Algorithms and Data Structures

or, Classical Algorithms of the 50s, 60s, 70s

Mary Cryan
School of Informatics
University of Edinburgh

▶ Emphasis is “Algorithms” rather than “Data Structures”.
▶ More proving in ADS than in Inf2B (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 Inf2B 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.

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

- 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]).

Pre-requisites

**Official pre-requisites:**
- Passes in Inf2B & DMMR/“Probability with Applications” (or year 2 Honours Maths).

**Un-official recommendation (not enforced):**
- Should be better than 50% at first attempt in Inf2B and your 2nd year Maths courses (and better again if second or later attempt).
- At the end, it is your call (drop-me-an-email if you like).
- If you are a Visiting student or MSc, drop-me-an-email.

If you didn’t take Inf2B, but have excellent Maths, will be ok... should be happy doing small proofs, not just applying Maths.

Other References


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.

Course Webpage

(with transparencies and lecture log)

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.
- Office Hours. TBA. This is an opportunity to ask questions about material and tutorial questions and to get feedback on your own work.
- Also ... it’s a good idea to try coding-up a few of the algorithms :)

Coursework (summative assessment)

There will be 2 Assessed Courseworks, equally weighted. Taken together they are worth 25% of the course mark (so 12.5% each). Details of release-date, the due-date, and date for return-of-feedback are:

- Coursework 1 (written problem/proof set)
  - OUT Fri, 26th Sept (Fri week 2)
  - DUE 4pm Fri, 17th Oct (Fri week 5)
  - FEEDBACK returned by Fri, 31st Oct (Fri week 7)
- Coursework 2 (mix of programming and theoretical work)
  - OUT Tues, 21st Oct (Tues week 6)
  - DUE 4pm Tues, 11th Nov (Tues week 9)
  - FEEDBACK returned by Tues, 25th Nov (Tues week 11)

Feedback given will include marks to individual sub-parts of questions, comments on scripts to explain why marks were lost, plus a description of common errors.

Tutorials start week 3

We have reserved slots/tutors for Tuesday at 2:10pm, Wednesday at 3:10pm and Friday at 1:10pm.

The initial allocation of students to tutorials will be linked-to from the ADS course webpage soon. I will send an email to the class-list also to announce this. If you cannot make your allocated slot, please tell the ITO all other times that will work for you.

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 $T_A : \mathbb{N} \rightarrow \mathbb{N}$ 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.

Bound

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 $n$. 

Average-Case Running Time

**Definition**
The average-case running time of an algorithm $A$ is the function $AVT_A : \mathbb{N} \rightarrow \mathbb{N}$ where $AVT_A(n)$ 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.
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., $a^n \mod m$.

**Algorithm** $\text{POWER-REM}_1(a, n, m)$

1. $r \leftarrow a$
2. for $j \leftarrow 2$ to $n$ do
3. \hspace{0.5cm} $r \leftarrow r \cdot a$
4. \hspace{0.5cm} return $r \mod m$

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

**Algorithm** $\text{POWER-REM}_2(a, n, m)$

1. $x \leftarrow a \mod m$
2. $r \leftarrow x$
3. for $j \leftarrow 2$ to $n$ do
4. \hspace{0.5cm} $r \leftarrow r \cdot x \mod m$
5. \hspace{0.5cm} return $r$

Much better than $\text{POWER-REM}_1$.

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

**Algorithm** $\text{POWER-REM}_3(a, n, m)$

1. if $n = 1$ then
2. \hspace{0.5cm} return $a \mod m$
3. else if $n$ even then
4. \hspace{0.5cm} $r \leftarrow \text{POWER-REM}_3(a, n/2, m)$
5. \hspace{0.5cm} return $r^2 \mod m$
6. else
7. \hspace{0.5cm} $r \leftarrow \text{POWER-REM}_3(a, (n-1)/2, m)$
8. \hspace{0.5cm} return $(r^2 \mod m) \cdot a \mod m$

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 Inf2B).

If you did not take Inf2B ... read all of the ADS part of Inf-2B.

**Problems**

1. Analyse the asymptotic worst-case running time of the three $\text{POWER-REM}$ algorithms.
   
   $\text{Hint:}$ The worst-case running time of $\text{POWER-REM}_1$ and $\text{POWER-REM}_2$ is $\Theta(n)$, and the worst-case running time of $\text{POWER-REM}_3$ 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]).