Ανάλυση Αλγορίθμων

Σύντομη επανάληψη (ΕΠΛ 035).

Περίληψη

- Ανάλυση αλγορίθμων
  - *O, Θ, Ω*
  - Ανάλυση μη αναδρομικών αλγόριθμων
  - Ανάλυση αναδρομικών αλγόριθμων
  - Εμπειρική Ανάλυση
  - Visualization

Απόδοση Αλγορίθμων

- Απόδοση Αλγορίθμου
  - Σε σχέση με το **χρόνο** που χρειάζεται για να τερματίσει
  - Σε σχέση με το **χώρο** (μνήμη) που χρειάζεται
- Η απόδοση ενός αλγορίθμου εξαρτάται από το μέγεθος της εισόδου (input size) n
  - Ο τρόπος με τον οποίο επιλέγεται το n επηρεάζει τον υπολογισμό της τάξης (order) του αλγορίθμου!
  - Υποθέστε πως σχεδιάζετε ένα αλγόριθμο για τον υπολογισμό του γινομένου δύο πινάκων $n \times n$. Ποιο το μέγεθος της εισόδου;
    - $n;$ (οι διαστάσεις του πίνακα)
    - $N=n \times n;$ (ο αριθμός στοιχείων του πίνακα)

Μονάδες Μέτρησης

- Χρόνος ολοκλήρωσης του αλγόριθμου (δευτερόλεπτα, ώρες, μέρες…)
  - **Αποδεκτή** μέθοδος για τη σύγκριση μεταξύ δύο ή περισσοτέρων αλγορίθμων
  - Μειονεκτήματα:
    - Εξαρτάται από πολλές παραμέτρους που δεν έχουν σχέση με τον αλγόριθμο! Π.χ., εξαρτάται από τη ταχύτητα και μνήμη του υπολογιστή, το λειτουργικό σύστημα και πιθανόν στις άλλες διεργασίες που τρέχουν παράλληλα στον υπολογιστή.
- Φορές που εκτελείται η **βασική λειτουργία** (basic operation) του αλγόριθμου
  - Βασική λειτουργία είναι η λειτουργία που επαναλαμβάνεται τις περισσότερες φορές και αναμένεται να χρειαστεί τον περισσότερο χρόνο.
  - Ποια λειτουργία χρειάζεται τον περισσότερο χρόνο;
    - Η πρόσθεση ή ο πολλαπλασιασμός;

«Εκτίμηση» του χρόνου εκτέλεσης (running time)

- Υποθέστε πως
  - ο χρόνος που χρειάζεται για να εκτελεστεί η **βασική λειτουργία** (basic operation) είναι c_{op}
  - $C(n)$ είναι ο αριθμός των φορών που εκτελείται η βασική λειτουργία εάν η είσοδος έχει μέγεθος n.
- Εάν $T(n)$ είναι ο χρόνος εκτέλεσης (running time) τότε ισχύει:

«Εκτίμηση» του χρόνου εκτέλεσης (running time)

- Παράδειγμα:
  - Υποθέστε πως για κάποιο αλγόριθμο ή βασική λειτουργία επαναλαμβάνεται $C(n)=½\ [n\times(n-1)]$ φορές
  - Τι θα συμβεί εάν το μέγεθος της εισόδου **διπλασιαστεί**;
  - Ο χρόνος εκτέλεσης του αλγόριθμου θα
- Παρατήρηση:
  - Οι σταθερές c_{op} και ½ απλοποιούνται.
- Συμπέρασμα:
  - Κάποιος μπορεί να προβλέψει την απόδοση του αλγόριθμου ξέροντας μόνο (κατά προσέγγιση) την τάξη αύξησης (order of growth) του αλγόριθμου!

Σενάρια Υπολογισμού

- Η τάξη αύξησης του χρόνου εκτέλεσης μπορεί να υπολογιστεί κάτω από τις πιο κάτω συνθήκες (υποθέτοντας πως η είσοδος είναι μεγέθους n):
- **Καλύτερη Περίπτωση (best case).**
  - Το σενάριο στο οποίο ο αλγόριθμος παρουσιάζει την καλύτερη απόδοση.
  - Αυτό το σενάριο δεν υλοποιείται συχνά!
- **Χειρότερη Περίπτωση (worst case)**
  -
  - Παρόλο που δεν είναι πάντοτε αντιπροσωπευτικό, στις πλείστες περιπτώσεις η τάξη του σεναρίου είναι αντιπροσωπευτική.
- **Μέση Περίπτωση (average case)**
  - Η πιο αντιπροσωπευτική περίπτωση
  - Δύσκολο να υπολογιστεί αφού χρειάζεται υποθέσεις για την κατανομή των δεδομένων εισόδου.

Παράδειγμα

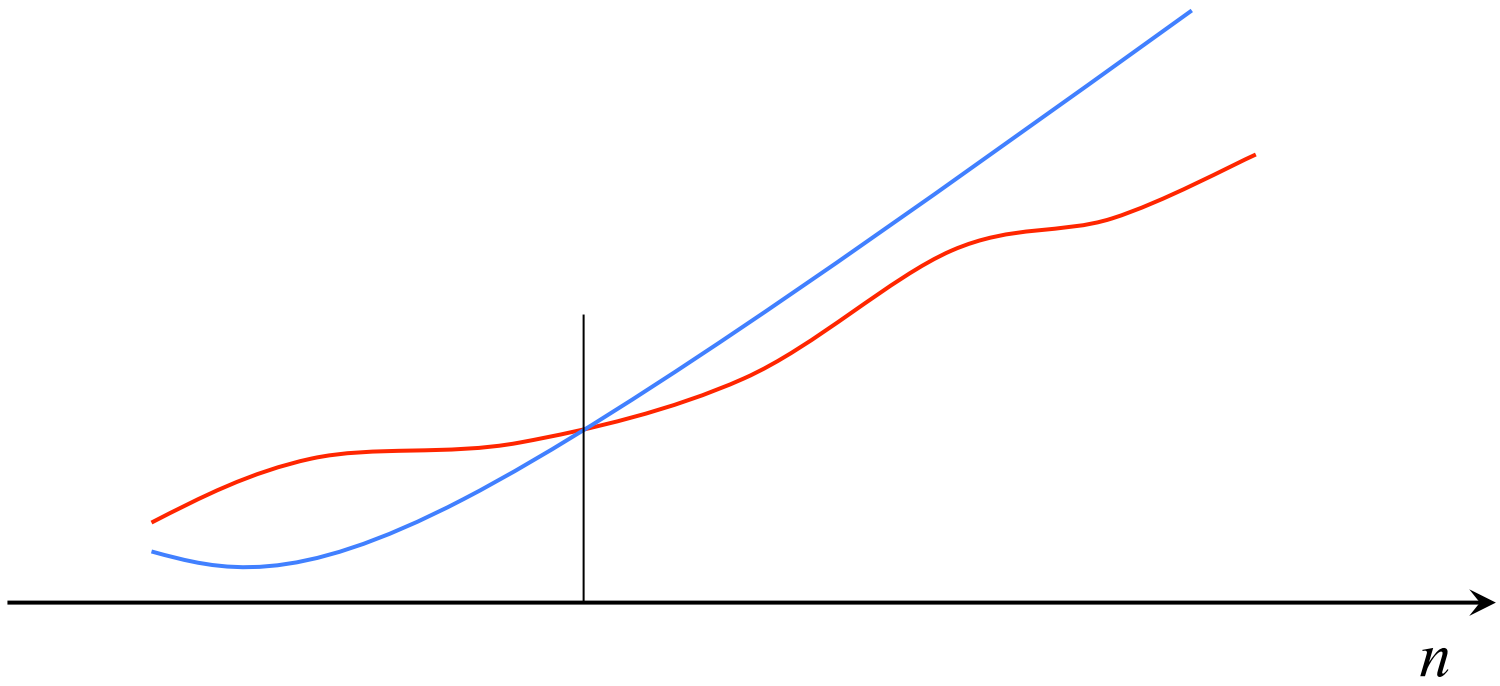
```
Search(A[], K)
  i=0
  While (i<n) and (A[i]!= K)
     i=i+1;
   If i<n return i;
   Return -1;
```

- **Καλύτερη Περίπτωση:**
- **Χειρότερη Περίπτωση:**
- **Μέση Περίπτωση:**

- p πιθανότητα επιτυχίας
- 1-p πιθανότητα το κλειδί *K* δεν βρίσκεται στη λίστα *A*[]
- Εάν το κλειδί βρίσκεται στην λίστα τότε μπορεί να βρεθεί σε οποιαδήποτε θέση με ίση πιθανότητα

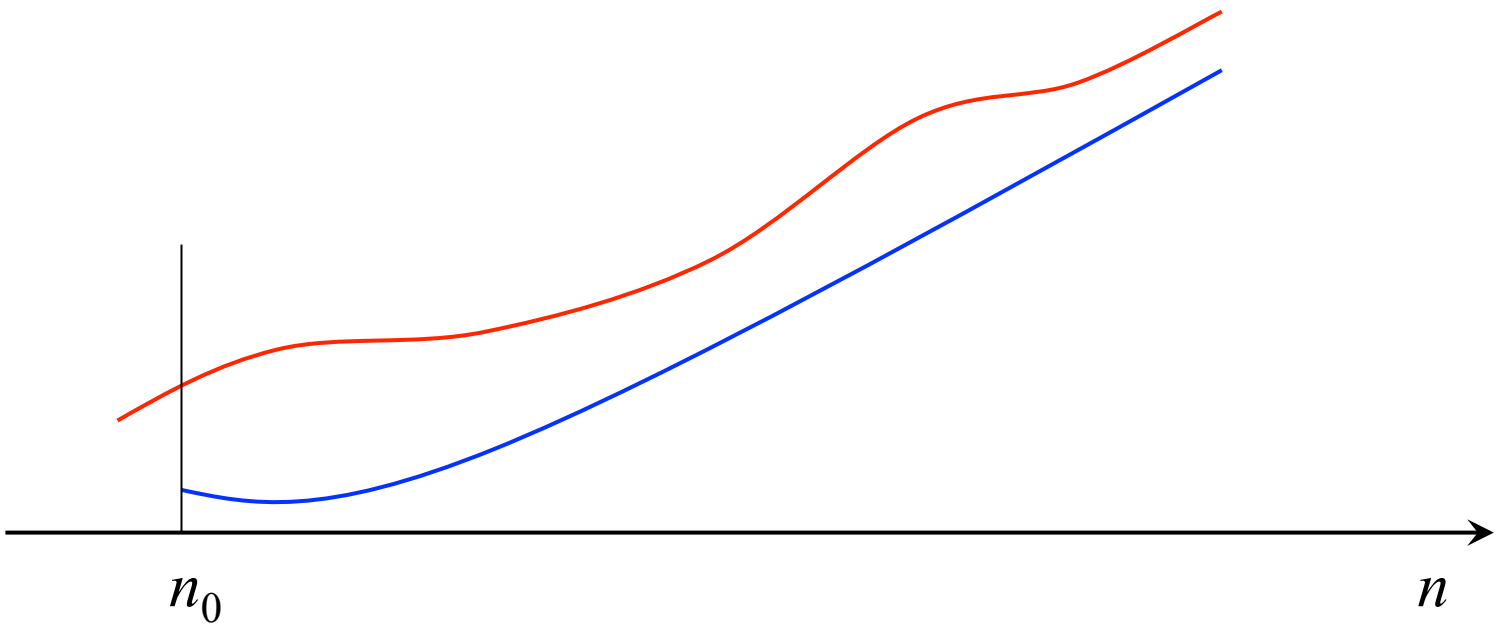
Συμβολισμός $O()$

- Μία συνάρτηση $f(n)$ ανήκει στο σύνολο $O(g(n))$ ($f(n) \in O(g(n))$) εάν υπάρχουν σταθερές c και n_0 έτσι ώστε



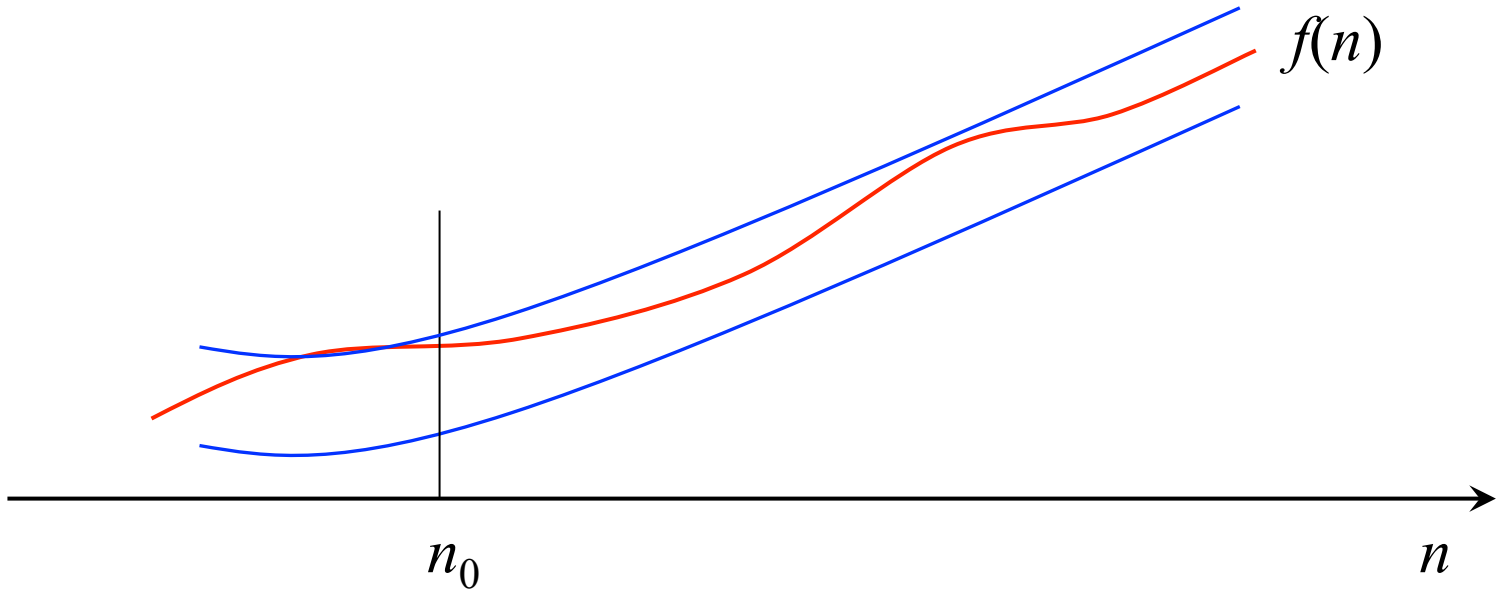
Συμβολισμός $\Omega()$

- Μία συνάρτηση $f(n)$ ανήκει στο σύνολο $\Omega(g(n))$ ($f(n) \in \Omega(g(n))$) εάν υπάρχουν σταθερές c και n_0 έτσι ώστε



Συμβολισμός $\Theta()$

- Μία συνάρτηση $f(n)$ ανήκει στο σύνολο $\Theta(g(n))$ ($f(n) \in \Theta(g(n))$) εάν υπάρχουν σταθερές c_1, c_2 και n_0 έτσι ώστε



Παράδειγμα

- Υποθέστε ότι: $f(n)=\frac{1}{2}n(n-1)$
- Ποια από τα πιο κάτω ισχύουν:

$f(n)\in O(1)?$

$f(n)\in \Omega(1)?$

$f(n)\in O(n)?$

$f(n)\in O(n^2)?$

$f(n)\in \Omega(n^2)?$

$f(n)\in \Theta(n^2)?$

Θεώρημα

- Υποθέστε ότι $f_1(n) \in O(g_1(n))$ και $f_2(n) \in O(g_2(n))$, τότε ισχύει ότι

$$f_1(n) + f_2(n) \in O\left(\max\left\{g_1(n), g_2(n)\right\}\right)$$

- Απόδειξη

$$f_1(n) \in O(g_1(n)) \Leftrightarrow \exists c_1, n_1, \text{s.t. } f_1(n) \le c_1 g_1(n), \forall n \ge n_1$$

$$f_2(n) \in O(g_2(n)) \Leftrightarrow \exists c_2, n_2, \text{s.t. } f_2(n) \le c_2 g_2(n), \forall n \ge n_2$$

$$\begin{aligned} f_1(n) + f_2(n) &\le c_1 g_1(n) + c_2 g_2(n), \forall n \ge \max\{n_1, n_2\} \\ &\le c_3 g_1(n) + c_3 g_2(n) \qquad \text{οπου} \quad c_3 = \max\{c_1, c_2\} \\ &\le 2c_3 \max\left\{g_1(n), g_2(n)\right\} \end{aligned}$$

- Γιατί το θεώρημα αυτό είναι σημαντικό;

Όρια (limits)

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & \Rightarrow f(n)\in O(g(n))\\ c>0 & \Rightarrow f(n)\in \Theta(g(n))\\ \infty & \Rightarrow f(n)\in \Omega(g(n))\end{cases}$$

- Γιατί η πιο πάνω «ιδιότητα» είναι πολύ χρήσιμη;

Κανόνας του L'Hospital

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{f'(n)}{g'(n)}$$

Παράδειγμα

- Συγκρίνεται την τάξη αύξησης των $n(n\text{-}1)/2$ και n^2

- Συγκρίνεται την τάξη αύξησης των συναρτήσεων $n!$ και 2^n

Φόρμουλα του Stirling για μεγάλα n: $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$$\lim_{n\to\infty}\frac{n!}{2^n} = \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n} = \lim_{n\to\infty}\sqrt{2\pi n}\left(\frac{n}{2e}\right)^n =$$

$$\Rightarrow 2^n \in O(n!)$$

Συνήθεις Συναρτήσεις

- **Σταθερά** $O(1)$: (constant)
  - Καλύτερη δυνατή περίπτωση, παρόλο που σπάνια θα βρει κάποιος αλγόριθμο αυτής της τάξης. Είναι επιθυμητό η βασική λειτουργία του αλγόριθμου να είναι $O(1)$.
- **Λογαριθμική** $O(\log n)$: (logarithmic)
  - Δεν παίζει ρόλο η βάση του λογαρίθμου. Γιατί;
  - Αποδοτικοί αλγόριθμοι οι οποίοι μοιράζουν το πρόβλημα σε υποπροβλήματα και λύνουν **μόνο ένα μέρος τους**
  - Αλγόριθμοι που δεν «χρησιμοποιούν» όλα τα δεδομένα. Γιατί;
- **Γραμμική** $O(n)$: (linear)
  - Αλγόριθμοι που σαρώνουν τα δεδομένα ένα σταθερό αριθμό φορών (που δεν εξαρτάται από το μέγεθος των δεδομένων).
- **n-log-n** $O(n \log n)$:
  - Αλγόριθμοι του τύπου divide-and-conquer (Αλγόριθμοι που μοιράζουν το πρόβλημα σε υποπροβλημάτα τα οποία λύνου συνήθως αναδρομικά).

Συνήθεις Συναρτήσεις

- **Δευτεροβάθμια** $O(n^2)$: (quadratic)
  - Αλγόριθμοι με δύο ενθυλακωμένους βρόγχους (embedded loops).
  - Π.χ., Απλοί αλγόριθμοι ταξινόμησης
- **Τριτοβάθμια** $O(n^3)$: (cubic)
  - Αλγόριθμοι με τρεις ενθυλακωμένους βρόγχους (embedded loops).
  - Π.χ., αλγόριθμοι γραμμικής άλγεβρας (πολλαπλασιασμός πινάκων)
- **Εκθετική** $O(2^n)$: (exponential)
  - Αλγόριθμοι που ελέγχουν όλα τα δυνατά υποσύνολα του προβλήματος
  - Παρόλο που η βάση της δύναμης αλλάζει τον ρυθμό αύξησης της συνάρτησης, ο όρος «εκθετική» χρησιμοποιείται για όλες τις βάσεις.
  - Δύσκολα προβλήματα!
- **Παραγοντική** $O(n!)$: (factorial)
  - Αλγόριθμοι που ελέγχουν όλους τους πιθανούς συνδυασμούς (permutations).

Μαθηματική Ανάλυση Μη-Αναδρομικών (non-recursive) Αλγορίθμων

- Ορίστε το μέγεθος της εισόδου n (input size)
  - Ποια παράμετρος χαρακτηρίζει τα δεδομένα εισόδου
- Αναγνωρίστε τη βασική λειτουργία (basic operation) του αλγορίθμου
  - Βρίσκεται στον πιο «εσωτερικό» βρόγχο (inner most loop).
  - Είναι $O(1)$.
- Από τι εξαρτάται ο αριθμός φορών που θα εκτελεστεί η βασική λειτουργία;
  - Από το μέγεθος των δεδομένων εισόδου.
  - Διαφέρουν η καλύτερη και χειρότερη περίπτωση εισόδου;
- Βρείτε το άθροισμα που αθροίζει τις φορές που εκτελείται η βασική λειτουργία του αλγόριθμου.
- Υπολογίστε το άθροισμα

Παράδειγμα

```
Search(A[], K)
  i=0
  While (i<n) and (A[i]!= K)
     i=i+1;
   if i<n return i;
   Return -1;
```

- **Καλύτερη Περίπτωση:**
- **Χειρότερη Περίπτωση:**
- **Μέση Περίπτωση:**

Αθροίσματα

Ορισμός: $\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots a_n$

Ιδιότητες:

$$\sum_{i=1}^{n} c a_i = c \sum_{i=1}^{n} a_i \qquad \sum_{i=1}^{n} c\left(a_i \pm b_i\right) = c \sum_{i=1}^{n} a_i \pm c \sum_{i=1}^{n} b_i$$

Συνήθη Αθροίσματα $(k \le n)$:

$$\sum_{i=k}^{n} 1 = 1 + 1 + \cdots + 1 = n - k + 1$$

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n\left(n+1\right)}{2}$$

$$\sum_{i=0}^{n} a^i = 1 + a + a^2 \cdots + a^n = \frac{1 - a^{n+1}}{1 - a}$$

Παράδειγμα

- Γράψτε και αναλύστε ένα απλό αλγόριθμο που να ελέγχει κατά πόσο τα στοιχεία μιας λίστας είναι μοναδικά (δεν υπάρχουν δύο τα ίδια).

```
unique(A[0…n-1])
    for i=0 to n-2
        for j=i+1 to n-1
            if A[i]= A[j] return false;
    return true;
```

- **Καλύτερη Περίπτωση:**
- **Χειρότερη Περίπτωση:**

Παράδειγμα

- Γράψτε και αναλύστε ένα απλό αλγόριθμο που να πολλαπλασιάζει δύο πίνακες *nxn*

```
matrixMulti(A[0…n-1],B[0…n-1])
   for i=0 to n-1
    for j=0 to n-1
      M[i,j]=0;
      for k=0 to n-1
         M[i,j]=M[i,j]+A[1,k]*B[k,j];
```

Μαθηματική Ανάλυση Αναδρομικών (recursive) Αλγορίθμων

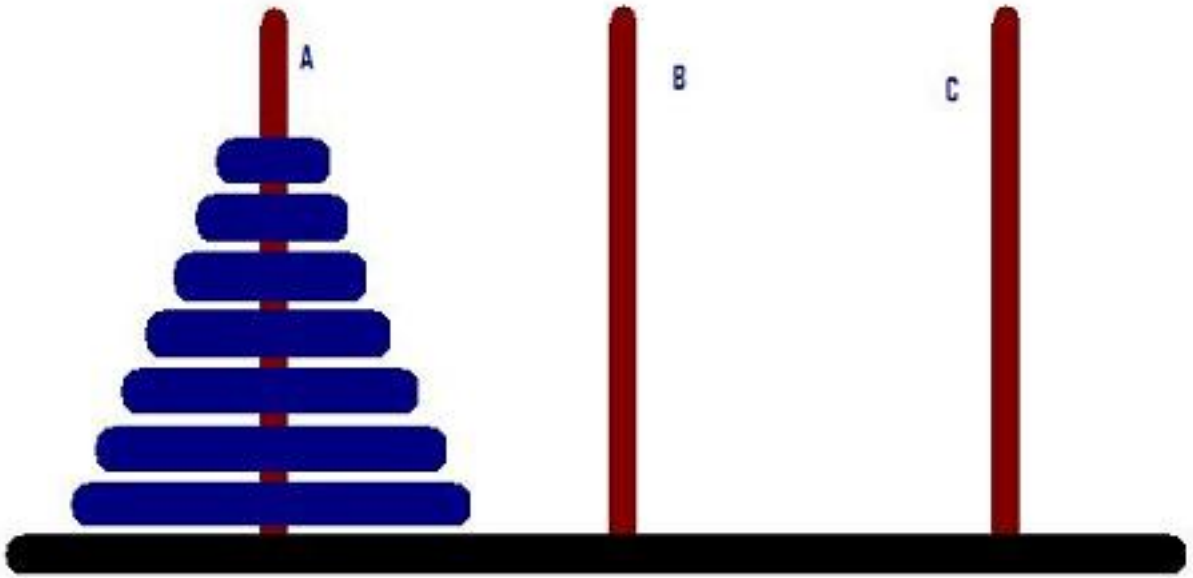
- Αναδρομικοί Αλγόριθμοι
  - Αλγόριθμοι οι οποίοι για να λύσουν ένα πρόβλημα «καλούν» το εαυτό τους με διαφορετικό όρισμα.
- Παράδειγμα
  - Factorial

```
Fact(n)
    if n<=1 return 1;
    return n*Fact(n-1);
```

  - Fibonacci numbers:

```
Fib(n)
    if n=0 return 0;
    if n=1 return 1;
    return Fib(n-1)+Fib(n-2);
```

Οι πύργοι του Hanoi



- Tower of Hanoi: Στόχος του παιγνιδιού είναι να μεταφερθούν (ένας-ένας) όλοι οι δίσκοι από την κολόνα Α στη C, χωρίς ποτέ να τοποθετηθεί μεγαλύτερος δίσκος πάνω από μικρότερο.

Οι Πύργοι του Hanoi

- Αναδρομικός αλγόριθμος
  - Βασική ιδέα:
    - Να μεταφερθούν οι n-1 δίσκοι στην ενδιάμεση κολόνα
    - Να μεταφερθεί ο μεγαλύτερος δίσκος στη σωστή κολόνα
    - Τέλος, να μεταφερθούν οι υπόλοιποι n-1 δίσκοι στην σωστή κολόνα

```
hanoi(n, source, dest, by)
      if (n==1) move disc from source to dest;
      else
          hanoi(n-1, souce, by, dest);
          move disc from source to dest;
          hanoi(n-1, by, dest, source);
```

Μαθηματική Ανάλυση Αναδρομικών (recursive) Αλγορίθμων

- Ορίστε το μέγεθος της εισόδου n (input size)
  - Ποια παράμετρος χαρακτηρίζει τα δεδομένα εισόδου
- Αναγνωρίστε τη βασική λειτουργία (basic operation) του αλγορίθμου
- Από τι εξαρτάται ο αριθμός των φορών που θα εκτελεστεί η βασική λειτουργία;
  - Από το μέγεθος των δεδομένων εισόδου.
  - Διαφέρουν η καλύτερη και χειρότερη περίπτωση εισόδου;
- Βρείτε τη αναδρομική σχέση που χαρακτηρίζει τον αριθμό φορών που εκτελείται η βασική λειτουργία και τη σχετική αρχική κατάσταση (initial condition).
- Λύστε την αναδρομική σχέση η τουλάχιστον βρείτε την τάξη αύξησης της λύσης (HMY320).

Παράδειγμα

```
Fact(n)
    if n<=1 return 1;
    return n*Fact(n-1);
```

- Καλύτερη, Χειρότερη, ή Μέση Περίπτωση;

- Συμβολίστε με $M(n)$ τον αριθμό των πολλαπλασιασμών που θα εκτελεστούν για μια είσοδο n.

- Λύση με τη μέθοδο αντικατάστασης.

Παράδειγμα

Πόσα bits χρειάζονται για να γράψουμε το n σε δυαδική μορφή.

```
Bin(n)
    if n=1 return 1;
    return Bin(floor(n/2))+1;
```

- Συμβολίστε με $A(n)$ τον αριθμό των προσθέσεων που θα εκτελεστούν για μια είσοδο n.
- Υποθέστε πως το n είναι δύναμη του 2, δηλαδή $n=2^k$
- Λύση με τη μέθοδο αντικατάστασης.
- Το αποτέλεσμα ισχύει για όλες τις τιμές του n.

Λύση Δευτεροβάθμιας Ομοιογενούς Αναδρομικής Εξίσωσης

- Υποθέστε την ακόλουθη ομοιογενή δευτεροβάθμια αναδρομικής εξίσωση

$$ax(n)+bx(n-1)+cx(n-2)=0,$$

και δεδομένες κάποιες αρχικές συνθήκες

- Η λύση μπορεί να βρεθεί με μετασχηματισμούς z…
- Βρίσκουμε και λύνουμε τη χαρακτηριστική εξίσωση

- Εάν $\zeta_1 \neq \zeta_2$ πραγματικές,
- Εάν $\zeta_1 = \zeta_2 = \zeta$ πραγματικές,
- Εάν μιγαδικές $\zeta_{1,2} = u \pm iv$ $x(n) = \gamma^n \left[\alpha \cos n\theta + \beta \sin n\theta\right]$

$$\gamma = \sqrt{u^2 + v^2} \qquad \theta = \arctan\frac{v}{u}$$

Παράδειγμα: Fibonacci Numbers

$$F(n) = F(n-1) + F(n-2) \qquad F(0) = 0, F(1) = 1.$$

- Χαρακτηριστική εξίσωση:

Παράδειγμα: Αναδρομικός αλγόριθμος για τον υπολογισμό αριθμών Fibonacci.

```
Fib(n)
    if n=0 return 0;
    if n=1 return 1;
    return Fib(n-1)+Fib(n-2);
```

- Συμβολίστε με $A(n)$ τον αριθμό των προσθέσεων που θα εκτελεστούν για μια είσοδο n

- Υποθέστε $A(n)=B(n)-1$

$$B(n)-B(n-1)-B(n-2)=0 \qquad B(0)=1, B(1)=1.$$

- Τελικά βρίσκουμε ότι

$$A(n)=B(n)-1=\frac{1}{\sqrt{5}}\left(\zeta_1^{n+1}-\zeta_2^{n+1}\right)-1$$

Παράδειγμα: Μη-Αναδρομικός αλγόριθμος για τον υπολογισμό αριθμών Fibonacci.

```
Fib(n)
    F[0]=0;   F[1]=1;
    for i=2 to n
        F[i]=F[i-1]+F[i-2];
    return F[n];
```

Μέγεθος εισόδου n

- Παρόλο που οι αναδρομικοί αλγόριθμοι φαίνονται απλοί, πολλές φορές μπορεί να μην είναι καθόλου αποδοτικοί!
- Τι πήγε «λάθος» στην αναδρομική έκδοση του αλγορίθμου υπολογισμού των αριθμών Fibonacci;

Εμπειρική Ανάλυση Αλγορίθμων

- Κατανόηση του πειράματος
  - Τι προσπαθείτε να δείξετε;
  - Πολλές φορές υπάρχουν τυποποιημένα προβλήματα με τα οποία μπορείτε να συγκρίνεται τα αποτελέσματα σας.
- Μονάδες μέτρησης (χρόνος (ms) ή αριθμός φορών εκτέλεσης βασικών λειτουργιών)
  - Σε συστήματα πολυπρογραμματισμού (multiprogramming) είναι σημαντικό να μετράτε μόνο τον χρόνο κατά τον οποίο το πρόγραμμα σας είχε πρόσβαση στο CPU.
  - Τι συμβαίνει αν ο χρόνος που χρειάζεται ο αλγόριθμος είναι πολύ μικρός;
- Τι χαρακτηριστικά θα πρέπει να έχουν τα δεδομένα εισόδου
  - Καλύτερη, χειρότερη ή μέση περίπτωση.
  - Γεννήτρια τυχαίων αριθμών (random number generator).

Εμπειρική Ανάλυση Αλγορίθμων

- Υλοποίηση του αλγορίθμου
  - Βεβαιωθείτε ότι ο αλγόριθμος και οι συναρτήσεις μετρήσεων υλοποιήθηκαν σωστά.
- Τρέξτε τον αλγόριθμο για διαφορετικές εισόδους και κρατήστε τις μετρήσεις ή άλλα δεδομένα
- Αναλύστε (συνήθως με στατιστικές μεθόδους) τα δεδομένα.

Οπτική Παρακολούθηση (visualization) Αλγορίθμων

- Ψάξτε στο Ίντερνετ για υλοποίηση αλγορίθμων με animation.
  - Στείλτε μου email με τη διεύθυνση κάθε «ενδιαφέρον» αλγορίθμου με animation που βρίσκετε.
- Μπορεί να είναι ιδιαίτερα βοηθητικοί για την καλύτερη κατανόηση του αλγορίθμου.