CSC2401 Introduction to Complexity, Fall 01-02
Lecture 1, Tuesday, 12th September
Lecturer: Allan Borodin
Scribe: Matei David

1 Administrative Details

- Web page: http://www.cs.toronto.edu/˜bor/2401f01/index.html
- E-mail: Allan Borodin <bor@cs.toronto.edu>
- Book: *Theory of Computational Complexity*, by Ding-Zhu Du and Ker-I KO
- Evaluation: 3 or 4 homework assignments, maybe less difficult than Charlie Rackoff's
- Will see some research ideas, yet this is not the goal of the course.
- Generally, try to finish assignments on time.
- Collaboration: No joint submissions, discuss but make sure you understand (watch TV for 2 hours...), do your own work. Can work in common for hard, open ended questions (like P=NP...).

2 Introduction

The underlying goal of Complexity Theory is to answer questions such as:

- What does it mean to say that a function f is *intrinsically difficult* to compute?
- How to measure the complexity of a computation? Different options: sequential time, space, parallel time.
- What are the appropriate models of computation within which to formulate these issues?
- How do these models and measures relate to each other?
- What is the power of randomization?

In general we will think of a *function* as a computational problem, ideally independent of the machine or the language used to compute it.

The results we are looking for are of two types: *lower bounds*, also called negative or impossibility results, which prove that "no algorithm can do better than ..."; and *upper bounds*, also called positive results. The latter are obtained in general by (good) algorithms, while most of the former (in our context) will involve *diagonalization* arguments. There are intuitive lower bound conjectures for which we haven't found proofs yet: as an example, Gauss raised such a question as how to factor numbers, long before computers became a reality. We haven't been able to prove that there are no polytime algorithms for that. And indeed integer factoring is now known to be computable in polynomial time by quantum computers.

The success of the complexity theory can be seen in other fields. Specif ically, compolexity theory generated the field of complexity based cryptography, which is to this day based on conjectures resulting from complexity theory. This field requires that $P \neq NP$ but we still do not know if SAT (satisfiability) is polytime, although we conjectire that it is not computable in polynomial time. However, we also thought of matrix multiplication to be $\Omega(n^3)$, and it turned out to be $O(n^\alpha)$ for $\alpha < 3$. (I think the best current bound is $\alpha \approx 2.38$ and it is still not known if $\alpha > 2$.)

3 Basics

Our basic object of study will be

$$f : \Sigma^* \rightarrow \Sigma^*$$

with Σ a finite alphabet. Of special concern will be

$$f : \Sigma^* \rightarrow \{yes, no\}$$

The latter are called decision problems. Another problem type comes from optimization questions: given x, find y such that the relation $R(x, y)$ holds, where $R \subset \Sigma^* \times \Sigma^*$.

There are a number of hidden assumptions in this notation. For example, how to represent numbers? Any base other than 1 is good enough, as the representations differ only by a constant factor. The input may consist of letters, numbers, etc, but everything can be coded in a binary alphabet: $\{0, 1\}$

A problem can be defined in terms of basic objects: addition is $O(n)$ if seen bitwise.

Our *Basic Uniform Model of Computation* is a multi-tape deterministic Turing Machine. When we will say that something can or cannot be done in $O(n^2)$, we will understand that it can or cannot be done in that time/space using one such TM.

We will not look into Quantum computation, but instead will restrict ourselves to the classical TM model. Maybe a lecture at the end for Quantum computation.

We will also consider probabilistic and nondeterministic Turing Machines.