Εισαγωγή στην Επιστήμη των Υπολογιστών

4ο εξάμηνο Σ.Η.Μ.Μ.Υ. & Σ.Ε.Μ.Φ.Ε.

http://www.corelab.ece.ntua.gr/courses/

1η ενότητα: Εισαγωγή, Αλγόριθμοι

Στάθης Ζάχος
Άρης Παγουρτζής

Επιμέλεια: Πάνος Χείλαρης, Βαγγέλης Μπαμπάς, Γεωργία Καούρη
Ευχαριστίες: Ηλίας Κουτσουπιάς (ΕΚΠΑ)

Περιεχόμενα

Εισαγωγή

- Κλάδοι της Επιστήμης των Υπολογιστών
- Τι είναι Επιστήμη των Υπολογιστών;
- Πρόγραμμα και γλώσσα προγραμματισμού
- Αλγόριθμος
- Υπολογισιμότητα
- Πολυπλοκότητα
- Επανάληψη, επαγωγή, αναδρομή, ορθότητα
- Δομημένος προγραμματισμός
- Παράλληλες, ταυτόχρονες, κατανεμημένες διεργασίες
- Παράδειγμα: πύργοι Hanoi, ταξινόμηση treesort με binary search tree
- Το Θεώρημα τεσσάρων χρωμάτων (four color theorem)

Introduction

- The 21st century has been called the Computer Era. Computers are used not only as a professional tool, but as communication and entertainment media.
- Young students already have a lot of experience in using computers and the Internet. Their natural fascination can be used in order to boost their interest in mathematics and science.
- Algorithmics (algorithmic methodology) is a new way for solving problems.
- Proposal for high school: replace computer literacy with an elementary programming language that prompts students to concentrate on the design and precise formulation of solutions for problems.

Κεντρικό ερώτημα Επιστήμης Υπολογιστών

Τι μπορεί να μηχανοποιηθεί και μάλιστα αποδοτικά ;

Ποια προβλήματα μπορούμε να λύσουμε με υπολογιστή και πόσο καλά ;

Υπολογιστές: ταχύτητα - ακρίβεια

What is Computer Science?

- The scientific and engineering discipline that studies representation, storage and transfer of information by computers and networks, algorithmic solutions and their efficiency, computational complexity.
- What can be automated, and even better automated efficiently.
- 'Computer Science' vs. 'Informatics' vs. 'Computing Science'
- Dijkstra (astronomy - telescope)

Computer Science

Informatics

Computing Science

Dijkstra

Κλάδοι Επιστήμης Υπολογιστών

1. Αλγόριθμοι και Δομές Δεδομένων
2. Γλώσσες Προγραμματισμού και Μεταγλωττιστές
3. Αρχιτεκτονική Υπολογιστών και Δικτύων (hardware)
4. Αριθμητικοί και Συμβολικοί Υπολογισμοί
5. Λειτουργικά - Παράλληλα - Κατανεμημένα Συστήματα
6. Μεθοδολογία - Τεχνολογία Λογισμικού (software)
7. Βάσεις Δεδομένων και Διαχείριση Πληροφοριών
8. Τεχνητή Νοημοσύνη και Ρομποτική
9. Επικοινωνία ανθρώπου - υπολογιστή. Πολυμέσα
10. Δίκτυα Επικοινωνιών - Ευφυή Δίκτυα - Διαδίκτυο

Different aspects of CS

- Networks and Internet.
- Information flow (ordering Chinese food, movie theaters, holidays in Togo, poets of Sakhalin, Pursuit of trivia).
- Business applications (online financial transactions, e-commerce, e-banking, e-anything).
- Modern technology (AI, multimedia, ...).
- Administration (big brother, encryption, security, e-voting, digital warfare).
- Change in every aspect of everyday life.
- Futurology ....

- **Our view. The value of a new mathematical methodology: Algorithmic problem solution.**

Algorithmic problems may arise in:

- Internet (routing, congestion, game theory, cost allocation)
- Biology (protein folding, genome, evolution).

Disorientations

- The question of whether a **computer** can think is no more interesting than the question of whether a **submarine** can swim.
- Computer science is no more about computers than astronomy is about telescopes.

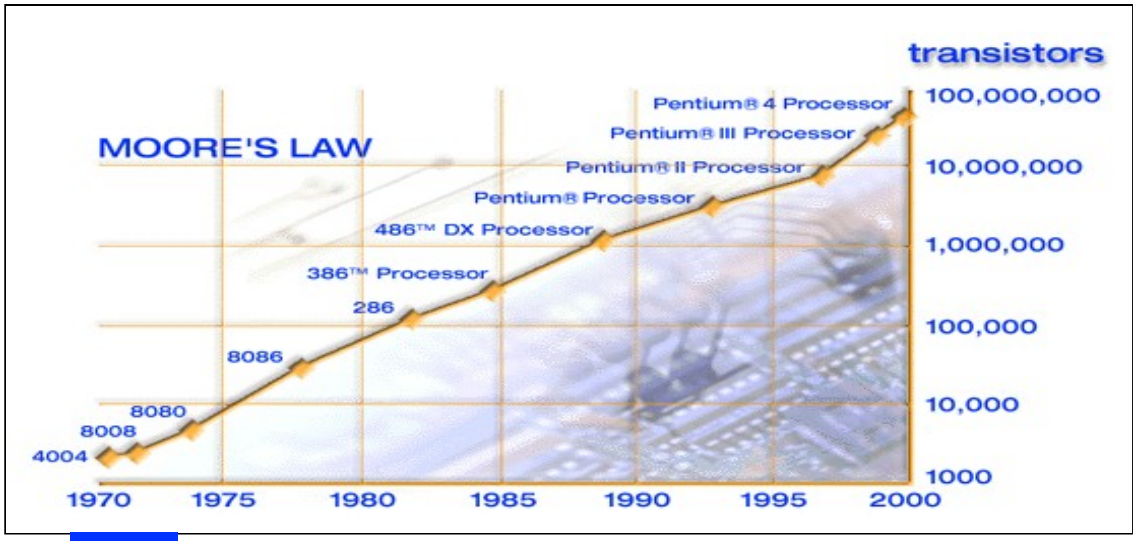
[Dijkstra]

- *The world doesn't need more than five computers*

[Tom Watson, IBM, 1945]

Hardware exponential growth

- Gordon Moore, Intel, 1965
  Hardware density in integrated circuits doubles every 18 months



© intel. http://www.intel.com/research/silicon/mooreslaw.htm

But do not forget:

To err is human.

To mess everything up, you need a computer.

Mathematical formalization of engineering methodology

- Problem requirements
- Specifications
- Design
- Implementation
- Testing - Verification
- Optimization
- Complexity (cost of resources)
- Documentation
- Maintenance

Concepts that had been used by engineers, were formalized in CS, obtained a mathematical form; and thus we can argue about them using proofs.

Program

- Program
  - Precise description of an algorithm in a formal language that is called programming language
  - Actions are applied to data

Formulation in a language

- Natural language
  - No strict syntactic rules
  - Great density and semantic capability
- Formal language
  - Strict syntax and semantics
- Programming language
  - Formal language in which computations can be described
  - Executable by an electronic computer

Knowledge representation

Modeling

Abstraction:

- (Directed) Graphs
- (Formal) Logic

- Data Models
- Data Structures
- Algorithms

Αλγόριθμοι (not in Webster's 50 years ago, in 1971 in Oxford's: erroneous refashioning of algorism: calculation with Arabic numerals)

- Abu Jaffar Mohammed Ibn Musa Al-Khowarizmi, 9ος αι. μ.Χ. محمد بن موسى الخوارزمي,
  - Αυστηρά καθορισμένη (πεπερασμένη) ακολουθία ενεργειών που περιγράφει μέθοδο επίλυσης ενός προβλήματος
  - Οι ενέργειες εφαρμόζονται σε δεδομένα

Παραδείγματα:

- Ευκλείδειος αλγόριθμος (Ευκλείδης, 3ος αι. π.Χ.) για εύρεση ΜΚΔ
- Αριθμοί Fibonacci (Leonardo Pisano Filius Bonacci, 13ος αι. μ.Χ.)
- Τρίγωνο Pascal (Yang Hui, 楊輝 , 13ος αι. μ.Χ.)

Fibonacci numbers

- 1, 1, 2, 3, 5, 8, 13, 21,…
  $F_n=F_{n-1}+F_{n-2}$
- Problem: Given n, compute F_n?
- recursion, iteration, …
- How fast can we compute F_n?

  $O(1.618^n)$, $O(n)$, $O(\log n)$

Pascal Triangle (Yang Hui)

Binomial coefficients / $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$ / Combinations

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

. . . . . .

Algorithmics as a branch of Mathematics

- Mathematics is a methodology for solving problems.

  G. Polya: How to solve it.
- Classically the solution is described statically, e.g., as a solution of an equation.
- In modern 20th century mathematics, the solution can also be dynamic, e.g., a description of an algorithm that produces the answer.
- Furthermore, the cost of solving the problem algorithmically is of interest: Computational complexity.

  D. Harel, W. Feldman. Algorithmics: the spirit of computing

Υπολογισιμότητα - Πολυπλοκότητα

- Υπολογισιμότητα (Computability)
  - Τι μπορεί να υπολογιστεί και τι όχι;
- Υπολογιστική πολυπλοκότητα (Computational Complexity)
  - Τι μπορεί να υπολογιστεί γρήγορα (ή σε λίγο χώρο) και τι όχι;
  - Πόσο γρήγορα μπορεί να υπολογιστεί;

Μη επιλύσιμα προβλήματα – Πρόβλημα τερματισμού

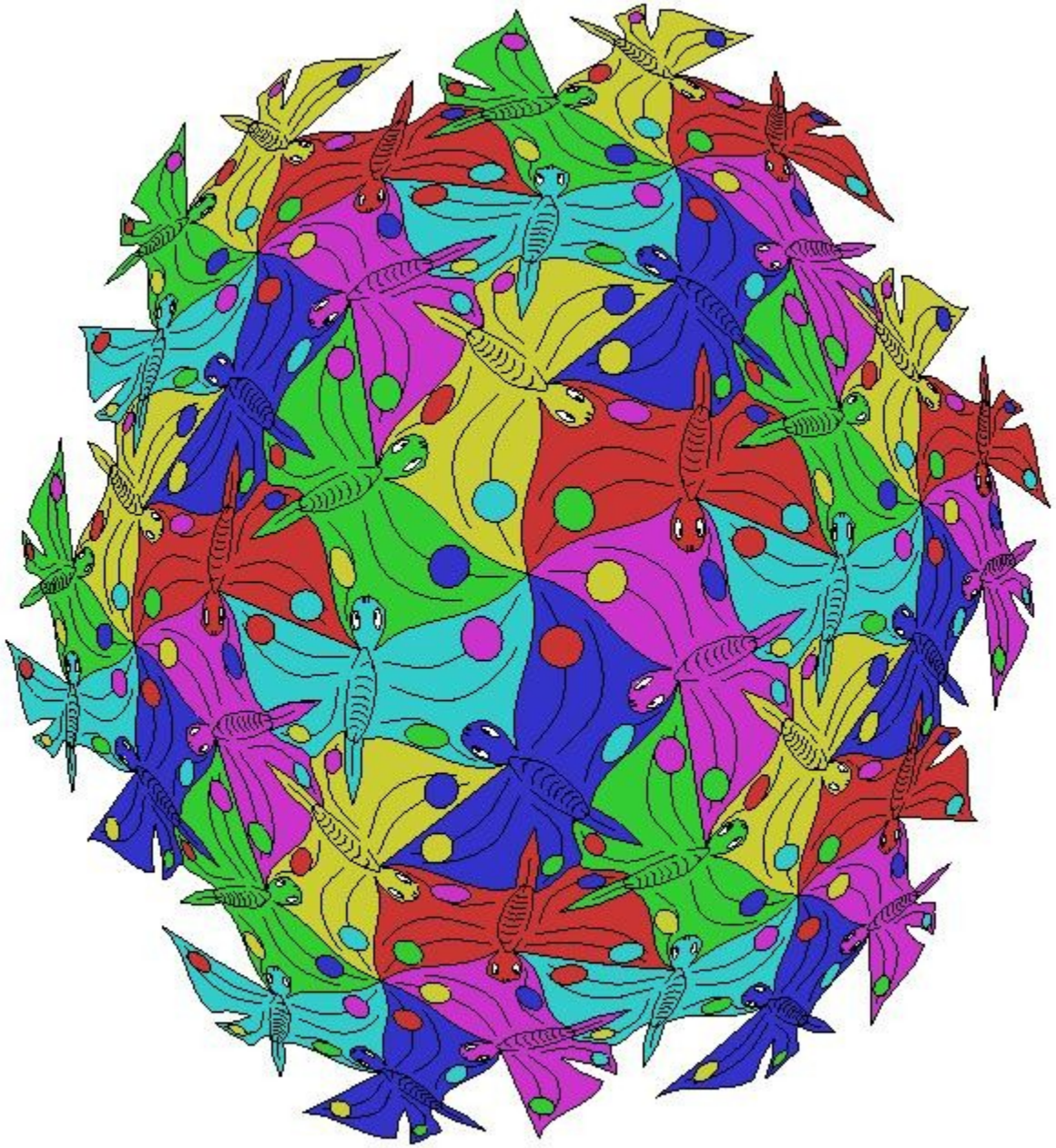
- Το πρόβλημα του Collatz (Ulam):

```
while x!=1 do
  if (x is even) then x=x/2
  else x=3*x+1
```

- 7 → 22 → 11 → 34 → 17 → 52 → 26 → 13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1
- Πρόβλημα: Δίνεται x. Τερματίζει το πρόγραμμα;
- Πρόβλημα: Τερματίζει το πρόβλημα για κάθε φυσικό αριθμό x;
- Δεν γνωρίζουμε την απάντηση (είναι δηλαδή ανοικτά προβλήματα).

Τι είναι πολυπλοκότητα;

- Το `101101011101` είναι πιο πολύπλοκο από το `010101010101` (Kolmogorov complexity)
- Τα θηλαστικά είναι πιο πολύπλοκα από τους ιούς.
- Το σκάκι είναι πιο πολύπλοκο από την τρίλιζα.
- Οι επικαλύψεις του Escher είναι πιο πολύπλοκες από τα πλακάκια του μπάνιου.
- Οι πρώτοι αριθμοί είναι πιο πολύπλοκοι από τους περιττούς (υπολογιστική πολυπλοκότητα).



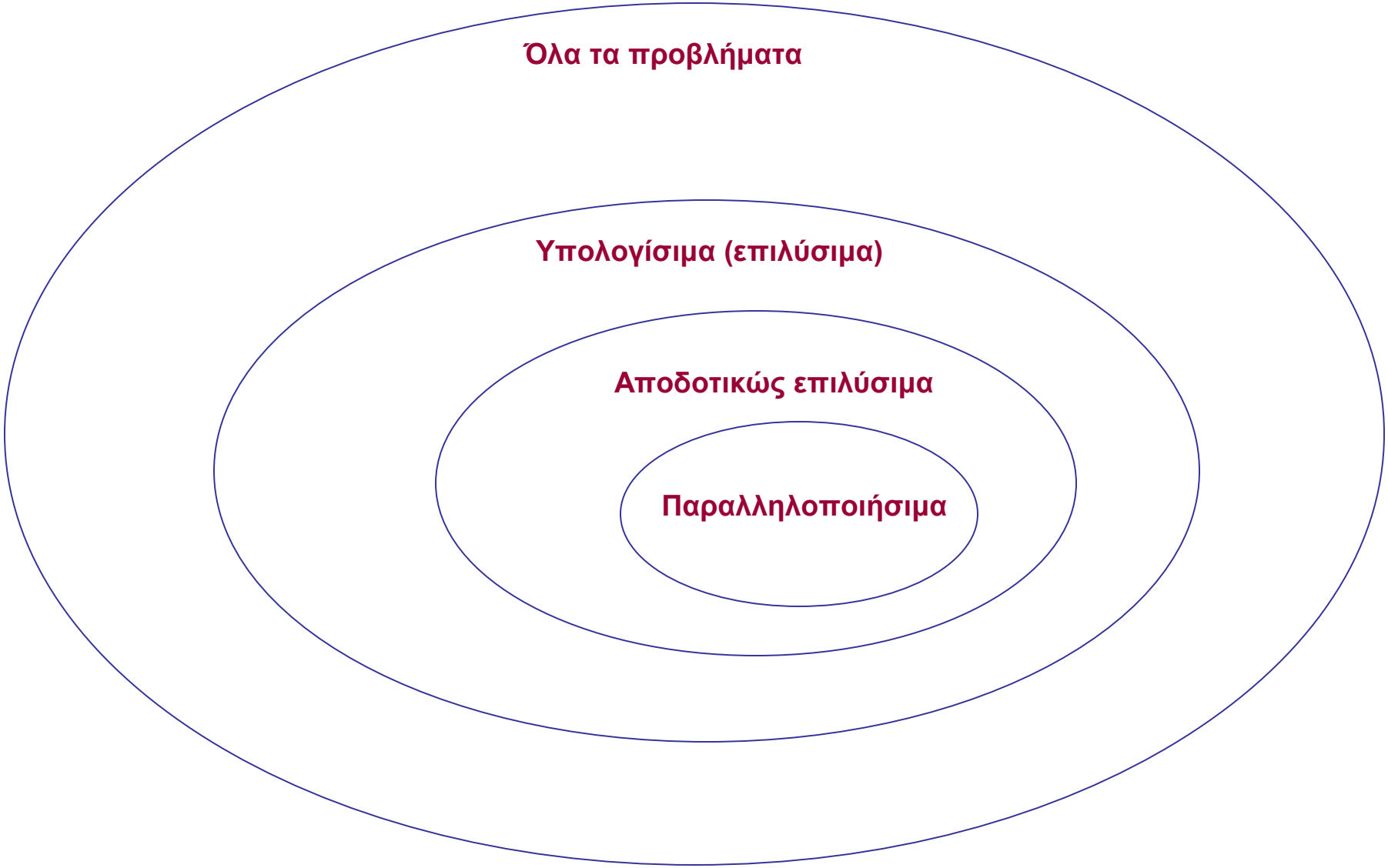
Τι είναι υπολογιστική πολυπλοκότητα;

- Ένας τρόπος για να συλλάβουμε γιατί οι πρώτοι αριθμοί είναι πιο πολύπλοκοι από τους περιττούς είναι η υπολογιστική πολυπλοκότητα.
- Το πρόβλημα **«Δίνεται x. Είναι πρώτος;»** είναι πιο δύσκολο από το πρόβλημα **«Δίνεται x. Είναι περιττός;»**

Computational Complexity

- Efficiently computable, feasible.
- Comparing algorithms with respect to their efficiency.
- Classifying problems with respect to the use of resources, e.g., time (computational steps), space (memory), number of processors, number of non-deterministic choices in every step, use of external help (oracles), use of random bits, etc.

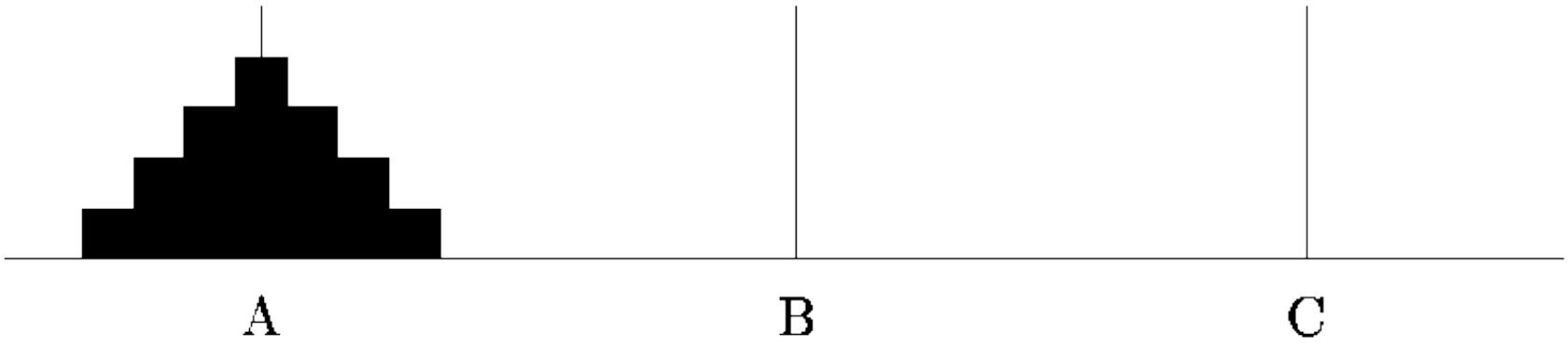
Κατηγοριοποίηση προβλημάτων



Asymptotic behavior

$\log n$	n	n^2	2^n
3.322	10	100	1024
6.644	100	10000	1267650600228229401496703205376
9.966	1000	1000000	$(1267650600228229401496703205376)^{10}$

Iteration-Recursion-Induction

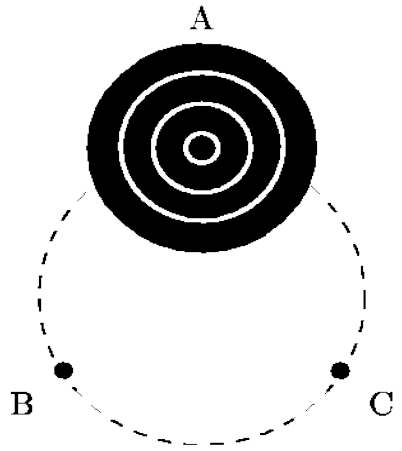


Σχήμα 1.2: Πύργοι του Ανόι $(n = 4)$.

Πύργοι Ανόι (Hanoi Towers)

```
procedure move_anoi(n from X to Y using Z)
begin
    if n = 1 then move top disk from X to Y
    else begin
            move_anoi(n−1 from X to Z using Y);
            move top disk from X to Y;
            move_anoi(n−1 from Z to Y using X)
         end
end
```

1. *μετακίνησε κατά την θετική φορά τον μικρότερο δίσκο·*
2. *κάνε την μοναδική επιτρεπτή κίνηση που δεν αφορά τον μικρότερο δίσκο.*



Μερική και Ολική Ορθότητα

- **Λειτουργική σημασιολογία** (operational semantics). Περιγράφει την υπολογιστική ακολουθία που εκτελείται.
- **Δηλωτική σημασιολογία** (denotational semantics). Ορίζει μόνο τη συνάρτηση εισόδου-εξόδου.
- **Αξιωματική σημασιολογία** (axiomatic semantics). Περιγράφει τις σχετικές ιδιότητες που πρέπει απαραίτητα να ικανοποιούνται από την είσοδο και την έξοδο.

Στην περίπτωση που, αντί για κατηγορηματικό λογισμό και φυσικούς αριθμούς, χρησιμοποιούμε πράξεις και ιδιότητες (αξιώματα) κάποιας άλλης συγκεκριμένης αλγεβρικής δομής η αξιωματική σημασιολογία ονομάζεται συνήθως **αλγεβρική σημασιολογία** (algebraic semantics).

Euclid's idea for finding the GCD of two natural numbers

- If a>b then GCD(a, b) = GCD(a **mod** b, b)
- If a<b then GCD(a, b) = GCD(a, b **mod** a)

a mod b = the remainder of the integer division a div b

This is the best known algorithm for GCD.

It is an open problem whether this is an optimal algorithm.

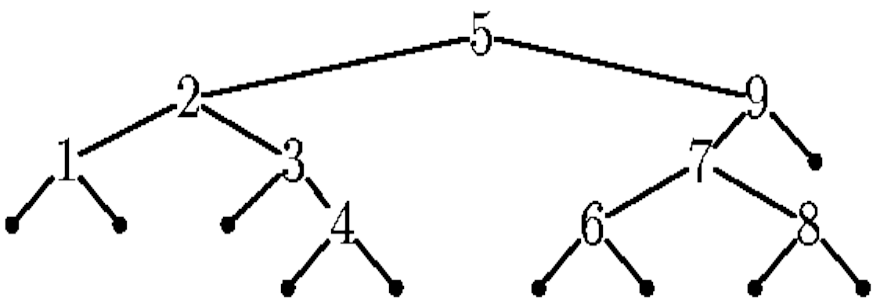
Euclid's algorithm for GCD

GCD of 172 and 54. (iteration)

10	54
10	4
2	4
2	0

The GCD is 2.

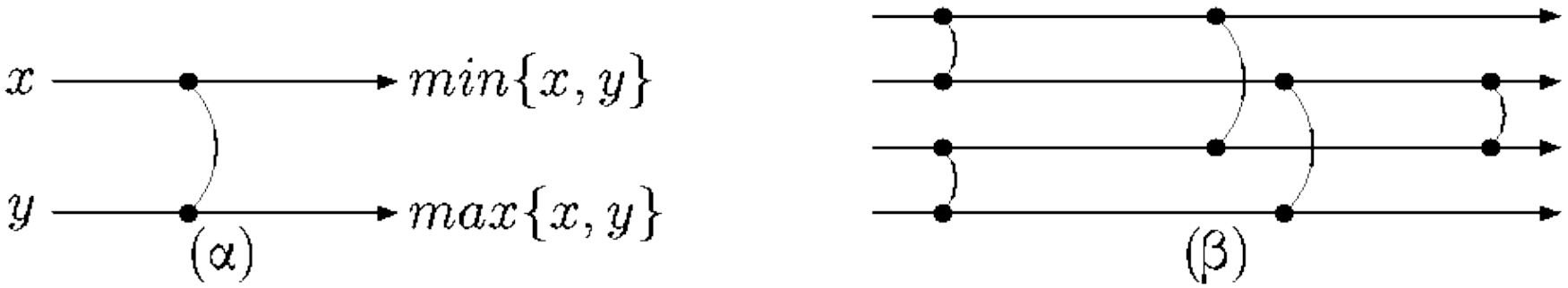
Treesort με χρήση Binary Search Tree




```
procedure inorder(t: treenode)
begin
    if t is not empty then
    begin
        inorder(left branch of t);
        write(element at t);
        inorder(right branch of t)
    end
end
```

- Structured Programming
- Modularity
- Parallel Systems
- Concurrent Systems
- Distributed Systems

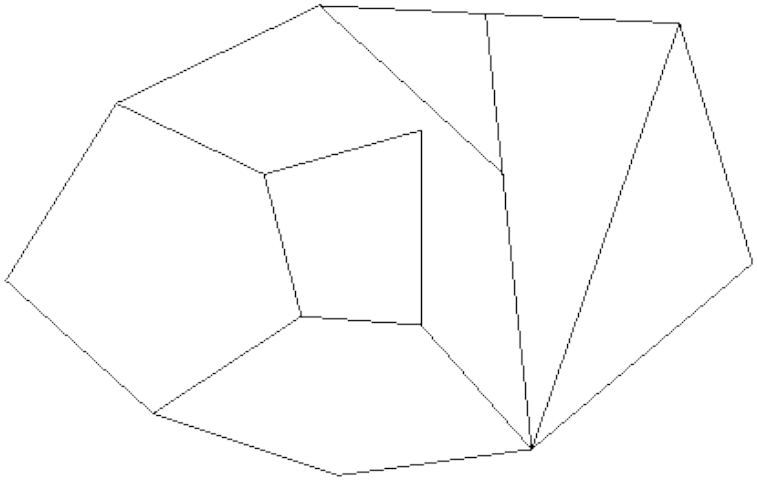
Δίκτυα Ταξινόμησης (Sorting Networks)



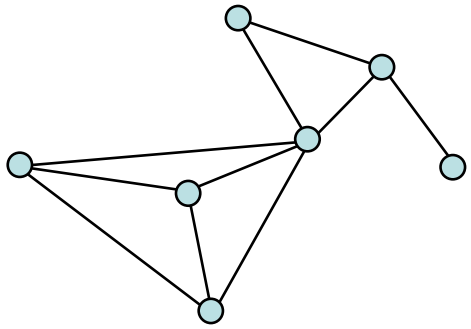
Σχήμα 1.4: (α) Συγκριτής (β) Δίκτυο ταξινόμησης 4 εισόδων

Four Color Theorem (1852-1977)

Πόσα χρώματα απαιτούνται για τον χρωματισμό όλων των χωρών, ούτως ώστε χώρες που συνορεύουν (δηλαδή έχουν γραμμή, όχι απλώς σημείο για κοινό σύνορο) να έχουν διαφορετικό χρώμα;



Σχήμα 1.6: Επίπεδος χάρτης



Appel

Haken

(απόδειξη με πρόγραμμα!)

Αλγόριθμοι

Η έννοια αλγόριθμος είναι πρωταρχική έννοια της θεωρίας αυτής. Γι' αυτό δεν ορίζεται. Εδώ δίνουμε μία άτυπη εξήγηση.

Αλγόριθμος είναι ένα πεπερασμένο σύνολο κανόνων, οι οποίοι περιγράφουν μία μέθοδο (που αποτελείται από μία σειρά υπολογιστικών διεργασιών) για να λυθεί ένα συγκεκριμένο πρόβλημα. Τα αντικείμενα πάνω στα οποία επενεργούν αυτές οι διεργασίες λέγονται **δεδομένα** (*data*).

Ο αλγόριθμος χαρακτηρίζεται από τα παρακάτω πέντε στοιχεία:

- Κάθε εκτέλεση είναι *πεπερασμένη*, δηλαδή τελειώνει ύστερα από έναν πεπερασμένο αριθμό διεργασιών ή βημάτων (*finiteness*).
- Κάθε κανόνας του ορίζεται επακριβώς και η αντίστοιχη διεργασία είναι συγκεκριμένη (*definiteness*).
- Έχει μηδέν ή περισσότερα μεγέθη *εισόδου* που δίδονται εξαρχής, πριν αρχίσει να εκτελείται ο αλγόριθμος (*input*).
- Δίδει τουλάχιστον ένα μέγεθος σαν αποτέλεσμα (*έξοδο-output*) που εξαρτάται κατά κάποιο τρόπο απ'τις αρχικές εισόδους.
- Είναι *μηχανιστικά* αποτελεσματικός, δηλαδή όλες οι διαδικασίες που περιλαμβάνει μπορούν να πραγματοποιηθούν με ακρίβεια και σε πεπερασμένο χρόνο «με μολύβι και χαρτί» (*effectiveness*).

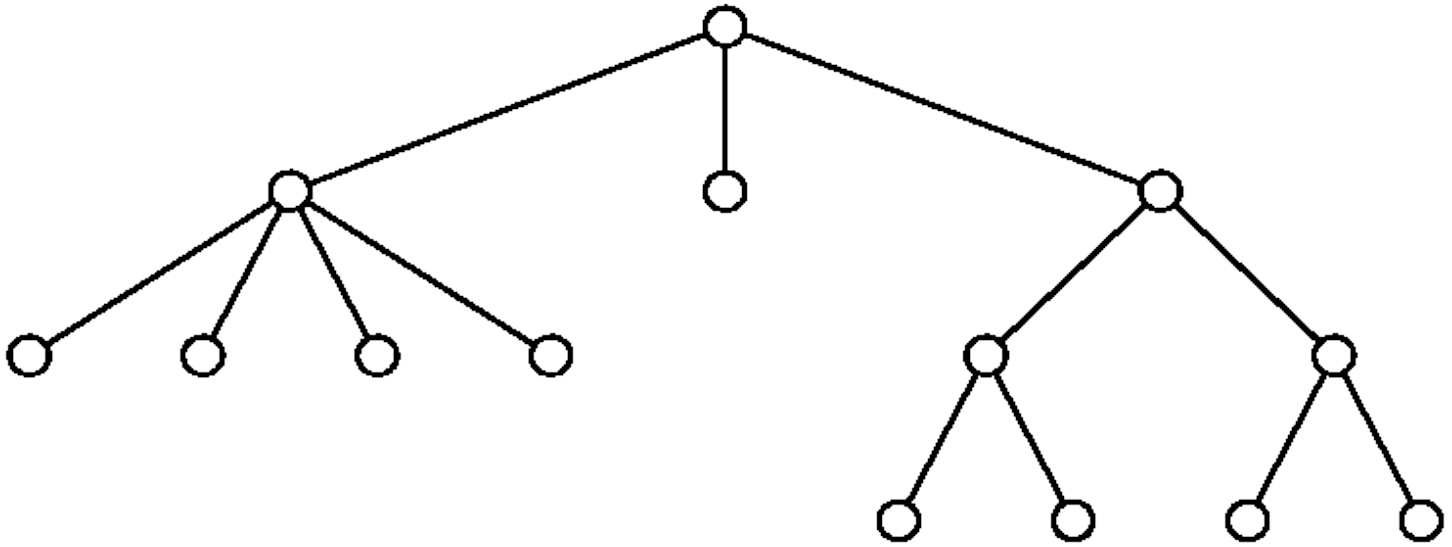
Determinism

Ένας αλγόριθμος είναι **ντετερμινιστικός** (*deterministic*) ή **μη ντετερμινιστικός** (*nondeterministic*). Ο ντετερμινιστικός αλγόριθμος διακρίνεται από τα παρακάτω στοιχεία:

- Ο υπολογισμός που προτείνει είναι γραμμικός. Για κάθε υπολογιστική διαμόρφωση (configuration) υπάρχει ακριβώς μία νόμιμη επόμενη διαμόρφωση.
- Η υπολογιστική διαδικασία προχωρεί βήμα προς βήμα και είναι σε θέση να σταματήσει για οποιαδήποτε δυνατή είσοδο.

Σχήμα 2.1: Ντετερμινιστικός αλγόριθμος

Nondeterminism



Σχήμα 2.2: Μη ντετερμινιστικός αλγόριθμος

Μοντέλα Υπολογισμού

Θέλοντας να τυποποιήσουμε δηλαδή να ορίσουμε αυστηρά την έννοια του αλγορίθμου, είναι απαραίτητο να ορίσουμε ένα συγκεκριμένο υπολογιστικό μοντέλο. Πολλοί επιστήμονες, όπως οι *A. Turing, A. Church, S. Kleene, E. Post, R. Markov* κ.α., ασχολήθηκαν με το θέμα αυτό και όρισαν διάφορα υπολογιστικά μοντέλα.

Το υπολογιστικό μοντέλο που αντιστοιχεί στον πιο «φυσικό» και διαισθητικό ορισμό του αλγορίθμου είναι η **μηχανή Turing**. Σύμφωνα με την αξιωματική **«θέση του Church»**:

> *«Κάθε αλγόριθμος μπορεί να περιγραφεί με τη βοήθεια μιας μηχανής Turing»*

Θέση των Church-Turing (ισοδύναμη διατύπωση)

«Όλα τα γνωστά και άγνωστα υπολογιστικά μοντέλα είναι μηχανιστικά ισοδύναμα»

δηλαδή:

«Για μια συγκεκριμένη συνάρτηση f, δοθέντος ενός αλγορίθμου σ'ένα υπολογιστικό μοντέλο μπορούμε με τη βοήθεια μηχανής (ή προγράμματος: compiler) να κατασκευάσουμε, για την ίδια συνάρτηση f, αλγόριθμο σ'ένα άλλο υπολογιστικό μοντέλο».

Πολυπλοκότητα

Στην πράξη, το ενδιαφέρον δεν σταματά στο να βρεθεί ένας αλγόριθμος που επιλύει ένα πρόβλημα, αλλά προχωρά στη μελέτη των μετρήσιμων ιδιοτήτων που χαρακτηρίζουν την **αποδοτικότητα** μιας υπολογιστικής μεθόδου. Αυτά τα μεγέθη (αγαθά-*resources*) είναι π.χ. ο χρόνος υπολογισμού, ο χώρος σε μνήμη υπολογιστή, ο αριθμός προκαταρκτικών διαδικασιών που προαπαιτούνται και είναι αυτά που ορίζουν την **πολυπλοκότητα** (*complexity*) του αλγορίθμου. Ονομάζουμε **πολυπλοκότητα ενός προβλήματος** την πολυπλοκότητα ενός **βέλτιστου** (*optimal*) αλγορίθμου που λύνει το πρόβλημα.

Ο τρόπος που προσεγγίζει κανείς την πολυπλοκότητα οδηγεί σ'έναν αρχικό διαχωρισμό της έννοιας **πολυπλοκότητα**, σε **συγκεκριμένη** (*concrete*) πολυπλοκότητα και **μη συγκεκριμένη, περισσότερο θεωρητική** (*abstract*) **πολυπλοκότητα**.

Ο κλάδος της συγκεκριμένης (*concrete*) πολυπλοκότητας ασχολείται με την περιγραφή συστηματικών τεχνικών αξιολόγησης των μετρήσιμων αγαθών (*resources*) που χαρακτηρίζουν την αποδοτικότητα ενός συγκεκριμένου αλγορίθμου (κυρίως του χρόνου και του χώρου που απαιτούνται απ'τον αλγόριθμο) σ'ένα συγκεκριμένο **υπολογιστικό μοντέλο**.

Είδη πολυπλοκότητας

Η συμπεριφορά του αλγορίθμου μελετάται κυρίως σε δύο περιπτώσεις. Στην **χειρότερη** (*worst case*) και στην **μέση** (*average case*), μιας δεδομένης κατανομής πιθανών **στιγμιοτύπων** (*instances*) του προβλήματος. Μια άλλη ανάλυση ενδιαφέρεται για την **μακροπρόθεσμη απόσβεση** (*amortization*) επαναληπτικής χρήσης ενός αλγορίθμου. Η μελέτη της πολυπλοκότητας ενός αλγορίθμου μας επιτρέπει πολλές φορές να αποφανθούμε αν αυτός είναι **βέλτιστος** (*optimal*) για το συγκεκριμένο πρόβλημα. Αυτό προϋποθέτει ότι έχουμε **τα άνω (με αλγόριθμο) και κάτω (με απόδειξη) φράγματα του χρόνου** (ή και **του χώρου**) που επαρκούν και απαιτούνται για την επίλυση ενός προβλήματος και επίσης προϋποθέτει ότι αυτά ταυτίζονται.

Πολυπλοκότητα: κόστος αλγορίθμου / κόστος προβλήματος

Το κόστος ενός αλγορίθμου ορίζεται με τη βοήθεια της παρακάτω συνάρτησης:

$$\text{κόστος αλγορίθμου}(n) = \max_{\substack{\text{για όλες τις δυνατές εισόδους μεγέθους } n}} \{\text{κόστος αλγορίθμου για είσοδο } x\}$$

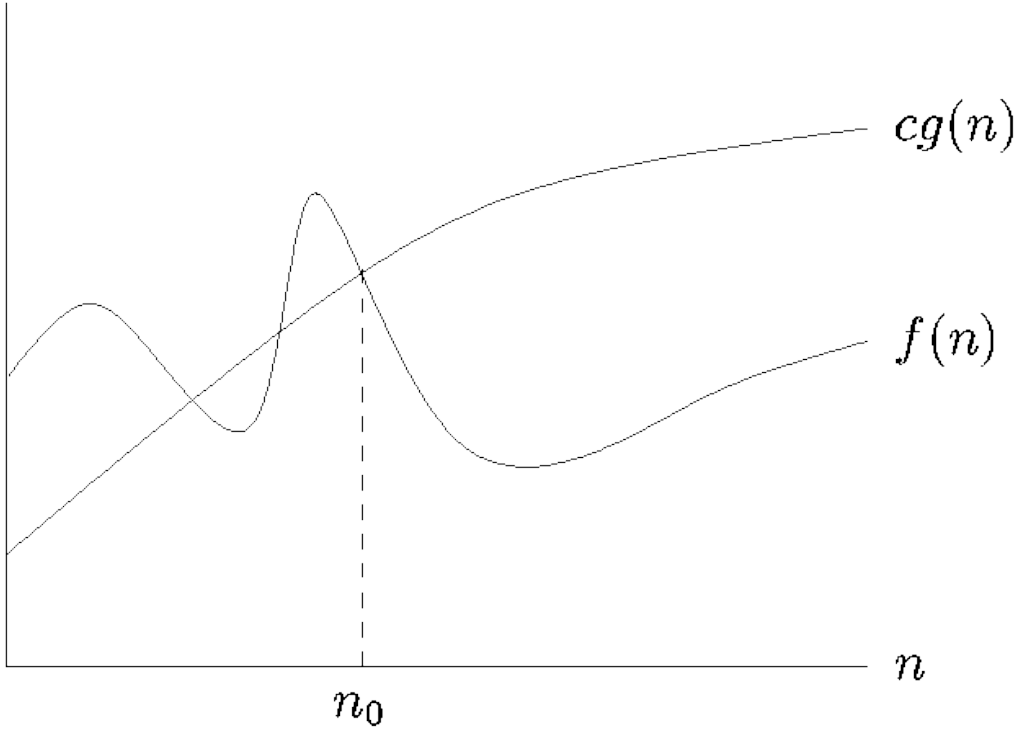
Και το κόστος ενός προβλήματος, με τη βοήθεια της συνάρτησης:

$$\text{κόστος προβλήματος } (n) = \min_{\substack{\text{για όλους τους αλγόριθμους A που επιλύουν το πρόβλημα}}} \{\text{κόστος του αλγορίθμου } A(n)\}$$

Παράδειγμα: ταξινόμηση

- Bubblesort: $c \cdot n^2$
- Mergesort: $c' \cdot n \log n$
- Πολυπλοκότητα του προβλήματος (επίλυση με συγκρίσεις): $\leq c' \cdot n \cdot \log n$

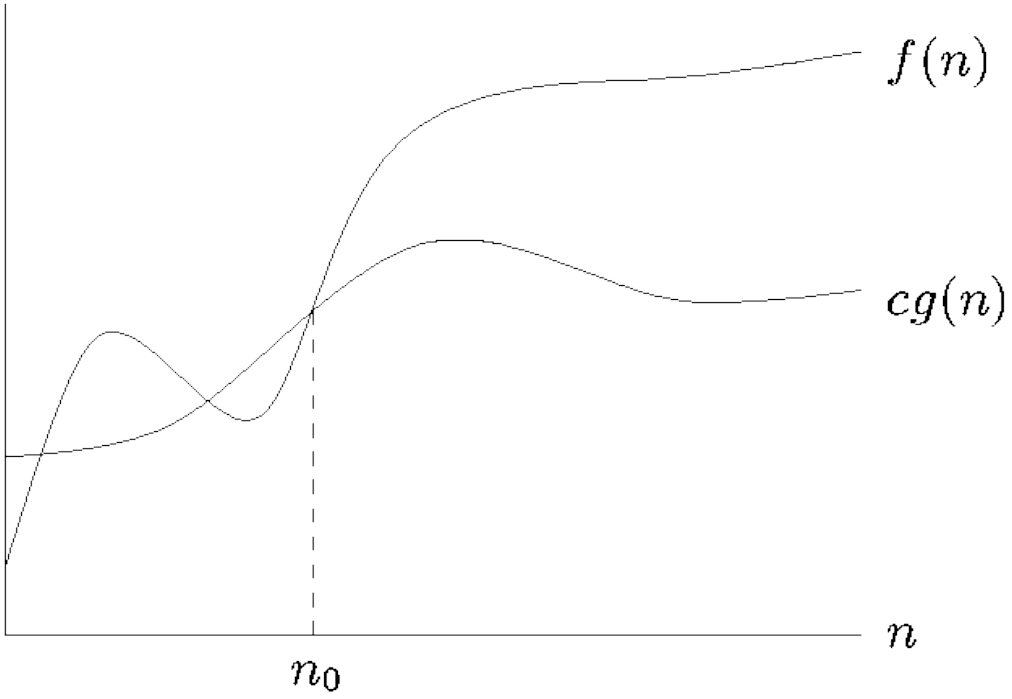
Μαθηματικοί Συμβολισμοί



Σχήμα 2.3: $f = O(g)$

$$O(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \quad f(n) \leq cg(n)\}$$

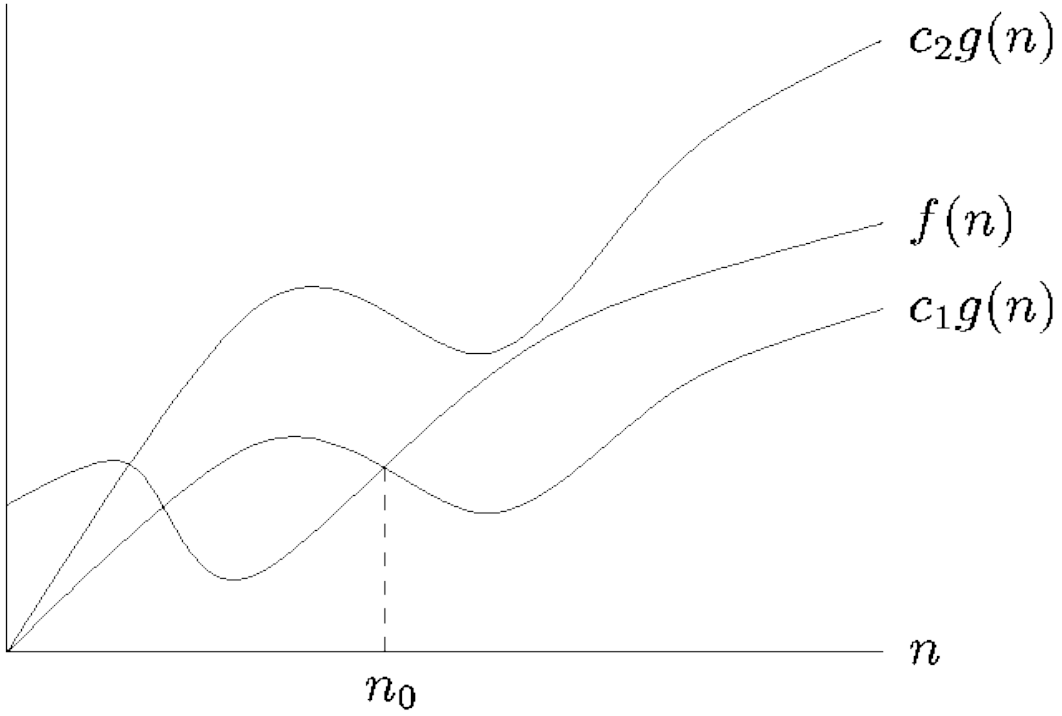
Μαθηματικοί Συμβολισμοί



Σχήμα 2.4: $f = \Omega(g)$

$$\Omega(g) = \{f \mid \exists c > 0,\ \exists n_0 : \forall n > n_0 \quad f(n) \geq cg(n)\}$$

Μαθηματικοί Συμβολισμοί



Σχήμα 2.5: $f = \Theta(g)$

$$\Theta(g) = \{f \mid \exists c_1 > 0,\ \exists c_2 > 0,\ \exists n_0 : \forall n > n_0\ \ c_1 \leq \frac{f(n)}{g(n)} \leq c_2\}$$

Η "προπαίδεια"

Ισχύουν: $\Theta(f) = O(f) \bigcap \Omega(f)$ και $f \in \Theta(g) \iff (f \in O(g)$ και $g \in O(f))$.

Αν $p = c_k n^k + c_{k-1} n^{k-1} + \cdots + c_0$, δηλαδή πολυώνυμο βαθμού k, τότε $p \in \mathrm{O}(n^k)$ ή $p(n) = \mathrm{O}(n^k)$. Επίσης $p \in \Omega(n^k)$ ή $p(n) = \Omega(n^k)$. Συνεπώς $p(n) = \Theta(n^k)$.

Ορίζουμε $\mathrm{O}(poly) = \bigcup \mathrm{O}(n^k)$.

Γενικά ισχύει ότι: $O(1) < O(\alpha(n)) < O(\log^* n) < O(\log(n)) < O(\sqrt{n}) < O(n) < O(n \log(n)) < O(n^2) < \ldots < O(poly) < O(2^n) < O(n!) < O(n^n) < O(A(n))$

Σημείωση: γράφουμε «<» αντί «⊂».

log*n: πόσες φορές πρέπει να λογαριθμήσουμε το n για να φτάσουμε κάτω από το 1 (αντίστροφη υπερεκθετικής)

A: Ackermann.

α: αντίστροφη της A.

Εύρεση Μέγιστου Κοινού Διαιρέτη (gcd)

Δεν είναι καλή ιδέα να το αναγάγουμε στο πρόβλημα εύρεσης πρώτων παραγόντων γιατί αυτό δεν λύνεται αποδοτικά.

Απλός αλγόριθμος: *O(*min*(a,b))*

$$z := min(a, b);$$
$$\textbf{while } (a \textbf{ mod } z \neq 0) \textbf{ or } (a \textbf{ mod } z \neq 0) \textbf{ do } z := z - 1;$$

Αλγόριθμος με αφαιρέσεις: *O(*max*(a,b))*

$$i := a;\ j := b;$$
$$\textbf{while } i \neq j \textbf{ do if } i > j \textbf{ then } i := i - j \textbf{ else } j := j - i;$$
$$\textbf{return } (i)$$

Αλγόριθμος του Ευκλείδη: *O(*log min*(a,b))*

$$i := a;\ j := b;$$
$$\textbf{while } (i > 0) \textbf{ and } (j > 0) \textbf{ do}$$
$$\quad \textbf{if } i > j \textbf{ then } i := i \textbf{ mod } j \textbf{ else } j := j \textbf{ mod } i;$$
$$\textbf{return } (i + j)$$

Πολυπλοκότητα Ευκλείδειου Αλγόριθμου

- Άνω φράγμα: $O(\log \min(a,b))$
  Επειδή σε 2 επαναλήψεις ο μεγαλύτερος αριθμός υποδιπλασιάζεται (άσκηση: αποδείξτε το!)
- Κάτω φράγμα: $\Omega(\log \min(a,b))$
  Επειδή για διαδοχικούς Fibonacci, F_{k+1}, F_k, χρειάζεται k βήματα, και $F_k \approx \varphi^k/\sqrt{5}$, όπου $\varphi=(1+\sqrt{5})/2$ (χρυσή τομή).
- Επομένως: $\Theta(\log \min(a,b))$

Ύψωση σε δύναμη

```
power(a, n)
    result := 1
    for i := 1 to n do
        result := result*a
    return result
```

Πολυπλοκότητα: $O(n)$

– εκθετική ως προς το μήκος της εισόδου!

Ύψωση σε δύναμη με *επαναλαμβανόμενο τετραγωνισμό*

```
fastpower(a, n)
    result := 1
    while n>0 do
        if odd(n) then result:=result*a
        n := n div 2
        a := a*a
    return result
```

Ιδέα: $a^{13} = a^{1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0}$

Πολυπλοκότητα: $O(\log n)$

– πολυωνυμική ως προς το μήκος της εισόδου!

Αριθμοί Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

$F_0 = 0,\ F_1 = 1,$

$F_n = F_{n-1} + F_{n-2},\ n >= 2$

Πρόβλημα: Δίνεται n, να υπολογιστεί το F_n

Πόσο γρήγορο μπορεί να είναι το πρόγραμμά μας;

Αριθμοί Fibonacci: αναδρομικός αλγόριθμος

- ```
  F(n)
      if (n<2) then return n
      else return F(n-1)+F(n-2);
  ```

- *Πολυπλοκότητα*: $T(n) = T(n-1) + T(n-2) + c$, δηλ. η $T(n)$ ορίζεται όπως η $F(n)$ (+ κάτι μικρό), οπότε:

$$T(n) > F(n) = \Omega(1.62^n)$$
  ```

Αριθμοί Fibonacci: καλύτερος αλγόριθμος

- ```
  F(n)
      a:=0; b:=1;
      for i:=2 to n do
          c:=b; b:=a+b; a:=c;
      return b;
  ```

- Πολυπλοκότητα: $O(n)$
  ```

Χρόνος εκτέλεσης αλγορίθμων

- Θεωρήστε 4 προγράμματα με αριθμό βημάτων $O(2^n)$, $O(n^2)$, $O(n)$, και $O(\log n)$ που το κάθε χρειάζεται 1 δευτερόλεπτο για να υπολογίσει το F(100).
- Πόσα δευτερόλεπτα θα χρειαστούν για να υπολογίσουν το F(n);

	$c \cdot 2^n$	$c \cdot n^2$	$c \cdot n$	$c \cdot \log n$
F(100)	1	1	1	1
F(101)	2	1.02	1.01	1.002
F(110)	1024	1.21	1.1	1.02
F(200)	??????	4	2	1.15

Αριθμοί Fibonacci: ακόμα καλύτερος αλγόριθμος

Μπορούμε να γράψουμε τον υπολογισμό σε μορφή πινάκων:

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

Από αυτό συμπεραίνουμε

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Ο αριθμός των **αριθμητικών πράξεων** μειώνεται στο $O(\log n)$.

Προβλήματα πρώτων αριθμών

- Primality testing: Δίνεται ακέραιος *n*. Είναι πρώτος;
  - Σχετικά εύκολο. Επιλύεται σε πολυωνυμικό χρόνο όπως έδειξαν πρόσφατα κάποιο προπτυχιακοί Ινδοί φοιτητές.
- Factoring: Δίνεται ακέραιος *n*. Να βρεθούν οι πρώτοι παράγοντες του.
  - Δεν ξέρουμε αν είναι εύκολο ή δύσκολο. Πιστεύουμε ότι δε λύνεται σε πολυωνυμικό χρόνο, αλλά ούτε ότι είναι τόσο δύσκολο όσο τα NP-complete προβλήματα.
  - Σε κβαντικούς υπολογιστές (που δεν έχουμε ακόμα καταφέρει να κατασκευάσουμε) επιλύεται πολυωνυμικά.

Factoring και κρυπτογραφία (η δικαίωση του Ευκλείδη!)

- RSA: Κρυπτογραφικό σχήμα δημοσίου κλειδιού για να στείλει η A (Alice) στον B (Bob) ένα μήνυμα m.
- Ο B διαλέγει 2 μεγάλους πρώτους αριθμούς p και q, υπολογίζει το γινόμενο n=pq, και διαλέγει ακέραιο e σχετικά πρώτο με το φ(n)=(p-1)(q-1).
- Ο B στέλνει στην A τα n και e.
- Η A στέλνει στον B τον αριθμό $c=m^e \pmod n$.
- Ο B υπολογίζει το m: $m=c^d \pmod n$, όπου το $d=e^{-1} \pmod{(p-1)(q-1)}$.
- Παράδειγμα: p=11, q=17, n=187, e=21, d=61, m=42, c=9

- Η ασφάλεια του RSA στηρίζεται στην (εκτιμώμενη) δυσκολία του factoring.

- Για την υλοποίηση του RSA χρησιμοποιούνται, μεταξύ άλλων, ο αλγόριθμος επαναλαμβανόμενου τετραγωνισμού και ο επεκτεταμένος Ευκλείδειος αλγόριθμος (που επιπλέον εκφράζει τον gcd(a,b) σαν γραμμικό συνδυασμό των a και b).

Πολυπλοκότητα: ανοικτά ερωτήματα

- Εκτός από κάποιες ειδικές περιπτώσεις, για κανένα πρόβλημα δεν γνωρίζουμε πόσο γρήγορα μπορεί να λυθεί.
- Ακόμα και για τον πολλαπλασιασμό αριθμών δεν γνωρίζουμε τον ταχύτερο αλγόριθμο.
- Ο σχολικός τρόπος πολλαπλασιασμού αριθμών με n ψηφία χρειάζεται $O(n^2)$ βήματα.
- Υπάρχουν καλύτεροι αλγόριθμοι που χρειάζονται περίπου $O(n \log n)$ βήματα.
- Υπάρχει αλγόριθμος που χρειάζεται μόνο $O(n)$ βήματα; Αυτό είναι ανοικτό ερώτημα.