The aim of this book is to provide an accessible text for students, covering each of the elements in the AQA 8520 Computer Science GCSE (9-1) specification. It can be used both as a course text and as a revision guide for students nearing the end of their course. It is divided into eight sections, each broken down into manageable chapters of roughly one lesson.

Sections 1, 2A and 2B of the textbook cover algorithms and programming concepts with a theoretical approach to provide students with experience of writing, tracing and debugging pseudocode solutions without the aid of a computer. These sections would complement practical programming experience.

Each section contains in-text questions and practice exercises, which can be set as homework. Answers to all these are available to teachers only, in a free Teachers' Supplement, which can be ordered from our website www.pgonline.co.uk

About the authors
Susan Robson worked for International Computers Ltd after graduating from Manchester University with a degree in Computer Science. She spent the following 12 years in technical pre-sales for ECI Telecom, before moving into teaching. As a Head of Computer Science, she gained years of experience teaching GCSE and A Level Computing and has written successful textbooks and teaching materials. She is currently an Assistant Head at Our Lady's Abingdon.

Pat Heathcote is a well-known and successful author of Computing textbooks. She has spent many years as a teacher of A Level Computing courses and has significant examining experience. She has also worked as a programmer and systems analyst, and was Managing Director of Payne-Gallway Publishers until 2005.
Acknowledgements

The answers in the Teacher’s Supplement are the sole responsibility of the authors and have neither been provided nor approved by the examination boards.

We would also like to thank the following for permission to reproduce copyright photographs:

Server Room © Google/Connie Zhou

How Secure is My Password screenshot © RoboForm, Siber Systems, Inc

PayPal screenshot © PayPal Inc

Other photographic images © Shutterstock

Cover picture © ‘Dawn Vista’ 2015
Screenprint, 93cm x 91cm
Reproduced with the kind permission of Hetty Haxworth
www.hetty-haxworth.co.uk

Cover artwork, graphics and typesetting by PG Online Ltd

First edition 2016

A catalogue entry for this book is available from the British Library

ISBN: 978-1-910523-09-4
Copyright © S. Robson and P.M. Heathcote 2016
All rights reserved
No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the prior written permission of the copyright owner.

Printed and bound in Great Britain by Lightning Source Inc., Milton Keynes
Preface

This is a brand new book from two popular and experienced authors. Aimed at GCSE students, it provides detailed coverage of all the topics covered in the new AQA 8520 Computer Science specification, written and presented in a way that is accessible to teenagers. It can be used as a course text and as a revision guide for students nearing the end of their course.

It is divided into eight sections covering every element of the specification. Sections 1, 2A and 2B of the textbook cover algorithms and programming concepts with a theoretical approach to provide students with experience of writing, tracing and debugging pseudocode solutions without the aid of a computer. These sections would complement practical programming experience.

Each section contains in-text questions and practice exercises. Answers to all these are available to teachers only in a free Teachers' Pack which can be ordered from our website www.pgonline.co.uk.

Approval message from AQA

This textbook has been approved by AQA for use with our qualification. This means that we have checked that it broadly covers the specification and we are satisfied with the overall quality. Full details of our approval process can be found on our website.

We approve textbooks because we know how important it is for teachers and students to have the right resources to support their teaching and learning. However, the publisher is ultimately responsible for the editorial control and quality of this book.

Please note that when teaching the GCSE Computer Science course, you must refer to AQA's specification as your definitive source of information. While this book has been written to match the specification, it cannot provide complete coverage of every aspect of the course.

A wide range of other useful resources can be found on the relevant subject pages of our website: www.aqa.org.uk.
# Contents

## Section 1
**Fundamentals of algorithms**

<table>
<thead>
<tr>
<th>Section 1.1 Algorithms, decomposition and abstraction</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 Developing algorithms using flowcharts</td>
<td>6</td>
</tr>
<tr>
<td>1.3 Developing algorithms using pseudocode</td>
<td>9</td>
</tr>
<tr>
<td>1.4 Searching algorithms</td>
<td>13</td>
</tr>
<tr>
<td>1.5 Sorting algorithms</td>
<td>15</td>
</tr>
</tbody>
</table>

## Section 2A
**Programming basics**

<table>
<thead>
<tr>
<th>Section 2A.1 Data types and operations</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A.2 Sequence and selection</td>
<td>30</td>
</tr>
<tr>
<td>2A.3 Iteration</td>
<td>33</td>
</tr>
<tr>
<td>2A.4 Arrays</td>
<td>36</td>
</tr>
<tr>
<td>2A.5 Records and files</td>
<td>39</td>
</tr>
</tbody>
</table>

## Section 2B
**Programming techniques**

<table>
<thead>
<tr>
<th>Section 2B.1 Procedures and functions</th>
<th>47</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B.2 Validation and authentication</td>
<td>52</td>
</tr>
<tr>
<td>2B.3 Determining the purpose of algorithms</td>
<td>54</td>
</tr>
<tr>
<td>2B.4 Errors and testing</td>
<td>57</td>
</tr>
<tr>
<td>2B.5 Classification of programming languages</td>
<td>62</td>
</tr>
</tbody>
</table>

## Section 3
**Data representation**

<table>
<thead>
<tr>
<th>Section 3.1 Storage units and binary numbers</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2 Binary arithmetic and hexadecimal</td>
<td>71</td>
</tr>
<tr>
<td>3.3 ASCII and Unicode</td>
<td>75</td>
</tr>
<tr>
<td>3.4 Images</td>
<td>76</td>
</tr>
<tr>
<td>3.5 Sound</td>
<td>79</td>
</tr>
<tr>
<td>3.6 Compression</td>
<td>80</td>
</tr>
</tbody>
</table>
Section 4
Computer systems

Section 4.1 Boolean logic

Section 4.2 Application and system software

Section 4.3 Systems architecture

Section 4.4 The CPU and Fetch-Execute cycle

Section 4.5 Memory

Section 4.6 Secondary storage

Section 5
Fundamentals of computer networks

Section 5.1 Wired and wireless networks

Section 5.2 Network topologies and transmission

Section 5.3 Network security

Section 5.4 Protocols and layers

Section 6
Fundamentals of cyber security

Section 6.1 Cyber security threats

Section 6.2 Social engineering

Section 6.3 Malicious code

Section 6.4 Detecting and preventing cyber security threats

Section 7
Impacts of digital technology

Section 7.1 Ethical issues

Section 7.2 Digital technology in society

Section 7.3 Legislation and privacy
Section 1 – Fundamentals of algorithms

1.1 Algorithms, decomposition and abstraction ................................................. 2
1.2 Developing algorithms using flowcharts ..................................................... 6
1.3 Developing algorithms using pseudocode .................................................. 9
1.4 Searching algorithms .................................................................................. 13
1.5 Sorting algorithms ..................................................................................... 15

Objectives

• Understand and explain the term algorithm
• Understand and explain the term decomposition
• Understand and explain the term abstraction
• Use a systematic approach to problem solving and algorithm creation using pseudocode and flowcharts
• Use meaningful identifier names and know why it is important to use them
• Determine the purpose of simple algorithms
• Understand that more than one algorithm can be used to solve the same problem
• Compare the efficiency of algorithms, explaining how some algorithms can be more efficient than others in solving the same problem
• Understand and explain how the linear search algorithm works
• Understand and explain how the binary search algorithm works
• Compare and contrast the linear and binary search algorithms
• Understand and explain how the merge sort algorithm works
• Understand and explain how the bubble sort algorithm works
• Compare and contrast merge sort and bubble sort algorithms
1.4 Searching algorithms

Before starting to write algorithms for our own problems, we will look at some well-known algorithms for searching and sorting, which are both very common operations in the real world.

Thousands of software applications, including databases or commercial search engines such as Google, depend on the ability to quickly search through huge amounts of data to find a particular item.

**Q13** Name some other organisations that store huge amounts of data which often need to be searched quickly find a particular item.

We are going to consider two search algorithms in this section. Two of the most common search routines are:

- Linear search
- Binary search

**A linear search**

When the data is unsorted, the only sensible option when searching for a particular item is to start at the beginning and look at every item until you find the one you want. You could be lucky and find the item quite quickly if it’s near the beginning of the list, or you could be unlucky and find it right at the end of the list.

**Q14** If you have a list of 10,000 unsorted names, on average how many items will need to be examined until you find the one you are looking for?

Here is an algorithm for a linear search:

1. found \(\leftarrow\) False
2. Start at the first name
3. REPEAT
4. Examine the current name in the list
5. IF it's the one you are looking for THEN
6. \(\text{found} \leftarrow \text{True}\)
7. ENDIF
9. UNTIL \(\text{found} = \text{True}\) OR reach end of list
9. IF \(\text{found} = \text{True}\) THEN
10. OUTPUT name
11.ELSE
12. OUTPUT "Not found"
13.ENDIF

The algorithm as written is a long way from something you can turn into program code, but it describes how you might go about solving the problem.

**Example 3**

Look at the following list of integers:

\[
\begin{array}{cccccccc}
14 & 2 & 3 & 11 & 1 & 9 & 5 & 8 & 10 & 6 \\
\end{array}
\]

The items you would examine to find the number 5 would be: 14, 2, 3, 11, 1, 9, 5
Q15 Write down the items you would examine to locate data item 7 in the above data list.

A binary search

If the list is sorted, (i.e. in numerical or alphabetical order), you can use a much more efficient algorithm called a binary search. It works by repeatedly dividing in half the portion of the data list that could contain the required data item. This is continued until there is only one item in the list you are examining.

This is the algorithm:

1. found \leftarrow \text{False}
2. \text{REPEAT}
3. Examine the middle data item in the list
4. IF this is the required item THEN
5. \hspace{1cm} found \leftarrow \text{True}
6. ELSE
7. IF required item > middle item THEN
8. \hspace{1cm} discard the first half of the list including middle item
9. ELSE
10. \hspace{1cm} discard the second half of the list including middle item
11. ENDIF
12. ENDIF
13. UNTIL found = \text{True} OR there are no more items in the list

Example 3

Consider the following ordered list of 15 items. We want to find out whether the number 50 is in the list of 10 items.

\begin{center}
\begin{tabular}{cccccccccccc}
15 & 21 & 29 & 32 & 37 & 40 & 42 & 43 & 48 & 50 & 60 & 64 & 77 & 81 & 90 \\
\end{tabular}
\end{center}

\textbf{Stage 1:} The middle term is 43; we can therefore discard all data items less than or equal to 43.

\begin{center}
\begin{tabular}{cccc}
48 & 50 & 60 & 64 & 77 & 81 & 90 \\
\end{tabular}
\end{center}

\textbf{Stage 2:} The middle term is 64, so we can discard all data items greater than or equal to 64.

\begin{center}
\begin{tabular}{ccc}
48 & 50 & 60 \\
\end{tabular}
\end{center}

\textbf{Stage 3:} middle term is 50 – so we have found the data item.

Note that if there are an even number of items in the list, for example 8 items, the fourth, not the fifth, item is taken to be the middle item.

Q16 Suppose we have the following sorted list of 10 items:

\begin{center}
\begin{tabular}{cccccccccccc}
3 & 5 & 6 & 8 & 11 & 12 & 14 & 15 & 17 & 18 \\
\end{tabular}
\end{center}

Which one of the following is the correct sequence of comparisons when used to locate the data item 8?

(i) 12, 6, 8 \hspace{1cm} (ii) 11, 5, 6, 8 \hspace{1cm} (iii) 3, 5, 6, 8 \hspace{1cm} (iv) 11, 6, 5, 8
1.5 Sorting algorithms

In the last sub-section we looked at methods of searching for data. The binary search method required the data to be sorted before the search could take place. There are many algorithms for sorting data and we will look at two of them:

- Bubble sort
- Merge sort

### Bubble sort

A bubble sort works by repeatedly going through the list to be sorted comparing each pair of adjacent elements. If the elements are in the wrong order they are swapped. A short algorithm to do the swapping is:

```
temp ← a
a ← b
b ← temp
```

If \( a = 9 \) and \( b = 6 \), the trace table below shows that the values of \( a \) and \( b \) have been swapped.

<table>
<thead>
<tr>
<th>temp</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>9</td>
</tr>
</tbody>
</table>
Q20 Why could we not just write the two statements below to swap the values?
\[
\begin{align*}
    a & \leftarrow b \\
    b & \leftarrow a
\end{align*}
\]

Example 4: Working through the Bubble sort algorithm

The figure below shows how the items change order in the first pass, as the largest item ‘bubbles’ to the end of the list. Each time an item is larger than the next one, they change places.

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>9</th>
<th>5</th>
<th>4</th>
<th>15</th>
<th>3</th>
<th>8</th>
<th>11</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>9</td>
<td>4</td>
<td>15</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>15</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>15</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>15</td>
<td>8</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>8</td>
<td>15</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>

After the first pass as shown above, the largest item is in the correct place at the end of the list. On the second pass, only the first seven numbers are checked.

<table>
<thead>
<tr>
<th>End of pass 2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>8</th>
<th>9</th>
<th>2</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
</table>

11 and 15 are in the correct place; so only the first 6 numbers are checked.

<table>
<thead>
<tr>
<th>End of pass 3</th>
<th>4</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>2</th>
<th>9</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
</table>

9, 11 and 15 are now in the correct place; so only the first 5 numbers are checked.

<table>
<thead>
<tr>
<th>End of pass 4</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>2</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
</table>

8, 9, 11 and 15 are now in the correct place; so only the first 4 numbers are checked.

<table>
<thead>
<tr>
<th>End of pass 5</th>
<th>3</th>
<th>4</th>
<th>2</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
</table>

5, 8, 9, 11 and 15 are now in the correct place; so only the first 3 numbers are checked.

<table>
<thead>
<tr>
<th>End of pass 6</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
</table>

Finally, the first two numbers are checked and swapped

<table>
<thead>
<tr>
<th>End of pass 7</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>11</th>
<th>15</th>
</tr>
</thead>
</table>

The numbers are now in the correct order, and no further pass is required.
1. Which of the flowcharts below represents a **WHILE...ENDWHILE** loop and which a **REPEAT...UNTIL** loop?

![Flowchart A](image1)

![Flowchart B](image2)

2. An integer 65 can be converted to a character using the statement:

   letter = CODE_TO_CHAR(65)

   The ASCII value corresponding to the letter “A” is the integer 65, so this statement would assign “A” to `letter`.

   (a) Write an algorithm which accepts three integers as input and outputs the corresponding letters as one word.

   (b) What word will be output if the user enters 66, 69 and 68?

3. An algorithm has been written to simulate a race. Each time the space bar is pressed, the position of the player moves up by 1. When the position reaches 100, the player has won. Here is the algorithm.

   constant PlayerKey = " "
   Position ← 0
   REPEAT
       KeyPressed ← USERINPUT
       IF KeyPressed = PlayerKey THEN
           Position ← Position + 1
       ENDIF
   UNTIL Position = 100

   (a) State what is meant by selection and iteration using examples from the algorithm.

   (b) To make the game more interesting, the rules are changed. Each time the spacebar is pressed, the position of the player will now move up by a random number between 1 and 4.

   State two changes that need to be made to include this new rule. Justify each change.
Section 3 – Data representation

3.1 Storage units and binary numbers .................................................. 68
3.2 Binary arithmetic and hexadecimal ................................................. 71
3.3 ASCII and Unicode ................................................................. 75
3.4 Images .................................................................................. 76
3.5 Sound .................................................................................... 79
3.6 Compression ........................................................................... 80

Objectives

• Understand the following number bases: decimal, binary, hexadecimal
• Convert between number bases
• Know that a bit is a fundamental unit of information, and a byte is a group of 8 bits
• Know the names and values of kB, MB, GB, TB
• Be able to perform binary arithmetic and binary shifts
• Describe the ASCII and Unicode character encoding systems and their purposes
• Describe how a bitmap represents an image using pixels and colour depth
• Calculate bitmap image file sizes based on the number of pixels and colour depth
• Convert binary data into a black and white image and vice versa
• Understand that sound must be converted to a digital form for storage in a computer
• Describe the digital representation of sound in terms of sampling rate and sample resolution
• Calculate sound file sizes
• Explain what data compression is
• Understand why data may be compressed and that there are different methods to compress data
• Explain how data can be compressed using Huffman coding
• Be able to interpret Huffman trees
• Be able to calculate the number of bits required to store compressed and uncompressed data
• Explain how data can be compressed using Run Length Encoding (RLE)
Binary shifts

If a binary number is shifted to the left this is equivalent to multiplying the number by 2 for each shift to the left.

For example: If we shift:

![Binary number](image)

**TWO places** to the left we get the binary number:

![Binary number](image)

(NOTE: we fill empty binary positions with 0s as we shift to the left)

The original binary number has a value of 15 (i.e. 8+4+2+1 = 15); the number after shifting two places to the left has the value 60 (i.e. 32+16+8+4 = 60). It is multiplied by 4, or 2².

Shifting binary numbers to the right has the opposite effect i.e. each shift to the right has the effect of dividing by 2. Thus if we shift:

![Binary number](image)

**THREE places** to the right we get the binary number:

![Binary number](image)

The original binary value was 112 (i.e. 64 + 32 + 16 = 112) and the value after shifting three places to the right is 14 (i.e. 8 + 4 + 2 = 14). The number was divided by 8, and becomes 2³.

(NOTE: we fill empty binary positions with 0s as we shift to the right)

Multiplication/division by powers of 2

This gives an easy way to multiply and divide binary numbers by powers of 2, but can come at the expense of accuracy. For example 00000110 shifted right twice to divide by 4 would be 00000001. This is the equivalent of decimal 1, but 6 / 4 = 1.5.

- Shifting right one place divides the number by 2
- Shifting left one place multiplies the number by 2

This is equivalent to shifting a decimal number right or left – for example shifting 12300 right gives 1230, i.e. it divides the number by 10. Shifting left multiplies a decimal number by 10.

Q7 Write down the results after the following shift operations and write down the decimal values before and after the shifts:

(a) The number 11001100 is shifted TWO places to the right
(b) The number 00011001 is shifted TWO places to the left
(c) The number 11001000 is shifted THREE places to the right
(d) The number 00000111 is shifted FOUR places to the left
(e) The number 10000000 is shifted FIVE places to the right
Lossless compression

This is a data encoding method where files are compressed but no data is lost – an essential factor for text and data files. For example, bank records must keep all of the data; you cannot transmit a bank statement and miss out a few zeros because they don’t matter too much!

It could be used to compress data files, for example by “zipping” them using a utility program such as WinZip, before attaching them to an email.

The following table shows different file types and file extensions used for different file formats.

<table>
<thead>
<tr>
<th>Type</th>
<th>File suffix</th>
<th>Compression Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitmap</td>
<td>.bmp</td>
<td>-</td>
<td>Uncompressed still image file</td>
</tr>
<tr>
<td>JPEG</td>
<td>.jpg</td>
<td>Lossy</td>
<td>Good for photographs. Colour depth = 24 bits, RGB, 16.7 million different colours</td>
</tr>
<tr>
<td>Graphic Interchange Format</td>
<td>.gif</td>
<td>Lossless</td>
<td>Colour depth = 8 bits (only 256 colours) Good for images with large areas of solid colour Ideal for web graphics</td>
</tr>
<tr>
<td>MP3</td>
<td>.mp3</td>
<td>Lossy</td>
<td>Audio files: Designed for downloading music from the Internet. In MP3 format you could fit 120 songs on a CD.</td>
</tr>
</tbody>
</table>

Run length encoding

Run length encoding (RLE) is a simple form of lossless data compression in which runs of data (sequences of data all having the same value) are stored using frequency/data pairs. For example, the black and white image below in uncompressed form would occupy 64 bits, with 1 representing white and 0 representing black.

It could be represented as:
11111111 10111111 00000011 00000011 00000000 00000000 00000000 10011001.

Using RLE, the first row of pixels could be represented as 8 1, meaning that there are 8 pixels each having a value of 1. The second row can be represented as 1 1 1 0 6 1 representing 1 white pixel, 1 black pixel and then 6 white pixels.

RLE is not so useful with files that don’t have many runs, and can in fact increase the file size. It is most useful on simple images such as icons that contain many pixels that are the same colour.
Q17 Using RLE, show how the image below would be coded, if black is encoded as 0 and white as 1.

Huffman coding

Huffman coding is a compression technique used to reduce the number of bits used to represent each letter. The more frequently a letter appears in the text, the fewer bits are used to represent it in a text file.

Example 3

Consider the sentence PIPPA ATE A PEPPER. A table showing the frequency of each character, including spaces is created as the first step in building the Huffman tree. For example, there is one “I”, one “R”, and six “P”s in the sentence.

<table>
<thead>
<tr>
<th>Character</th>
<th>I</th>
<th>R</th>
<th>T</th>
<th>A</th>
<th>E</th>
<th>SPACE</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

You will only be required to interpret the tree, not build it. A Huffman tree for this sentence is shown below. It is a binary tree in which characters that occur most frequently are nearer the top and therefore require fewer characters to encode them, as described below.

Using this Huffman tree, the coding for each character is derived from the path taken from the root node to the character. Branching left at a node is coded as 0, branching right is coded as 1.

Thus the character ‘A’ would be represented by the bit pattern 110 because from the top of the tree, you go right, right, left to reach ‘A’. The encoding for ‘T’ would be 010 and for ‘E’, 111.

The total number of bits needed to represent the word “ATE” would be 3 + 3 + 3 = 9. In 7-bit ASCII, the number of bits required would be 3 x 7 = 21, representing a saving of 12 bits in the compressed format, with a 57% reduction in size.

Q18 (a) What would be the coding for the letters P, I, T?

(b) How many bits would these three letters take using the Huffman code?

(c) The sentence PIPPA ATE A PEPPER is represented in a total of 47 bits. How many bits would be required to represent the sentence in ASCII?

(d) How many bits are saved by compressing PIT using Huffman coding?
Index

A
abstraction, 4
actuator, 93
ADC, 79
adware, 130
algorithm, 2, 37
AlphaGo program, 136
ALU, 97
analogue signal, 79
Analogue-to-Digital Converter, 79
AND gate, 87
anti-malware software, 132
API, 93
Applications Program Interface, 93
Arithmetic Logic Unit, 97
arithmetic operations, 97
array, 36
1-dimensional, 37
2-dimensional, 38
artificial intelligence, 136
ASCII, 75
assembler, 63
assembly language, 62
authentication, 118
routines, 53
automatic software updates, 132
automatic updating, 96

B
bandwidth stealing, 114
binary
division, 72
logic, 86
multiplication, 72
shifts, 72
to decimal, 68, 70
to hexadecimal, 73
binary numbers
adding, 71
binary search, 14
biometric methods, 133
bit, 68
bitmap image, 76
blagging, 128
Blu-Ray, 105
Boolean
data type, 30, 86
expression, 31, 86
variable, 86
bootstrap loader, 101
boundary data, 60
bubble sort, 15
bus, 97
topology, 116
byte, 68
bytecode, 64

C
cache memory, 99
Caesar shift cipher, 119
CAPTCHA, 133
CD, 105
CD-R, 105
CD-ROM, 105
CD-RW, 105
ciphertext, 119
clock, 97
speed, 99
cloud storage, 107
colour depth, 77, 78
compiler, 64
compression, 80
lossless, 81
lossy, 80
software, 96
computational thinking, 4
Computer Misuse Act 1990, 144
computer-based implants, 141
concatenation, 29
constant, 26
Control Unit, 97
cookies, 146
Copyright Designs and Patents Act 1988, 144
cores, 100
CPU performance, 99
cyber security, 126

data types, 25
decimal to binary conversion, 69
decomposition, 3
Deep Blue, 136
defensive design, 52
designer baby, 137
device driver, 93
disk defragmentation, 95
DIV, 27
driverless cars, 137
dual-core, 100
DVD, 105

data types, 25
decimal to binary conversion, 69
decomposition, 3
Deep Blue, 136
defensive design, 52
designer baby, 137
device driver, 93
disk defragmentation, 95
DIV, 27
driverless cars, 137
dual-core, 100
DVD, 105

e-mail confirmation, 134
embedded systems, 94
encryption, 119
software, 95
environmental impact, 141
errors
logic, 58
syntax, 57
Ethernet, 116
transmission, 117

e-mail confirmation, 134
embedded systems, 94
encryption, 119
software, 95
environmental impact, 141
errors
logic, 58
syntax, 57
 Ethernet, 116
transmission, 117

F
fetch-execute cycle, 98
file
reading from, 42
writing to, 41
file size, 78, 80
File Transfer Protocol (FTP), 122
firewall, 120
flash memory, 103
float, 26
flowchart symbols, 6
FOR...ENDFOR, 11, 33
format check, 52
frames, 117
FTP, 122
function, 48
G
GB, 68
gigabyte, 68
Go, 136

H
hacking, 146
hardware, 91
hertz, 79
hexadecimal, 73
to binary, 74
to decimal, 74
high-level languages, 62
advantages, 63
HTTP, 122
HTTPS, 122
Huffman coding, 82

I
I/O device management, 93
identifier, 7, 25
IF…THEN…ELSE, 10
images, 76
input statement, 27
Internet, 111
Internet protocol (IP), 121
interpreter, 64
IP address, 112
iteration, 11
nested, 36

J
JavaScript, 64

K
kB, 68
key, 119
kilobyte, 68

L
LAN, 114
layer, 120, 123
application, 123
link, 123
network/Internet, 123
transport, 123
legal issues, 143
length check, 52
linear search, 13
local area network, 114
logic circuits, 88
logic diagram, 86
logic errors, 58
logic gates, 87, 88
logical operations, 97
lossless compression, 81
lossy compression, 80
low-level languages, 62, 63
advantages, 63

M
MAC address, 113
filtering, 120
machine code, 62
magnetic
disk, 102
media, 102
main memory, 99, 100
maintainability, 53
maintenance utilities, 95
malicious code, 130
malware, 130
MB, 68
Media Access Control, 113
megabyte, 68
memory
cards, 103
management, 92
sticks, 103
metadata, 145
misconfigured access rights, 127
mnemonics, 62
MOD, 27

N
network
forensics, 131
hardware, 113
security, 118
topologies, 115
Network Interface Card, 113
nibble, 68
NIC, 113
non-volatile, 101
NOT gate, 87

O
object code, 64
operating system, 91
operation
arithmetic, 27
comparison, 27
string handling, 28
optical
devices, 105
media, 105
OR gate, 88
outdated software, 127
output statement, 27

P
packet, 112
PAN, 116
parameter, 48
password protection, 118
passwords
default, 126
weak, 126
PB, 68
pen drives, 103
penetration testing, 131
black box, 132
white box, 132
personal area network, 116
petabyte, 68
pharming, 129
phishing, 128
pixel, 76
plaintext, 119
presence check, 52
privacy issues, 145
procedure, 47
programming constructs, 9
protocol
   Ethernet, 121
   HTTP, 122
   HTTPS, 122
   IMAP, 122
   SMTP, 122
   TCP/IP, 121
   UDP, 122
protocols, 116, 120

Q
pseudocode, 9
quad-core, 100

R
RAM, 99, 100
random access memory, 100
random number generation, 38
range check, 52
rating culture, 138
read only memory, 101
Read-Only access, 119
Read-Write access, 119
record, 39
removable media, 126
REPEAT…UNTIL, 11, 35
RLE, 81
robust code, 31
ROM, 101
router, 113
run length encoding, 81

S
sample
   rate, 79
   resolution, 79
secondary storage, 100, 101
security management, 94
sequence, 9
shift operations, 97
shouldering, 129
signal amplitude, 79
Simple Mail Transfer Protocol (SMTP), 122
smart meters, 95
social engineering, 128
social networking, 137
software, 91
   application, 91
   system, 91
Solid State Drives, 103
sorting algorithms, 15
sound, 79
source code, 64
spyware, 130
SSD, 103
star topology, 115
storage, 101
string handling operations, 28
structured programming, 51
subroutine, 47
   advantages of, 51
switch, 113
syntax errors, 57
system bus, 98

T
TB, 68
terabyte, 68
test plan, 59
testing, 57, 59
text file, 41
topology
   bus, 116
   network, 115
   star, 115
trace table, 54, 61
transmission media, 117
Trojan, 131
truth table, 87
type check, 52

U
Uber, 138
UDP, 122
unicode, 76
unpatched software, 127
user access levels, 117
utility software, 95

V
validation, 52
variable
   declaration, 25
   global, 49
   local, 49
variables, 25
verification, 52
virus, 130
volatility, 101
Von Neumann architecture, 97
vulnerabilities, 131

W
WAN, 111
WAP, 121
wearable technologies, 141
web browser, 111
WHILE…ENDWHILE, 11, 34
wide area network, 111
Wi-Fi, 113, 121
WinZip, 96
wireless networking, 113, 142
The aim of this book is to provide an accessible text for students, covering each of the elements in the AQA 8520 Computer Science GCSE (9-1) specification. It can be used both as a course text and as a revision guide for students nearing the end of their course. It is divided into eight sections, each broken down into manageable chapters of roughly one lesson.

Sections 1, 2A and 2B of the textbook cover algorithms and programming concepts with a theoretical approach to provide students with experience of writing, tracing and debugging pseudocode solutions without the aid of a computer. These sections would complement practical programming experience.

Each section contains in-text questions and practice exercises, which can be set as homework. Answers to all these are available to teachers only, in a free Teachers’ Supplement, which can be ordered from our website www.pgonline.co.uk

About the authors
Susan Robson worked for International Computers Ltd after graduating from Manchester University with a degree in Computer Science. She spent the following 12 years in technical pre-sales for ECI Telecom, before moving into teaching. As a Head of Computer Science, she gained years of experience teaching GCSE and A Level Computing and has written successful textbooks and teaching materials. She is currently an Assistant Head at Our Lady’s Abingdon.

Pat Heathcote is a well-known and successful author of Computing textbooks. She has spent many years as a teacher of A Level Computing courses and has significant examining experience. She has also worked as a programmer and systems analyst, and was Managing Director of Payne-Gallway Publishers until 2005.

Cover picture:
‘Dawn Vista’
Screenprint, 93x91cm
© Hetty Haxworth
www.hetty-haxworth.co.uk

This book has been approved by AQA.