Next Generation Higher National Unit Specification

Algorithms and Data Structures (SCQF level 8)

Unit code: J7DH 48

SCQF level: 8 (24 SCQF credit points)

Valid from: session 2023–24

Prototype unit specification for use in pilot delivery only (version 1.0) June 2023

This unit specification provides detailed information about the unit to ensure consistent and transparent assessment year on year.

This unit specification is for teachers and lecturers and contains all the mandatory information required to deliver and assess the unit.

This edition: June 2023 (version 1.0)

Unit purpose

This unit introduces learners to common abstract data structures. It gives learners an understanding of their operation and use in the implementation of digital data storage solutions. Learners also learn the common algorithms used to manipulate, search and sort data stored in these data structures.

This is a specialist unit, intended for learners with an interest in computer programming. It is particularly suitable for learners who are undertaking an HND in Computer Science or Software Development.

The unit does not require any previous knowledge of data structures, or how to search and sort algorithms. However, learners should understand at least one programming language at SCQF level 7.

On completion of the unit, learners have key fundamental knowledge of the principles behind various commonly used algorithms and data structures. This knowledge will help them with any future programming languages they encounter.

Unit outcomes

Learners who complete this unit can:

1 manipulate data storage formats
2 implement abstract data structures
3 write programs to implement common algorithms for sorting and searching data
4 compare common algorithms for sorting and searching data
5 secure data using appropriate algorithms

Evidence requirements

Learners must provide product evidence. Knowledge is inferred from the product evidence. The product evidence comprises one or more computer programs that cover all the skills components. You can sample specific skills components. For example, you do not need to cover every sorting algorithm. However, you must include programs that require learners to write, test and run code for:

- reading and writing data between different file formats
- at least one built-in data collection structure
- the creation and use of singly and doubly linked lists
- the creation and use of a binary search tree
- the implementation of at least two from the following structures: stacks, queues, deques, and heaps
- the implementation of at least two search and two sort algorithms, including a time and space complexity comparison between them
- the implementation of recursion in a search or sort algorithm
- locating the highest and lowest values within data
- encryption, decryption and compression of data using existing code libraries

Learners can produce their product evidence over an extended period of time in lightly-controlled conditions. Give learners access to learning materials. Evidence produced in lightly controlled conditions must be authenticated. The Guide to Assessment provides further advice on methods of authentication.

The standard of evidence should be consistent with the SCQF level of the unit.

Knowledge and skills

The following table shows the knowledge and skills covered by the unit outcomes:

Knowledge	Skills
Learners should understand: ♦ common data storage formats ♦ data storage format manipulation ♦ built-in data structures ♦ abstract data structures ♦ the implementation of static abstract data structures ♦ the implementation of dynamic abstract data structures ♦ desk-check algorithms ♦ how to compare different search algorithms ♦ how to compare different sorting algorithms ♦ how to use data encryption algorithms ♦ how to use data compression algorithms	Learners can: ♦ interpret common data storage formats ♦ produce code to read, write and update common data storage formats ♦ produce code to transfer data between different storage formats ♦ produce code to access, add, remove and update data within built-in collection structures ♦ produce code that creates singly linked list data structures from first principles ♦ produce code to access, add, remove, and update data in a singly linked list ♦ produce code that creates doubly linked list data structures from first principles ♦ produce code to access, add, remove, and update data in a doubly linked list ♦ produce code that uses binary search tree data structures from first principles ♦ produce code to access, add, remove, and update data by traversing a binary search tree ♦ produce code to implement stacks, queues, deques, and heap structures, using both the abstract linked list and built-in array or list structures ♦ produce recursive functions or methods to solve a variety of problems ♦ perform desk-checking of algorithms ♦ produce code that uses two or more sorting algorithms ♦ produce code that uses two or more searching algorithms

Knowledge	Skills
	Learners can: ♦ produce code to locate the largest and smallest items in a collection of values ♦ select the best search algorithm based on time and space complexity ♦ select the best sort algorithm based on time and space complexity ♦ use existing libraries to produce code, using common cryptographic algorithms ♦ use existing libraries to produce code, using common data compression algorithms

Meta-skills

Throughout this unit, learners develop meta-skills to enhance their employability in the IT sector.

Self-management

This meta-skill includes:

- focusing: sorting and maintaining documentation; attention to detail
- adapting: critically reflecting on own skills; evaluating user experience; self-learning to develop wider skills and extend development beyond requirements and taught content
- initiative: displaying independent thinking; taking ownership of own development, time and learning; satisfying user requirements

Social intelligence

This meta-skill includes:

- communicating: receiving and giving information; establishing user requirements and ensuring an understanding of the brief; storytelling through technical documentation
- collaborating: working in groups to discuss, analyse and formulate a solution to a given problem
- leading: producing independent solutions; providing a walk-through and details of the system

Innovation

This meta-skill includes:

- curiosity: exploring the features and libraries available within a programming language; recognising problems and devising solutions
- creativity: using imagination to meet the needs of the user; comparing existing knowledge to generate ideas; visualising
- sense-making: analysing a programming language; seeing the bigger picture
- critical thinking: logical thinking to ensure a coherent approach and to meet requirements appropriately

Delivery of unit

This unit provides learners with an understanding of how common abstract data structures work and how these can be used to implement various storage solutions. Give learners several problems to solve to allow them to gain a good understanding of the principles involved.

The time required varies depending on the previous experience of individual learners. Based on 120 hours delivery and assessment time, we suggest the following distribution:

Outcome 1 — Manipulate data storage formats
(20 hours)

Outcome 2 — Implement abstract data structures
(30 hours)

Outcome 3 — Write programs to implement common algorithms for sorting and searching data
(30 hours)

Outcome 4 — Compare common algorithms for sorting and searching data
(20 hours)

Outcome 5 — Secure data using appropriate algorithms
(20 hours)

Additional guidance

The guidance in this section is not mandatory.

We encourage you to allow learners to use the data structures and algorithms in different languages — although they can use a single language, depending on the level of their ability.

Content and context for this unit

Manipulate data storage formats (outcome 1)

Teach learners about common formats used for data storage and transfer between systems. They should understand the structure of common file data formats such as JavaScript Object Notation (JSON), comma separated values (CSV) and extensible markup language (XML). Learners must write code that reads data from each of these files, perform updates to it and then write the update to a file of the same or alternative format. For example, they could read data from XML and write it to JSON.

You should encourage learners to explore the features and libraries available within their chosen programming language, to find out what support it offers in relation to data structures and algorithms. They can then compare these to their existing knowledge, to find out if there are better ways of solving problems.

Implement abstract data structures (outcome 2)

Teach learners how to create and manipulate language-specific built-in abstract structures for storing collections of values. These should include simple and multi-dimensional arrays, sets (unique values) and tuples (unchangeable values). Learners should also know how to use hash table (map) structures to store data as key/value pairs. For each of the above data structures, learners must know how to add, locate, remove and update data anywhere within the structure.

Learners must know how to create both a singly and doubly linked list abstract data structure from first principles. They must also know how to add, locate, remove and update elements anywhere within the structure.

Learners must know how to create a binary search tree abstract data structure from first principles. They should be able to traverse it and add, locate, remove and update elements at any point within it.

Learners should know how to apply these abstract data structures by writing code containing stacks, queues, deques and heaps, using both linked lists and appropriate built-in structures. Learners should know how to add, locate, remove and update elements at any point within them.

Write programs to implement common algorithms for sorting and searching data (outcome 3)

Teach learners how recursion works and how recursive functions are structured. They should be capable of applying recursion to common problems. Examples might include factorial or Fibonacci sequence calculations, as well as string operations such as reversal and character counting.

Learners must know how to sort collections of data, both in ascending and descending order, using a variety of different sorting algorithms. They must know how to code common sorting algorithms such as bubble sort, selection sort, merge sort and quicksort. They should understand how each works and demonstrate their application.

Learners must know how to search through data. They should know how to perform linear searches of unsorted data. Learners should also know how to perform the more efficient binary searches on sorted data. They should do this using both iteration and recursion. They should also use binary tree structure searching capabilities. You should also make learners aware of the effects of sorting data before searching within it.

You should teach learners how to find the largest and smallest values in a given list of values.

Learners critically review their program code with a view to increasing its efficiency. This not only tells them how well the code is working, but also offers them insights into any improvements they should apply.

Compare common algorithms for sorting and searching data (outcome 4)

Learners should know how linear searching becomes less efficient as the volume of data increases. They should also know that, while binary searching can solve the problem for large data sets, the additional overhead of initial sorting of data can reduce this.

Although learners do not need to derive the time and space complexity of algorithms and data structures, they should understand what these terms mean and know the different time complexities of accessing, searching, inserting and deleting elements within arrays, stacks, queues, singly and doubly linked lists, hash tables, and binary trees. Learners should also know the different time complexities of the different sorting algorithms (bubble sort, selection sort, merge sort and quicksort). You can use common notation, such as Big O, to describe these.

To consolidate their understanding, you should give learners a range of problems to solve that cover the range of algorithms and data structures taught during the unit. For example, you can give learners a large XML data set from which they must extract some of the data, and search and sort it. They should then store the sorted data in a JSON format and the results of the search in CSV. You should also encourage learners to write multiple versions of their code using different algorithms and/or data structures and compare their efficiency.

There are opportunities for learners to work in groups to discuss, analyse and formulate a solution to a given problem. Learners could then produce independent solutions and compare and contrast.

Secure data using appropriate algorithms (outcome 5)

Learners should understand the need for data encryption. They should learn how to write code to encrypt and/or decrypt data using both symmetric and asymmetric encryption algorithms. You should also teach them about associated techniques such as hashing and compression algorithms. We do not expect learners to use these algorithms from first principles but instead you should teach them how to use pre-written programming libraries.

Approaches to assessment

We expect learners to demonstrate their learning from outcomes holistically. Learners write an application that can take data from at least one file type and apply it to appropriate data structures for analysis through searching and sorting, in line with the evidence requirements. They should then store the analysed data permanently in a suitably formatted file.

If a more individual approach is used, learners could prepare a portfolio of programs created throughout the unit that cover the evidence requirements.

Equality and inclusion

This unit is designed to be as fair and as accessible as possible with no unnecessary barriers to learning or assessment.

You should take into account the needs of individual learners when planning learning experiences, selecting assessment methods or considering alternative evidence.

Guidance on assessment arrangements for disabled learners and/or those with additional support needs is available on the assessment arrangements web page: www.sqa.org.uk/assessmentarrangements.

Information for learners

Algorithms and Data Structures (SCQF level 8)

This information explains:

- what the unit is about
- what you should know or be able to do before you start
- what you need to do during the unit
- opportunities for further learning and employment

Unit information

This unit gives you an understanding of the common formats used for data storage and transfer in computer systems and how to convert from one to another. You learn how common abstract data structures work and how you can implement them to store and manipulate collections of data values by using the features of a programming language. You learn how the process of recursion works, and the common algorithms used to search and sort data stored within data structures. You also learn how time and space complexity affects their performance.

This is a specialist unit, intended for learners with an interest in computer programming. It is particularly suitable if you are studying for an HND in Computer Science or Software Development. The unit does not require you to have any previous knowledge of data structures or algorithms such as search and sort, however you should be competent in at least one programming language at SCQF level 7.

Throughout the unit, you develop skills in creating and testing programs in one or more programming languages that can read and write data between different formats, implement the various data structures, and perform algorithms on data. You also develop meta-skills covering self-management, social intelligence and innovation.

You are assessed through the design, creation and testing of one or more programs that implement data structures and algorithms covered in the course. When you finish the unit, you have important knowledge and skills in selecting and using various algorithms and data structures. This knowledge enables you to progress to more advanced programming concepts with further study in any programming language you choose.

Administrative information

Published: June 2023 (version 1.0)

Superclass: RB

History of changes

Version	Description of change	Date

Note: please check SQA's website to ensure you are using the most up-to-date version of this document.