Unit 19: Data Structures & Algorithms

Unit code	**T/618/7430**
Unit level	**5**
Credit value	**15**

Introduction

Knowing how to implement algorithms and data structures that solve real problems, and knowing the purpose, complexity and use of algorithms is part of an essential toolkit for software engineers. An algorithm is a sequence of instructions used to manipulate data held in a structured form and together with data structures constitute design patterns for solving a diverse range of computer problems, including network analysis, cryptography, data compression and process control.

This unit introduces students to data structures and how they are used in algorithms, enabling them to design and implement data structures. Students are introduced to the specification of abstract data types and will explore their use in concrete data structures. Using this knowledge, students should be able to develop solutions by specifying, designing and implementing data structures and algorithms in a variety of programming paradigms for an identified need.

Among the topics included in this unit are abstract data types specification, formal data notations, data encapsulation, complex data structures, programming language implementations using handles, pointers, classes and methods, algorithm types, data structure libraries, algorithm complexity, asymptotic testing and benchmarking.

On completion of this unit, students should be able to identify program data requirements, specify abstract data types using a formal notation, translate into concrete data structures and be able to develop, using a programming paradigm, different sorting, searching and navigational algorithms that implement complex data structures and evaluate their effectiveness. As a result, students will have developed skills such as communication literacy, critical thinking, analysis, synthesis, reasoning and interpretation, which are crucial for gaining employment and developing academic competence.

Learning Outcomes

By the end of the unit students will be able to:

LO1 Examine abstract data types, concrete data structures and algorithms

LO2 Specify abstract data types and algorithms in a formal notation

LO3 Implement complex data structures and algorithms

LO4 Assess the effectiveness of data structures and algorithms.

Essential Content

LO1 Examine abstract data types, concrete data structures and algorithms

Abstract Data Types (ADTs):

Specification of ADTs with formal notation.

Data structures:

Array, set, stack, queue, list, tree, types, e.g. active, passive, recursive.

Algorithm types:

Recursive, backtracking, dynamic, divide and conquer, branch and bound, greedy, randomised, brute force.

Algorithms:

Sort, insertion, quick, merge, heap, bucket, selection, search linear, binary, binary search tree, recursive, e.g. binary tree traversals, find path, travelling salesman.

LO2 Specify abstract data types and algorithms in a formal notation

Design specification:

Specify ADTs using formal notation, e.g. ASN.1.

Use non-executable program specification language, e.g. SDL, VDM.

Issues, e.g. complexity in software development, design patterns, parallelism, interfaces, encapsulation, information hiding, efficiency.

Creation:

Pre-conditions, post-conditions, error-conditions.

LO3 Implement complex data structures and algorithms

Implementation:

Apply algorithms, logic and data structures, multidimensional arrays, linked lists, stacks, queues, trees, hash table, heap, graph algorithms, sorting, searching, tree traversal, list traversal, hash functions, string manipulation, scheduling and recursive algorithms, using handle, pointer, class, methods, using an executable programming language.

Create logical and maintainable codes.

Testing and debugging:

Testing code to ensure it is secure and can handle user errors, identifying and creating test scenarios, applying structured techniques to problem solving, debugging code, understanding the structure of programmes to identify and resolve issues.

LO4 **Assess the effectiveness of data structures and algorithms**

Use of data structure libraries (DSL):

Limitations of DSL, manual selection of data structures, theoretical analysis, asymptotic analysis, size of N, Big O notation.

Algorithm effectiveness:

Run time benchmark, compiler/interpreter dependencies, resource usage, degree of parallelism, time, space, power performance, efficiency of garbage collection.

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction
LO1 Examine abstract data types, concrete data structures and algorithms		
P1 Create a design specification for data structures, explaining the valid operations that can be carried out on the structures. **P2** Determine the operations of a memory stack and how it is used to implement function calls in a computer.	**M1** Illustrate, with an example, a concrete data structure for a First in First out (FIFO) queue. **M2** Compare the performance of two sorting algorithms.	**D1** Analyse the operation, using illustrations, of two network shortest path algorithms, providing an example of each.
LO2 Specify abstract data types and algorithms in a formal notation		
P3 Specify the abstract data type for a software stack using an imperative definition.	**M3** Examine the advantages of encapsulation and information hiding when using an ADT.	**D2** Discuss the view that imperative ADTs are a basis for object orientation offering a justification for the view.

Pass	Merit	Distinction
LO3 Implement complex data structures and algorithms		
P4 Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem. **P5** Implement error handling and report test results.	**M4** Demonstrate how the implementation of an ADT/algorithm solves a well-defined problem.	**D3** Critically evaluate the complexity of an implemented ADT/algorithm.
LO4 Assess the effectiveness of data structures and algorithms		
P6 Discuss how asymptotic analysis can be used to assess the effectiveness of an algorithm. **P7** Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.	**M5** Interpret what a trade-off is when specifying an ADT, using an example to support your answer.	**D4** Evaluate three benefits of using implementation independent data structures.

Recommended Resources

Textbooks

Cormen, T. et al (1990) *Introduction to Algorithms*. MIT Press.

Cormen, T. et al (2002*) Instructor's Manual: Introduction to Algorithms.* MIT Press.

Heineman, G. et al (2009) *Algorithms in a Nutshell*. O'Reilly Publishing.

Larmouth, J. (1999) *ASN.1 Complete*. Kaufman Publishing.

Leiss, E. (2007) *A Programmer's Companion to Algorithm Analysis*. Chapman & Hall.
Sedgewick, R. (1983) *Algorithms*. Addison-Wesley.

Wirth, N. (2004) *Algorithms and Data Structures.* Oberon.

Links

This unit links to the following related units:

Unit 1: Programming

Unit 20: Applied Programming and Design Principles

Unit 30: Applied Cryptography in the Cloud.

