2023 VCE Algorithmics (HESS) external assessment report

General comments

Students performed well in Section A of the 2023 Algorithmics examination, with all but two questions answered correctly by more than 50% of the students. Question 9 of Section A could be best attempted by drawing a recursive tree, which is an important skill that students would benefit from practicing.

Many questions in Section B of the examination were structured in such a way as to scaffold students in their engagement with a given problem context. Many students did not seem to fully capitalise on this fact. For example:

- Question 4ci. provided pseudocode for an algorithm that closely resembled the structure of the inductive argument that students needed to demonstrate in Question 4cii.
- Question 6 involved a problem context with two competing optimisation goals. It then directed students to engage with each one in turn before considering them both together.
- Question 10 was gradually scaffolded to allow students to make connections between the problem context and the graph colouring problem in order to analyse the problem more deeply and discuss algorithmic solutions.

In examination questions that require students to provide answers in pseudocode it is quite common to include helper functions that students are expected to incorporate into their responses. These functional abstractions are often poorly engaged with by students. Interestingly Question 1 of Section B required students to engage with the concept of functional abstractions by describing a motivation for using them. In the context of answering an examination question in pseudocode:

- functional abstractions reduce the complexity, allowing students to focus on the key ideas being examined in a question
- improve readability, allowing assessors to more easily interpret a student's pseudocode.

Question 2c. included the helper function *isPrereq(P,a,b)*, Question 5c. included the helper function *nodePR(G, r, u)*, and Question 6b. included the helper function *Prim(G)*. Students should carefully study such questions to ensure that they can appropriately utilise the provided functions in their responses, and correctly interpret what they are being asked to do in answering the questions.

The nature of the subject and the number and sophistication of problem contexts that students need to engage with during the examination means that time management is an important consideration. Students should be strategic in ensuring that they give themselves sufficient time on particularly weighty questions such as Question 10 in Section B.

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	C	11	1	**86**	2	
2	B	1	**91**	4	4	
3	C	12	5	**82**	1	
4	C	4	0	**91**	5	
5	D	4	9	2	**85**	
6	B	7	**80**	10	2	
7	A	**76**	7	9	7	
8	D	6	11	7	**76**	
9	B	22	**48**	23	6	To identify the number of redundant recursive calls, students needed to draw the recursive tree.
10	A	**47**	12	13	28	The values in the array correspond to the number of coins required to make change for amounts of 0, 1, 2, 3, 4, 5 and 6, respectively. Hence [0, 1, 2, 1, 1, 2, 2].
11	C	2	7	**56**	34	Option C describes a pure hill climbing algorithmic approach. Option D describes a simulated annealing algorithmic approach. While simulated annealing does incorporate hill climbing, the question requires students to identify the answer that best describes the approach of hill climbing on a heuristic function.
12	B	3	**55**	0	42	Option B has the largest Big-O complexity. Option D is incorrect because 0.5^n indicates an exponential decay.
13	D	6	2	3	**89**	
14	C	14	14	**63**	9	
15	D	2	3	14	**81**	
16	C	2	1	**97**	0	
17	B	11	**57**	24	8	
18	A	**94**	2	3	2	
19	A	**55**	12	7	27	
20	B	4	**84**	9	4	

Section B

Question 1

Marks	0	1	2	Average
%	51	20	29	0.8

A possible answer could be:

One motivation for using functional abstractions when designing algorithms is to enable a more modular approach to algorithmic problem-solving by decomposing problems into smaller parts that can be solved independently.

Some students described the process of abstraction in relation to identifying salient information in a problem context, rather than using functional abstractions when designing algorithms.

Question 2a.

Marks	0	1	2	3	Average
%	10	13	39	38	2.1

A directed graph could be used to model the project. Each task is represented by a node, with the cost and number of people required stored as node attributes. For each task and immediate prerequisite task pair, connect a directed edge from the immediate prerequisite task to the task.

High-scoring responses explicitly mapped the given features of the problem to the data structure.

Question 2b.

Marks	0	1	2	Average
%	7	78	15	1.1

A correct answer would be:

add_new_task: graph × element x integer × integer × list → graph,

where graph refers to the project, element refers to the task identifier, the integers refer to time and people, and list refers to the immediate prerequisite tasks.

To achieve full marks, students should avoid using variable names such as ‘minutes’ and ‘people’ in signature specifications.

Question 2c.

Marks	0	1	2	3	Average
%	24	46	17	13	1.2

Sample answer:

```
Algorithm requiresTask(P, a, b):
  Foreach u in P do
    If isPrereq(P, a, u) then
      If u = b then
        Return True
      Elseif requiresTask(P, u, b) then
        Return True
  Return False
```

Some students did not make use of the *isPrereq(P,a,b)* function. Some students incorrectly tried to write pseudocode for the *isPrereq(P,a,b)* function. Many students did not realise that a prerequisite may not be an immediate prerequisite.

Question 3a.

Marks	0	1	Average
%	64	36	0.4

The number of possible trips is proportional to the factorial of the number of shops. This would prevent the data model from being used due to practical space constraints. This is an example of the soft limits of computability.

Some students talked about issues that related to whether or not the proposed data model was the most suitable structure but did not engage with notions that would prevent the proposed data model from being used.

Question 3bi.

Marks	0	1	2	Average
%	41	30	30	0.9

Sample solution:

Add the weights of all ingoing edges for every node. The node with the largest sum is the most visited shop. Exclude the 'outside' node. Each node has *n-1* incoming edges, and hence this operation would take $O(n^2)$ time to execute.

Many students correctly described summing the weights of all ingoing edges for every node but did not explicitly state that the node with the largest sum is the most visited shop.

Question 3bii.

Marks	0	1	2	Average
%	78	11	11	0.4

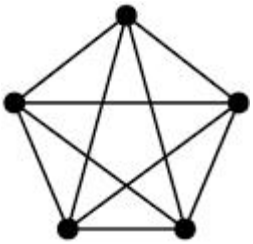
Sample solution:

Change the graph to be undirected and change the weight of each edge (u, v) to be the number of visits to the shopping centre that contain both the shops u and v.

Many students provided answers that did not involve modifying the given data model. Moreover, some answers only accounted for pairs of shops that were visited immediately after one another.

Question 4a.

Marks	0	1	2	Average
%	9	24	67	1.6



A complete graph is a graph in which an edge exists between all pairs of nodes.

Some students provided a definition of the class of connected graphs rather than the class of complete graphs.

Question 4b.

Marks	0	1	2	Average
%	23	32	45	1.2

Sample solution:

The original graph is a complete graph and therefore contains an edge for all pairs of nodes. Hence, the sub-graph with nodes U will contain an edge for all pairs of nodes in U and the sub-graph with nodes V will contain an edge for all pairs of nodes in V. Hence, these sub-graphs will also be complete graphs.

This question was generally answered well, although some students provided vague responses or conflated the notion of complete with the notion of connected.

Question 4ci.

Marks	0	1	Average
%	33	67	0.7

$V\_in = \{F\}$

$V\_out = \{B, C, D, E\}$

Question 4cii.

Marks	0	1	2	3	4	Average
%	41	25	17	14	2	1.1

Sample solution:

Base case

- For a graph with 0 nodes, an empty path contains all nodes in the graph.
- For a graph with 1 node, a path containing only that one node contains all nodes in the graph.
- Therefore, the statement is true for graphs with $n \leq 1$ nodes.

Induction

- Assume that all such graphs with $n \leq k$ nodes contain a path with all nodes.
- Consider a graph with $k + 1$ nodes.
- Select an arbitrary node *v* and divide the other vertices into two groups based on whether their edge to *v* connects inwards to *v* or outwards from *v*. Call these groups *v_in* and *v_out*.
- The subgraphs with the vertices *v_in* and *v_out* both have at most k vertices, and therefore contain a path containing each vertex.
- A path can be created using *path_in* → *v* → *path_out*. Therefore, if the statement is true for $n \leq k$, then it is true for $n = k + 1$.

Conclusion

- Since there is a path for n $= 0, 1$ and for $n = k + 1$ if $n \leq k$, then we have shown that there is such a path for $n \geq 0$.

Many students did not leverage insights that could be gleaned from the findPath(G) algorithm provided in Question 4ci. in writing their inductive argument. For example, the base case could be understood from lines 1 to 4 of that algorithm. Some students did not follow the structure of an inductive argument or left out necessary steps.

Question 5a.

Marks	0	1	2	Average
%	26	64	10	0.9

When explaining how one problem is analogous to another problem, it is important to clearly identify relevant similarities between the problems.

With regards to the problem of estimating the importance of urban spaces based on their connectivity and the problem of ranking the importance of web pages, relevant similar features include:

- the urban spaces and web pages
- paths between urban spaces and web links
- the likelihood that an imaginary pedestrian will keep going to another urban space and the likelihood that an imaginary web surfer will keep clicking links.

Many students provided vague answers that did not carefully relate the similar features of the two problems.

Question 5b.

Marks	0	1	Average
%	22	78	0.8

This question was answered well. Given that there are four nodes in David's graph, each node would be initialised to $\frac{1}{4}$.

Question 5c.

Marks	0	1	2	3	4	Average
%	37	26	17	11	10	1.3

For this question students were given a function *nodePR(G, r, u)* that returns the updated PageRank of node *u* based on PageRanks calculated in the previous iteration of the PageRank algorithm.

A sample solution could be:

```
Algorithm pageranks(G):
  n ← number of nodes in G
  r1 ← new dictionary
  r2 ← new dictionary
  Foreach u in G do
    r1[u] ← 1/n
    r2[u] ← None
  While r1 is not equivalent to r2 do
    Foreach u in r1 do
      r2[u] ← nodePR(G, r1, u) round to a set number of decimal places
    Swap r1 and r2
  Return r1
```

High-scoring responses clearly initialised appropriate variables and data structures, and the starting PageRanks of each node. They tracked the old and new PageRanks at each iteration. They utilised the *nodePR(G, r, u)* function. They recognised that PageRank is a convergence algorithm and established a terminating condition based on a difference function, a rounding parameter, or a set number of iterations.

Question 6a.

Marks	0	1	Average
%	15	85	0.9

This question was answered well.

Sample solution:

A functional solution requires a spanning tree. To minimise the total cost of the pipes, the total length of piping used must be minimised. Prim's algorithm applied to the graph would find a minimum spanning tree. A minimum spanning tree is a solution that involves the least total length of pipe that spans all nodes.

Question 6b.

Marks	0	1	2	3	4	Average
%	61	20	12	5	1	0.7

Students were given a function *Prim(G)* that returns a minimum spanning tree of a weighted graph. The question required them to write pseudocode for an algorithm that returns all possible minimum spanning trees. A number of different approaches could be taken to writing such an algorithm, including a brute force approach. The sample solution provided here exploits the property that more than one minimum spanning tree can only exist when a graph contains duplicate edge weights.

```
1     ALGORITHM MSTs(G)
2     BEGIN
3       cost ← edge weight sum of Prim(G)
3       MST = {} //initialised as an empty set
4       Foreach edge in G do
5         Foreach Edge in G do:
6           If edge is not equivalent to Edge then
7             If weight of Edge is equivalent to weight of edge do
8               If G excluding edge is connected do
9                 Add Prim(G excluding edge) to set MST
10              If G excluding Edge is connected then
11                Add Prim(G excluding Edge) to set MST
12      Foreach T in MST do
13        If edge sum of T is not equivalent to cost then
14          Remove T from MST
12      Return MST
```

Some students incorrectly attempted to provide pseudocode for Prim's algorithm.

Question 6c.

Marks	0	1	2	Average
%	41	33	26	0.9

A suitable algorithmic approach needs to return the single-source shortest path tree, where the source node corresponds to the central hot water system and the tree shows the shortest paths to each node in the graph, where those nodes represent the apartments. Both Dijkstra's algorithm and Bellman-Ford's algorithm would need to be modified to return the single-source shortest path tree. The modification to Bellman-Ford's algorithm would be more straightforward and would not impact its time complexity. The modification to Dijkstra's algorithm could be done in a number of different ways resulting in different time complexities.

Moreover, the time complexity may vary due to graph properties such as maximum path length or edge density.

Answers involving a modified Bellman-Ford algorithm or a modified Dijkstra's algorithm were accepted.

Question 6d.

Marks	0	1	2	3	Average
%	60	23	13	4	0.7

Central to understanding this question is realising that there are two competing optimisation goals that may require a trade-off. Some students mistakenly treated this as an intractable problem that required approaches such as simulated annealing. Rather, it required an algorithmic approach that attempted to return an acceptable compromise solution.

A possible answer could be:

Apply a modified Bellman-Ford's algorithm to return a single-source shortest path tree. Apply Prim's algorithm to find a minimum spanning tree (MST). Determine the longest path length from the central hot water system to an apartment in the single-source shortest path tree. Compare that to the path length from the central hot water system to that apartment in the MST. If the former is shorter than the latter, replace the latter in the MST with the path from the single-source shortest path tree. Remove any edges not part of that added path that make a cycle. Repeat the process so long as there are paths in the MST from the source node that are longer than the longest path length from the source node in the single-source shortest path tree. In the worst case this will return the single-source shortest path tree. In the best case the tree will be equal to or only slightly more in cost than the MST, but there will be no path lengths longer than the longest path in the single-source shortest path tree, which seems like a fair and reasonable compromise solution.

Question 7a.

Marks	0	1	2	Average
%	32	36	32	1.0

If the bag contains 0 or 1 ball, return the bag as the output.

Question 7bi.

Marks	0	1	Average
%	40	60	0.6

Since all bags have a similar number of remaining balls, each has approximately $\frac{n-2}{3}$ or $\frac{n}{3}$ balls. The machine is called recursively on the three bags, giving $3\,S\left(\frac{n}{3}\right)$. To sort the balls takes $O(n)$ comparisons. This gives $S(n) = 3S\left(\frac{n}{3}\right) + O(n)$.

Many students did not explain the $O(n)$ term.

Question 7bii.

Marks	0	1	2	Average
%	10	5	84	1.8

This can be solved using the Master Theorem, with *a = 3, b = 3 and c = 1.*

This yields $O(n \log(n))$.

Question 7ci.

Marks	0	1	Average
%	51	49	0.5

The machine takes $O(n)$ time to sort the balls into three bags; with sizes 0, 0 and n − 2. This gives $T(n) = T(0) + T(0) + T(n-2) + O(n)$

$T(0)$ takes $O(1)$

Hence, $T(n) = O(1) + O(1) + T(n-2) + O(n)$

$= T(n-2) + O(n)$

Question 7cii.

Marks	0	1	2	Average
%	58	17	25	0.7

From part ci., T(n) = T(n – 2) + $O(n)$= T(n – 4) + $O(n-2)$ + $O(n)$ = sum of all odd or even numbers < n (depending on whether n is odd or even). This is approximately half of the sum of all numbers < n.

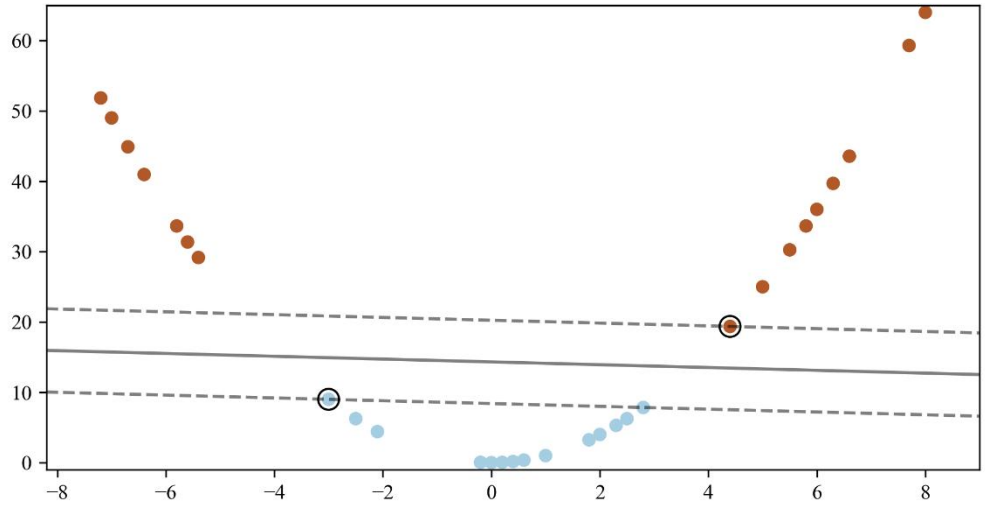
So, it is approximately $\frac{n^2}{4}$. So, $T(n) = O(n^2)$.

Question 8a.

Marks	0	1	2	Average
%	28	28	44	1.2

Sample solution:

This data is not separable by a single point, and hence there is no one-dimensional support vector machines (SVM). However, by squaring the data, and plotting the rainfall measurement against its square, we can find



a separator that is a straight line.

Many students answered this question correctly, though some students gave general explanations of SVMs and did not engage with the specific question context.

Question 8b.

Marks	0	1	2	Average
%	14	33	53	1.4

An example of a technical risk could be that data obtained from community volunteers may be noisy and inaccurate. And an example of an ethical risk could be that there may be data privacy breaches when sharing the model to other communities.

This question was generally answered well, with a number of technical and ethical risks being acceptable.

Question 8c.

Marks	0	1	2	3	Average
%	23	31	26	20	1.5

Sample solution:

Perfect Tomatoes is an example of what Searle would describe as weak AI because while it is trained on data, it is limited to manipulating symbols and rules to achieve its goals. Perfect Tomatoes is not an example

of what Searle would describe as strong AI because it lacks mental states and understanding as to the meaning of the rules and data.

Many students answered this question well. Some students misclassified Perfect Tomatoes. Some students conflated the notions of weak and strong AI with the notions of narrow and general AI. While these conceptions share similarities, they are not entirely the same.

Question 9a.

Marks	0	1	2	3	Average
%	13	33	37	17	1.6

Sample solution:

Undecidable problems are decision problems for which there exists no computational formalism that can always decide the problem. This means that there are fundamental limitations on what software can do, as no program can be written that can always decide an undecidable problem. A classic example of an undecidable problem is the Halting Problem, i.e. given the description of a program and its input, can we determine whether the program will eventually finish running or will run forever?

Many students engaged with some, but not all, elements of the question. In order to be awarded full marks students needed to convey an understanding of undecidable problems, provide an example such as the Halting Problem, and relate their understanding to the notion that software is limited in what it can do.

Question 9b.

Marks	0	1	2	3	Average
%	41	30	21	9	1.0

Sample solution:

Hilbert and Ackermann, leading mathematicians of the early 20th century, were keen to establish the verity of the Entscheidungsproblem, i.e. whether or not there was an algorithm that could determine the truth of any mathematical statement. This could act as a ‘mathematical umpire’ in case any dispute arose. Both Church and Turing independently proved that there is no such algorithm. Both proofs involved developing formal models of computation (lambda-calculus for Church, Turing machines for Turing), and then using these formalisms to show that the Entscheidungsproblem was undecidable.

Question 9c.

Marks	0	1	2	Average
%	58	26	15	0.6

Sample solution:

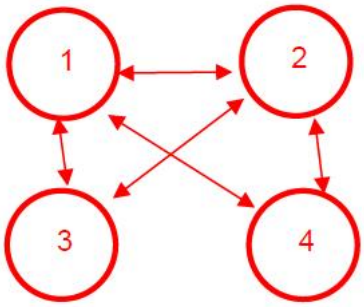
Undecidable problems run into the hard limits of computability; there are also intractable problems that run into the soft limits of computability. Intractable problems are problems that cannot be solved in polynomial time. These problems are impractical to solve for large inputs due to the significant computational resources required.

Some students did not consider the soft limits of computation in formulating their response.

Question 10a.

Marks	0	1	Average
%	53	47	0.5

The correct answer was:



Question 10b.

Marks	0	1	Average
%	31	69	0.7

Sample solution:

The graph colouring problem is the problem of determining the least number of colours to colour the vertices of a network graph such that no adjacent vertices share the same colour.

Question 10c.

Marks	0	1	2	Average
%	48	37	15	0.7

Sample solution:

The timetabling problem can be mapped to the graph colouring problem by constructing a network graph where the vertices represent programs, and the edges represent conflicts such as if instructors, members, or workout rooms are assigned to multiple programs. In order to solve the timetabling problem, courses with conflicts need to be in different blocks. The blocks are represented by the colours, i.e. no conflicting course can be in the same block or have the same colour.

Many students did not explicitly map all features of the problem context to the graph colouring problem, and instead provided vague responses.

Question 10d.

Marks	0	1	2	Average
%	66	28	6	0.4

Sample solution:

This problem is analogous to the decision version of the graph colouring problem because the timetable is limited to 6 blocks. The decision version of the graph colouring problem is NP-Complete, hence this problem is NP-Complete. Solutions to NP-Complete problems can be verified in polynomial time, and all problems in NP can be reduced to NP-Complete problems in polynomial time.

Many students were unable to correctly state the complexity class of the problem despite being scaffolded to appreciate that the problem is analogous to the graph colouring problem. Students should be able to categorise problems mentioned in the study in terms of their complexity classes.

Question 10e.

Marks	0	1	2	3	4	5	6	7	8	9	10	Average
%	23	9	14	12	16	12	9	4	2	0	0	2.9

This question required students to demonstrate an understanding of both the backtracking and simulated annealing algorithm design patterns and how they could be applied to the given problem. High-scoring responses provided clear and detailed explanations of the application of these patterns to the problem. Moreover, students were expected to demonstrate a complex understanding of how changes to the size of the problem variables would impact the practical applicability of each design pattern, including a time complexity analysis.

Finally, high-scoring responses were required to develop a cogent, well-substantiated recommendation for which approach would be best, taking into consideration factors such as the quality of the solution achieved and the feasibility of determining a solution and giving consideration to particular situations that would merit each option.

Sample response:

With regards to the backtracking approach, consider a state space tree such that nodes represent different colourings of the graph. Use depth-first search (DFS) to generate the state space tree by exploring the solution space, i.e. the root node of the tree would relate to an uncoloured graph; neighbouring nodes of the root node would pertain to the 6 possible colours of some node ‘1’; subsequent neighbouring nodes of the node ‘1’ row of the tree would entail all possible colourings of some node ‘2’, which is a neighbour to node ‘1’. If neighbouring nodes in a state have a colour clash, then DFS backtracks, otherwise DFS keeps exploring neighbours until reaching a state where all nodes in the graph are coloured, i.e. an optimal solution. Backtracking does not result in time complexity improvements that are orders of magnitude better than a brute force approach, and so a solution may not be found by this method in a reasonable amount of time. Given that there are 6 timetable blocks, m, and $|V|$ courses the time complexity would be $O(m^{|V|})$, i.e. $O(6^{|V|})$. If the gym is offering only a small number of courses, this approach may be viable. Further efficiencies may be capitalised on by exploiting symmetry in the state graph.

The Intractable nature of the problem may lend itself to a randomised metaheuristic approach such as simulated annealing, which would return a non-optimal solution, but in a reasonable amount of time. In relation to the graph colouring problem the simulated annealing approach works by starting with a randomised colouring of the graph; it then randomly generates a neighbouring solution, i.e. one that differs by one node colouring. If the solution results in less clashes, it is accepted. If it results in the same number or more colour clashes, it is accepted with some probability less than 1. The probability decreases exponentially

with the ‘badness’ of the move – the amount by which the evaluation is worsened. The probability also decreases as the ‘temperature’ goes down: ‘bad’ moves are more likely to be allowed at the start when the temperature is high, and they become more unlikely as the temperature decreases. The runtime of this approach is dependent on the cooling schedule selected.

It is important to realise that a 6 colouring solution may not exist for the problem instance. Therefore, even in the case of backtracking the solution may still have clashes that will need to be addressed through staffing, making changes to how facilities are used, or not meeting all members’ program enrolment preferences. Certainly, the simulated annealing solution is likely to be suboptimal, i.e. still have clashes that will need to be resolved through resource management by the gym.

As to which solution is most appropriate in this problem context: a gym may only offer a relatively small number of courses. A reasonably powerful computer could be tasked with the computation and given a decent window of time. It may well be that backtracking would be possible in this instance.