VCE Algorithmics (HESS) 2023
Unit 3 School-based Assessment

Video 4
Background to the
Unit 3 Outcome 2 SAC

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Acknowledgement of Country

The VCAA respectfully acknowledges the Traditional Owners of Country throughout Victoria and pays respect to the ongoing living cultures of First Peoples.

VCE Algorithmics (HESS) 2023
Unit 3 School-based Assessment

Video 4
Background to the
Unit 3 Outcome 2 SAC

Phil Feain
Digital Technologies Curriculum Manager
VCAA

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Purpose of this presentation

- to build the capacity of teachers to develop compliant, rigorous and engaging VCE assessment tasks in line with the VCE assessment principles
- provide an overview of the Unit 3 Outcome 2 School-assessed Coursework (SAC) task.

Outline of the presentation

This presentation will cover:

- Planning and considerations
- The Unit 3 Outcome 2 task
- Key knowledge and key skills
- VCAA Performance descriptors
- Designing the assessment task
- Developing the marking scheme
- VASS dates for 2023

Unit 3 Outcome 2
School-assessed Coursework (SAC)

Developing a compliant, engaging and rigorous assessment task

Planning

When you are ready to write the assessment task have the following documents in front of you (These are all on the Algorithmics (HESS) study page):

- Algorithmics (HESS) Study Design – U3 AoS2 Algorithm design – pages 10–13
  - Area of Study statement, Outcome statement, Key knowledge and Key skills
- Support material: *Planning and Assessment:*
  - Unit 3 Area of Study 2 – Sample approaches to developing an assessment task
  - Unit 3 and 4 Performance descriptors: Unit 3 Outcome 2 – Performance descriptors
- On-demand videos:
  - Unit 3 Outcome 2 Background to the SAC
  - Unit 3 Outcome 2 Planning the SAC

Some considerations

The teacher must decide the most appropriate task for their cohort, time and conditions for conducting the task and inform the students ahead of the date. This decision is a result of several considerations including:

- the outcome being assessed and the task type
- the estimated time it will take to teach the key knowledge and key skills for the outcome
- the likely length of time required for students to complete the task
- the classroom environment the assessment task will be completed in
- whether the assessment task will be completed under open-book or closed-book conditions
- any additional resources required by students
- when tasks are being conducted in other subjects and the workload implications for students.

Unit 3 Outcome 2 – The outcome

On completion of this unit the student should be able to define and explain algorithmic design principles, design algorithms to solve information problems using basic algorithm design patterns, and implement the algorithms.

Unit 3 Outcome 2 – The assessment task

Contribution to final assessment

School-assessed Coursework for Unit 3 will contribute 12 per cent to the study score.

Outcomes	Marks allocated	Assessment tasks
Outcome 1 Define and explain the representation of information using abstract data types, and devise formal representations for modelling various kinds of real-world information problems using appropriate abstract data types.	50	In response to given stimulus material, create one or more designs of a data model using abstract data types to capture the salient aspects of a real-world information problem.
Outcome 2 Define and explain algorithmic design principles, design algorithms to solve information problems using basic algorithm design patterns, and implement the algorithms.	50	In response to given stimulus material: • create one or more designs of algorithms that apply algorithm design patterns or select appropriate graph algorithms to solve information problems • implement an algorithm.
Total marks	**100**	

Key knowledge

- basic structure of algorithms
- pseudocode concepts, including variables and assignment, sequence, iteration, conditionals and functions
- programming language constructs that directly correspond to pseudocode concepts
- conditional expressions using the logical operations of AND, OR, NOT
- recursion and iteration and their uses in algorithm design
- modular design of algorithms and ADTs
- characteristics and suitability of the brute-force search and greedy algorithm design patterns
- graph traversal techniques, including breadth-first search and depth-first search
- specification, correctness and limitations of the following graph algorithms:
  - Prim's algorithm for computing the minimal spanning tree of a graph
  - Dijkstra's algorithm and the Bellman-Ford algorithm for the single-source shortest path problem
  - the Floyd-Warshall algorithm for the all-pairs shortest path problem and its application to the transitive closure problem
  - the PageRank algorithm for estimating the importance of a node based on its links
- induction and contradiction as methods for demonstrating the correctness of simple iterative and recursive algorithms

Teaching towards the assessment task

Learning activities have been developed to meet the Unit 3 Outcome 2 Key knowledge bullet points.

These learning activities can be found in the Support material.

Examples of learning activities

- Develop understanding of the key concepts of sequence, branching and looping through appropriately selected worked examples.

- Create simple algorithms in class activities using conditional, iterative and defined function structures to control action execution, incorporating variables and ADTs, including algorithms that require the combination of multiple conditions using logical operations.

- Implement simple algorithms from pseudocode using either a visual or block-based programming environment, such as **Edgy**, or a general-purpose or text-based programming language, such as **Python.**

Key skills

- interpret pseudocode and execute it manually on given input
- write pseudocode
- identify and describe recursive, iterative, brute-force search and greedy design patterns within algorithms
- design recursive and iterative algorithms
- design algorithms by applying the brute-force search or greedy algorithm design pattern
- write modular algorithms using ADTs and functional abstractions
- select appropriate graph algorithms and justify the choice based on their properties and limitations
- explain the correctness of the specified graph algorithms
- use search methods on decision trees and graphs to solve planning problems
- implement algorithms, including graph algorithms, as computer programs in a very high-level programming language that directly supports a graph ADT
- demonstrate the correctness of simple iterative or recursive algorithms using structured arguments that apply the methods of induction or contradiction

VCAA Performance descriptors

VCE Algorithmics (HESS): Performance descriptors

ALGORITHMICS (HESS) UNIT 3 OUTCOME 2 SCHOOL-ASSESSED COURSEWORK					
Performance descriptors					
	DESCRIPTOR: typical performance in each range				
	Very low	**Low**	**Medium**	**High**	**Very high**
Unit 3 **Outcome 2** **On completion of this unit the student should be able to define and explain algorithmic design principles, design algorithms to solve information problems using basic algorithm design patterns, and implement the algorithms.**	Identifies limited elements of sequence, selection and repetition in a given algorithm. Manually executes a simple sequence of simple steps within pseudocode.	Interprets simple structured pseudocode with sequence, selection and simple iteration with minimal errors. Identifies some recursive, iterative, brute-force search design pattern and greedy design pattern features within pseudocode for an algorithm. Describes some elements of the concept of modularisation.	Interprets and manually executes pseudocode containing nested iteration and the use of ADTs. Describes the concepts of the recursive, iterative, brute-force search and greedy design patterns. Describes how the concept modularisation has been applied within a particular piece of pseudocode for an algorithm.	Interprets and manually executes pseudocode containing complex use of ADTs or simple recursion. Describes how the recursive, iterative, brute-force search or greedy design pattern have been applied within a particular piece of pseudocode for an algorithm.	Interprets and manually executes pseudocode containing complex use of ADTs and recursion. Completely and precisely describes how the recursive, iterative, brute-force search or greedy design pattern have been applied within a particular piece of pseudocode for an algorithm.
	Designs simple algorithms and writes these in pseudocode with significant task scaffolding required, and the final algorithm is only an effective method for a trivial subset of problem instances. Identifies some algorithm design approaches.	Designs simple algorithms and writes these in pseudocode, with some task scaffolding. The algorithm is an effective method for a non-trivial subset of problem instances. Explains the principles of the brute-force search or greedy algorithm design patterns, utilising appropriate examples.	Designs and applies algorithms including use of iteration and writes these in pseudocode with minimal errors. Designs modular algorithms. Applies a given algorithm design pattern to design an algorithm to solve a problem.	Designs algorithms using iteration and recursion for problems that have a structure that does not allow for the direct application of one of the studied algorithms. Explains the attributes required of problems for one of the algorithm design patterns to be applied.	Designs algorithms using iteration, recursion and non-trivial functions for problems that have a structure that does not allow for the direct application of one of the studied algorithms. Selects suitable algorithm design patterns for solving information problems and applies the design patterns to design algorithms and find solutions.

VCAA Performance descriptors

	Names and states correctly the computational applications of most of the specified graph algorithms. States informally the input types of the specified graph algorithms.	Explains informally how some of the specified graph algorithms perform their computation and writes the approximate pseudocode for these.	Selects and explains graph algorithms. Selects a suitable graph algorithm to apply to solve a complex problem. States precisely the input types of the specified graph algorithms.	Executes, without error, any of the specified graph algorithms using manual techniques for complex graphs.	Justifies a selection of a suitable graph algorithm for solving a complex problem based on the properties and limitations of the algorithm. Explains in precise terms why any of the specified graph algorithms are not valid for some classes of graph or graphs with certain properties.
	Identifies the first few nodes visited by either the breadth-first or depth-first search algorithm when applied to a decision tree.	Executes the breadth-first or depth-first search algorithm on a decision tree.	Applies a given graph search method to a decision tree to solve a planning problem.	Selects a suitable graph search method and applies it to a decision tree to solve a planning problem.	Evaluates the relative advantages of different graph search methods for solving a planning problem.
	Implements simple algorithms with sequential, conditional and iterative elements.	Implements simple iterative algorithms that utilise collection ADTs.	Implements graph traversal algorithms and simple recursive algorithms. Applies an implementation of a simple iterative algorithm that utilises ADTs to solve a particular problem instance.	Implements shortest-path graph algorithms. Applies an implementation of a graph traversal algorithm or simple recursive algorithm to solve particular problem instances.	Efficiently implements shortest-path graph algorithms and applies them to solve particular problem instances.
	Limited and unstructured arguments given for correctness of graph algorithms.	Describes an argument for the correctness of a graph algorithm that considers only the correctness of a specific example.	Demonstrates the correctness of specified graph algorithms using induction and contradiction for some input cases.	Describes an argument for the correctness of one of the specified graph algorithms that considers the general case of the problem, but not all steps in the chain of argument are explained.	Describes a valid argument for the correctness of at least one of the specified graph algorithms using either the induction or contradiction methods.

Designing the assessment task

- Students should be advised of the timeline and conditions under which the task is to be completed. The assessment task must directly assess the student's understanding of the key knowledge and key skills as well as their ability to apply these to the assessment task. Due dates and duration of assessment is a school-based decision.
- Students should be given instructions regarding the requirements of the task, including time allocation, format of student responses and the marking scheme/assessment criteria. The marking scheme/assessment criteria used to assess the student's level of performance should reflect the VCAA performance descriptors and key skills.

Use of Key knowledge, Key skills and Performance descriptors

- Consider how the Outcome statement, Key knowledge, Key skills and VCAA Performance descriptors connect together.
- By reading the Key knowledge, Key skills and performance descriptors alongside each other, tasks can be developed for assessment that covers the performance descriptors.

Developing the marking scheme

- List the VCAA performance descriptors and key skills.
- For each performance descriptor or key skill, list the activities required to demonstrate competency.
- Consider how many marks out of 50 that you would allocate for each descriptor.
- Determining the weightings of the descriptors or components of the task:
  - Think of the time expended by students for each part of the task, and allocate marks according to likely student effort areas.
  - Think of the difficulty of specific tasks. Ensure that there is a chance for your struggling students to demonstrate levels of competency in the task.
- Develop your marking scheme/assessment criteria.
- You need to have a range of marks allocated for the levels of performance. This helps you to spread your student marks out.

VASS SAC dates for 2023

- **Unit 3 School-based Assessment – September**
  - Algorithmics (HESS): Unit 3 Outcome 1
  - Algorithmics (HESS): Unit 3 Outcome 2
- **Unit 4 School-based Assessment – November**
  - Algorithmics (HESS): Unit 4 Outcome 3

Teachers should be aware of the dates for submission of scores into VASS in September and November. These dates are published in the 2023 Important Administrative Dates and Assessment Schedule, published annually on the VCAA website.
vcaa.vic.edu.au/pages/schooladmin/admindates/index.aspx.

Review of presentation

This presentation covered:

- Planning and considerations
- The Unit 3 Outcome 2 task
- Key knowledge and key skills
- VCAA Performance descriptors
- Designing the assessment task
- Developing the marking scheme
- VASS dates for 2023

Contact

- **Phil Feain – Digital Technologies Curriculum Manager (VCAA)**
- **Ph: (03) 9059 5146**
- **Philip.Feain@education.vic.gov.au**

Authorised and published by the
Victorian Curriculum and Assessment Authority