2019 VCE Algorithmics (HESS) examination report

General comments

The 2019 Algorithmics (HESS) examination contained two sections. Section A was composed of 20 multiple-choice questions and Section B was composed of 16 short-answer and extended-answer questions. Students achieved scores across the range of available marks, with higher-scoring responses demonstrating an impressive grasp of challenging content.

Students attempted most questions, demonstrating a thorough engagement with the study design. Most also paid careful attention to the requirements of each question and offered explanations when the question called for a justification as part of the answer.

In Section B, students showed a good understanding of theoretical computing concepts such as complexity classes, decidability and tractability (Questions 5, 6a., 7b., 8 and 9). The proportion of responses relying on memorisation rather than understanding was low, and the use of dot points for multi-part responses appeared to help students write clear and concise responses. However, many responses used the concept of the NP complexity class incorrectly, most often by suggesting that problems that are in NP are hard. This is not the case.

In Section B, several questions included diagrams to aid students in their responses. It is important that students distinguish between diagrams that are used to illustrate a context (as in Question 13), and those that are integral to the question and/or response (as in Questions 6, 7 and 15). The former will often be flagged in a question by terms like ‘for example’, and in such instances students must answer the question by considering the general problem rather than the specific example. For instance, in Question 13, considering a school with some unknown number of classrooms was appropriate, as the table was given as an example only.

All three questions asking for pseudocode in Section B (Questions 5, 7c. and 12c.) were complex in nature, and elicited a wide range of student responses, including some outstanding responses. Where pseudocode adhered too closely to the syntax of a programming language, it was primarily Python, which is generally accepted due to its easy readability. However, many students who seemingly had an understanding of what they wanted to write were often unable to employ the necessary looping structure. Students are encouraged to experiment with different looping structures throughout the year, and in particular to be familiar with each of the following three which often prove useful in solving Algorithmics (HESS) problems:

- for i = 1 to n
- for element in Abstract Data Type (e.g. for node in graph)
- while condition A is not met.

Specific information

Note: 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 – Multiple-choice questions

The table below indicates the percentage of students who chose each option. The correct answer is indicated by shading.

Question	% A	% B	% C	% D	% No answer	Comments
1	3	1	9	86	0	
2	56	20	18	4	2	The NP class, by definition, contains all problems for which solutions are quickly verifiable.
3	0	10	84	6	0	
4	13	72	3	11	1	
5	8	3	1	86	1	
6	12	36	25	28	0	Both mergesort and quicksort have the recurrence relation $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
7	64	9	14	12	2	
8	40	22	30	8	0	Whether quicksort will run in its worst case runtime depends on the choice of pivot. Since this is not known here, randomising the input will minimise the chance of the worst case occurring.
9	5	70	5	19	0	
10	8	3	58	31	0	
11	1	3	31	65	0	
12	19	11	61	9	0	
13	8	86	5	1	0	
14	28	54	14	3	2	Students are encouraged to approach this type of question by a process of elimination.
15	0	97	1	3	0	
16	23	1	72	4	0	Both options A and C were accepted.
17	10	25	61	4	0	

18	23	42	19	16	0	Big-Theta is a 'sandwich' bound, which implies both an upper and a lower bound.
19	5	5	19	70	0	
20	46	14	8	32	1	Dynamic programming solves sub-problems once and uses the solutions for larger subproblems.

Section B

Question 1

Marks	0	1	2	3	Average
%	7	25	48	19	**1.8**

High-level responses included two common elements: the idea that the tape is read and written to, and the idea that it is used for storage, input and output. Various examples were used by students to support their answer, with some students comparing the tape to the memory in a digital computer, while others referenced Turing's original conception of the tape as akin to unlimited working paper.

Lower-scoring responses gave the definition of a Turing Machine rather than engage with the question, and some conflated the tape with the table of instructions in a Turing Machine.

Question 2

Marks	0	1	2	Average
%	34	45	21	**0.9**

Any permutation of the array [0, 2, 4, 6, 8, 10, 12, 14] would lead to the best case running time, as the asymptotic running time of mergesort is $O(nlogn)$ irrespective of the structure of the input.

Some students focused on the idea that mergesort would perform roughly half as many comparisons in the merge step if the two sub-arrays were sorted. So, for instance, [0, 2, 4, 6, 8, 10, 12, 14] or [8, 10, 12, 14, 0, 2, 4, 6,] would both lead to the best running time when taking into account the constant term.

Both of the above approaches were accepted, with the former the more common. In general, considering the constant term in running time analysis is beyond the scope of the course, so students are encouraged to focus on asymptotic running time.

The most common misconception about this question was the idea that if an input is already sorted, the sorting algorithm would recognise this and avoid going through its usual procedure. There were many one-mark responses to this question because of incomplete or unclear explanations.

Question 3a.

Marks	0	1	2	3	Average
%	5	4	1	90	**2.8**

After 8 and 3 are pushed into the 'special location', we encounter a '+'. So, both numbers will be 'popped' from the special location to perform the addition. The answer is 11 = (8 + 3) and 11 is

pushed back into the special location. Then, both 18 and 2 are pushed into the special location. We then have a '÷' sign, so the result is 9, and 9 is pushed back into the special location. Finally, we have a '×' sign. So, 11 and 9 are popped from the special location. The final answer will be 99 (= 11 × 9).

Most students scored full marks for this question, either through a verbal explanation as above or through a diagrammatic approach.

Question 3bi.

Marks	0	1	2	Average
%	24	10	66	**1.4**

Most students were able to identify a stack as the appropriate ADT, due to its 'last in first out' structure.

Students who selected an inappropriate ADT for this question were still able to receive full marks for the subsequent question if their ADT specification was correct for their chosen ADT.

Question 3bii.

Marks	0	1	2	3	Average
%	17	20	29	33	**1.8**

This question was generally done to a high standard. The following was a typical 3-mark response, although other valid operations were included in some responses.

Name : stack

Import : element, boolean

Operations: newStack : → stack

isEmpty : stack → boolean

push : stack x element → stack

pop : stack → stack

peek : stack → element

Students needed to include a constructor (newStack), an isEmpty operation and the standard push, pop and peek operations on a stack. Other unambiguous names for the same operations (e.g. 'top' instead of 'peek') were also accepted. A few students mistakenly attempted to define a pop operation that both returned an element and altered the stack.

Question 4a.

Marks	0	1	2	Average
%	10	42	47	**1.4**

The worst case time complexity is $O(n^2)$. As the calls to sortAscending and median are made sequentially, the overall running time will be $O(n^2 + n^2 + 1 + 1) = O(n^2)$.

Most students recognised the worst case time complexity of isGreaterOrEQual, but some were unable to clearly explain their answer. In particular, many students did not include that the two sortAscending calls were made sequentially in their responses.

Question 4b.

Marks	0	1	2	Average
%	34	17	48	**1.2**

Most students correctly identified that a median is not a proxy for whether one array is greater than another. Some students provided a correct example but did not explain it. When asked to explain with an example, students are reminded that an explanation is required.

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

> *It is incorrect. For example if A was {1, 2, 50, 51, 52} with a median of 50 and B was {10, 20, 40, 60, 70} with a median of 40. Although A has a larger median, all other elements are smaller and thus the pseudocode is incorrect.*

Question 5

Marks	0	1	2	3	4	Average
%	24	18	27	20	11	**1.8**

The following two algorithms are required to demonstrate the undecidability of the Halting Problem.

```
Halt(P, i)
     If P(i) does halt Then
          return True
     Else
          return False

H*(Q)
     If Halt(Q, Q) Then
          Loop forever
     return True
```

By a diagonalization argument, when we run H*(H*) we encounter a contradiction:

- If H* halts, when it gets passed into H*, H* will run forever. This implies H* both halts and doesn't halt, a contradiction.
- If H* doesn't halt, when it gets passed into H*, H* will halt. This implies H* both doesn't halt and does halt, a contradiction.

Therefore, H* cannot exist, and thus the Halt algorithm, the solution to the Halting Problem, cannot exist either. As no solution to the Halting Problem can exist, it is undecidable.

Students struggled to answer this question in full, but most were able to engage with the concept and make a serious attempt. There were, however, a number of misconceptions evident in student responses. Most commonly, while students were able to express the idea that a contradiction exists somewhere, few could clearly explain the contradiction or what this contradiction implies for the decidability of the Halting Problem. Other errors included incorrect or incomplete inputs, swapping the return of True and False and passing Halt into H* instead of H* itself.

Some high-scoring responses provided an alternative approach to this question, defining and explaining the Halting Problem itself rather than demonstrating its undecidability.

Question 6a.

Marks	0	1	2	3	4	Average
%	14	41	23	21	2	**1.6**

The proposed network is small with 6 stations and fewer than 32 paths, therefore brute-force is feasible. If the tracks were to expand, brute-force may become infeasible as the problem scales poorly, with a brute-force solution having an asymptotic running time of $O(2^n)$ where *n* is the number of platforms. This is because for each platform the train can either be at the platform or not be at the platform.

This question required students to consider both the specific network given as a diagram in the description for Question 6, as well as the more general idea of an exponential problem. Many students who achieved partial marks considered only one of these aspects. Students should take care to note the language referring to any diagram, and in this case the words 'the proposed network is modelled below' suggest that this particular network should be engaged with in some way.

Some responses gave $O(n!)$ as the running time of the brute-force algorithm, and seemed to suggest that this running time is inevitable when a problem scales poorly. This is not the case. Students are encouraged to consider what structures give rise to $O(n!)$ running time (e.g. as in the Travelling Salesman Problem) and what structures give rise to $O(2^n)$ running time (e.g. as in the Knapsack Problem). Understanding this makes it simpler to derive the time complexity for an unfamiliar problem.

Question 6b.

Marks	0	1	2	3	Average
%	21	39	25	15	**1.4**

Dynamic programming is useful on problems that have optimal substructure and overlapping subproblems. This problem has optimal substructure because having the optimal schedule for the $n-1$ station subproblem makes it simple to solve the n station subproblem. The subproblems are overlapping since, for instance, the 6 station subproblem contains the 5 station subproblem.

Many students were able to recall the properties that problems require to be solvable by dynamic programming; however, students are encouraged to make the connection between their understanding of the concept and the specifics of the context more explicit.

Question 7a.

Marks	0	1	2	Average
%	21	51	28	**1.1**

The ideal sample can be coloured using exactly two colours. A sample with various impurities such as Figure 2 cannot be 2-coloured. Therefore, graph colouring can be used to distinguish between pure and impure samples.

Many students were able to show an understanding that as a substance becomes less pure, the number of colours required to colour it increases. However, some did not engage with the specific properties of the ideal structure as illustrated in Figure 1.

Question 7b.

Marks	0	1	2	3	Average
%	22	30	29	19	**1.5**

A variety of responses were accepted as students approached the question from a range of perspectives. High-scoring responses that answered in the affirmative justified their answer in one of two broad ways:

- Greedy graph colouring runs in pseudo-polynomial time, and so would be sufficiently fast to handle larger strongsheets. A greedy approach is sufficient as there is not a strict mapping between the number of colours and purity, so an optimal number of colours in not required.
- Greedy graph colouring runs in pseudo-polynomial time, and so would be sufficiently fast to handle larger strongsheets. A greedy approach is sufficient as testing for 2-colouring can be done correctly with a greedy approach.

High-scoring responses that answered in the negative tended to explain that as the size of the strongsheet increases, the search space for graph colouring will suffer a combinatorial explosion. As graph colouring is an NP-Complete problem, there is no known algorithm that will solve it in polynomial time.

Two misconceptions were evident in student responses. The most common was confusing the concept of verifying a particular colouring of a graph (which can be done in polynomial time) with the concept of checking whether a graph **can** be coloured with a certain number of colours (which is the decision version of the graph colouring problem, and is in NP-Complete). The broader error evident in many responses was conflating NP and NP-Complete, and arguing that if a problem is in NP it cannot be solved in polynomial time.

Question 7c.

Marks	0	1	2	3	4	5	Average
%	31	28	15	15	8	3	**1.5**

This question required students to demonstrate both a strong understanding of a studied algorithm and the ability to write pseudocode that would handle the details of a particular given context. Most students were able to construct a response that contained elements of what was required, but complete and correct responses were rare.

Of the high-scoring responses, some students calculated the number of colours required for a given graph, while others opted to check whether a graph was able to be coloured with exactly two colours (see Question 7b.). The solution below is an example of the latter.

```
Input: A graph, G, representing a given structure
Output: True if graph can be coloured with 2 colours (here denoted 1 and
2), False otherwise
Colour_two(G):
      Let Q be an empty queue
      Let A be a randomly selected node in G
      Colour A with 1
      Enqueue A to Q
      While Q is not empty:
            v = Q.front()
            Q.pop()
            for neighbour of v:
                  if neighbour is uncoloured:
```

```
                    colour neighbour with opposite colour to v
                    Enqueue neighbour to Q
               else if neighbour has same colour as v:
                    return False
    return True
```

There were a number of errors in the student responses. The most prevalent was the omission of some sort of looping mechanism that would iterate over all the nodes in the graph as well as the omission of a return statement or output. Another common error was an attempt to describe or use dot points for the required algorithm rather than writing it in pseudocode. Students are reminded that when instructed to **write** an algorithm, rather than to **describe** or **outline** one, the answer should be in pseudocode.

Question 8

Marks	0	1	2	3	4	Average
%	14	25	30	23	8	**1.9**

Most students attempted to give a definition of both theses, although the precision in responses varied. When a concept is explicitly included in the study design, it is important for students to have engaged with this concept in some detail. For example, the statement that 'the Church-Turing thesis outlines what Turing Machines can do' is not incorrect, but it doesn't represent a definition within the scope of this study. Fewer students tried to explicitly explain the relationship between the two theses.

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

- *The Church-Turing Thesis states that any function is effectively calculable by some method if it is computable on a Turing Machine. This thesis defines the hard limits of computation*
- *Cobham's Thesis states that a problem is feasibly computable if it can be computed in polynomial time, that is it lies in the P complexity class. Cobham's thesis defines the soft limits of computation.*
- *Both these theses attempt to describe what is computable, with Church-Turing describing what's theoretically computable and Cobham's describing what's practically computable.*

Question 9

Marks	0	1	2	3	4	Average
%	13	30	31	22	4	**1.8**

In DNA computing, components of a particular problem are encoded using synthesised DNA sequences. Trillions of DNA sequences are combined in the lab and, through biological processes, longer sequences encoding all possible candidate solutions to the problem are created. Then the best candidate solutions can be filtered out using certain lab processes. As DNA computing represents massively parallel processing as opposed to the sequential processing of digital computers, it can (theoretically) be used to overcome the soft limits of computation.

No specific terminology pertaining to the biological or lab processes was required. A common mistake was to focus on only one aspect of the question in detail, rather than addressing both how DNA computing works and how it can overcome the limits of computation.

Question 10

Marks	0	1	2	3	4	5	Average
%	17	28	18	17	8	12	**2.1**

Most students were able to attempt an argument by induction, often demonstrating that they understood the elements that are required even if some responses lacked rigour. In particular, the establishment of the base case and a clear statement of the inductive hypothesis were done well in a majority of responses.

The most common error was the failure to reference the inductive hypothesis anywhere after the original assumption. Students are reminded that there is no value in making an assumption if that assumption is not used anywhere in the argument.

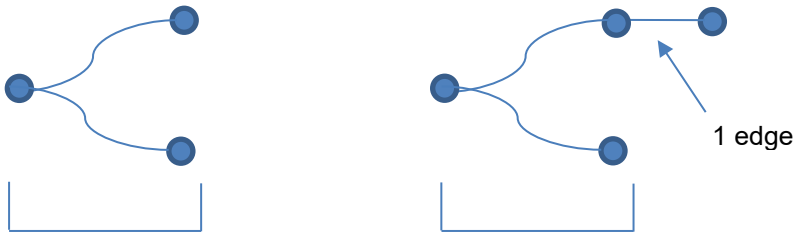
The following is an example of a high-level response:

Let $|K|$ denote the no. of vertices in tree T and $|E|$ denote the no. of edges

BASE CASE: let $n = 1$, $|E| = 0 = (n - 1)$ (TRIVIAL) $\therefore n = 1$ *is TRUE*

INDUCTIVE HYPOTHESIS: assume n=k is true

adding one more node to T gives n = k + 1



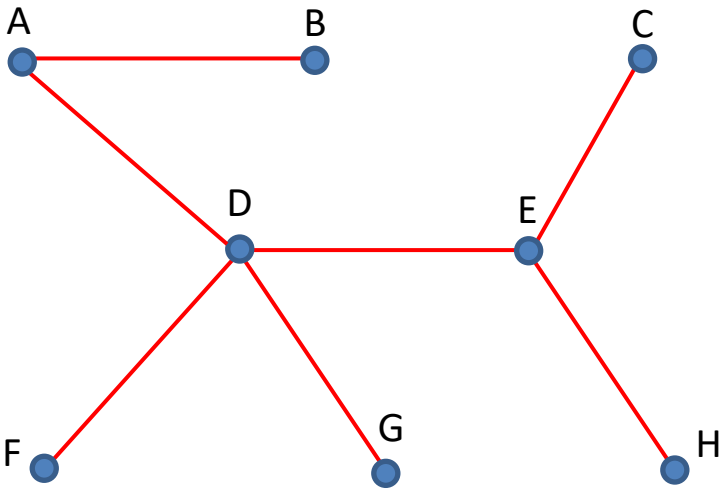
$$\therefore |E| = (k - 1) + 1$$
$$= k = (k + 1) - 1$$
$$\therefore n = k + 1 \; is \; TRUE$$

Question 11

Marks	0	1	2	Average
%	19	13	69	**1.5**

The majority of the responses gave both the correct tree and the correct series of edges. Of the responses that were awarded 1 mark, the most common error was omitting the order of edges or recording them incorrectly.

Order of edges: AB, AD, DG, DE, EH, EC, DF



Question 12a.

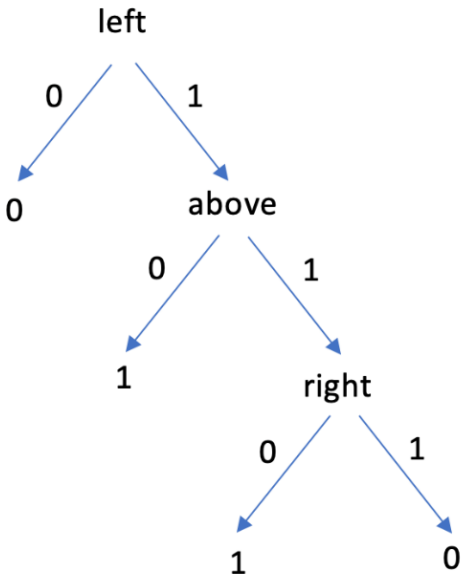
Marks	0	1	Average
%	40	60	**0.6**

The next row is 0001101011. The most common error was incorrectly handling the edge values.

Question 12b

Marks	0	1	2	3	Average
%	13	17	19	51	**2.1**

The decision tree below captures the decisions required to implement the cellular automata.



Complete decision trees (with eight leaves) were also accepted.

Students who understood the concept of using a decision tree to represent decisions being made within an algorithm completed the question correctly; however, a significant number of students needed to develop a greater understanding of the connection between decision trees and conditional statements in algorithms.

Question 12c.

Marks	0	1	2	3	4	Average
%	37	11	19	23	11	**1.6**

Most students engaged with the question and demonstrated an understanding of how a worded procedure – as described in the question – could be written in pseudocode. Some mid-range responses tended to use only a single loop through the array, thereby completing only one iteration of the cellular automata. Other mid-range responses did not handle assignment of new values well, often overwriting the array too early or not changing the array at all. Additionally, some responses called a function that would correctly generate the next array without explicitly writing this function anywhere.

More broadly, there was little evidence that students proofread or tested their pseudocode. Students are highly encouraged, where time permits, to execute their code line-by-line on some example inputs to check for correctness.

High-scoring responses tended to take one of three approaches: returning the n^{th} row (with n as input), returning the first n rows (with n as input) or returning the row number corresponding to when a row entirely composed of 0s was first reached. All three were deemed reasonable readings of the question and awarded full marks if correct.

The following is a solution for the third interpretation, returning the row number corresponding to when a row entirely composed of 0s was first reached.

```
Cellular_automata(an array A):
     prev = [0] + input + [0]
     n = 1
     while not prev = [0,0,0,0,0,0,0,0,0,0]:
          row = [0,0,0,0,0,0,0,0,0,0]
          for j = 1 to 8:
               if prev[j-1] = 1
                    if min(prev[j],prev[j+1]) = 0
                         row[j] = 1
          prev = row
          n = n + 1
     return n
```

Question 13

Marks	0	1	2	3	4	Average
%	21	42	15	15	7	**1.5**

Donna's problem is akin to the Travelling Salesman Problem. Assuming the total number of forms is not too heavy to carry in one folder or bag, Donna could model her school using a complete, weighted graph, with each node representing a classroom and the weight on the edge between any two classrooms representing the time it takes to walk between those two classrooms. Donna could then use a heuristic such as simulated annealing to find a good solution to the TSP on her graph, with her classroom as the source, and deliver the forms according to the solution found.

This question was not answered well, as many students focused too heavily on the example table given and were therefore unable to relate the problem back to studied algorithmic concepts. Where a diagram, table or illustration is prefaced with 'For example', students must consider the general problem rather than the specific instance.

Some students also tried to answer the question in a novel way that did not relate directly to the course, for instance by requiring that Donna asks students in her class to deliver the forms for her. This style of response never scores well, and students are encouraged to always look for ways to solve problems by applying studied concepts or modifications of them.

Question 14a.

Marks	0	1	2	Average
%	8	24	68	**1.6**

The answer is $O(n^3)$, due to three nested **for** loops that each iterate through n integers.

Most students correctly identified the time complexity. Most 1-mark responses provided an incomplete justification, often omitting the idea that each **for** loop iterates through n integers.

Question 14b.

Marks	0	1	2	3	Average
%	59	12	22	7	**0.8**

This question was not answered well. Some students did not engage with the question at all, while others discussed what the given time recurrence might mean in general, rather than relating it to Betty's algorithm. Such responses were awarded no marks.

For full marks, responses were required to discuss the following two elements:

- each multiplication recursively calls 8 new multiplications of subarrays of half the size, thus $8T\left(\frac{n}{2}\right)$
- addition of resultant (n/2) x (n/2) subarrays is of $O\left(\left(\frac{n}{2}\right)^2\right) \in O(n^2)$

The majority of students who scored 1 or 2 marks were able to articulate the first point, but not the second.

Question 14c.

Marks	0	1	2	Average
%	11	10	79	**1.7**

Using the Master Theorem $T(n) = O(n^3)$, because $\log_b a = \log_2 8 = 3 > 2 = c$

Most students applied the Master Theorem correctly.

Question 14d.

Marks	0	1	2	Average
%	26	15	59	**1.3**

Using the Master Theorem $T(n) = O(n^{\log_2 7})$, because $\log_b a = \log_2 7 = 3 > 2 = c$. Therefore, the new method is faster as $\log_2 7 < 3$.

Question 15

Marks	0	1	2	Average
%	42	29	29	**0.9**

Many students were able to write down a version of the Page Rank formula, but the majority did not correctly explain the function of the damping factor in the formula. In this subject, when learning an algorithm or a formula, understanding the derivation is crucial.

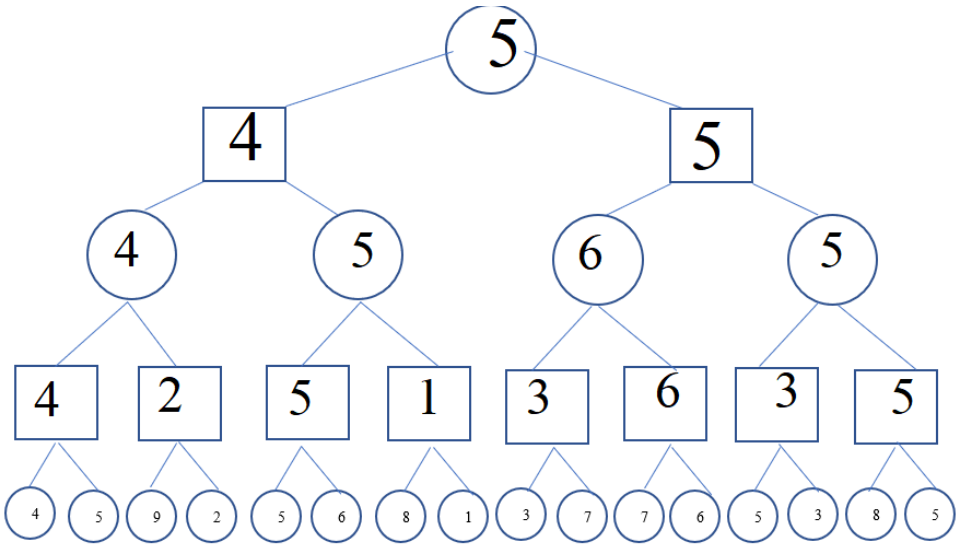
The following is an example of a high-scoring response:

PR(D) = $\frac{1-d}{4}$ *+ probability from incoming links*

The Page Rank is dependent not only on incoming links, but also the chance that it is randomly stumbled upon, which is $\frac{1-d}{4}$ *where d is normally 0.85. So, despite having no incoming links, it has non-zero probability.*

Question 16

Marks	0	1	2	3	4	Average
%	23	19	31	18	9	**1.7**



The minimax algorithm can be applied to a two-player, turn-based game on the assumption that both players play optimally. That is, Stella will want to maximise her score at each turn, and Cameron will want to minimise her score at each turn. Running the minimax algorithm will allow Stella to generate a game tree like the one above, and then traversing the tree starting at the root node would allow Stella to find the sequence of moves that will maximise her chances of winning.

It was evident that some students had studied the minimax algorithm but had not applied it to an actual scenario using a game tree, while other students were able to complete the game tree but could not provide an explanation.