματοποιηθεί σε

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

Απαιτείται: Πίνακας A με στοιχεία στις θέσεις από lo ώς hi, ακέραιος a.
Επιστρέφεται: `True` αν υπάρχουν i, j τέτοια ώστε $A[i] + A[j] = a$, `False`, διαφορετικά.

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

$O(n \lg n)$ ΣΥΒ και άρα αυτή είναι η πολυπλοκότητα του παραπάνω αλγόριθμου.

3.3 Αφηρημένες δομές δεδομένων

Η αναφορά που έγινε στις δομές δεδομένων στην Ενότητα 1.2 αφορούσε κυρίως συμπαγείς δομές δεδομένων (concrete data types). Οι δομές αυτές χρησιμοποιούνται για την υλοποίηση των αφηρημένων δομών δεδομένων (abstract data types): μία αφηρημένη δομή δεδομένων είναι ένα σύνολο με μία συλλογή λειτουργιών επί των στοιχείων του συνόλου. Στην ενότητα αυτή θα περιγράψουμε τις αφηρημένες δομές της *στοίβας (stack),* της *ουράς (queue)* και της *λίστας (list)* και θα δούμε πως υλοποιούνται οι βασικές λειτουργίες κάθε μίας από αυτές με τη χρήση πινάκων.

3.3.1 Στοίβα

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

Για την υλοποίηση των παραπάνω διαδικασιών, θεωρούμε ότι με τη στοίβα Q σχετίζεται η μεταβλητή $size\_Q$ η οποία περιέχει το πλήθος των στοιχείων της. Η στοίβα Q μπορεί να υλοποιηθεί σαν μοναδιάστατος πίνακας. Στην περίπτωση αυτή, η τιμή της $size\_Q$ διαφέρει από την τιμή της `SizeOf`(Q) αφού η τελευταία επιστρέφει τον

αριθμό των θέσεων μνήμης του πίνακα. Οι λειτουργίες push και pop, στην περίπτωση που η στοίβα υλοποιείται από τον μονοδιάστατο πίνακα Q υλοποιούνται από τους Αλγόριθμους 19 και 20, αντίστοιχα. Η μεταβλητή $size\_Q$ αρχικοποιείται στο μηδέν.

Αλγόριθμος 19 Εισαγωγή στοιχείου στη στοίβα.

Απαιτείται: Πίνακας Q, ακέραιος $size\_Q$, ακέραιος a.
Επιστρέφεται: Ο πίνακας Q ενημερωμένος με τον ακέραιο a.

```
1: function Push(int Q[], int size_Q, int a)
2:     if size_Q = SizeOf(Q) then
3:         print Σφάλμα πλήρους στοίβας;
4:     else
5:         size_Q++;
6:         Q[size_Q] ← a;
7:     end if
8: end function
```

Αλγόριθμος 20 Εξαγωγή στοιχείου από στοίβα.

Απαιτείται: Πίνακας Q, ακέραιος $size\_Q$
Επιστρέφεται: -1 αν η στοίβα δεν περιέχει κανένα στοιχείο. Διαφορετικά, το στοιχείο που μπήκε πιο πρόσφατα στον Q.

```
1: function Pop(int Q[], int size_Q )
2:     if size_Q = 0 then
3:         print Σφάλμα κενής στοίβας;
4:         return −1;
5:     else
6:         a ← Q[size_Q];
7:         size_Q − −;
8:         return a;
9:     end if
10: end function
```

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

3.3.2 Ουρά

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

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