Algorithms and Data Structures

Richard Mayr

School of Informatics
University of Edinburgh

Algorithms and Data Structures

- Emphasis is “Algorithms” rather than “Data Structures”.
- More proving in ADS than in Inf2 (ADS part).
- Most of the algorithms we study were *breakthroughs* at the time when they were discovered (50s, 60s, and 70s).
- Two concerns in ADS:
  1. *Designing* clever algorithms to solve problems.
  2. *Proving* that our algorithms are *correct*, and satisfy certain *bounds on running time*.
- We use three main techniques to design algorithms:
  1. Divide-and-Conquer
  2. Greedy approach (also called “hill climbing”)
  3. Dynamic programming

Algorithms and Data Structures

- Emphasis is “Algorithms” rather than “Data Structures”.
- More proving in ADS than in Inf2 (ADS part).
- Most of the algorithms we study were *breakthroughs* at the time when they were discovered (50s, 60s, and 70s).
- Two concerns in ADS:
  1. *Designing* clever algorithms to solve problems.
  2. *Proving* that our algorithms are *correct*, and satisfy certain *bounds on running time*.
- We use three main techniques to design algorithms:
  1. Divide-and-Conquer
  2. Greedy approach (also called “hill climbing”)
  3. Dynamic programming

Algorithms and Data Structures

- Emphasis is “Algorithms” rather than “Data Structures”.
- More proving in ADS than in Inf2 (ADS part).
- Most of the algorithms we study were *breakthroughs* at the time when they were discovered (50s, 60s, and 70s).
- Two concerns in ADS:
  1. *Designing* clever algorithms to solve problems.
  2. *Proving* that our algorithms are *correct*, and satisfy certain *bounds on running time*.
- We use three main techniques to design algorithms:
  1. Divide-and-Conquer
  2. Greedy approach (also called “hill climbing”)
  3. Dynamic programming

Algorithms and Data Structures

- Emphasis is “Algorithms” rather than “Data Structures”.
- More proving in ADS than in Inf2 (ADS part).
- Most of the algorithms we study were *breakthroughs* at the time when they were discovered (50s, 60s, and 70s).
- Two concerns in ADS:
  1. *Designing* clever algorithms to solve problems.
  2. *Proving* that our algorithms are *correct*, and satisfy certain *bounds on running time*.
- We use three main techniques to design algorithms:
  1. Divide-and-Conquer
  2. Greedy approach (also called “hill climbing”)
  3. Dynamic programming

Algorithms and Data Structures

- Emphasis is “Algorithms” rather than “Data Structures”.
- More proving in ADS than in Inf2 (ADS part).
- Most of the algorithms we study were *breakthroughs* at the time when they were discovered (50s, 60s, and 70s).
- Two concerns in ADS:
  1. *Designing* clever algorithms to solve problems.
  2. *Proving* that our algorithms are *correct*, and satisfy certain *bounds on running time*.
- We use three main techniques to design algorithms:
  1. Divide-and-Conquer
  2. Greedy approach (also called “hill climbing”)
  3. Dynamic programming

Syllabus

Introductory Review of Inf2 basics. Time and space complexity; upper and lower bounds; $O(\cdot)$, $\Omega(\cdot)$ and $\Theta(\cdot)$ notation; average and worst case analysis.

Algebraic algorithms Matrix multiplication: Strassen's algorithm.
Polynomial arithmetic: the Discrete Fourier transform (DFT), the Fast Fourier transform (FFT); recurrence relations for recursive algorithms.

Sorting Analysis of Quicksort; best-case, worst-case and average-case analysis.
Sorting for restricted-values; counting sort, radix sort.

Syllabus cont.

Dynamic programming: Introduction to the technique; matrix-chain multiplication, other examples.

Advanced data structures: Data structures for disjoint sets; Union-by-rank, path-compression, etc., "heuristics".

Minimum spanning trees: Prim's algorithm (using priority queues); Kruskal's algorithm (using disjoint sets).

Graph/Network algorithms Network flows; Ford-Fulkerson algorithm for finding max flow.

Geometric algorithms: Convex hull of a set of points in two dimensions; Graham's scan algorithm.

Course Book

- T. H. Cormen, C. E. Leiserson and R. L. Rivest, C. Stein
  Introduction to Algorithms (3rd Edition),
  MIT Press 2009.
- Called **[CLRS]** from now on.
- *Essential* for the course.

It will be possible to work with the 2nd edition (or even the 1st edition, which is just CLR (without Stein)). I'll try to reference the 2nd edition numbering as well as the 3rd (but if using [CLR], it is your responsibility to find the right sections in [CLR]).

Other References

- Kleinberg and Tardos: *Algorithm Design*. Addison-Wesley, 2005. (Nice book - but doesn't cover many of our topics).
- Sedgewick: *Algorithms in C (Part 1-5)*, Addison Wesley, 2001.

Course Webpage
(with slides, tutorials, coursework, etc.)

`http://www.inf.ed.ac.uk/teaching/courses/ads/`

Pre-requisites

Official pre-requisites:

- *Passes* in Inf2 & Discrete Math/"Probability with Applications" (or year 2 Honours Maths).

Un-official recommendation (not enforced):

- Should be better than 50% at first attempt in Inf2 and your 2nd year Maths courses (and better again if second or later attempt).
- If you are a Visiting student or MSc, drop-me-an-email.

If you didn't take Inf2, but have *excellent* Maths, will be ok . . .

should be happy doing small proofs, not just applying Maths.

Pre-requisites

Official pre-requisites:

- *Passes* in Inf2 & Discrete Math/"Probability with Applications" (or year 2 Honours Maths).

Un-official recommendation (not enforced):

- Should be better than 50% at first attempt in Inf2 and your 2nd year Maths courses (and better again if second or later attempt).
- If you are a Visiting student or MSc, drop-me-an-email.

If you didn't take Inf2, but have *excellent* Maths, will be ok . . .

should be happy doing small proofs, not just applying Maths.

Pre-requisites

Official pre-requisites:

- *Passes* in Inf2 & Discrete Math/"Probability with Applications" (or year 2 Honours Maths).

Un-official recommendation (not enforced):

- Should be better than 50% at first attempt in Inf2 and your 2nd year Maths courses (and better again if second or later attempt).
- If you are a Visiting student or MSc, drop-me-an-email.

If you didn't take Inf2, but have *excellent* Maths, will be ok . . .

should be happy doing small proofs, not just applying Maths.

Pre-requisites

Official pre-requisites:

- *Passes* in Inf2 & Discrete Math/"Probability with Applications" (or year 2 Honours Maths).

Un-official recommendation (not enforced):

- Should be better than 50% at first attempt in Inf2 and your 2nd year Maths courses (and better again if second or later attempt).
- If you are a Visiting student or MSc, drop-me-an-email.

If you didn't take Inf2, but have *excellent* Maths, will be ok . . .

should be happy doing small proofs, not just applying Maths.

Math Pre-requisites

You should know:

- how to multiply matrices or polynomials,
- some probability theory,
- some graph theory,
- what it means to *prove* a theorem (induction, proof by contradiction, . . .) and to be confident in your ability to do this.

The appendices of [CLRS] might be useful for reviewing your math.

Tutorials start week 3

Details of the tutorial group allocations, times, and places will be available on the course webpage

`http://www.inf.ed.ac.uk/teaching/courses/ads/`

Your own work (formative assessment)

- Tutorial sheet every week.
  It is very important that you attempt these problems BEFORE tutorials! Preparing for tutorials will make a huge difference in what you get out of the course - it will massively improve your final grade.
  - You should participate in tutorial discussions. There is often more than one way to solve a question.
- Also . . . it's a good idea to try coding-up a few of the algorithms :)

Coursework (summative assessment)

There will be 2 Assessed Courseworks, one formative and one summative (counting 50% towards the course mark).

- Coursework 1 (formative; not counting towards the course mark)
- Coursework 2 (summative; counting 50% towards the course mark)

See the course webpage
`http://www.inf.ed.ac.uk/teaching/courses/ads/`
for coursework deadlines.

Basic Notions

Model of Computation: An abstract sequential computer, called a *Random Access Machine* or *RAM*. Uniform cost model.

Computational Problem: A specification in general terms of *inputs* and *outputs* and the desired input/output relationship.

Problem Instance: A particular collection of inputs for a given problem.

Algorithm: A method of solving a problem which can be implemented on a computer.
Usually there are many algorithms for a given problem.

Program: Particular implementation of some algorithm.

Algorithms and “Running time”

- Formally, we define the running time of an algorithm on a particular input instance to be the *number of computation steps performed by the algorithm on this instance*.
- This depends on our *machine model* - need the algorithm to be written as a program for such a machine.
- number of basic arithmetic operations - abstract way of only counting the *essential* computation steps.
- Both notions are abstractions of the *actual running time*, which also depends on factors like
  - Quality of the implementation
  - Quality of the code generated by the compiler
  - The machine used to execute the program.

Worst-Case Running Time

Assign a *size* to each possible input (this will be proportional to the length of the input, in some *reasonable* encoding).

Definition

The *(worst-case) running time* of an algorithm A is the function $T_A : \mathbb{N} \to \mathbb{N}$ where $T_A(n)$ is the maximum number of computation steps performed by A on an input of size n.

- A similar definition applies to other measures of resource.

Average-Case Running Time

Definition

The *average-case running time* of an algorithm A is the function $\mathrm{AVT}_A : \mathbb{N} \to \mathbb{N}$ where $\mathrm{AVT}_A(n)$ is the *average* number of computation steps performed by A on an input of size n.

For a genuine average–case analysis we need to know for each n the probability with which each input turns up. Usually we assume that all inputs of size n are equally likely.

Bounds

Given a *problem*, a function $T(n)$ is an:

Upper Bound: If there is an algorithm which solves the problem and has worst-case running time at most $T(n)$.

Average-case bound: If there is an algorithm which solves the problem and has average-case running time at most $T(n)$.

Lower Bound: If *every* algorithm which solves the problem must use at least $T(n)$ time on some instance of size n for infinitely many n.

A little thought goes a long way

Problem: Remainder of a power.

Input: Integers a, n, m with $n \geq 1$, $m > 1$.

Output: The remainder of a^n divided by m, i.e., $a^n \bmod m$.

Algorithm POWER-REM1(a, n, m)

1. $r \leftarrow a$
2. **for** $j \leftarrow 2$ **to** n **do**
3. $\quad r \leftarrow r \cdot a$
4. **return** r mod m

- Real world: integer overflow even for small a, m and moderate n.
- Even without overflow numbers become needlessly large.

A little thought . . . (cont'd)

Algorithm POWER-REM2(a, n, m)

1. $x \leftarrow a \bmod m$
2. $r \leftarrow x$
3. **for** $j \leftarrow 2$ **to** n **do**
4. $\quad\quad r \leftarrow r \cdot x \bmod m$
5. **return** r

Much better than POWER-REM1.

- No integer overflow (unless m large).
- Arithmetic more efficient — numbers kept small.

A little thought . . . (cont'd)

Algorithm POWER-REM3(a, n, m)

1. **if** $n = 1$ **then**
2. **return** $a \bmod m$
3. **else if** n even **then**
4. $r \leftarrow$ POWER-REM3$(a, n/2, m)$
5. **return** $r^2 \bmod m$
6. **else**
7. $r \leftarrow$ POWER-REM3$(a, (n-1)/2, m)$
8. **return** $(r^2 \bmod m) \cdot a \bmod m$

Even better.

- No integer overflow (unless a, m large), nums kept small.
- Number of arithmetic operations even less.

Reading Assignment

[CLRS] Chapters 1 and 2.1-2.2 (pp. 1–27) (all this material should be familiar from Inf2).

If you did not take Inf2 . . . read all of the ADS part of Inf2.

Problems

1. Analyse the asymptotic worst-case running time of the three POWER-REM algorithms.
   Hint: The worst-case running time of POWER-REM1 and POWER-REM2 is $\Theta(n)$, and the worst-case running time of POWER-REM3 is $\Theta(\lg n)$
2. Exercise 1.2-2, p. 13 of [CLRS] (*Ex 1.4-1, p 17 in [CLR]*).
3. Exercise 1.2-3, p. 13 of [CLRS] (*Ex 1.4-2, p 17 in [CLR]*).