2.2 Queues Mark Scheme

Q1.

(a) **All marks AO2 (analyse)**

	1	2	3	4	5	6
1	0	2	5	3	0	8
2	2	0	1	0	0	0
3	5	1	0	0	0	4
4	3	0	0	0	1	0
5	0	0	0	1	0	5
6	8	0	4	0	5	0

Alternative answer

	1	2	3	4	5	6
1	0	2	5	3	0	8
2		0	1	0	0	0
3			0	0	0	4
4				0	1	0
5					0	5
6						0

Alternative answer

	1	2	3	4	5	6
1	0					
2	2	0				
3	5	1	0			
4	3	0	0	0		
5	0	0	0	1	0	
6	8	0	4	0	5	0

Mark as follows:

1 mark 0s in correct places

1 mark all other values correct

I. non-zero symbols used to denote no edge but only for showing no edge going from a node to itself

2

(b) **All marks for AO1 (understanding)**

Adjacency list appropriate when there are few edges between vertices // when graph/matrix is sparse; **NE**. few edges

Adjacency list appropriate when edges rarely changed;

Adjacency list appropriate when presence/absence of specific edges does not need to be tested (frequently);

A. Alternative words which describe edge, eg connection, line, arc

Max 2

2

(c) **Mark is for AO2 (apply)**

It contains a cycle / cycles;

1

(d) **Mark for AO1 (knowledge)**

A graph where each edge has a weight/value associated with it;

1

(e) **All marks AO2 (apply)**

Mark as follows:

I. output column
1 mark first value of `A` is 2
1 mark second value of `A` is 5 and third value is 3
1 mark fourth and subsequent values of `A` are 8, 3, 7, 4, 9 with no more values after this
1 mark `D[2]` is set to 2 and then does not change

				D						P					
U	Q	V	A	1	2	3	4	5	6	1	2	3	4	5	6
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

1 mark `D[3]` is set to 5 and then changes to 3 and does not change again

1 mark correct final values for each position of array `P`

				D						P					
U	Q	V	A	1	2	3	4	5	6	1	2	3	4	5	6
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

1 mark correct final values for `D[1]`, `D[4]`, `D[5]`, `D[6]`

				D						P					
U	Q	V	A	1	2	3	4	5	6	1	2	3	4	5	6
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

Max 6 marks if any errors

7

(f) **Mark is for AO2 (analyse)**

The shortest distance / time between locations/nodes 1 and 6;

NE distance / time between locations/nodes 1 and 6

R. shortest route / path

1

(g) **All marks AO2 (analyse)**

Used to store the previous node/location in the path (to this node);

Allows the path (from node/location 1 to any other node/location) to be recreated // stores the path (from node/location 1 to any other node/location);

Max 1 if not clear that the values represent the shortest path

Alternative answer

Used to store the nodes that should be traversed;

And the order that they should be traversed;

Max 1 if not clear that the values represent the shortest path

2

[16]

Q2.

(a) **Mark is for AO2 (analyse)**

`Len` (Python/VB only);
`Length` (Pascal/Java only);
`IndexOf` (C#/VB only);

I. case
I. spacing
R. if any additional code

1

(b) **Mark is for AO2 (analyse)**

```
Item // RandNo // Count;
```

`Rnd;` (Java only)

A. `MaxSize`

I. case
I. spacing
R. if any additional code
R. if spelt incorrectly

1

(c) **All marks AO1 (understanding)**

Mark as follows

- Check for 1st mark point from either solution 1 or solution 2.
- 2nd mark point for Solution 1 only to be awarded if 1st mark point for Solution 1 has been awarded.
- 2nd mark point for Solution 2 only to be awarded if 1st mark point for Solution 2 has been awarded

Solution 1
1st mark:
With a linear queue there could be locations available that are not able to be used **A.** there could be wasted space
(where there is space available in the data structure but it is unusable as it is in front of the data items in the queue);
2nd mark:
(To avoid this issue) items in the queue are all shuffled forward when an item is deleted from (the front of the) queue;
//
Circular lists “wrap round” so (avoid this problem as) the front of the queue does not have to be in the first position in the data structure;

Solution 2
1st mark:
Items in a linear queue are all shuffled forward when an item is deleted from (the front);
//
No need to shuffle items forward after deleting an item in a circular queue;
2nd mark:
this makes (deleting from) (large) linear lists time inefficient;
//
meaning circular queues are more time efficient (when deleting);

2

(d) **Mark is for AO2 (analyse)**

The queue is small in size (so the time inefficiency is not significant);

1

(e) **Mark is for AO1 (understanding)**

Front // pointer to the front of the queue;

1

(f) **All marks for AO2 (analyse)**

Change the `Add` method;

Generate a random number between 1 and 2; **NE**. so there is a 50% chance
Note for examiners: needs to be clear how a 50% chance is created

If it is a 1 then generate a random number from 0, 4, 8, 13, 14, 17, 18, 19 // if it is a 1 then generate a random number from those equivalent to 1-point tiles;

Otherwise generate a random number from the other numbers between 0 and 25 // otherwise generate a random number from those equivalent to non 1-point tiles;

A. equivalent methods to the one described
Note for examiners: refer unusual answers that would work to team leader

4

(g) **All marks for AO2 (analyse)**

Iterate over the characters in the string;

Get the character code for the current character;

Subtract 32 from the character code // AND the character code with the bit pattern 1011111 / 11011111 // AND the character code with (the decimal value) 95 / 223;
A. Hexadecimal equivalents

Convert that value back into a character and replace the current character with the new character;

A. answers that create a new string instead of replace characters in the existing string

Alternative answer

Iterate over the characters in the string;

Using a list of the lowercase letters and a list of the uppercase letters;

Find the index of the lowercase letter in the list of lowercase letters;

Get the character in the corresponding position in the uppercase list and replace the current character with the new character;

A. answers that create a new string instead of replace characters in the existing string

4

[14]

Q3.

(a) Values/cards need to be taken out of the data structure from the opposite end that they are put in // cards removed from top/front and added at end/bottom/rear;
Values/cards need to be removed in the same order that they are added;
A. It is First In First Out // It is FIFO;
A. It is Last In Last Out // It is LILO;

Max 1

(b) (i) `FrontPointer` = 13
`RearPointer` = 52
`QueueSize` = 40
1 mark for all three values correct

1

(ii) `FrontPointer` = 13
`RearPointer` = 3
`QueueSize` = 43

1 mark for all three values correct
A. Incorrect value for `FrontPointer` if it matches the value given in part (i) and incorrect value for `QueueSize` if it is equal to the value given for `QueueSize` in part (i) incremented by three (follow through of errors previously made). However, `RearPointer` must be 3.

1

(c)
```
If DeckQueue is empty THEN
  Report error
ELSE
  Output DeckQueue[FrontPointer]
  Decrement QueueSize
  Increment FrontPointer
  IF FrontPointer > 52 THEN
    FrontPointer ← 1
  ENDIF
ENDIF
```

1 mark for `IF` statement to check if queue is empty – alternative for test is `QueueSize = 0`.
1 mark for reporting an error message if the queue is empty // dealing with the error in another sensible way – this mark can still be awarded if there is an error in the logic of the `IF` statement, as long as there is an `IF` statement with a clear purpose.

1 mark for only completing the rest of the algorithm if the queue is not empty – this mark can still be awarded if there is an error in the logic of the `IF` statement, as long as there is an `IF` statement with a clear purpose.
1 mark for outputting the card at the correct position
1 mark for incrementing `FrontPointer` and decrementing `QueueSize`
1 mark for `IF` statement testing if the end of the queue has been reached
1 mark for setting `FrontPointer` back to 1 if this is the case – this mark can still be awarded if minor error in logic of `IF` statement, eg >= instead of =
A. `FrontPointer = (FrontPointer MOD 52) + 1` for **3 marks** or `FrontPointer = (FrontPointer MOD 52)` for **2 marks**, both as alternatives to incrementing and using and the second `IF` statement – **deduct 1 mark** from either of the above if `QueueSize` has not been decremented
A. Any type of brackets for array indexing
I. Additional reasonable `ENDIF` Statements
MAX 5 unless all of the steps listed above are carried out and algorithm fully working

6

[9]

Q1.

In previous years there have been questions asking students to complete an adjacency matrix based on a diagram of a graph and most students were able to answer question (a) this year. This was the first time that an adjacency matrix for a weighted graph had been asked for and some students had clearly not seen this type of question before and only included an indicator that there was an edge between two nodes rather than the weight of the edge between the two nodes; this meant they only got one of the two marks available for this question.

Questions (b)-(d) were about graph theory. Question (c) was well-answered with students identifying that it was not a tree because there were cycles. The most common incorrect answer was to say that it wasn't a tree because the edges have weights associated with them. Question (d) was also well-answered. Answers to (b) often showed that students were not as familiar with adjacency lists as they are with adjacency matrices.

For question (e) students had to complete a trace of Djikstra's Algorithm. This topic was not on the previous A-level specification and was often poorly answered suggesting many students had not tried to complete a trace for the algorithm before. For question (f) many students gave an answer that explained the point of Djikstra's Algorithm (find the shortest route from a node to each other node) rather than what the specific output in the algorithm given in the question would be (the distance of the shortest route from node 1 to node 6).

Q2.

An error on the paper meant that it was not possible for students using the Java programming language to provide an answer for question (a). All students (for all languages) were awarded this mark irrespective of what they wrote for their answer.

Question (b) asked students to identify a local variable in a method in the `QueueOfTiles` class. There were a number of potential correct answers with `Item` being the most commonly seen. Some students gave an example of a private attribute belonging to the class rather than a local variable in a method in the class.

Questions (c)-(e) were about circular and linear queues. Some students stated that a rear pointer would be needed for a circular queue which is true but does not answer question (e) as the rear pointer was already present in the Skeleton Program. Some answers for (c) talked, incorrectly, about circular queues being a dynamic data structure and linear queues as being a static data structure. Good answers for (d) made it clear that with the queue only being very small in size the overhead of moving all items in the queue up one after deleting an item was negligible.

Most answers for question (f) and (g) showed some understanding of suitable approaches that could be taken but were rarely precise enough for full marks to be awarded. Some students gave answers for question (f) that changed the values of some of the tiles despite the question stating that this should not be done.