2022 VCE Algorithmics (HESS) external assessment report

General comments

Students generally performed well on Section A of the examination, with more than half of the questions in this section correctly answered by over 80 per cent of students. Moreover, in Section B, students provided clear and explicit responses to questions that involved the modelling of salient features of problems with abstract data types, such as Question 3a. They also answered questions that required them to show their understanding of an algorithmic process, such as Question 4b., well.

Too often, students provided answers that did not directly respond to the questions, instead providing general definitions or explanations of terms used within the questions. In Section B this was particularly evident in some responses to Questions 1, 3c., 13a. and 13b.

Some students would benefit from further review of the structure of proof by induction and proof by contradiction arguments, which were assessed in Questions 4a. and 6b. respectively. The first step was often done well; the students could state the base case or the initial assumption. However, they often struggled with communicating the appropriate logic needed to build the argument from the initial step to the conclusion. Exposure to a wider variety of practice problems may help with this.

Student responses to questions requiring pseudocode were for the most part clearly structured and followed conventions such as indentation well; that is, it was usually straightforward to understand what students were trying to do. Questions 2c. and 4bii. required students to write pseudocode that incorporated predefined functions. Many students handled those predefined functions incorrectly by either misinterpreting or ignoring them.

Some students would benefit from further review of the structure of recursive algorithms, with many answers for Questions 4bii. involving errors in relation to terminating conditions. Some students did not have enough familiarity with tree traversal and generating search trees, which were assessed in Questions 2c. and 7a., respectively. A detailed working knowledge of an algorithm such as depth-first search may have assisted some students in relation to Question 7a.

Further points for consideration include:

- An understanding of how the modelling of data can have an impact on the efficiency of algorithmic solutions, which was relevant to Questions 3bi. and 9c.
- A more formal appreciation of conventions with respect to decision trees, writing signature specifications and working with Turing machines as assessed in Questions 2, 3bii. and 10 respectively.

Specific information

This report provides sample answers or an indication of what answers may have included. Unless otherwise stated, these are not intended to be exemplary or complete responses.

The statistics in this report may be subject to rounding resulting in a total more or less than 100 per cent.

Section A

The correct answer is indicated by shading.

Question	Correct Answer	%A	%B	%C	%D	Comment
1	B	7	**92**	1	0	
2	A	**93**	2	5	1	
3	A	**82**	4	6	7	
4	C	5	12	**63**	18	Creating a truth table to evaluate the expressions and careful reading of the prompt, particularly in regard to the defining of the Boolean variable Z as 'the year is divisible by 400', may have assisted students in this question.
5	D	5	6	5	**84**	
6	C	2	3	**95**	0	
7	B	18	**73**	2	8	When executing the pseudocode manually for the given input, students should be careful to correctly observe the looping condition. The while loop is exited when `a` is 24. The function returns `a - b`, where `b` is 12, hence the function returns 12.
8	D	2	1	2	**95**	
9	C	4	3	**93**	0	
10	B	4	**75**	11	11	The minimax algorithm assumes both players play optimally. Player B is the minimising player and hence the algorithm would assign the values of 5 and 2 to the nodes corresponding to the Player B row. Player A is the maximising player and hence the algorithm would assign the value of 5 to the root node.
11	C	9	1	**83**	7	
12	C	2	7	**77**	14	The recursive calls decrease `n` by a constant of 1 and 2, respectively. Other steps in the function execute in constant time.
13	A	**89**	2	4	5	
14	D	12	5	9	**74**	From the prompt it is known that the running time has a loose upper bound of $O(n^3)$. The tight upper bound is unknown, therefore $O(n^2)$ could be a tighter upper bound. $\Omega(n^2)$ is acceptable because as a loose lower bound it must be lower or equal to $\Omega(n^3)$. $O(n^4)$ is acceptable because it is a looser upper bound. $\Omega(n^4)$ could not be true because a loose lower bound cannot exceed an upper bound.
15	B	6	**89**	2	3	
16	B	36	**47**	1	16	The prompt requires students to identify what the Halting Problem was used to demonstrate, which is that there exist some problems that cannot be solved by an algorithm. The prompt did not require students to identify what the Halting Problem is. Moreover, the statement that we can never know whether a specific computer program will halt or not for given input, is not a correct description of

Question	Correct Answer	%A	%B	%C	%D	Comment
						the Halting Problem.
17	A	**63**	22	6	8	A Turing machine can solve all computable problems. If the computer can emulate a Turing machine it too can solve all computable problems. Showing that a Turing machine can emulate the computer does not demonstrate that the computer can solve all computable problems.
18	A	**85**	3	5	7	
19	D	15	8	9	**68**	The statement that all problems in NP are NP-complete can be shown to be incorrect by way of a counterexample. Consider a problem in P. Such a problem is also in NP, but is not NP-complete.
20	C	29	22	**34**	15	Unless it is stated explicitly or it is implicit in a given problem context, students should assume that graphs are simple, that is they are unweighted, undirected, contain no self-loops or instances of multiple edges between pairs of nodes. This question requires students to consider simple graphs only. The statement that 'a complete graph can be acyclic' is true because complete graphs containing one or two nodes are acyclic. The statement 'if $n \geq 3$, a graph with n vertices and at least n edges must be cyclic', and the statement 'if an edge is added to a tree, the resulting graph will be cyclic' must be true because a tree has $n-1$ edges, and introducing one extra edge will create a cycle. The statement 'if $n \geq 3$, a graph with n vertices must have at least n edges to be cyclic' is incorrect. This can be shown by way of a counterexample. For instance, consider a graph with four nodes and three edges for which one node is disconnected, and the other three nodes are connected to each other by edges forming a cycle.

Section B

Question 1

Marks	0	1	2	Average
%	44	20	36	0.9

A possible answer was:

Initialise an empty result list. Compare the first item in each sublist from the conquer step and take the lowest of the two and append it to the end of the result list. Repeat this process until one of the sublists is empty. Lastly, add any remaining items in the non-empty sublist to the end of the result list.

Some responses discussed the mergesort algorithm in general terms rather than providing a detailed description of the merge step.

Question 2a.

Marks	0	1	2	Average
%	45	37	18	0.8

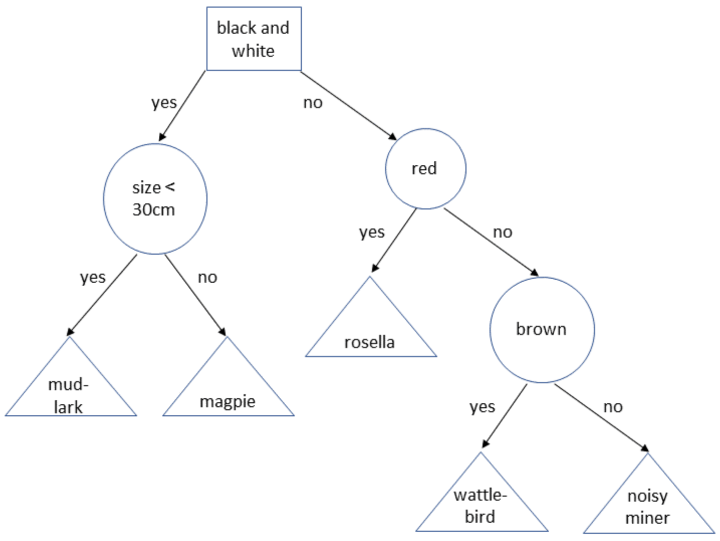
A decision tree is a rooted tree, a type of graph, in which each non-leaf node represents a decision to be made and the outgoing branches represent the possible decision pathways. The leaf nodes represent the outcomes of the decision process.

Students who scored highly explained the concept of a decision tree by relating all elements of the structure of a decision tree to their respective functions.

Question 2b.

Marks	0	1	2	3	4	Average
%	4	5	42	13	36	2.7

A possible answer was:



Most students drew some form of rooted tree. Students who scored highly did not include redundant questions and drew decision trees in which:

- non-leaf nodes represented decisions to be made
- edges represented answers
- leaf nodes represented outcomes.

Question 2c.

Marks	0	1	2	3	4	Average
%	44	20	14	12	10	1.3

A correct answer would be:

```
Algorithm birdIdentifier(T, x)
    Initialise u as the root node of the tree T
    while u is not a leaf node do
        if P(u, x) then
            set u to the child node of u corresponding to True
        else
            set u to the child node of u corresponding to False
    return the bird value of u
```

Many responses did not correctly respond to the question. The question stated 'Let *T* be a decision tree for identifying birds …', many students provided pseudocode that specifically related to trees that they drew in Question 2b. rather than *T*. Few students correctly incorporated the function $P(u, x)$ into their answers.

Question 3a.

Marks	0	1	2	Average
%	6	21	74	1.7

A directed, weighted graph where junctions are nodes and streets are edges, whose weights are the speeds.

This question was generally answered well. High-scoring responses clearly mapped the salient features of the problem to nodes, edges and edge-weights.

Question 3bi.

Marks	0	1	2	Average
%	4	50	46	1.4

Kashvi's method for storing the data allows for very fast identification of how many drivers have trips on routes of interest, while Hamed's method for storing the data would result in a comparatively slower process, and hence Kashvi's method should be used.

High-scoring responses provided a justification for the suitability of Kashvi's method for storing data that focused on the implications for the efficiency of its usage. Responses that did not score well provided justifications that were based only on features of the data structure.

Question 3bii.

Marks	0	1	2	Average
%	16	43	42	1.3

For the add operation an example of a correct answer is:

- add: element × element × element × dictionary → dictionary

For the totalTrips operation an example of a correct answer is:

- totalTrips: dictionary → integer

Students should avoid using variable names such as origin and destination in signature specifications.

Question 3c.

Marks	0	1	2	Average
%	14	50	36	1.2

This question required students to state what algorithm would be suitable and justify why it would be suitable. The question did not require students to describe how the algorithm would function to solve the problem. High-scoring responses identified that Dijkstra's algorithm would be a good choice because the graph only has positive edge weights and/or because the algorithm is efficient.

Question 4a.

Marks	0	1	2	3	Average
%	41	25	16	17	1.1

A correct answer would be:

The base case $P(1)$ is true because if the bag has one ball then the picking process is complete and the bag has one ball.

Let k be some natural number and assume that $P(k)$ is true. We now have to prove that the truth of $P(k)$ necessarily implies the truth of $P(k + 1)$.

Reverse the ball-picking process by taking out the ball that was added to the bag and returning the two that were drawn. The bag had k balls after this process, and hence had $k – 1 + 2 = k + 1$ balls beforehand. Therefore, if $P(k)$ is true then $P(k + 1)$ is true.

By the principle of induction since $P(1)$ is true, and if $P(k)$ is true then $P(k + 1)$ is true, therefore $P(n)$ is true for every natural number k.

Some students misidentified the base case as $P(2)$. Another error that was seen was students conflating $P(k)$ with k, such as through statements like $P(k) = k$ or $k = k + 1$. Moreover, a very common error in the inductive step was to argue that $P(k + 1)$ implies $P(k)$, whereas an inductive proof appropriate to this problem context requires that $P(k)$ implies $P(k + 1)$ in order for the argument to build from the base to the necessary truth of $P(n)$.

Question 4bi.

Marks	0	1	Average
%	15	85	0.9

The correct answer was:

- *LB*(1, 0) evaluates to ‘Red’
- *LB*(0, 1) evaluates to ‘Green’

Question 4bii.

Marks	0	1	Average
%	15	85	0.9

When both green balls are picked, the bag now has r red balls and g - 2 green balls. Since a new red ball is then put into the bag, the bag has now r + 1 red balls and g - 2 green balls. Hence, *LB*(r, g) = *LB*(r + 1, g - 2) after the first pick.

This question was generally answered well.

Question 4biii.

Marks	0	1	2	3	4	5	6	Average
%	31	8	10	8	12	10	22	2.8

Sample answer:

```
Algorithm LB(r, g)
    if r = 1 and g = 0 then
        return 'Red'
    if r = 0 and g = 1 then
        return 'Green'
    x ← redInPick(r, g)
    if x = 0 then
        return LB(r + 1, g - 2)
    else
        return LB(r - 1, g)
```

This question required students to write pseudocode for a recursive algorithm for *LB*(r, g). Some students incorrectly provided responses involving iterative algorithms. Errors often related to incorrect terminating conditions and/or recursive calls, including ordering them incorrectly.

The question provided the function *redInPick*(r, g) that simulates a random pick of two balls from the bag and returns the number of red balls obtained. Many students did not incorporate the *redInPick*(r, g) function correctly into their responses. Some students mistakenly attempted to write pseudocode for the *redInPick* function.

Students should have a clear understanding of the structure of recursive algorithms. They should carefully read the questions and integrate functions that have been provided into their responses.

Question 5a.

Marks	0	1	2	Average
%	18	53	29	1.1

DNA computing involves using pieces of DNA to perform parallel exhaustive searches for search problems such as the Hamiltonian circuit problem. The parallelism comes from chemical reactions happening in parallel, rather than sequential searching.

Question 5b.

Marks	0	1	2	Average
%	21	54	25	1.1

A sample answer is:

Two limitations of DNA computing that have slowed its application to real-world problems are:

- The effort and time required to analyse solutions once the ‘computation’ is completed.
- The exponential growth of the physical quantity of DNA required for intractable problems with large input sizes.

The following reasons were not considered as they are not the primary limiting factors:

- the manual labour required
- the technical/academic exercise of mapping problems to DNA
- not being able to solve all problems.

Question 6a.

Marks	0	1	Average
%	59	41	0.4

A sample answer is:

Prim’s algorithm selects the edge with the lowest cost that connects a connected node to an unconnected node.

Almost all students identified the requirement to select a lowest cost edge, but many did not clearly describe how the new edge selected at each step needs to connect a node in the subtree to one that was not.

Question 6b.

Marks	0	1	2	3	4	Average
%	36	15	14	21	13	1.7

The following is an example of a possible response:

Assume that *G* contains all distinct edge weights.

- Let G_1 and G_2 be two different subgraphs of G returned by Prim's algorithm as minimum spanning trees.
- Select a leaf node in G_1 and traverse G_1 until an edge (u, v) is reached that is not in G_2.
- Let $G^{¿}$ be the connected component in G_1 and G_2 that is connected to u.
- Find an edge (m, n) in G_2 that connects a node in $G^{¿}$ to a node not in $G^{¿}$.
- Since $weight(u, v) \neq weight(m, n)$, swap the edge with the higher weight with that with the lower weight.
- This creates a new graph with a lower cost. Therefore, either G_1 or G_2 is not a minimum spanning tree.
- This contradicts this initial statement that both G_1 or G_2 are minimum spanning trees, and hence the G cannot have distinct edge weights.

Students were asked to outline an argument by contradiction. Some students provided general explanations that did not follow the structure of an argument by contradiction. High-scoring responses constructed a cut of the graph/swap of an edge, or provided an explanation for why if the edge weights are all distinct then Prim's algorithm makes the same decision at each step and so must always produce the same graph.

Question 7a.

Marks	0	1	2	3	4	Average
%	53	17	20	8	2	1.0

The following is an example of a possible solution:

```
Function create_tree(current_node, T, sizes, values)
    i ← current_node.index
    while i < length of sizes do
        new_s ← current_node.s with sizes[i] appended
        new_v ← current_node.v with values[i] appended
        add a child_node of current_node to T
        Set index parameter of child_node to i + 1
        Set parameters s and v of the child_node to new_s and new_v
        create_tree(child_node, T, sizes, values)


Algorithm knapsack(c, sizes, values)
    Initialise root_node of a new tree T
    Set index parameter of root_node to 0
    Set parameters s and v of the root_node to empty lists
    create_tree(root_node, T, sizes, values)
```

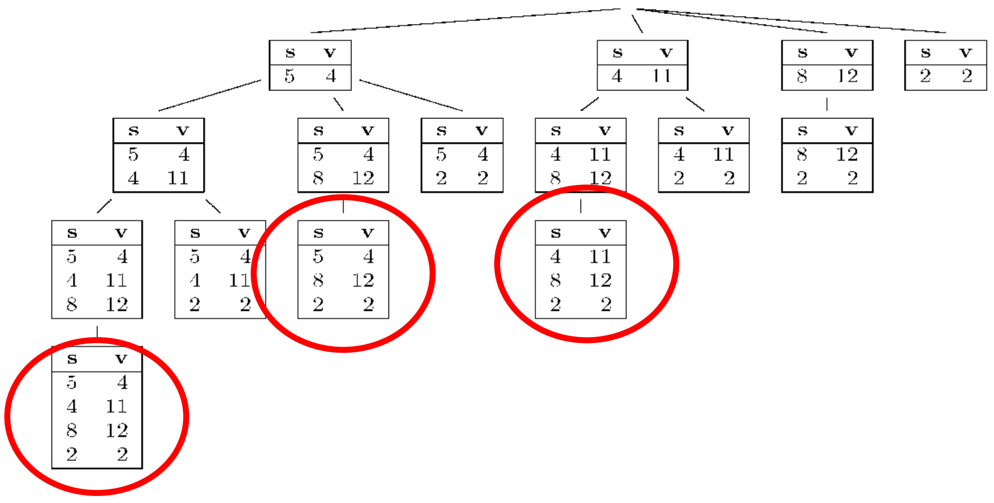
The question required students to write pseudocode for an algorithm that generates a search tree in the same style as Rosa's. The search tree provided in the question represents an exhaustive search of all states, including those that exceed the capacity of the knapsack, therefore it was unnecessary for students to consider the capacity in the logic of their algorithm.

An exhaustive search algorithm like depth-first search can serve as the basis of a solution. Some students could benefit by taking a modular approach to the design of their solutions.

Question 7b.

Marks	0	1	Average
%	34	66	0.7

The answer was:



Question 8a.

Marks	0	1	2	Average
%	25	44	31	1.1

The recurrence relation for the running time of search is $T(n)=2T\left(\frac{n}{2}\right)+O(n)$. Giving O¿).

An example input for the worst case is a `Value` of `7`, with `List = [1,2,3,4,5,6,7]`

Question 8b.

Marks	0	1	2	Average
%	25	41	35	1.1

In the best case, the item is found in the first call of the algorithm. `split` is called once, with time $O(n)$, and then the item is found in the first comparison.

The best-case time complexity is $O(n)$. This occurs when `Value` is equal to the center item of `List`.

Question 9a.

Marks	0	1	2	3	Average
%	17	35	15	33	1.7

Sample answer:

The function `isAdjacent` loops over a list of edges, which is at most $(n-1)$ in length. Hence, it will run in $O(n)$.

The function `floydWarshall` loops over the list of nodes in three nested loops, hence the inside body of the loop is run n^3 times. The body includes two calls to `isAdjacent` and two appends to `edge` lists. Hence, the body has a running time that is $O(n)$.

The overall time complexity of the algorithm is $O(n^4)$.

The question required students to analyse and determine the time complexity of the algorithm. High-scoring responses identified the key features of the algorithm that contributed to the time complexity and detailed their impact.

Question 9b.

Marks	0	1	Average
%	48	52	0.5

Complete graphs will have a worst-case asymptotic running time of $O(n^4)$. For complete graphs, `isAdjacent` loops over a list of $(n-1)$ edges in length each time it is called.

Question 9c.

Marks	0	1	Average
%	71	29	0.3

A possible answer was:

Philippe could store the edge data as an adjacency matrix so that checks for adjacency can be performed in $O(1)$ time.

Students who scored highly drew on their knowledge of the Floyd-Warshall algorithm in answering this question.

Question 10

Marks	0	1	2	Average
%	48	14	38	0.9

Acceptable answers took the form of:

- X as (blank, <any direction>, halt)

and

- Y as (1, <any direction>, halt) or (1, right, A)

Question 11a.

Marks	0	1	2	Average
%	17	33	50	1.4

A possible answer was:

- Problem 1: Maps can be modelled as graphs. Not every graph is 3-colourable, so there is no guarantee that she can colour every map as stated.
- Problem 2: 3-colouring is NP-complete, and so the best-known algorithms will take exponential time, making it infeasible to do this for maps of any significant size.

A common misconception was that the number of regions adjacent to a particular region is relevant. The error in this thinking can be seen through a counterexample of a circle surrounded by 10 regions of alternating colours in a ring.

Question 11b.

Marks	0	1	2	Average
%	14	41	45	1.3

A possible answer was:

- Problem 1: Jia can use four colours (say green, gold, purple and blue), as every map can be 4-coloured, or restrict the map arrangements allowed to only those that are 3-colourable.
- Problem 2: Jia will have to limit the size of each map to something fairly small in order to make the problem feasible, or limit the time allowed for colouring and reject any map that takes too long to colour.

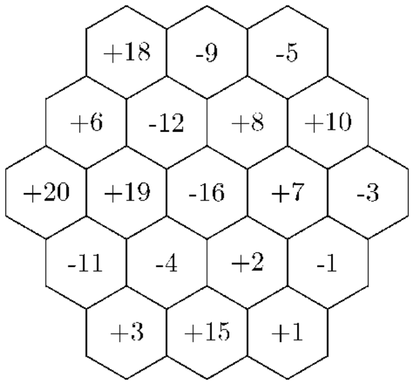
Question 12ai.

Marks	0	1	2	3	Average
%	9	20	49	22	1.9

A number of acceptable approaches were possible for this question. Responses were awarded marks for describing how a single ADT or a combination of ADTs could be used to model data about the tiles in a HexoReverso game. High-scoring responses clearly described how rows were to be identified.

Three distinct example answers relating to different acceptable approaches are provided below:

- Two-dimensional array solution. Store the tile values in a two-dimensional grid where each row of the HexaReverso board is stored in a row of the grid.



+18	-9	-5	0	0
+6	-12	+8	+10	0
+20	+19	-16	+7	-3
0	-11	-4	+2	-1
0	0	+3	+15	+1

- Graph of tiles solution. Store the board as an undirected graph in which each tile in the game board is represented by a node with a value equal to the value of the tile. Adjacent tiles are connected by an edge and this edge is labelled with the orientation of that edge within the game board, with three possible orientations.
- Graph of rows solution. Store the board as an undirected, weighted graph in which each row of tiles is represented by a node and the node stores the current value of all tiles in that row. Rows that share tiles are connected by an edge and the weight of that edge is the cost of flipping the tile shared by the two rows.

Question 12aii.

Marks	0	1	2	3	Average
%	20	24	33	23	1.6

Responses were rewarded for providing an explanation of how the flip operation would be performed within the data structure that a student had described in part ai. Responses needed to include the concept of multiplying the row values by -1.

These three distinct example answers follow the respective different example answers given above:

- Two-dimensional array solution. The flip operation would be performed by multiplying any orthogonal row, column or top-left to bottom-right diagonal within the table by -1.
- Graph of tiles solution. To flip a row, find the node from that row within the graph and flip that node and all other nodes adjacent to it only by edges of the relevant direction, where flipping it involves multiplying the nodes value by -1.
- Graph of rows solution. To flip a row, multiply the value of that node by -1 and then for each edge of that node, subtract the cost of the edge from the corresponding neighbour's value. Lastly, adjust the cost for each edge by multiplying it by -1.

Question 12bi.

Marks	0	1	2	Average
%	39	35	26	0.9

A possible answer was:

Large boards, hundreds of tiles wide, will have hundreds of possible rows to flip. With n rows there are 2^n possible arrangements. This exponential number of possible states will cause an exhaustive search to be infeasible.

Question 12bii.

Marks	0	1	Average
%	27	73	0.8

A correct answer was:

A limitation of a heuristic approach is that an optimal solution will not necessarily be found.

Question 12ci.

Marks	0	1	Average
%	58	42	0.4

A possible answer was:

Select a possible row to flip at random and flip that row.

Question 12cii.

Marks	0	1	Average
%	69	31	0.3

The question required students to describe how Kim could modify the parameters of the algorithm to address the issue of it only accepting candidate solutions that are improvements on the current solution. The key parameter to be modified is the temperature, by increasing it, and this needed to be mentioned explicitly.

Question 13a.

Marks	0	1	2	3	Average
%	30	29	26	15	1.3

The question asked students how Ishan could refine his initial statement to Yan to better explain what he has learnt about decidability and refute Yan's counterexample. Many students provided general statements about decidability and related theory rather than carefully addressing the question. High-scoring responses touched on the following themes:

- It is not possible to write a program that can determine the correctness of an arbitrary mathematical statement.
- The CAS calculator can only determine the truth of some mathematical statements, not all.
- For example, your CAS calculator would not be able to solve the Halting problem.

Question 13b.

Marks	0	1	2	3	4	Average
%	23	19	20	22	16	1.9

A possible answer was:

Ishan is incorrect because Turing did not show that Turing machines could only run polynomial time algorithms, only that there existed at least one problem that they could not solve. Hilbert wanted an algorithm that could be used to determine whether a statement was universally true or not. Turing's work on Turing machines was part of efforts to define what was meant by the term 'algorithm'.

Some common misconceptions included that Turing machines utilise infinite time and space and Hilbert's Program being treated as a computer program, including statements to the effect that Hilbert's Program is undecidable.

Question 13c.

Marks	0	1	2	Average
%	15	32	53	1.4

The Chinese Room Argument shows that a formal system can produce the appearance of understanding Chinese without there being true understanding of the Chinese language. The Chinese Room thought experiment is analogous to current approaches to AI, in that such approaches are limited to the manipulation of symbols according to syntactic rules, but do not involve understanding of semantics. As strong AI presupposes such understanding, the Chinese Room Argument makes the case that strong AI is not possible, at least based on current approaches.

The question required students to explain how the Chinese Room Argument is used to argue against the possibility of strong AI. The question was generally answered well, though some students went into detail about the particulars of the Chinese Room Argument without linking back to it serving as an argument against the possibility of strong AI.