Πολυπλοκότητα Αλγορίθμων

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Πρόβλημα, Στιγμιότυπο, Αλγόριθμος

Εμπειρική – Θεωρητική Ανάλυση Αλγορίθμων

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

Αναδρομικές Εξισώσεις

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: *η ανάλυση και ο υπολογισμός της πολυπλοκότητας χρόνου και χώρου (time and space complexity) των αλγορίθμων και ο έλεγχος αν και πότε ένας αλγόριθμος είναι* **άριστος** *(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 τον χώρο εκτέλεσής του σε συγκεκριμένο υπολογιστή.
- **Θεωρητικά** μπορούμε να υπολογίσουμε το χρόνο και το 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 στιγμιοτύπου.

  π.χ. Πρόβλημα: ταξινόμηση λίστας.

  Στιγμιότυπο: λίστα με n στοιχεία.

  Μέγεθος:
- Πλεονεκτήματα της θεωρητικής προσέγγισης υπολογισμού αποτελεσματικότητας αλγορίθμων περιλαμβάνουν:
  1. δεν εξαρτάται από το υλικό του Η/Υ (μνήμη, cache, κλπ)
  2. δεν εξαρτάται από τη γλώσσα προγραμματισμού ή το 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

- Υπολογιστής που εκτελεί οδηγίες διαδοχικά.
- **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 το 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.

Παράδειγμα Ανάλυσης 1

```
int largest( int X[], int n){
1       int current=0;
2       int i=0;
3       while ( i < n ){
4             if ( X[i] > current){
5             current = X[i]; }
6             i = i+1;
7       }
8  return current;
}
```

Μέγεθος δεδομένων εισόδου: n
Στόχος: υπολογισμός του αριθμού βασικών πράξεων
t(n) = ...

Μπορούμε να πούμε πως ο αλγόριθμος είναι άριστος;

Παράδειγμα Ανάλυσης 2

Γραμμική Διερεύνηση

```
int index( int X[],int n,int k){
1       int i=0;
2       while (i < n and X[i]!= k)
3             i= i+1;
4       return i;
}
```

Μέγεθος δεδομένων εισόδου: n

t(n) = ...

Ανάλυση Χειρίστης Περίπτωσης

- Αν D_n είναι το σύνολο όλων των εισόδων (στιγμιοτύπων) 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

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

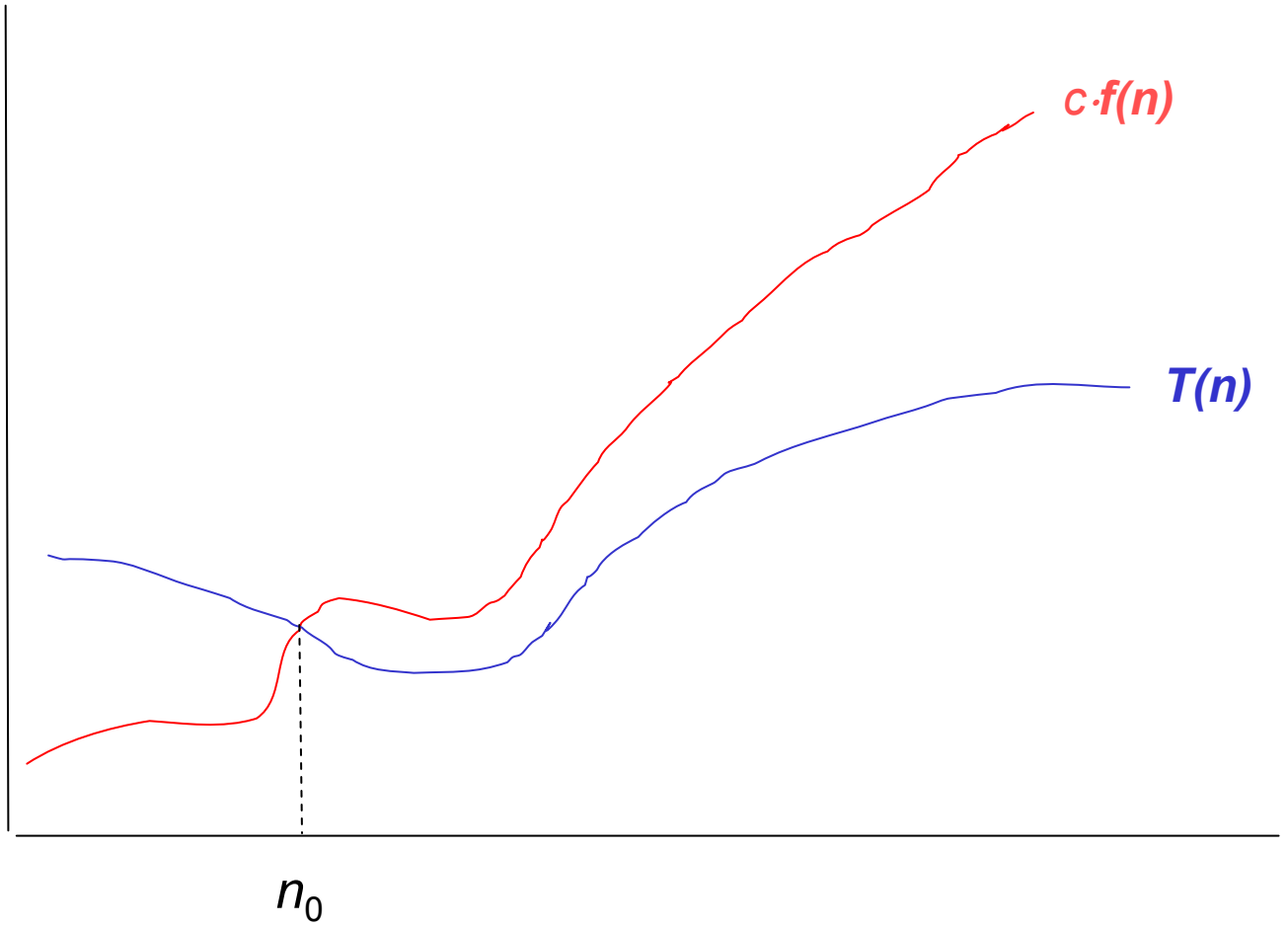
Εργαλεία Εκτίμησης Πολυπλοκότητας

Ορισμός: Θεωρούμε συνάρτηση $T(n)$. Ορίζουμε

1. $T(n) \in O(f(n))$, αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ τέτοιες ώστε $T(n) \leq c \cdot f(n)$, για κάθε $n \geq n_0$.

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

Γραφική Απεικόνιση $T(n) \in O(f(n))$



Εργαλεία Εκτίμησης Πολυπλοκότητας

Ορισμός: Θεωρούμε συνάρτηση $T(n)$. Ορίζουμε

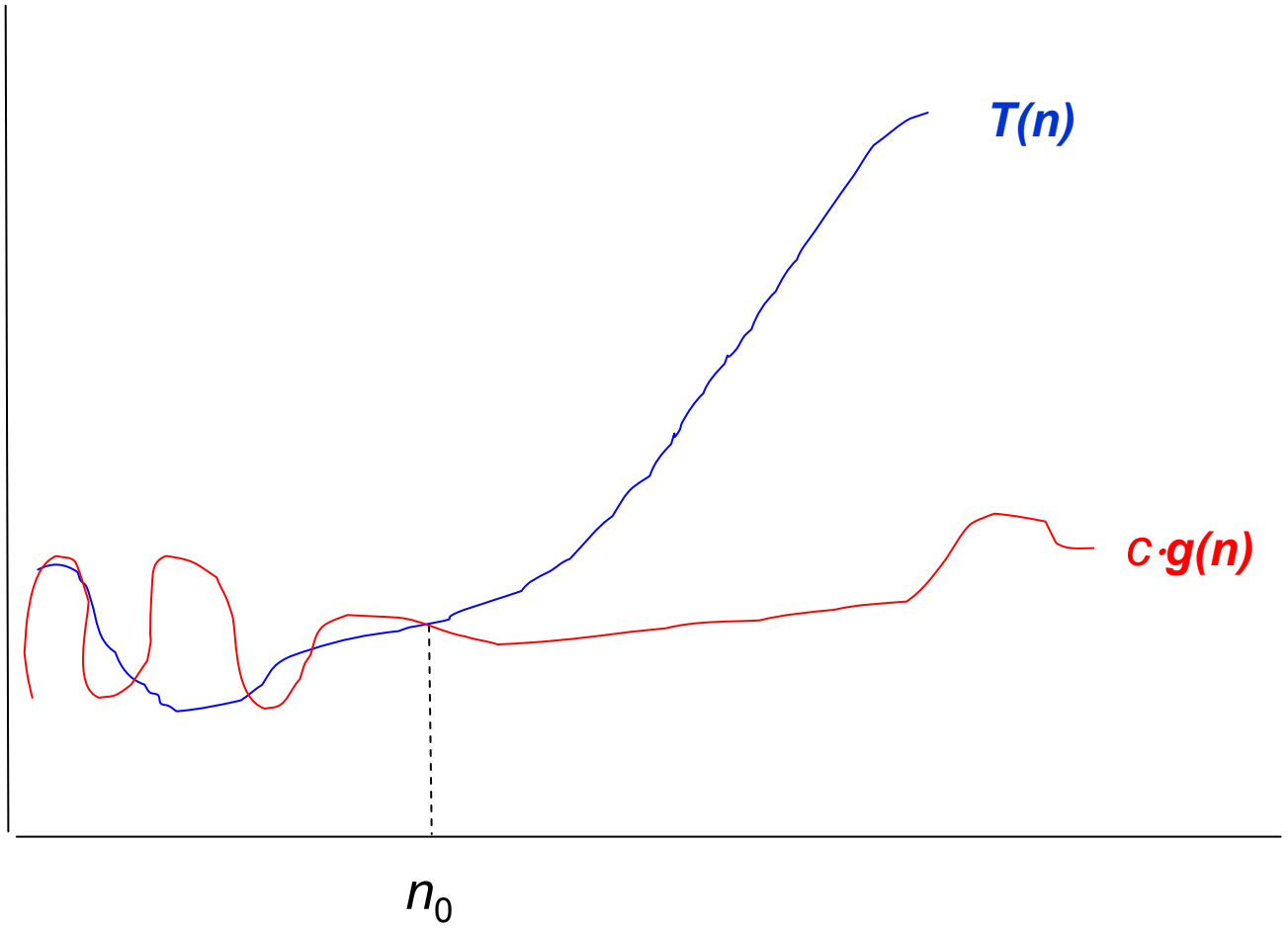
1. $T(n) \in O(f(n))$, αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ τέτοιες ώστε $T(n) \leq c \cdot f(n)$, για κάθε $n \geq n_0$.

2. $T(n) \in \Omega(g(n))$, αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ τέτοιες ώστε $T(n) \geq c \cdot g(n)$, για κάθε $n \geq n_0$.

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 $f(n)$.*

Γραφική Απεικόνιση $T(n) \in \Omega(g(n))$



Εργαλεία Εκτίμησης Πολυπλοκότητας

Ορισμός: Θεωρούμε συνάρτηση $T(n)$. Ορίζουμε

1. $T(n) \in O(f(n))$, αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ τέτοιες ώστε $T(n) \leq c \cdot f(n)$, για κάθε $n \geq n_0$.

2. $T(n) \in \Omega(g(n))$, αν υπάρχουν σταθερές $c > 0$ και $n_0 \geq 0$ τέτοιες ώστε $T(n) \geq c \cdot g(n)$, για κάθε $n \geq n_0$.

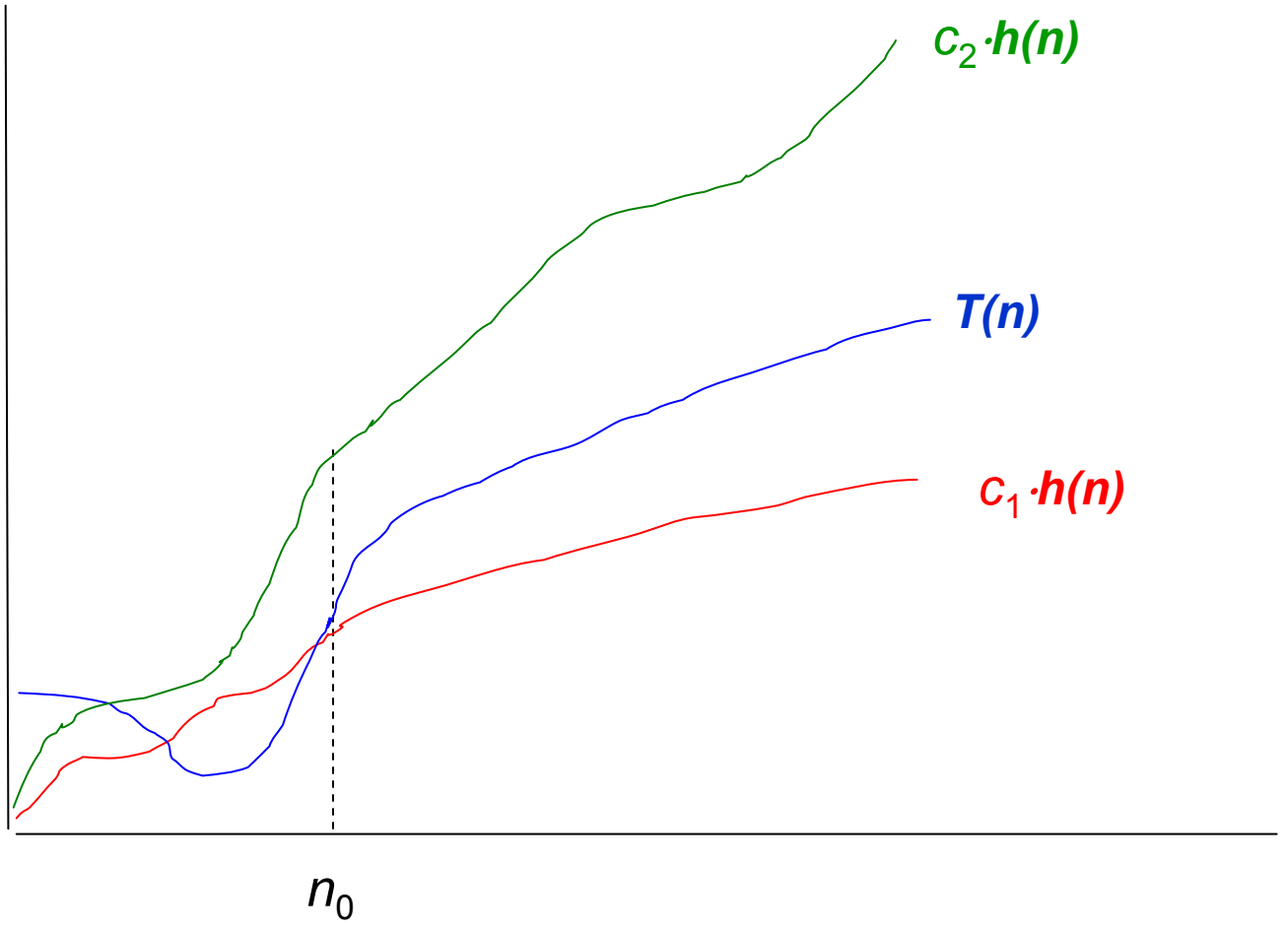
3. $T(n) \in \Theta(h(n))$, αν $T(n) \in O(h(n))$ και $T(n) \in \Omega(h(n))$.

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 $T(n) \in \Theta(h(n))$



Εργαλεία Εκτίμησης Πολυπλοκότητας

1. Αν $T_1 \in O(f)$ και $T_2 \in O(g)$, τότε

   a) $T_1 + T_2 \in \max(O(f), O(g))$,
   b) $T_1 \cdot T_2 \in O(f \cdot g)$

2. Αν $f \in O(g)$ και $g \in O(h)$, τότε $f \in O(h)$
   (παρόμοια ισχύει για Ω και Θ)

3. $f \in O(g)$ *αν και μόνο αν* $g \in \Omega(f)$

4. $f \in \Theta(g)$ *αν και μόνο αν* $g \in \Theta(f)$

5. Αν $T(x)$ είναι πολυώνυμο βαθμού k τότε $T(x) \in O(x^k)$

6. $\log^k n \in O(n)$ για κάθε σταθερά k.

Εργαλεία Εκτίμησης Πολυπλοκότητας

Απόδειξη του 1(b):

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 το 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.

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´`.

Παραδείγματα

1.
```
int k=0;
 for ( int  i=0; i<n; i++)
     for ( int j=0; j<n; j++)
         k++;
```

2.
```
int k=0;
 for ( int i=1; i<n; i = 2*i)
    for ( int j=1; j<n; j++)
        k++
```

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*:

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*;

Αναδρομικές διαδικασίες

- Αν σε κάθε επανάληψη μιας αναδρομικής διαδικασίας το μέγεθος του προβλήματος *μειώνεται κατά μια σταθερά τιμή*, και ο χρόνος εκτέλεσης κάθε επανάληψης είναι T, τότε έχουμε αλγόριθμο τάξης: O(T·n)

- Αν σε κάθε επανάληψη ενός αναδρομικού αλγορίθμου το μέγεθος του προβλήματος *μοιράζεται κατά μια σταθερά τιμή*, και ο χρόνος εκτέλεσης κάθε επανάληψης είναι T, τότε έχουμε αλγόριθμο τάξης: $O(T \cdot \log n)$

Αριθμοί Fibonacci – Αναδρομικές Διαδικασίες

```
int Fib( int N) {
if (N <= 1)
   return 1;
else
   return Fib(N-1) + Fib(N-2);
}
```

- Ας υποθέσουμε πως το Fib(n) υπολογίζεται από τον αλγόριθμο σε χρόνο T(n). Τότε

  $T(0) = 1$

  $T(1) = 1$

  $T(k) = T(k-1) + T(k-2) + 2$

- Αναλύοντας την πιο πάνω αναδρομική σχέση μπορούμε να αποδείξουμε πως

$$\left(\frac{3}{2}\right)^n \leq T(n) \leq \left(\frac{5}{3}\right)^n$$

- Άρα ο αλγόριθμος είναι *εκθετικός*.

Αριθμοί Fibonacci – Αναδρομικές Διαδικασίες

```
long int Fib2(int n){
  long int res [ max ];
  if (n >=  max) return Error;
  res [0] = res [1] = 1;
  for (i = 2; i<= n; i++)
      res [i] = res[i-1] + res [i-2];
  return res[n]
}
```

Αποφεύγοντας επανάληψη υπολογισμών, ο πιο πάνω αλγόριθμος είναι *γραμμικός*, δηλ. εκτελείται σε χρόνο τάξης O(n).

Αναδρομικές Εξισώσεις

- Όπως είδαμε στην περίπτωση της διαδικασίας Fib, ο υπολογισμός του χρόνου εκτέλεσης μιας αναδρομικής διαδικασίας συνήθως συνεπάγεται τον προσδιορισμό και τη λύση κάποιας αναδρομικής εξίσωσης.

- Για τους σκοπούς του μαθήματος συνιστώνται οι πιο κάτω τεχνικές για την επίλυση αναδρομικών εξισώσεων.
  1. Η μέθοδος της επαγωγής.
  2. Η μέθοδος της αντικατάστασης.

Μέθοδος της επαγωγής

1. Προβλέπουμε μια συνάρτηση f(n) ως λύση της εξίσωσης, και
2. Επαληθεύουμε τη λύση με επαγωγή προσδιορίζοντας κατάλληλα τις σταθερές.

Παράδειγμα

Έχουμε την αναδρομική εξίσωση

$T(n) = 2 \cdot T(n/2) + n, \qquad n \geq 2$

$T(1) = 1$

Προβλέπουμε ότι $T(n) \in O(n^2)$. *Δηλαδή, θα πρέπει να υπάρχουν σταθερά c και τιμή m τέτοιες ώστε για κάθε* $n > m$

$T(n) \leq cn^2$

Θα αποδείξουμε το πιο πάνω επαγωγικά.

Για n=1 και c>=1 η πρόταση ισχύει.

Μέθοδος της επαγωγής: Παράδειγμα

Υπόθεση της επαγωγής: Έστω ότι $T(k) \leq ck^2$ για κάθε $m<k<n$.

Θα αποδείξουμε ότι $T(n) \leq cn^2$ προσδιορίζοντας κατάλληλα τη σταθερά c.

Εχουμε:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + n \\ &\leq 2 \cdot c(n/2)^2 + n \\ &= (c/2) \cdot n^2 + n \\ &= c\, n^2 - ((c/2) \cdot n^2 - n) \end{aligned}$$

Άρα $T(n) \leq cn^2$ αν $(c/2) \cdot n^2 - n \geq 0$.

Δηλαδή, αν $c \geq 2/n$ $(n>0)$.

Για να ισχύει η ανισότητα αρκεί $c \geq 2$.

Μέθοδος της επαγωγής: Παράδειγμα

Αποδείξαμε ότι $T(n) \in O(n^2)$. Μπορούμε όμως να βρούμε ακριβέστερη λύση…

Προβλέπουμε ότι $T(n) \in O(n \lg n)$. Δηλαδή, θα αποδείξουμε την ύπαρξη σταθεράς c και τιμής m τέτοιες ώστε για κάθε $n>m$

$$T(n) \leq c \cdot n \lg n$$

Θα αποδείξουμε το πιο πάνω επαγωγικά.

Βάση της επαγωγής: για n=2, το ζητούμενο ισχύει για οποιαδήποτε τιμή $c \geq 2$.

Υπόθεση της επαγωγής: Έστω ότι $T(k) \leq c \cdot k \lg k$ για κάθε $k<n$.

Θα αποδείξουμε ότι $T(n) \leq c \cdot n \lg n$ προσδιορίζοντας κατάλληλα τη σταθερά c.

Εχουμε:

$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + n \\ &\leq 2c \cdot n/2 \lg (n/2) + n \\ &= c \cdot n (\lg n - 1) + n \\ &= c \cdot n \lg n - (cn - n) \end{aligned}$$

Άρα $T(n) \leq c \cdot n \lg n$ αν $c \cdot n - n \geq 0$. Δηλαδή, αν $c \geq 2$ $(n>0)$.

Επομένως $T(n) \in O(n \lg n)$.

Μέθοδος της αντικατάστασης

Χρησιμοποιούμε το βήμα της αναδρομής επανειλημμένα, ώστε να εκφράσουμε το T(n) ως συνάρτηση που περιέχει μόνο τη βασική περίπτωση, δυνάμεις του n και σταθερές τιμές.

Παράδειγμα

Έχουμε την αναδρομική εξίσωση

$$T(n) = 4 \cdot T(n/2) + n, \qquad \text{για κάθε } n \geq 2$$
$$T(1) = 1$$

Τότε, αντικαθιστώντας το T(n/2) με την τιμή του παίρνουμε

$$\begin{aligned} T(n) &= 4 \cdot T(n/2) + n \\ &= 4(4 \cdot T(n/4) + n/2) + n \\ &= 4^2 \cdot T(n/4) + 2n + n \\ &= 4^3 \cdot T(n/8) + 2^2 n + 2n + n \\ &= \ldots \end{aligned}$$

Μέθοδος της αντικατάστασης - Παράδειγμα

Διακρίνουμε τη γενική μορφή

$$T(n) = 4^i \cdot T\left(\frac{n}{2^i}\right) + 2^{i-1}n + \ldots + 2n + n$$

Υποθέτουμε ότι το n είναι δύναμη του 2 και k=lg n. Τότε

$$T(n) = 4^k \cdot T\left(\frac{n}{2^k}\right) + 2^{k-1}n + \ldots + 2n + n$$

$$= 4^k + n \cdot \sum_{i=0}^{k-1} 2^i = (2^k)^2 + n \cdot \frac{2^k - 1}{2 - 1}$$

$$= n^2 + n(n-1)$$

$$= 2n^2 - n$$

Εργασία 1

- Να λύσετε τις πιο κάτω αναδρομικές εξισώσεις:

$T(1) = 1$

$T(n) = 2 \cdot T(n/2) + 1000n$

Εργασία 2

$T(1) = 1$

$T(n) = 7 \cdot T(n/2) + 18n^2$

Εργασία 3

- Να υπολογίσετε τον χρόνο εκτέλεσης της παρακάτω αναδρομικής διαδικασίας λύνοντας οποιαδήποτε αναδρομική εξίσωση συναντήσετε.

```
recursive1(int n){
int sum=0;
    for ( int i=1; i <= n; i++)
            sum++;
    if (n>1)      return recursive1(n/2);
    else   return 1;
}
```

Εργασία 4

- Να υπολογίσετε τον χρόνο εκτέλεσης της παρακάτω αναδρομικής διαδικασίας λύνοντας οποιαδήποτε αναδρομική εξίσωση συναντήσετε.

```
recursive2(int n){
int sum=0;
    for ( int i=1; i <= n; i++)
            sum++;
    if (n>1)      return (recursive2(n/2) +
                                recursive2(n/2));
    else   return 1;
   }
```