GCST

2.3.2
ALGORITHMS - SORTING & SEARCHING & BIG O
TOPIC WISE EXAM QUESTIONS

ANSWERS

A-LEVEL OCR

7	(a)		**Mark Band 3 – High level** **(7-8 marks)** The candidate demonstrates a thorough knowledge and understanding of Big O; 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 Big O; 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 Big O 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: Knowledge and Understanding** **Indicative content** • Big O measures the number of steps and memory usage change according to the data as the amount of data being processed increases • Linear - grows in proportion to amount of data • Exponential – the rate of increase is at the rate k^n as n increases • Constant - it does not change • Logarithmic – means the rate of increase gets smaller as the amount of data increases time / time increases at a rate of $\log_k n$ as n increases. **AO2: Application** • Algorithm 1 – The time taken increases as the data set grows. The space taken also significantly increases. This algorithm is not memory efficient. • Algorithm 2 – The time increases significantly and therefore this algorithm is not time efficient. The space will never change which means the amount of memory will not change as the data set grows. • Algorithm 3 – The time will grow less fast as the data set grows relative to the other algorithms. The space required will also increase, but not insurmountably. This is therefore an efficient algorithm with large data sets compared to algorithm 1 and 2 overall. **AO3: Evaluation** • Number of elements is unknown. Exponential is least appropriate because this could increase significantly and be unmanageable. • Constant is the most ideal as the time will not increase. • Algorithm 3 is more suitable because it has a logarithmic time complexity, so it increases less quickly than the other algorithms. It will be reasonable with a small amount (2 items) of data, but then when very large amounts (2 billion items) are needed it will not be significantly more.
7	(c)	(i)	1 mark each to max 5 • The data list is split into two lists • These sublists continue to be (recursively) split… • …until each sublist is one item • The first element in two different sublists is compared… • …the smaller item is then selected… • …and written to a new list • …until both sublists fully merged • Repeated until all sorted sublists are recombined	5	Allow array/list as equivalent

1	a		1 mark per bullet • 1st swap of 5 and 3 • Remainder of first pass • Pass 2 • Pass 3 1 5 3 9 2 7 1 **3** **5** 9 2 7 1 3 5 **2** **9** 7 1 3 5 2 **7** **9** End of pass 1 1 3 2 5 7 9 End of pass 2 1 2 3 5 7 9 End of pass 3	4	Candidates do not need to show each swap, so if the candidate has clearly shown the end of pass 1, they have met the first two marking points. Marks can be awarded for correctly showing the results of each pass.
1	b	i	1 mark per bullet • by reference will reorder the contents of the array • …so the new order can be accessed by the main program // so will be saved when the procedure ends • by value will change the array only in this procedure • … and so would need to return the array.	3	
1	b	ii	• A loop that repeats a fixed / set number of times	1	
1	b	iii	• To temporarily hold a value (for numbers[x])… • …while it is being transferred from one position to another… • ….in the array numbers • To stop values over writing each other	3	
1	b	iv	• Add a (second outer) loop • That will repeat for each pass // repeat until the flag is set to true at the end of a pass	2	
1	c	i	• 355	1	
1	c	ii	• Insertion sort	1	Accept any valid sorting algorithm e.g. Merge sort, Quick sort
6	a	i	1 mark per bullet • Start with the first element • Compare it to the number input • If it is equal, return the index // True • If not equal, move to the next element and repeat • Repeat until it is found, or the end of the array is reached • If found, return the index where the data was found • If the end of the list was reached, return -1 // False // "not found" message	5	
6	a	ii	1 mark per bullet • If the data is not in any order // binary search requires the data to be in order • When the number of items to search is small	1	

GCST

2022

1ai	The array/data must be in order/sorted	1	
1aii	1 mark per bullet • Compare the search item with the first value • ....then compare the search item with the next value • ....repeat the above process until either • ....the end of the array has been reached or • ....the search item is found and then stop • ....then return the array position // return -1 / False if not found	4	Not all mark points are dependent, b points awarded must following logica in sequence.
3a	1 mark for: • Can refer to all 50 only using one identifier // all values can be indexed in one array • The numbers can be passed as a single parameter • Does not need 50 variables to be declared/passed	1	

3b	1 mark for each completed statement
	
	`arrayLength = 50 // numberArray.length`
	
	`tempValue =0`
	
	`do`
	
	`  flag = false`
	
	`  for y = 0 to arrayLength - 2`
	
	`  if numberArray[y] > numberArray[y + 1] then`
	
	`  tempValue = numberArray[y]`
	
	`  numberArray[y] = numberArray[y + 1]`
	
	`  numberArray[y + 1] = tempValue`
	
	`  flag = true`
	
	`  endif`
	
	`  next y`
	
	`until flag == false`
		5	Note if numberArray - 1 // 49 used, t for loop for y will need to be 0 to arrayLength - 1
			
			Allow other suitable valid identifier ir place of tempValue e.g. temp

3c	1 mark for each stage shown • Splitting into individual items 2 12 1 9 3 5 15 7 • Combining in pairs 2 12 1 9 3 5 7 15 • Merge pairs 1 2 9 12 3 5 7 15 Merge for final 1 2 3 5 7 9 12 15	4	Do not award a mark for the final stage unless candidate has shown the previous sorting stage(s).

3d	**Mark Band 3 – High level** **(9-12 marks)** The candidate demonstrates a thorough knowledge and understanding of 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. *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 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 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** **(1-4 marks)** The candidate demonstrates a basic knowledge of 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 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.	12 AO1.1 (3) AO1.2 (3) AO2.1 (3) AO3.3 (3)	**AO1: Knowledge and Understanding** **Indicative content** • Merge sort splits data into individual lists an merges • Insertion makes first value sorted list, then inserts each item into the sorted list • Bubble sort looks through each item in turn, number of items times **AO2: Application** • Merge uses more memory as new lists are needed. Insertion and Bubble need constant memory. • Bubble and Insertion have the best best-times, both O(n) because they run through the data once. merge sort requires a minimum number of stages so best case is longer (O(n log(n)) • Merge average is the same as best. Insertion and Bubble has average o(n^2). • Worst time merge has same has best and average because same number of stages are needed. Bubble sort and insertion all have worse O(n^2) **AO3: Evaluation** • There are a small number of elements (10) therefore a bubble sort of insertion would be better space wise because no further space is needed. • Merge would not need excessive amounts of more memory as there are only a small number of elements. • Time complexity, there is a small number of elements therefore Bubble and Insertion may be preferable. Differences are unlikely to be significant, so either would be more appropriate.

AS - Level

3	a		1 mark for each correct row up to a maximum of 4 marks. 89 25 75 37 45 25 89 75 37 45 — 1 mark 25 75 89 37 45 — 1 mark 25 75 37 89 45 — 1 mark 25 75 37 45 89 — 1 mark	4 A02.1 (4)	Marks should be awarded for correct swapping of adjacent items that are out of order. Therefore if the previous step is incorrect but the candidate has followed through with the correct answer then marks should be awarded.
3	b	i	• Line 5	1 A02.1 (1)	Allow line 4 and 12
3	b	ii	numbers	1 A02.1 (1)	
3	b	iii	1 mark per bullet up to a maximum of 2 marks, e.g.: • To temporarily hold data… • …To allow the contents of two variables to be swapped • …to ensure that data is not overwritten	2 A01.2 (2)	
3	b	iv	1 mark per bullet up to a maximum of 2 marks, e.g.: • signifies whether or not any swaps have been made in a pass • if still set to true at the end of a pass, then the list is sorted	2 A01.2 (2)	

5	a		1 mark per bullet up to a maximum of 3 marks, e.g.: • To start from the beginning // first booking slot • Search each slot in order // sequentially • Search until the first empty slot is found	3 A02.2 (3)	

5	b		1 mark per bullet up to a maximum of 7 marks, e.g.:
			• Defining the `findFirst` function correctly
			• Suitable logic for checking the first time slot
			• Suitable logic for checking the next time slot…
			• Suitable loop to check all time slots
			• Suitable logic for returning the available time slot
			• Suitable logic for returning `-1` if no time slots available
			• Suitable use of variable names and indentation
				7
				
				A03.1
				(7)
					Example solution:
					`function findFirst()`
					`count = 0`
					`do`
					`found = false`
					`if customerID[2, count] == "" then`
					`found = true`
					`else`
					`count = count + 1`
					`endif`
					`until count == 10 or found == true`
					`if found == True`
					`return customerID[0,count]`
					`else`
					`return -1`
					`endif`
					`endfunction`
					There are many different ways that this function could have been achieved. Therefore other alternative methods should be given credit.

5	c	iii	A procedure does not return a value / a function will return a value.	1 A02.2 (1)	

GCST

2021

2ai

1 mark for each number/statement up to a maximum of 6 marks:

```
function binarySearch(dataArray:byref, upperbound, lowerbound, searchValue)
  while true
    middle = lowerbound + ((upperbound - lowerbound) DIV 2)
    if upperbound < lowerbound then
      return -1
    else
      if dataArray[middle] < searchValue then
        lowerbound = middle + 1

      elseif dataArray[middle] > searchValue then
        upperbound = middle - 1
      else
        return middle
      endif
    endif
   endwhile
endfunction
```

6 AO1.2 (2) AO3.3 (4)

2aii

Do…until // repeat…until // post condition

1 AO1.2 (1)

2b

1 mark for each tick up to a maximum of 6 marks

Worst-case space complexity:

	Binary search	Linear search
O(log(n))		
O(1)	✓	✓
O(n)		

Best-case space complexity:

	Binary search	Linear search
O(log(n))		
O(1)	✓	✓
O(n)		

Average time complexity:

	Binary search	Linear search
O(log(n))	✓	
O(1)		
O(n)		✓

6 AO1.1 (6)

2c

1 mark for any example

e.g.

- Data is not sorted
- Item you are looking for is the first item in the list
- Small number of items

1 AO1.2 (1)

3a

1 mark per bullet, each must be applied correctly to data

- Choose a pivot // identify start and end pointers
- Compare each element to the pivot… // compare start and end pointers
- Put items < pivot in the left sublist
- Put items > pivot in the right sublist
- Choose a pivot in each sublist
- If start pointer is larger than end pointer…
- …then swap data items around
- And repeat the process until each item becomes a pivot

5 AO1.2 (2) AO2.2 (3)

3b

1 mark per bullet to max 2

- decomposing data sets into smaller subsets
- and then sorting each split subset
- until each subset is sorted
- and then combining the subsets to provide a solution

2 AO1.1 (1) AO2.1 (1)

GCSE

2020

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.

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)	

2 (a)	**Mark Band 3 – High level (7-9 marks)** The candidate demonstrates a thorough knowledge and understanding of bubble sorts; 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 bubble sorts within the context which results in a supported and realistic judgment as to whether it is suitable to use within the 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 knowledge and understanding of bubble sorts; 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 bubble sorts 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 bubble sorts 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** No attempt to answer the question or response is not worthy of credit.	9 AO1.1 (2) AO1.2 (2) A02.1 (2) A03.3 (3)	**Knowledge and Understanding** • All adjacent items are compared against each other. • The biggest number in the adjacent pair is swapped over with the smallest number. A temp variable is used to hold the data while it's being moved. • When a swap is made a flag is set. • This is repeated for all adjacent values, known as one pass. • At the end of one pass, the largest item should appear at the end of the list. • If at the end of the list the flag has been set the flag is unset and the algorithm starts from the beginning of the list again. • When the algorithm gets to the end of the list and the flag is unset the list is sorted. **Application** • As there are 250,000 items a bubble sort would perform very slowly as a lot of passes will need to be made in order to sort the items. • Bubble sorts are better suited to data sets where the items are almost/partly sorted. However the smaller numbers are currently towards the end and the larger numbers are towards the start. • This will therefore increase the amount of comparisons / passes/swaps required which will therefore slow the performance of the sort down. **Evaluation** • The algorithm is easy to implement as the number of lines of code is less than other standard sorting algorithms. • Although a bubble sort will be able to sort the items into order, it will take longer than other sorting algorithms due to the number of items and the current order or items in the unsorted list.

2	**(b)**	**(ii)**	• 249,999	1 A02.2 (1)	
2	**(c)**	**(iii)**	1 mark per bullet up to a maximum of 4 marks: • The inner loop will compare all of the adjacent items…. • ….in a single pass • The outer loop will repeat the process of checking adjacent items…. • …until all passes are complete / the items are sorted/no swaps have been made in a pass	4 A01.2 (4)	Allow other valid interpretations e.g. conditional while loop … used to compare against swap flag after each pass; counter controlled for loop .. used to check adjacent items on each pass
2	**(d)**	**(iv)**	• Insertion sort	1 A02.1 (1)	Allow other sorting algorithms not listed in the specification (e.g. Merge Sort, Quick Sort etc)
3	**(b)**	**(i)**	1 mark per bullet up to a maximum of 3 marks: • 5 (Jimmy) • 8 (Siad) • 9 (Tommy)	3 A02.1 (3)	
3	**(b)**	**(ii)**	1 mark per bullet up to a maximum of 2 marks, e.g: • Efficient • ...as does not need to search every single element/uses divide and conquer	2 A01.2 (2)	
3	**(b)**	**(iii)**	• Linear Search / Serial Search	1 A02.1 (1)	
3	**(b)**	**(iv)**	• The items are in alphabetical order / the items are sorted	1 A02.1 (1)	

1biii	1 mark per bullet to max 5. Max 3 if no application to data in `processedData` • Compares each pair of data e.g. 0 and 0.5 • If they are in the correct order it moves to the next pair e.g. 0.5 and 0 • If they are in the wrong order it swaps them e.g. 0.5 and 0 becomes 0 and 0.5 • Continues to the end of the array e.g. Pass 1 complete • If there has been a swap it checks again e.g. Pass 2 complete • If there have been no swaps it is sorted 0 | 0.5 | 0 | 1 | 2 | 1.5 | 1 | Pass 1 0 | 0 | 0.5 | 1 | 2 | 1.5 | 1 0 | 0 | 0.5 | 1 | 1.5 | 2 | 1 0 | 0 | 0.5 | 1 | 1.5 | 1 | 2 0 | 0 | 0.5 | 1 | 1 | 1.5 | 2 | Pass 2 0 | 0 | 0.5 | 1 | 1 | 1.5 | 2 | No swaps	**5** **AO1.1 (2)** **AO1.2 (1)** **AO2.1 (1)** **AO2.2 (1)**	Allow (full) credit for tables showing the bubblesort being completed.
1biv	1 mark per bullet O(n) • Linear • Best time grows at the same rate as the number of elements • This is the case when the data is already in order O(n^2) • Polynomial / Quadratic • Worst and average time is proportional to the square (polynomial) of the number of elements • Worst case is when the data is initially in the reverse order O(1) • Constant • Will always take the same amount of memory (in addition to the list itself).	**6** **AO1.1 (3)** **AO1.2 (3)**	Note: First Mark Point is for the identification, second Mark Point is for the description Note: Do not allow descriptions relating to time complexity for 'Worst Space O(1)' Note: Do not allow 'equal to' in descriptions, O(n) and O(n^2) grow in *proportion* to the number of items

5a	1 mark per bullet to max 6. Max 4 if generic description given with no application Max 4 if a diagrammatic solution is given with no description • Splits the list in half repeatedly… • … until it is in independent arrays / elements e.g. 2, 18, 6, 4, 12, 3 • Compare the first two items (index 0 and 1) e.g. 2, 18 • … and combine to create a new array in descending order i.e. 18, 2 • Repeat with indexes 2 and 3 (6, 4), then 4 and 5 (12, 3) • Compare the first element in the first two new arrays • …Choose the largest element, writing this to the new array first • …repeat until no elements left • Combine the two remaining lists into one list e.g. [2, 18, 6, 4, 12, 3] [2, 18, 6] [4, 12, 3] [2, 18] [6] [4, 12] [3] [2] [18] [6] [4] [12] [3] [18, 2] [6, 4] [12, 3] [18, 6, 4, 2] [12, 3] [18, 12, 6, 4, 3, 2] e.g. [2, 18, 6, 4, 12, 3] [2, 18, 6] [4, 12, 3] [2, 18] [6] [4, 12] [3] [2] [18] [6] [4] [12] [3] [18 2] [6] [12 4] [3] [18 6 2] [12 4 3] [18, 12, 6, 4, 3, 2]	6 **AO1.2 (3)** **AO2.2 (3)**	Allow max 5 if correct description but in ascending order.
5b	1 mark per bullet • Merge sort might create a new array each time it splits and merges / often implemented recursively which places additional data on the stack • Insertion sort does not use any additional arrays//Insertion sort is an in-place algorithm.	2 **AO1.2 (2)**	

2	c	i	1 mark per set of swaps 10 95 5 33 100 77 45 10 5 95 33 100 77 45 — 1 10 5 33 95 100 77 45 10 5 33 95 77 100 45 — 1 10 5 33 95 77 45 100 5 10 33 95 77 45 100 5 10 33 77 95 45 100 — 1 5 10 33 77 45 95 100 5 10 33 45 77 95 100 — 1	4 AO1.1 (1) AO1.2 (1) AO2.1 (2)	Pair swaps may be shown individually or at the end of each traversal (award mark point 1 and 2 if first traversal is shown on one line)

2	c	ii	1 mark per bullet max 8
			• Procedure declaration
			• …taking array as parameter by reference
			• Outer loop, looping the number of times there are elements…
			• Inner loop, looping through all elements of the loop (or one less element each iteration)
			• Comparing two elements
			• Swapping two elements if out of order
			• …using an appropriate temp memory space or equivalent
			• Efficiency: stopping as soon as no swaps have been made
			
			e.g.
			`procedure sortData(data[]:byRef)`
			
			`  swapped = true`
			`  while swapped == true`
			`   swapped = false`
			
			`   for innerCount = 0 to data.length - 2`
			`      if data[innerCount] > data[innerCount + 1] then`
			`         temp = data[innerCount]`
			`         data[innerCount] = data[innerCount + 1]`
			`         data[innerCount+1] = temp`
			`         swapped = true`
			`       end if`
			`    next innerCount`
			
			`  endwhile`
			
			`endprocedure`
				8
				
				AO1.1 (2)
				AO3.2 (6)	

2	c	iii	1 mark per row(s) as shown: 95 10 5 33 100 77 45 10 95 5 33 100 77 45 — 1 5 10 95 33 100 77 45 — 1 5 10 33 95 100 77 45 5 10 33 95 100 77 45 — 1 5 10 33 77 95 100 45 5 10 33 45 77 95 100 — 1	4 AO1.2 (2) AO2.1 (1) AO2.2 (1)	In the 3rd mark point the line may appear just once

GCST

2018

5	(d)	(i)	1 mark per row (after first row) 100 22 5 36 999 12 22 100 5 36 999 12 1 mark 5 22 100 36 999 12 1 mark 5 22 36 100 999 12 1 mark 5 22 36 100 999 12 1 mark 5 12 22 36 100 999 1 mark	5 AO2.2 (5)	
5	(d)	(ii)	1 mark per bullet to max 7 • Repeat • Calculating an array midpoint… • …by adding the array lower bound to the array upper bound, dividing by 2 and rounding • Compare array midpoint with value to search for… • …if equal set found flag to true • …if array midpoint < value to search for, change lowerbound to equal midpoint + 1 • …if array midpoint > value to search for, change upperbound to equal midpoint – 1 • Until lowerbound is greater than or equal to upperbound • Return/output found flag	7 AO1.1 (2) AO1.2 (3) AO2.1 (1) AO2.2 (1)	

AS - Level

2	d	i	1 mark per bullet to max 2 • Bubble sort is an inefficient algorithm… • Meaning it will take more time/processing cycles to sort the list. • Generally outperformed by Insertion sort/quick sort/ merge sort (accept any other sensible sorting algorithm) • The item to be sorted is at the end of the list (and so can only move back one place per pass) which is the worst case scenario for bubble sort.	2 AO1.2 (1) AO2.2 (1)	
2	d	ii	There are only a small number of data items	1 AO2.2 (1)	

2	d	iii	1 mark per bullet to max 6
			• Procedure declaration
			• Outer loop until no swaps made using flag
			• Inner loop to iterate through the list…
			• …allowance for largest value at end (in bounds)
			• Comparing elements
			• Swapping elements
			
			e.g.
			`procedure sort_scores()`
			`  swapped = True`
			`  while swapped:`
			`  swapped = False`
			`  for i = 1 to 9`
			`  if scores[i-1] < scores[i]:`
			`  # swap pair being compared`
			`  temp = scores[i-1]`
			`  scores[i-1] = scores[i]`
			`  scores[i] = temp`
			`  swapped = True`
			`end procedure`
				6
				AO1.2 (2)
				AO2.2 (1)
				AO3.2 (3)	

2	d	iv	1 mark for each completed space
			
			`procedure insertionSort()`
			`  for count = 0 to numbers.length-1`
			`  position = count`
			`  while position > 0 and numbers[position]<numbers[position-1]`
			`  temp = numbers[position-1]`
			`  numbers[position-1] = numbers[position]`
			`  numbers[position] = temp`
			`  position = position-1`
			`  endwhile`
			`  next count`
			`endprocedure`
				4
				AO2.2
				(3)
				AO3.2
				(1)	

3	b	i	The data needs to be sorted / in alphabetical order	1 AO2.1 (1)	
3	b	ii	1 mark per bullet to max 4 • Start at the first item (Cavalry) • Compare with departure station 'Bridge Heights' • If matched, report found • Otherwise continue to the next item in list (Bridge) • Continue until item found, or end of list reached… • and then False returned	4 AO2.1 (2) AO2.2 (2)	

2017

1	a	i	1 mark for each correct item in **bold**
			
			`procedure sortit(dataArray, lastIndex)`
			`  for x = 1 to lastIndex`
			`  currentData = dataArray[x]`
			`  position = x`
			`  while (position > 0 AND dataArray[x - 1] > currentData)`
			`  dataArray[position] = dataArray[position-1]`
			`  position = position - 1`
			`  endwhile`
			`  dataArray[position] = currentData`
			
			`  next x`
			`endprocedure`
				3
				AO1.1
				(3)
					answers must be in the correct case given
					e.g. **currentData**

1	a	ii	1 mark for contents of each row in table	6 AO2.1 (6)	… each row is dependent upon the preceding row being correct

6	**1**	**15**	**12**	**5**	**6**	**9**		
1	**6**	**15**	**12**	**5**	**6**	**9**	6 is the sorted list 1 is the compared to sorted list 1 is put in place in sorted list	1
1	**6**	**15**	**12**	**5**	**6**	**9**	15 is compared 15 is in place in sorted list	1
1	**6**	**12**	**15**	**5**	**6**	**9**	12 is compared 12 is in place in sorted list	1
1	**5**	**6**	**12**	**15**	**6**	**9**	5 is compared 5 is in place in sorted list	1
1	**5**	**6**	**6**	**12**	**15**	**9**	6 is compared 6 is in place in sorted list	1
1	**5**	**6**	**6**	**9**	**12**	**15**	9 is compared and put in place	1

1	b	i	O(n)	1 AO1.1 (1)	
1	b	ii	1 mark per bullet to max 3 • The best case is for a sorted list (O(n)) • As the number of elements increases • … the number of steps increases in a linear fashion	3 AO1.2 (3)	B(ii) dependent upon b(i) being correct i.e. answers for O(n) only Accept appropriate graph for bullet points 2 and 3
1	c		**Mark Band 3 – High level** **(7-9 marks)** The candidate demonstrates a **thorough** knowledge and understanding of how bubble sort works and Big O complexity; 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 of how bubble sort works and Big O complexity; 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 of how bubble sort works and Big O complexity 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** • Description of bubble sort: ◊ Starting at the beginning of the list Items are swapped with their neighbour if they are out of order. ◊ Each pair of neighbours is checked in order. ◊ When a swap is made a flag is set. ◊ If at the end of the list the flag has been set the flag is unset and the algorithm starts from the beginning of the list again. ◊ When the algorithm gets to the end of the list and the flag is unset the list is sorted and the algorithm finishes. • $O(n^2)$ denotes as the data size increases the time the list takes to sort increases in a quadratic manner. • O(1) denotes the space used is constant **AO2: Application** • As data set gets bigger, bubble sort's time gets larger at an increasing rate.. • Complexity doesn't denote the actual time but the order with which the time/space grows. • O(1) space complexity means no matter how big the data set becomes the amount of space (extra to the data itself) remains the same. • $O(n^2)$ time complexity means as n increases time increases by n^2 / if n doubles the time taken is squared. • Bubble sort can be tweaked with improvements (e.g. checking one less item per iteration and alternating

					sorting directions). • These optimisations don't change the complexity. IT will run a little quicker on smaller sets but time taken increases rapidly with data size. • When choosing an algorithm we may also want to take into account the average and best case scenarios. (in this case they are also the same for both algorithms.) **AO3: Evaluation** • The algorithms may have the same time complexity but this does not mean they take the same time to execute on the same data set. • Insertion sort generally performs quicker than bubble sort and is therefore preferable. (Neither scale well however.) • Both algorithms have a space complexity of O(1). This is because both algorithms are in-place (i.e. all sorting takes place within the actual data).
					• Both have a time complexity of $O(n^2)$ as a consequence of their nested loops. (NB last two points are only likely to appear in the very highest mark answers.)

AS - Level

1	a	i	1 mark each • Binary [1] • Linear [1]	**2 AO1.1 (2)**	

1	a	ii	If the algorithm name does not match the description given then max 4 marks for description.
			
			Binary
			• Works on an ordered set of data [1]
			• Find mid-point [1]
			• If equal to mid-point [1]
			  ○ Report found [1]
			• If less than mid-point[1]
			  ○ Make sub-list from left [1]
			• If greater than mid-point [1]
			  ○ Make sub-list from right [1]
			• Repeat with sub-list until found / sub-list is empty [1]
			
			Linear
			• Can work on both ordered and unordered data sets [1]
			• Get first element [1]
			• If equal [1]
			  ○ Report found [1]
			• If not equal [1]
			  ○ Move to next element [1]
			• Repeat for all elements, until found / end of list reached [1]
					For binary search, allow movement of upper bound and lowe bound instead of making sub-lists

1	b		1 mark per bullet
			• Looping through/outputting for each row [1]
			• Calculating/using the middle value [1]
			• Outputting the 6th value from the array correctly [1]
			e.g.
			`for x = 0 to 15`
			`  print(data[x,5])`
			`next x`
				3
				AO3.2
				(3)	

1	c		1 mark per bullet
			• Looping through each column [1]
			• Looping through each row [1]
			• ...Adding to a total [1]
			• ...Calculating average correctly [1]
			• Outputting average [1]
			e.g.
			`for y = 0 to 10`
			`  total = 0`
			`  for x = 0 to 15`
			`  total = total + data[x,y]`
			`  next x`
			`  print(total / 16)`
			`next y`
				5
				AO3.2
				(5)
					Note that the running counter for the sum in each row needs to be reset to get the calculating average correct.
					
					Note average output must appear inside the outer loop

4	a		• Find the middle point in the list / 21 / element 4 • Compare it to the value 47, false • Is 47 greater than middle point, true New subset is 46-51 / change lower bound to 46 / element 5 • Find the middle of the new subset / 47 / element 6 Is this value equal to 47, true Search finishes	**4** **AO2.1** **(4)**	Some marks such as the comparison may be by implication if the candidate's logic works Must refer to the list given in the question i.e. not a generic description
	b		• Finding `midpoint` and correctly checking if `midPoint` value is target value … • … and if so returning `true` • Correctly checking that all elements have been checked … • … and if so returning `false` • Identify top or bottom of list … • … if top then `leftPtr` set/passed as `midPoint + 1` … • … if bottom then `rightPtr` set/passed as `midPoint - 1` • Correct use of indentation (AO2.1)	**8** **AO3.2** **(7)** **AO2.1** **(1)**	Max 8 marks Note: candidates may have given a recursive algorithm and this should is perfectly acceptable.

Example iterative example

```
function findItem (numberArray integer[2000], targetNumber:integer, leftPtr:integer,
rightPtr:integer): boolean

  while (leftPtr <= rightPtr)

      midPoint = (leftPtr + rightPtr) DIV 2
      if (numberArray[midPoint] == targetNumber)
        return true
      else if (numberArray[midPoint] < targetNumber)
        leftPtr = midPoint + 1
      else
        rightPtr = midPoint - 1
      endif

  endwhile
  return false
endfunction
```

	c	i	• The integers in the list are unsorted (1)	**1** **AO2.1** **(1)**	
		ii	Identification (Max 1) • Perform a linear search Description (Max 2) • starting at the first element / each item is checked… • until value is found • or end of list reached and not found	**3** **AO1.1**	Accept serial

1			1 mark for linear search, 2 for justification Justification: • The array is not sorted (1) • Linear does not need ordered / linear goes through all elements from beginning / binary needs a sorted array (1)	3	
			Total	**3**	
2			Algorithm, max 1 • linear Justification, 1 mark per bullet to max 2 • Items do not have to be in a specific order • Binary needs items in order	**3** **AO1.1** **(1)** **AO2.1** **(2)**	No marks for justification if linear has not been identified **Examiner's Comment:** Many candidates correctly identified a linear search and could justify the need for it. However, a lot of candidates did answer binary search without appreciating that the data set needed to be in order first.
4		i	The data needs to be sorted / in alphabetical order	**1** **AO2.1** **(1)**	Examiner's Comments Most candidates knew that the list had to be sorted before a binary search could be performed.
		ii	1 mark per bullet to max 4 • Start at the first item (Cavalry) • Compare with departure station 'Bridge Heights' • If matched, report found • Otherwise continue to the next item in list (Bridge) • Continue until item found, or end of list reached… • and then False returned	**4** **AO2.1** **(2)** **AO2.2** **(2)**	Examiner's Comments The majority of candidates described a linear search, but some described a binary search by mistake. Where candidates did describe a linear search, they generally did so with sufficient precision.
7			1 mark for each bullet • Duck is smaller than goat • Duck is less than frog/elephant • Duck is equal to duck/less than elephant so only duck left	3	

If you found this useful, drop a follow to help me out!

THANK YOU!

GCST

GCSECOMPUTERSCIENCETUTOR.COM