Algorithms for Data Processing
Lecture I: Introduction to Computational Efficiency

Alessandro Artale

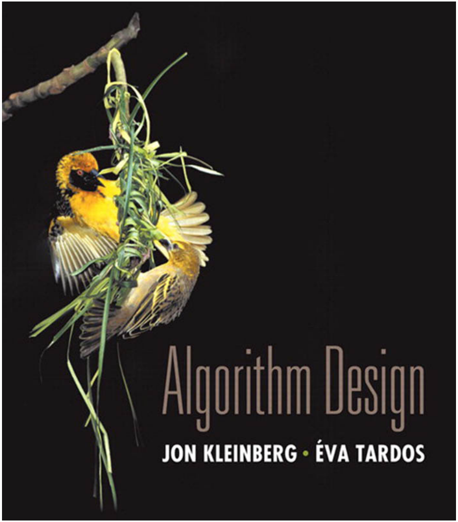
Free University of Bozen-Bolzano
Faculty of Computer Science
http://www.inf.unibz.it/˜artale
artale@inf.unibz.it

2019/20 – First Semester
MSc in Computational Data Science — UNIBZ

Course Overview

- Basic notions of algorithms complexity
- Basic notions of Graphs
- Algorithms for graph-based data structures
- Network Flow algorithms
- Computability and complexity of problems
  - Hard Problems: NP-complete problems
- Algorithms for Hard Problems:
  - Exact algorithms for NP problems
  - Approximation algorithms for NP problems

Reading Book



Algoritm Design. Jon Kleinberg and Éva Tardos. Addison-Wesley, 2005.

Algorithms and Problems

Analyzing algorithms involves thinking about how the amount of time and space they use will scale with increasing input size.

There is a set of critical questions when speaking of algorithms solving a given problem:

- How do we know if a problem can be solved algorithmically at all?
- How do we know if a problem can be solved efficiently?
- How do we measure efficiency?

Attempts at Defining Efficiency

An algorithm is efficient if it runs quickly on real input instances.

On the other hand:

- Bad Algorithms can run quickly when applied to small test cases on extremely fast processors.
- Good Algorithms can run slowly when they are coded sloppily.
- We don't know the full range of input instances: some input instances can be much harder than others. Thus, *Testing* is not always adequate.

Attempts at Defining Efficiency (cont.)

- A concrete definition of efficiency must be platform-independent, instance-independent, and of predictive value with respect to increasing input sizes.
- We should define an objective measure describing how well, or badly, an algorithm scales as problem sizes grow.

A bit of Complexity – Objective Measures

The Complexity of an algorithm is measured considering two important resources:

1. Time Complexity: Considers the Running Time of the algorithm.
2. Space Complexity: Considers the Amount of Memory the algorithm needs to complete the computation.

- Time-complexity is usually more critical due to the fact that nowadays memory is largely available.
- High space-complexity implies high time-complexity. The viceversa is not always the case.

Worst Case and Average Running Time Analysis

- **Worst-Case Running Time:** Looks for a bound on the largest possible running time the algorithm could have over all inputs of a given size N, and see how this scales with N.
- **Average Running Time:** Studies the performance of an algorithm averaged over "random" instances.

Worst Case Analysis Vs Average Running Time

- How a random input should be generated?
- The same algorithm can perform very well on one class of random inputs and very poorly on another.
- Average-case analysis risks telling us more about how the random inputs were generated than about the algorithm itself.

Brute Force Algorithms

For many problems, there is a natural brute-force search algorithm that checks every possible solution (e.g., Sorting can be performed checking the $N!$ possible permutation)

- Typically takes 2^N steps (or worse) for inputs of size N.
- Unacceptable in practice.
- It is an intellectual escape: it provides us with absolutely no insight into the structure of the problem we are studying.

Attempts at Defining Efficiency (2)

An algorithm is efficient for a given problem if it achieves qualitatively better worst-case performance (at an analytical level) than brute-force search.

- Algorithms that improve substantially on brute-force search usually contain a valuable heuristic idea.
- They tell us something about the intrinsic structure of the underlying problem.

In the following, we consider the running time of algorithms more carefully, and try to quantify what a reasonable running time should be.

Polynomial running time

- Polynomial-time algorithm (**PTime**). When on every input instance of size N the algorithm running time is bounded by cN^d, for some constants $c, d > 0$.
- Desirable scaling property. When the input size doubles, the algorithm slow down by at most some constant factor k.

An Algorithm is Efficient if it has a polynomial running time.

Worst-Case Analysis and Algorithm Complexity

Worst Case. Running time is guaranteed for any input of size N.

- Draconian view, but hard to find better objective alternatives.
- Is only an abstraction of practical situations.
- PTime generally captures efficiency in practice.
- There is a fundamental benefit to making our definition of efficiency so specific: it becomes falsifiable.
  - It becomes possible to express the notion that there is no efficient algorithm for a particular problem!

Thank You!