NAME DATE STARTED TARGET END DATE

GCE AS LEVEL — COMPUTER SCIENCE UNIT 1 QUESTION PACK

2500U10-1 · 2015 spec Unit 1 Topic 6 · AS unit, first sat 2017, 100 marks, 2h paper

REVISE.wales

COMPUTER SCIENCE — UNIT 1 · Algorithms — Searching, Sorting & Trace

Topic 1.5 — Algorithm design, linear and binary search, bubble sort, dry-running and compression

Writing pseudo-code algorithms with appropriate constructs and identifiers, tracing algorithms to predict output, fixing logic errors in bubble-sort and other code, comparing linear and binary search, and the principles of data compression.

2015 specification · current

Estimated time for entire question pack: ~2 h 26 min

Derived from the Unit 1 pace of ~1.2 min/mark, padded for written-prose answers (97 marks over 12 questions).

*You are advised to **not** attempt to complete all of this in one sitting.*

ABOUT THIS QUESTION PACK

This is a **comprehensive topic question pack**, not a single mock paper. It contains every question from the WJEC AS Unit 1 papers (Summer 2017 – Summer 2024, COVID gap) that maps onto Topic 1.5 of the 2015 specification.

Questions are ordered by source paper date.

INSTRUCTIONS

Use black ink or black ball-point pen. Show all working. A calculator is allowed where useful.

For Examiner's use only

Q	Source	Max	Mark
1	S17 Q10	8	
2	S17 Q12	4	
3	S18 Q8	8	
4	S18 Q9	9	
5	S18 Q10	11	
6	S19 Q2	6	

Q	Source	Max	Mark
7	S19 Q10	13	
8	S22 Q10	8	
9	S23 Q7	8	
10	S23 Q10	6	
11	S24 Q5	8	
12	S24 Q9	8	
Total		**97**	

Algorithms — Searching, Sorting & Trace — what the spec asks

WJEC GCE AS Computer Science (from 2015) · Unit 1: Fundamentals of Computer Science · Topic 1.5.

Algorithm design

- Use self-documenting identifiers (e.g. total, count, isPrime).
- Indent control structures consistently.
- Combine sequence, selection (IF / CASE) and iteration (WHILE / FOR / REPEAT).
- Validate input early; handle edge cases (empty, single, max).

Linear search

- Walk array start-to-end comparing each element to target.
- O(n) time; works on unsorted data.
- Best for small or unsorted lists.
- Worst case = whole list scanned with no match.

Binary search

- Requires sorted array; repeatedly halves the search range.
- Compare middle element with target; recurse / iterate on left or right half.
- O(log n) time; large speed-up for big sorted lists.
- Maintain start, end, mid pointers; terminate when start > end (not found).

Bubble sort

- Repeatedly pass through list, swap adjacent out-of-order pairs.
- After each pass, largest element is at end (its final position).
- $O(n^2)$ worst case; O(n) if already sorted with 'swapped' flag.
- Simple but slow for large lists; useful pedagogically.

Insertion sort

- Builds sorted sublist one element at a time.
- Each new element shifted backwards into its position.
- Efficient for small or nearly-sorted lists.
- $O(n^2)$ worst case but in-place and stable.

Compression principles

- Lossless: reversible; e.g. run-length encoding, Huffman, ZIP.
- Lossy: discards perceptually unimportant data; e.g. JPEG, MP3.
- Trade-off: smaller files vs computational cost (and quality for lossy).
- Dictionary methods replace repeated substrings with short codes.

Crib sheet »

Algorithms — Searching, Sorting & Trace in one page

Quick-reference notes — revisit before each question.

Pseudocode conventions

Use named procedures, clear identifiers, indented control structures.
FOR / WHILE / REPEAT, IF / THEN / ELSE, CASE.
Declare types of variables.

Linear vs binary

Linear: any list, O(n).
Binary: sorted only, O(log n).
1024 items: linear ~1024 steps; binary ~10 steps.

Bubble sort with flag

Repeat passes through array.
If a pass makes no swaps (flag stays false) ⇒ sorted, exit early.
Best O(n), worst $O(n^2)$.

Dry-running

Trace table: columns for each variable + output.
Step through every iteration.
Spot infinite loops, off-by-one, missing increments.

Algorithm debugging

Identify the bug: trace; check loop bounds; check initialisation; check IF conditions; check parameter passing.
State the line and the fix.

Compression types

Lossless: ZIP, PNG, FLAC — reversible.
Lossy: JPEG, MP3, MP4 — smaller, irreversible quality loss.

Examiner only

10. Write an algorithm using pseudo-code that determines if a number entered (between 3 and 256 inclusive) is a prime number.

A prime number is a positive number that is divisible only by itself and 1.

Your algorithm should be written using self-documenting identifiers and include suitable inputs and outputs. [8]

Examiner only

12. Describe how bubble sort and insertion sort algorithms operate. [4]

Examiner only

8. *EuroTravel* is a travel agency that offers its customers the option of purchasing foreign currency.

Assume the conversion from pounds (£) into euros (€) is:

£1.00 = €1.14

EuroTravel wants to be able to enter a value in pounds and provide its customers with a conversion into euros for each value £5 below and £5 above the initial value input.

For example, if the user inputs £500.00, the algorithm will output:

£495.00	=	€564.30
£496.00	=	€565.44
£497.00	=	€566.58
£498.00	=	€567.72
£499.00	=	€568.86
£500.00	**=**	**€570.00**
£501.00	=	€571.14
£502.00	=	€572.28
£503.00	=	€573.42
£504.00	=	€574.56
£505.00	=	€575.70

Write an algorithm for *EuroTravel* to meet these requirements, using pseudo-code.

Your algorithm should output a suitable error message for any data entered that is not a number.

Your algorithm should be written using self-documenting identifiers. [8]

Examiner only

9. Two different types of search algorithm are linear search and binary search.

(a) Explain how these search algorithms operate. [6]

(b) Describe appropriate circumstances for the use of each search algorithm. [3]

Examiner only

10. The following bubble sort algorithm attempts to sort integers stored in myArray, but contains an error.

```
1   Start Procedure SortMyArray
2   n is integer
3   temp is integer
4   swapped is boolean
5
6   set n = length(myArray) {returns the length of myArray}
7   repeat
8     set swapped = FALSE
9     for i = 0 to (n - 1)
10      if myArray[i] < myArray[i + 1] then
11         temp = myArray[i + 1]
12         myArray[i + 1] = myArray[i]
13         myArray[i] = temp
14         swapped = TRUE
15      end if
16    end for
17   until (swapped = TRUE)
18
19   End Procedure
```

(a) Suggest appropriate test data to dry-run this type of algorithm in order to identify possible errors. [3]

Test Data Set 1

Test Data Set 2

Test Data Set 3

(b) Describe how a bubble sort algorithm should operate. [2]

.....

.....

.....

.....

.....

.....

Examiner only

(c) Explain why the bubble sort algorithm in this question will fail. [2]

(d) Suggest a suitable change that could be made to the algorithm to overcome this problem. [1]

(e) Name and describe a different sort algorithm. [3]

Examiner only

2. This is an algorithm which should have four outputs, `a`, `b`, `c` and `d`. The algorithm does not work as intended.

```
1    define myFunction
2    declare c as integer
3    declare d as integer
4          set c = 3
5          set d = 4
6    end myFunction
7
8    Start MainProg
9    declare a as integer
10   declare b as integer
11
12   set a = 0
13
14   if a = 0 then
15         set b = 1
16   end if
17
18   call myFunction
19
20   output a
21   output b
22   output c
23   output d
24
25   End MainProg
```

(a) State the outputs that this algorithm will give. [2]

Examiner only

10. Two different types of search algorithm are binary search and linear search.

(a) The following is a binary search algorithm.

```
1   declare myArray[0 to 10]
2   declare start is integer
3   declare end is integer
4   declare found is Boolean
5   declare mid is integer
6
7   set start = 0
8   set end = 10
9   set found = FALSE
10
11  input searchValue
12
13  repeat
14        set mid = (start + end) DIV 2
15        if searchValue = myArray[mid] then
16              set found = TRUE
17              Output "searchValue found at position", mid
18        end if
19
20        if searchValue > myArray[mid] then
21              set start = mid + 1
22        end if
23
24        if searchValue < myArray[mid] then
25              set end = mid – 1
26        end if
27
28  until (found = TRUE) OR (end < start)
```

By crossing out or shading the discarded elements in the diagram below, show how the algorithm will reduce the part of the array being searched at each repetition.

`searchValue` = 22 [3]

	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
Original Data	12	22	27	31	38	54	63	71	73	87	92
Repetition 1	12	22	27	31	38	54	63	71	73	87	92
Repetition 2	12	22	27	31	38	54	63	71	73	87	92
Repetition 3	12	22	27	31	38	54	63	71	73	87	92

`myArray`

Examiner only

(b) Write a linear search algorithm, using pseudo-code, for the following array.

(1)	(2)	(3)	(4)	(5)
45	12	98	54	56

`myArray`

Your algorithm should output the position of the `searchValue` if it is found or a suitable message if the `searchValue` is not found.

Your algorithm should be written using self-documenting identifiers. [8]

Examiner only

10. The following algorithm calculates the area of a circle for a radius input by the user.

```
1    StarProc areaOfCircle
2    A is real
3    B is real
4    C is real
5
6    set C = 3.14
7
8    output "Please enter the radius"
9
10   input A
11
12   B = C * (A * A)
13
14   output "The Area is ", B
15
16   End Proc
```

(a) Identify and define the following terms in the algorithm above.

(i) Constant. [2]

(ii) Variable. [2]

(b) Describe why the use of self-documenting identifiers is important in programs and suggest suitable changes that you would make to the algorithm above to achieve this. [4]

Examiner only

7. The area and circumference of a circle can be calculated using the following formulae:

Area of a circle	**Circumference of a circle**
$A = \pi r^2$	$C = 2\pi r$

Where $\pi = 3.14$ and r = *radius*.

Write an algorithm, using pseudo-code, which will allow a user to enter the radius to calculate the area and circumference of a circle. The values input should be in metres, but the output should be converted into centimetres ($1\,m^2 = 10{,}000\,cm^2$ and $1\,m = 100\,cm$).

For example:

Input	**Output**
r = 1.5 (metres)	A = 70650 cm^2 C = 942.0 cm

Your algorithm should output a suitable error message if the value entered is not valid.

Your algorithm should be written using self-documenting identifiers. [8]

Examiner only

10. Describe the principles of data compression algorithms. [6]

Examiner only

5. A builders' merchant wants to convert imperial measurements for length (inches) into metric measurements.

The conversion from inches into metric measurements is as follows:

1 inch = 2.54 cm
1 inch = 0.0254 m

Write an algorithm, using pseudo-code, that will allow a user to input:

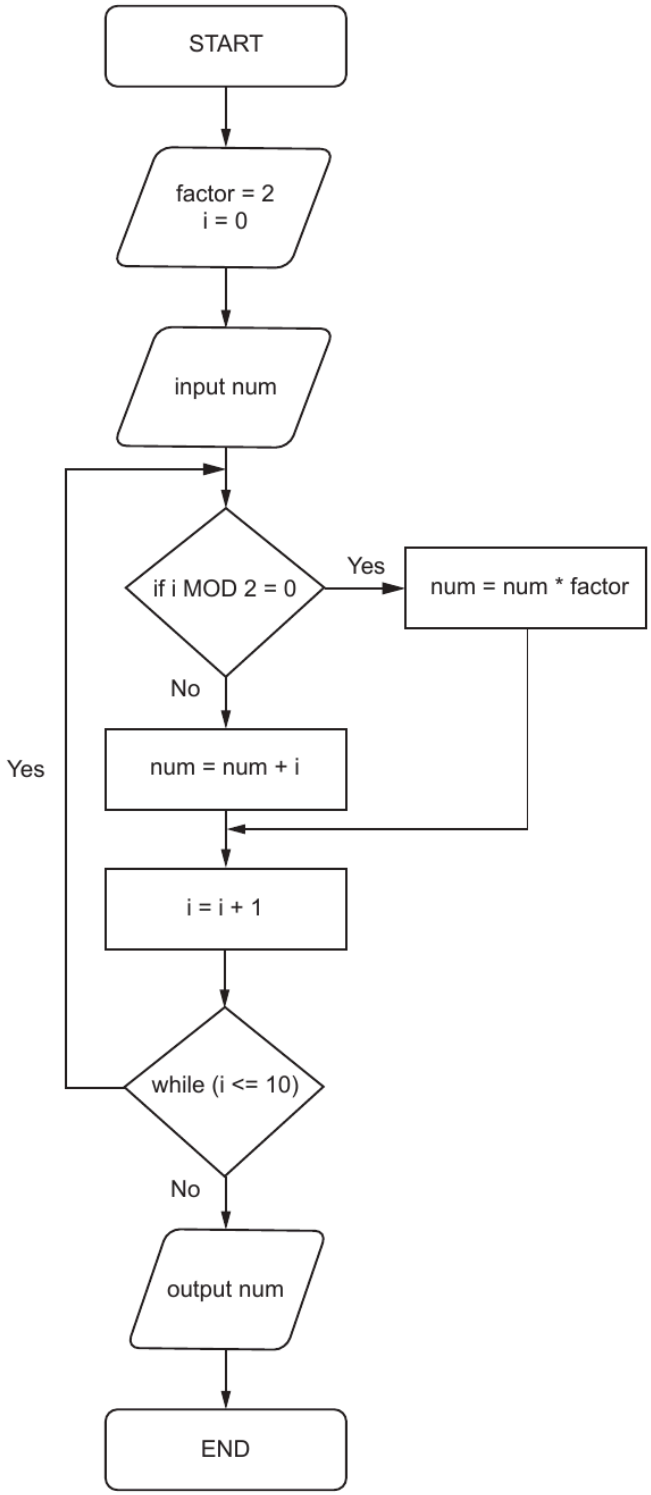
- the imperial measurement (in inches)
- whether the output is to be in cm or m.

As the number of values to be input is unknown, your algorithm should terminate when a user inputs a rogue value less than 0.0.

Your algorithm should output a suitable error message for any measurement entered that is not a real data type, for example a, B, z, @.

Your algorithm should be written using self-documenting identifiers. [8]

9. Consider the constructs in the following algorithm.
Note: there is no need to dry-run the algorithm.



Examiner only

Identify and describe the purpose of the following constructs in the algorithm:

(a) Constant. [2]

(b) Variable. [2]

(c) Selection. [2]

(d) Repetition. [2]

END OF QUESTION PACK

12 questions · 97 marks · ~2 h 26 min
Source: WJEC AS Computer Science Unit 1 (2500U10-1), Summer 2017–2024, COVID gap
Curated for WJEC Computer Science 2015 spec AS Unit 1 — Topic 6 (1.5)