

**ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ**



ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Special Topics on Algorithms

Introduction

Vangelis Markakis

Special Topics on Algorithms

- A continuation of the Algorithms course
- Emphasis on topics not covered during the Algorithms course and also on some more modern topics and applications
- You can take this course during your 3rd year or later
- Prerequisites:
 - You have **passed** the Algorithms course
 - You **liked** the Algorithms course

Content – Topics to be covered

- Introduction
 - Some basic concepts
 - Distinction between polynomial, pseudopolynomial and exponential time algorithms
- Problems on numbers
 - Exponentiation/Fibonacci/Euclid's Algorithm for GCD
 - Modular arithmetic, prime numbers, primality testing
 - Applications: public key cryptosystems, RSA and digital signatures

Content – Topics to be covered

- Average case analysis
 - Sorting: Insertionsort, Quicksort
 - Binary Search Trees, hashing
- Coping with NP-completeness – Approximation algorithms
 - Greedy and other combinatorial algorithms
 - Vertex Cover, Set Cover, TSP
 - Knapsack, Job Scheduling, Bin Packing
- Flows and Matchings
 - Algorithms for the Maximum Flow in a network graph and the Maximum Matching in bipartite graphs.

Content – Topics to be covered

- Randomized Algorithms
 - The power and limitations of allowing random decisions
 - Intersects most other sections
 - Primality testing, Max Cut, Min Cut, Max k-SAT
- Linear and Integer Programming
 - Modeling problems as linear or integer programs
 - Applications and LP based Approximation Algorithms
 - LP duality (if time permits)
- Local search algorithms
 - Basic local search, tabu search, application to MAX CUT

Content – Topics to be covered

Further topics – if enough time

- Pattern matching and string processing
 - The Knuth-Morris-Pratt algorithm
- Invited lectures
 - We may have 1 or 2 lectures by other faculty members and collaborators on some application areas

Bibliography

- [DPV] S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani :
“Algorithms”
- [CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein:
“Introduction to Algorithms”
- [KT] J. Kleinberg, E. Tardos: “Algorithm Design”
- ...
- **and many resources on the WWW**

Communication

- Office hours:
 - Mondays: 13:00 – 14:00
 - Fridays: 14:00 – 15:00
- You can always email me regarding questions
 - If I do not reply within 3 days, send it again
- Eclass: Ειδικά Θέματα Αλγορίθμων
 - Please check the announcements there at least once per week

Tutorials

- Teaching Assistant: Panagiotis Tsamopoulos
- Office hours for the TA to be announced soon
- Tutorials start next week

Grading

Final exam **75%**

Midterm exam **20%**

Individual Assignments (x2) **15%**

Note: The midterm is used only if it helps your final grade, otherwise the final exam will count as 95%

Final grade = $\max\{0.95 * \text{final}, 0.75 * \text{final} + 0.2 * \text{midterm}\} + \text{assignments}$

**Introductory concepts:
Polynomial, Pseudo-Polynomial and
Exponential Algorithms**

What are we interested in?

Problems to be solved by a machine: precisely defined; no ambiguities

- We want **to transform appropriately** the input data (problem instances) to output data
- Problems can be classified as **decision** (output = YES/NO), **search** (find an object with desired properties or calculate some quantity) or **optimization** (optimize a given objective function) problems.

COMPUTATIONAL PROBLEM

A problem where we are given **input** instances and some computational question and we want to find an answer/**output**:

Example: given a graph, find the set of vertices of odd degree or compute a set of k vertices where every pair of them is connected by an edge.

Examples of Problems

EXP(onentiation)

I: positive integers a, n

Q: **calculate** a^n

FIBONACCI NUMBERS

I: a positive integer n

Q: **calculate** the n -th Fibonacci number F_n

SUBSET SUM

I: a set $S = \{a_1, a_2, \dots, a_n\}$ of n positive integers and an integer B

Q: **is there** a subset $A \subseteq S$ s. t. $\sum_{i \in A} a_i = B$?

SAT(isfiability)

I: a boolean formula ϕ

Q: **Is ϕ satisfiable** ?

(is there a value assignment to its variables making ϕ TRUE ?
= truth assignment)

Algorithms

Three crucial questions about any **algorithm** for any **problem**:

1. Is it correct ?

- Does it always terminate?
- Does it give a correct answer for any instance of the problem ?

2. How much time/space does it take, as a function of its input?

- “time” = number of steps / “space” = number of bits in memory
- “time” independent of language/implementation/machine
- We mostly focus on time, expressed as a function $T(n)$, where n is the **size of the instance** we try to solve
- Interested in asymptotic behavior of $T(n)$
- Notation: O , Ω , Θ , o , ω

3. Can we do better ?

Time Complexity of an algorithm

There are many instances of the same size

How does the algorithm work over all these instances?

Best-case complexity

- The **minimum** number of steps taken on any instance of size n
- Not useful, too optimistic

Worst-case complexity

- The **maximum** number of steps taken on any instance of size n
- An upper bound on the complexity of the problem
- The most usual analysis

Average case complexity

- The **average** number of steps taken on any instance of size n
- Depends on the distribution of instances (use of probabilities)

Time Complexity of a problem and lower bounds

Complexity of a problem Π : $T_{\Pi}(n)$

The (worst case) complexity of the best (known) algorithm A

$$T_{\Pi}(n) = \min_A \{T_A(n)\}$$

Obtaining a lower bound on a problem's complexity $L_{\Pi}(n)$:

- By proving that there is no algorithm with $T_A(n) < L_{\Pi}(n)$
- Rare results (e.g., $\log(n!)$ for sorting), very difficult to prove lower bounds on the required time for a given problem!

Optimal algorithm

- An algorithm A, for which $T_A(n) = L_{\Pi}(n)$
- For many problems we still do not know if we have found an optimal algorithm
- Even for well-studied problems, new improvements arise over the years

Algorithm Analysis

- Evaluation of time complexity
 - Average, worst, best case
- Appropriate solution depending on the application requirements

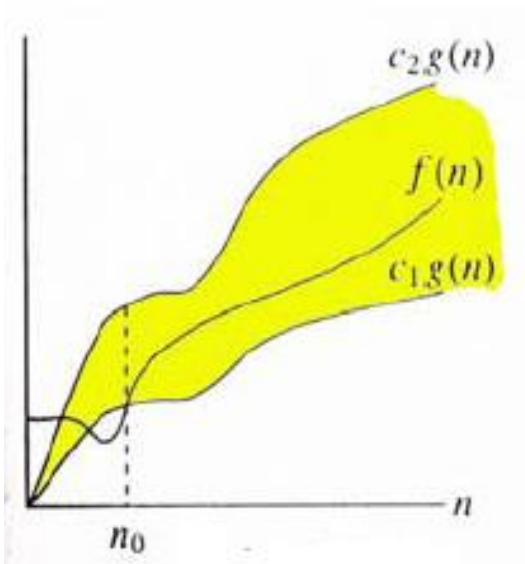
Benefits of theoretical analysis:

- Do not require experimental evaluation but only concrete description of the algorithm
- Results into general conclusions easy to verify, by considering all input instances, determining the time complexity as a function of the input size

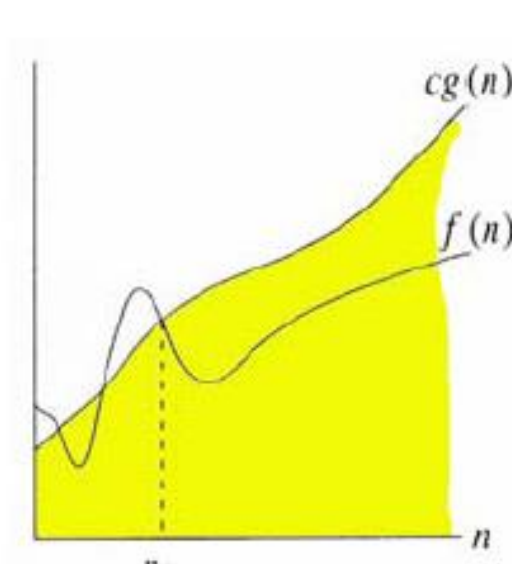
Mathematical background: discrete math (graphs, recurrence relations, combinatorics), mathematical logic, induction in all its forms (simple, strong, structural)

Asymptotic Notation

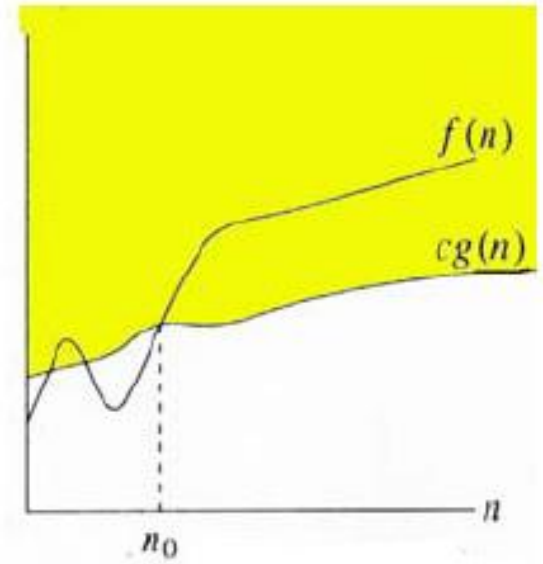
In pictures:



$$f(n) = \Theta(g(n))$$



$$f(n) = O(g(n))$$



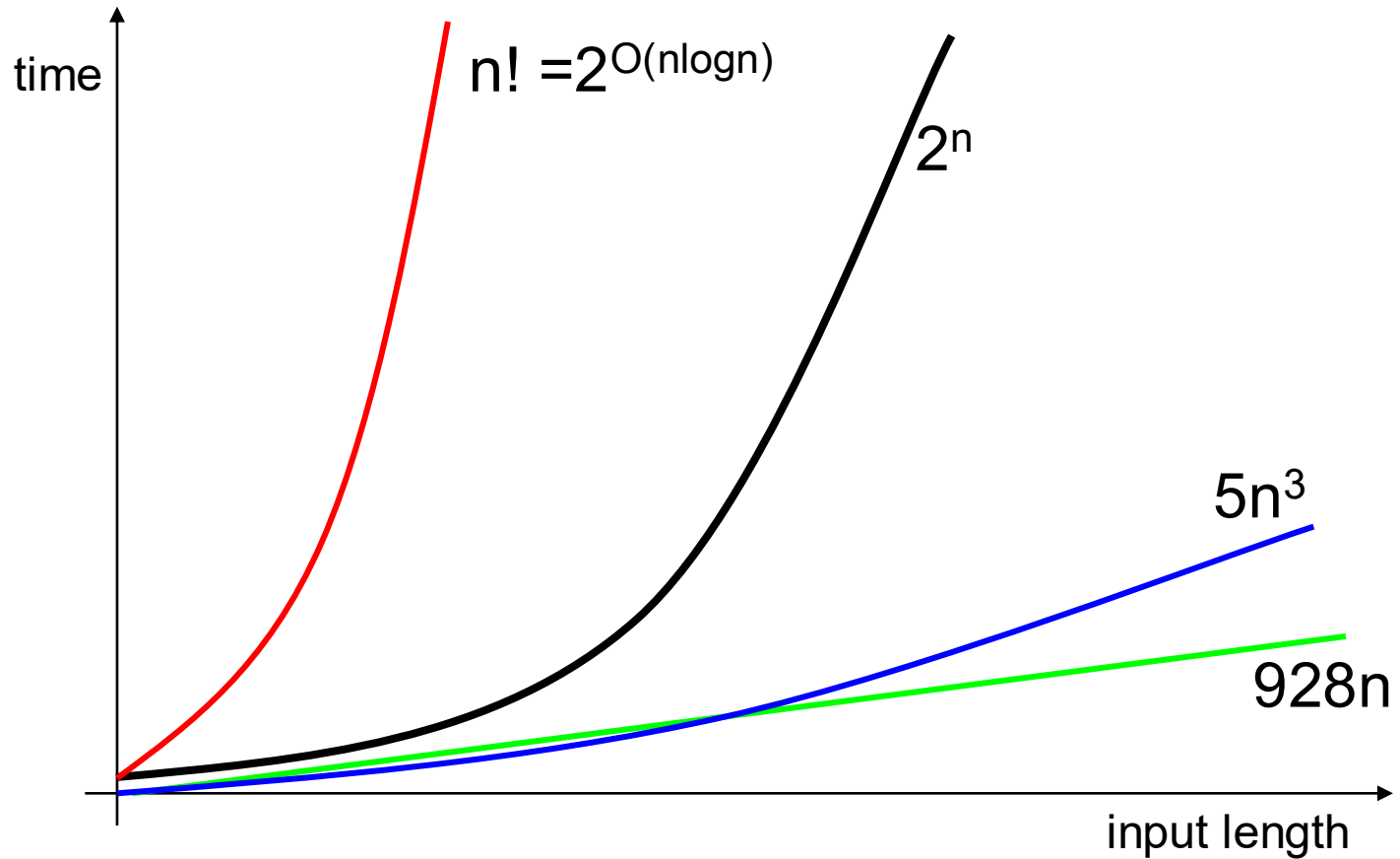
$$f(n) = \Omega(g(n))$$

Asymptotic Notation

More formally:

- A function $f(n)$ is $O(g(n))$ if there exist positive constants c_0 and n_0 such that $f(n) \leq c_0 g(n)$ for every $n \geq n_0$
 - The constant c_0 might be large (but still constant, **independent of** n)
 - Examples:
 - $2n + 10$ is $O(n)$. It suffices to set $c_0 = 3$ and $n_0 = 10$
 - $4n \log n + 150n + 3000 \sqrt{\log n} = O(n \log n)$. Set $c_0 = 3154$, $n_0 = 1$
- A function $f(n)$ is $\Omega(g(n))$ if there exist positive constants c_0 and n_0 such that $f(n) \geq c_0 g(n)$ for every $n \geq n_0$
- A function $f(n)$ is $\Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$

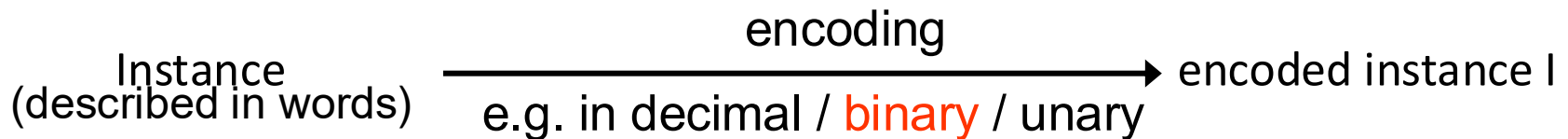
Growth of various functions



Size of instance and complexity

Consider the description of an instance (i.e., of all the parameters and constraints)

$||$ = length of encoded instance/input



$||$ = # of digits of the encoded input

Integer n:	Decimal	Binary	Unary
# bits	: $\lfloor \log_{10} n \rfloor + 1$	$\lfloor \log_2 n \rfloor + 1$	n

Size of instance and complexity

- We typically use the binary encoding
 - but there are reasons to consider other encodings too in complexity theory
- Hence, unless otherwise stated, $|I|$ = # of bits of the encoded input
- Let also $N(I)$ = the largest number in the input
 - Applicable only for problems that have numeric parameters in their input, like Knapsack
- Classification of algorithms
 - Polynomial algorithms: running time $O(\text{poly}(|I|))$
 - Exponential algorithms: running time $\Theta(\exp(|I|))$
 - Pseudo-Polynomial algorithms: $\Theta(\text{poly}(N(I)))$, which in worst case is $\Theta(\exp(|I|))$
 - We can say that they are $O(\text{poly}(|I|))$ if we consider I encoded in unary ! (i.e, polynomial when $N(I)$ not too large)
 - **Example:** Knapsack admits a dynamic programming algorithm with running time $O(n^2 v_{\max})$, where $v_{\max}$ is max value in the instance
 - Only relevant for problems with numeric parameters!
 - E.g., not relevant for SAT or for problems with unweighted graphs

Analyzing Recurrence Relations

The Master Theorem

- How do we analyze recurrence relations?
- There are essentially two methods
- **The substitution method:**
 - Keep substituting until you guess the solution
 - We can use induction to prove it formally

Example: $T(n) = T(n-1) + n$, $T(1) = 1$

- $T(n) = T(n-1) + n$
- $= (T(n-2) + n-1) + n$
- $= T(n-2) + n + n-1$
- $= (T(n-3) + n-2) + n + n-1$
- $= \dots$
- $= n + n-1 + n-2 + \dots + 2 + 1 = O(n^2)$

Is there a general result that could be applicable to the recurrence relations we will encounter?

The Master Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for some constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d), & \text{if } d > \log_b a \quad (b^d > a) \\ \Theta(n^d \log_b n), & \text{if } d = \log_b a \quad (b^d = a) \\ \Theta(n^{\log_b a}), & \text{if } d < \log_b a \quad (b^d < a) \end{cases}$$

- Usually convenient to think of n as a power of b , so that n/b is an integer.
- In many cases of interest, $b = 2$
- More general versions of this theorem are available as well

The Master Theorem - Examples

- Naive integer multiplication (by divide and conquer)
 - $T(n) = 4T(n/2) + O(n)$
 - $a = 4, b = 2, \log_b a = \log_2 4 = 2$
 - $d = 1 < 2 = \log_b a$
 - Case (iii) applies: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$
- Karatsuba's algorithm for integer multiplication
 - $T(n) = 3T(n/2) + O(n)$
 - $a = 3, b = 2, \log_b a = \log_2 3 = 1.59$
 - $d = 1 < \log_b a$
 - Case (iii) applies again: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{1.59})$

The Master Theorem - Examples

- $T(n) = 5T(n/25) + O(n^2)$
 - $a = 5, b = 25, \log_b a = \log_{25} 5 = 0.5$
 - $d = 2 > 0.5 = \log_b a$
 - case (i) applies: $T(n) = \Theta(n^d) = \Theta(n^2)$
- $T(n) = T(2n/3) + O(1)$
 - $a = 1, b = 3/2, \log_b a = \log_{3/2} 1 = 0$
 - $d = 0 = \log_b a$
 - case (ii) applies: $T(n) = \Theta(n^0 \log_{3/2} n) = \Theta(\log n)$
- $T(n) = 9T(n/3) + O(n)$
 - $a = 9, b = 3, \log_b a = \log_3 9 = 2$
 - $d = 1 < 2 = \log_b a$
 - case (iii) applies: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$