4.1 Abstraction and automation part 2
Mark Scheme

Q1.

(a)
```
x ← 5
y ← 3
Result ← 1
REPEAT
Result ← Result * x
y ← y – 1
UNTIL y=0
```

x	y	Result
5	3	1
5	**2**	**5**
5	**1**	**25**
5	**0**	**125**

1 mark for each entry in column Y (max 3)
1 mark for each entry in column Result (max 3)
1 mark for not changing value of x (max 1)

7

(b) Calculate 5^3 // calculate 5×5×5 // calculate x^3 // calculate x^y // multiply x by itself y times;

1

[8]

Q2.

(i) Allow addresses in the Pointer column.

Position	Name	Running Time	Address	Pointer
1	Process6	7	01400	4 (02300)
3	Process7	17	01700	5 (04100)
4	Process2	17	02300	3 (01700);
5	Process9	45	04100	-1; **A** 0;
6	Process5	2	01200	8 (01900)
8	Process19	5	01900	1 (01400);

1 mark for 4,5,3 correct

1 mark for null pointer correct **A** sensible const. Name representing null pointer

1 mark for 8,1 correct

3

(ii) Array; of records; *OR* linked list; of records; *OR* 4 1-D arrays;
One for each column; *OR* one 1-D array for process name;
One 2-D arrays for numerical data;

2

(iii) Marks to be allocated as follows:

1 for initialisation	ListPointer ← HeadPointer;
1 for while not at end of list	While ListPointer <>-1 Do;
1 for printing	Print ListArray[ListPointer].Name;
1 for getting next pointer	ListPointer ← ListArray[ListPointer].Pointer;
P1 if headpointer is reassigned	

Any name acceptable for ListPointer and ListArray
Note: a sorting method gets a maximum of 3 marks (inefficient)

Alternative solution
REPEAT UNTIL next=-1 OR IF listpointer <>-1 then REPEAT..

4

(iv)

List	*Reason*
List of suspended/blocked/halted/ unrunnable processes;	waiting for a resource or complete a requested I/O transfer;
List of inactive/dormant jobs;	Waiting to be admitted to the system;

I currently running processes
I interrupt

2

[11]

Q3.

Result	Index
0	0
24	1
24	2
57	3
57	4

(a) Mark for each correct entry in Result – *max 4 marks*
A blank as a repeat of the entry above

1 mark for all the entries in Index;

5

(b) Obtain the largest value;

1

[6]

Q4.

(a) A procedure/routine which calls itself//is defined in terms of itself;
R re-entrant
A function instead of procedure
R program iteration Talked Out (no mark)

1

(b) (i)

E	L	H	M	List[M]	Printed Output
6502	1	11 ;	6	5789 ;	
6502	7	11 ;	9	8407 ;	
6502	7	8 ;	7	6502 ;	
					True;

Accept True in row 3
Marks in each row for all three/two parts correct
Accept empty cell to mean: same as in previous row.
Stop marking when logic goes wrong

7

(ii) Binary search;;
Search;
R any other type of search

2

[10]

Q5.

(a) EBCDIC/EBCD;
ASCII;
UNICODE;
A minor spelling variations

Any 2

2

(b) (i)

X	Index	Result			
		[3]	[2]	[1]	
835	0	–	–	–	
83	1	–	–	53	
8	2	–	51		
0	3	56			

1 mark for each correct entry

6

(ii) Convert a number into its character codes;

1

[9]

Q6.

(a) It calls itself / is defined in terms of itself / is re-entrant / contains within its body a reference to itself;

Ensure 'it' refers to procedure, if meaning program or object no mark

1

(b) The current state of the machine must be saved/preserved so can return correctly to previous invocation of B;
OR
Return address / procedure parameter / status register / other register values / local variables must be saved/preserved so can return correctly to previous invocation of B**)**;

1

(c)

Call Number	Parameter
1	53
2	26
3	13
4	6 ;
5	3 ;
6	1 ;

Printed Output: 1 1 0 1 0 1;;;

1 mark for each correct pair of bits
Mark from left and stop marking when error encountered ignore punctuation. if more than 6 bits give a max of 2 marks

6

(d) Conversion (of a denary number) into binary;

1

[9]

Q7.

(a) (i) Var S1: String / Var S2: String / Var Ptr : Integer / Var L : String; 1

(ii) IF S1 = S2;

1

(iii) For Ptr := 1 To 3 Do;

1

(b)

subroutine	procedure	function
copy		Y;
concat		Y;
print	Y;	

3

(c)

S1	Ptr	L	S2
"PAT"			""
	1	"P"	"P"
	2	"A"	"AP"
	3	"T"	"TAP"
Printed Output:			**False**

8

If S2 at end contains "PAT" then f.t. for True;
(If S2 does not contain "TAP" check that printer output is correct
Depending on what is in S1 and S2 in the candidate's answer)
1 mark for each correct entry, 1 mark for S1 correctly left as "PAT" or empty

[14]

Q8.

(a) Head (Tail (Days)) = Mon
R [Mon], MON (1)

Tail([Head(Days)]) = [ ] (1)

Empty(Tail(Tail(Tail(Days))))=False (1)

3

(b) Elements in a list can only be accessed sequentially;
elements in an array can be accessed directly;
using the subscript;

Any 2 points

2

[5]

Q9.

(a)

Ptr	Temp	List									
		[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
		43	25	37	81	18	70	64	96	52	4
1	43	25	43								
2	43		37	43							
3											
4	81				18	81					
5	81					70	81				
6	81						64	81			
7											
8	96								52	96	
9	96									4	96
10											

Ignore Ptr & Temp column
1 mark for each of rows 1, 2, 4, 5, 6, 8, 9
(Final list 25, 37, 43, 18, 70, 64, 81, 52, 4, 96)

7

(b) Control will pass to the instruction after Endwhile;
/the instruction/command/statement after Endwhile will be executed;
Program will exit while-block; loop stops;
A algorithim stops; **R** program stops;

Max 1

(c) (i) 25;

If part (a) not fully correct allow follow through: or lower of [1] & [2]

3

(ii) 81;

Only allow follow through mark if the list at the end of part(a) is still a partially sorted list

(iii) 96;

Must be 96 in all cases

[11]

Q10.

(a)

y	x	Index	Result		
			[3]	[2]	[1]
–	5	0	–	–	–
1	2	1	–	–	1
0	1; 2;			0;	
1	0; 3;		1;		

Mark for x=1
Mark for index=2
Mark for y=0 AND Result[2]=0
Mark for x=0
Mark for index=3
Mark for y=1 AND Result[3]=1
Ignore other cells

6

(b) Convert an integer into its binary equivalent;

1

[7]

Q11.

(a) 11, 17, 9,21,15,23;

(2 if all right, 1 if 4 of 6)
If > misinterpreted, follow through for 1 mark

2

(b) A bubble sort;

1

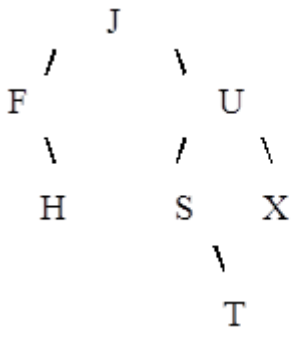
(c) To detect when all the numbers have been sorted
Efficiency (to stop procedure repeating unnecessarily);
R to detect when numbers have switched

1

[4]

Q12.

(a)



J 1 mark

F and U 1 mark

H 1 mark

S, X, T 1 mark

A mirror image (this time)

4

(b) ‘T’ 4; ‘U’;

‘T’ 5; ‘S’;

‘T’ 7; ‘T’;

No penalty if candidate gets ‘item’ wrong
Ignore ‘item’ column

6

[10]

Q13.

(a) A group of bits representing a single character / (usually) 8 bits;

1

(b) (i) A

1

(ii) 9

1

(c)

Code	Number	
–	0	given
55	7	
51	73	
50	732	
49	7321	

1 mark for each correct entry

6

[9]

Q14.

(a) (i) VAR/CONST/TYPE/DIM/FUNCTION/PROCEDURE/LABEL
Or similar, name and type;; keyword and name;;

2

(ii) Eg x:=5 / y ← y – 1

1

(iii) Example of IF / CASE / SWITCH statement

(1 mark for keyword, 1 mark for selection criteria)

2

(b) (i)

	s		s	
	D	[5]	M	[5]
	K	[4]	T	[4]
	C	[3]	C	[3]
	T	[2]	K	[2]
	M	[1]	D	[1]

1 mark for each correct entry

10

(ii) Algorithm: reverse content of array
R re-arrange

1

[16]

Q15.

(a) Optical Mark Recognition/Reading. (Not Optical mark reader)

1

(b) Extra digit added to the transaction code (1)
To detect if data has been corrupted (1)

2

(c) (i) Unique field of a record/filed used to identify record

1

(ii) Transaction code

1

(iii) Not indexed sequential / Serial or Sequential (1)
Because all the records have to be examined (1)

Or

Direct access based on a hash code of the chosen numbers(1)
Only a few records will need to be checked (when collisions occurred)(1)

Max 2

(iv) Random or direct access(1)
Record can be located by simple transformation of transaction code /hashing technique used/algorithm used to store and retrieve records(1)
Indexed sequential with transaction code as key field(1)
Rapid access via the index is possible to find the necessary record(1)

Max 2

(d) *Any 4 points × 1 each*
Ticket scanned/ Read ticket
Check digit used to check accuracy of scanning Ticket validated, (e.g. not out-of-date, draw not yet made)
Operator informed if ticket does not scan/is invalid
Transaction code sent to central computer
Correct file selected
Ticket's record found/Look up ticket's record/Look up record with given transaction code
Get draw date from transaction record
Get numbers from system (for the correct draw date)
Ticket numbers checked against draw
If a winning ticket prize money determined
Result sent to point of sale machine
Result displayed at point of sale machine

Max 4

[13]

Q16.

(a) Array must be sorted (1), on the field being used as the search key (1)

2

(b) Description must include the following points: Find median record of array **(I)** Compare key field of record at median position with required search key, exit if found (1) If search key lower (i.e. required record in first half), discard second half, else discard first half (1) Repeat process (1) until either found, or no further division possible so record does not exist (1)

5

(c) On each iteration, half the possible matches are eliminated, compared with only one for the linear search (2)
Linear search on average scans n/2 records, compared with $\log_2 n$ which is smaller "Looks at fewer records" without further explanation (1)

2

[9]

Q17.

(a) Causes process to repeat indefinitely

NOT repeats until maintain is TRUE

1

(b) Maintain has two values, TRUE / FALSE (1), => must be Boolean (1) n is used as an array subscript (1) => must be integer (1) or n is used as a loop control and can never be non-integer within the algorithm (1) => integer (1)

NOT numeric - too vague

4

(c) See table for model solution 1 mark each indicated section completed correctly, including follow–through (7x1); additional 1 mark for correctly modifying n downwards in penultimate section If candidates go completely wrong but clearly deserve some credit marks can be awarded on the following criteria, up to a maximum of 2 marks for correct sequence of loop repetitions, including the change from 6 to 5 then 6 - i.e. the column for n, including correct exit 2 marks for correct completion of the sequence of stations, ie the org, dest, start, finish columns

2 marks for correct completion of totalkm column, i.e. correct lookups and totalling 2 marks for correctly executing inner if branches, i.e. setting maintain and resetting totalkm in correct places. Total 8 marks for all-correct trace follow-through marks should be awarded where appropriate

8

(d) 2 marks for diagram, or explanation, showing that the journeys indicated above cover all routes in both directions *marks can be awarded for any reasoned answer (indicating achievement or not) providing it is consistent with the candidate's trace table* Note: the sequence is MK -> SW -> CW -> SW -> TW -> HK -> MK -> QB -> SW etc., which does cover all lines in both directions. Strictly speaking, whether the objective is achieved depends whether journeys to/from MK depot are passenger-carrying / revenue-earning or not. Either interpretation is acceptable - the marks are awarded for the explanation.

n	org	dest	last	start	finish	totalkm	maintain	Remarks
	0	3	1					
							FALSE	
				MK				
					SW			
						15		
	3							
0								

Given

1								
		4						if ignored
				SW				if ignored
					CW			
						+27 =		

						42		
	4							N<6 so rpt

1 mark

2								
		3						if ignored
				CW				if ignored
					SW			
						+27 = 69		
	3							N<6 so rpt

1 mark

3								
		1						if ignored
				SW				if ignored
					TW			
						+37=106		
	1							N<6 so rpt

1 mark

4								
		5						if ignored
				TW				if ignored
					HK			
						+34=140		
	5							N<6 so rpt
5								

1 mark

		2						if ignored
		0						>140 so if executed
			5					
							True	
				HK				
					MK			
						+12=152		
	0							N<6 so rpt

1 mark

6								TRUE so if executed
5								
						0		
							False	
		2						
								if ignored
				MK				
					QB			
						+28 = 28		
	2							N<6 so rpt

1 mark

6								
		3						if ignored
				QB				if ignored
					SW			
						+43 = 71		

	3							
								N = 6 so stop repeat loop
								end while

1 mark

2

[15]

Q18.

(a) Elephant 4
Deer 1
Bear 5
Rabbit 0
Cow 2

1 mark for rabbit having a pointer of 0
1 mark for the others correct

2

(b) Start = 3
Freestorage = 6

2

(c) Check for free space
Put data into the array at the position indicated by freestorage (animals[6])
Find position where "Monkey" must go in list (between Elephant and Rabbit)
Method for finding position
Alter "elephant" pointer to point to "Monkey"
Make "Monkey" pointer point to "rabbit"
Alter the freestorage pointer to point to next space / to indicate no more free space (0 / –1)

This may be answered as a pseudocode algorithm but any method that makes the steps clear is acceptable
Any 5 × 1

5

[9]

Q19.

(a) See trace table

10

(b) Insertion sort

1

(c) Time taken (1)
To move many items / to make space for one insertion.(1)

2

[13]

Trace table

Comment	count	rp	max	cp	temp	numbers			
						1	2	3	
Global values on call			3			13	25	24	
rp:=1		1							1 mark for assigning and incrementing rp and assigning cp
repeat									
rp:=rp+1		2							
cp:=1				1					
while rp>cp do									
if numbers[rp] > numbers[cp] then									
temp:= numbers[rp]					25				1 mark for temp
for count:=rp to cp+1 step- 1	2								1 mark for count starting from 2 and numbers [2] correct (no need to show count dropping to 1)
numbers[count]:= numbers[count-1]						13			
endfor	1								
numbers[cp]:=temp						25			1 mark for copying temp to numbers[1]
endif									
cp:=cp+1				2					1 mark for incrementing cp (carry forward error)
endwhile									
until rp=max									
rp:=rp+1		3							1 mark for rp incremented
cp:=1				1					And cp assigned 1

While rp>cp do									
if numbers[rp] > numbers[cp] then									
endif									
cp=cp+1				2					1 mark for cp incremented

endwhile									
if numbers[rp] > numbers[cp] then									
temp:= numbers[rp]					24				1 mark for numbers[3] copied to temp
for count:=rp to cp+1 step-1	3								1 mark for count starting from 3 and numbers [3]
numbers[count]:= numbers[count-1]								13	correct (no need to show count dropping to 2)
endfor	2								
numbers[cp]:=temp							24		1 mark for numbers[2] assigned 24
endif									And cp incremented
endif									
cp:=cp+1				3					
endwhile									
until rp=max									

Q20.

(a) Information passed to / from a function or procedure to define the values it is to use - e.g. in calling OPENSCREEN, the values “Admin Computer” and 10 [actual or real parameters] are to be used as the values of COMPUTERNAME and CHANNEL [formal parameters or placeholders]

3

(b) Enables same function to be used in a number of contexts, enables “black-box” programming, or enables different programmers to work on various modules

Accept: saves memory by not using global variables, or prevents inadvertent modification of variables in a procedure

2

(c) As a series of contiguous / consecutive memory locations

1

(d) Extract a character from the MSG array, at the position indicated by COUNT
Call the SENDCHARACTER function, passing it CH and other data Increment COUNT and COL
Repeat the process as long as CH does not have the value 13 [4 important points are: repetition, what condition determines whether to repeat (examine current value of CH variable), if condition is true what happens (execute block to endwhile), effect of instructions in loop (next character in sequence sent to other computer and screen coordinates adjusted)]

4

(e) Prints one of the specified messages, depending on the value of ERR

2

(f) Would be too long for the MSG array and so might overwrite other data or code
Accept: data truncated, interpreter produces runtime error (array bounds exceeded), compiler error causes program build to abort

1

[13]

Q21.

(a) Structure shown correctly with data in consecutive locations, and start and stop pointers

Must have data in boxes or a label

3

(b) Procedure insert

```
{Check for overflow}
If (Start pointer=1 and) stop pointer = max size
```

2 marks

```
(or Start pointer = stop pointer+ 1) then
     Report queue full
     End
Endif
```

1 mark

Parts in brackets refer to circular queue not essential. 2 marks for checking. 1 mark for an attempt to check. 1 mark for stopping if full

```
        {Insertion of data}
        (If stop pointer = maxsize then
               Stop pointer:= 1
        Else)
               Stop pointer:=stop pointer+ 1
        (Endif)
        queue(stop pointer):=data
endproc
```

1 for changing stop pointer. 1 for inserting data
2 marks

5

[8]

Q22.

See trace table below. Sections corresponding to marks are shaded.

1 mark for newstring, message and procedure call correct. 1 mark for x and piece correct. 1 mark for outstring correct. 1 mark for changing x and a. 1 mark for tracing the second call to docharacter. 1 for section correct. 1 mark for third call correct. 1 mark for endprocs all correctly traced. 1 mark for outputs correct

		↓						↓	Trace table				↓						
	New string	message	a	Out string	x	piece	x>0?	a	Out string	x	piece	x>0?	a	Out string	x	piece	x>0?	output	marks
Input message		CAT																	
New string: = “”	‘’’																		
Output message																		CAT	
Docharacter (message, new string)			CAT	‘’’															
x :=1en(a)					3														
Piece := Right$(a,1)						T													
Outstring:=outstring+piece				T															
x := x-1					2														
If x>0 then							true												
a= Left$(a,x)			CA																
Docharacter (message, new string)								CA	T										
x :=1en(a)										2									
Piece :=Right$(a, 1)											A								
Outstring: = outstring+piece									TA										
x :=x-1										1									
If x>0 then												true							
a= Left$ (a, x)								C											
Docharacter (message, new string)													C	TA					
x :=1en(a)															1				
Piece :=Right$(a, l)																C			
Outstring: = outstring+piece														TAC					
x :=x-1															0				
If x>0 then																	false		
Endif																			
endproc								C	TAC										
Endif																			
endproc			C	TAC															
Endif																			
endproc	TAC	C																	
Output new string																		TAC	

Note that there is no need to trace a and outstring separately in each cell as there are parameters passed by reference. If they are included in the trace then recursive calls must be shown. X and piece must be shown as additional columns for each cell.

[9]

Q23.

(a) (i) 00000010
AND

1 for mask. 1 for AND

2

(ii) 10000000
OR

1 for mask. 1 for OR

2

(b)

```
Repeat
     Test active
     If active = true then
       Test external motion sensor
       If external motion sensor = true then
            security light: = on
       else
            security light: = off
       endif
       Test internal motion sensor
       Test window contact
       Test door contact
       If ((internal motion sensor = true)or (window contact =
false)or
       (door contact = false)) then
            Alarm: =on
       else
       Alarm off
       endif
     endif
until set = false
```

1 mark for suitable loop including termination.
1 mark for testing set.
1 mark for If correctly used with endif.
1 mark for testing external sensor and handling light.
1 mark for testing all three sensors (and alarm on.)
2 marks for a single if construct, just 1 if there are 3 separate ifs

7

[11]

Q24.

(a)

Low	High	Middle	Found
		5	
6		*8*	
	7	*6*	
7		*7*	true

1 mark for each entry above (as far as first incorrect entry)
Mark row by row

Max 7

(b) Binary search/chop
Iterative (no synonyms)
(Specific searches not on AS syllabus - search sufficient for mark)

1

[8]

Q25.

Repeat
S → Q
Until Q empty
Queue emptied to a stack
Elements taken from front of queue and placed / pushed on stack

2 marks

Repeat
Q → S
Until S empty
Stack emptied to a queue
Elements popped/taken from top of stack placed in queue

2 marks

Or suitable diagram

1 mark

[4]

Examiner reports

Q1.

(a) It was pleasing to see the number of candidates who scored well on this question.

(b) Fewer candidates were able to give a correct purpose for the algorithm even when they completed the table correctly. Many described the processing of the values rather than trying to identify the purpose of the algorithm.

Q2.

Many candidates were able to complete the pointer column in the table correctly but could not adequately describe a suitable data structure for this table. Most stated array or linked list but very few noticed that the columns required different data types and therefore an **array of records** or a **linked list of records** or several arrays were required for full marks. In part (iii) many candidates could write a suitable algorithm, some even provided very elegant, recursive, solutions. However, a few candidates reassigned values to the head pointer as they worked their way through the list. This is not appropriate. Others printed the pointer rather than the name. A possible solution gaining full marks would be:

```
ListPointer ← HeadPointer
While ListPointer <> -1 Do
  Print ListArray[ListPointer].Name
  ListPointer ← ListArray[ListPointer].Pointer
EndWhile
```

In part (iv) very few candidates seemed to remember that the list they were working with in the question was that of runnable processes, and that only one process at any one time can be running, so a list of running processes would not be sensible. This leaves suspended processes (waiting for a resource) and inactive jobs (waiting to be admitted to the system).

Q3.

(a) It was pleasing to see the number of candidates who scored well on this question. Most candidates were able to fill in the Index column correctly. Fewer obtained the correct entries for the Result column.

(b) Fewer candidates were able to give a correct purpose for the algorithm even when they completed the table correctly.

Q4.

(a) Most candidates could correctly state that recursively defined means that a procedure is defined in terms of itself or that it calls itself. Some candidates failed to gain marks because they could not express this clearly enough. A common misconception was that the procedure was in a loop.

(b) Many candidates managed to gain full marks for completing the trace table:

E	L	H	M	List[M]	Printed Output
6502	1	11 ;	6	5789 ;	

6502	7	11 ;	9	8407 ;	
6502	7	8 ;	7	6502 ;	
					True;

A common mistake was to have False as printed output in the first 2 rows until it changed to True.

Some candidates who correctly completed the trace table could not see that the process was a binary search and some who did not complete the table correctly did manage to identify the process correctly.

Candidates should be aware of the need to fill in trace tables carefully, showing how values change chronologically. This may mean leaving some cells empty if no values are assigned to variables initially.

Q5.

(a) Most candidates gave ASCII and many were able to state either EBCDIC or Unicode. A common error was to give BCD.

(b) It was pleasing to see that few candidates were unable to get some of the trace table correct. The Index entries were nearly always correct. Most candidates also obtained the correct entries for X. Result [2] was often correct but Result [3] was often incorrect. Unfortunately few candidates were able to express the purpose of the algorithm.

Q6.

Those candidates who clearly understood recursion scored high marks in this question, but a worrying number of candidates could not explain that a procedure is recursively defined when it is defined in terms of itself. A stack is needed so that register values such as return address and parameter values can be saved and can be returned to in the correct order. Most candidates managed to complete the trace table correctly but many did not give the correct printed output. Many candidates did not provide the correct number of digits, or in reverse order. A large majority wrongly thought that procedure B described a binary search. Interestingly, some of the candidates who got the printed output wrong still stated the correct purpose of the procedure: converting the denary number provided as parameter into binary.

Q7.

In part (a), a large number of candidates failed to differentiate correctly between a selection statement and iteration.

For (b), very few candidates correctly identified the given subroutines as 2 functions and one procedure. Even though the description of the subroutines in the question stem should have made candidates realise that both *copy* and *concat* were the same type of subroutine, many seemed to hedge their bets and opted for one of each type. Many other candidates got the choice exactly the wrong way round.

In (c), the response to the dry run was much more promising, suggesting that candidates are getting more opportunity to practice this type of skill. A large number of candidates were able to follow the algorithm with only the S2 value causing any problems.

Q8.

Parts (a) (i) and (a) (iii) were answered correctly by most candidates as Mon and False respectively, (not [Mon]). However, in (a) (ii) [Head(Days)} is the list [Sun] and the tail of this list is the empty set [ ]. Had these data been stored in a one dimensional array, instead of in a list, each element could have been accessed directly using the subscript, rather than having to be found through a sequential search or using the Head / Tail functions defined.

Q9.

(a) There was a definite improvement in the dry running of a piece of pseudo-code. Many candidates were able to correctly complete the table to gain the 7 marks. Some candidates instantly recognised the bubble sort and decided to skip the dry run and write the final values on the bottom line (the mark scheme penalised this severely). Others assumed that it was a complete bubble sort and simply wrote down the list of numbers in ascending order. However, a significant number of candidates still did not seem to be prepared for dry-running an algorithm. The grid was left blank or was completely filled in with totally irrelevant numbers by such candidates.

(b) This was poorly answered with many candidates stating that the program stopped or that 'nothing will happen'. A few candidates gained credit by stating that control will pass to the instruction after EndWhile.

(c) Those candidates who answered (a) correctly obtained at least 2 marks for this part. Even good candidates often did not spot that another run through the algorithm would put 81 into the ninth element of List.

Q10.

Candidates find following pseudo-code very difficult and very few were able to complete the table correctly, though many got some parts correct – usually the next values of x and Index. Of those who did complete the table correctly only a few spotted that the initial value of x (i.e. 5) had been converted into binary.

Q11.

This was on the bubble sort algorithm. Space was left here for candidates to work through the algorithm if they wished. Many candidates could identify the algorithm as a bubble sort, but failed to work through it correctly to gain marks in part (a). Many could see what the flag did, but could not explain that its purpose in the algorithm was to detect when all the numbers had been sorted so that the procedure was not repeated unnecessarily.

Q12.

(a) The tree was generally well created. A few candidates produced mirror images, which were accepted this time. However, candidates need to be aware that a binary search tree stores lower values in the left sub-tree and higher values in the right sub-tree. A few candidates produced a balanced binary tree with each node having two sub nodes, which was not correct.

(b) Only the better candidates seemed to score full marks here. Dry-running an algorithm does not seem to have received sufficient attention in the preparation of a number of candidates. The fact that 'Item' did not change its value seemed to be noticed by only a few candidates. Candidates did not appear to appreciate that the algorithm was using the binary search tree, which the candidates had drawn in part

(a).

Q13.

(a) Most candidates knew that a byte was 8 bits. Some quoted January's paper again and (wrongly) stated that a kilobyte was 1000 bytes.

(b) The majority of candidates correctly converted the bit patterns to 'A' and '9', but a significant minority quoted the ranges given in the question.

(c) Most candidates who attempted the dry run correctly calculated the codes, but only the better candidates noticed that Number grew in size with each digit processed.

Q14.

Part (a) was always attempted, but many candidates confused a declaration with an assignment. Some candidates tried to answer in general terms rather than with examples asked for. In part (b) some candidates got full marks and demonstrated clearly their ability to follow through an algorithm. Many candidates did not seem to understand how to dry run algorithms; or this was not taught in some centres. Many candidates guessed wrongly that it was a sort of algorithm and simply wrote down the letters in alphabetical order into the array. Some candidates thought that the algorithm reversed the letters and then back again. Most candidates who correctly followed through the algorithm then correctly recognised that the order of the letters in the array was reversed.

Q15.

Many candidates correctly identified the method as Optical Mark Recognition. Candidates who answered "Optical Mark Reader" referred to the device, not the method, and so were not rewarded. Some candidates answered incorrectly that a check digit checked that the chosen numbers were unique. An answer that stated that a check digit is an extra digit added to the transaction code obtained a mark. The mark scheme allocated a second mark to an answer that stated that the check digit was used to detect if data was corrupted. The emphasis was on error detection, hence the non-specificity in stating what was being corrupted. Several candidates went into detail and described how the check digit is calculated using a modulo- II method. This was really answering more than was required for one mark, as this response described both what it is and how it is generated.

The majority of candidates correctly defined the term 'primary key'. However, several of these candidates then incorrectly identified "Point of Sale Identification Code" as the primary key for the transaction records instead of "Transaction Code". These candidates appeared to lack an understanding of the term 'transaction'.

In part (c) (iii) the better candidates realised that all the records have to be examined to find the ticket(s) with the winning numbers. These candidates then showed good knowledge of file organisations by answering "serial". Weaker candidates seemed to be unfamiliar with the term 'file Organisation', responding with answers such as "use a database or spreadsheet". Other successful candidates answered that if each transaction's chosen numbers were hashed, then the generated address could be used to store each transaction's details. After the draw, the winning numbers could be hashed to locate the transaction(s) that had won.

In part (d) many candidates did not appreciate the detail of the processing steps that have to take place if the computing system is to check if the ticket is a winning ticket. These candidates showed a distinct lack of insight into the operation of the computer. Their answers were superficial and from the perspective of the ticket holder not the computer system. The better answers were on a different analytical plain. These answers used

appropriate technical terms e.g. ticked “scanned”, “check digit” used to check accuracy of “scanning”, “transaction code” sent to “central computer”, correct “file” located/ “record” with given “transaction code” found or “transaction code” compared with winning “transaction codes”, etc. Compare this with an answer pitched at the level of a ticket holder. “The numbers on the ticket are entered. The computer system checks if these are the winning numbers. The user is informed”. Such an answer gained no marks.

Q16.

Knowledge of sort procedures seems good, but the ability to express it with anything like the precision of language expected at A-level is not. In particular, far too many candidates treat file, record and field as interchangeable terms, which is unacceptable at this level.

For part (a), nearly all candidates pointed out that an array needed to be sorted for a binary chop search to work, but very few realised that, say, an array with ten fields could be sorted in ten different orders and only one would work - that in which the sort key was the field being searched.

For part (b), most scored reasonably well, although as usual descriptions were full of waffle. Candidates seemed to be trying to paraphrase an algorithm they had been taught - while formal algorithms are not in the syllabus for this paper, if a candidate wants to present an answer in pseudocode or even flowchart form it will receive full credit. A common mistake was to say that the search key was compared to “the middle record” (rarely the more accurate “key field of the median record”) and one half or the other discarded, ignoring the possibility that it might be matched enabling the search to end. Many candidates failed to indicate how the search could indicate failure if the desired key did not exist in the array. Another mistake is to describe the process by describing the progress of a search of specimen data - rarely adequate because it misses many of the things that can happen with different data, and does not bring out the iterative nature of the process because the list is so short.

In part (c), it was necessary to indicate something about the workings of both techniques to gain both marks (preferably referring back to part (b)) - the bald (and common) “it looks at fewer records” shows little understanding. Not many appreciated the significance of the word “normally” - a linear search is actually much faster than a binary chop in finding a key such as “aardvark”!

Q17.

This question divided the candidates into two groups, those who could answer it and those who could not - an encouraging number scored nearly full marks, but an alarming number scored close to zero.

For part (a), although the while (TRUE) construction is standard terminology for an infinite loop few realised it, instead many tried wrongly to relate it to the state of the maintain variable, presumably because it was the only Boolean in sight.

Most candidates understood the data types for (b) - not many gave the correct reason for n being an integer (it is used as an array subscript), but many commented that it was only required to take the values 0 to 6 and so didn’t need to be anything else, perfectly valid reasoning. Many students could not even attempt the trace table, but many perfect attempts were seen, conversely some students made marking difficult by making elementary arithmetic errors early on which had to be laboriously followed through to give appropriate credit.

For the last part, which should have demonstrated ability to interpret the table in terms of the original problem, many elaborate flights of logic appeared, which were rewarded

provided they were plausible and argued from the student's trace table.

Q18.

This question was answered well by most candidates although there were some who had clearly never heard of a linked list

In part (a) most candidates were able to give the correct pointer values. The commonest error was to give positions in the alphabetical list.

In part (b) the values of start and free storage were usually correct.

Part (c) gave candidates the opportunity to explain what had to be done to add an item to the list, without requiring re-call of an algorithm. Most candidates were able to explain some of the steps needed and good candidates gained all the marks. Weak candidates often placed the new item in the freestorage pointer instead of the array and forgot to increment freestorage. The method used to find the position of "monkey" in the list was rarely described in detail. Some candidates wanted to sort the array when the new item was added.

Q19.

In part (a) the majority of candidates managed to trace the algorithm and many obtained all ten marks. Some of the trace tables were very well laid out and easy to follow. Candidates should not cross out values when they change during the trace. Marks are obtained for showing the changes and crossing out values makes it difficult to read what was there. Some candidates are unwilling to use a new line for each stage of the trace, filling in the next available space under the appropriate heading. The resulting table gives no indication of sequence. Few candidates made any use of the comment column. A few candidates decided that the array contents would be sorted into ascending order and tried to make this happen with no regard for the algorithm. There were also candidates with no idea of how to trace the changes in the array elements.

In part (b) while many candidates correctly identified the insertion sort, a large minority gave bubble sort. This is surprising as the bubble sort is not on the syllabus.

In part (c) some candidates gained a mark for saying that the sort would take too much time but few understood that the method is inefficient because of the very large number of comparisons and exchanges needed.

Q20.

This question also asked candidates to relate theory to practical examples to illustrate their understanding of it rather than simply recalling, and in some respects fell down badly. In (a) and (b) the able candidates could describe what the term parameters means and quote examples of where they were mentioned in the pseudocode, but only the very ablest described their use: a good answer would have been something like "parameters are values passed into subprograms, such as the variable values col and row being passed into the SendCharacter procedure, where they take the place of the variables (formal parameters) x and y". Several good candidates explained the difference between passing by value and by reference, although this was not necessary to gain the marks. Part (c) elicited hardly any correct answers, although the fact that array elements are stored in consecutive locations in memory is virtually the definition of an array.

Part (d) attracted varied replies, although many did not really say very much about the way the loop works, the point of the question: many answers actually described a "repeat until" construction which is quite different in important respects. The important points

needed for the four marks were condition test and selection, what happened in the two possible cases, and repetition. The commonest failing was no mention of what happened when the condition failed (ie character 13 entered): " the loop stops" was common, but " a jump to the instruction following endwhile" wasn't. Part (e) was well done. The last part gave a mark to most candidates who gave it any thought at all, although "it would crash" without explanation was inadequate. The answer is actually dependent on the language of the program: most compilers will generate code that would produce an array-bounds execution error, although some (notoriously C or C++), would cheerfully allow the last ten characters to overflow the allotted memory and overwrite some other, potentially vital, data, causing a possible crash later.

Q21.

This question deals with a standard data structure that should have been familiar to all candidates. A clear, labelled diagram was required for part (a). A collection of empty boxes, linked by arrows, with no labels scored zero. Linear and circular queues were both acceptable.

In part (b) candidates tended to score very well or very badly. Many candidates were obviously repeating an algorithm they had learned but others seemed to be inventing the algorithm in the examination. Most attempted to check for a full queue although not always in a sensible way. Some candidates failed to stop if the queue was full but proceeded to insert the data regardless. A fairly common approach was to make space by deleting an item. It was common to see assignment of data to the pointer rather than the queue location. Assignment was sometimes written the wrong way round so that the new data would be overwritten with the contents of the empty queue location or the pointer.

The presentation of the algorithm often left a great deal to be desired. If candidates make a large number of alterations then it is in their best interests to write out the final version neatly. Indenting correctly makes the algorithm more readable. Where the candidate uses identifiers the purpose of these should be explained.

Q22.

The majority of candidates had problems tracing this recursive algorithm. Many failed to trace the main program at all. Very few candidates indicated what each line of the trace showed making it difficult to give credit for success in some sections.

Candidates should be encouraged to show the trace as a table rather than as a rewritten algorithm with values included. In this algorithm the identifiers piece and x are local to the procedure and therefore separate instances are needed for each recursive call. The parameters a and outstring are passed by reference so there is no need to create extra columns for these as the identifiers message and newstring will be used throughout. Candidates who chose to trace a and outstring were given credit but were expected to be consistent in their use of this approach.

Most candidates scored the marks available for the first procedure call and for the correct outputs.

Q23.

A minority of weaker candidates made no attempt to answer this question. In part (a) the general principles of masking to read or set individual bits without changing others seemed not to be known by many candidates. Even when the masks were correct the logical operators were sometimes wrong.

Masks were not actually required for the algorithm in part (b), credit was given for simply

stating the bits to be tested and the action needed. Where masks were given they were credited. Very few candidates bothered to check that the system was active. The vast majority neglected to provide any kind of loop to allow for continuous monitoring. Another common error was to use an ELSE clause in a way that resulted in testing the internal motion sensor and door and window contacts only when the security light had been activated. Others required both the door and window contacts to be broken before activating the alarm. Credit was given for appropriate use of a loop and IF THEN (ELSE) ENDIF constructs within the algorithm.

Answers tended to suffer where candidates failed to indent correctly. Where identifiers are used they should be explained. Many candidates might have gained more marks if they had added comments to explain what they were trying to do.

Q24.

Those candidates who read this question carefully gained full marks, and most of those were then able to identify the routine as a binary search routine. Candidates were given credit for simply saying “a search routine” as particular types of search routine are not specified on the AS syllabus. Although the function *Int* was explained, many candidates gave the first middle value as 5.5 or rounded up to 6. A very common error was to confuse the subscripts with the actual data. Surprisingly, many calculated the first three values correctly, and then made an error on the fourth.

Q25.

Many candidates grasped that the queue should be emptied into the stack, element by element, removing from the front of the queue and adding to the top of the stack. Candidates who answered by diagram lost marks if they did not clearly indicate that removal was from the front of the queue and insertion was at the top of the stack. Candidates then went on to state that the stack was emptied into the queue by popping elements in turn from the top of the stack. However, several candidates lost a mark by failing to reference the queue as the destination for the popped elements.