

Complexity Theory

Lecture 1

Anuj Dawar

University of Cambridge Computer Laboratory
Easter Term 2010

<http://www.cl.cam.ac.uk/teaching/0910/Complexity/>

Texts

The main texts for the course are:

Computational Complexity.

Christos H. Papadimitriou.

Introduction to the Theory of Computation.

Michael Sipser.

Other useful references include:

Computers and Intractability: A guide to the theory of NP-completeness.

Michael R. Garey and David S. Johnson.

Computational complexity: a conceptual perspective.

O. Goldreich.

Computability and Complexity from a Programming Perspective.

Neil Jones.

Outline

A rough lecture-by-lecture guide, with relevant sections from the text by Papadimitriou (or Sipser, where marked with an S).

- **Algorithms and problems.** 1.1–1.3.
- **Time and space.** 2.1–2.5, 2.7.
- **Time Complexity classes.** 7.1, S7.2.
- **Nondeterminism.** 2.7, 9.1, S7.3.
- **NP-completeness.** 8.1–8.2, 9.2.
- **Graph-theoretic problems.** 9.3

Outline - contd.

- **Sets, numbers and scheduling.** 9.4
- **coNP.** 10.1–10.2.
- **Cryptographic complexity.** 12.1–12.2.
- **Space Complexity** 7.1, 7.3, S8.1.
- **Hierarchy** 7.2, S9.1.
- **Descriptive Complexity** 5.6, 5.7.

Complexity Theory

Complexity Theory seeks to understand what makes certain problems algorithmically difficult to solve.

In [Data Structures and Algorithms](#), we saw how to measure the complexity of specific algorithms, by asymptotic measures of number of steps.

In [Computation Theory](#), we saw that certain problems were not solvable at all, algorithmically.

Both of these are prerequisites for the present course.

Algorithms and Problems

[Insertion Sort](#) runs in time $O(n^2)$, while [Merge Sort](#) is an $O(n \log n)$ algorithm.

The first half of this statement is short for:

If we count the number of steps performed by the [Insertion Sort](#) algorithm on an input of size n , taking the largest such number, from among all inputs of that size, then the function of n so defined is **eventually** bounded by a **constant multiple** of n^2 .

It makes sense to compare the two algorithms, because they seek to solve the same problem.

But, what is the complexity of the **sorting problem**?

Lower and Upper Bounds

What is the running time complexity of the fastest algorithm that sorts a list?

By the analysis of the [Merge Sort](#) algorithm, we know that this is no worse than $O(n \log n)$.

The complexity of a particular algorithm establishes an **upper bound** on the complexity of the problem.

To establish a **lower bound**, we need to show that no possible algorithm, including those as yet undreamed of, can do better.

In the case of sorting, we can establish a lower bound of $\Omega(n \log n)$, showing that [Merge Sort](#) is asymptotically optimal.

Sorting is a rare example where known upper and lower bounds match.

Review

The complexity of an algorithm (whether measuring number of steps, or amount of memory) is usually described asymptotically:

Definition

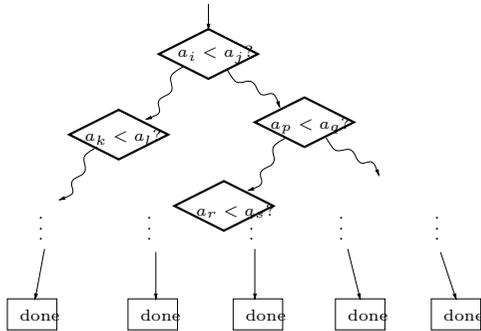
For functions $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N} \rightarrow \mathbb{N}$, we say that:

- $f = O(g)$, if there is an $n_0 \in \mathbb{N}$ and a constant c such that for all $n > n_0$, $f(n) \leq cg(n)$;
- $f = \Omega(g)$, if there is an $n_0 \in \mathbb{N}$ and a constant c such that for all $n > n_0$, $f(n) \geq cg(n)$.
- $f = \theta(g)$ if $f = O(g)$ and $f = \Omega(g)$.

Usually, O is used for upper bounds and Ω for lower bounds.

Lower Bound on Sorting

An algorithm A sorting a list of n distinct numbers a_1, \dots, a_n .



To work for all permutations of the input list, the tree must have at least $n!$ leaves and therefore height at least $\log_2(n!) = \theta(n \log n)$.

Travelling Salesman

Given

- V — a set of nodes.
- $c : V \times V \rightarrow \mathbb{N}$ — a cost matrix.

Find an ordering v_1, \dots, v_n of V for which the total cost:

$$c(v_n, v_1) + \sum_{i=1}^{n-1} c(v_i, v_{i+1})$$

is the smallest possible.

Complexity of TSP

Obvious algorithm: Try all possible orderings of V and find the one with lowest cost.

The worst case running time is $\theta(n!)$.

Lower bound: An analysis like that for sorting shows a lower bound of $\Omega(n \log n)$.

Upper bound: The currently fastest known algorithm has a running time of $O(n^2 2^n)$.

Between these two is the chasm of our ignorance.

Formalising Algorithms

To prove a **lower bound** on the complexity of a problem, rather than a specific algorithm, we need to prove a statement about **all** algorithms for solving it.

In order to prove facts about all algorithms, we need a mathematically precise definition of algorithm.

We will use the *Turing machine*.

The simplicity of the Turing machine means it's not useful for actually expressing algorithms, but very well suited for proofs about all algorithms.