ΗΥ240: Δομές Δεδομένων

Διδάσκουσα: Παναγιώτα Φατούρου

Υποχρεωτικό Μάθημα 2ου έτους
Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Ενότητα 1
Εισαγωγή

Εισαγωγικά Θέματα

- Αντικείμενο του μαθήματος των Δομών Δεδομένων είναι η αναπαράσταση και η διαχείριση συνόλων αντικειμένων (δεδομένα), τα οποία επιδέχονται πράξεις εξαγωγής πληροφορίας ή αλλαγής της συνθέσεως τους.

Εισαγωγικά Θέματα

Αφηρημένοι Τύποι Δεδομένων

Ένα ή περισσότερα σύνολα αντικειμένων και μια συλλογή λειτουργιών (πράξεων) επί των στοιχείων των συνόλων.

Παράδειγμα (εύρεση στοιχείου σε πίνακα)

- Τα δεδομένα είναι κάποιου τύπου, έστω Type, και υπάρχει μια γραμμική διάταξη ανάμεσά τους:
  - $\forall$ *u,v τύπου Type, είτε u < v, ή v < u, ή v = u.*
- Δίδεται ένα σύνολο S από στοιχεία τύπου Type και ζητείται απάντηση στο ερώτημα: «u $\in$ S?»
- **Σύνολα αντικειμένων**: στοιχεία τύπου Type (π.χ., u,v) και πεπερασμένα σύνολα αυτών (π.χ., S)
- **Σύνολο λειτουργιών**: Απάντηση της ερώτησης «u $\in$ S?», όπου: u είναι στοιχείο τύπου Type και S είναι πεπερασμένο σύνολο από στοιχεία τύπου Type.

Εισαγωγικά Θέματα – Δομές Δεδομένων

- Μια δομή δεδομένων υλοποιεί έναν αφηρημένο τύπο δεδομένων.

- Μια δομή δεδομένων επομένως συμπεριλαμβάνει:
  - ένα σύνολο δεδομένων τα οποία επιδέχονται επεξεργασία μέσω του συνόλου λειτουργιών που υποστηρίζονται από τον αφηρημένο τύπο δεδομένων που υλοποιεί η δομή
  - μια δομή αποθήκευσης των δεδομένων (π.χ., έναν πίνακα, μια λίστα, κλπ.)
  - ένα σύνολο από ορισμούς συναρτήσεων/διαδικασιών, όπου η κάθε συνάρτηση/διαδικασία αντιστοιχίζεται σε μια από τις λειτουργίες της δομής,
  - ένα σύνολο από «αλγόριθμους», κάθε ένας εκ των οποίων υλοποιεί μια από τις παραπάνω συναρτήσεις/διαδικασίες.

Εισαγωγικά Θέματα – Δομές Δεδομένων

Οι βασικές λειτουργίες που υποστηρίζει μια δομή δεδομένων είναι:

- Προσπέλαση
- Αναζήτηση
- Εισαγωγή
- Διαγραφή
- Ταξινόμηση
- Αντιγραφή
- Συγχώνευση
- Διαχωρισμός

Εισαγωγικά Θέματα – Αλγόριθμοι

- *Αλγόριθμος* είναι μια πεπερασμένη ακολουθία υπολογιστικών βημάτων (ή εντολών) αυστηρά καθορισμένων (που κάθε ένα εκτελείται σε πεπερασμένο χρόνο), τα οποία αν ακολουθηθούν επιλύεται κάποιο πρόβλημα.

- Ο αλγόριθμος δέχεται κάποια τιμή ή κάποιο σύνολο τιμών ως είσοδο και δίνει κάποια τιμή ή κάποιο σύνολο τιμών ως έξοδο.
  - *Είσοδος*: Δεδομένα που παρέχονται στον αλγόριθμο (εξ αρχής).
  - *Έξοδος*: Δεδομένα που αποτελούν το αποτέλεσμα του αλγορίθμου.

- **Πρόγραμμα**
  Υλοποίηση ενός αλγορίθμου σε κάποια γλώσσα προγραμματισμού.

Ένα Απλό Παράδειγμα Αλγορίθμου

Ύψωση ενός αριθμού x σε μια ακέραια δύναμη n

Algorithm Power(x, n) {// **Περιγραφή με C-like ψευδο-κώδικα**
// Είσοδος: ένας πραγματικός αριθμός x και ένας ακέραιος αριθμός n
// Έξοδος: ένας πραγματικός αριθμός που ισούται με x^n

```
    int j = 0;
    double y = 1;

    while (j < n) {
          y = y*x;
          j = j+1;
    }
    return y;
}
```

Τεχνικές Απόδειξης

Χρήση παραδείγματος ή αντιπαραδείγματος

- **Ισχυρισμός:** «Υπάρχει τουλάχιστον ένα στοιχείο x στο σύνολο S που έχει την ιδιότητα P» -> Για να δείξουμε ότι ο ισχυρισμός ισχύει, αρκεί να βρούμε ένα στοιχείο x του S που ικανοποιεί την P.

- **Ισχυρισμός:** «Κάθε στοιχείο του S ικανοποιεί την ιδιότητα P» -> Για να δείξουμε ότι ο ισχυρισμός δεν ισχύει, αρκεί να βρούμε ένα στοιχείο x του S που δεν έχει την P.

Τεχνικές Απόδειξης

Αντιθετο-αντιστροφή & Απαγωγή εις Άτοπο

- **Ισχυρισμός:** «Αν το γινόμενο ab είναι περιττό, τότε και το a είναι περιττό και το b είναι περιττό»

  - Αντιθετο-αντιστροφή: Για να αποδειχθεί ότι «αν ο ισχυρισμός p είναι αληθής τότε και ο ισχυρισμός q είναι αληθής», αποδεικνύουμε ότι «αν ο ισχυρισμός NOT(q) είναι αληθής, τότε και ο NOT(p) είναι αληθής».
    - Αποδεικνύουμε ότι «Αν ή το a είναι άρτιο ή το b είναι άρτιο, τότε το γινόμενο ab είναι άρτιο» . Έστω, χωρίς βλάβη της γενικότητας ότι το a είναι άρτιο, δηλαδή a = 2*k, για κάποιο ακέραιο k. Τότε, το ab = (2*k)*b = 2 * (k*b) είναι άρτιο. Η περίπτωση που το b είναι άρτιο είναι παρόμοια.

  - Με εις άτοπο απαγωγή.
    - Έστω ότι το ab είναι περιττό. Υποθέτουμε για να καταλήξουμε σε άτοπο πως ή το a είναι άρτιο ή το b είναι άρτιο. Έστω, χωρίς βλάβη της γενικότητας, πως το b είναι άρτιο, δηλαδή b = 2 * k, για κάποιο ακέραιο k. Τότε, το ab = a* (2*k) = 2*(a*k) είναι άρτιο.
      Αυτό αντιτίθεται στην υπόθεσή μας ότι το ab είναι περιττό.
      Η περίπτωση που το a είναι άρτιο είναι παρόμοια.

Μαθηματική Επαγωγή

Ζητείται να αποδειχθεί πως ένας ισχυρισμός Ι, που σχετίζεται με κάποια μεταβλητή j, ισχύει για κάθε τιμή της μεταβλητής j, όπου η j παίρνει τιμές από μια ακολουθία τιμών $x_1, x_2, \ldots, x_n$.

Αποδεικνύουμε τα εξής:

- Αν ο ισχυρισμός Ι ισχύει για κάποια τιμή του j (π.χ., την x_k) που μπορεί να είναι οποιαδήποτε τιμή εκτός της τελευταίας της ακολουθίας τιμών της j, τότε ο Ι ισχύει και για την επόμενη τιμή της ακολουθίας τιμών της j (δηλαδή για την x_{k+1}). **(1)**
- Ο Ι ισχύει για την πρώτη τιμή της ακολουθίας τιμών της j (την x_1). **(2)**

- Εφόσον από **(2)** ο ισχυρισμός ισχύει για την x_1, από **(1)** ισχύει και για την x_2. **(3)**
- Εφόσον από **(3)** ο ισχυρισμός ισχύει για την x_2, από **(1)** ισχύει και για την x_3. **(4)**
- Εφόσον από **(4)** ο ισχυρισμός ισχύει για την x_3, από **(1)** ισχύει και για την x_4. **(5)**
- ... **(...)**
- Εφόσον από **(...)** ο ισχυρισμός ισχύει για την x_{n-1}, από **(1)** ο ισχυρισμός ισχύει και για την x_n.

☺ ***Ο ισχυρισμός ισχύει για όλες τις τιμές της j!!!***

Μαθηματική Επαγωγή - Παράδειγμα

Ορθότητα Αλγόριθμου Power

Συμβολίζουμε με y_j την τιμή της μεταβλητής y στην αρχή της j-οστής ανακύκλωσης, j = 1, …, n+1. Θα δείξουμε πως $y_j = x^{j-1}$, $\forall j=1, \ldots, n+1$.

Με επαγωγή στο j.

Βάση επαγωγής

j = 1. Αρχικά, $y_1 = 1 = x^0 = x^{1-1} = x^{j-1}$.

Επαγωγική Υπόθεση

Έστω ότι για κάποιο k, $1 \leq k < n+1$, $y_k = x^{k-1}$ (1)

Επαγωγικό Βήμα

Θα δείξουμε ότι ο ισχυρισμός ισχύει για (k+1): $y_{k+1} = x^k$.

Από αλγόριθμο: $y_{k+1} = y_k * x$.

Από επαγωγική υπόθεση: $y_k = x^{k-1}$.

Άρα, $y_{k+1} = x^k$, όπως απαιτείται.

```
Algorithm Power(x, n)
   int j = 0;
   double y = 1;

   while (j < n) {
         y = y*x;
         j = j+1;
   }
   return y;
}
```

Μαθηματική Επαγωγή – Ισχυρή Επαγωγή

Για κάποιους ισχυρισμούς Ι, δεν μπορεί να αποδειχθεί η (1), **αλλά μπορεί να αποδειχθεί ότι:**

- Αν ο ισχυρισμός Ι ισχύει **για κάθε** τιμή του j που προηγείται στην ακολουθία τιμών της j από οποιαδήποτε τιμή $x_k \neq x_n$ (δηλαδή αν ο ισχυρισμός ισχύει για τις τιμές $x_1, x_2, \ldots, x_{k-1}$), τότε ο Ι ισχύει και για την x_k. **(1')**

Αποδεικνύεται επίσης ότι:

- Ο Ι ισχύει για την πρώτη τιμή της ακολουθίας τιμών της j (την x_1). **(2)**

Τότε:

- αφού από **(2)** ο ισχυρισμός ισχύει για την x_1, από **(1')** ισχύει και για την x_2. **(3)**
- αφού από **(2)** και **(3)** ο ισχυρισμός ισχύει για τις x_1, x_2, από **(1')** ισχύει και για την x_3. **(4)**
- αφού από **(2), (3)** και **(4)** ο ισχυρισμός ισχύει για τις x_1, x_2, x_3, από **(1')** ισχύει και για την x_4. **(5)**
- … **(…)**
- αφού από **(2), (3), (4), (5), …, (…)** ο ισχυρισμός ισχύει για τις $x_1, x_2, x_3, x_4, \ldots, x_{n-1}$, από **(1')** ο ισχυρισμός ισχύει για την x_n.

☺ *Ο ισχυρισμός ισχύει για όλες τις τιμές της j!!!*

Μαθηματική Επαγωγή – Ισχυρή Επαγωγή

Αιγυπτιακός Πολλαπλασιασμός

Δίνεται η συνάρτηση:

$$m(x,y) = \begin{cases} 0, & \text{αν } y = 0 \\ m(x+x, y/2), & \text{αν } y \text{ άρτιος } \& \neq 0 \\ x + m(x, y-1), & \text{αν } y \text{ περιττός } \& \neq 0 \end{cases}$$

Θα δείξω ότι $m(x,y) = x*y$, $\forall$ θετικό ακέραιο x,y

Απόδειξη: Με επαγωγή στο y.

Βάση της επαγωγής: Αν y = 0, m(x,y) = 0, αλλά και x*y = 0, οπότε ο ισχυρισμός ισχύει.

Επαγωγική Υπόθεση: Έστω οποιαδήποτε τιμή $y > 0$. Υποθέτουμε ότι $m(x,z) = x*z$, για κάθε τιμή z, $0 \leq z < y$.

Επαγωγικό Βήμα: Αποδεικνύουμε τον ισχυρισμό για την τιμή y, δηλαδή αποδεικνύουμε πως $m(x,y) = x*y$. Διακρίνουμε περιπτώσεις:

- Ο y είναι περιττός: $m(x,y) = x + m(x,y-1)$. Από επαγωγική υπόθεση ($z = y-1 < y$) ισχύει ότι $m(x,y-1) = x*(y-1)$. Άρα: $m(x,y) = x + x*(y-1) = x + x*y - x = x*y$.
- Ο y είναι άρτιος: $m(x,y) = m(x+x, y/2)$. Από επαγωγική υπόθεση ($z = y/2 < y$) ισχύει ότι $m(x,y) = (x+x)*y/2 = x*y$. Άρα, $m(x,y) = x*y$.
- Και στις δύο περιπτώσεις, ο ισχυρισμός ισχύει.

Κριτήρια Επιλογής Αλγορίθμων

- Χρόνος Εκτέλεσης (χρονική πολυπλοκότητα ή πολυπλοκότητα χρόνου)
- Απαιτούμενος χώρος – αποθηκευτικές θέσεις (θέσεις μνήμης) που χρησιμοποιούνται (χωρική πολυπλοκότητα ή πολυπλοκότητα χώρου)
- Ευκολία προγραμματισμού
- Γενικότητα

Ανάλυση Αλγορίθμου αποκαλείται η εύρεση των πόρων (χρόνος, χώρος) που αυτός απαιτεί για να εκτελεστεί.

Μας ενδιαφέρει κυρίως η χρονική και η χωρική πολυπλοκότητα. Οι δύο αυτοί παράμετροι καθορίζουν την αποδοτικότητα του αλγορίθμου.

Το μοντέλο υπολογισμού αποτελεί την αφαιρετική θεώρηση του υλικού που διατίθεται για την εκτέλεση του αλγορίθμου.

Το Μοντέλο Υπολογισμού RAM

Μηχανή Τυχαίας Προσπέλασης (Random Access Machine, RAM)

Το σύστημα παρέχει:

- τους αναγκαίους καταχωρητές (registers)
- έναν συσσωρευτή (accumulator)
- μια ακολουθία αποθηκευτικών θέσεων (memory cells) με διευθύνσεις 0, 1, 2, … που αποτελούν την κύρια μνήμη

Το σύστημα είναι σε θέση:

- να εκτελεί αριθμητικές πράξεις (+, - , *, /, mod)
- να παίρνει αποφάσεις διακλαδώσεως (if… else…) βάσει των τελεστών (==, < , > , =<, >=, !=)
- να διαβάζει ή να γράφει από και προς οποιαδήποτε θέση μνήμης.

Το Μοντέλο Υπολογισμού RAM

Στοιχειώδεις εντολές

- Καταχώρηση τιμής σε μια μεταβλητή
- Κλήση μιας μεθόδου
- Εκτέλεση μιας αριθμητικής πράξης
- Σύγκριση δύο αριθμών
- Δεικτοδότηση πίνακα (συμβολίζουμε με $A[s..u]$ έναν πίνακα του οποίου η πρώτη θέση δεικτοδοτείται από την τιμή s ενώ η τελευταία από την τιμή u, συμβολίζουμε επίσης με $A[i]$ το i-οστό στοιχείο του πίνακα A, $s =< i =< u$)
- Πρόσβαση στη διεύθυνση μνήμης που δείχνει ένας δείκτης
- Επιστροφή από μια μέθοδο (return)

Το Μοντέλο Υπολογισμού RAM

Μέτρηση Μοναδιαίου Κόστους

- Ο χρόνος εκτέλεσης κάθε στοιχειώδους εντολής εξαρτάται από το υλικό (είναι ανεξάρτητος από τη γλώσσα προγραμματισμού) και ισούται με κάποια σταθερά. Θεωρούμε ότι κάθε τέτοια εντολή εκτελείται σε μία μονάδα χρόνου.

Μέτρηση Λογαριθμικού Κόστους

- Η στοιχειώδης εντολή απαιτεί χρόνο ανάλογο του μήκους της δυαδικής αναπαράστασης των τελεσταίων.

  Παράδειγμα: Η μετακίνηση ενός αριθμού n από την κύρια μνήμη προς έναν καταχωρητή μπορεί να απαιτεί $\lfloor \log n \rfloor + 1$ μονάδες χρόνου.

Συνήθως, πραγματοποιείται ανάλυση των αλγορίθμων βάσει μετρήσεως μοναδιαίου κόστους (εκτός και αν γίνεται εκτενής χρήση πράξεων επί συμβολοσειρών bit).

Χρονική Πολυπλοκότητα

- Εισάγεται μια μεταβλητή n που εκφράζει το μέγεθος της εισόδου.

Παραδείγματα

- Στο πρόβλημα ανυψώσεως σε δύναμη το μέγεθος αυτό είναι το n, η δύναμη στην οποία πρέπει να υψωθεί ο δεδομένος αριθμός.
  - *Η ύψωση ενός αριθμού στη δύναμη 10 απαιτεί την εκτέλεση περισσότερων στοιχειωδών εντολών από την ύψωση του αριθμού σε μια μικρότερη δύναμη.*
- Σε ένα πρόβλημα ταξινόμησης ενός πίνακα, το μέγεθος του προβλήματος είναι το πλήθος των στοιχείων του πίνακα.
  - *Η ταξινόμηση 1000 αριθμών απαιτεί περισσότερο χρόνο από την ταξινόμηση 10 αριθμών.*

- **Αυτό ισχύει γενικότερα:**
  - *«Ο χρόνος που απαιτεί ένας αλγόριθμος για να εκτελεστεί συχνά εξαρτάται από το μέγεθος της εισόδου»!*

Χρόνος Εκτέλεσης για συγκεκριμένη είσοδο

- Ο χρόνος εκτέλεσης (running time) ή η χρονική πολυπλοκότητα ενός αλγορίθμου **για μια συγκεκριμένη είσοδο** είναι το πλήθος των στοιχειωδών εντολών που εκτελούνται κατά την εκτέλεση του αλγορίθμου με αυτήν την είσοδο.

Υπολογίζοντας το πλήθος των στοιχειωδών εντολών – Αλγόριθμος Power

Algorithm Power(x, n)

```
    int j = 0;              ──────→ 1
    double y = 1;           ──────→ 1

    while (j < n) {         ──────→ n+1
        y = y*x;            ──────→ 2*n
        j = j+1;            ──────→ 2*n
    }
    return y;               ──────→ 1
}
```

Πλήθος Στοιχειωδών Εντολών

- T(n) = 1 + 1 + n+1 + 2*n + 2*n + 1 = 5n + 4
- Ο παραπάνω τύπος ισχύει για οποιαδήποτε τιμή του n.
- Το πλήθος των στοιχειωδών εντολών που εκτελεί ο αλγόριθμος είναι μια συνάρτηση του μεγέθους n της εισόδου!

Ένα Ακόμη Παράδειγμα

Πρόβλημα

Είσοδος
Μια ακολουθία από n αριθμούς
$\langle a_1, a_2, ..., a_n \rangle$.

Έξοδος (output):
Μια μετάθεση (αναδιάταξη)
$\langle a'_1, a'_2, ..., a'_n \rangle$ της ακολουθίας εισόδου έτσι ώστε:

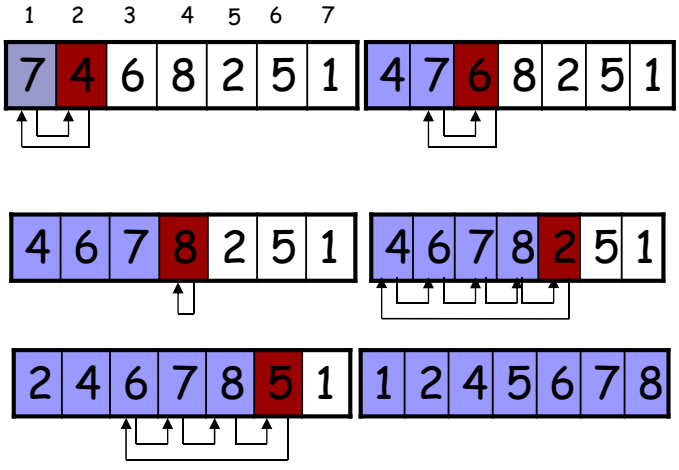
$$a'_1 \leq a'_2 \leq ... \leq a'_n$$

```
Algorithm InsertionSort (A[1..n]) {
// Είσοδος: ένας μη-ταξινομημένος
    πίνακας A ακέραιων αριθμών
// Έξοδος: ο πίνακας A με τα στοιχεία
    του σε αύξουσα διάταξη
    int key, i, j;
    for (j = 2; j ≤ n; j++) {
        key = A[j];
        i = j-1;
        while (i > 0 && A[i] > key) {
            A[i+1] = A[i];
            i = i-1;
        }
        A[i+1] = key;
    }
    return A;
}
```

Πως λειτουργεί ο αλγόριθμος αν A = <7,4,6,8,2,5,1>;

Ένα Ακόμη Παράδειγμα

```
Algorithm InsertionSort (A[1..n])
{
    int key, i, j;
    for (j = 2; j ≤ n; j++) {
        key = A[j];
        i = j-1;
        while (i > 0 && A[i] > key) {
            A[i+1] = A[i];
            i = i-1;
        }
        A[i+1] = key;
    }
    return A;
}
```



Πως λειτουργεί ο αλγόριθμος αν
A = <7,4,6,8,2,5,1>;

Υπολογίζοντας το πλήθος των στοιχειωδών εντολών – Αλγόριθμος InsertionSort

```
Algorithm InsertionSort(A[1..n]) {                          Κόστος
    int key, i, j;
    for (j = 2; j =< n; j=j+1) {          ---------->   1+ n + 2*(n-1)
        key = A[j];                       ---------->   2* (n-1)
        i = j-1;                          ---------->   2*(n-1)
        while (i > 0 && A[i] > key) {     ---------->   4 * Σ_{j=2..n} s_j
            A[i+1] = A[i];                ---------->   4 * Σ_{j=2..n} (s_j - 1)
            i = i-1;                      ---------->   2 * Σ_{j=2..n} (s_j - 1)
        }
        A[i+1] = key;                     ---------->   3 * (n-1)
    }
    return A;                             ---------->   1
}
```

s_j: ο αριθμός των φορών που ο έλεγχος του while loop εκτελείται για τη συγκεκριμένη τιμή του j, $2 \leq j \leq n$.

Υπολογίζοντας το πλήθος των στοιχειωδών εντολών – Αλγόριθμος InsertionSort

Συνολικός Χρόνος Εκτέλεσης

$T(n) = 1 + n + 2*(n-1) + 2 * (n-1) + 2*(n-1) + 4 * \Sigma s_j + 4* \Sigma (s_j-1) + 2 * \Sigma (s_j-1) + 3*(n-1) + 1$

$= 10n - 7 + 10*\Sigma_{j=2..n} s_j - 6* \Sigma_{j=2..n} 1$

$= 10n - 7 + 10* \Sigma_{j=2..n} s_j - 6 * (n-1)$

$= 4n - 1 + 10 * \Sigma_{j=2..n} s_j$

Ποιος είναι ο καλύτερος χρόνος εκτέλεσης που μπορεί να επιτευχθεί από την InsertionSort?

Ο πίνακας είναι εξ αρχής ταξινομημένος σε αύξουσα διάταξη.

Τότε?

$s_j = 1, \forall\ j = 2, \ldots, n.$

*Επομένως, T(n) = 4n - 1+10 *(n-1) = 14n – 11*

⇒ **γραμμική συνάρτηση του n.**

Υπολογίζοντας το πλήθος των στοιχειωδών εντολών – Αλγόριθμος InsertionSort

Ποιος είναι ο χειρότερος χρόνος εκτέλεσης κατά την εκτέλεση της InsertionSort?

Τα στοιχεία του πίνακα είναι σε φθίνουσα διάταξη.

Τότε?

$s_j = j, \forall\ j = 2, \ldots, n$, και

$\Sigma_{j=2..n}\ s_j = \Sigma_{j=2..n}\ j \quad = 2 + \ldots + n = (n+2)(n-1)/2$

Επομένως:

$T(n) = 4n - 1 + 5(n+2)(n-1) = 5n^2 + 9n - 11$

$\Rightarrow$ **τετραγωνική συνάρτηση του n.**

Ανάλυση Χειρότερης περίπτωσης

- Ο χείριστος χρόνος εκτέλεσης (ή η χρονική πολυπλοκότητα χειρότερης περίπτωσης) ενός αλγορίθμου ορίζεται να είναι ο **μέγιστος** χρόνος εκτέλεσης για οποιαδήποτε είσοδο με συγκεκριμένο μέγεθος n (και συνήθως είναι συνάρτηση του n).

- Μας ενδιαφέρει να βρούμε το χαμηλότερο άνω φράγμα και το υψηλότερο κάτω φράγμα.

Ανάλυση Αναμενόμενης Περίπτωσης

- Αν είναι γνωστή κάποια κατανομή πιθανότητας επί του συνόλου των στιγμιοτύπων του εν λόγω προβλήματος (δηλαδή, όλων των δυνατών εισόδων μεγέθους n), τότε είναι δυνατή η ανάλυση αναμενόμενης περίπτωσης, η οποία μας δίνει την αναμενόμενη χρονική πολυπλοκότητα (ή τον αναμενόμενο χρόνο εκτέλεσης) του αλγορίθμου, όταν του δοθεί μια νόμιμη είσοδος με μέγεθος n.

Παράδειγμα

- Ας υποθέσουμε ότι τα n στοιχεία του πίνακα στον αλγόριθμο InsertionSort έχουν επιλεγεί ομοιόμορφα με τυχαίο τρόπο από κάποιο αρχικό σύνολο στοιχείων (universe).
- Πόσες φορές θα εκτελεστεί το σώμα του while loop του αλγορίθμου για οποιαδήποτε τιμή του j;
  - Κατά μέσο όρο, τα μισά στοιχεία στον A[1..j-1] θα είναι μικρότερα του A[j] και τα άλλα μισά μεγαλύτερα.
  - Επομένως, το αναμενόμενο πλήθος στοιχείων που θα ελεγχθούν είναι j/2. Άρα, $s_j = j/2$.
- Ο αναμενόμενος χρόνος εκτέλεσης της InsertionSort είναι επομένως:
  $T(n) = 4n - 1 + 10 * \Sigma_{j=2..n} s_j = 4n - 1 + 5 * \Sigma_{j=2..n} j = 4n - 1 + 5 * (n+2)(n-1)/2$
  $= (6n + 5n^2 + 5n - 10 - 2)/2 = (5n^2 + 11n - 12)/2.$

$\Rightarrow$ **τετραγωνική συνάρτηση του n.**

Ανάλυση Χειρότερης περίπτωσης – Ανάλυση Αναμενόμενης Περίπτωσης

- Στο μάθημα αυτό θα εστιάσουμε στην ανάλυση χειρότερης περίπτωσης για τους ακόλουθους λόγους:
  - Ο χείριστος χρόνος εκτέλεσης ενός αλγορίθμου είναι ένα πάνω φράγμα στον χρόνο εκτέλεσης για οποιαδήποτε είσοδο μεγέθους n.
  - Σε πολλά προβλήματα, η χείριστη περίπτωση συμβαίνει συχνά (π.χ., αποτυχημένη αναζήτηση).
  - Ο αναμενόμενος χρόνος εκτέλεσης συχνά δεν είναι πολύ καλύτερος από τον χείριστο χρόνο εκτέλεσης.

- Από εδώ και στο εξής, οι όροι χρονική πολυπλοκότητα και χρόνος εκτέλεσης ενός αλγορίθμου θα αναφέρονται στη χρονική πολυπλοκότητα χειρότερης περίπτωσης και στο χείριστο χρόνο εκτέλεσης του αλγορίθμου.

Ασυμπτωτική Ανάλυση

- Η χρονική πολυπλοκότητα της InsertionSort είναι $5n^2+9n-11$.
- Οι σταθερές 5, 9, -11 δεν μας δίνουν χρήσιμη πληροφορία.
- Μας ενδιαφέρει κυρίως ο ρυθμός μεταβολής της συνάρτησης της χρονικής πολυπλοκότητας:
  - Από το άθροισμα $5n^2+9n-11$ μας ενδιαφέρει μόνο ο κυρίαρχος όρος $5n^2$, αφού οι άλλοι δύο όροι είναι μη σημαντικοί για μεγάλες τιμές του n.
  - Αγνοούμε επίσης το συντελεστή 5, αφού οι σταθεροί παράγοντες δεν είναι σημαντικοί για μεγάλες τιμές του n.
- Λέμε πως η χρονική πολυπλοκότητα της InsertionSort είναι τετραγωνικής τάξης.

Ασυμπτωτική Ανάλυση – Συμβολισμός O

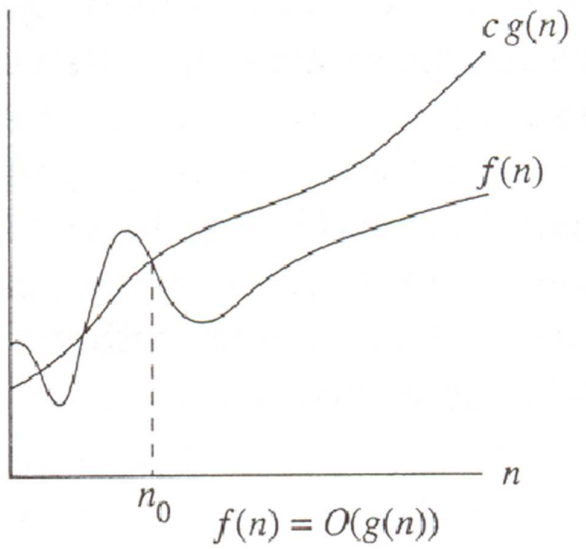
Ορισμός

Έστω f(n) και g(n) δύο συναρτήσεις. Η f(n) ανήκει στο $O(g(n))$, $f(n) \in O(g(n))$ ή $f(n) = O(g(n))$, αν υπάρχουν σταθερές $c \in R^+$ (c πραγματικός, $c > 0$) και ακέραιος $n_0 \geq 0$, έτσι ώστε για κάθε $n \geq n_0$ να ισχύει:

$$0 \leq f(n) \leq cg(n)$$

- Ο συμβολισμός O υποδηλώνει ότι μια συνάρτηση (f(n)) είναι ασυμπτωτικά άνω φραγμένη.
- Το $O(g(n))$ είναι ένα σύνολο συναρτήσεων: το σύνολο των συναρτήσεων για τις οποίες η g(n) αποτελεί ασυμπτωτικό άνω φράγμα.
  - Ο συμβολισμός $f(n) = O(g(n))$ (αν και συνηθίζεται γιατί είναι βολικός σε κάποιες περιπτώσεις) δεν είναι μαθηματικά σωστός. Δηλώνει απλά ότι η f(n) είναι μέλος του συνόλου $O(g(n))$.

Ασυμπτωτική Ανάλυση – Συμβολισμός O



Σχήμα 3.1(α): Cormen, Leiserson, Rivest & Stein, *Εισαγωγή στους αλγόριθμους*, Τόμος Ι, Πανεπιστημιακές Εκδόσεις Κρήτης

Ασυμπτωτική Ανάλυση – Συμβολισμός O

Παράδειγμα 1

Έστω $f(n) = an^2 + bn$, όπου a,b θετικές σταθερές.

Ισχύει ότι $f(n) = O(n^2)$;

Απάντηση

Αναζητούμε σταθερές $c \in R^+$ & $n_0 \geq 0$, τ.ω.:

$an^2 + bn \leq cn^2$, για κάθε $n \geq n_0$

$0 \leq (c-a)n^2 - bn \Leftrightarrow 0 \leq n\,[(c-a)n - b] \Leftrightarrow (c-a)n - b \geq 0$

(αφού $n \geq n_0 \geq 0$)

Αν επιλέξουμε $c = a+1$ και οποιοδήποτε $n_0 \geq b$, η ανισότητα $(c-a)n - b \geq 0$ ισχύει.

Ασυμπτωτική Ανάλυση – Συμβολισμός O

Παράδειγμα 2
Έστω $f(n) = 20n^3 + 10n\log n + 5$. Ισχύει ότι $f(n) = O(n^3)$;

Απάντηση
$20n^3 + 10n\log n + 5 \leq 35n^3$, για $n \geq 1$. Αν αποδείξουμε πως υπάρχουν σταθερές $c \in \mathbb{R}^+$ & $n_0 \geq 0$ τ. ω. $35n^3 \leq cn^3$, $\forall\ n \geq n_0$, θα ισχύει και πως $f(n) \leq cn^3$, $\forall\ n \geq n_0$. Άρα, αν επιλέξουμε $c = 35$ και οποιοδήποτε $n_0 \geq 1$, ο ισχυρισμός $f(n) = O(n^3)$ αποδεικνύεται.

Παράδειγμα 3
Έστω $f(n) = 3\log n + \log\log n$. Ισχύει ότι $f(n) = O(\log n)$;

Απάντηση
$3\log n + \log\log n \leq 4\log n$, αν $n \geq 1$. Άρα, αν επιλέξουμε $c = 4$ και οποιοδήποτε $n_0 \geq 1$, ο ισχυρισμός $f(n) = O(\log n)$ αποδεικνύεται.

Ασυμπτωτική Ανάλυση – Συμβολισμός O

- Λέγοντας ότι ο χρόνος εκτέλεσης ενός αλγορίθμου είναι $O(n^2)$ εννοούμε ότι υπάρχει μια συνάρτηση $f(n)$ η οποία ανήκει στο $O(n^2)$, τέτοια ώστε για οποιαδήποτε τιμή του n (εκτός ίσως από κάποιες μικρές τιμές), ανεξάρτητα από τη μορφή της κάθε συγκεκριμένης εισόδου μεγέθους n, ο χρόνος εκτέλεσης για αυτή την είσοδο φράσσεται εκ των άνω από την τιμή $f(n)$.
- Δεν είναι συνηθισμένο να συμπεριλαμβάνονται σταθεροί παράγοντες και χαμηλότερης τάξης όροι στο συμβολισμό O.
  Ο ισχυρισμός $2n^2 = O(4n^2 + n\log n)$ είναι σωστός αλλά δεν θεωρείται «κομψός».
- Ο ισχυρισμός $f(n) \leq O(g(n))$ δεν είναι επίσης κομψός αφού το O εμπεριέχει την έννοια του «μικρότερου ή ίσου»: υποδηλώνει ένα μη αυστηρό ασυμπτωτικό άνω φράγμα της f.
- Επιτρέπεται η χρήση του συμβολισμού O σε αριθμητικές εκφράσεις:
  - Η $f(n)$ είναι στο $h(n) + O(g(n)) \Rightarrow$ υπάρχουν σταθερές c και n_0 τ.ω., για κάθε $n \geq n_0$, $f(n) \leq h(n) + cg(n)$.

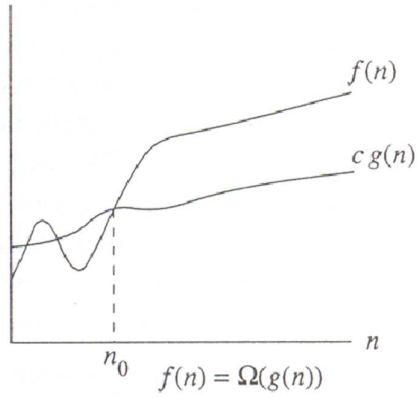
Ασυμπτωτική Ανάλυση - Συμβολισμός Ω

Ορισμός

Έστω f(n) και g(n) δύο συναρτήσεις. Η f(n) ανήκει στο Ω(g(n)), $f(n) \in \Omega(g(n))$ ή $f(n) = \Omega(g(n))$, αν υπάρχουν σταθερές $c \in R^+$ (c πραγματικός, $c > 0$) και ακέραιος $n_0 \geq 0$, έτσι ώστε, για κάθε $n \geq n_0$, να ισχύει:

$$0 \leq cg(n) \leq f(n)$$

- Ο συμβολισμός Ω δηλώνει ότι μια συνάρτηση (f(n)) είναι ασυμπτωτικά κάτω φραγμένη.



Σχήμα 3.1(β): Cormen, Leiserson, Rivest & Stein, *Εισαγωγή στους αλγόριθμους*, Τόμος Ι, Πανεπιστημιακές Εκδόσεις Κρήτης

Ασυμπτωτική Ανάλυση – Συμβολισμός Ω

Παράδειγμα 1.

Έστω $f(n) = an^2 + bn$, όπου a,b θετικές σταθερές.
Θα αποδείξουμε ότι $f(n) = \Omega(n)$.

Απόδειξη

Αναζητούμε σταθερές $c \in R^+$ & $n_0 \geq 0$ τ.ω., για κάθε $n \geq n_0$ να ισχύει:
$an^2 + bn \geq cn \Leftrightarrow an^2 + (b-c)n \geq 0 \Leftrightarrow an + b-c \geq 0$.
Αν επιλέξουμε $c = b$ ισχύει ότι $an \geq 0$, για κάθε $n \geq 0$ (αφού $a \in R^+$).
Επομένως, για $c = b$ & $n_0 = 0$, ο ισχυρισμός αποδεικνύεται.

Παράδειγμα 2

Έστω $f(n) = 20n^3 + 10n\log n + 5$. Ισχύει ότι $f(n) = \Omega(n^3)$;

Απάντηση

$20n^3 + 10n\log n + 5 \geq 20n^3$, για $n \geq 1$. Άρα, αν επιλέξουμε $c = 20$ και οποιοδήποτε $n_0 \geq 1$, ο ισχυρισμός $f(n) = \Omega(n^3)$ αποδεικνύεται.

Ασυμπτωτική Ανάλυση – Συμβολισμός Ω

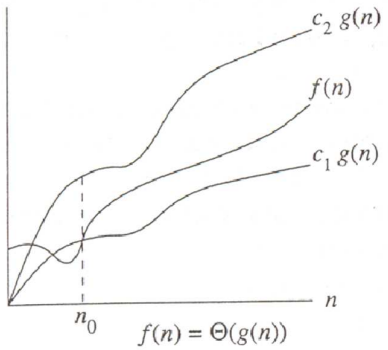
Παράδειγμα 3
Έστω $f(n) = n^3 - 10n\log n - 5$. Ισχύει ότι $f(n) = \Omega(n^3)$;

Απάντηση
$n^3 - 10n\log n - 5 \geq n^3 - 10n^2 - 5n^2$, για κάθε $n \geq 1$.
Αναζητούμε σταθερές $c \in R^+$ & $n_0 \geq 0$, τ.ω. , για κάθε $n \geq n_0$ να ισχύει:
$n^3 - 10n\log n - 5 \geq n^3 - 10n^2 - 5n^2 \geq cn^3 \Leftrightarrow$
$n^3 - 15n^2 \geq cn^3 \Leftrightarrow (1-c)n^3 - 15n^2 \geq 0 \Leftrightarrow (1-c)n - 15 \geq 0$.

Επιλέγοντας π.χ., $c = 1/2$ και $n_0 = 30$, η τελευταία ανισότητα ισχύει για κάθε $n \geq n_0$. Άρα, ο ισχυρισμός $f(n) = \Omega(n^3)$ ισχύει.

Ασυμπτωτική Ανάλυση – Συμβολισμός Θ

Ορισμός
Έστω $f(n)$ και $g(n)$ δύο συναρτήσεις. Η $f(n)$ ανήκει στο $\Theta(g(n))$, $f(n) \in \Theta(g(n))$ ή $f(n) = \Theta(g(n))$, αν ισχύει ότι $f(n) = O(g(n))$ και $f(n) = \Omega(g(n))$.



Σχήμα 3.1(γ): Cormen, Leiserson, Rivest & Stein, *Εισαγωγή στους αλγόριθμους*, Τόμος Ι, Πανεπιστημιακές Εκδόσεις Κρήτης

Αν $f(n) \in \Theta(g(n))$, ισχύει ότι υπάρχουν σταθερές $c_1, c_2 \in R+$ και ακέραιος $n_0 \geq 0$, έτσι ώστε για κάθε $n \geq n_0$ να ισχύει:

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

@ Ο συμβολισμός Θ υποδηλώνει ότι μια συνάρτηση ($f(n)$) είναι ασυμπτωτικά φραγμένη με αυστηρό τρόπο (η $g(n)$ είναι ένα ασυμπτωτικά αυστηρό φράγμα για την $f(n)$).

Ασυμπτωτική Ανάλυση – Ιδιότητες

Μεταβατική Ιδιότητα

- $f(n) = O(g(n))$ και $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$ (το ίδιο ισχύει και για τους συμβολισμούς Ω, Θ).

Ανακλαστική Ιδιότητα

- $f(n) = O(f(n))$, $f(n) = \Omega(f(n))$ και $f(n) = \Theta(f(n))$

Συμμετρική Ιδιότητα

- $f(n) = \Theta(g(n))$ αν και μόνο αν $g(n) = \Theta(f(n))$

Αναστροφική Ιδιότητα

- $f(n) = O(g(n))$ αν και μόνο αν $g(n) = \Omega(f(n))$

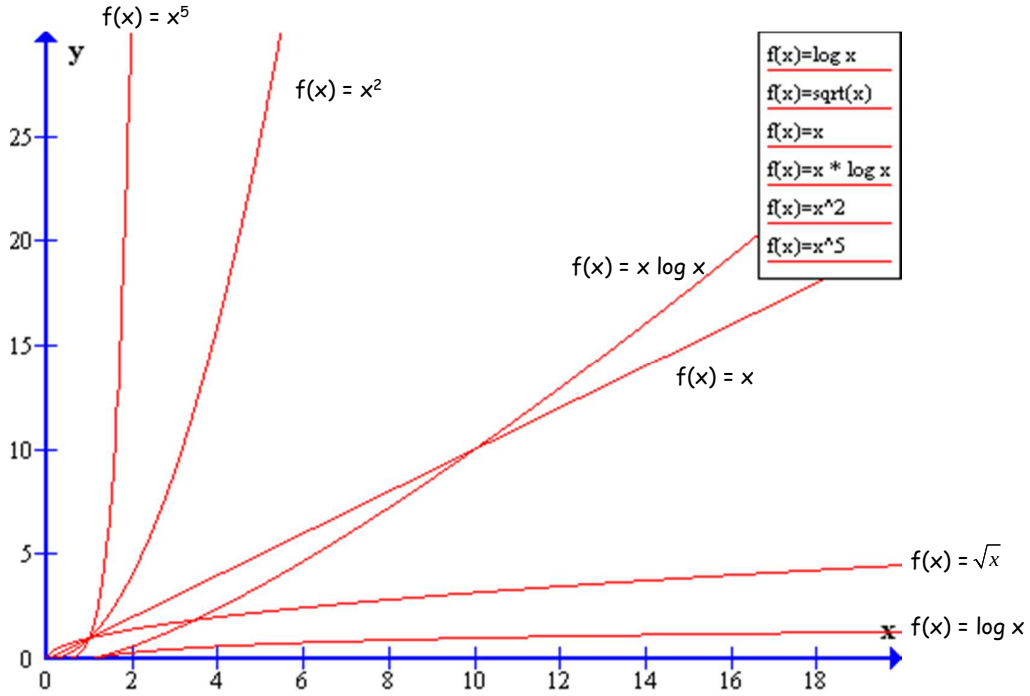
Άλλες Ιδιότητες

- Αν $f(n) = O(g(n))$, τότε $(cf)(n) = O(g(n))$, για οποιαδήποτε σταθερά $c \in \mathbb{R}^+$.
- Αν $f(n) = O(g(n))$ και $h(n) = O(g(n))$, τότε $(c_1f+c_2h)(n) = O(g(n))$ για οποιεσδήποτε σταθερές $c_1, c_2 \in \mathbb{R}^+$.

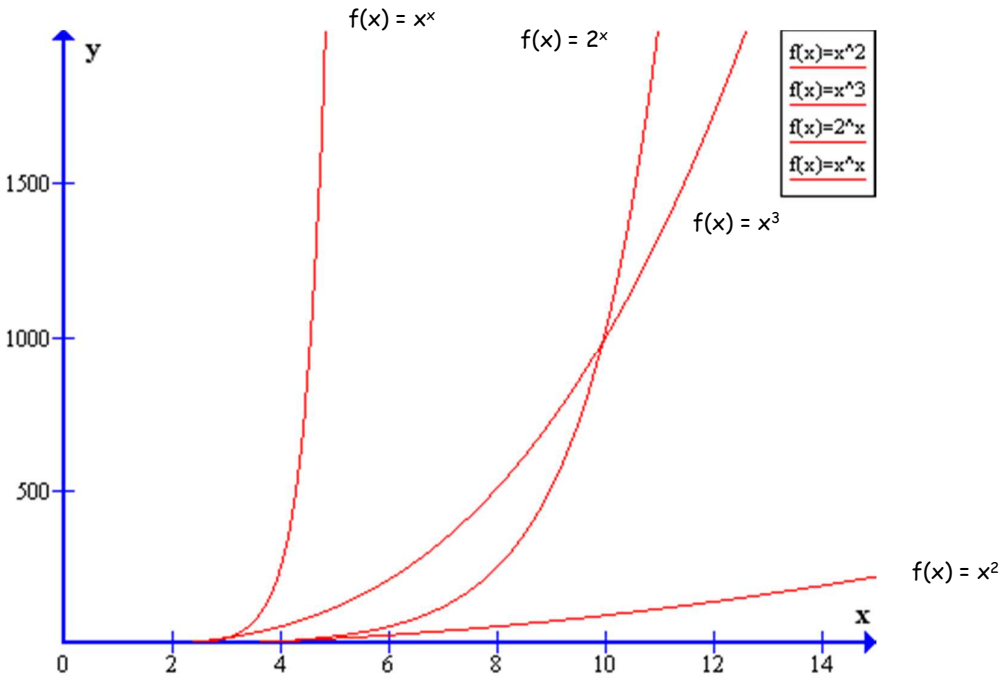
Συνήθεις τάξεις πολυπλοκότητας

Χαρακτηριστικές κλάσεις συναρτήσεων και οι μεταξύ τους σχέσεις
$O(1)$ $\subset$
$O(\log n) \subset$
$O(\log^m n)$, για κάθε $m > 1 \subset$
$O(n^{1/2}) \subset$
$O(n) \subset$
$O(n \log n) \subset$
$O(n^2) \subset$
$O(n^m)$, για κάθε $m > 2 \subset$
$O(2^n) \subset$
$O(n!) \subset$
$O(n^n)$

Γραφικές Παραστάσεις Συναρτήσεων



Γραφικές Παραστάσεις Συναρτήσεων



Ασυμπτωτική Ανάλυση – Γιατί είναι σημαντική;

Size of input ↓	log n	$\sqrt{n}$	n	n log n	n^2	n^3	2^n
2	1	1.4	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2.8	8	24	64	512	256
16	4	4	16	64	256	4096	65.536
32	5	5.7	32	160	1024	32.768	4.294.967.296
64	6	8	64	384	4096	262.144	$1,84 * 10^{19}$
128	7	11	128	896	16384	2.097.152	$3,40 * 10^{38}$
256	8	16	256	2048	65536	16.777.216	$1,15 * 10^{77}$
512	9	23	512	4608	262144	134.217.728	$1,34 * 10^{154}$
1024	10	32	1024	10240	1048576	1.073.741.824	$1,79 * 10^{308}$

Αυξητικός Χαρακτήρας Συναρτήσεων

Ασυμπτωτική Ανάλυση – Γιατί είναι σημαντική;

Size of input ↓	n	n^2	n^5	2^n	3^n
10	0,01 msec	0,1 msec	0,1 sec	1 msec	59 msec
20	0,02 msec	0,4 msec	3,2 sec	1 sec	58 min
40	0,04 msec	1,6 msec	1,7 min	12,7 days	3855 centuries
50	0,05 msec	2,5 msec	5,2 min	35,7 years	$2*10^8$ centuries
60	0,06 msec	3,6 msec	13 min	366 centuries	$1,3 * 10^{13}$ centuries

Ενδεικτικές Ανάγκες σε Χρόνο Υποτιθέμενων Αλγορίθμων

Figure 1.2: Garey, M. R., Johnson, D. S., Victor Klee. *Computers and Intractability: A Guide to the Theory of NP-Completeness.*

Ασυμπτωτική Ανάλυση – Γιατί είναι σημαντική;

Μέγεθος του μεγαλύτερου στιγμιότυπου ενός προβλήματος που μπορεί να επιλυθεί σε 1 ώρα

	n	n^2	n^5	2^n	3^n
Με έναν τρέχον υπολογιστή	N_1	N_2	N_3	N_4	N_5
Με έναν υπολογιστή 100 φορές πιο γρήγορο	$100N_1$	$10N_2$	$2,5N_3$	$N_4+6,64$	$N_5+4,19$
Με έναν υπολογιστή 1000 φορές πιο γρήγορο	$1000N_1$	$31,6N_2$	$3,98N_3$	$N_4+9,97$	$N_5+6,29$

Επίδραση βελτιωμένης τεχνολογίας σε διάφορους πολυωνυμικούς και εκθετικούς αλγορίθμους

Figure 1.3: Garey, M. R., Johnson, D. S., Victor Klee. *Computers and Intractability: A Guide to the Theory of NP-Completeness.*

Χρήσιμο Μαθηματικό Υπόβαθρο
Εκθέτες & Λογάριθμοι

Για οποιουσδήποτε πραγματικούς αριθμούς $x > 0$, m και n, ισχύουν τα εξής:

- $x^0 = 1$, $x^1 = x$, $x^{-1} = 1/x$,
- $(x^m)^n = x^{mn}$, $(x^m)^n = (x^n)^m$
- $x^m x^n = x^{m+n}$, $x^m/x^n = x^{m-n}$

Συμβολίζουμε με e = 2,71828… τη βάση των φυσικών (νεπέρειων) λογαρίθμων.

Για οποιουσδήποτε πραγματικούς αριθμούς $x,y,z > 0$, ισχύουν τα εξής:

$\log_z xy = \log_z x + \log_z y$,

$\log_z(x/y) = \log_z x - \log_z y$, $\log_z(1/x) = -\log_z x$

$\log_z x^y = y \log_z x$

$\log_y x = (\log_z x) / \log_z y$

$x = y^{\log_y x}, y^{\log_z x} = x^{\log_z y}$ $\log(\Pi_{k=1}^{n} x_k) = \sum_{k=1}^{n} \log x_k$

Χρήσιμο Μαθηματικό Υπόβαθρο – Κάτω και πάνω ακέραιο μέρος & Παραγοντικά

Για κάθε πραγματικό αριθμό x:

- το σύμβολο $\lfloor x \rfloor$ δηλώνει το μεγαλύτερο ακέραιο που είναι μικρότερος ή ίσος του x και το σύμβολο $\lceil x \rceil$ δηλώνει το μεγαλύτερο ακέραιο που είναι μεγαλύτερος ή ίσος του x
- $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$
- Για οποιουσδήποτε ακεραίους a,b > 0 ισχύει ότι:
  - $\lfloor \lfloor x/a \rfloor / b \rfloor = \lfloor x/ab \rfloor$
  - $\lceil \lceil x/a \rceil / b \rceil = \lceil x/ab \rceil$

Για κάθε ακέραιο n, ισχύει ότι $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$

n! = 1*2* … * n, ή $n! = \begin{cases} 1 & \text{αν } n=0 \\ n*(n-1)! & \text{διαφορετικά} \end{cases}$

$n! \leq n^n$

$\log(n!) = \Theta(n \log n)$

Χρήσιμο Μαθηματικό Υπόβαθρο – Αθροίσματα

Αριθμητική Πρόοδος

$$\sum_{k=1}^{n} k = 1+2+\ldots+n = \frac{n(n+1)}{2}$$

$$a_1 + a_2 + \ldots + a_n = \frac{n}{2}(a_1 + a_n)$$

Γεωμετρική Πρόοδος

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1}-1}{x-1}$$

Αν $|x| < 1$, τότε $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

Τηλεσκοπική (Telescoping) Σειρά

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0$$

Παράδειγμα

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} = \sum_{k=1}^{n-1} \left(\frac{1}{k} - \frac{1}{k+1}\right) = 1 - \frac{1}{n}$$

Ανάλυση Αναδρομικών Αλγορίθμων

Είναι η Power ο πιο αποδοτικός αλγόριθμος για το πρόβλημα ύψωσης αριθμού σε δύναμη;

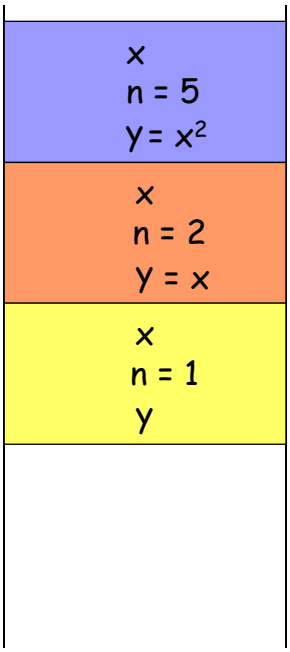
```
Algorithm RPower(x, n) {
    double y;

    if (n == 1) return x;
    y = RPower(x, ⌊n/2⌋);
    if n is even
        return y*y;
    else
        return x*y*y;
}
```


Ιχνηλάτιση (trace) της RPower()

```
RPower(x, 5)
    if (5 == 1) //evaluates to false
    y = RPower(x, 2);
        if (2 == 1) // evaluates to false
        y = RPower(x,1);
            if (1 == 1) return x;
        if (2 is even) return y*y; // x^2
    if (5 is even) //evaluates to FALSE
    return x*y*y; // x^5
```

RPower(x,5) RPower(x,2) RPower(x,1)

Memory




```
Algorithm RPower(x, n) {
    double y;

    if (n == 1) return x;
    y = RPower(x, ⌊n/2⌋);
    if n is even
        return y*y;
    else
        return x*y*y;
}
```

Ανάλυση Αναδρομικών Αλγορίθμων: RPower()

```
Algorithm RPower(x, n) {
    double y;

    if (n == 1) return x;     ----> 2
    y = RPower(x, ⌊n/2⌋);     ----> 1 + T(⌊n/2⌋)
    if n is even              ----> 2
        return y*y;           ----> 2
    else
        return x*y*y;         ----> 3
}
```

Έστω οποιοσδήποτε ακέραιος n και έστω ότι συμβολίζουμε με T(n) τη χρονική πολυπλοκότητα του RPower() όταν αυτός καλείται με τη δεύτερη παράμετρο ίση με το n.

Τότε:

$T(1) = 2$ και $T(n) = T(\lfloor n/2 \rfloor) + 7$ (1)

Ο όρος **αναδρομική σχέση** δηλώνει μια εξίσωση ή ανίσωση η οποία ορίζει μια συνάρτηση μέσω της τιμής της για κάποιο μικρότερο όρισμα.

Πως μπορούμε να λύσουμε την αναδρομική σχέση (1) (δηλαδή να βρούμε ασυμπτωτικά φράγματα τύπου Θ ή Ο για το T(n);

Ανάλυση Αναδρομικών Αλγορίθμων

Μέθοδοι επίλυσης αναδρομικών εξισώσεων

- Μέθοδος εικασίας
  - Διατυπώνουμε μια εικασία για τη λύση.
  - Με μαθηματική επαγωγή προσδιορίζουμε τις σταθερές και αποδεικνύουμε ότι η λύση ισχύει.
- Μέθοδος επαναληπτικής αντικατάστασης
  - Εφαρμόζουμε επαναληπτικά τον ορισμό της αναδρομικής σχέσης μέχρι το T(n) να εκφραστεί ως ένα άθροισμα όρων που εξαρτώνται μόνο από το n και τις αρχικές συνθήκες.
- Γενική μέθοδος

Επίλυση Αναδρομικών Σχέσεων – Μέθοδος Εικασίας

Παράδειγμα 1: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

$$T(n) = \begin{cases} 2 & \text{αν } n = 1 \\ T(\lfloor n/2 \rfloor) + 7 & \text{διαφορετικά} \end{cases}$$

- Εικάζουμε ότι η λύση είναι $O(\log n)$. Αποδεικνύουμε επαγωγικά ότι $T(n) \leq c\log n$, για κατάλληλη τιμή της σταθεράς $c > 0$. (1)

Βάση Επαγωγής

- Ελέγχω αν η (1) ισχύει για την τιμή $n = 1$. Από αναδρομική σχέση, $T(1) = 2 > c \log 1 = 0 \Rightarrow$ Η (1) δεν ισχύει για την τιμή $n = 1$.
- Ο ασυμπτωτικός συμβολισμός απαιτεί να αποδείξουμε $T(n) \leq c\log n$, για $n \geq n_0$, όπου n_0 οποιαδήποτε σταθερά.
- Ελέγχω αν η (1) ισχύει για την τιμή $n = 2$. Από αναδρομική σχέση, $T(2) = T(1) + 7 = 2+7 = 9$. Εξετάζω αν υπάρχει $c > 0$, τ.ω. $T(2) = 9 \leq c \log 2 \Leftrightarrow c \geq 9$. (2)
- Ελέγχω αν η (1) ισχύει για την τιμή $n = 3$. Από αναδρομική σχέση, $T(3) = T(1) + 7 = 2+7 = 9 \leq c \log 3$. Ισχύει αν $c \geq 9$. (3)

Επίλυση Αναδρομικών Σχέσεων – Μέθοδος Εικασίας

Επαγωγική Υπόθεση

Θεωρούμε οποιαδήποτε ακέραια τιμή $n > 2$. Υποθέτουμε ότι η (1) ισχύει για κάθε ακέραια τιμή m, τέτοια ώστε: $2 \leq m < n$, δηλαδή υποθέτουμε ότι ισχύει η ακόλουθη ανίσωση: $T(m) \leq c\log m$, $\forall$ m, $2 \leq m < n$, όπου $c \geq 9$ είναι μια πραγματική σταθερά.

Επαγωγικό Βήμα

Αποδεικνύουμε ότι ο ισχυρισμός ισχύει για την τιμή n, δηλαδή αποδεικνύουμε ότι $T(n) \leq c\log n$.

Αν $n = 3$, ο ισχυρισμός προκύπτει από (3).

Υποθέτουμε ότι $n > 3$. Από αναδρομική σχέση:
$T(n) = T(\lfloor n/2 \rfloor) + 7$. (4)
Εφόσον $n > 3 \Rightarrow \lfloor n/2 \rfloor \geq 2$. Από επαγωγική υπόθεση (για όπου $m = \lfloor n/2 \rfloor$) προκύπτει ότι
$T(\lfloor n/2 \rfloor) \leq c\log(\lfloor n/2 \rfloor) \leq c(\log n - \log 2) = c(\log n - 1)$ (5)

Από (4), (5) $\Rightarrow T(n) \leq c \log n - c + 7 \leq c \log n$, αν $c \geq 9$.

- Αν επιλέξω επομένως τη σταθερά $c = 9$, ο ισχυρισμός ισχύει.

Επίλυση Αναδρομικών Σχέσεων – Μέθοδος Εικασίας

Παράδειγμα 2: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

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

Εικάζουμε ότι η λύση είναι $O(n\log n)$. Αποδεικνύουμε επαγωγικά ότι $T(n) \leq cn\log n$, για κατάλληλη τιμή της σταθεράς $c > 0$.

- Αντικαθιστώντας στην παραπάνω εξίσωση και εφαρμόζοντας ισχυρή επαγωγή:

$T(n) \leq 2(c \lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \leq cn\log(n/2)+n = cn\log n - cn\log 2 + n$
$= cn\log n - cn + n \leq cn\log n$, αν $c \geq 1$.

- Ελέγχουμε τις αρχικές συνθήκες:

$n=1$: $T(n) \leq cn\log n \Rightarrow T(1) \leq cn\log 1 = 0$, το οποίο αντιτίθεται στη συνθήκη $T(1) = 1$.

- Ο ασυμπτωτικός συμβολισμός απαιτεί να αποδείξουμε $T(n) \leq cn\log n$, για $n \geq n_0$, όπου n_0 οποιαδήποτε σταθερά. Θέτοντας $n_0 = 2$, αποδεικνύεται ότι η βάση της επαγωγής ισχύει για κάθε $n \geq n_0$ αν $c \geq 2$:
  - Από αναδρομική σχέση: $T(2) = 4$
  - Ισχυρισμός: $T(2) \leq c2\log 2 \leq 2*2 = 4$. Ισχύει!

Επίλυση Αναδρομικών Σχέσεων – Μέθοδος Εικασίας

Παράδειγμα 3: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

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

Εικάζουμε ότι η λύση είναι $O(n)$. Προσπαθούμε να αποδείξουμε ότι $T(n) \leq cn$, για κάποια σταθερά $c > 0$. (1)

Ωστόσο, αντικαθιστώντας στην παραπάνω εξίσωση και εφαρμόζοντας ισχυρή επαγωγή:

$T(n) \leq c \lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \leq cn+1 > cn$, $\forall c > 0$. Η (1) δεν ισχύει!

Δοκιμάζουμε να ισχυροποιήσουμε την επαγωγική υπόθεση. Αποδεικνύουμε ότι $T(n) \leq cn - b$, όπου $b \geq 0$ κάποια σταθερά.

Άσκηση για το σπίτι!

Επίλυση Αναδρομικών Σχέσεων – Μέθοδος Επαναληπτικής Αντικατάστασης

Παράδειγμα 1: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

$$T(n) = \begin{cases} 2 & \text{αν n = 1} \\ T(\lfloor n/2 \rfloor) + 7 & \text{διαφορετικά} \end{cases}$$

T(n) = T(⌊n/2⌋) + 7
= (T(⌊n/4⌋) + 7) + 7 = T(⌊n/4⌋) + 2* 7
= (T(⌊n/8⌋) + 7) + 2 * 7 = T(⌊n/8⌋) + 3* 7
= …
= T(⌊n/2^i⌋) + i*7

Ποτέ σταματά η επαναληπτική αντικατάσταση;

Όταν ⌊n/2^i⌋ = 1 ⇒ i ≥ logn.

Τότε: T(n) ≤ T(1) + 7logn = 2 + 7 logn = *O*(logn).

Επίλυση Αναδρομικών Σχέσεων – Μέθοδος Επαναληπτικής Αντικατάστασης

Παράδειγμα 2: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

$$T(n) = \begin{cases} 1 & \text{αν n = 1} \\ 3T(\lfloor n/4 \rfloor) + n & \text{διαφορετικά} \end{cases}$$

T(n) = 3 * T(⌊n/4⌋) + n
= 3 * (3*T(⌊n/4^2⌋) + ⌊n/4⌋) + n = 3^2*T(⌊n/4^2⌋) + 3* ⌊n/4⌋ + n
= 3^2*(3*T(⌊n/4^3⌋)+ ⌊n/4^2⌋)+ 3* ⌊n/4⌋ + n =
3^3*T(⌊n/4^3⌋) + 3^2*⌊n/4^2⌋ +3* ⌊n/4⌋ + n
≤ 3^3*T(⌊n/4^3⌋) + n((3/4)^2+ (3/4)^1 + $(\frac{3}{4})^0$)
= …
≤ 3^i*T(⌊n/4^i⌋) + n*((3/4)$^{i-1}$ + … + (3/4) + 1)
= 3^i*T(⌊n/4^i⌋) + n* (1-(3/4)^i/1-3/4)
= 3^i*T(⌊n/4^i⌋) + 4n* (1 – (3/4)^i)
≤ 3^i*T(⌊n/4^i⌋) + 4n

Ποτέ σταματά η επαναληπτική αντικατάσταση;

Όταν ⌊n/4^i⌋ = 1 ⇒ i ≥ $\log_4 n$.

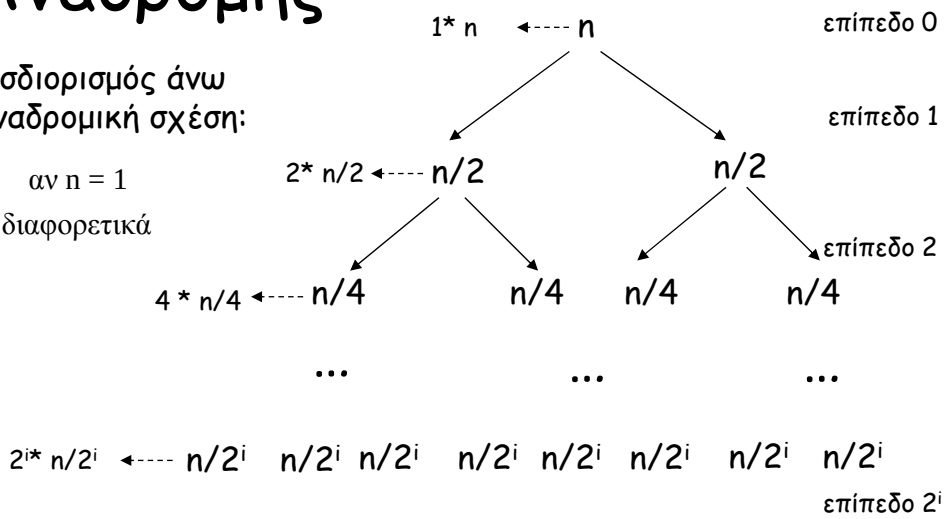
Τότε: T(n) ≤ 3^i*T(1) + 4n = $3^{\log_4 n}$ + 4n = $n^{\log_4 3}$ + 4n ≤ 4n + n = *O*(n)

Δένδρο Αναδρομής

Παράδειγμα 1: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

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

Λύση: $O(n \log n)$



Οι αναδρομικές κλήσεις σταματούν όταν $n/2^i = 1 \Rightarrow i = \log n$

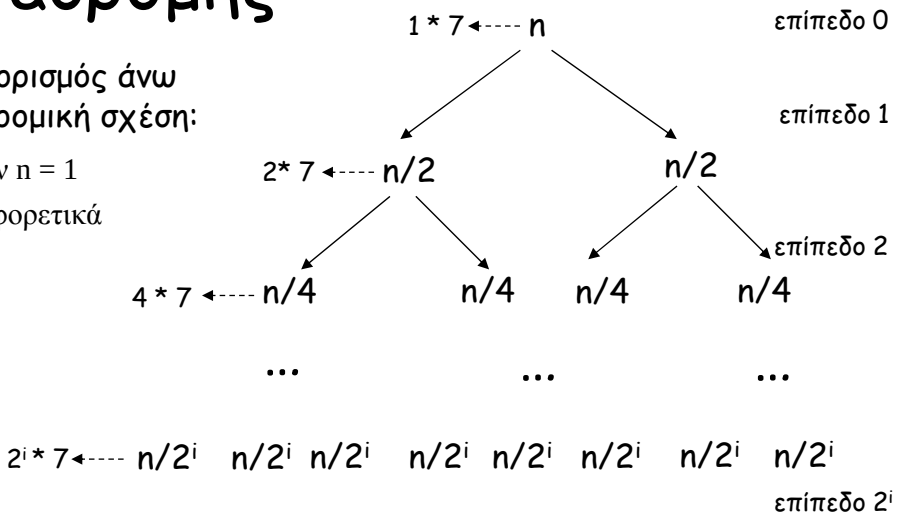
- Το δένδρο έχει (log n +1) επίπεδα.
- Το k-οστό επίπεδο αντιστοιχεί σε 2^k αναδρομικές κλήσεις που κάθε μια απαιτεί την εκτέλεση $n/2^k$ στοιχειωδών εντολών
- Συνολικό κόστος αναδρομικών κλήσεων επιπέδου = $\Theta(n)$.
- Συνολικό κόστος όλων των αναδρομικών κλήσεων = $\Theta(n \log n)$

Δένδρο Αναδρομής

Παράδειγμα 2: Προσδιορισμός άνω φράγματος για την αναδρομική σχέση:

$$T(n) = \begin{cases} 2 & \text{αν n = 1} \\ 2T(\lfloor n/2 \rfloor) + 7 & \text{διαφορετικά} \end{cases}$$

Λύση: $O(n \log n)$



Οι αναδρομικές κλήσεις σταματούν όταν $n/2^i = 1 \Rightarrow i = \log n$

- Το δένδρο έχει (log n +1) επίπεδα.
- Το k-οστό επίπεδο αντιστοιχεί σε 2^k αναδρομικές κλήσεις που κάθε μια απαιτεί την εκτέλεση 7 στοιχειωδών εντολών
- Συνολικό κόστος όλων των αναδρομικών κλήσεων = $7 * (1 + 2 + 4 + \ldots + 2^i)$
  $= 7 * (2^{i+1}-1)/(2-1) = 7 * (2*2^{\log n} - 1) \leq 14n = \Theta(n)$.

Πειραματική Ανάλυση

- Η χρήση των O, Ω, Θ μπορεί να οδηγήσει σε λάθος συμπεράσματα όταν τα O, Ω, Θ υποκρύπτουν σταθερές που είναι μεγάλες.
  - **Παράδειγμα:** Η συνάρτηση $10^{30}n = \Theta(n)$, αλλά ένας αλγόριθμος με χρονική πολυπλοκότητα $10n\log n$ θα ήταν προτιμότερος αφού σε όλες τις λογικές εισόδους ο 2ος αλγόριθμος θα συμπεριφερόταν πολύ καλύτερα από τον πρώτο.

- Οι αλγόριθμοι που έχουν πολυωνυμική χρονική πολυπλοκότητα θεωρούνται αποτελεσματικοί, ενώ αυτοί που απαιτούν εκθετικό χρόνο εκτέλεσης είναι μη-αποτελεσματικοί.

- Ωστόσο, ένας αλγόριθμος που απαιτεί χρόνο εκτέλεσης $\Theta(n^{50})$ δεν θεωρείται αποτελεσματικός!

Πειραματική Ανάλυση

Η ασυμπτωτική ανάλυση:

- δεν παρέχει πληροφορίες για τους σταθερούς παράγοντες που κρύβονται κάτω από τους συμβολισμούς O, Ω και Θ
- δεν καθορίζει διαχωριστικές γραμμές μεταξύ ασυμπτωτικά αργών αλγορίθμων με μικρούς σταθερούς παράγοντες και ασυμπτωτικά πιο γρήγορων αλγορίθμων με μεγάλους σταθερούς παράγοντες
- εστιάζει κύρια σε εισόδους χειρότερης περίπτωσης, που μπορεί να μην είναι οι πιο αντιπροσωπευτικές για συγκεκριμένα προβλήματα
- για πολύ πολύπλοκους αλγορίθμους μπορεί να μην μπορεί να επιτευχθεί.
- ✘ Για όλους αυτούς τους λόγους, είναι χρήσιμο να μπορούμε να μελετάμε αλγορίθμους και με πειραματικό τρόπο.

Πειραματική Ανάλυση - Επιλέγοντας μετρικά που θα μελετηθούν

- Εκτίμηση του ασυμπτωτικού χρόνου εκτέλεσης ενός αλγόριθμου στη μέση περίπτωση.
- Σύγκριση δύο ή περισσότερων αλγορίθμων προκειμένου να αποφασιστεί ποιος είναι πιο γρήγορος για κάποιο εύρος τιμών εισόδου $[n_0,n_1]$.
- Για αλγορίθμους που αποσκοπούν στην ελαχιστοποίηση ή στη μεγιστοποίηση κάποιας συνάρτησης, μελέτη της απόστασης της εξόδου του αλγορίθμου από τη βέλτιστη έξοδο.

Πειραματική Ανάλυση – Τι είδους μετρήσεις θα πραγματοποιηθούν

- Πραγματικός χρόνος εκτελέσης αλγορίθμου (χρήση gettimeofday() ή άλλων συναρτήσεων που μας παρέχει το σύστημα)

  Υπάρχουν πολλοί παράγοντες που μπορούν να επηρεάσουν τις μετρήσεις, όπως:
  - Υπάρχουν άλλα προγράμματα που εκτελούνται ταυτόχρονα;
  - Χρησιμοποιεί ο αλγόριθμος την κρυφή μνήμη του συστήματος αποτελεσματικά;
  - Είναι αρκετή η μνήμη του συστήματος για την αποτελεσματική εκτέλεση του αλγορίθμου;
- Μέτρηση πρωταρχικών λειτουργιών που επιτελούνται από τον αλγόριθμο
  - Αναφορές στη μνήμη, Συγκρίσεις, Αριθμητικές λειτουργίες

Παραγωγή Δεδομένων Εισόδου που θα χρησιμοποιηθούν για τα πειράματα

- Παραγωγή αρκετών δειγμάτων ώστε ο υπολογισμός μέσων τιμών να παρέχει στατιστικώς σημαντικά δεδομένα.
- Παραγωγή δειγμάτων διαφορετικών μεγεθών ώστε να είναι εφικτή η εξαγωγή συμπερασμάτων για διαφορετικά μεγέθη της εισόδου
- Παραγωγή δεδομένων αντιπροσωπευτικών εκείνων που ο αλγόριθμος θα συναντήσει στην πράξη.

Πειραματική Ανάλυση - Θέματα Υλοποίησης

- Ο χρόνος εκτέλεσης ενός αλγορίθμου εξαρτάται από το υλικό, τη γλώσσα προγραμματισμού και το μεταφραστή, αλλά και από το πόσο δεινός είναι ο προγραμματιστής.
- Κατά τη σύγκριση μέσω πειραματικής μελέτης δύο αλγορίθμων, ο ίδιος βαθμός βελτιστοποίησης του κώδικα πρέπει να εφαρμοστεί στις υλοποιήσεις και των δύο αλγορίθμων (και οι αλγόριθμοι καλό είναι να έχουν υλοποιηθεί από προγραμματιστές ανάλογης εμπειρίας).
- Τα χαρακτηριστικά (πλήθος ΚΜΕ και ταχύτητα αυτών, μέγεθος κύριας και κρυφών μνημών, ταχύτητα του καναλιού επικοινωνίας με τη μνήμη) του συστήματος στο οποίο πραγματοποιείται το πείραμα θα πρέπει να καταγράφονται λεπτομερώς.

Βιβλιογραφία

- Harry Lewis and Larry Denenberg, *Data Structures and Their Algorithms*, Harper Collins Publishers, Inc., New York, 1991
  - Chapter 1: Introduction
- Cormen, Leiserson and Rivest, *Introduction to Algorithms*, MIT Press, 1990. (Το βιβλίο κυκλοφορεί μεταφρασμένο από τις Πανεπιστημιακές Εκδόσεις Κρήτης σε δύο μέρη και μεγάλο μέρος της ύλης βασίζεται στο 1ο μέρος του.)
  - 1 Θεμελιώδεις έννοιες: Ο ρόλος των αλγορίθμων στις υπολογιστικές διαδικασίες,
  - 2 Προκαταρκτικές έννοιες και παρατηρήσεις
  - 3 Ρυθμός αύξησης συναρτήσεων
  - 4.1-4.2 Αναδρομικές σχέσεις
- Michael T. Goodrich and Roberto Tamassia, *Algorithm Design – Foundations, Analysis and Internet Examples*, John Wiley & Sons, Inc., 4th edition.
  - Μέρος του υλικού που αφορά την πειραματική ανάλυση αλγορίθμων προέρχεται από το βιβλίο αυτό.
- Garey, M. R., Johnson, D. S., Victor Klee. *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman and Co., 1979.