COMP2054
Spring Semester
Algorithms Data structures & Efficiency (**ADE**)

Introduction

Andrew Parkes (Room C78)
andrew.parkes@nottingham.ac.uk
http://www.cs.nott.ac.uk/~pszajp/

"ADE": Algorithms Data structures & Efficiency

- Covers
  - **A**lgorithms,
  - **D**ata Structures, and
  - **E**fficiency
- **Algorithm:** step-by-step procedure for solving a problem in a **finite** amount of time.
  - No "MAGIC steps" allowed! (No "Zeno machines")
- **Data structures** : lists, binary trees, heaps, Hashmaps, graphs, etc.
- **Efficiency**: We will start with methods to analyse, classify and describe the efficiency of algorithms
  - Needed in order to be able to select "best algorithms"

The main topics are, as it says on the tin, a bunch of algorithms and data structures, and the main emphasis being on Efficiency.
The slide gives a rough list of the topics.

Firstly, as you know an algorithm is just a fully specified and implementable procedure to solve something in a finite amount of time.
Needs to be fully specified – so implementable on a standard machine.

It needs to finish in a finite number of steps – not allowed "Zeno machines" that do an infinite number of steps in a finite time (look them up if interested.)
Data structures will include many standard structures, some of these you might have already seen, but will be doing them from the viewpoint of EFFICIENCY.

Also, we will start with the language of how to talk about and describe the efficiency of algorithms – that is the "Big-Oh family".

This includes "big-oh" but we will do it in more detail and with more motivation then you might have seen before, and also doing members of the family that you have probably not seen.

Rough Comments

- On average, the ADE module is less formal than the COMP2065-IFR. However:
  - Some portions (definitions and proofs) are formal – and should be treated as such
    - (Optional: think how to convert them to Lean.)
  - Some portions – interpretations and "every day usage" are less formal
    - E.g. targeted to what would a software development team want to know

First a comment about the style.

It will be generally quite different from Thorsten's IFR module.

Some portions will be formal, and then we could think about how to express at least parts of them, or some simpler cases in Lean.

Though as we have seen, formally representing things tends to require a lot of setting up of machinery.

This is natural as the point of correctness is that a proof is actually a proof, and can be mechanically checked, and then know for certain that nothing was missed.

However, many parts are rather less formal – as the interpretations and usage would be designed for everyday usage.

ADE Assessments

- 25% "standard coursework"
  - THREE in-class **CLOSED BOOK IN-PERSON** exam-conditions tests of 30 minutes during the Monday lab (either written or on Moodle).
  - Final mark based on "best 2 of 3" !
  - Dates: To Be Confirmed (TBC). Provisionally:

    C/W 1: Mon 26 Feb
    C/W 2: Mon 18 March or Mon 25 March
    C/W 3: Mon 13 May

    Put these dates in your diary! There are no "late submissions" – but you ask for an EC for missed tests, and can miss one anyway as it is best 2 or 3
- 75 % ExamSys/written exam
  - Summer exam session (May/June)
  - **CLOSED BOOK, IN-PERSON** !!
  - Two hours ExamSys; possibly with "on-paper" addition

Naturally, you want to know about assessments, so here is a rough guide. Some rough guide as to assessments.
As you know 25% is the module is C/W.

The topics of each C/W will be announced in advance, but will typically be all topics covered up to the Monday lecture in the week before the test and since any previous C/W.
The lab and tutorials before the C/W will support understanding of the lectures and the preparation for the C/W.

Tutorials

- ONE 1-hour tutorial per person
  - ADE sessions will just be some weeks, not all – I will notify you on Moodle and email
  - These are essential for understanding the material, and will help with C/Ws, etc.
  - Please use the hour that is assigned to you on your personal timetable

As from the timetable, we continue with tutorials.

Labs

- ONE 1-hour lab session per person
- Labs will be sessions for doing some formative exercises and getting help as needed in preparation for the C/Ws.
  - ADE labs will just be some weeks - not all – I will notify you on Moodle and email
- Please use the hour that is assigned to you on your personal timetable

Similarly for the lab hours.

Email Etiquette

- **Only send email from your University account**! (Except in exceptional circumstances).
  - Only this allows staff to know who is really sending. We cannot discuss anything with random external email addresses as they may not be you.
  - Such emails may well just be treated as spam and deleted
- Do not expect an immediate reply – 3 working days is the norm. After that, send a gentle reminder.
- Include the module code in the subject line
  - This is a good practice for all modules. Staff teach multiple modules; hence writing "I need help with your module" will just cause delays.
  - To check which emails need to be answered, I may well search by the module code.
- Include all relevant details. E.g. lecture and slide number, tutorial question, etc.
  - If it is about personal process, or scores, etc, then include your student ID number.
  - Please do not just write "Help, I'm stuck". It does not help us to help you !
  - Often the process of fixing the details leading to answering the question

Please note that the "etiquette" is to

- make communications more efficient for both staff and students

- might also be considered as part of training in being a "professional"

Study Skills Suggestion

Personal View:

- Often problems arise from going too fast
- Likely to often need to backtrack or start again
  - Backtracking is normal; do not see it as a failure
- Learn to seek chances to identify when need to "fix a bug in your thinking"
  - Allows to make better overall progress
  - Rebuilding 'internal models' is positive!
    - (The main goal of "real learning"?)

Just a note that I like to repeat.

My experience talking with people in labs is that there is often a desire to go as fast as possible, and a reluctance to backtrack and do some portions again.

However, this often leads to people getting stuck on something when the cause of the trouble arises from a misunderstanding of a much earlier topic.

This happens to me personally all the time.
Hence, never be afraid to just backtrack, or even start again on some material. Think of it as a positive experience, as fixing a bug in thinking can mean rebuilding some internal understanding of a topic and this is a sign of real learning – not the memorisation of facts.

As I said, I have to do this all the time myself ☺.

"Exercises" & Hints

- Will often put "**Exercise**" in slides
  - Please do them! At least think about them
  - They are not assessed, but are vital
  - They might well be a vital part of an exam question
- Hints:
  - If stuck because of 'algebra' then "generate your own examples" simply insert small numbers. E.g. put a=2, etc.
  - **Strongly encourage to start with the very simplest examples and work 'upwards'**
    - Many problems arise from reluctance to focus on the basics first

In terms of study methods, also please note that I will give exercises in the slides, not assessed, but recommend that you try them.
Another trick that I find helps me is just to really not be afraid to try very small examples and then thoroughly understand them.
As before, many problems seem to arise from a real focus on the simplest examples.
Hence, I will often include very simple examples.
Often the answer will be obvious, but they are designed to illustrate a method, and doing so in a context that is free of extra secondary details.

What are Algorithms?

- Algorithm: a well-defined step-by-step procedure for solving a problem in a finite amount of time.

For example:

- A recipe for cooking pizza
- A set of instructions for assembling a piece of furniture
- A procedure for finding web pages containing keywords
- A procedure for computing n! given n
- Sorting and searching algorithms
  - E.g. how to sort a pack of cards.
- A deep learning algorithm for deciding whether to grant someone a mortgage

So first of all some quick reminders.
Examples of algorithms.
Part of the reason for saying this is that in the real world non-technical people are often confused by what algorithms are, and you might want to be able to explain it to them.

Personally, I find that a useful real-world example of an algorithm is how to sort a pack of cards.
It can be interesting to try asking people outside of computer science how they do this, and then see if you can recognise their algorithm as a standard sorting algorithm.

What are Data Structures?

- Ways in which to store data in a well-defined and structured fashion to allow algorithms to:
  - make better use of it
  - maintain it more easily as the data changes

We will need to also define "better" and "more easily"

Data Structures in Everyday Life? ...

Data structures are "as it says on the tin" ways to store data in a structured fashion.
Why do we need anything more than an array?
Well the aim is that storing data in structured fashion allows to make better use of it.
Also, in most circumstances data changes and so it is very important that we can easily retain the structure whenever we need to change the data.
This aspect can often be forgotten, but is usually vital.

A large part of this part of the module is to be able to make well-defined formal statements about "better" and "more easily".

So a quick reminder of data structures in every day.

Data Structure Example: Dictionary

- Organising the data correctly helps to find it
- Predates computers by thousands of years
- Good for lookups, but bad for maintenance
  - Deletion leaves holes
  - Insertion is physical cut-and-paste
    - E.g. why a thesis might be double-spaced

Picture from https://en.wikipedia.org/wiki/Dictionary

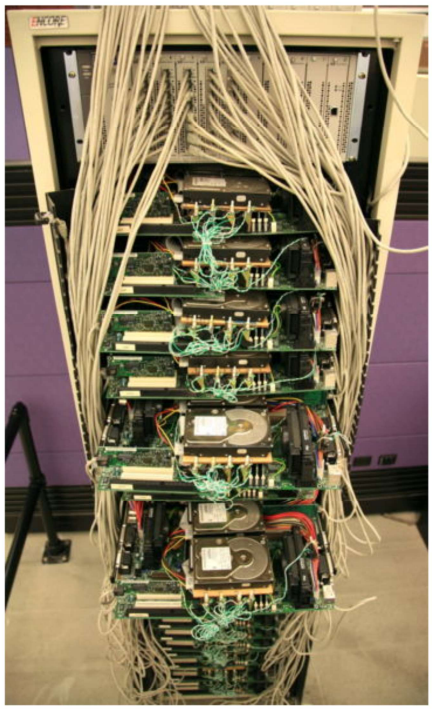
TA: Picture of a large old physical dictionary

A physical dictionary is an ancient data structure. The wiki article says the first known one was from over four thousand years ago.
They are good for lookups.
Not so great for maintenance of the data.
If you want to delete an entry it might leave a hole.
Insertion is hard – the term of cut-and-paste that we use so much now originated, I think, from a true physical cut and paste.
You write the new data on some paper and cut it out and physically paste it in a gap.

If you look at old PhD dissertations you will often find that they are double spaced. This is because they were typed, and it took too long to do corrections by retyping, so would leave space to paste (glue) in new lines.

Data Structure Examples:

- Any guesses as to what this is?

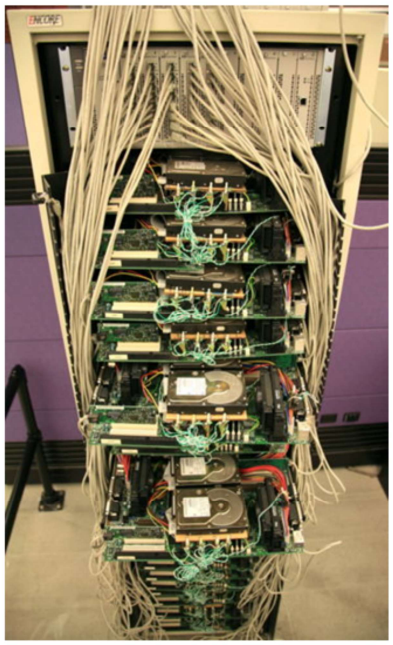


TAG: Picture of a rack computer with about 8 motherboards, and lots of wires

As it says – what might you think this is?
Obviously, a real hack of a machine, and so clearly did not lead to anything important.
Except it did.

Data Structure Examples: (ans)

- Any guesses as to what this is?
- Google's first (production) server



From https://en.wikipedia.org/wiki/History_of_Google

TAG Same picture as previous slide.

Google is a now a massive user of algorithms and data structures, and electricity!

Aims of the ADE Module

Aim: understanding of issues involved in program design; good working knowledge of some common algorithms and data structures

Objectives:

- design data structures and algorithms which express this functionality in way that makes **efficient** use of
  - time
  - memory
  - **(and so indirectly: grid electricity usage, and battery usage)**
- be able to evaluate a given implementation in terms of its **efficiency**

So coming back to the aims of the ADE portion.

Noting that when in the middle of technical details it can help to be able to keep in mind why we are learning these topics.
We care about efficiency.
We want to make good usage of time – faster algorithms when possible.
But also in real machines we can care about memory usage.
A very important reason for this is that every machine operation uses some energy. So greater efficiency reduces electricity usage.

Why care about efficiency?

Beware:

- Recent history is that hardware improved rapidly
  - This compensated for many poor programs
- 3GHz desktop with 2GB RAM on a small "first-year-style" program
  - almost any implementation will work 'instantly'

These might give a false sense that efficiency does not matter.

The need for efficiency can be obscured when you have a computer that is very fast for what you are doing.

Why care about efficiency? (batteries!)

But:

- Clock speeds are 'stuck'
  - Not all programs can exploit multi-core, GPU, clouds, but must run on the single core
- Mobile devices are much less powerful
- Also 'inefficient' consumes more energy
  - **poorly written apps will drain the mobile device battery**
  - **2% (?) of world electricity goes in server farms**

However, consider what happens with less powerful of battery operated machinery.

Not it becomes more important for algorithms to be efficient. The machines are slower and we need to save battery.

On the big side, a lot of electricity is used for running server farms.
Imagine you could make algorithms at Google run 10% faster, then they are likely to pay you a lot of money – and help save the planet as well.

Problem Oriented Approach

- The module has 'maths' and 'theory'
- But these are designed to help with solving real problems/tasks
- What kinds of tasks? …

Well, what kinds of problems can we consider in this ADE part.

Here are some examples.

Example task 1

- "What method would you use to look up a word in a dictionary? (Correctly and efficiently)"

A very common question of just looking up entries.

Example task 2
(optional offline exercise)

- "You are given a list of numbers. When you reach the end of the list you will come back to the beginning of the list (a circular list).
- Write the most efficient algorithm to find the minimum number in this list.
- Find any given number in the list.
- The numbers in the list are always increasing but you don't know where the circular list begins, e.g.: 38, 40, 55, 89, 6, 13, 20, 23, 36."

Consider the example given here. I will not go through it now, but notice it is about lists, and they are sorted, but in a strange fashion, in that they are sorted from some unknown point in the list.

Example task 3
(optional offline exercise)

- "Write a function f(a, b) which takes two character string arguments and returns a string containing only the characters found in both strings in the order of a. Write a version which is order N-squared and one which is order N."

Another somewhat simpler case, of string processing. Can you some it is So-called order-n and then also improve it to order N.

Example task 4

- "Describe the algorithm for a depth-first graph traversal."

A direct task – and standard "bookwork".

Example task 5

- "Tree search algorithms. Write BFS and DFS code, explain run time and space requirements. Modify the code to handle trees with weighted edges and loops with BFS and DFS, make the code print out path to goal state."

Another example. And this is again something we (may) cover in the ADE half.

Example task 6
(optional offline exercise)

- "There is an array A[N] of N numbers. You have to compose an array Output[N] such that Output[i] will be equal to multiplication of all the elements of A[N] except A[i]. For example Output[0] will be multiplication of A[1] to A[N-1] and
  Output[1] will be multiplication of A[0] and from A[2] to A[N-1].
  Solve it without division operator and in O(n)."

Yet another example. This time again about finding an algorithm.

Example task 7

- How long it would take to sort 1 billion numbers?
- Come up with a rough estimate.
- "Billion" is 10^9 (US and modern UK usage)
  In old UK usage it was 10^{12}

- **EXERCISE: Offline – try to estimate it using "back of an envelope" style**

Finally a task that is about developing a general feeling for run times.

Suppose you have to sort 1 billion numbers – without actually trying to do it, try to do an estimate of how long you think it might take.

Note this is not after a precise answer, but just a ball-park rough estimate. E.g. the order-of-magnitude – meaning to the nearest power of ten.

Again this is offline question.
Note "billion" in the UK used to be 10^12.
See https://en.wikipedia.org/wiki/Long_and_short_scales for the long history, and noting that it varies between languages and countries, even when the same spelling is used.

Previous "Example tasks" are irrelevant?

- **If you think the previous were irrelevant then be aware:**
- **They were taken (mostly) from "Google Interview Questions: Software Engineer"**
  - (http://blog.seattleinterviewcoach.com/2009/02/140-google-interview-questions.html#software_engineer but page is no longer active)

And if you think these were irrelevant, you should know that I got them from a defunct website of google interview questions.
ADE topics do arise in job interviews!

Interview Question

- Question: "There is an array A[N] of N numbers. … Solve it … in O(n)."
- What happens if your answer is
  - "what does 'O(n)' mean?"
  - ☹

This is just a bit of motivation for learning the language of efficiency well.

Rough Context within "Programming"

- "Low-level" Programming (Modules in C, Assembly)
  - for, while loops etc
- **ADE: Algorithm Efficiency**
  - **trees, search algorithms, etc**
  - **code: "small but difficult" (each line might be 'hard')**
- Object-oriented design (Java / C++ modules)
  - software reuse, inheritance, etc
- Software systems architecture design (DMS?)
  - designing software components and interactions, etc
  - code: "easy but large" (design is hard, but each line tends to be 'easy')
- Algorithm Correctness – e.g. COMP2065-IFR
  - Proving correctness and/or termination
  - Requires "formal reasoning about programs" – VERY challenging for real-world programs

Also, to give some context of the place of the ADE portion with respect to other CS topics.
It is not the low-level programming, nor is it (mostly) the software engineering of object-oriented methods, or large scale design,

But something in between.

Rough Contents of the ADE portion

Basics/Theory:

- **Algorithm analysis, "big-Oh" and its family**
- (Revision) Abstract vs. Concrete Data Types
- (Revision?) Stacks, queues, lists, trees.
- Recurrence relations. "Master Theorem".

Sorting Algorithms:

- Bubble-sort, …, mergesort, quicksort, "stability", efficiency

Standard data structures

- Priority queues, Heaps, Hash tables

Search:

- Maps, hashmaps, binary search trees, balanced vs. unbalanced trees

Graphs & Algorithms:

- Minimum Spanning Trees, "dynamic programming", Shortest Paths (Dijkstra's algorithm, and Floyd-Warshall),

Here we just seem some typical topics.

Some of these you will have seen before but the aim is to be able to do them with a much greater emphasis on the analysis of the efficiency.

Towards the end of the spring semester will be doing graphs, and part of that will be shortest paths, and also methods based on something called "dynamic programming" – which is a technique that can lead to very efficient solutions.

Maths needed includes:

- Properties of inequalities:
  - Reasoning about efficiency involves comparisons
- Properties of logarithms and exponents
  - Used to describe efficiency
    - (Generally don't need sin/cos etc.).
- Arithmetic and Geometric Series
- (Function fitting
  - "Regression" a.k.a. "trend lines in Excel")
- Proof by induction
- (Simple) Propositional and predicate logic

(A lot of this should be "revision".)
Tutorials and other material will support these skills.

Here is a list of topics needed for the mathematics in the ADE portion.
Many will be already familiar with some or all of these.

However, we have self-study or tutorials for those that need extra help in any of the topics.

For example, many will have done exponents and logarithms, but it might have been a long time ago, and might have been forgotten.

Some people might never have done them – but they are absolutely essential for ADE

Maths needed includes: 'comparisons"

- **Properties of inequalities**:

$a < b$ implies $2a < 2b$
$a \leq b$ implies $2a \leq 2b$
$a \leq b$ implies $a + c \leq b + c$

$a < b$ implies $-a > -b$
$a \leq b$ implies $-a \geq -b$

Etc.

Hopefully most of you find this totally trivial.

Maths needed includes: (logs!)

- **Properties of exponentials**: (examples)

  $a^{(b+c)} = a^b a^c$ $\quad\quad$ $a^0 = 1$

  $a^{bc} = (a^b)^c$ $\quad\quad$ $a^b / a^c = a^{(b-c)}$

- **Properties of logarithms:**

  $b = a^{\log_a b}$ ("definition of 'log base a' ")

  Offline future exercises:
  derive each of the following

  $b^c = a^{c*\log_a b}$

  $\log_b(xy) = \log_b x + \log_b y$

  $\log_b(x/y) = \log_b x - \log_b y$

  $\log_b x^a = a \log_b x$

  $\log_x a = \log_x b \log_b a$

  Quick (offline) self-test:
  Without a calculator, **what is $\log_8(2)$?**

  (Yes, we do need this!)

Some of these might be confusing even to those that have done logarithms.

We need to be able to use logarithms with different bases.

A tutorial is available for ensuring the needed skills with exponentials and logarithms.
Otherwise, do ask for help in labs or tutorials!!

Maths: "Geometric and Arithmetic Series"

- **(Offline) Without using a calculator What are:**

"Geometric"

1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 + 256 + 512

"Arithmetic"

1 + 3 + 5 + 7 + 9 + 11 + 13 + 15 + 17 + 19

Many of you will already have seen these, but material will be provided for anyone that needs a refresher.

Even if you did them in first year, it might not be that everyone taking ACE did those modules.

Summary

- Main points of ADE part:
  - Develop habit of thinking about efficiency
    - E.g. Develop skills (mostly 'big-Oh' and family) to describe efficiency, and reason about efficiency
- But note:
  - This does not mean you always have to write the most efficient program
  - The skills (big-Oh family) will not be a panacea; they do not answer all questions about efficiency (but this does not mean they are irrelevant)

So in summary the ultimate aim of the ADE is to encourage a habit of thinking about efficiency.
And being able to talk about it, and having good judgement about how to use this is practical decisions.

Implementing the absolute most efficient algorithm possible might not be the best and most practical choice, but efficiency should always be part of the decision making.

Notes

- Last year there was a module with code COMP2054
  - it was the Autumn portion of COMP2009-ACE – it had some overlap, but was not the same as this module !
  - Do not rely on any previous exams for "COMP2054"