COM1009
Introduction to Algorithms and Data Structures

Week 6 Revision Lecture

Read your notes and the previous slides
Do the practice quiz more than once

Aims of today's lecture

- To remind you how the module is assessed
- To remind you of the module's formal learning outcomes
- To indicate which material is related to which outcome(s)
  - Which things might be covered in the threshold assessment?
  - Which things might be covered in the end-of-module exam?

Two-part assessment

- Threshold Assessment (Online Quiz, Week 7)
  - Checks you've met the module's learning outcomes to a minimum standard
  - Only covers the basic material that everyone ought to know and understand. Does not cover any advanced material.
  - Passing this assessment means you automatically pass the entire module (with a grade of 40%)
- Grading Assessment (End of Year Exam)
  - Covers only the more advanced material. Asks you to apply what you've learned.
  - This is your chance to raise your grade from 40 as high as you can. You do not need to "pass" this exam separately.

▶COM1009 Learning Outcomes
(taken from the com1009 module description page)

- You should be able to
  - [LO1] appreciate what constitutes an efficient and an inefficient solution to a computational problem;
  - [LO2] analyse the efficiency of an algorithm;
  - [LO3] evaluate and choose data structures that support efficient algorithmic solutions;
  - [LO4] identify and apply design principles such as greediness, divide and conquer and dynamic programming in the design of efficient algorithms;
  - [LO5] describe efficient algorithms for fundamental computational problems, along with their computational complexity

[LO1] Efficient/inefficient solutions

Two things here!

- what does efficient mean?
  - uses relatively few resources to run
  - in particular, low worst-case runtime and/or runs in place
  - inefficient = "not efficient"
- what is a solution?
  - solves a problem (what is a problem?)
  - does it correctly

[LO2] Analyse efficiency

- Efficiency in this context means runtime
  - worst case, best case
  - number of elementary instructions executed in finding the answer
- You should know things like:
  - "for i = 1 to n" involves $\theta(n)$ iterations
  - "exchange x and y" takes constant time
  - "the height of a binary tree with n leaves is $O(n.\log n)$"
- Threshold: knowing these sorts of things is expected
- Exam: being able to prove them is more advanced

▶[LO3] Evaluate and choose data structures

- If someone says they want an algorithm that has certain properties, what would be a good choice of data structure?
  - I need to sort data quickly using only a constant amount of extra memory: *"I suggest implementing a heap. Then you can use heapsort to sort data efficiently and in place"*
  - I'll never need to store more than 100 pieces of data, and I want to insert and extract values in constant time: *"Either a queue and a stack might work (you know how much data needs to be stored, so you should be able to implement it using an array). Do you want it to be last-in-first-out?"*

[LO4] Identify/apply design principles

- What approach should I use to sort these 7 cards?
  - *"This is a really simple problem and you don't have a lot of data to worry about – why not use an incremental approach? Simply insert the cards into your hand one at a time."*
- I need to find the best way to do something, but doing so involves solving various overlapping subproblems – fortunately my problem has optimal substructure.
  - *"That's useful to know - it sounds to me like dynamic programming ought to work well. I suggest you work out the solutions to the subproblems, starting with the smallest, and keep a note of the solutions as you go along. That way you won't need to recalculate the results over and over again."*

▶[LO5] Describe efficient algorithms, including their computational complexity

- My favourite algorithm
  - uses this design strategy to do ….
  - has this worst case runtime …
  - has this best case runtime …
  - does/does not run in place…
- I see! But what do you mean by "in place"?
- And what's that strange asymptotic notation you're using?

What about asymptotic notation?

- Asymptotic notation is not mentioned in the learning outcomes
- BUT … *you need it to express runtime* [LO2, LO3, LO5]
- For threshold you ought to know
  - what it means
  - how to use it
  - how the different symbols are related to each other e.g. if $f = \theta(g)$, then $f = O(g)$ and $f = \Omega(g)$
- More advanced:
  - proving that $f = \theta(g)$ for some given f and g

What about correctness proofs?

- Correctness is not mentioned in the learning outcomes
- BUT … *being "correct" is part of being a "solution"* [LO1]
- For threshold you ought to know
  - what it means for a solution to be correct
  - that correctness can be proven for standard algorithms
  - what a loop invariant is and how it's used (general principles)
- More advanced:
  - inventing a suitable loop invariant
  - actually proving that an algorithm is correct

▶TOPIC SUMMARIES

▶Topic 1: Algorithms

[LO1, LO2]

- Algorithms
  - What is an algorithm? What's a good one?
  - What is a problem? What is an instance of a problem?
- Correctness
  - What is it? How do we prove an algorithm is correct?
  - Loop invariants – what are they?
    [Advanced: formal proof that InsertionSort is correct.]
- Runtime analysis
  - What is runtime and how do we work it out?
  - RAM machines, pseudocode, problem size

▶Topic 2: Runtime, asymptotic notation

[LO2, LO5]

- Best and worst case runtimes
  - e.g., InsertionSort: best is linear, worst is quadratic
  - why we usually focus on worst case
- Comparing runtimes
  - Logs vs polynomials vs exponential functions
- Asymptotic notation (o, O, θ, Ω, ω)
  - Informal meanings (drawing graphs) and examples [Advanced: formal definitions and proofs]
  - Analogy with (<, ≤, =, ≥, >)
- Using the notation, e.g. θ(n). θ(log n) = θ(n.log n)

▶Topic 3: Elementary data structures
[LO3]

- Dynamic sets
  - Keys and satellite data
  - Typical operations on dynamic sets
- Stacks
  - implementation using arrays; runtimes
- Queues
  - Implementation using arrays; runtimes
- Linked lists
  - Implementation using pointers; runtimes

Topic 4: Divide and conquer

[LO2, LO3, LO4, LO5]

- divide-and-conquer vs. incremental design
  - Divide the problem; conquer the bits; merge the results
- MergeSort
  - The Merge operation and its runtime
  - [Advanced: proof of correctness for Merge]
  - Runtime of MergeSort
  - [Advanced: Solving recurrence relations using recursion trees]
- Comparison of InsertionSort and MergeSort
  - What "in place" means

Topic 5: Heapsort

[LO2, LO3, LO5]

- Heaps
  - Max-Heap: binary tree, parent never smaller than child
- Switching viewpoints between “array” and “binary tree”
- Heapsort
  - Runs both fast and in place
  - Max-Heapify and Build-Max-Heap
- Calculating the runtimes
  - Using the fact that $h = O(\log n)$
    [Advanced: proving this is true]

▶Topic 6: Lower bound for comparison sorts, and how to beat it [LO2, LO5]

- *NP-completeness, P vs NP – there are no questions about these in either the threshold assessment or the exam for this module* (not on module syllabus)
- Comparison sorts
  - Require $\Omega(n.\log n)$ time in worst case
    [Advanced: proof of this using decision trees]
  - $\log(n!) = \theta(n.\log n)$
    [Advanced: proof of this equation]
  - HeapSort and MergeSort are optimal comparison sorts
- CountingSort and Radix Sort, and when/why they're faster
  - What are their runtimes
  - What is a stable sorting algorithm?

▶Topic 7: Dynamic programming

[LO4, LO5]

- Different approaches to computing Fibonacci numbers
  - Fib(n) grows exponentially
    [Advanced: proof of this]
  - Computation time depends on the algorithm used
- Dynamic programming
  - Overlapping subproblems
  - Optimal substructure
  - Used for optimisation problems
  - Bellman equations
- Example: rod-cutting

▶Still to come in the module

- I hope to cover all of these (time permitting):
  - Topic 8: Greedy algorithms
  - Topic 9: Randomisation
  - Topic 10: Binary search trees and AVL trees
  - Topic 11: Elementary graph algorithms
  - Topic 12: Minimum spanning trees

Good luck in the quiz!