Κεφάλαιο 1

Βασικές έννοιες

1.1 Προβλήματα και Αλγόριθμοι

Ένα *(υπολογιστικό) πρόβλημα* είναι μια γενική ερώτηση η οποία μπορεί να απαντηθεί με την επεξεργασία *παραμέτρων*. Το πρόβλημα ορίζεται από μία γενική περιγραφή των παραμέτρων και μία πρόταση που περιγράφει τις ιδιότητες που πρέπει να πληροί *η απάντηση (λύση)*. Δηλαδή, ο ορισμός του δηλώνει τόσο τους περιορισμούς που πρέπει να πληρούν τα δεδομένα εισόδου αλλά και η απάντηση (αποτέλεσμα).

Σαν παράδειγμα, θεωρούμε το πρόβλημα εύρεσης *ελαχίστου* σε ένα δοθέν σύνολο φυσικών αριθμών. Ισοδύναμα, με την μορφή ερώτησης: ποιο είναι το μικρότερο στοιχείο σε ένα σύνολο φυσικών αριθμών; Τα στοιχεία του συνόλου αποτελούν τις παραμέτρους του προβλήματος ενώ η ιδιότητα που πρέπει να πληροί η απάντηση είναι αυτή του μικρότερου ανάμεσα σε αυτά. Ο ορισμός του προβλήματος υποδηλώνει την εγκυρότητα των παραμέτρων της εισόδου (δηλαδή το σύνολο να περιέχει φυσικούς αριθμούς) όσο και του αποτελέσματος (δηλαδή το αποτέλεσμα να είναι ο μικρότερος από αυτούς). Μία εναλλακτική διατύπωση θα μπορούσε να είναι: δεδομένου ενός συνόλου ακεραίων $A = \{a_1, a_2, \ldots, a_n\}, A \subseteq \mathbb{N}$, να βρεθεί το στοιχείο $a' \in A$, τέτοιο ώστε $a' \leq a''$, για κάθε $a'' \in A$. Βλέπουμε δηλαδή, ότι για την περιγραφή ενός προβλήματος αρκεί μια αναφορά στις παραμέτρους με αφηρημένο τρόπο. Όταν οι παράμετροι του προβλήματος εξειδικεύονται σε συγκεκριμένες, *έγκυρες* τιμές (δηλαδή σε τιμές που ικανοποιούν τον ορισμό του προβλήματος) έχουμε ένα *στιγμιότυπο* του προβλήματος. Για παράδειγμα, στο παραπάνω πρόβλημα ένα στιγμιότυπο αποτελεί η εξειδίκευση $A = \{4, 2, 3, 7, 8\}$, ενώ το σύνολο $A = \{\sqrt{2}, 1, 3\}$ όχι. Κάθε στιγμιότυπο έχει - αντιστοιχεί σε - καμία, μία ή πολλές απαντήσεις (λύσεις) που πληρούν τις ιδιότητες που ορίζονται από το πρόβλημα. Στο παραπάνω έγκυρο στιγμιότυπο του προβλήματος ελαχίστου υπάρχει μία λύση: η τιμή 2.

Οι διαδικασίες που χρησιμοποιούνται για την επίλυση υπολογιστικών προβλημάτων ονομάζονται *αλγόριθμοι*.

Ορισμός 1. Ένας αλγόριθμος είναι μία ακολουθία βημάτων, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, με στόχο την επίλυση ενός υπολογιστικού προβλήματος.

Όπως στην περιγραφή ενός προβλήματος καθορίζονται οι ιδιότητες των παραμέτρων και του αποτελέσματος έτσι και στην περιγραφή ενός αλγορίθμου πρέπει να αναφέρεται τι αναμένουμε ο αλγόριθμος να επιτύχει όταν τροφοδοτηθεί με είσοδο που

πληροί συγκεκριμένες υποθέσεις. Αυτή είναι η *προδιαγραφή* του αλγόριθμου η οποία αποτελείται από δυο μέρη:

Απαιτείται: περιγράφονται οι συνθήκες που πρέπει να πληρούν τα δεδομένα εισόδου,

Επιστρέφεται: περιγράφεται το αποτέλεσμα της αλγοριθμικής επεξεργασίας (υπό τον όρο ότι τηρούνται οι συνθήκες για τα δεδομένα εισόδου).

Θα λέμε ότι ένας αλγόριθμος *επιλύει* ένα πρόβλημα όταν παράγει το σωστό αποτέλεσμα για κάθε στιγμιότυπο του προβλήματος. Η διάκριση ανάμεσα σε έναν αλγόριθμο και ένα πρόγραμμα (υπολογιστή) είναι ότι το δεύτερο αποτελεί υλοποίηση του πρώτου με εντολές ενός υπολογιστικού περιβάλλοντος (γλώσσας προγραμματισμού). Αντίστροφα, ένας αλγόριθμος είναι μία *μαθηματική αφαίρεση* ενός προγράμματος. Ουσιαστικά, ένα πρόγραμμα το οποίο περιέχει εντολές μίας γλώσσας προγραμματισμού και *εκτελείται (τρέχει)* σε έναν υπολογιστή, αντιστοιχεί σε έναν αλγόριθμο ο οποίος περιγράφεται σε *ψευδογλώσσα* οι εντολές της οποίας εκτελούνται σε ένα *θεωρητικό μοντέλο υπολογισμού*.

Ο όρος «ψευδογλώσσα» παραπέμπει σε γλώσσα περιγραφής ενός αλγόριθμου· μπορεί να είναι αρκετά κοντά στη φυσική γλώσσα αλλά και να χρησιμοποιεί βασικές δηλωτικές εκφράσεις μίας γλώσσας προγραμματισμού. Με τον τρόπο αυτό επιτυγχάνεται τυποποίηση στην περιγραφή των αλγορίθμων. Σαν παράδειγμα, θεωρούμε την περιγραφή για τον αλγόριθμο που επιλύει το πρόβλημα ελαχίστου που διατυπώθηκε παραπάνω (Αλγόριθμος 1). Η περιγραφή του μας επιτρέπει να διατυπώσουμε με καθαρό

Αλγόριθμος 1 Εύρεση του μικρότερου στοιχείου του συνόλου A

Απαιτείται: σύνολο $A \subset \mathbb{N}$.
Επιστρέφεται: min_el περιέχει το μικρότερο στοιχείο του συνόλου A.

```
1: min_el ← ∞;
2: for all a ∈ A do
3:     if a ≤ min_el then
4:         min_el ← a;
5:     end if
6: end for
```

συστηματικό τρόπο την κεντρική ιδέα παραλείποντας λεπτομέρειες που αφορούν την υλοποίηση όπως αυτή θα γινόταν σε μία συνήθη γλώσσα προγραμματισμού.[1] Για παράδειγμα, η υλοποίηση του Αλγόριθμου 1 σε μία γλώσσα προγραμματισμού, μεταξύ άλλων, πιθανώς απαιτεί: να απεικονιστεί με κάποιο τρόπο η τιμή ∞ (γραμμή 1), να χρησιμοποιηθεί μια δομή δεδομένων (πίνακας, λίστα) για την αναπαράσταση του συνόλου A (Γραμμή 2), να προσδιοριστεί ο τύπος των στοιχείων της δομής αυτής κλπ. Επιπλέον πλεονέκτημα της χρήσης ψευδογλώσσας αποτελεί η αφαιρετικότητα που επιτυγχάνεται αφού επιτρέπει την επέκταση της ιδέας που περιγράφεται σε διάφορες δομές· στον Αλγόριθμο 1 το A μπορεί να είναι υποσύνολο ενός οποιουδήποτε σύνολου διακριτών στοιχείων με σχέση *καλής διάταξης*.

Καθοριστικό ρόλο στη διαμόρφωση της ψευδογλώσσας είναι το θεωρητικό μοντέλο υπολογισμού. Αυτό καθορίζει τις λειτουργίες που επιτρέπεται να εκτελεί ο αλγόριθμος

[1] Παρατηρήστε την άμεση σχέση του αλγόριθμου με την μαθηματική έκφραση

$$min\_el = \min\{a \in A\}.$$

καθώς και το κόστος της κάθε λειτουργίας. Ένα ευρέως χρησιμοποιούμενο μοντέλο υπολογισμού είναι η *Μηχανή Άμεσης Προσπέλασης (ΜΑΠ) - Random Access Machine (RAM)*. Η μηχανή αυτή περιέχει μνήμη κάποιου μεγέθους, έναν επεξεργαστή με συγκεκριμένο αριθμό καταχωρητών στους οποίους μπορούμε να φορτώνουμε δεδομένα από τη μνήμη και να εκτελούνται αριθμητικές/λογικές πράξεις τα αποτελέσματα των οποίων αποθηκεύονται και πάλι στη μνήμη. Οι λειτουργίες αυτές υλοποιούνται με τη χρήση εντολών (της ΜΑΠ) οι οποίες εκτελούνται ακολουθιακά. Εναλλακτικό θεωρητικό μοντέλο υπολογισμού αποτελεί η *μηχανή Turing* με βάση την οποία έχει οριστεί η ιεραρχία των υπολογιστικών προβλημάτων. Τα δύο μοντέλα είναι ισοδύναμα.

Για τους αλγόριθμους που θα παρουσιαστούν στη συνέχεια θα υιοθετήσουμε το θεωρητικό μοντέλο ΜΑΠ. Για λόγους τυποποίησης και απλότητας, η ψευδογλώσσα που θα χρησιμοποιηθεί βασίζεται στο «πακέτο» algorithmicx του συστήματος στοιχειοθεσίας LaTeX (πρότυπο algpseudocode). Όπως φαίνεται και από την περιγραφή του Αλγόριθμου 1, ο κώδικας στην ψευδογλώσσα αυτή είναι άμεσα δηλωτικός και εύκολα μεταγράψιμος σε γλώσσα προγραμματισμού. Μία συνοπτική περιγραφή των εντολών (δομών) που θα χρησιμοποιηθούν για την περιγραφή των αλγορίθμων που θα παραθέσουμε στα επόμενα κεφάλαια ακολουθεί στην Παράγραφο 1.4.

1.2 Δομές Δεδομένων

Η αλγοριθμική διαδικασία συνίσταται στην επεξεργασία πληροφορίας προκειμένου να εξαχθεί ένα ζητούμενο αποτέλεσμα. Προκειμένου να γίνει η επεξεργασία αυτή η πληροφορία θα πρέπει να είναι οργανωμένη στη μνήμη της ΜΑΠ με κατάλληλο τρόπο. Ο όρος «δομές δεδομένων» αναφέρεται ακριβώς σε αυτό· υποδηλώνει τις *λογικές οντότητες*, που αντιστοιχούν σε χώρο στη μνήμη, οι οποίες αποθηκεύουν την πληροφορία (ισοδύναμα, τα δεδομένα) μαζί με ένα σύνολο βασικών λειτουργιών που επιτρέπεται να εκτελεστούν στα δεδομένα αυτά.

Στα μοντέλο ΜΑΠ η στοιχειώδης δομή δεδομένων είναι η *μεταβλητή (variable)*. Αυτή αποτελεί ένα *μνημονικό όνομα* που αντιστοιχεί σε κάποια θέση μνήμης (της ΜΑΠ) και ειδικότερα στην πληροφορία (τιμή) που αυτή περιέχει. Κύριες λειτουργίες που αφορούν την δομή αυτή αποτελούν:

- Ανάκληση της τιμής της μεταβλητής (από τη μνήμη). Η λειτουργία αυτή απαιτείται κυρίως για τον υπολογισμό μίας μαθηματικής ή λογικής έκφρασης στην οποία συμμετέχει η μεταβλητή. Επίσης χρησιμοποιείται όταν η μεταβλητή παίζει το ρόλο *δείκτη* σε μία σύνθετη δομή δεδομένων.
- Απόδοση (ισοδύναμα, εκχώρηση) τιμής σε μεταβλητή. Υπονοεί την αποθήκευση της τιμής αυτής στη μνήμη της ΜΑΠ, στην αντίστοιχη θέση, για μελλοντική της χρήση ή ανακατεύθυνση της σαν έξοδο του αλγόριθμου.

Βασική ιδιότητα της μεταβλητής πέρα από το όνομα της είναι ο *τύπος* της. Ο τύπος υποδηλώνει το χώρο στη μνήμη που δεσμεύεται διαμέσου της συγκεκριμένης μεταβλητής. Κυρίως όμως υποδηλώνει το είδος της πληροφορίας στην οποία αναφέρεται η μεταβλητή (π.χ. ακέραιος, πραγματικός, χαρακτήρας, λογική τιμή, κλπ).

Συνήθως η αναφορά στον όρο «δομή δεδομένων» αφορά πιο σύνθετες διατάξεις από αυτή της απλής μεταβλητής· ο όρος υποδηλώνει μία συστοιχία από ομάδες τιμών *(εγγραφές)*. Κάθε εγγραφή αποτελείται από τιμές όχι απαραίτητα ίδιου τύπου, οι οποίες συνήθως σχετίζονται με μία οντότητα. Για παράδειγμα, μία εγγραφή που αφορά τα στοιχεία ενός φοιτητή περιέχει την ηλικία του (αριθμός), το όνομα του (συμβολοσειρά χαρακτήρων), το φύλλο του (λογική τιμή), κλπ. Αυτά ονομάζονται και *πεδία* της

εγγραφής. Όπως είναι εμφανές για κάθε πεδίο θα πρέπει να είναι δηλωμένος ο τύπος του.

Μία δομή δεδομένων ονομάζεται *στατική* αν το μέγεθος της δεν αλλάζει κατά τη διάρκεια εκτέλεσης του αλγόριθμου. Δηλαδή είναι γνωστό εκ'των προτέρων το μέγεθος που θα καταλάβει στην κύρια μνήμη της ΜΑΠ. Παράδειγμα στατικών δομών είναι οι μεταβλητές και οι πίνακες (arrays). Αντίθετα όταν το μέγεθος (στη μνήμη) μίας δομής μεταβάλλεται τότε έχουμε μια *δυναμική* δομή δεδομένων. Παράδειγμα τέτοιας δομής αποτελούν *η λίστα, η ουρά (με προτεραιότητα), το λεξικό,* κλπ.
Οι παραπάνω δομές αφορούν την κύριο μνήμη της ΜΑΠ. Αυτή αντιστοιχεί στην *εσωτερική μνήμη (internal storage)* ενός σύγχρονου υπολογιστικού συστήματος. Αντίστοιχες δομές δεδομένων υπάρχουν και για την *εξωτερική μνήμη (external storage)*. Δηλαδή για την πληροφορία η οποία βρίσκεται αποθηκευμένη σε μέσα όπως σκληροί δίσκοι, memory sticks, CDs, DVDs, κλπ. Η βασική δομή δεδομένων σε αυτή την περίπτωση είναι το *αρχείο*.

Βασικές λειτουργίες επί των δομών δεδομένων αποτελούν: η *προσπέλαση* κάποιας εγγραφής και η *ανάκτηση* (λήψη-ανάγνωση) της, η *εισαγωγή* (προσθήκη) μίας νέας εγγραφής, η *διαγραφή* μίας εγγραφής, κλπ.
Πιο σύνθετες λειτουργίες, που σε κάποιες περιπτώσεις, προϋποθέτουν μερικές από τις παραπάνω είναι οι ακόλουθες.

Αντιγραφή: Δημιουργία μίας νέας δομής δεδομένων που περιέχει αντίγραφα εγγραφών μίας ήδη υπάρχουσας δομής.

Συγχώνευση: Οι εγγραφές δύο οι περισσοτέρων δομών ίδιου τύπου δεδομένων συγχωνεύονται σε μία δομή ακολουθώντας κάποια προτεραιότητα.

Προσάρτηση: Λειτουργία αντίστοιχη της συγχώνευσης με τη διαφορά ότι οι εγγραφές κάποιας δομής *εισάγονται μετά* τις εγγραφές κάποιας άλλης δομής ιδίου τύπου.

Αναζήτηση: Απαιτείται η ανάκτηση μίας εγγραφής της οποίας (κάποιες από τις) τιμές πληρούν συγκεκριμένες ιδιότητες. Μπορεί η αναζήτηση να αφορά παραπάνω από μία εγγραφή. Για παράδειγμα, αναζητούμε τα ονοματεπώνυμα όλων των υπαλλήλων με ηλικία μεγαλύτερη από τα σαράντα έτη.

Ταξινόμηση: Οι εγγραφές μίας δομής θέλουμε να διαταχθούν σε αύξουσα (φθίνουσα) σειρά σε σχέση με τις τιμές κάποιου(ων) από τα πεδία τους.

Η ευκολία ή η δυσκολία με την οποία εκτελούνται οι παραπάνω λειτουργίες εξαρτάται από τον τύπο της εκάστοτε δομής δεδομένων. Η δομή δεδομένων που θα χρησιμοποιηθεί στους αλγόριθμους που θα παρουσιαστούν στα επόμενα κεφάλαια (πέρα από τις απλές μεταβλητές) είναι κυρίως αυτή του *πίνακα ακεραίων* - θα αναφερθούμε αναλυτικά σε αυτή τη δομή και στις λειτουργίες που την αφορούν σε επόμενο κεφάλαιο. Όμως στις περισσότερες περιπτώσεις οι ίδιες αλγοριθμικές τεχνικές έχουν εφαρμογή και σε άλλες δομές δεδομένων αφού ουσιαστικά αυτό που αλλάζει είναι ο τρόπος που υλοποιούνται οι παραπάνω βασικές λειτουργίες. Στην κατεύθυνση αυτή έχει περιληφθεί ένα κεφάλαιο που αφορά *σειριακά αρχεία* στο οποίο περιγράφονται αλγόριθμοι ταξινόμησης/αναζήτησης που παρατίθενται σε προηγούμενο κεφάλαιο για πίνακες.

1.3 Ιδιότητες Αλγορίθμων

Ένας *καλός* αλγόριθμος πρέπει να «διέπεται» από τις ακόλουθες αρχές [13, Ενότητα 1.1].

Είσοδος - Έξοδος: Το εκάστοτε στιγμιότυπο πάνω στο οποίο εκτελείται ο αλγόριθμος αποτελεί την είσοδο, ενώ το παραγόμενο αποτέλεσμα την έξοδο. Ένας αλγόριθμος που επιλύει ένα πρόβλημα μπορεί να θεωρηθεί σαν μία συνάρτηση από το σύνολο των στιγμιοτύπων του προβλήματος στο σύνολο των αποτελεσμάτων. Τεχνικά, αν θεωρήσουμε ότι κάθε στιγμιότυπο είναι μία συμβολοσειρά από κάποιο *αλφάβητο* Σ και το παραγόμενο αποτέλεσμα είναι μία συμβολοσειρά από ένα αλφάβητο Γ τότε ο αλγόριθμος F μπορεί να οριστεί σαν συνάρτηση $F : S \to \Gamma^*$, όπου $S \subseteq \Sigma^*$ το σύνολο των στιγμιοτύπων των προβλήματος.[2]

Καθοριστικότητα: Ο Ορισμός 1 δηλώνει ότι κάθε βήμα ενός αλγόριθμου *(εντολή)* πρέπει να είναι *επακριβώς* ορισμένο· οι ενέργειες που συνεπάγεται η εντολή πρέπει να είναι αυστηρά και χωρίς αμφιβολία καθορισμένες.

Αποτελεσματικότητα: Κάθε εντολή θα πρέπει να είναι επαρκώς απλή και να μπορεί να εκτελεστεί τη συγκεκριμένη στιγμή στην οποία καλείται. Για παράδειγμα, η εντολή $z \leftarrow x/y$ δεν μπορεί να εκτελεστεί αν η μεταβλητή y έχει τιμή μηδέν (τη στιγμή της εκτέλεσης).

Τερματισμός: Η εκτέλεση του αλγόριθμου θα πρέπει να ολοκληρώνεται σε πεπερασμένο χρόνο: μετά από την εκτέλεση πεπερασμένου - αλλά πιθανώς αρκετά μεγάλου - αριθμού βημάτων.

1.4 Αποτύπωση Αλγορίθμων

Για λόγους ομοιομορφίας όλοι οι αλγόριθμοι που θα παρουσιαστούν στα επόμενα θα έχουν τη μορφή *συνάρτησης*. Μια συνάρτηση είναι μία αυτόνομη αλγοριθμική μονάδα (unit) η οποία μπορεί να εκτελεστεί στα πλαίσια ενός άλλου αλγόριθμου (καλών αλγόριθμος). Ο καλών αλγόριθμος χρησιμοποιεί το όνομα της συνάρτησης για να την εκτελέσει (καλέσει) και παρέχει σε αυτή τα δεδομένα που θα χρησιμοποιήσει η συνάρτηση. Οι τύποι των δομών στις οποίες βρίσκονται αποθηκευμένα τα δεδομένα δηλώνονται στον ορισμό της συνάρτησης μαζί με μνημονικά ψευδο-ονόματα για τις δομές αυτές. Αυτά αποτελούν τις παραμέτρους της συνάρτησης και δηλώνονται σε παρένθεση δίπλα από το όνομα της. Όταν η συνάρτηση καλείται στα πλαίσια κάποιου αλγόριθμου τότε για κάθε παράμετρο ο καλών αλγόριθμος παραθέτει το όνομα μίας «πραγματικής» δομής δεδομένων (π.χ. μεταβλητής) που έχει άμεση αναφορά στα περιεχόμενα κάποιας περιοχής της μνήμης. Θα ακολουθείται η σύμβαση ότι η συνάρτηση μπορεί κατά τη διάρκεια της εκτέλεσης της να αλλάζει τις τιμές των δομών που της παρέχει ο καλών αλγόριθμος· οι παράμετροι της συνάρτησης αποτελούν τη *διεπαφή* εισόδου-εξόδου της συνάρτησης με τον καλούντα αλγόριθμο. Επιπλέον η συνάρτηση έχει τη δυνατότητα να επιστρέφει στο όνομα της κάποια τιμή στον καλούντα αλγόριθμο. Στην περίπτωση αυτή θα πρέπει στον καλούντα αλγόριθμο η τιμή να εκχωρείται σε κάποια μεταβλητή (αντίστοιχου τύπου) ή να χρησιμοποιείται στα πλαίσια κάποιας έκφρασης. Συντακτικά λοιπόν ένας αλγόριθμος θα έχει τη μορφή

function <όνομα>(<παράμετροι>)
…
end function

[2] Ένα αλφάβητο Σ είναι ένα σύνολο από σύμβολα. Με Σ^* συμβολίζουμε το σύνολο των συμβολοσειρών που παράγονται από τα σύμβολα του Σ.

Ένα πρώτο παράδειγμα είναι η συνάρτηση Swap η οποία ανταλλάσσει τις τιμές των δύο μεταβλητών που θα αντιστοιχιστούν στις παραμέτρους n_1, n_2.

Αλγόριθμος 2 Ανταλλαγή τιμών δύο μεταβλητών

Απαιτείται: Ακέραιες μεταβλητές n_1, n_2.
Επιστρέφεται: n_1, n_2 με ανταλλαγμένες τιμές.

```
1: function Swap(int n_1, int n_2)
2:     n ← n_1;
3:     n_1 ← n_2;
4:     n_2 ← n;
5: end function
```

Αν η συνάρτηση επιστρέφει κάποια τιμή στο όνομα της, τότε αυτό γίνεται με τη χρήση της εντολής **return**. Η εντολή αυτή ακολουθείται από μία σταθερά ή μία μεταβλητή ή μία παράσταση που πρέπει να υπολογιστεί. Η τιμή που προκύπτει επιστρέφεται στο όνομα της συνάρτησης στον καλούντα αλγόριθμο.

Στη συνέχεια θα κάνουμε μία σύντομη παρουσίαση των δηλωτικών δομών (εντολών) που μπορούν να εμφανίζονται μέσα σε έναν αλγόριθμο. Στην περιγραφή των εντολών (που ακολουθεί) γίνεται χρήση των παρακάτω όρων μέσα σε $<>$. Συγκεκριμένα,

- <συνθήκη> υποδηλώνει μία λογική έκφραση που αποτιμάται σε `True` ή `False`,
- <εντολές> υποδηλώνει ένα σύνολο (block) εντολών που εκτελείται ακολουθιακά.

Μία εντολή υποδηλώνει οτιδήποτε από τα παρακάτω.

- Εκχώρηση τιμής σε μεταβλητή. Η μεταβλητή, ανάλογα με τον τύπο της, μπορεί να πάρει τιμή από μία παράσταση (έκφραση) αποτελούμενη από μεταβλητές (ή/και σταθερές) και τελεστές οι οποίοι ενεργούν στις μεταβλητές αυτές. Παραδείγματα αποτελούν,

  (i) $x \leftarrow 3$

  (ii) $y \leftarrow x + 5$

  Οι εκφράσεις (i), (ii) αποδίδουν αριθμητικές τιμές και επομένως οι μεταβλητές που λαμβάνουν αυτές τις τιμές (δηλαδή η x και y αντίστοιχα) θεωρούνται αντίστοιχου τύπου. Προφανώς, για να αποτιμηθεί μία έκφραση θα πρέπει προηγουμένως οι μεταβλητές που συμμετέχουν σε αυτή να έχουν πάρει τιμές και να πληρούται η αρχή της αποτελεσματικότητας (Ενότητα 1.3).

  Για λόγους συντομίας, στους αλγόριθμους που θα παρουσιάσουμε, δεν θα δηλώνεται ο τύπος μίας μεταβλητής· αυτός θα προκύπτει εμμέσως από την πρώτη ανάθεση τιμών που θα της γίνεται. Για τους ίδιους λόγους η αύξηση (μείωση) της τιμής μίας ακέραιας μεταβλητής κατά μία μονάδα θα υποδηλώνεται με τη χρήση του τελεστή $++$ $(--)$. Περαιτέρω, για λόγους παραστατικότητας, θα χρησιμοποιούνται συμβολισμοί και συναρτήσεις από τα μαθηματικά για την αναπαράσταση εκφράσεων που αποδίδουν αριθμητικές τιμές. Για παράδειγμα η εντολή

  $$y \leftarrow 5\sqrt{\frac{x^3}{z}}$$

  εκχωρεί στη μεταβλητή y το γινόμενο του 5 επί την τετραγωνική ρίζα του λόγου του κύβου του x δια z.

Όταν μία μεταβλητή εμφανίζεται και στο αριστερό και στο δεξί μέρος μίας εντολής εκχώρησης η προϋπάρχουσα τιμή της μεταβλητής χρησιμοποιείται για τον υπολογισμό της τιμής της παράστασης του δεξιού μέρους η οποία στη συνέχεια εκχωρείται στη μεταβλητή.

- Εκτέλεση ενός *block* εντολών υπο-συνθήκη. Η λειτουργία αυτή επιτυγχάνεται μέσω της εντολής **if**. Παρακάτω, απεικονίζονται τρεις «εκδόσεις» της εντολής αυτής.

if <συνθήκη 1> **then**
  <εντολές 1>
end if
	if <συνθήκη 1> **then**
	  <εντολές 1>
	else
	  <εντολές 2>
	end if
		if <συνθήκη 1> **then**
		  <εντολές 1>
		else if <συνθήκη 2> **then**
		  <εντολές 2>
		…
		else
		  <εντολές n>
		end if

(I)	(II)	(III)

Αν η <συνθήκη 1> έχει τιμή `True` τότε εκτελείται το block των εντολών <εντολές 1> ενώ αν έχει τιμή `False` αγνοούνται οι εντολές αυτές (Έκδοση (I)). Εναλλακτικά, αν θέλουμε να εκτελεστεί το block εντολών <εντολές 2> (όταν η <συνθήκη 1> είναι `False`) έχουμε την Έκδοση (II) της εντολής **if**. Τέλος στην Έκδοση (III) μπορούμε να έχουμε εκτέλεση διαφορετικών block εντολών καθένα από τα οποία εκτελείται όταν η αντίστοιχη συνθήκη αποτιμάται σε `True` ενώ οι προηγούμενες συνθήκες που εξετάστηκαν είχαν τιμή `False`. Στην περίπτωση που όλες οι συνθήκες αποτιμώνται σε `False` εκτελείται το block <εντολές n>.

- Δομή επανάληψης *(βρόχος)*

while <συνθήκη> **do**
  <εντολές>
end while

Επαναληπτικά εκτελείται η ακόλουθη διαδικασία: ελέγχεται η <συνθήκη> και αν αποτιμάται σε `True` εκτελείται το block <εντολές>. Η διαδικασία τερματίζει αν η συνθήκη αποτιμηθεί σε `False`. Στην περίπτωση αυτή, εκτελείται η πρώτη εντολή μετά την **end while**. Πριν την εκτέλεση της εντολής **while** θα πρέπει να έχουν λάβει τιμές όλοι οι μεταβλητές που συμμετέχουν στη <συνθήκη> ώστε να είναι δυνατή η αποτίμηση της (βήμα αρχικοποίησης). Επίσης (τουλάχιστον) μία από τις εντολές του block <εντολές> θα πρέπει να μεταβάλλει την τιμή κάποιας από τις μεταβλητές που συμμετέχουν στη συνθήκη. Αν δεν συμβαίνει αυτό και με δεδομένο ότι μετά το βήμα αρχικοποίησης η συνθήκη έχει αποτιμηθεί σε `True`, ο αλγόριθμος δεν θα πληροί την ιδιότητα του Τερματισμού (δες Παράγραφο 1.3).

Μία ειδική μορφή βρόχου αποτελεί η εντολή **for**. Η συντακτική μορφή της εντολής αυτής είναι

for <αρχικοποίηση μεταβλητής>; <συνθήκη>; <μεταβολή τιμής> **do**
  <εντολές>
end for

$a \leftarrow 1;$
$i \leftarrow 1;$
while $i \leq 3$ **do**
  $a \leftarrow 2a;$
  $i++;$
end while
	$a \leftarrow 1;$
	for $i \leftarrow 1; i \leq 3; i++$ **do**
	  $a \leftarrow 2a;$
	end for

Πίνακας 1.1: Υπολογισμός του 2^3 με δύο ισοδύναμους τρόπους

Όπως και στην εντολή **while**, εκτελείται το block <εντολές> όσο η <συνθήκη> αποτιμάται σε `True`- επαναληπτικά πρώτα ελέγχεται η συνθήκη και μετά εκτελούνται οι <εντολές>. Συνήθως χρησιμοποιείται στην περίπτωση που η <συνθήκη> αφορά μία μεταβλητή - τότε θα αναφερόμαστε στη μεταβλητή αυτή ως **δείκτη.** Τόσο η αρχικοποίηση του δείκτη όσο και η εντολή που επιφέρει μεταβολή στην τιμή του (<μεταβολή τιμής>) αποτελούν τμήμα του συντακτικού της εντολής. Στον Πίνακα 1.1 γίνεται υπολογισμός του 2^3 (τιμή της μεταβλήτής a μετά την εκτέλεση) τόσο με τη χρήση της δομής **while** όσο και με τη χρήση της δομής **for**.

Μία αφαιρετική έκδοση της εντολής **for** αποτελεί η εντολή **forall**. Έχει τη μορφή

forall <στοιχείο συλλογής> **do**
  <εντολές>
end for

Το <στοιχείο συλλογής> αντιπροσωπεύει την αναφορά σε στοιχείο μίας (διακριτής) συλλογής αντικειμένων. Η επαναληπτική δομή εκτελεί το block <εντολές> μία φορά για κάθε στοιχείο της συλλογής Χαρακτηριστικό παράδειγμα χρήσης της συγκεκριμένης δομής απεικονίζεται στον Αλγόριθμο 1, όπου συλλογή αποτελεί το σύνολο των φυσικών A το οποίο δίνεται σαν είσοδο στον αλγόριθμο.

- Εντολή **break**.
  Μπορεί να εμφανιστεί μέσα σε ένα block <εντολές> μίας από τις προηγούμενες δομές επανάληψης. Σταματά η εκτέλεση του block των εντολών και εκτελείται η εντολή που βρίσκεται μετά από **end while** ή **end for**. Συνήθως χρησιμοποιείται σε συνδυασμό με την εντολή **if**.

- Εντολή **print**.
  Χρησιμοποιείται για να αποστείλει στην τυπική μονάδα εξόδου (standrd output stream) τις τιμές των μεταβλητών που εμφανίζονται μετά την εντολή Για παράδειγμα, αν θέλουμε να εμφανιστούν στην έξοδο οι τιμές των μεταβλητών n, m γράφουμε
  print n, m;

- Κλήση συνάρτησης.
  Γίνεται με την παράθεση του ονόματος της συνάρτησης ακολουθούμενο από τις παραμέτρους-δομές δεδομένων εισόδου-εξόδου σε παρένθεση.

- Μία πρόταση με μονοσήμαντη ερμηνεία.

Παραδείγματα αποτύπωσης αλγορίθμων με τη χρήση των παραπάνω εντολών ακολουθούν.

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

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

Το παραπάνω ανάπτυγμα χρησιμοποιείται από τον Αλγόριθμο 3 για τον υπολογισμό της ποσότητας z. Ένα παράδειγμα κλήσης του αλγόριθμου είναι

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

Απαιτείται: θετικοί ακέραιοι n, a, t;
Επιστρέφεται: άθροισμα n όρων αριθμητικής προόδου με πρώτο όρο το a και βήμα t

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

$$z \leftarrow \text{Sum\_of\_Terms}(3, 2, 1).$$

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

j	sum	a (Γραμμή 6)
1	2	3
2	5	4
3	**9**	7

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

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

[3] Το πλεονέκτημα της διαδικασίας αυτής είναι ότι οι πράξεις που χρησιμοποιούνται εκτελούνται πολύ γρήγορα στο δυαδικό σύστημα αρίθμησης· ο πολλαπλασιασμός και η διαίρεση με το δύο πραγματοποιείται μέσω ολίσθησης (σε επίπεδο bit) καθώς και ο υπολογισμός του υπόλοιπου της διαίρεσης.

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

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

```
1:  function Russian_Mult(int a, int b)
2:      prod ← 0;
3:      while b > 0 do
4:          if b mod 2 = 1 then
5:              prod ← prod + a;
6:          end if
7:          b ← ⌊b/2⌋;
8:          a ← 2 · a;
9:      end while
10:     return prod;
11: end function
```

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

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

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

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

i	a	b
1	1	12
2	2	6
3	3	4

Πίνακας 1.4: Διαιρέτες του 12

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

```
1: function Divisors(int n)
2:     bound ← ⌊√n⌋;
3:     for i ← 1; i ≤ bound; i++ do
4:         if n mod i = 0 then
5:             a ← i;
6:             b ← n/i;
7:             print a, b;
8:         end if
9:     end for
10: end function
```

1.5 Ορθότητα

Δεδομένου ενός αλγόριθμου, ένα καίριο ερώτημα που τίθεται είναι αν πραγματικά επιλύει το πρόβλημα για το οποίο δημιουργήθηκε. Προφανώς η σωστή επίλυση συγκεκριμένων στιγμιοτύπων του προβλήματος δεν αποτελεί απόδειξη παρόλο που ενδυναμώνει την πεποίθησή μας ότι ο αλγόριθμος αυτός «δουλεύει», δηλαδή πληροί την προδιαγραφή του. Επομένως προϋπόθεση για να αποδείξουμε την ορθότητα ενός αλγόριθμου είναι να έχει διασαφηνιστεί πλήρως τι δέχεται σαν είσοδο καθώς και τι πρέπει να παράγει σαν έξοδο προκειμένου αυτή να αποτελεί λύση. Μετά την αποτύπωση της προδιαγραφής και της κωδικοποίησης, η ορθότητα του αλγόριθμου μπορεί να αποδειχθεί με τη χρήση εργαλείων όπως η *επαγωγή (induction)* ή η *αναλλοίωτη βρόχου (loop invariant)* (Κεφάλαιο 6).

Η παραπάνω θεώρηση βοηθά σημαντικά στο σχεδιασμό αλγορίθμων. Έχοντας κατανοήσει τους μηχανισμούς των αποδείξεων ορθότητας, μπορούμε να σχεδιάσουμε αλγόριθμους έχοντας κατά νου μια τέτοια απόδειξη. Αυτή η προσέγγιση καθιστά το σχεδιασμό σωστών αλγορίθμων πολύ πιο εύκολο. Δεύτερον, δουλεύοντας μέσω μιας απόδειξης ορθότητας - ή ακόμα και σκιαγραφώντας μια τέτοια απόδειξη - συχνά αποκαλύπτει *λεπτά λάθη* που θα ήταν δύσκολο να βρεθούν μόνο με δοκιμές. Τρίτον, η απόδειξη ορθότητας ενός αλγόριθμου οδηγεί στην κατανόηση του σε πολύ βαθύτερο επίπεδο. Επίσης η λειτουργία αυτή ενισχύει σημαντικά τις ικανότητες προγραμματισμού.

1.6 Ανάλυση Αλγορίθμων

Η ανάλυση ενός αλγόριθμου αφορά τον υπολογισμό των *πόρων* που ο αλγόριθμος «καταναλώνει» κατά την εκτέλεση του. Ο όρος «πόρος» αναφέρεται συνήθως στο χρόνο εκτέλεσης και στο χώρο - θέσεις στη μνήμη - που ο αλγόριθμος χρησιμοποιεί. Τα δύο αυτά μέτρα αποτελούν τα βασικά κριτήρια αποτελεσματικότητας των αλγορίθμων. Επίσης με βάση αυτά μπορούν να συγκριθούν αλγόριθμοι που επιλύουν το ίδιο πρόβλημα.

Στο παρόν εγχειρίδιο θα δοθεί έμφαση στη ανάλυση των αλγορίθμων με βάση το χρόνο εκτέλεσης. Το θεωρητικό μοντέλο υπολογισμού ΜΑΠ που θα υιοθετηθεί δεν είναι μακρυά από τα σύγχρονα υπολογιστικά συστήματα και άρα κατάλληλο να αποτελέσει την κύρια πλατφόρμα εκτέλεσης των αλγορίθμων που θα παρουσιαστούν.

Στο περιβάλλον αυτό ο *χρόνος εκτέλεσης* ανάγεται στον αριθμό των *στοιχειωδών υπολογιστικών βημάτων (ΣΥΒ)*. Σαν ΣΥΒ θεωρούνται οι βασικές αριθμητικές και λογικές πράξεις, η ανάθεση τιμής σε μεταβλητή και γενικότερα η προσπέλαση στη μνήμη για ανάγνωση ή εγγραφή, η κλήση συνάρτησης κλπ. Στην ανάλυση μας θα υιοθετήσουμε τις ακόλουθες απλουστευτικές υποθέσεις.

1. Καθε ΣΥΒ κοστίζει το ίδιο από άποψη χρόνου *(υπόθεση σταθερού κόστους)*. Προφανώς κάτι τέτοιο δεν ισχύει στην πράξη ούτε καν για τον ίδιο τύπο ΣΥΒ. Για παράδειγμα, η πρόσθεση απαιτεί συνήθως λιγότερο χρόνο απ'ότι ο πολλαπλασιασμός.

2. Σαν ΣΥΒ θα θεωρούμε μόνο τις αριθμητικές/λογικές πράξεις, τις εκχωρήσεις τιμών σε μεταβλητές και τις κλήσεις συναρτήσεων περιλαμβανομένων και των συναρτήσεων βιβλιοθήκης (π.χ., τετραγωνική ρίζα, υπόλοιπο διαίρεσης mod, πάτωμα ή ταβάνι ενός αριθμού, κλπ). Κατά την κλήση μίας συνάρτησης, η αντιστοίχηση μεταβλητών/τιμών στις παραμέτρους της δεν θα υπολογίζεται.

Οι υποθέσεις αυτές δεν επηρεάζουν τα αποτελέσματα της ανάλυσης και όπου θα υπάρχει διαφοροποίηση αυτό θα αναφέρεται.

Είναι προφανές ότι ο αριθμός των ΣΥΒ που εκτελεί ένας αλγόριθμος δεν είναι απαραίτητα ο ίδιος για κάθε στιγμιότυπο ενός προβλήματος. Για παράδειγμα, είναι αναμενόμενο ότι η εύρεση του ελάχιστου στοιχείου ανάμεσα σε δέκα ακέραιους να απαιτεί λιγότερα ΣΥΒ από ότι ανάμεσα σε χίλιους ακέραιους. Για αυτό είναι λογικό τα αποτελέσματα της ανάλυσης ενός αλγόριθμου (αριθμός ΣΥΒ) να εκφράζονται σε σχέση με το *μέγεθος του στιγμιότυπου (το μέγεθος της εισόδου)*. Σαν μέγεθος ενός στιγμιότυπου ορίζεται ο αριθμός των χαρακτήρων που απαιτούνται, μέσα από συγκεκριμένο *σχήμα κωδικοποίησης*, για την αναπαράσταση της συμβολοσειράς που περιγράφει το στιγμιότυπο. Το σχήμα κωδικοποίησης αφορά το αλφάβητο σύμβολα του οποίου χρησιμοποιούνται για την αναπαράσταση της εισόδου όσο και τον τρόπο με τον οποίο χρησιμοποιούνται τα σύμβολα αυτά στην αναπαράσταση. Για παράδειγμα, προκειμένου για ακεραίους μπορούμε να θεωρήσουμε το σχήμα που χρησιμοποιεί το δυαδικό αλφάβητο με την προσθήκη ενός διαχωριστικού χαρακτήρα. Ο τρόπος που τα σύμβολα του αλφάβητου κωδικοποιούν τους αριθμούς προκύπτει άμεσα από τη δυαδική αναπαράσταση κάθε ακεραίου.[4]

Επειδή η παραπάνω θεώρηση μπορεί να περιπλέξει αρκετά την ανάλυση, στην πράξη ακολουθείται μία πιο απλουστευτική προσέγγιση· το μέγεθος ενός στιγμιότυπου αντιπροσωπεύεται από μία παράμετρο, ονομαστικά n, η οποία παίρνει ακέραιες θετικές τιμές ($n \in \mathbb{N}$). Η τιμή της παραμέτρου n είναι ανάλογη του μεγέθους της συμβολοσειράς εισόδου και διαμορφώνεται ανάλογα με το πρόβλημα. Για παράδειγμα, σε ένα πρόβλημα ταξινόμησης ακεραίων το n εκφράζει το πλήθος τους, σε ένα πρόβλημα γραφημάτων τον αριθμό των κορυφών, κοκ.

Η αντιπροσώπευση του μεγέθους του στιγμιότυπου από την παράμετρο n μας δίνει τη δυνατότητα να εκφράσουμε τον αριθμό των ΣΥΒ που εκτελεί ένας αλγόριθμος σε σχέση με αυτή.

Παράδειγμα 1 (συνέχεια)**.** Θεωρούμε ότι το μέγεθος του στιγμιότυπου εκφράζεται από τον αριθμό των όρων της αριθμητικής προόδου, δηλαδή τη μεταβλητή n.
Ο Αλγόριθμος 3 εκτελεί δύο εκχωρήσεις για να αρχικοποιήσει τις μεταβλητές sum και j (Γραμμές 2,3), τρεις προσθέσεις και εκχωρήσεις - συνολικά 6 ΣΥΒ - για κάθε τιμή

[4] Προφανώς το μέγεθος της εισόδου εξαρτάται και από το σχήμα κωδικοποίησης, όμως αυτό δεν αποτελεί παράγοντα αν υιοθετήσουμε το ίδιο σχήμα για κάθε στιγμιότυπο ενός προβλήματος.

του δείκτη j (Γραμμές 5,6,7), που είναι μικρότερη-ίση με n. Εφόσον, ο δείκτης j (μέσα στο βρόχο) παίρνει n διαφορετικές τέτοιες τιμές (για $j = 2, \ldots, n+1$) ο συνολικός αριθμός των ΣΥΒ που εκτελούνται εσωτερικά στο βρόχο είναι ίσος με $6 \cdot n$. Επίσης εκτελούνται $n+1$ συγκρίσεις στη Γραμμή 4. Έτσι ο συνολικός αριθμός των ΣΥΒ είναι ίσος με $2 + 6n + n + 1 = 7n + 3$.

Παρόλο που ο αριθμός των ΣΥΒ διαμορφώνεται από την τιμή της παραμέτρου n, σε αρκετές περιπτώσεις δεν αποτελεί συνάρτηση της με την αυστηρή μαθηματική έννοια. Για παράδειγμα, στον Αλγόριθμο 4, αν θεωρήσουμε σαν μία είσοδο το στιγμιότυπο $a = 5, b = 11$, και σαν μία δεύτερη είσοδο αυτό για $a = 5, b = 9$, παρατηρούμε ότι στο πρώτο θα εκτελεστούν μία φορά παραπάνω τα ΣΥΒ που περιγράφονται στη Γραμμή 5 από ότι στο δεύτερο. Αυτό συμβαίνει παρόλο που τα δύο στιγμιότυπα είναι (περίπου) του ιδίου μεγέθους: η δυαδική αναπαράσταση των αριθμών του παραδείγματος είναι $11 = 1011_2, 9 = 1001_2, 5 = 101_2$ και επομένως κάθε στιγμιότυπο μπορεί να κωδικοποιηθεί από ίδιο αριθμό bits $(3 + 4 = 7)$.

Το παραπάνω παράδειγμα δείχνει ότι ο αριθμός των ΣΥΒ δεν διαμορφώνεται αποκλειστικά από το μέγεθος ενός στιγμιότυπου αλλά και από τις δομικές ιδιότητες που διέπουν τα δεδομένα που αποτελούν το στιγμιότυπο. Αν οι ιδιότητες αυτές δεν μας επιτρέπουν να εκτιμήσουμε επακριβώς τον αριθμό των ΣΥΒ που εκτελεί ένας αλγόριθμος μπορούμε να τον προσεγγίσουμε μέσω φραγμάτων όπως γίνεται στο επόμενο παράδειγμα.

Παράδειγμα 3 (συνέχεια)**.** Σε κάθε επανάληψη του βρόχου του Αλγόριθμου 5 διακρίνουμε δύο περιπτώσεις σε σχέση με τον αριθμό των ΣΥΒ. Συγκεκριμένα, οι εντολές στις Γραμμές 5-7 εκτελούνται ανάλογα με το αν η τιμή της μεταβλητής i διαιρεί την τιμή της παραμέτρου n $(i \mid n)$ ή όχι $(i \nmid n)$ ως εξής.

$i \nmid n$: Δεν εκτελούνται οι εντολές των Γραμμών 5-7. Ο αριθμός των ΣΥΒ στην επανάληψη συνίσταται στον υπολογισμό του υπόλοιπου $(n \bmod i)$ και στη σύγκριση του με το μηδέν- συνολικά δύο ΣΥΒ.

$i \mid n$: Επιπλέον των δύο ΣΥΒ της προηγούμενης περίπτωσης, εκτελούνται τα τρία ΣΥΒ των Γραμμών 5-6, ήτοι δύο εκχωρήσεις και το πηλίκο της διαίρεσης του n με το i - το πηλίκο αυτό είναι εξ' ορισμού ακέραιος αφού το i διαιρεί το n. Ο συνολικός αριθμός των ΣΥΒ στην επανάληψη, στην περίπτωση αυτή, είναι $2 + 3 = 5$.

Επομένως ο αριθμός των ΣΥΒ ποικίλει ανάμεσα με το πλήθος των διαιρετών του n. Αν το n είναι πρώτος για μία μόνο τιμή του δείκτη i του βρόχου της Γραμμής 3 (δηλαδή για $i = 1$) θα εκτελεστούν τα ΣΥΒ των Γραμμών 5-6. Στο άλλο άκρο μπορεί σε κάθε επανάληψη του βρόχου να έχουμε την περίπτωση όπου $i \mid n$ (πχ., $n = 12$). Επειδή ο αριθμός των επαναλήψεων είναι ίσος με $\lfloor\sqrt{n}\rfloor - 1 + 1 = \lfloor\sqrt{n}\rfloor$, ο συνολικός αριθμός των ΣΥΒ που εκτελείται στις γραμμές από 4 έως 8 κυμαίνεται από $2\lfloor\sqrt{n}\rfloor$ έως $5\lfloor\sqrt{n}\rfloor$. Ο αριθμός των ΣΥΒ που δηλώνονται στη Γραμμή 3 και αφορούν τον δείκτη i είναι $3\lfloor\sqrt{n}\rfloor + 2$. Επίσης εκτελούνται τρεις ΣΥΒ στη Γραμμή 2 του αλγόριθμου. Άρα ο συνολικός αριθμός των ΣΥΒ προσεγγίζεται από επάνω (άνω φράγμα) από την ποσότητα

$$5\lfloor\sqrt{n}\rfloor + 3\lfloor\sqrt{n}\rfloor + 2 + 3 = 8\lfloor\sqrt{n}\rfloor + 5, \tag{1.2}$$

και από τα κάτω (κάτω φράγμα) από[5]

$$2\lfloor\sqrt{n}\rfloor + 3\lfloor\sqrt{n}\rfloor + 2 + 3 = 5\lfloor\sqrt{n}\rfloor + 5. \tag{1.3}$$

[5]Το κάτω φράγμα μπορεί να βελτιωθεί παρατηρώντας ότι οι εντολές των Γραμμών 5, 6 θα εκτελεστούν τουλάχιστον μία φορά - ακριβώς μία φορά αν το n είναι πρώτος. Μέσω της ανάλυσης αυτής στο κάτω φράγμα προστίθεται η σταθερά 3.

Όπως θα δούμε σε επόμενο κεφάλαιο η ανάλυση εστιάζει στα φράγματα του αριθμού των ΣΥΒ παρά στον επακριβή προσδιορισμό του αριθμού τους σε κάποιο στιγμιότυπο.

1.7 Ασκήσεις

1. Ποιος είναι ο αριθμός των ΣΥΒ που εκτελεί ο ψευδικώδικας που απεικονίζεται στην πρώτη στήλη του Πίνακα 1.1. Είναι ίδιος με τον αριθμό των ΣΥΒ που εκτελεί ο ψευδοκώδικας που απεικονίζεται στη δεύτερη στήλη;

2. Θεωρούμε τις παρακάτω εξειδικεύσεις της εντολής **for** που παρουσιάστηκε στην Ενότητα 1.4:

   (α΄) **for** $i \leftarrow m; i \leq n; i++$ **do**

   (β΄) **for** $i \leftarrow m; i \leq n; i \leftarrow i + a$ **do**

   (γ΄) **for** $i \leftarrow 1; i \leq n; i \leftarrow i \cdot a$ **do**

   Έστω ότι τα m, n, a είναι φυσικοί αριθμοί μεγαλύτεροι της μονάδας και $n \geq a$. Για κάθε μία από τις παραπάνω εξειδικεύσεις, να υπολογίσετε τον αριθμό των επαναλήψεων που θα εκτελεστούν καθώς και τον αριθμό τον ΣΥΒ που αφορούν τον δείκτη i.

3. Να εκπονήσετε έναν αλγόριθμο που να δέχεται σαν είσοδο έναν μη-αρνητικό ακέραιο n και να υπολογίζει την παράσταση $n!$.

4. Έστω σύνολο n διαφορετικών χρωμάτων εκ'των οποίων επιλέγουμε k προκειμένου να δημιουργήσουμε μία παλέτα για να ζωγραφίσουμε έναν πίνακα. Να εκπονήσετε έναν αλγόριθμο ο οποίος να δέχεται σαν είσοδο δύο μη-αρνητικούς ακέραιους n, k, όπου $n \geq k \geq 1$ και να υπολογίζει τον αριθμό των διαφορετικών παλετών.

5. Να εκπονήσετε έναν αλγόριθμο που να δέχεται σαν είσοδο έναν μη-αρνητικό ακέραιο n και υπολογίζει τον μικρότερο ακέραιο k για τον οποίο ισχύει ότι $n \leq 2^k$.

6. Να εκπονήσετε τον αλγόριθμο που χρησιμοποιεί το ΑΤΜ μίας τράπεζας.

7. Στο Παράδειγμα 1 δείξαμε ότι ο αριθμός των ΣΥΒ του Αλγόριθμου 3 είναι $7n+3$. Να εκπονήσετε αλγόριθμο ο οποίος να υπολογίζει ότι και ο Αλγόριθμος 3 αλλά με μικρότερο αριθμό ΣΥΒ.

8. Να εκπονήσετε έναν αλγόριθμο που να υπολογίζει το πλήθος των τριψήφιων αριθμών για τους οποίους κανένα ψηφίο δεν είναι ίδιο.

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

10. Να υπολογίσετε την τιμή που επιστρέφει η συνάρτηση που περιγράφεται στους Αλγόριθμους 6, 7 και 8.

11. Να υπολογίσετε τον αριθμό των ΣΥΒ που εκτελούνται από τον Αλγόριθμο 6.

Αλγόριθμος 6

Απαιτείται: ακέραιος n
Επιστρέφεται: ακέραιος r

```
function Guess01(int n)
    r ← 0;
    for i ← 1; i ≤ n − 1; i + + do
        for j ← i + 1; j ≤ n; j + + do
            for k ← 1; k ≤ j; k + + do
                r + +;
            end for
        end for
    end for
    return r;
end function
```

Αλγόριθμος 7

Απαιτείται: ακέραιος n
Επιστρέφεται: ακέραιος r

```
function Guess02(int n)
    r ← 0;
    for i ← 1; i ≤ n; i + + do
        for j ← 1; j ≤ i; j + + do
            for k ← j; k ≤ i + j; k + + do
                r + +;
            end for
        end for
    end for
    return r;
end function
```

Αλγόριθμος 8

Απαιτείται: ακέραιος n
Επιστρέφεται: ακέραιος r

```
function Guess03(int n)
    r ← 0;
    for i ← 1; i ≤ n − 1; i + + do
        for j ← 1; j ≤ i; j + + do
            for k ← j; k ≤ i + j; k + + do
                for l ← 1; l ≤ i + j − k; l + + do
                    r + +;
                end for
            end for
        end for
    end for
    return r;
end function
```

12. Έστω n, a φυσικοί αριθμοί μεγαλύτεροι της μονάδας. Να υπολογίσετε το πλήθος των επαναλήψεων του βρόχου

    for $t \leftarrow n; t \geq 1; t \leftarrow \lfloor \frac{t}{a} \rfloor$ **do**

13. Έστω πολυώνυμο $p(n) = \sum_{i=0}^{d} a_i n^i$, όπου $n \in \mathbb{N}$ και d μη-αρνητικός ακέραιος. Να εκπονήσετε αλγόριθμο που να δέχεται σαν είσοδο τους αριθμούς n, $a_0, a_1, \ldots, a_d$ και να υπολογίζει την τιμή $p(n)$ χωρίς να χρησιμοποιεί τελεστή που να υπολογίζει τη δύναμη. Ο αριθμός των ΣΥΒ του αλγόριθμου σας θα πρέπει να είναι συνάρτηση της παραμέτρου d.

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