Αλγόριθμοι και Πολυπλοκότητα

Στην ενότητα αυτή θα μελετηθούν τα εξής θέματα:

Πρόβλημα, Στιγμιότυπο, Αλγόριθμος

Εργαλεία εκτίμησης πολυπλοκότητας: οι τάξεις $O(n)$, $\Omega(n)$, $\Theta(n)$

Ανάλυση Πολυπλοκότητας Αλγορίθμων

Βασικές Έννοιες

- Για πληροφορίες για τις πιο κάτω προκαταρτικές/ βασικές έννοιες (τις οποίες αναμένεται ότι ήδη γνωρίζετε) μπορείτε να συμβουλευτείτε τα κεφάλαια I και II του βιβλίου του μαθήματος, *Introduction to Algorithms, Cormen, Leiserson, Rivest,* ή/και τις σημειώσεις του ΕΠΛ231.

  - Σειρές, σχέσεις, συναρτήσεις, λογάριθμοι
  - Απόδειξη με τη μέθοδο της επαγωγής
  - Απόδειξη με τη μέθοδο της αντίφασης
  - Γράφοι, δένδρα και οι σχετικές έννοιες
  - Αναδρομικές Εξισώσεις.
  - Αλγόριθμοι Ταξινόμησης

Τι είναι ένας αλγόριθμος;

- *Δοθέντος ενός προβλήματος, αλγόριθμος είναι μια καλά-προσδιορισμένη διαδικασία η οποία παρέχει τις οδηγίες σύμφωνα με τις οποίες τα δεδομένα του προβλήματος μετασχηματίζονται και συνδυάζονται για να προκύψει η λύση του προβλήματος.*
- al-Khowarizmi, Πέρσης μαθηματικός, 9ος αιώνας μ.Χ.
- Αλγόριθμος του Ευκλείδη για υπολογισμό του μέγιστου κοινού διαιρέτη.
- Ένας αλγόριθμος πρέπει να αποτελείται από αντικειμενικές/ακριβείς οδηγίες.

  Και οι πιθανοτικοί αλγόριθμοι;;

  Η λήψη μιας πιθανοτικής απόφασης δεν είναι το ίδιο με τη λήψη μιας τυχαιοποιημένης απόφασης.

Πολυπλοκότητα Αλγορίθμων

- Έννοιες: Πρόβλημα
  Αλγόριθμος
  Στιγμιότυπο (instance).

- Ένας αλγόριθμος πρέπει να ικανοποιεί τις εξής προϋποθέσεις:

  1. πρέπει να εργάζεται ***σωστά*** για κάθε σύνολο δεδομένων εισόδου, δηλ. για κάθε στιγμιότυπο του πεδίου ορισμού του προβλήματος που λύνει.
  2. πρέπει να είναι ***αποδοτικός*.**

- Υπάρχει ένα *σύνολο* σωστών αλγορίθμων για κάθε πρόβλημα. Όλοι οι αλγόριθμοι έχουν θεωρητικό ενδιαφέρον. Και πρακτικό ενδιαφέρον όμως παρουσιάζουν αυτοί που είναι αποδοτικοί, δηλαδή αυτοί που ελαχιστοποιούν:
  - τον χρόνο που εκτελούνται,
  - τον χώρο που χρησιμοποιούν.

- Στόχος: Εξοικείωση με τεχνικές σχεδιασμού αλγορίθμων που ελαχιστοποιούν τον χρόνο εκτέλεσης και τον χώρο που χρησιμοποιούν. Έλεγχος αν και πότε ένας αλγόριθμος είναι άριστος/βέλτιστος (optimal), δηλαδή ο πιο αποδοτικός για το πρόβλημα για το οποίο σχεδιάστηκε.

Εμπειρική – Θεωρητική Ανάλυση Αλγορίθμων

- Ένας αλγόριθμος μπορεί να μελετηθεί ***εμπειρικά*** μετρώντας τον χρόνο και χώρο εκτέλεσής του σε συγκεκριμένο υπολογιστή.
- ***Θεωρητικά*** μπορούμε να υπολογίσουμε τον χρόνο και τον χώρο που απαιτεί ο αλγόριθμος σαν συνάρτηση του ***μεγέθους*** των εξεταζομένων στιγμιότυπων.
- Τυπικά, μέγεθος ενός στιγμιότυπου αντιστοιχεί στο μέγεθος της μνήμης που απαιτείται για αποθήκευση του στιγμιότυπου στον υπολογιστή. Για απλούστευση της ανάλυσης θα μετρούμε το μέγεθος ως τον ακέραιο (ή τους ακέραιους) που αντιστοιχούν στο πλήθος των ποσοτήτων του στιγμιότυπου.

  π.χ. Πρόβλημα: ταξινόμηση λίστας.

  Στιγμιότυπο: λίστα με n στοιχεία.

  Μέγεθος: n
- Πλεονεκτήματα της θεωρητικής προσέγγισης υπολογισμού αποδοτικότητας αλγορίθμων περιλαμβάνουν:
  1. δεν εξαρτάται από το υλικό του Η/Υ (μνήμη, cache, κλπ)
  2. δεν εξαρτάται από τη γλώσσα προγραμματισμού ή τον μεταφραστή
  3. δεν εξαρτάται από τις ικανότητες του προγραμματιστή.
  4. είναι ΓΕΝΙΚΗ

Μονάδα Έκφρασης της Αποδοτικότητας

Η αρχή της σταθερότητας

Δύο διαφορετικές υλοποιήσεις του ίδιου αλγορίθμου (σε διαφορετικές μηχανές ή σε διαφορετικές γλώσσες ή από διαφορετικούς προγραμματιστές) δεν διαφέρουν στον χρόνο εκτέλεσής τους περισσότερο από κάποιο σταθερό πολλαπλάσιο.
δηλ. αν E_1 είναι ο χρόνος εκτέλεσης της μίας υλοποίησης και E_2 της άλλης, τότε ισχύει $E_1 = c \cdot E_2$ για κάποια σταθερά c.

Θα λέμε ότι ένας αλγόριθμος *απαιτεί (ή εκτελείται σε) χρόνο* t(n) αν υπάρχει σταθερά c και εφαρμογή του αλγορίθμου τέτοια ώστε, για κάθε στιγμιότυπο μεγέθους n ο χρόνος εκτέλεσης του αλγορίθμου είναι μικρότερος ή ίσος του $c \cdot t(n)$.

Είδη αλγορίθμων

t(n)	Όνομα
n	γραμμικός
n^k	πολυωνυμικός
c^n	εκθετικός
$\log^k n$	λογαριθμικός

Μοντέλο Υπολογισμού

- Στο μεγαλύτερο μέρος του μαθήματος θα υποθέσουμε τη χρήση υπολογιστή που εκτελεί οδηγίες διαδοχικά, RAM (random-access machine).

  Αντιπαραθέστε το μοντέλο RAM με ένα μοντέλο το οποίο, χρησιμοποιώντας $n>1$ επεξεργαστές, έχει τη δυνατότητα να εκτελεί m, $n \geq m \geq 0$, οδηγίες παράλληλα.

- **Βασική πράξη** θεωρούμε ότι είναι οποιαδήποτε πράξη της οποίας ο χρόνος εκτέλεσης είναι φραγμένος από κάποια σταθερά (δηλ. $\leq c$, για κάποια σταθερά c). Συνεπώς ο χρόνος εκτέλεσης μιας βασικής πράξης είναι ανεξάρτητος από το μέγεθος ή τις παραμέτρους στιγμιότυπου οποιουδήποτε προβλήματος.

- Επειδή ορίζουμε τον χρόνο εκτέλεσης ενός αλγορίθμου με την έννοια του σταθερού πολλαπλασίου, για την ανάλυση θα χρειαστούμε μόνο τον αριθμό των βασικών πράξεων που εκτελούνται από ένα αλγόριθμο και όχι τον ακριβή χρόνο που απαιτούν η κάθε μια από αυτές.

- **Άρα για τον υπολογισμό του χρόνου εκτέλεσης ενός αλγόριθμου απλά μετρούμε τον αριθμό των βασικών πράξεων που εκτελεί.** Με τον όρο "βασικές πράξεις" εννοούμε μαθηματικές πράξεις (πρόσθεση, αφαίρεση…), σύγκριση, καταχώρηση μεταβλητής, επιστροφή αποτελέσματος.

Ανάλυση Χειρίστης Περίπτωσης

- Αν D_n είναι το σύνολο όλων των εισόδων (στιγμιότυπων) μεγέθους n, και t(I) ο αριθμός βασικών πράξεων που εκτελούνται από τον αλγόριθμο για κάθε $I \in D_n$ τότε ορίζουμε την ***πολυπλοκότητα Χειρίστης Περίπτωσης*** του αλγορίθμου ως

$$W(n) = \max \{ t(I) \mid I \in D_n \}$$

- Δηλαδή, ο ορισμός δίνει ένα άνω φράγμα της πολυπλοκότητας του αλγορίθμου.

Ανάλυση Μέσης Περίπτωσης

- Υποθέτουμε πως μπορούμε να αντιστοιχίσουμε μια πιθανότητα p(I) σε κάθε είσοδο $I \in D_n$.

- Ορίζουμε την ***πολυπλοκότητα Μέσης Περίπτωσης*** ως

$$A(n) = \sum_{I \in D_n} p(I) \cdot t(I)$$

Εργαλεία Εκτίμησης Πολυπλοκότητας

Ορισμός: Θεωρούμε συνάρτηση T(n). Ορίζουμε

1. $T(n) \in O(f(n))$, αν υπάρχουν θετικές σταθερές c και n_0 ώστε $T(n) \leq c \cdot f(n)$, για κάθε $n \geq n_0$.

2. $T(n) \in \Omega(g(n))$, αν υπάρχουν σταθερές c και n_0 ώστε $T(n) \geq c \cdot g(n)$, για κάθε $n \geq n_0$.

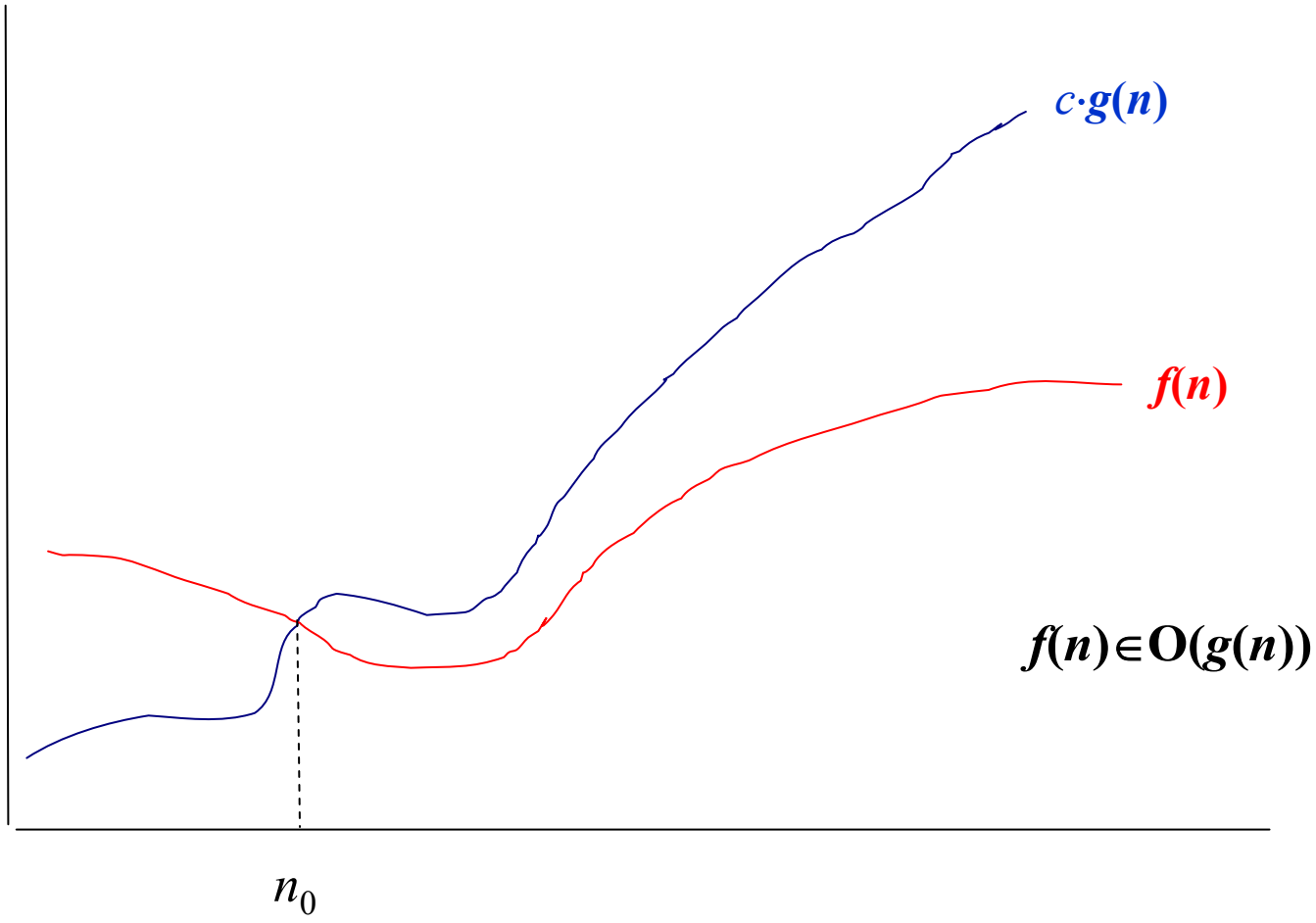
3. $T(n) \in \Theta(h(n))$, αν $T(n) \in O(h(n))$ και $T(n) \in \Omega(h(n))$.

Αν $T(n) \in O(f(n))$, τότε λέμε πως η συνάρτηση T είναι *της τάξεως f(n).*

Αν $T(n) \in \Omega(f(n))$, τότε λέμε πως η T είναι της *τάξεως ωμέγα της f (n).*

Αν $T(n) \in \Theta(f(n))$, τότε λέμε πως η T είναι της *τάξεως θήτα της f (n).*

Γραφική Απεικόνιση της τάξης "O"



Παραδείγματα

- $3n^2 \in O(n^2)$

  $c = 3, n_0 = 1$

- $3n^2 \in O(n^{2.5})$

  $c = 1, n_0 = 9$

- $\sum_{k=0}^{n} k = \frac{n(n+1)}{2} \in O(n^2)$

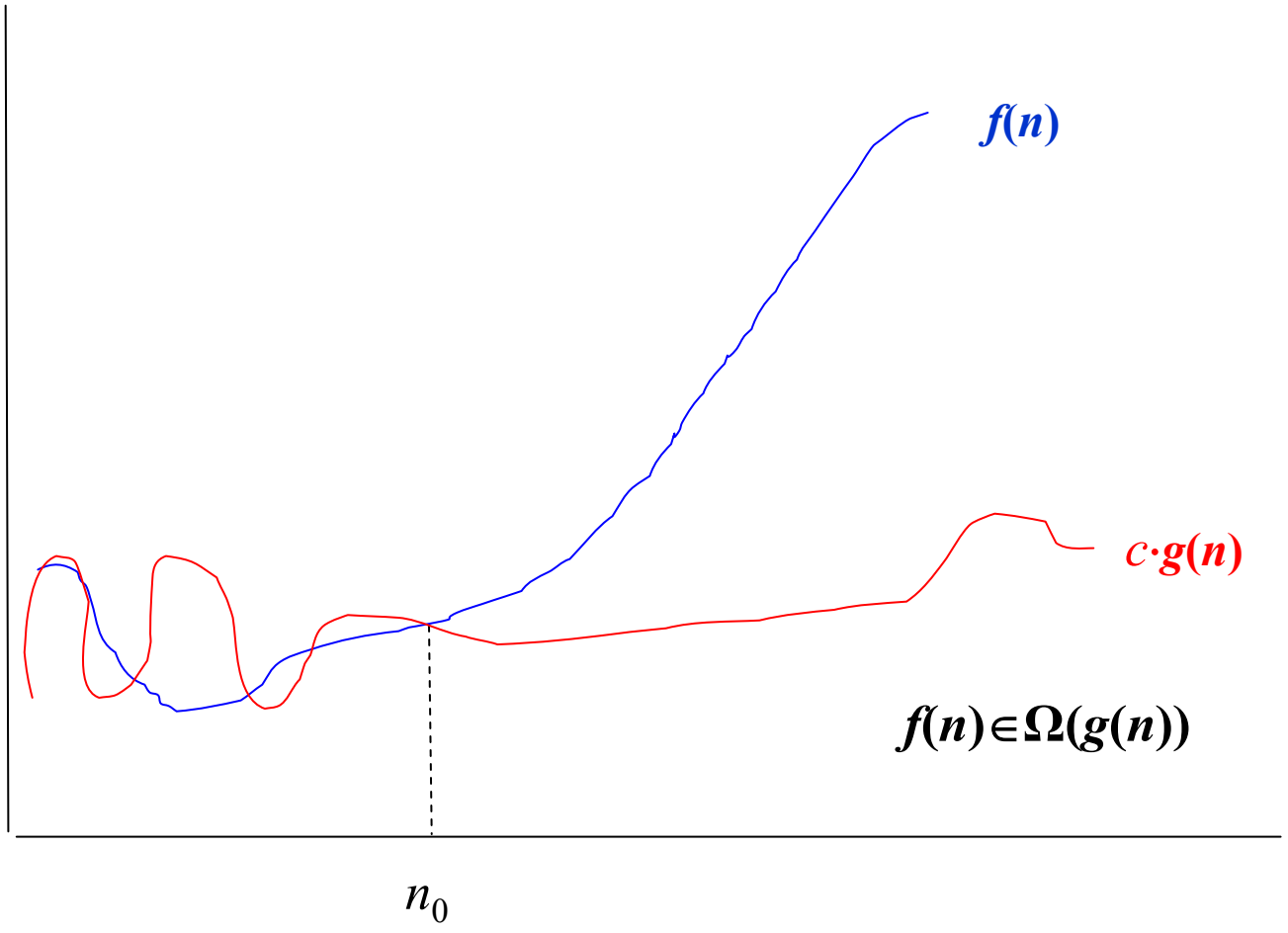
Προσοχή

- Αν $f \in O(g)$ και $h \in O(g) \Rightarrow f = h$
  $g(n) = n^3, f(n) = n, h(n) = n^2$

- Αν $f \in O(g)$ και $h \in O(g) \Rightarrow f \in O(h)$
  $g(n) = n^3, f(n) = n^2, h(n) = n$

- Αν $f \in O(g)$ και $g \in O(h) \Rightarrow f \in O(h)$

Γραφική Απεικόνιση της τάξης "Ω"

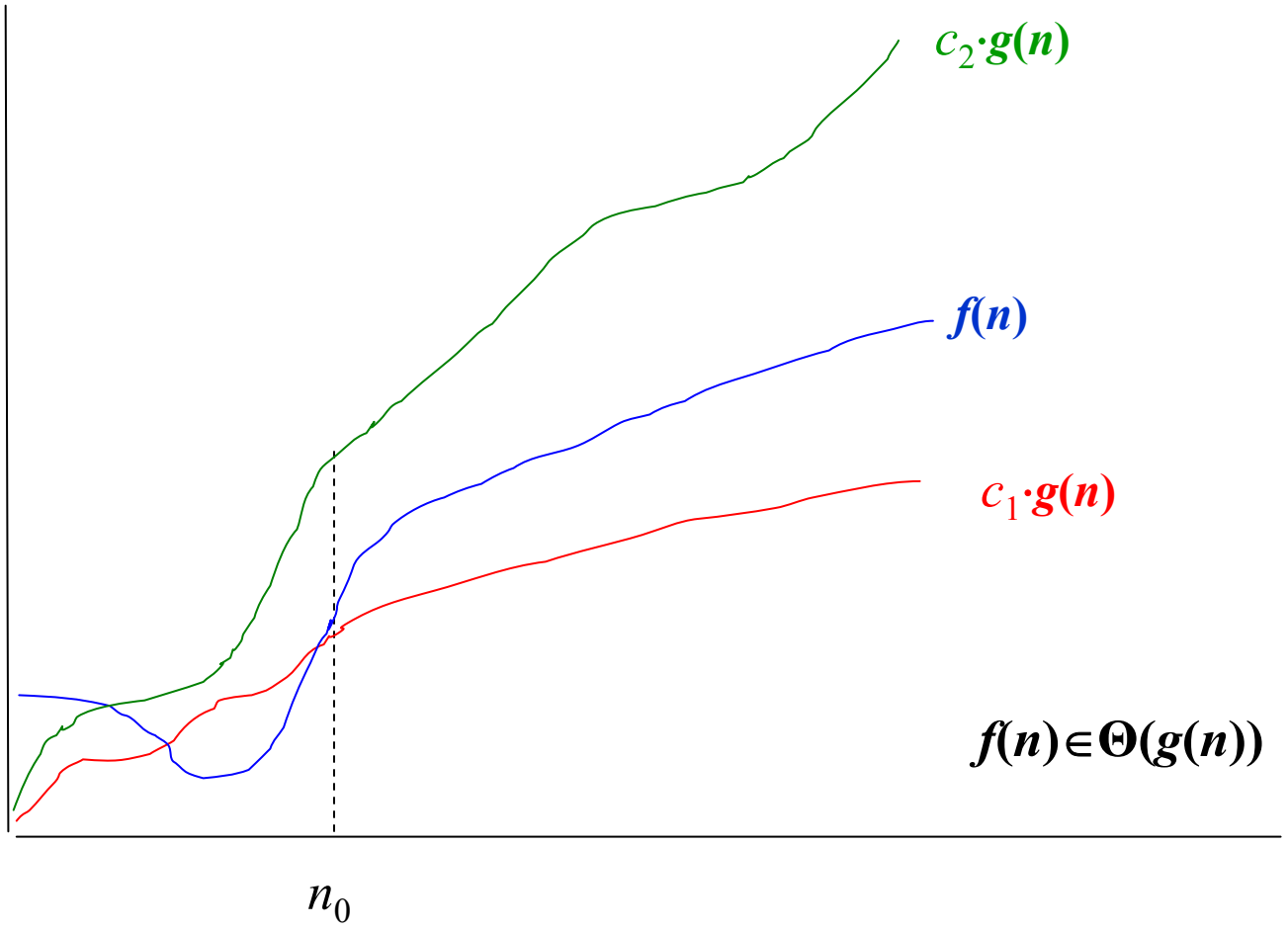


Κανόνες

- Αν $f = \Omega(g)$ και $g = \Omega(h) \Rightarrow f = \Omega(h)$

- $f = \mathrm{O}(g)$ αν και μόνο αν $g = \Omega(f)$

Γραφική Απεικόνιση της τάξης "Θ"



Παραδείγματα

- $\frac{1}{2}n^2 - 3n \in \Theta(n^2)$

  $c_1 = \frac{1}{14}, c_2 = \frac{1}{2}, n_0 = 7$

- $\alpha n^2 + \beta n + \gamma \in \Theta(n^2), \alpha > 0$

Εργαλεία Εκτίμησης Πολυπλοκότητας

Θεώρημα: Αν $T_1 \in O(f)$ και $T_2 \in O(g)$, τότε

1. $T_1 + T_2 \in \max(O(f), O(g))$,
2. $T_1 \cdot T_2 \in O(f \cdot g)$,
3. αν $T(x)$ είναι πολυώνυμο βαθμού k τότε $T(x) \in O(x^k)$,
4. $\log^k n \in O(n)$ για κάθε σταθερά k.

Εργαλεία Εκτίμησης Πολυπλοκότητας

Απόδειξη του 2:

Αφού $T_1 \in O(f)$ και $T_2 \in O(g)$ τότε υπάρχουν n_1, n_2, c_1, c_2 τέτοια ώστε

$$T_1(n) \leq c_1 \cdot f(n),$$

για κάθε $n \geq n_1$ και

$$T_2(n) \leq c_2 \cdot g(n),$$

για κάθε $n \geq n_2$.

Θέτουμε $c = c_1 \cdot c_2$ και $m = \max(n_1, n_2)$. Τότε

$$T_1(n) \cdot T_2(n) \leq c_1 \cdot f(n) \cdot c_2 \cdot g(n) = c \cdot f(n) \cdot g(n),$$

για κάθε $n \geq m$.

Επομένως το ζητούμενο έπεται. □

Εργαλεία Εκτίμησης Πολυπλοκότητας

Παραδείγματα:

$15n + 32 \in O(n)$

$1324 \in O(1)$

$5n^2 \in \Theta(n^2)$

$2n^2 + 4n + 2 \in O(n^2), O(n^3), \ldots$

Με αυτό τον τρόπο μπορούμε να εκφράζουμε ανώτερα (τάξη O) και κατώτερα όρια (τάξη Ω) του χρόνου εκτέλεσης ενός αλγορίθμου.

Προφανώς, κατά την ανάλυση αλγορίθμων, στόχος μας είναι αυτά τα όρια να είναι όσο το δυνατό πιο ακριβή.

Υπολογισμός Χρόνου Εκτέλεσης

1. Ο χρόνος που απαιτείται για την εκτέλεση μιας εντολής `for` είναι το πολύ ο χρόνος εκτέλεσης του βρόχου επί τον αριθμό επαναλήψεων του βρόχου.

2. Φωλιασμένοι βρόχοι: Η ανάλυση γίνεται από τα μέσα προς τα έξω. Προσοχή: αν ο χρόνος εκτέλεσης του εσωτερικού βρόχου εξαρτάται από τον εξωτερικό βρόχο, η ανάλυση πρέπει να το λάβει υπόψη.

3. Ο χρόνος εκτέλεσης της εντολής `S ; S´` παίρνει χρόνο ίσο του αθροίσματος των χρόνων εκτέλεσης των `S` και `S´`.

4. Ο χρόνος εκτέλεσης της εντολής `if b then S else S´` παίρνει χρόνο μικρότερο του αθροίσματος των χρόνων εκτέλεσης των `b`, `S` και `S´`.

5. Ο χρόνος εκτέλεσης μιας αναδρομικής διαδικασίας μπορεί να υπολογιστεί με τη λύση μιας αναδρομικής εξίσωσης.

Παραδείγματα

1.
```
int k=0;
for ( int  i=0; i<n; i++)
    for ( int j=0; j<n; j++)
        k++;
```
n, n^2, $n^2+1 \in O(n^2)$

2.
```
int k=0;
for ( int i=1; i<n; i = 2*i)
   for ( int j=1; j<n; j++)
       k++
```
n, nlg n, $\in O(\text{nlg } n)$

3.
```
int sum=0;
for ( int i=0; i<n; i++)
   for ( int j=0; j<i*i; j++)
       sum++;
```

Λύση παραδείγματος 3

Παρατηρούμε ότι ο χρόνος εκτέλεσης του εσωτερικού βρόχου εξαρτάται από την τιμή `i`, η οποία καθορίζεται από τον εξωτερικό βρόχο. Ο πιο κάτω πίνακας δείχνει το πόσες φορές εκτελείται ο εσωτερικός βρόχος, `N(i)`, σαν συνάρτηση του `i`:

i	0	1	2	…	n-1
N(i)	0	1	4	…	$(n-1)^2$

Άρα $N(i)=i^2$.

Ο χρόνος εκτέλεσης του προγράμματος είναι ίσος με το άθροισμα του χρόνου εκτέλεσης κάθε επανάληψης του εσωτερικού βρόχου, δηλαδή:

$$T(n)=\sum_{i=0}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6} \in \Theta(n^3)$$

Γραμμική - Δυαδική Διερεύνηση

- *Δεδομένα Εισόδου*: Πίνακας X με n στοιχεία, ταξινομημένος από το μικρότερο στο μεγαλύτερο, και ακέραιος k.
- *Στόχος*: Να εξακριβώσουμε αν το k είναι στοιχείο του X.
- *Γραμμική Διερεύνηση*: εξερευνούμε τον πίνακα από τα αριστερά στα δεξιά.

```
int linear( int X[], int n, int k){
 int i=0;
 while ( i < n )
   if (X[i] == k) return i;
   if (X[i] > k)  return -1;
   i++;
 return -1;
}
```

- *Χρόνος εκτέλεσης*: Εξαρτάται από το που (και αν) ο k βρίσκεται στον X[n].
- *Χείριστη περίπτωση*: O(n)

Γραμμική - Δυαδική Διερεύνηση

- *Δυαδική Διερεύνηση:* βρίσκουμε το μέσο του πίνακα και αποφασίζουμε αν το k ανήκει στο δεξιό ή το αριστερό μισό. Επαναλαμβάνουμε την ίδια διαδικασία στο "μισό" που μας ενδιαφέρει.

```
int binary( int X[],int n,int k){
 int low = 0, high = n-1;
 int mid;
 while ( low < high ){
   mid = (high + low)/2;
   if (X[mid] < k) low = mid + 1;
   else
   if (X[mid] > k) high=mid-1;
   else return mid;
  }
 return -1;
}
```

- *Χρόνος εκτέλεσης*; O(lg n)

Ο Αλγόριθμος Ταξινόμησης InsertionSort

```
void InsertionSort(int A[], int n){
   int temp;

   for (int i=2; i<=n; i++){
        temp=A[i];
        for (int j=i;(j>1)&&(temp < A[j-1]); j--)
              A[j]=A[j-1];
              A[j]=temp;
   }
}
```

Χρόνος Εκτέλεσης Χειρίστης Περίπτωσης: $O(n^2)$

Ανάλυση Μέσης Περίπτωσης

- Έστω ότι έχουμε n στοιχεία, το καθένα διαφορετικό από όλα τα άλλα.
- Υπάρχουν n! διαφορετικές τοποθετήσεις των στοιχείων. Έστω ότι είναι όλες εξίσου πιθανές.
- Για να υπολογίσουμε τον χρόνο εκτέλεσης μέσης περίπτωσης μπορούμε
  1. να προσθέσουμε τους χρόνους που απαιτούνται για τις ταξινομήσεις της κάθε μιας από τις πιθανές τοποθετήσεις και να διαιρέσουμε δια n!, ή
  2. να αναλύσουμε άμεσα το χρόνο εκτέλεσης μέσης περίπτωσης επιχειρηματολογώντας πιθανοτικά:
- Για κάθε $2 \leq i \leq n$, θεωρήστε τον υποπίνακα A[1..i].
- Ορίζουμε τη συνάρτηση **θέση(**i) που για κάθε i μας δίνει τη θέση στην οποία θα βρισκόταν το στοιχείο A[i] αν ο πίνακας A[1..i] ήταν ταξινομημένος.

  π.χ. Αν A=[4,2,1,5,7,3] τότε **θέση**[3]=1.

Ανάλυση Μέσης Περίπτωσης

- Παρατήρηση: Αν όλες οι τοποθετήσεις των στοιχείων του πίνακα Α είναι εξίσου πιθανές, τότε όλες οι τιμές της συνάρτησης **θέση**(i), οι οποίες μπορεί να ανήκουν στο σύνολο [1..i] είναι επίσης εξίσου πιθανές.

- Αυτή η ιδιότητα ισχύει κάθε φορά που εκτελούμε τον εσωτερικό βρόχο του InsertionSort.

- Αν θέση(i) = k, τότε ο εσωτερικός βρόχος θα εκτελεστεί i-k+1 φορές.

- Άρα ο μέσος όρος βημάτων κατά την i-οστή εκτέλεση του εξωτερικού βρόχου είναι $$c_i = \frac{1}{i} \cdot \sum_{k=1}^{i} (i-k+1) = \frac{i+1}{2}$$

  Και ο συνολικός μέσος όρος βημάτων είναι

$$\sum_{i=2}^{n} c_i = \sum_{i=2}^{n} \frac{i+1}{2} = \frac{(n-1)(n+4)}{4} = \Theta(n^2)$$

Ανάλυση Αναδρομικού Αλγόριθμου

MergeSort

Η πιο κάτω διαδικασία ταξινομεί τον πίνακα A[l, r] με συγχώνευση, όπου θεωρούμε ότι η διαδικασία `Merge(A[l,mid], A[mid+1,r])` συγχωνεύει τους δύο ταξινομημένους υποπίνακες `A[l,mid]`, και `A[mid+1,r]` σε ένα ταξινομημένο πίνακα.

```
MergeSort(A[],int l, int r){
   if (l == r) return;
   int mid = (l+r)/2;
   MergeSort(A, l, mid);
   MergeSort(A, mid + 1, r);
   Merge(A[l,mid], A[mid+1,r]);
}
```

Ανάλυση Αναδρομικού Αλγόριθμου

- Έστω T(n) ο χρόνος εκτέλεσης της διαδικασίας MergeSort με δεδομένο εισόδου l=1, r=n. Τότε έχουμε

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2 \cdot T(n/2) + n, & \text{otherwise} \end{cases}$$

- Λύση της αναδρομικής εξίσωσης με τη μέθοδο της επανάληψης:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + n \\ &= 2 \cdot (2 \cdot T(n/4) + n/2) + n = 2^2 \cdot T(n/2^2) + 2n \\ &= \ldots \\ &= 2^i \cdot T(n/2^i) + in \\ &= \{k = \lg n\} \\ &\quad 2^k \cdot T(n/2^k) + kn \\ &= n \cdot T(1) + n \lg n = n + n \lg n \end{aligned}$$

- Μπορεί να αποδειχθεί ότι ο αλγόριθμος είναι άριστος για το πρόβλημα. δηλ. ότι το πρόβλημα είναι της τάξης Θ(n lg n).