

Module 1: Algorithm Design and Analysis

Stage		1				
Semester		1				
Module Title		Algorithm Design and Analysis				
Module Number		1				
Module Status		Mandatory				
Module ECTS Credits		5				
Module NFQ level		9				
Pre-Requisite Module Titles		None				
Co-Requisite Module Titles		None				
Capstone Module?		No				
List of Module Teaching Personnel		Eamonn Nolan				
Contact Hours				Non-contact Hours		
42				58		
Lecture	Practical	Tutorial	Seminar	Assignment	Placement	Independent Work
24	18			30		28
Allocation of Marks (Within the Module)						
	Continuous Assessment	Project	Practical	Final Examination	Total	
Percentage Contribution	50			50	100	

Intended Module Learning Outcomes

On successful completion of this module the learner will be able to:

1. Use mathematical techniques to analyse the running time and space use of algorithms.
2. Compare the efficiency of algorithms over different domains based on asymptotic analysis of running times
3. Analyse and calculate the complexity of algorithms using recurrence relations.
4. Apply appropriate design strategies when implementing algorithmic solutions to computational problems.
5. Implement tree and graph algorithms efficiently and analyse their performance in particular problem domains
6. Classify computational problems P, NP, or NP-completeness
7. Identify and implement approximation algorithms for some NP-complete problems
8. Analyse the performance of algorithms that perform parallel computation with respect to sequential solutions
9. Assess the feasibility of parallelising given problems, design functional parallel algorithms, and determine and interpret their performance.

Module Objectives

This module aims to equip the learner with the skills to design, analyse, compare and implement a range of important and commonly used algorithms across a number of problem domains. Learners learn the relationship between data structures, algorithms, and the problem which is to be solved. Mathematical methods of analysing algorithms are introduced and worked out in example. Classic algorithms are used to demonstrate problem solving paradigms, design strategies, and data structures. Learners are taught how to classify algorithms using asymptotic analysis and use this as a tool for comparing algorithms both with other algorithms and theoretical lower bounds for running times. An understanding of computation tractability, NP-Completeness, and the impact of these topics on theoretical computer science is a significant goal. Learners are taught different approaches to deal with intractability. Parallel algorithm design including identifying problems that do or do not lend themselves to parallelisation are presented. Learners are able to design functional parallel code that handles and/or avoids common parallelisation problems and issues. The theoretical prediction and practical interpretation of parallel performance metrics must be understood.

Module Curriculum

Mathematical Foundations

Abstraction and Abstract data types / Solving recurrences and Summations / The Master Method / Asymptotic Analysis / Optimality / Case studies including complexity analysis of algorithms such as quadratic sorting methods, heapsort, linear-time sorts and searches / Hash tables and chaining and probing

Algorithm Design strategies

Divide and Conquer / Greedy strategies / Recursive solutions / Tail Recursion / Case studies of each design strategy to include optimal sorting: Quicksort, Mergesort / Dynamic Programming

Tree and graph algorithms

Representation of trees and graphs / Basic algorithms / adjacency lists, matrices / Breadth-first search / Depth-first search / Minimum spanning trees / Algorithms such as Kruskal, Prim, Dijkstra, Bellman-Ford and Floyd-Warshall / Dealing with negative cycles / Tree balancing

Testing, Presenting and Interpreting results

Error analysis / Proper testing strategies to include appropriate domain and range / Randomness / Timing / Machine arithmetic and errors / Data presentation and analysis / Extrapolation

Tractability

NP-completeness theory / Practical ramifications of the theory / Cook's Theorem / NP-Hard problems / Approximation algorithms / Hamiltonian cycles / The travelling salesman problem

Parallel algorithm design and analysis

Parallelisable problems / embarrassingly parallel / fine-coarse granularities / Shared memory / Distributed memory / Race conditions / Deadlocks / Mutexes / Speedup / Efficiency and overheads / Examples to include linear algebra problems and/or other classical mathematical/physical science problems

Reading Lists and other learning materials

Recommended Reading

Cormen, T.H., 2009. *Introduction to algorithms*, Cambridge, Massachusetts; London: The MIT Press.

Sedgewick, R., 2012. *An introduction to the analysis of algorithms* 2nd ed., Reading, Mass. ; Wokingham: Addison-Wesley.

Secondary Reading

Sedgewick, R., 2011. *Algorithms* 4th ed., Boston, Mass. ; London: Addison-Wesley.

Knuth, D.E., 2011. *The art of computer programming. Volumes 1-4a*, Boston, Mass.; London: Addison-Wesley.

Levitin, A., 2012. *Introduction to the design & analysis of algorithms*, Boston [etc.]: Pearson Education.

Grama et al., 2003. *Introduction to Parallel Computing*, Addison-Wesley

Module Learning Environment

Accommodation

Lectures are carried out in class rooms / lecture halls in the College. Lab tutorials are carried out in computer labs throughout the Campus. All have the software required to deliver the programme.

Library

All learners have access to an extensive range of physical and electronic (remotely accessible) library resources. The library monitors and updates its resources on an on-going basis, in line with the College's Library Acquisition Policy. Lecturers update reading lists for this course on an annual basis as is the norm with all courses run by Griffith College.

Module Teaching and Learning Strategy

Each week involves both classes and practical laboratory sessions

Classes are used to deliver theoretical content and may be supported by online delivery of notes, examples, and web resources. Algorithmic techniques are exemplified with specific case studies of classic algorithms and data structures. These algorithms are worked in example and presented in explanation and with pseudo-code. Mathematical content is presented in theory and worked in example.

Laboratory Practicals are used to reinforce design strategies and problem solving approaches. These sessions are used to implement and test algorithms and their associated data structures and provide practical support to theory covered in lecture.

Module Assessment Strategy

Continuous assessment is based on a combination of some of the following:

- Programming assignments
- On-going labwork
- Class tests
- Practical tests
- Oral examination

Element	Weighting	Type	Description	Learning Outcomes Assessed
1	20%	Programming Assignment	Implementation of a Data Type from the module	4,5
2	30%	Programming Assignment	Implementation of an Algorithm from the module	3-5
3	50%	Closed Book Examination	End of Module Examination	1-3,6-9