W4231: Analysis of Algorithms

9/7/1999 (revised 9/8/1999)

- Introduction
- Models of Computation
- Lower Bounds

People

Lecturer

Luca Trevisan (`luca@cs.columbia.edu`)

Office 462CSB — Office hours Mondays 6-7pm, Thursdays 11-12am

TA

Dario Catalano (`dario@cs.columbia.edu`)

Office 509CSB — Office hours TBA

Book

[CLR] Thomas H. Cormen, Charlie E. Leiserson and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

In stock at Labyrinth bookstore.

Handouts etc.

All handouts, notes, slides, etc. are available on the web page

`http://www.cs.columbia.edu/≈luca/w4231/fall99`

Check the page often for announcements and for revised versions of notes etc.

Topics

Review of basic material. Models of computation, space and time complexity, lower bounds, recurrences.

Sorting and searching. Applications of divide and conquer; hashing; binomial heaps and Fibonacci heaps.

Graph Algorithms. Connectivity, flows, cuts, matchings.

Hard Problems. Dynamic programming, NP-completeness.

Cryptographic Algorithms. Operations on big integers, RSA, primality testing.

Policies

- Deadlines are strict. They are two days later for CVN students.
- Collaboration is not allowed.
- Grades are 55% from homeworks, 20% from midterm, 25% from final.

Algorithms

In computer science we want to solve computational problems using a computer.

An algorithm is an abstract description of a method to do so.

In this course we study how to design algorithms that: are correct; use as little memory and time as possible.

The emphasis is on how to *prove* that our algorithms are correct and use limited time and memory.

Efficiency

We mostly concentrate on running time.

When you have to process large data sets, a more efficient algorithm can make all the difference as of whether you can solve your problem in your lifetime or not.

Exponential versus quadratic

Say that you have a problem that, for an input consisting of n items, can be solved by going through 2^n cases.

Say you have a computer like the version of Deep Blue that challenged Kasparow (can analyse 200 million cases per second).

An input with 15 items will take 163 microseconds.

An input with 30 items will take 5.36 seconds.

An input with 50 items will take more than two months.

An input with 80 items will take 191 million years.

Another algorithm uses $300n^2$ clock cycles on a 80386, and you use a PC running at 33MHz.

An input with 15 items will take 2 milliseconds.

An input with 30 items will take 8 milliseconds.

An input with 50 items will take 22 milliseconds.

An input with 80 items will take 58 milliseconds.

Role of improved hardware

The largest instance solvable in a day by the 2^n algorithm using Deep Blue has 44 items. Using a computer 10 times faster we can go to 47 items. (In general, we go from I to $I+3$ or $I+4$ items.)

The largest instance solvable in a day by the $300n^2$ algorithm on the old PC has 97488 items. Using a computer 10 times faster we can go to 308285 items. (In general, from I to $\sqrt{10}I$)

Polynomial time and efficiency

Whenever an algorithm runs in $O(n^c)$ time, where n is the size of the input and c is a constant, we say that the algorithm is "efficient".

We want to find polynomial-time algorithms for every interesting problem, and with the smallest exponent.

An algorithm running in $O(n \log n)$ time is always preferable to an $O(n^2)$ algorithm, for all but finitely many instances.

Asymptotic Notation

Recall that when we say that the running time of an algorithm is $O(n^2)$ we mean that for all but finitely many n the time is at most cn^2 where c is a fixed constant.

In general $g(n) = O(f(n))$ means that $g(n) \leq cf(n)$ for a fixed constant c and for all but finitely many n.

Danger of Asymptotic Notation

We typically try to get algorithms with the best $O(\cdot)$ running time.

Then we might prefer an algorithm requiring $1,000,000n$ operations to an algorithm requiring $1,000n \log n$ operations for inputs of length n.

Even though the former algorithm is better for all but finitely many instances, the latter is better for all the instances that can exist in the known universe.

Danger of Worst Case Analysis

All the algorithms we will see in this course work well even in the presence of input data “adversarially” designed in order to make the algorithms perform poorly. Sometimes this strong requirement comes at the expense of major complications.

We may have a complicated algorithm working well for every input, and a simpler one that works as well (or even better) on most inputs, but that is really bad on certain particular input data.

The latter algorithm may be preferable in practice but we would only see algorithms of the former type.

Analysis of Algorithms

The principles of doing worst-case analysis, ignoring the constants hidden in the $O(\cdot)$ notation, and emphasizing proofs of correctness and efficiency led to a beautiful theory and to very useful ideas.

When doing research on algorithms, and when learning how to design algorithms there are no better principles.

The actual design of practical algorithms for specific problems may involve different principles.

Goals of this Course

To show, by example, ways to reason about problems, and to find unexpected and brilliant solutions.

You will see that sometimes the best way to solve a problem is a very counter-intuitive one; that a procedure that seems the only possible one may be improved substantially; and that problems that look very different have deep connections (and similar efficient algorithms).

This knowledge and set of skills are very useful in practice. (Possibly more than the actual examples.)

Example 1: Integer Multiplication

Suppose you have two really big integers (say, a million digits) that you have to multiply.

The standard way of multiplying two n-digits integers takes $O(n^2)$ operations. The procedure looks optimal.

Still, you can easily do multiplication in $O(n^{1.6})$ time, and even about $O(n \log n)$.

Example 2: Median

Suppose you have a non-sorted vector of integers $a_1, \ldots, a_n$.

Suppose you want to find the value a that would be in the middle of the vector if it was sorted.

If you solve the problem using sorting it will take $O(n \log n)$ time.

You can find the median in $O(n)$ time.

Example 3: Integer Partition

Suppose you are given integers $a_1, \ldots, a_n$ whose sum is $A = \sum_i a_i$. You want to find a subset $S \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in S} a_i = A/2$$

if such a set exists.

You can try all possible sets S. There are 2^n of them!

But there is also an algorithm that takes time $O(An)$, which is much less than 2^n if A is not too big.

Models of Computation

We want to design algorithms that are as efficient as possible, and prove that they really are efficient.

If we want mathematical theorems that talk about algorithms, we need to have a mathematical model of an algorithm (running on a computer) and a formal quantitative definition of efficiency.

The RAM Model

The RAM model is an abstract model of computation that is essentially a processor equipped with a register, an unbounded amount of memory and the usual operations. Each memory location, and the register, holds an arbitrary integer. In one step, one machine-language instruction is executed. An algorithm is formalized as a machine-language program.

Up to $O(\cdot)$ notation, it is the same as we think of C programs as our model of computation, and as elementary instructions as taking unit time.

The Decision Tree Model

A decision tree is a way of specifying the way an algorithms works for inputs of a certain length.

We see an input of length n as a string of integers $x_1, \ldots, x_n$. Our computation at any step reads one of the elements of the input and moves to a new "state." Then we can arrange the states as a tree, where the root is the initial state, and branches correspond to the direction taken by the computation.

Leaves determine the output. The "time" taken by a decision tree computation is the depth of the tree.

Lower Bound for Sorting

A sorting algorithm for inputs of length n has $n!$ possible ways of re-arranging its input.

In the decision tree model, the tree must have $n!$ leaves. Then the depth has to be $\log_2 n! > n \ln n - en$.

Meaning of the Lower Bound

A RAM algorithm that accesses the input only by means of operators returning boolean values must take time at least $\Omega(n \log n)$.

Merge-sort runs in time $O(n \log n)$ with this type of access to the input, and so it is optimal up to the constant in the $O(\cdot)$ notation.