2020 VCE Algorithmics (HESS) examination report

General comments

In 2020 the Victorian Curriculum and Assessment Authority produced an examination based on the *VCE Algorithmics (HESS) Adjusted Study Design for 2020 only*.

The 2020 VCE Algorithmics (HESS) examination contained two sections. Section A was composed of 20 multiple-choice questions, and Section B was composed of 13 short-answer and extended-answer questions. Students achieved scores across the range of available marks, with higher-scoring students demonstrating an impressive grasp of challenging content.

The majority of students attempted most of the questions on the paper, demonstrating a thorough engagement with the study design. Additionally, many students heeded what each question required of them, usually answering questions in their entirety. There were relatively few responses relying on memorisation alone.

This year, students were required to read (Questions 7a., 9b. and 9c.) and write (Questions 6a., 6d., 8b., 9a. and 9d.) algorithms that often required both a deep content knowledge and the ability to apply it in novel contexts (whether in pseudocode or in English), and overall this was done to a high standard. The quality of student pseudocode was high, with Question 6d. a shining example of students' capacity to express a given idea in pseudocode. The use of looping was done to a high standard in most responses. It was evident, however, that students were much more comfortable applying Unit 3 concepts such as shortest path and search algorithms than Unit 4 concepts, where the understanding shown tended to be more cursory. In particular, minimax (Question 9d.) and randomised heuristic (Question 9a.) approaches would benefit from more careful applied study.

Questions 1a., 1b., 2a., 3a. and 4a. required students to provide or use a definition of a key concept from the study. Few students demonstrated a precise understanding of these concepts, with many relying on informal descriptions. While it may seem tedious to focus on definitions, it was evident in later questions that poor understanding of a fundamental concept made it difficult for students to answer applied questions well.

Certain concepts from Unit 4 continue to prove challenging for students, including the NP complexity class (Question 12a.) and decidability (Questions 10a. and 11). Both concepts underpin a good deal of Unit 4 work, and students are advised to work towards developing a precise understanding of each. Students must take care not to consider NP as ‘Not Polynomial’ and avoid conflating decidability with tractability or feasibility.

Students who answered multi-part questions using bullet points tended to write clearer and more concise responses than their peers. The use of illustrative diagrams or examples (particularly in Questions 2b., 9d. and 13) was also a common feature of some of the highest scoring responses. While neither is strictly necessary, practising the use of bullet points, diagrams and examples in responding to questions throughout the year would aid students to clearly articulate their thinking in the examination.

Specific information

Student responses reproduced in this report have not been corrected for grammar, spelling or factual 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 following table indicates the percentage of students who chose each option. The correct answer is indicated by shading.

Question	% A	% B	% C	% D	Comments
1	78	12	4	6	
2	3	72	22	3	
3	0	12	4	85	
4	4	27	7	62	
5	63	4	6	27	
6	6	49	13	29	Students are encouraged to draw a Venn diagram or similar.
7	8	11	79	3	
8	9	62	15	14	
9	21	23	6	50	PageRanks are probabilities, so should sum to 1 after each iteration.
10	27	17	42	14	$\text{PageRank}_2(C) = \frac{0.15}{5} + 0.85\left(\frac{PageRank_1(A)}{2}\right)$ $= 0.03 + 0.85\left(\frac{0.2}{2}\right) = 0.115$
11	5	5	88	1	
12	31	11	55	3	$2n^3 + n^2 + 5$ is $O(n^4)$ as Big-O describes an upper bound.
13	14	4	82	0	
14	31	63	2	4	
15	23	13	9	54	Drawing a recursion tree would help visualise the number of *s printed.
16	64	33	1	2	
17	13	5	69	13	
18	7	66	9	18	
19	1	33	48	18	Real numbers are not computable on a Turing machine as they are uncountable.
20	79	3	13	5	

Section B

Question 1a.

Marks	0	1	2	Average
%	9	28	63	1.5

A dictionary abstract data type (ADT) is a container for key-value pairs. It is used where one of the pairs (the key) is commonly used to look up the associated value. An English dictionary could be modelled with a dictionary ADT with words as keys and their definitions as the values.

Many students achieved full marks on this question. Some students omitted the example or did not explain the central idea of key-value pairs. As ADTs are central to much of the course, students are encouraged to work towards a precise understanding of these and practise considering real-world examples where these ADTs would be suitable.

Question 1b.

Marks	0	1	2	3	Average
%	9	19	24	48	2.1

The following is an accepted complete signature specification for a dictionary ADT.

newDict(): → dictionary

add(): key x value x dictionary → dictionary

update(): key x value x dictionary → dictionary

remove(): key x dictionary → dictionary

get(): key x dictionary → value

size(): dictionary → integer

It is not an exhaustive signature specification (other operations could be included). Higher scoring responses included a constructor operation, one or more manipulators (add, update, remove) and one or more property operations (get, size). Some students omitted a constructor. It is good practice to write a constructor as the first operation in a signature specification to avoid this omission as a matter of course.

Lower scoring responses tended to demonstrate confusion about what operations were standard for a dictionary ADT or were unable to correctly specify the inputs and outputs of the operations.

A number of different names for the same operation were seen. For instance, the get() operation was often referred to as lookup(). No preference was given to particular names of operations as long as it was clear what the operation would do.

Question 2a.

Marks	0	1	Average
%	12	88	0.9

The overwhelming majority of students were able to recognise that the first in, first out structure of a queue ADT was what made it suitable. A small number of responses described a queue in the informal sense of 'lining up' or simply used the language of the question ('first come, first served') rather than first in, first out. Marks could not be awarded for these responses.

The following is an example of a high-scoring response.

Planes enter airspace on a first come, first served basis, which is the same as the queue's first in, first out nature, so a queue is a suitable ADT.

Question 2b.

Marks	0	1	2	3	4	Average
%	6	20	34	23	17	2.2

A wide variety of responses were seen. Most responses were able to describe a combination of ADTs that stored the appropriate information. However, many responses used a combination of ADTs that would not have allowed an air traffic controller to efficiently access the three factors described. For example, some students suggested the use of three priority queues, each providing an ordering of the aeroplanes on the basis of a given factor. This is not suitable, given queues only provide access to the front element, making it impossible to calculate an overall priority for each aeroplane.

Students should also take care to ensure they clearly specify how a particular ADT is modelling the aspect of the problem they are describing. When describing a dictionary, for instance, it is crucial that students make clear what the key-value pairs will be. A few students provided a small example of their combination of ADTs; while this is not strictly necessary, it can be a good way to illuminate their response and to ensure their approach would be a workable one. Students who provided such an example tended to score highly. The following is an excerpt from a response that used such an example.

the information about each plane can be represented in a dictionary e.g. {Qantas 453 : [0.7, 15, true], Jetstar 916 : [0.4, 36, false], Virgin 172 : [0.05, 0, false]}

The following is an example of a high-scoring response.

A dictionary should be used to store details about each plane, the keys could be "fuel", "delay" and "local" where "fuel" is associated with the amount of fuel in litres on the plane, "delay" stores the number of minutes the plane is delayed, and "local" is associated with a Boolean which is only true if the plane is local. A maximum priority queue could be used to score the airplanes before they are approved to land, by assigning the priorities using the values in the dictionary and a math function, the planes with higher priority can be permitted to land first.

Question 3a.

Marks	0	1	2	Average
%	16	65	19	1.0

A directed graph ADT is composed of a vertex set containing all vertices in the graph and an edge set containing ordered pairs of vertices. The edge is directed from the first vertex in the ordered pair to the second edge in the ordered pair.

While most students showed an informal understanding of what a directed graph ADT was, few were able to use subject-specific terminology to describe it. Responses that attempted to use the term ‘directed graph’ as a key part of their definition did not score well.

Question 3bi.

Marks	0	1	2	Average
%	5	20	75	1.7

This question was done well by most students. A common oversight was simply stating what the directedness of the edge would model without engaging more broadly with the requirement to provide some reasoning.

The following is an example of a high-scoring response.

Roads can be modelled as edges. Some roads are one-way roads, and directed edges can model these types of roads. This is important for being able to generate a route that adheres to road rules.

Question 3bii.

Marks	0	1	2	Average
%	3	25	72	1.7

As with the previous part of Question 3b., the most common error was stating what the weight of the edge would model without any reasoning.

The following is an example of a high-scoring response.

- *The weighted edges represent the distance between adjacent nodes (where a node is an intersection).*
- *This is useful (for example) for finding shortest paths between two intersections.*

Question 4a.

Marks	0	1	2	Average
%	54	32	15	0.6

The transitive closure of *G* is a graph that contains all vertices and edges in *G* and additional edges such that for any two vertices u and v in *G*, if a path $u \rightarrow v$ exists in *G* an edge uv exists in the transitive closure of *G*.

Common incorrect responses included descriptions of how to find transitive closure and definitions of complete graphs.

Question 4b.

Marks	0	1	2	Average
%	54	31	15	0.6

A range of different search and single-source shortest path algorithms were used in high-scoring responses, executed on each of the nodes as source. Students who did not score well typically omitted any description of how the described algorithms would be used to find the transitive closure.

The following is an example of a high-scoring response.

For each node A and each node B, run DFS to see if A is reachable from B. Create a new graph G to represent the transitive closure and add a directed edge from B to A whenever DFS decides that a node A is reachable from B.*

Question 5a.

Marks	0	1	2	Average
%	14	38	48	1.3

This question was generally answered well. Students are reminded that when discussing a model for a given scenario, it is important that they provide an explicit mapping between the features of the ADT and the scenario. In this instance, this meant clearly stating that edges would represent roads, and nodes would represent intersections and fire stations.

The following is an example of a high-scoring response.

Each station and intersection could be a node on the graph, and the edges connecting them could represent roads connecting intersections/stations. Each of these roads (edges) could be assigned a weight based on how long it takes to travel through as traffic conditions are constant.

Question 5b.

Marks	0	1	2	3	4	5	Average
%	46	17	8	14	6	9	1.4

If the city had very few roads connecting intersections, $|E| \in O(|V|)$ (for a tree, $|E| = |V| - 1$). If the city had roads connecting almost every intersection, $|E| \in O(|V|^2)$ (for a complete graph, $|E| = \frac{|V| \times (|V|-1)}{2}$).

In the case of a city with few roads where $|E| \in O(|V|)$, then:

Dijkstra's: $O\big((|V| + |E|)\log|V|\big) = O\big((|V| + |V|)\log|V|\big) = O(|V|\log|V|)$

Floyd's Algorithm: $O(|V|^3)$

But Dijkstra's must be run on each of the fire stations, so its total running time will be $O(n|V|\log|V|)$ where *n* is the number of fire stations. Even if *n* is very large (i.e. $n \approx |V|$), Dijkstra's would still be preferred to Floyd's as $|V|^2\log|V| < |V|^3$.

In the case of a city with many roads and $|E| \in O(|V|^2)$, then:

Dijkstra's: $O\big((|V| + |E|)\log|V|\big) = O\big((|V| + |V|^2)\log|V|\big) = O(|V|^2\log|V|)$

Floyd's Algorithm: $O(|V|^3)$

Again, as Dijkstra's must be run on every node, the asymptotic running time will be $O(n|V|^2\log|V|)$. In this instance, Dijkstra's would be better if $n\log|V|$ is smaller than $|V|$, while Floyd's would be better otherwise.

A wide range of responses was seen for this question, with some outstanding high-scoring responses demonstrating a very strong understanding of this part of the course. Several common misconceptions were evident, including:

- Giving the lower bound of |E| as O(1). While this is defensible in the abstract, it does not make sense in the context of a road network where nodes represent intersections. Students were still able to score highly despite this assumption; however, they could not achieve full marks.
- Ignoring or missing the bracket around $(|V| + |E|)$ in the given running time for Dijkstra's.
- Not requiring that Dijkstra's be run from every node.

Some high-scoring responses did not consider a separate *n* value for the number of fire stations and instead used |V| throughout in their analysis. This was accepted.

A number of students did not engage with the question.

Question 6a.

Marks	0	1	2	3	Average
%	22	12	26	40	1.8

This question was answered well, with most students demonstrating an understanding of what a brute-force approach requires. There were two common approaches that scored highly.

A brute-force approach that incremented the number of towers starting with a single tower:

- Start with 1 tower. For each integer *k* away from the lighthouse, check if placing 1 tower there provides full coverage. Stop when *k* is larger than the farthest house from the light house.
- Repeat the process with 2, 3, 4 and so on towers, each time generating the complete set of possibilities of tower placement.
- The first time that full coverage is achieved, return the number of towers.

OR

A brute-force approach that generates all possibilities at once, and then searched through those possibilities for the valid solution with the fewest towers. The following is an example of such an approach in a high-scoring response.

TeleG could list every combination of mobile phone towers along the road (for every kilometre k there can be a tower or no tower), and then sort through each one of these combination verifying whether they provide service to all houses in r kilometres. Then, TeleG could choose the combination with the least towers.

Of responses that did not score well, common errors included providing a generic description of brute-force algorithms without engaging with the context and outlining a greedy approach instead of a brute-force approach. Responses using a greedy approach could not be awarded any marks, and students are reminded that it is crucial that they adhere to the algorithm design required by the question.

Question 6b.

Marks	0	1	2	Average
%	24	26	50	1.3

The brute-force approach will run in exponential (or factorial) time as the problem suffers from a combinatorial explosion, and as such it will be infeasible given that the road is more than 1000 km long.

This question was done well by students, with most realising that the approach of generating all combinations of tower placement would not be feasible for the size of the problem.

Question 6c.

Marks	0	1	Average
%	40	60	0.6

Most students recognised the outlined approach as being a greedy design pattern.

Question 6d.

Marks	0	1	2	3	4	Average
%	19	23	23	18	17	1.9

Let `houses` be the ordered array containing the distances to the lighthouse

```
fewest_towers(houses, r)
      tower = houses[0]+r \\ place the first tower r down the road
                          \\ from the first house
      towers = 1
      for house in houses:
            if house > tower + r
                  tower = house + r
                  towers = towers + 1
      return towers
```

There were many excellent attempts at writing the approach outlined in the question as pseudocode, demonstrating that most students had a good facility with the conventions and structures of pseudocode. In particular, the use of indentation and looping structures was good, and few students omitted key steps such as initialising a variable and failing to return a solution.

Common mistakes for otherwise high-level responses included having an incorrect conditional and returning an array of tower locations rather than the number of towers required.

While most students attempted the question using an iterative approach, some responses used a recursive approach, often with good success.

Question 7a.

Marks	0	1	Average
%	32	68	0.7

The algorithm uses a divide and conquer design pattern.

Some responses incorrectly claimed that it was a decrease and conquer design pattern.

Question 7b.

Marks	0	1	2	3	Average
%	33	10	23	35	1.6

Most responses were able to identify the two recursive calls and that the input size in each case was halved. It was common for responses to omit any discussion of the +1 in the first line, and few responses discussed the base case.

The following is an example of a high-scoring response.

The function calls itself twice, each time calling with half the input size, hence the "$2T\left(\frac{n}{2}\right)$*". Each function call runs in constant time outside of the recursion, hence the "*+1*". The base case of the algorithm above runs in constant time, hence* $T(n) = 1$ if $n = 1$.

Question 7c.

Marks	0	1	2	Average
%	29	10	61	1.3

Applying the Master Theorem, we have $a = 2, b = 2, k = 1, c = 0, d = 0$. As $\log_2 2 = 1 > 0$, $T(n) = O(n)$.

While most students were able to achieve full marks in this question, a significant number of students did not engage with the question or misremembered the Master Theorem. Students are reminded that the Master Theorem is provided for them in the instructions for Section B, and they should refer to it when attempting any question involving the Master Theorem.

Question 7d.

Marks	0	1	2	Average
%	40	11	49	1.1

A brute-force search would run in linear time $O(n)$ and hence, based on part c., would have a similar running time to the algorithm given in part a.

Some responses incorrectly assumed a brute-force approach necessarily meant the running time would be exponential or infeasible. Students are encouraged to ensure they consider the algorithm design pattern in context, rather than relying on pre-learnt associations between design patterns and properties thereof.

Question 8a.

Marks	0	1	2	3	Average
%	39	16	40	5	1.1

Many responses were able to identify that the described problem was analogous to the knapsack problem. Lower-scoring responses did not provide a clear description of why this was the case, often simply discussing the knapsack problem in generic terms.

Of those students that did identify the problem as analogous to knapsack and provided a clear mapping to this specific problem, most suggested that the problem was intractable because the best solution for the knapsack problem runs in exponential time or similar. It is important that students are familiar with the standard dynamic programming solution to knapsack, which runs in $O(nW)$ time where *n* is the number of items considered and *W* is the capacity of the knapsack.

The following is an example of a high-scoring response.

This is tractible, as it can be formulated as a modified knapsack where length is the weight, cost is value, and we want to fill the knapsack with at least 3000 weight while minimising value. Knapsack is $O(nw)$, and here $n = 100$ while $w = 3000$, so on the order of 30000 operations is needed which is tractible for computers.

Question 8b.

Marks	0	1	2	3	4	Average
%	18	16	12	23	31	2.3

A greedy algorithm could proceed as follows:

- Calculate a ratio of cost/length of road for each business.
- Sort the businesses according to this ratio.

- Select businesses with the lowest cost/length ratio until 3000 km or more of road has been selected.

This would not produce an optimal result.

This question was done well, with most students able to describe a greedy approach, and the majority of responses correctly identifying this approach as suboptimal. Lower scoring responses did not make clear what their greedy criterion was or chose a greedy criterion that was inappropriate to the problem (for example, chose the businesses with highest cost first).

Question 9a.

Marks	0	1	2	3	4	Average
%	34	27	16	16	6	1.3

An iterative improvement approach could be used as follows:

- Generate an initial candidate solution by randomly selecting a valid set of regions.
- Swap a randomly chosen region in the set with another available region to generate a new candidate solution.
- If the new candidate solution is invalid, score it 0.
- If valid, check the new total. If the new total is higher than the previous total, accept the new candidate solution. Otherwise, revert to previous candidate solution.
- Repeat until a certain number of iterations is reached and return the current candidate solution.

It was evident in responses that many students would benefit from deeper engagement with randomised heuristics algorithms. A wide array of low- to medium-scoring responses was seen, with common problems including:

- describing an algorithm that only generates one random candidate solution before halting
- describing an algorithm that has no random component whatsoever
- describing simulated annealing in general terms without engaging with the context, or describing it in context but incorrectly
- describing a greedy algorithm
- describing an algorithm to solve the graph colouring problem instead.

High-scoring responses demonstrated a capacity to apply the concept of simulated annealing or iterative improvement (or similar) to the given context. Students are encouraged to practise this application across different problems and contexts, remembering that most of the NP-complete problems they encounter can be approached with a randomised heuristic algorithm. When working through such an application, useful guiding questions might include:

- What is the initial candidate solution? How would this be generated?
- How would new candidate solutions be generated? How would they be scored?
- Under what condition would the new candidate solution be accepted?
- When would the algorithm terminate?

Question 9b.

Marks	0	1	Average
%	46	54	0.5

Backtracking and brute force were both accepted.

Question 9c.

Marks	0	1	2	3	Average
%	43	12	11	35	1.4

Responses that were able to interpret the given algorithm generally had success in deriving the time complexity. Some common errors included not explaining why the time complexity is $O(2^n)$ or failing to describe the implications for solving game boards with hundreds of regions. Students must ensure that they engage with each part of a question to achieve full marks.

The following is an example of a high-scoring response.

For every region, there exists two different potential solutions that can be generated. Thus the number of possible solutions grows as 2^n where n = the number of regions. Thus the worse case time complexity is $O(2^n)$. As this is intractable, the implication is that the brute force algorithm will not be feasible for game boards with hundreds of regions as will take an unreasonable of time to calculate the optimal solution this way.

Question 9d.

Marks	0	1	2	3	4	Average
%	38	14	19	18	11	1.5

There were many responses that demonstrated a cursory or memorised knowledge of minimax without the ability to apply it to a particular context. These responses did not score highly. Few responses dealt with the requirement to accommodate limiting the time the algorithm runs for.

Students are advised to begin any minimax questions by describing what the game tree would look like. Students that bypassed this step tended to have muddled answers that often misrepresented how minimax works. Common errors included the idea that scores would be added going up the tree and incorrect assumptions about what player 1 and player 2 would do in the game.

The following is an example of a high-scoring response.

Create a game tree, where the children of each node are the possible regions the players could claim. Beyond a certain depth, a heuristic would be used to estimate the value of each state, for example, assume the players choose regions of equal value from that point on, so Value (node) = ($\sum P1's$ regions) - ($\sum P2's$ regions). Then, apply the minimax algorithm as normal, assuming P1 picks moves that maximise the eventual heuristic value, while P2 attempts to minimise this. When the current state is reached, a positive value (node) means P1 will win, while value < 0 means P2 wins.

Question 10a.

Marks	0	1	2	Average
%	36	36	28	0.9

Any two of:

- Mathematics would be complete if all mathematical statements could be derived from a finite set of axioms.
- Mathematics would be consistent if no contradiction could exist within mathematics.
- Mathematics would be decidable if there was a single algorithm for deciding the truth of any mathematical statement.

Many responses demonstrated an informal understanding of two of the given goals; however, few responses used subject-specific terminology in a precise way. Students are encouraged to ensure they work towards a strong understanding of 'decidable' as the concept of decidability is crucial to much of Unit 4 Outcome 3.

Some common incorrect responses included suggesting that 'consistent' referred to something not changing and 'complete' referring to the idea that nothing was missing.

Question 10b.

Marks	0	1	2	Average
%	41	38	21	0.8

The Halting problem asks whether there exists an algorithm such that, given a program and an input, the algorithm can determine whether the program will halt on the input.

Many responses strayed far from the question and attempted to show why the Halting problem is undecidable. This was not required, nor was it sufficient to receive marks for this question.

Of the responses that did try to define the Halting problem itself, omitting any mention of input from the description was a common error, as was posing the problem as a statement. For instance, it was common to see 'The Halting problem states that no algorithm can exist …'.

Question 10c.

Marks	0	1	2	Average
%	26	17	58	1.3

In contrast to the previous part of the question, this question was answered well by most students. Some responses correctly described the challenge that the Halting problem posed to the decidability goal then went further and incorrectly claimed that the Halting problem also invalidated the completeness and/or the consistency goals.

The following is an example of a high-scoring response.

Turing showed that no solution of the Halting problem can exist – that the Halting Problem is undecidable. As such, this showed that "mathematics should be decidable" was not possible to be achieved in Hilbert's program.

Question 11

Marks	0	1	2	3	Average
%	46	13	33	9	1.0

Ming's problem is decidable as there is an algorithm that terminates on all inputs for it (the time complexity is irrelevant here). Nothing can be deduced about the decidability of Amira's problem as the algorithm she proposed does not terminate on all inputs (which means it cannot be used to show the problem is decidable), and to be undecidable requires showing no algorithm exists, which has not been done.

This question was not done well. Many students were able to identify that Amira's algorithm not terminating on 5 was problematic, but most incorrectly argued that this meant her algorithm was undecidable. It was also evident that students were often conflating problems with algorithms. Algorithms cannot be decidable nor undecidable.

In discussing Ming's problem, some students confused the concepts of decidability and tractability and incorrectly concluded that Ming's problem is undecidable because Ming's algorithm had a non-polynomial running time.

Question 12a.

Marks	0	1	2	Average
%	68	14	18	0.5

There is a widespread misconception among Algorithmics students that NP is short for Not Polynomial. Problems in NP are those whose solutions can be verified in polynomial time.

In order to score full marks, responses also needed to ensure they referenced Peter's specific running time in some way to show that they are engaging with the specific context rather than simply stating the definition of NP in a generic way.

The following is an example of a high-scoring response.

Peter's algorithm does not verify a solution to the three-colouring problem in polynomial time, as it has time complexity $O(2^n)$. This does not satisfy requirements for NP.

Question 12b.

Marks	0	1	2	Average
%	23	47	29	1.1

If Jane is correct, then all other NP-Complete problems could also be solved in polynomial time. This is because all NP-Complete problems (of which TSP is one) can be reduced to all other NP-Complete problems in polynomial time.

Most responses showed an understanding of the implications of solving any NP-Complete problem in polynomial time; however, some of these responses did not use subject-specific terminology and others did not provide a valid explanation.

Question 13

Marks	0	1	2	3	Average
%	43	45	10	2	0.7

- Assume that a shorter path to v exists via some other path.
- Such a path takes the form $s, \ldots, w, x, \ldots, v$, where w is in Q and x is not in Q.
- Since weights in the graph are not negative, the cost of the path $x, \ldots, v$ is greater than or equal to 0.
- When selecting a vertex to finalise, Dijkstra's algorithm selects the non-finalised vertex with the shortest known path. As such, the cost of the path $s, \ldots, w, x$ is greater than or equal to the cost of the path $s, \ldots, u, v$.
- Therefore, $Cost_{s,\ldots,w,x} + Cost_{x,\ldots,v} \geq Cost_{s,\ldots,u,v}$, contradicting the original assumption and therefore showing that such a shorter path cannot exist.

The overwhelming majority of students were unable to correctly respond to this question, although most students did make an attempt. Even where students were unsure how to proceed, they were able to use the structure of an argument by contradiction. Such responses often scored one mark.

Some responses had an assumption that could not possibly lead to a proof by contradiction, usually by assuming that a shorter edge or path from *u* to *v* exists and then trying to refute this assumption. To show

that no shorter path can exist, a generalised shorter path must be assumed that does not necessarily go through *u* (as in the sample solution above).

A few responses set up the correct assumption and proceeded with a clear argument but did not invoke the crucial condition that edge weights cannot be negative. Without this condition, it is not possible to prove that Dijkstra's algorithm is correct. It is easy to remember this given that Dijkstra's cannot be used reliably on graphs with negative edge weights for solving the single-source shortest path problem.