VCE Algorithmics (HESS) 2021 School-assessed Task

Unit 4 SAT – Criteria 1–6

Georgia Gouros

State Reviewer VCE Algorithmics (HESS)

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Copyright

The copyright in this PowerPoint presentation is owned by the Victorian Curriculum and Assessment Authority or in the case of some materials, by third parties.

No part may be reproduced by any process except in accordance with the provisions of the *Copyright Act 1968* or with permission from the Copyright Officer at the Victorian Curriculum and Assessment Authority.

In this session

- Unit 4 SAT Criteria 1–6
- Authentication
- Assessment

Unit 4 SAT Overview

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Unit 4 School-assessed Task

Three components:

- Two written explanations
- An algorithm design task in two parts
- An explanation of the universality of computation and algorithms

Unit 4 School-assessed Task

Three components, three outcomes:

- Two written explanations
- An algorithm design task in two parts
- An explanation of the universality of computation and algorithms

Assessed against six criteria:

1. understanding of formal algorithm analysis
2. skills in establishing the efficiency of a naïve algorithm
3. understanding of advanced algorithm design
4. skills in developing an improved algorithm in response to naïve algorithm
5. understanding the principles of computation
6. skills in the discussion and evaluation of computation concepts

Unit 4 SAT Outcome/Criteria

Algorithmics	Criterion 1	Criterion 2	Criterion 3	Criterion 4	Criterion 5	Criterion 6
Outcome 1 (Formal Algo Analysis)	Written Expl'n (Test)	Algorithm Design Task 1				
Outcome 2 (Advanced Algo Design)			Written Expl'n (Test)	Algorithm Design Task 2		
Outcome 3 (Universality of computation & algorithms)					Written Report	Written Report

Written explanation (Criterion 1 and 3)

Two components:

- Formal analysis techniques and the practical limits of computability
- Algorithm design patterns and techniques for addressing the limits of computation.
- Each completed under test conditions within a 45–60 minutes timeframe
- Assessed against Criterion 1 and 3. The task must allow for students to demonstrate the full range of performance.
- A mixture of question types is appropriate

Algorithm design (Criterion 2 and 4)

The design of an algorithm, consisting of two components:

- Part 1, a formal analysis of a given naïve algorithm (approximately 400 words)
- Part 2, a response to a naïve algorithm consisting of:
  - an improved algorithm design
  - an analysis of the improved design, including its correctness. (approximately 600 words)
- assessed against Criterion 2 and 4

The naïve algorithm should have scope for improvement through the application of one of the algorithm design patterns studied in Outcome 2. It should take a straightforward approach to the problem and not be unnecessarily complicated.

Explanation (Criterion 5 and 6)

Explanation of the universality of computation and algorithms:

- An explanation of the universality of computation and algorithms in one or more of the following forms:
  - a written report (approximately 700–800 words)
  - visual report (images supported by approximately 400–500 words)
  - an oral report (10–15 minutes)
- Students develop a report using core concepts from theoretical computer science studied in class. They should be able to give detailed descriptions, explanations and evaluations of these core concepts.
- Assessed against Criterion 5 and 6
- Consider giving students a choice in the mode of the report
- Authentication

Unpacking the criteria

How each of the Unit 4 Outcomes, key knowledge (KK) and key skills (KS) lead to the criteria

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Written explanation (Criterion 1 and 3)

- Ensure a good spread of difficulty in questions
- Design questions based on the performances described in the criteria

Written explanation (Criterion 1 and 3)

A written explanation of:

- Outcome 1, criterion 1, formal analysis techniques and the practical limits of computability
- Outcome 2, criterion 3, algorithm design patterns and techniques for addressing the limits of computation.
- Each task to be completed under test conditions within a 45–60 minutes timeframe
- Each task must allow for students to demonstrate the full range of performance, a mixture of question types is appropriate

Criterion 1 Test (Outcome 1)

KK

- the concept of algorithm complexity including time and space complexity

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Describes the meaning of algorithm complexity using everyday language.	Describes the meaning of algorithm complexity and describes a motivation for the study of the asymptotic time complexity of algorithms. Describes the difference between time and space complexity using everyday language.	Explains the concept of order of growth as it relates to the asymptotic complexity of algorithms. Explains the distinction between the time and space complexity of an algorithm.	Explains clearly and precisely the concept of the asymptotic complexity of algorithms. Explains clearly and precisely the distinction between the time and space complexity of an algorithm.	Compares the limits placed on the use of an algorithm by its time complexity with those placed by its space complexity.

Criterion 1 Test (Outcome 1)

KK

- the concept of P and NP-complete complexity classes

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
	Identifies some relevant distinctions between algorithm complexity and the complexity class of problems.	Explains the concept of the P complexity class.	Defines clearly and precisely the P and NP-Complete complexity classes.	Compares the P and NP-Complete computational complexity classes through the use of well-chosen example problems. Discusses the practical consequences of a problem belonging to either the P or NP-Complete complexity class.

Criterion 1 Test (Outcome 1)

KK

- the concepts of Big-O, Big-Ω and Big-Θ notation
- differences between best case and worst case complexity analysis of algorithms
- the P time complexity class and examples of algorithms that have complexities of O(1), O(log *n*), O(*n*), O(*n* log *n*), O(n^2) and O(n^3)

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Uses the written notation of Big-O notation. Orders two different Big-O expressions by their order of growth. States an example of an algorithm with a given time complexity.	Describes the meaning of a given Big-O expression. Identifies that running time may vary between inputs of equal size. Describes some elements of the algorithmic structures that give rise to O(n^2) or O(n^3) time complexities. Makes simple comparisons between the efficiency of algorithms based on their algorithmic complexity.	Determines the order of growth associated with a range of mathematical expressions. Determines whether the running time of a specific algorithm varies for different inputs of equal size. Makes practical comparisons between the efficiency of algorithms based on their algorithmic complexity, distinguishing between differences in algorithmic complexity that lead to significant changes in the size of tractable inputs and those that do not.	Explains the general mathematical meaning of statements made using Big-O, Big-Ω and Big-Θ notation. Discusses some differences between best case and worst case analysis of algorithms. Explains some elements of the algorithmic structures that give rise to O(n^2), O(n^3),O(log(n)) and O(n log(n)) time complexities.	Explains precisely the mathematical meaning of statements made using Big-O, Big-Ω and Big-Θ notation. Discusses the differences between and appropriateness of best and worst case complexity analysis for evaluating algorithms in practice. Explains the algorithmic structures that give rise to O(n^2), O(n^3),O(log(n)) and O(n log(n)) time complexities.

Criterion 1 Test (Outcome 1)

KK

- an understanding that problems can be harder than P problems and the implications for their solvability
- consequences of combinatorial explosions and indicators for them

KS

- demonstrate how exponentially sized search and solution spaces impose practical limits on computability.

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Informally describes the meaning of intractability and states an example of an intractable problem. Describes the concept of exponential growth using everyday language.	Describes the meaning of intractability and describe its practical relevance. Describes the concept of exponential growth in a search space.	Explains why some computable problems are intractable, with the use of examples of both tractable and intractable problems. Explains one indicator of exponentially sized search spaces based on examples introduced in the study. Demonstrates some features of how combinatorial algorithms create exponentially sized search spaces.	Demonstrates clearly how combinatorial algorithms create exponentially sized search spaces.	

Criterion 1 Test (Outcome 1)

KK

- recurrence relations as a method of describing the time complexity of recursive algorithms
- the Master Theorem for solving recurrence relations of the form: $T(n) = aT\left(\frac{n}{b}\right) + kn^c$

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Uses the written notation of recurrence relations. States the solutions to some input cases of the Master Theorem.	Evaluates a recurrence relation for a given input. Describes the Master Theorem, including the problem that it solves and the solution it provides, with minor errors or omissions.	Explains the Master Theorem, including the problem that it solves and the solution it provides.		Explains the connections between the call trees produced by recursive functions and the solutions to the three cases of the Master Theorem.

Criterion 3 Test (Outcome 2)

KK

- divide and conquer algorithms that have linear time split and merge steps, including mergesort and quicksort
- dynamic programming algorithms that require no more than a single dimension array for storage, including Fibonacci numbers, 1-D knapsack problem, change making problem
- tree search by backtracking and its applications
- differences between divide and conquer and dynamic programming and their applications
- induction and contradiction as methods for demonstrating the correctness of dynamic programming and backtracking algorithms
- minimax method to solve combinatorial problems

KS

- recognise and apply the divide and conquer, dynamic programming, and backtracking design patterns
- apply the minimax method
- propose an argument for the correctness of an algorithm

Criterion 3 Test (Outcome 2)

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
States an example of a backtracking, divide and conquer or dynamic programming algorithm. Identifies the divide and merge steps within the pseudocode of a given divide and conquer algorithm. States the problems solved by some of the mergesort, quicksort, knapsack, change making and minimax algorithms. States the algorithmic design paradigm associated with the mergesort, quicksort, knapsack or change making algorithms.	Identifies the attributes of a given algorithm that indicate the use of a specific algorithmic design pattern. Determines whether a given algorithm is an example of specific algorithmic design paradigm. Outlines the mergesort, quicksort, knapsack, change making or minimax algorithm, possibly with some minor errors. Completes the missing lines of mergesort, quicksort, knapsack, change making and minimax algorithm. Proposes an argument for the correctness of one of the specified divide and conquer or dynamic programming algorithms that does not consider the general case of the problem, rather only the correctness of a specific problem instance.	Describes the general structure of the backtracking, divide and conquer, or dynamic programming algorithmic design patterns. Describes in pseudocode any of the mergesort, quicksort, knapsack, change making or minimax algorithms. Proposes an argument for the correctness of one of the specified divide and conquer or dynamic programming algorithms that considers the general case of the problem but only includes a minority of the required steps in the chain of argument.	Explains, using appropriate metalanguage, some of the principles of the backtracking, divide and conquer, or dynamic programming algorithmic design patterns. Analyses the benefits and costs associated with any of the mergesort, quicksort, knapsack or change making algorithms in comparison with naïve algorithms for the same problem. Proposes an argument for the correctness of one of the specified divide and conquer or dynamic programming algorithms that considers the general case of the problem but not all steps in the chain of argument are included.	Explains precisely, using appropriate metalanguage, the principles of the backtracking, divide and conquer, or dynamic programming algorithmic design patterns. Proposes a valid argument for the correctness of one of the specified divide and conquer or dynamic programming algorithms using either the induction or contradiction method.

Criterion 3 Test (Outcome 2)

KK

- heuristics and randomised meta-heuristic algorithms, including simulated annealing, as approaches to overcome soft limits of computation
- limitations of heuristics and randomised meta-heuristic algorithms
- the graph colouring, the 1-D knapsack and travelling salesman problems and heuristic methods for solving them.

KS

- use heuristics to solve computationally hard problems
- explain the meaning of randomised approaches for intractable problems, including the graph colouring, the 1-D knapsack and travelling salesman problems.

Criterion 3 Test (Outcome 2)

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Describes using everyday language the use of heuristics or randomisation as general approaches to intractable problems. Describes using everyday language the graph colouring, knapsack or travelling salesman problem. Identifies a decision within the graph colouring, knapsack or travelling salesman problem that could be randomised.	Describes the principles of heuristic or randomisation algorithms as general approaches to intractable problems, with the inclusion of some technical detail and use of appropriate metalanguage. Determines whether a problem given in context is an instance of the graph colouring, knapsack or travelling salesman problem. Describes how a decision within the graph colouring, knapsack or travelling salesman problem could be randomised.	Outlines a specific algorithm based on either a heuristic or randomisation design approach. Defines clearly and precisely the graph colouring, knapsack or travelling salesman problem using appropriate metalanguage. Partially describes an algorithm for the graph colouring, knapsack or travelling salesman problem that would be appropriate for large input instances.	Describes in detail a specific algorithm based on either a heuristic or randomisation design approach. Describes some of the limitations of heuristic and randomised algorithms. Explains the decision and optimisation versions of the graph colouring, knapsack or travelling salesman problems. Describes clearly a specific algorithm for the graph colouring, knapsack or travelling salesman problem that would be appropriate for large input instances.	Discusses the merits and limitations of the use of heuristic and randomised algorithms in practice. Compares the decision and optimisation versions of the graph colouring, knapsack or travelling salesman problems with appropriate references to the computational complexity classes of the problems.

Design of an algorithm

The design of an algorithm, consisting of two components:

- Part 1, Outcome 1, criteria 2, formal analysis of a given naïve algorithm (approximately 400 words)
- Part 2, Outcome 2, criteria 4, a response to a naïve algorithm consisting of:
  - an improved algorithm design
  - an analysis of the improved design, including its correctness.

(approximately 600 words)

Assessed through two criteria

- skills in establishing the efficiency of a naïve algorithm
- skills in developing an improved algorithm in response to a naïve algorithm

Criterion 2 Algorithm Design (Outcome 1)

KS

- formally analyse the efficiency of algorithms using Big-O notation
- read off a recurrence relation for the running time of a recursive algorithm that can be solved by the Master Theorem or takes the form: $T(n) = \sum_{i=1}^{k} T(n - a_i) + b$ where $a_i \in N$
- use the stated Master Theorem to solve recurrence relations

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Identifies lines of the naïve algorithm's pseudocode that execute in constant running time and lines whose running time vary depending on the algorithm's input.	Applies step-counting methods with some accuracy to iterative or conditional components of the naïve algorithm whose running time varies depending on the input to analyse their worst case time complexity.	Analyses the worst case time complexity of the naïve algorithm through the selection and application of appropriate techniques. Some errors or omissions in the application of the techniques lead to an overall inaccurate analysis.	Analyses clearly and thoroughly the time complexity of the naïve algorithm through the selection and application of appropriate techniques.	Analyses precisely and elegantly the time complexity of the naïve algorithm through the efficient selection and application of appropriate techniques.
Applies step-counting concepts with limited accuracy to reason about the running time of parts of the naïve algorithm.	Applies recurrence relation methods with some accuracy to a recursive component of the naïve algorithm to analyse its worst case time complexity.	Accurately applies step-counting methods to a nested iterative component of the naïve algorithm to analyse its worst case time complexity.		
Applies recurrence relation methods with very limited accuracy to a recursive component of the naïve algorithm to analyse its worst case time complexity.	Identifies reasonable estimations for the performance of ADT operations used within the naïve algorithm.	Accurately applies recurrence relation methods to a recursive component of the naïve algorithm to analyse its worst case time complexity.		

Criterion 2 Algorithm Design (Outcome 1)

KS

- identify input patterns that lead to best and worst case performance

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
		Identifies a single input instance for the naïve algorithm that would result in the best-case or worst-case running time, respectively.	Describes the class of input instances for the naïve algorithm that would result in the best case or worst case running times, respectively.	Describes efficiently and precisely the classes of input instances for the naïve algorithm that would result in the best case and the worst case running times.

Criterion 4 Algorithm Design (Outcome 2)

KS

- develop and compare different algorithms for solving the same problem, using different algorithm design patterns

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Selects an algorithm design pattern and applies limited elements of its structure to design an algorithm that is either incomplete or incorrect.	Selects an algorithm design pattern and applies significant elements of its structure to develop an algorithm that is not correct. Proposes an argument for the correctness of the algorithm that does not consider the general case, rather only the correctness of a specific problem instance.	Selects an appropriate algorithm design pattern and applies it to develop an algorithm that is correct for most inputs and is an improvement in algorithmic complexity on a naïve algorithm. Proposes an argument for the correctness of the algorithm that considers the general case but only includes a minority of the required steps in the chain of argument.	Selects an appropriate algorithm design pattern and applies it to develop a clearly expressed and correct algorithm that is an improvement in algorithmic complexity on the naïve algorithm. Proposes an argument for the correctness of the algorithm that considers the general case of the problem but not all steps in the chain of argument are explained.	Selects an appropriate algorithm design pattern and applies it to develop an elegantly and precisely expressed, correct and efficient algorithm. Proposes a valid argument for the correctness of the algorithm, if appropriate using either the induction or contradiction method.

Criterion 4 Algorithm Design (Outcome 2)

KS

- develop and compare different algorithms for solving the same problem, using different algorithm design patterns

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Compares very limited aspects of the student-designed algorithm and the naïve algorithm for the problem.	**Compares simply the student-designed algorithm and the naïve algorithm for the problem.**	**Compares in detail the student-designed algorithm and the naïve algorithm for the problem, highlighting the improvements made.**	**Compares comprehensively and accurately the student-designed algorithm and the naïve algorithm for the problem, highlighting the improvements made and any trade-offs required.**	

Criterion 4 Algorithm Design (Outcome 2)

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Identifies lines of the improved algorithm's pseudocode that execute in constant running time and lines whose running time vary depending on the algorithm's input. Applies step-counting concepts with limited accuracy to reason about the running time of parts of the improved algorithm. Applies recurrence relation methods with very limited accuracy to a recursive component of the improved algorithm to analyse its worst case time complexity.	Applies step-counting methods with some accuracy to iterative or conditional components of the improved algorithm whose running time varies depending on the input to analyse their worst case time complexity. Applies recurrence relation methods with some accuracy to a recursive component of the improved algorithm to analyse its worst case time complexity. Identifies reasonable estimations for the performance of ADT operations used within the improved algorithm.	Analyses the worst case time complexity of the improved algorithm through the selection and application of generally appropriate techniques. Some errors or omissions in the application of the techniques lead to an overall inaccurate analysis. Accurately applies recurrence relation methods to a recursive component of the improved algorithm to analyse its worst case time complexity. Accurately applies step-counting methods to a nested iterative component of the improved algorithm to analyse its worst case time complexity.	Clearly and thoroughly analyses the time complexity of the improved algorithm through the selection and application of appropriate techniques. If relevant, describes the class of input instances for the improved algorithm that would result in the best case or worst case running times.	Analyses precisely and elegantly the time complexity of the improved algorithm through the efficient selection and application of appropriate techniques. If relevant, describes efficiently and precisely the class of input instances for the improved algorithm that would result in the best case or the worst case running times.

Explanation

Explanation of the universality of computation

An explanation of the universality of computation and algorithms in one or more of the following forms:

- a written report (approximately 700–800 words)
- visual report (images supported by approximately 400–500 words)
- an oral report (10–15 minutes)

Assessed through two criteria

- understanding of principles of computation
- skills in the discussion and evaluation of computation concepts

These criteria provide a broad range of performance descriptions so that a judgement of the students level of performance can be made regardless of the topics explored by a student in their report. Students are not expected to demonstrate all of the criteria statements.

Criterion 5 Explanation (Outcome 3)

KK

- David Hilbert’s 1927 program to fully formalise mathematical reasoning, including its goals, historical context, outcome and connection to the origin of computer science
- characteristics of a Turing machine
- implications of undecidability for Hilbert’s Program

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Outlines the goals of Hilbert’s program, with some minor errors.	Describes briefly aspects of the goals of Hilbert’s program, its outcome and its historical context, possibly with minor errors.	Describes, with appropriate use of metalanguage, the goals of Hilbert’s program, its outcome and its historical context and connection to the origin of computer science.	Describes in detail the goals of Hilbert’s program, its outcome, its historical context and connection to the origin of computer science.	Explains comprehensively and precisely the goals of Hilbert's program, its outcome, its historical context and connection to the origin of computer science.
Outlines some characteristics of a Turing machine, with some errors.	Describes the components of a Turing machine.	Describes, with appropriate use of metalanguage, some aspects of the operation of a Turing machine.	Describes in detail the operation of a Turing machine.	Explains comprehensively and precisely the operation of a Turing machine.

Criterion 5 Explanation (Outcome 3)

KK

- the concept of decidability and undecidability, including the Halting Problem and the demonstration that it is undecidable, and implications for automatic program verification
- the meaning of the Church-Turing Thesis and its limitations

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Outlines the Halting problem, with some minor errors. Uses the term “decision problem” appropriately.	Describes briefly some characteristics of decidable or undecidable problems.	Describes, with appropriate use of metalanguage, the concept of undecidability and a specific example of an undecidable problem, possibly with minor errors.	Describes in detail the concept of undecidability. Describes coherently and in detail the characteristics a specific example of an undecidable problem.	Explains comprehensively and precisely the concept of undecidability. Describes comprehensively and precisely the characteristics of a specific undecidable problem.
Outlines some details of concepts relevant to how the equivalence of computational formalisms is shown, with some errors.	Describes briefly some concepts relevant to how the equivalence of computational formalisms is shown.	Describes accurately and clearly concepts relevant to how the equivalence of computational formalisms is shown.	Explains in detail conceptually how the equivalence of computational formalisms is shown.	Explains comprehensively and precisely conceptually how the equivalence of computational formalisms is shown.

Criterion 5 Explanation (Outcome 3)

KK

- John Searle’s Chinese Room Argument, including …
- characteristics of alternative methods of computation (see 2021 VCAA).

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Outlines the thought experiment that underpins the Chinese Room argument.	Describes the thought experiment that underpins Chinese Room argument and briefly describes aspects of its core position, standard replies, or historical context.		Describes in detail and with appropriate use of metalanguage the Chinese Room argument, its historical context, motivation, and standard responses.	Describes comprehensively and precisely several standard responses to the Chinese Room argument. The standard responses are clearly distinguished, with no confusion or conflation.
Outlines some features of an alternative method of computation.	Describes using everyday language the core features of an alternative model of computation.	Describes, with appropriate use of metalanguage, an alternative model of computation, including some aspects of its operation.	Describes in detail an alternative model of computation, including some technical aspects of the method and its operation.	Explains comprehensively and precisely an alternative model of computation, including significant technical details of the method and its operation.

Criterion 6 Explanation (Outcome 3)

KS

- demonstrate the existence of hard limits of computability
- discuss the merit and limitations of the Church-Turing and Cobham theses
- define decidability and undecidability and explain selected examples

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Outlines the central premise of the Church-Turing thesis.	Discusses in brief the Church-Turing thesis, its historical context, and limited aspects of its merits or limitations.	Discusses clearly the Church-Turing thesis, its historical context, and some aspects of its merits or limitations.	Evaluates the merits or limitations of the Church-Turing thesis and discusses its implications.	Comprehensively and precisely evaluates the merits and limitations of Church-Turing thesis and discusses its implications.
Outlines some aspect of the consequences of undecidability	Discusses in brief the implications of undecidability for Hilbert's Program.	Discusses clearly several implications of undecidability.	Demonstrates the undecidability of the Halting problem through the use of an effective and detailed argument.	Demonstrates precisely the undecidability of the Halting problem through the use of a comprehensive and well-structured argument.
	Demonstrates the undecidability of the Halting problem through the use of a poorly explained and limited argument.	Demonstrates the undecidability of the Halting problem through the use of a relevant and clear argument.		

Criterion 6 Explanation (Outcome 3)

KK

- the relationship between Turing machines and computational complexity theory
- the meaning of Cobham's thesis for computational complexity theory

KS

- explain conceptually how the equivalence of computational formalisms is shown

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Outlines the relationship between Turing machines and computational complexity theory.	Discusses in brief the relationship between Turing machines and computational complexity theory.	Discusses clearly the relationship between Turing machines and computational complexity theory.	Discusses in detail and coherently the relationship between Turing machines and computational complexity theory.	Discusses comprehensively and precisely the relationship between Turing machines and computational complexity theory.
Outlines the central premise of Cobham's thesis.	Discusses in brief Cobham's thesis and limited aspects of its merits or limitations.	Describes clearly Cobham's thesis and some of its merits and limitations.	Evaluates the merits or limitations of the Cobham's thesis and discusses its meaning for computational complexity theory.	Comprehensively and precisely evaluates the merits and limitations of Cobham's thesis and discusses its meaning for computational complexity theory.

Criterion 6 Explanation (Outcome 3)

KS

- present and evaluate responses to the Chinese Room Argument
- explain how alternative methods of computation might be used to overcome current limits of computation.

1-2 (very low)	3-4 (low)	5-6 (medium)	7-8 (high)	9-10 (very high)
Discusses in brief the strength of one of the standard responses to the Chinese Room argument. Outlines some aspects of the connections between the Chinese Room argument and artificial intelligence.	Evaluates the strength of at least two of the standard responses to the Chinese Room argument. Discusses in brief the connections between the Chinese Room argument and artificial intelligence.	Compares the strength of two of the standard responses to the Chinese Room argument. Discusses clearly the connections between the Chinese Room argument and artificial intelligence.	Formulates a position either for or against the Chinese Room argument through an evaluation of standard responses and connects this position to its implications for artificial intelligence.	Formulates a substantiated position either for or against the Chinese Room argument through a sound evaluation of several standard responses and connects this position to its implications for artificial intelligence.
Outlines the merits of an alternative method of computation.	Discusses in brief how an alternative method of computation might be used to overcome current limits of computation.	Discusses clearly how an alternative method of computation might be used to overcome current limits of computation and some aspects of its merits and limitations.	Explains in detail how an alternative model of computation might be used to overcome current limits of computation and evaluates briefly its merits and limitations.	Explains comprehensively and precisely how an alternative model of computation might be used to overcome current limits of computation and evaluates its merits and limitations.

Authentication

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Authentication

Regularly observe student work during the life of the SAT

- some teachers use one class a week to do this formally
- use the VCAA Authentication Record Form and update with comments from observations as you go – **these can be requested** as part of the audit process
- both you and the student are to sign and date your initials for each observation at the time of the observation and for submission

Authentication Record Form VCE Algorithmics (HESS) 2021

Unit 4 School-assessed Task

This form must be completed by the class teacher. It provides a record of the monitoring of the student's work in progress for authentication purposes. This form is to be retained by the school and filed. It may be collected by the VCAA as part of the School-based Assessment Audit.

Student name: ..... Student No

School: .....

Teacher: .....

Component of School-assessed Task	Date observed/ submitted	Authentication comments	Teacher's initials	Student's initials
Observation 1: Submission of written explanation Student has submitted the written explanation of formal analysis techniques and the practical limits of computability.				
Observation 2: Analysis of algorithm Student has analysed a naïve algorithm.				
Observation 3: Progressive development of data model Student has continued to develop a data model and documentation.				
Observation 4: Submission of written explanation Student has submitted the written explanation of algorithm design patterns and techniques for addressing the limits of computation.				
Observation 5: Development of algorithm Student has submitted an improved algorithm and analysis.				
Observation 6: Report Student has commenced the process of developing their report.				
Observation 7: Report Student has submitted an explanation of the universality of computation and algorithms.				

I declare that all resource materials and assistance used have been acknowledged and that all unacknowledged work is my own.

Student signature ..... Date .....

© VCAA

Assessment

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government

Assessment

2021

Victorian Certificate of Education
Algorithmics (HESS) Assessment Sheet
School-assessed Task: Unit 4

STUDENT NAME

This assessment sheet will assist teachers to determine their score for each student. Teachers need to make judgments on the student's performance for each criterion. Teachers will be required to choose one number from 0–10 to indicate how the student performed on each criterion with comments, as appropriate. Teachers then add the subtotals to determine the total score.

STUDENT NUMBER

ASSESSING SCHOOL NUMBER

Criteria for the award of grades **The extent to which the student demonstrates:**	Not Shown (0)	Very Low (1–2)	Low (3–4)	Med (5–6)	High (7–8)	Very High (9–10)
1 understanding of formal algorithm analysis						
2 skills in establishing the efficiency of a naïve algorithm						
3 understanding of advanced algorithm design						
4 skills in developing an improved algorithm in response to a naïve algorithm						
5 understanding of the principles of computation						
6 skills in the discussion and evaluation of computation concepts.						
SUBTOTALS						

Performance on Criteria: Teacher's Comments
You may wish to comment on aspects of the student's work that led to your assessment.

If a student does not submit the School-assessed Task at all, N/A should be entered in the total score box.

TOTAL SCORE

Contact

Phil Feain – Digital Technologies Curriculum Manager (VCAA)

Ph: (03) 9059 5146

Philip.Feain@education.vic.gov.au

VICTORIAN CURRICULUM AND ASSESSMENT AUTHORITY

VICTORIA State Government