GCE

Computer Science

H446/02: Algorithms and programming

Advanced GCE

Mark Scheme for November 2020

OCR (Oxford Cambridge and RSA) is a leading UK awarding body, providing a wide range of qualifications to meet the needs of candidates of all ages and abilities. OCR qualifications include AS/A Levels, Diplomas, GCSEs, Cambridge Nationals, Cambridge Technicals, Functional Skills, Key Skills, Entry Level qualifications, NVQs and vocational qualifications in areas such as IT, business, languages, teaching/training, administration and secretarial skills.

It is also responsible for developing new specifications to meet national requirements and the needs of students and teachers. OCR is a not-for-profit organisation; any surplus made is invested back into the establishment to help towards the development of qualifications and support, which keep pace with the changing needs of today's society.

This mark scheme is published as an aid to teachers and students, to indicate the requirements of the examination. It shows the basis on which marks were awarded by examiners. It does not indicate the details of the discussions which took place at an examiners' meeting before marking commenced.

All examiners are instructed that alternative correct answers and unexpected approaches in candidates' scripts must be given marks that fairly reflect the relevant knowledge and skills demonstrated.

Mark schemes should be read in conjunction with the published question papers and the report on the examination.

Annotations

Annotation	Meaning
	Omission mark
	Benefit of the doubt
	Subordinate clause / consequential error
	Incorrect point
	Expansion of a point
	Follow through
	Not answered question
	No benefit of doubt given
	Point being made
	Repeat
	Correct point
	Too vague
	Zero (big)
	Blank Page – this annotation must be used on all blank pages within an answer booklet (structured or unstructured) and on each page of an additional object where there is no candidate response.

L1	Level 1
L2	Level 2
L3	Level 3

Question	Answer	Marks	Guidance
1a	1 mark for definition • Removal of unnecessary detail // Simplification to allow development of a program more easily 1 mark to max 2 for application e.g. • The actual movements are represented by vertices/lines • State of the move is represented by a letter/symbol rather than the actual move position • Tree does not show details about what the moves are	**3** AO1.1 (1) AO2.1 (1) AO2.2 (1)	Allow other suitable examples that are relevant to the scenario in the question.
1b	One node (node A) has more than 2 connections Nodes aren't ordered (e.g. F is C's left child)	**1** AO2.1 (1)	
1c	1 mark for identification • Null pointers	**1** AO2.1 (1)	
1d	1 mark per bullet • Take **A** as starting node • Visit **B, C** and **E** • Visit **D, F, G** and **H** • Visit **I** and **J**	**4** AO1.2 (2) AO2.2 (2)	Allow the reverse ordering from right to left e.g. A; E, C, B; H, G, F, D; J, I
1ei	1 mark per bullet to max 3 • Search the tree to find the location of Node E // by example of search • Replace the content of node E with blank/null/equivalent • Make node A point to the node H • Add node E to the empty node list	**3** AO1.2 (3)	
1eii	1 mark per bullet to max 3 • Search the tree to find the location of node G // by example of search • Create a new node with value K • Add a pointer from node G to the new node • Make node K point to null/equivalent	**3** AO1.2 (3)	

1f	1 mark per similarity to max 2 • Both consists of nodes • Both are connected by edges/links • Both are non-linear data structures • Both are dynamic data structures 1 mark per difference to max 2 • Tree is 1-directional whereas a graph is 2-directional • Tree has a root node whereas a graph does not have a (clear) root node • Tree will not have cycles whereas graphs can contain cycles • Tree will not be weighted whereas edges in a graph can be weighted	**4** AO1.1 (4)	
2a	1 mark per bullet to max 4 e.g. • Decomposition splits the problem into smaller sub problems • Repeated decomposition gives solvable parts • The division can lead to the development of subroutines/modules • The division can lead to a logical division between programmers/teams • ...e.g. one team works on one section and another concurrently on another	**4** AO1.1 (2) AO1.2 (2)	

2b	**Mark Band 3 – High level (7-9 marks)** The candidate demonstrates a thorough knowledge and understanding of concurrent processing; the material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. *There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.* **Mark Band 2 – Mid level (4-6 marks)** The candidate demonstrates reasonable knowledge and understanding of concurrent processing; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate provides a reasonable discussion, the majority of which is focused. Evaluative comments are, for the most part appropriate, although one or two opportunities for development are missed. *There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence.* **Mark Band 1 – Low Level (1-3 marks)** The candidate demonstrates a basic knowledge of concurrent processing with limited understanding shown; the material is basic and contains some inaccuracies. The candidates makes a limited attempt to apply acquired knowledge and understanding to the context provided. The candidate provides a limited discussion which is narrow in focus. Judgements if made are weak and unsubstantiated. *The information is basic and comunicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be clear.* **0 marks** No attempt to answer the question or response is not worthy of credit.	**9** AO1.1 (2) AO1.2 (2) AO2.1 (2) AO3.3 (3)	**AO1: Knowledge and Understanding** **Indicative content** • Processes are happening at the same time/at overlapping times • One process may need to start before a second has finished • Individual processes are threads, each thread has a life line • One request will be sent to the server, this will have a thread **AO2: Application** • Multiple requests to the server can be made at the same time • Programming on server will need to allow multiple threads to manipulate a list of requests • Programming will need to restrict access to the database of seats/sales etc. • Will allow those reading and writing to manipulate at the same time • Record locking will need implementing – more complex programming • May be selling alongside other systems, therefore needs to communicate with external systems that will also use record locking to avoid two different external companies accessing and selling the same tickets. **AO3: Evaluation** • Will allow for multiple access to the website at the same time by different customers – as it would happen in real life

<table>
<tr><td></td><td></td><td></td><td>• Will allow for multiple ticket sales for the same event without selling the same seat twice</td></tr>
<tr><td>3a</td><td>1 mark per bullet
• Calculation of result to 3
• Call with <code>thisFunction(theArray, num1=4, num2=7, num3=35)</code>
• Result = 5
• call with <code>thisFunction(theArray,num1=6,num2=7,num3=35)</code>
• (Result = 6) return of value 6

<table><tr><th>Function call</th><th>num1</th><th>num2</th><th>num3</th><th>result</th></tr><tr><td>thisFunction(theArray,0,7,35)</td><td>0</td><td>7</td><td>35</td><td>3</td></tr><tr><td>thisFunction(theArray,4,7,35)</td><td>4</td><td>7</td><td>35</td><td>5</td></tr><tr><td>thisFunction(thisArray,6,7,35)</td><td>6</td><td>7</td><td>35</td><td>6</td></tr><tr><td></td><td></td><td></td><td></td><td></td></tr></table></td><td>5
AO2.1
(3)
AO2.2
(2)</td><td></td></tr>
<tr><td>3b</td><td>Binary search</td><td>1
AO2.1
(1)</td><td></td></tr>
<tr><td>3c</td><td>1 mark per bullet to max 4, e.g.

• Recursion uses more memory…
• …iteration uses less memory
• Recursion declares new variables //variables are put onto the stack each time…
• ...iteration reuses the same variables
• Recursive can run out of memory/stack space…
• …while iteration cannot run out of memory
• Recursion can express a problem more elegantly // in fewer lines of code…
• …while iteration can take more lines of code // be harder to understand
• Recursion will be self-referential // will call itself…
• … whereas iteration does not</td><td>4
AO1.1
(2)
AO1.2
(2)</td><td></td></tr>
</table>

3d	1 mark per bullet to max 6
	• Retains function call
	• Uses a loop
	• …that will loop until all elements inspected or value found
	• Updates num1 appropriately
	• Updates num2 appropriately
	• Returns -1 in the correct place if the value has not been found
	• Returns the result in the correct place if the value has been found
	e.g.
	`function thisFunction(theArray, num1, num2, num3)`
	
	`  while (true)`
	`  result = num1 + ((num2 - num1) DIV 2)`
	`  if num2 < num1 then`
	`  return -1`
	`  else`
	`  if theArray[result] < num3 then`
	`  num1 = result + 1`
	`  elseif theArray[result] > num3 then`
	`  num2 = result - 1`
	`  else`
	`  return result`
	`  endif`
	`  endif`
	`  endwhile`
	`endfunction`
		6
		AO2.2
		(3)
		AO3.1
		(3)	

4a	1 mark per bullet • By reference will change the actual contents of the array in the main program// when control returns to the main program the array will be sorted • By value would create a copy and not change the original // when control returns to the main program the array will **not** be sorted • By value the array is local to the function. • By reference will use less memory	**2** AO1.2 (1) AO2.2 (1)	
4b	1 mark pet bullet to max 3 • Descending order • Line 07 (`dataArray[tempos]<temp`) has the comparison… • …that checks if current position is less than item to insert and… • …breaks out of loop when current position is less than or equal to item to insert	**3** AO1.2 (1) AO2.2 (2)	

4c	**Mark Band 3 – High level (7-9 marks)** The candidate demonstrates a thorough knowledge and understanding of big O and sorting algorithms; the material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation. The candidate is able to weigh up the use of the sorting algorithms which results in a supported and realistic judgment as to whether it is possible to use them in this context. *There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.* **Mark Band 2 – Mid level (4-6 marks)** The candidate demonstrates reasonable knoledge and understanding of big O and sorting algorithms; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate makes a reasonable attempt to come to a conclusion showing some recognition of influencing factors that would determine whether it is possible to use the sorting algorithms in this context. *There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence* **Mark Band 1 – Low Level (1-3 marks)** The candidate demonstrates a basic knowledge of big O and sorting algorithms with limited understanding shown; the material is basic and contains some inaccuracies. The candidates makes a limited attempt to apply acquired knowledge and understanding to the context provided. The candidate provides nothing more than an unsupported assertion. *The information is basic and comunicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be clear.* **0 marks**	**9** AO1. 1 (2) AO1. 2 (2) AO2. 1 (2) AO3. 3 (3)	**AO1: Knowledge and Understanding** **Indicative content** O(1) • Constant space, does not change O(n) • Linear • Same as number of elements • As number of elements increases so does the time/space O(n^2) • polynominal • As number of elements increases, time/space increases by *n O(n log(n)) • Linearithmic **AO2: Application** • Space: Merge sort will require more memory usage as the number of elements increases. Insertion will not require any more space than original. Quick will increase but not as much as merge. • Best time: Insertion increases at the same rate as the number of elements. Quick and merge increase at greater rate • Worst time: insertion and quick increase significantly by n for each additional item. Merge sort increases less per element. • Log more appropriate for large number of elements **AO3: Evaluation** e.g. • Small array – space is not important. Few number of elements. Look for consistency.

	No attempt to answer the question or response is not worthy of credit.		• Large array therefore memory important – could remove merge as inappropriate. Logarithmic more efficient.

4d	1 mark per bullet for description to max 6 • Compare each pair of adjacent elements • If they are not in the correct order then swap the elements • If they are in the correct order then do no swap elements • When you read the end of the array return to the start • Repeat n elements time • Set a flag to be false whenever a swap is made • …repeat the loop if the flag is not false	**6** AO1.1 (2) AO1.2 (4)	
5a	1 mark per pointer • queueHead: Point to the first element in the queue // next element to remove • queueTail**:** Point to the last element in the queue	**2** AO1.2 (2)	

5b	1 mark per bullet up to max 5 • first 3 jobs removed • 128 and 129 added in positions 4 and 5 respectively • no additional jobs • queueHead being 3 (FT errors) • queueTail being 5 (FT errors) `queueHead` 3 `queueTail` 5 6 | 5 `job-129` 4 `job-128` 3 `job-127` 2 | 1 | 0 |	**5** AO2.1 (2) AO2.2 (3)	The underlying implementation of the queue has not been specified, so allow alternative valid answers. e.g. queueHead = 0 queueTail = 2 Location 2: 129 Location 1: 128 Location 0: 127

5ci	1 mark per bullet to max 5
	• Function declaration
	• Checking if queue is empty
	• …returning `null`
	• (Otherwise) incrementing `queueHead`
	• …returning `buffer[queueHead-1]`
	e.g.
	`function dequeue()`
	`  if queueHead > queueTail then`
	`  return null`
	`  else`
	`  queueHead = queueHead + 1`
	`  return buffer[queueHead-1]`
	`  endif`
	`endfunction`
		5
		AO2.2
		(2)
		AO3.3
		(3)
			Note: Accept alternative valid underlying implementation answers e.g. Shifting all elements in queue forward.

5cii	1 mark per bullet to max 6
	• Function declaration taking parameter
	• Checking if queue is full
	• …returning -1
	• (Otherwise) incrementing queueTail
	• Adding newJob to buffer(queueTail)
	• Returning 1
	
	e.g.
	`function enqueue(newJob)`
	`  if queueTail == 99 then`
	`  return -1`
	`  else`
	`  queueTail = queueTail + 1`
	`  buffer[queueTail] = newJob`
	`  return 1`
	`  endif`
	`endfunction`
		6
		AO2.2
		(3)
		AO3.3
		(3)	

<table>
<tr>
<td>5ciii</td>
<td>1 mark per bullet to max 8
• Inputting user choice
• If enqueue chosen input job name
• …call enqueue with input value as parameter
• …check if return value is -1 and output full
• …otherwise output message that item is added
• If dequeue chosen
• …call dequeue and save returned value
• …output returned value (jobname) if not null
• …or output queue is empty
e.g.
<pre>
main()
  choice = input("Add or remove?")
  if choice == "ADD" then
    jobname = input("Enter job name")
    returnValue = enqueue(jobname)
    if returnValue == -1 then
       print("Queue full")
    else
       print("Job added")
    endif
  else
    returnValue = dequeue()
    if returnValue == null then
      print("Queue empty")
    else
      output returnValue
    endif
  endif
endmain
</pre></td>
<td>8
AO2.2 (2)
AO3.3 (6)</td>
<td>Allow equivalent checks / logic</td>
</tr>
<tr>
<td>5d</td>
<td>1 mark per bullet to 3
• Check if either head or tail are incremented to above 99
• … set to be 0 instead
• When checking if array is full check if (queueTail == queueHead – 1) OR (queueTail==99 AND queueHead==0)</td>
<td>3
AO2.1 (1)
AO2.2 (2)</td>
<td>Credit equivalent modulo arithmetic solution</td>
</tr>
</table>

5e	1 mark per bullet to max 3, e.g. • Use a different structure e.g. a linked list • …items can be added at different points in the linked list depending on priority • …by changing the pointers to items needing priority • Have different queues for different priorities • …add the job to the queue relevant to its priority • …print all the jobs in the highest priority queue first	**3** AO2.1 (2) AO2.1 (1)	Allow other suitable descriptions that show how the program could be amended.
6ai	1 mark per bullet • Points to where the next/first free node is • To add data into the linked listed.	**2** AO1.2 (1) AO2.2 (1)	
6aii	Points to the first element in the linked list	**1** AO1.2 (1)	

<table>
<tr>
<td>6aiii</td>
<td>1 mark per bullet
• No change made to nodes/pointers unaffected by this removal
• Index 0 points to 2 instead of 3
• Node 9 points to 3 instead of -1 // Node freeListPointer points to 3 instead of 4
• Node 3 points to 4 // -1 (must match MP3

Solution:
headPointer 0
freeListPointer 4
<table>
<tr><th>index</th><th>data</th><th>pointer</th></tr>
<tr><td>0</td><td>2.6</td><td>2</td></tr>
<tr><td>1</td><td>3.5</td><td>-1</td></tr>
<tr><td>2</td><td>1.8</td><td>1</td></tr>
<tr><td>3</td><td><s>6.9</s></td><td>-1</td></tr>
<tr><td>4</td><td></td><td>5</td></tr>
<tr><td>5</td><td></td><td>6</td></tr>
<tr><td>6</td><td></td><td>7</td></tr>
<tr><td>7</td><td></td><td>8</td></tr>
<tr><td>8</td><td></td><td>9</td></tr>
<tr><td>9</td><td></td><td>3</td></tr>
</table>
Alternative Solution:
headPointer 0
freeListPointer 3
<table>
<tr><th>index</th><th>data</th><th>pointer</th></tr>
<tr><td>0</td><td>2.6</td><td>2</td></tr>
<tr><td>1</td><td>3.5</td><td>-1</td></tr>
<tr><td>2</td><td>1.8</td><td>1</td></tr>
<tr><td>3</td><td><s>6.9</s></td><td>4</td></tr>
<tr><td>4</td><td></td><td>5</td></tr>
<tr><td>5</td><td></td><td>6</td></tr>
<tr><td>6</td><td></td><td>7</td></tr>
<tr><td>7</td><td></td><td>8</td></tr>
<tr><td>8</td><td></td><td>9</td></tr>
<tr><td>9</td><td></td><td>-1</td></tr>
</table>
</td>
<td>4
AO1.2 (1)
AO2.2 (1)</td>
<td>6.9 3 may or may not be written by candidates, both are acceptable.

Candidates may add the node freed up (node 3) to the start or the end of the free storage. Award marks for both approaches.</td>
</tr>
</table>

<table>
<tr>
<td>6bi</td>
<td>1 mark per bullet
• Class declaration and all code is nested within the class
• Two private identifiers data and pointer (with suitable data types if given)
• Public constructor heading as a procedure (public may be implied but cannot be private) taking both parameters as given in table
• Assigns parameters to the attributes

e.g.
<pre>public class node
  private data as real
  private pointer as integer
  public procedure new(newData, newPointer)
    data = newData
    pointer = newPointer
  endprocedure
endclass</pre></td>
<td>4
AO2.2
(1)
AO3.3
(3)</td>
<td>Accept

public node(newData, newPointer)
(may also have data stypes for parameters e.g. int newData)

Accept:

this.data = newData
this.pointer = newPointer

or similar</td>
</tr>
<tr>
<td>6bii</td>
<td>1 mark per bullet to max 2
• A get method allows the attribute to be accessed / returned
• A set method allows the attribute to be changed (with parameters)</td>
<td>2
AO2.2
(2)</td>
<td></td>
</tr>
<tr>
<td>6c</td>
<td>1 mark per bullet to max 6
• Initialise message string
• Start with the headPointer
• Check if the headPointer is null
• …return that the list is empty
• Check the pointer of the node at headPointer
• If it is not null/-1/the last element
• loop through all the linkedList elements
• …concatenate the pointer to the message
• …replacing the pointer with the current node’s pointer each time
• …if the data is found concatenate the pointer and “found” to the message and return it
• …if the loop ends and the data item is not found, concatenate “not found” to the message and return it</td>
<td>6
AO1.2
(2)
AO2.1
(2)
AO2.2
(2)</td>
<td></td>
</tr>
</table>

6di	1 mark for identifying error and correction (identification may be implicit)
	
	• Line 02 tempPointer should become headPointer, not -1
	`tempPointer = headPointer`
	• Line 05 message should say it's empty not full
	`print("List is empty")`
	• Line 07 pointer should be tempPointer
	`while linkedList[tempPointer].getPointer() != -1`
	• Line 08 Incorrect call to node pointer `dataToPrint = dataToPrint + " " + linkedList[tempPointer].getData()`
	• Line 09 assignment is wrong way
	`tempPointer = linkedList[tempPointer].getPointer()`
	• Line 11 missing final parenthesis `print(dataToPrint+ " " + linkedList[tempPointer].getData())`
		3
		AO2.1
		(2)
		AO2.2
		(2)
			Do not award marks for stating the line number without a valid correction.

6dii	1 mark per bullet • Stepping Through The Code… • …to run one line at a time to see where the error is • Syntax Error Highlighting…. • ….to distinguish syntax errors in the program code • Setting breakpoints… • ….to debug individual sections of code at a time • Variable watch window… • …To check that the variables are being updated corrected	**6**	The features must relate to debugging code. Allow other suitable features appropriate to debugging code. 1 Mark for identification and 1 mark for suitable expansion.

6e	**Mark Band 3 – High level (9-12 marks)** The candidate demonstrates a thorough knowledge and understanding of the object orientied techniques; the material is generally accurate and detailed. The candidate is able to apply their knowledge and understanding directly and consistently to the context provided. Evidence/examples will be explicitly relevant to the explanation.The candidate is able to weigh up the use of all of the object orientied techniques which results in a supported and realistic judgment as to whether it is possible to use them in this context. *There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.* **Mark Band 2 – Mid level (5-8 marks)** The candidate demonstrates reasonable knowledge and understanding of the object orientied techniques; the material is generally accurate but at times underdeveloped. The candidate is able to apply their knowledge and understanding directly to the context provided although one or two opportunities are missed. Evidence/examples are for the most part implicitly relevant to the explanation. The candidate makes a reasonable attempt to come to a conclusion showing some recognition of influencing factors that would determine whether it is possible to use each object orientied technique in this context. *There is a line of reasoning presented with some structure. The information presented is in the most part relevant and supported by some evidence* **Mark Band 1 – Low Level (1-4 marks)** The candidate demonstrates a basic knowledge of the object orientied techniques with limited understanding shown; the material is basic and contains some inaccuracies. The candidates makes a limited attempt to apply acquired knowledge and understanding to the context provided. The candidate provides nothing more than an unsupported assertion. *The information is basic and comunicated in an unstructured way. The information is supported by limited evidence and the relationship to the evidence may not be clear.*	**12** AO1.1 (3) AO1.2 (3) AO2.1 (3) AO3.3 (3)	**AO1: Knowledge and Understanding** **Indicative content** • Classes, this a template. It will define what attributes and methods an object should have. • Objects, when you create an instance of a class. Each object that is instantiated from the same class will share the same attributes and methods. • Inheritance, when a sub class takes on the attributes and methods from a superclass/parent class. It can also have its own extra attributes/methods. • Overriding, when a method name is the same in a parent and sub class, then the method in the parent/super class will be overridden • Encapsulation, this protects attributes of an object by making them private so that they can't be accessed or altered accidentally by other objects. **AO2: Application** • A class can be used to declare the attributes and methods for the linked list. These will initialise the nodes and join them. • Objects can then be used be used to instantiate the class each time a new linked list is needed. Each can be given a different identifier by the other programs. • Further subclasses may be used by other programs. These can therefore take on the attributes and methods from the base class. These can also be changed or overridden depending on the purpose of the other programs. • Encapsulation can be used by using set and get methods to ensure that the nodes in the linked list are changed in a way that is intended. **AO3: Evaluation**

	0 marks No attempt to answer the question or response is not worthy of credit.		• Use of OPP techniques will allow for code reusability. His linked list can be saved as library and then reused many times leading to less code. • OOP also allows programs to be easier to modify and maintain.

OCR (Oxford Cambridge and RSA Examinations)
The Triangle Building
Shaftesbury Road
Cambridge
CB2 8EA

OCR Customer Contact Centre

Education and Learning
Telephone: 01223 553998
Facsimile: 01223 552627
Email: general.qualifications@ocr.org.uk

www.ocr.org.uk