Year 9 Computing
Distance Learning
Quiz and Learn Booklet
Summer 2

Name :

Form :
Contents

Week 1: Binary ........................................................................................................................................ 3
Week 2: Representing text, images and sound ...................................................................................... 9
Week 3: Designing an algorithm ........................................................................................................... 15
Week 4: Searching ................................................................................................................................ 19
Week 5: Sorting ..................................................................................................................................... 25
Week 6: Logical reasoning .................................................................................................................... 29
Week 1: Binary
How computers see the world

There are a number of very common needs for a computer, including the need to store and view data.

Computers use electrical signals that are on or off, so they have to see everything as a series of binary numbers. This data is represented as a sequence of 1s and 0s (on and off). All data that we want a computer to process needs to be converted into this binary format.

What is binary?

Binary is a number system that only uses two digits: 1 and 0. All information that is processed by a computer is in the form of a sequence of 1s and 0s. Therefore, all data that we want a computer to process needs to be converted into binary.
The binary system is known as a ‘base 2’ system. This is because:

- there are only two digits to select from (1 and 0)
- when using the binary system, data is converted using the power of two.

- Converting from binary to denary

- Understanding denary

People use the denary (or decimal) number system in their day-to-day lives. This system has 10 digits that we can use: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

- The value of each place value is calculated by multiplying by 10 (ie by the power of 10). The first few place values look like this:

<table>
<thead>
<tr>
<th>Thousands (1000s)</th>
<th>Hundreds (100s)</th>
<th>Tens (10s)</th>
<th>Units (1s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

- Working out the value of 1024
Converting from binary to denary

To convert a binary number to denary, start by writing out the binary place values. In denary, the place values are 1, 10, 100, 1000, etc – each place value is 10 times bigger than the last. In binary, each place value is 2 times bigger than the last (i.e. increased by the power of 2). The first few binary place values look like this:

<table>
<thead>
<tr>
<th></th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1×128 +</td>
<td>0×64 +</td>
<td>1×32 +</td>
<td>0×16 +</td>
<td>1×8 +</td>
<td>0×4 +</td>
<td>0×2 +</td>
<td>0×1 +</td>
<td></td>
</tr>
<tr>
<td>128 +</td>
<td>0 +</td>
<td>32 +</td>
<td>0 +</td>
<td>8 +</td>
<td>0 +</td>
<td>0 +</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

So 1010 1000 in binary is equal to 168 in denary.

Converting from denary to binary

There is a very simple method to convert a denary number into a binary number. Let’s take the number 199.
Start by writing out the first few binary place values (128, 64, 32, 16, 8, 4, 2, 1).

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
</table>

Start at the far left point and say “Can 128 be taken away from 199?”. If it can, do that.

199 – 128 = 71. Because 128 could be taken off, put a 1 in the ‘128’ place value column:

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Now repeat for 64: 71 – 64 = 7

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

And again for 32: 7 – 32 won’t work, so put a 0 in that place value column.

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Try again for 16: 7 – 16 won’t work, so add a 0 to that place value column.

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Next is 8: 7 – 8 won’t work. Add a 0 to the ‘8’ place value column.

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Try again for 4: 7 – 4 = 3, so add a 1 to the ‘4’ place value column.

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Next try 2: 3 – 2 = 1, so add a 1 to the ‘2’ place value column.

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
And finally, \( 1 - 1 = 0 \) – add a 1 to the ‘1’ place value column.

<table>
<thead>
<tr>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

This means that 199 as a binary number is 1100 0111.

Note that binary numbers are usually written in blocks of four, separated by a space (eg 0111 1011). In denary, numbers are often written in blocks of three (eg 6 428 721).

A quick way to check whether your binary number is likely to be correct is by looking at the last digit. If the denary number was odd, this last binary digit should be a 1. If it was an even number this binary digit should be a 0.

**Adding binary**

When two numbers are added together in denary, we take the first number, add the second number to it and get an answer. For example, \( 1 + 2 = 3 \).

When we add two binary numbers together the process is different.

There are four rules that need to be followed when adding two binary numbers. These are:

- \( 0 + 0 = 0 \)
- \( 1 + 0 = 1 \)
- \( 1 + 1 = 10 \) (binary for 2)
- \( 1 + 1 + 1 = 11 \) (binary for 3)

**Overflow errors**

Sometimes, when adding two binary numbers we can end up with an extra digit that doesn’t fit. This is called an **overflow error**.

\[
\begin{array}{c}
0 & 1 \\
+ & 0 & 1 \\
\hline
1 & 0
\end{array}
\]

This sum is fine as the original numbers have two digits, and the result of the sum also has two digits.
This sum has an overflow error. The original numbers had two binary digits, but the answer is three binary digits long.

The effects of an overflow error can vary. It might make the program crash or it might just ignore the extra digit on the left and produce an unexpected result (in this case, $2 + 3 = 01$).
Week 2: Representing text, images and sound

Representing data

All data inside a computer is transmitted as a series of electrical signals that are either on or off. Therefore, in order for a computer to be able to process any kind of data, including text, images and sound, they must be converted into binary form. If the data is not converted into binary – a series of 1s and 0s – the computer will simply not understand it or be able to process it.

Representing text

When any key on a keyboard is pressed, it needs to be converted into a binary number so that it can be processed by the computer and the typed character can appear on the screen.
A code where each number represents a character can be used to convert text into binary. One code we can use for this is called ASCII. The ASCII code takes each character on the keyboard and assigns it a binary number. For example:

- the letter ‘a’ has the binary number 0110 0001 (this is the denary number 97)
- the letter ‘b’ has the binary number 0110 0010 (this is the denary number 98)
- the letter ‘c’ has the binary number 0110 0011 (this is the denary number 99)

Text characters start at denary number 0 in the ASCII code, but this covers special characters including punctuation, the return key and control characters as well as the number keys, capital letters and lower case letters.

ASCII code can only store 128 characters, which is enough for most words in English but not enough for other languages. If you want to use accents in European languages or larger alphabets such as Cyrillic (the Russian alphabet) and Chinese Mandarin then more characters are needed. Therefore another code, called Unicode, was created. This meant that computers could be used by people using different languages.

**Representing images**

Images also need to be converted into binary in order for a computer to process them so that they can be seen on our screen. Digital images are made up of pixels. Each pixel in an image is made up of binary numbers.

If we say that 1 is black (or on) and 0 is white (or off), then a simple black and white picture can be created using binary.

To create the picture, a grid can be set out and the squares coloured (1 – black and 0 – white). But before the grid can be created, the size of the grid needs be known. This data is called metadata and
computers need metadata to know the size of an image. If the metadata for the image to be created is 10x10, this means the picture will be 10 pixels across and 10 pixels down.

This example shows an image created in this way:

Adding colour

The system described so far is fine for black and white images, but most images need to use colours as well. Instead of using just 0 and 1, using four possible numbers will allow an image to use four colours. In binary this can be represented using two bits per pixel:

- 00 – white
- 01 – blue
- 10 – green
- 11 – red

While this is still not a very large range of colours, adding another binary digit will double the number of colours that are available:

- 1 bit per pixel (0 or 1): two possible colours
- 2 bits per pixel (00 to 11): four possible colours
- 3 bits per pixel (000 to 111): eight possible colours
- 4 bits per pixel (0000 – 1111): 16 possible colours
- …
- 16 bits per pixel (0000 0000 0000 0000 – 1111 1111 1111 1111): over 65 000 possible colours

The number of bits used to store each pixel is called the colour depth. Images with more colours need more pixels to store each available colour. This means that images that use lots of colours are stored in larger files.

Image quality
Image quality is affected by the resolution of the image. The resolution of an image is a way of describing how tightly packed the pixels are.

In a low-resolution image, the pixels are larger so fewer are needed to fill the space. This results in images that look blocky or **pixelated**. An image with a high resolution has more pixels, so it looks a lot better when you zoom in or stretch it. The downside of having more pixels is that the file size will be bigger.

**Representing sound**

Sound needs to be converted into binary for computers to be able to process it. To do this, sound is captured - usually by a microphone - and then converted into a digital signal.

An analogue to digital converter will sample a sound wave at regular time intervals. For example, a sound wave like this can be sampled at each time sample point:

![Graph of a sound wave](image)

The samples can then be converted to binary. They will be recorded to the nearest whole number.

<table>
<thead>
<tr>
<th>Time sample</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denary</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Binary</td>
<td>1000</td>
<td>0011</td>
<td>0111</td>
<td>0110</td>
<td>1001</td>
<td>0111</td>
<td>0010</td>
<td>0100</td>
<td>0110</td>
<td>0110</td>
</tr>
</tbody>
</table>

If the time samples are then plotted back onto the same graph, it can be seen that the sound wave now looks different. This is because sampling does not take into account what the sound wave is doing in between each time sample.
This means that the sound loses quality as data has been lost between the time samples. The way to increase the quality and store the sound at a quality closer to the original, is to have more time samples that are closer together. This way, more detail about the sound can be collected, so when it’s converted to digital and back to analogue again it does not lose as much quality.

The frequency at which samples are taken is called the sample rate, and is measured in Hertz (Hz). 1 Hz is one sample per second. Most CD-quality audio is sampled at 44 100 or 48 000 KHz.

**Compression**

**Why compress files?**

Processing power and storage space is very valuable on a computer. To get the best out of both, it can mean that we need to reduce the file size of text, image and audio data in order to transfer it more quickly and so that it takes up less storage space.

In addition, large files take a lot longer to download or upload which leads to web pages, songs and videos that take longer to load and play when using the internet.

Compression addresses these issues.

Any kind of data can be compressed. There are two main types of compression: lossy and lossless.

**Lossy compression**

Lossy compression removes some of a file’s original data in order to reduce the file size. This might mean reducing the numbers of colours in an image or reducing the number of samples in a sound file. This can result in a small loss of quality of an image or sound file.

A popular lossy compression method for images is the JPEG, which is why most images on the internet are JPEG images. A popular lossy compression method for sounds is MP3. *Once a file has been compressed using lossy compression, the discarded data cannot be retrieved again.*

**Lossless compression**
Lossless compression doesn’t reduce the quality of the file at all. No data is lost, so lossless compression allows a file to be recreated exactly as it was when originally created.

There are various algorithms for doing this, usually by looking for patterns in the data that are repeated. Zip files are an example of lossless compression.

The space savings of lossless compression are not as good as they are with lossy compression.
Week 3: Designing an algorithm

Designing an algorithm

An algorithm is a plan, a logical step-by-step process for solving a problem. Algorithms are normally written as a flowchart or in pseudocode.

The key to any problem-solving task is to guide your thought process. The most useful thing to do is keep asking ‘What if we did it this way?’ Exploring different ways of solving a problem can help to find the best way to solve it.

When designing an algorithm, consider if there is more than one way of solving the problem.

When designing an algorithm there are two main areas to look at:

- the big picture - What is the final goal?
- the individual stages – What hurdles need to be overcome on the way to the goal?

Understanding the problem

Before an algorithm can be designed, it is important to check that the problem is completely understood. There are a number of basic things to know in order to really understand the problem:

- What are the inputs into the problem?
- What will be the outputs of the problem?
- In what order do instructions need to be carried out?
- What decisions need to be made in the problem?
- Are any areas of the problem repeated?

Once these basic things are understood, it is time to design the algorithm.

Pseudocode

Most programs are developed using programming languages. These languages have specific syntax that must be used so that the program will run properly. Pseudocode is not a programming language, it is a simple way of describing a set of instructions that does not have to use specific syntax.

Common pseudocode notation

There is no strict set of standard notations for pseudocode, but some of the most widely recognised are:

- INPUT – indicates a user will be inputting something
- OUTPUT – indicates that an output will appear on the screen
- WHILE – a loop (iteration that has a condition at the beginning)
- FOR – a counting loop (iteration)
- REPEAT – UNTIL – a loop (iteration) that has a condition at the end
• IF – THEN – ELSE – a decision (selection) in which a choice is made

• any instructions that occur inside a selection or iteration are usually indented

Using pseudocode

Pseudocode can be used to plan out programs. Planning a program that asks people what the best subject they take is, would look like this in pseudocode:

```
REPEAT
    OUTPUT 'What is the best subject you take?'
    INPUT user inputs the best subject they take
    STORE the user's input in the answer variable
    IF answer = 'Computer Science' THEN
        OUTPUT 'Of course it is!'
    ELSE
        OUTPUT 'Try again!'
    UNTIL answer = 'Computer Science'
```

Flowcharts

A flowchart is a diagram that represents a set of instructions. Flowcharts normally use standard symbols to represent the different types of instructions. These symbols are used to construct the flowchart and show the step-by-step solution to the problem.

Common flowchart symbols
Using flowcharts

Flowcharts can be used to plan out programs. Planning a program that asks people what the best subject they take is, would look like this as a flowchart:
Start

Input
‘Which is the best subject?’

Does answer = ‘Computer Science’?

NO
Output
‘Try again!’

YES
Output
‘Of course it is!’

Stop
Week 4: Searching

Why do we need searching algorithms?

We often need to find one particular item of data amongst many hundreds, thousands, millions or more. For example, you might need to find someone’s phone number on your phone, or a particular business’s address in the UK.

This is why searching algorithms are important. Without them you would have to look at each item of data – each phone number or business address – individually, to see whether it is what you are looking for. In a large set of data, it will take a long time to do this. Instead, a searching algorithm can be used to help find the item of data you are looking for.

Search algorithms prevent you from having to look through lots of data to find the information you are searching for.

There are many different types of searching algorithms. Two of them are serial search and binary search.

Serial search

Searching for a keyword or value is the foundation of many computer programs. The most basic kind of search is a serial search.

Criteria are set up before the search begins. The search then starts with the first item and then moves to each item in turn, until either a match is found or it reaches the end of the data with no match found.

Example

Imagine that you have a database of sales made to customers. You need to deliver the goods that customer number 150 has bought, so need to find their address in the database.
The criterion is set first - **Find the address for customer 150**.

A serial search will begin at customer 1 and will search through each customer in turn until it reaches customer 150. It will then output the address for this customer. If it does not find customer 150, a message will be output to say that the customer is not found.

In pseudocode this would look like:

```plaintext
OUTPUT "Which customer number would you like to look up?"
INPUT user inputs customer number
STORE the user's input in the customer_number variable
counter = 1
(we need to count the number of records that we have searched through)
more_records = True
(we need a flag to say if more records are available to search through
or if we have reached the end of the database)
WHILE more_records = True:
    IF counter = customer_number THEN
        OUTPUT customer address
        Exit the loop
    ELSE
        add 1 to counter
```

As a flowchart, this would look like:
Binary search

Binary search is a faster method for searching for an item that is in an ordered list.

An ordered list is one where the sequence of items in the list is important. An ordered list does not necessarily contain a sequence of numbers (e.g., 1, 2, 3, 4) or characters (e.g., A, B, C, D). It might also contain, e.g., a list of names in alphabetical order, a list of files from smallest to largest or a list of records from earliest to most recent.

A binary search algorithm takes the data and keeps dividing it in half until it finds the item it is looking for.

Example
Imagine that you have a database of customers and want to search for the customer John Smith. We first need the database to be ordered into alphabetical order by surname. We then search for the record ‘John Smith’ by surname.

The binary search will split the database in half, and compare the midpoint (the middle name) to the name ‘John Smith’. It will see whether the midpoint comes before or after ‘Smith’ and will throw away the set of records that doesn’t contain ‘Smith’ as a surname. It will keep dividing the records in that way until it reaches two records left, one of which is ‘John Smith’. It will then throw away the other record and output John Smith’s details. Binary search effectively divides the data in half and throws away, or ‘bins’ the half that does not contain the search term.

In pseudocode this would look like:

```
OUTPUT "Which customer do you want to find?"
INPUT user inputs John Smith
STORE the user’s input in the customer_name variable
customer_found = False
(we need to create a flag that identifies if the customer is found)
WHILE customer_found = False:
    Find midpoint of list
    IF customer_name = record at midpoint of list THEN
        customer_found = True
    ELSE IF customer comes before the midpoint THEN
        throw away the second half of the list
    ELSE
        throw away the first part of the list
OUTPUT customer details
```

As a flowchart, this would look like:
Comparison of searches

Different algorithms might be best used in different situations. For example, sometimes an algorithm won’t work with a particular set of data, and in some instances one algorithm will be much quicker or more efficient than another.

Serial search

One of the main advantages of a serial search is that it is a very simple algorithm, which makes it very easy to write a computer program to carry it out. It can also be used on any set of data regardless of type and whether or not it is sorted.

The biggest problem with a serial search is that it is very slow. For example, when searching through a database of everyone in the UK to find a particular name, it might be necessary to search through 60 million names before you found the one you wanted.

Binary search
One of the main advantages of a binary search is that it is much quicker than a serial search because the data that needs to be searched halves with each step. For example, it is possible to search through 1024 values and find the one you want within 10 steps, every time.

The biggest problem with a binary search is that you can only use this if the data is sorted into an order.
Week 5: Sorting

Why do we need sorting algorithms?

A sorting algorithm will put items in a list into an order, such as alphabetical or numerical order. For example, a list of customer names could be sorted into alphabetical order by surname, or a list of people could be put into numerical order by age.

Sorting a list of items can take a long time, especially if it is a large list. A computer program can be created to do this, making sorting a list of data much easier.

There are many types of sorting algorithms. Two of them are bubble sort and bucket sort.

Bubble sort

A bubble sort algorithm goes through a list of data a number of times, comparing two items that are side by side to see which is out of order. It will keep going through the list of data until all the data is sorted into order. Each time the algorithm goes through the list it is called a ‘pass’.

Example

Imagine that you have a list of people who you want to sort by age, from youngest to oldest. A bubble sort can do this.

The list of ages is:

41, 15, 17, 32, 18, 28, 77 and 54

First pass

The highlighted numbers are the numbers that are being compared.

41, 15, 17, 32, 18, 28, 77, 54

This is the list before it is sorted.

41, 15, 17, 32, 18, 28, 77, 54

The first two numbers are compared. 15 is smaller than 41 so they switch places.

15, 41, 17, 32, 18, 28, 77, 54

The next two numbers are compared. 17 is smaller than 41 so they switch places.

15, 17, 41, 32, 18, 28, 77, 54

The next two numbers are compared. 32 is smaller than 41 so they switch places.

15, 17, 32, 41, 18, 28, 77, 54

The next two numbers are compared. 18 is smaller than 41 so they switch places.

15, 17, 32, 18, 41, 28, 77, 54

The next two numbers are compared. 28 is smaller than 41 so they switch places.

15, 17, 32, 18, 41, 77, 54
The next two numbers are compared. 41 is smaller than 77 so no change occurs.
15, 17, 32, 18, 28, 41, 77, 54

The next two numbers are compared. 54 is smaller than 77 so they switch places.
15, 17, 32, 18, 28, 41, 54, 77

This is what the list looks like after the first pass.
The list is now more ordered than it was originally, but it isn’t yet fully in order of youngest to oldest.
The list needs to go through another pass to compare the numbers again, so it can be sorted further.

Second pass
15, 17, 32, 18, 28, 41, 54, 77

This is what the list looks like after the first pass.
15, 17, 32, 18, 28, 41, 54, 77

The first two numbers are compared. 15 is smaller than 17 so no change occurs.
15, 17, 32, 18, 28, 41, 54, 77

The next two numbers are compared. 17 is smaller than 32 so no change occurs.
15, 17, 32, 18, 28, 41, 54, 77

The next two numbers are compared. 18 is smaller than 32 so they switch places.
15, 17, 18, 32, 28, 41, 54, 77

The next two numbers are compared. 28 is smaller than 32 so they switch places.
15, 17, 18, 28, 32, 41, 54, 77

The next two numbers are compared. 32 is smaller than 41 so no change occurs.
15, 17, 18, 28, 32, 41, 54, 77

The next two numbers are compared. 41 is smaller than 54 so no change occurs.
15, 17, 18, 28, 32, 41, 54, 77

The next two numbers are compared. 54 is smaller than 77 so no change occurs.
15, 17, 18, 28, 32, 41, 54, 77

This is what the list looks like after the second pass.
The set of data is now in order from youngest to oldest, but the algorithm does not know this yet as it made some changes in the second pass. The algorithm will only recognise that the list is in order if it makes no changes in a pass. The algorithm therefore needs to run for a third pass to compare the numbers again. As no changes will be made, the algorithm will then recognise that the data is in order.

If the data being sorted is a large set of data, it may take several passes to get the data sorted.
**Bucket sort**

A bucket sort algorithm separates a list of data into different collections of data, called 'buckets'. Empty buckets are set up at the start of the sort and are filled with the relevant data. Each bucket is then sorted, and the data is finally gathered back into a list.

**Example**

Imagine that you have a list of people who you want to sort by age, from youngest to oldest. A bucket sort can do this.

The list of ages is:

41, 15, 17, 32, 18, 28, 77 and 54

1. Set up a series of empty buckets.

2. Put the data into the correct buckets.

3. Buckets that have more than one item of data in them will be sorted (eg using bubble sort).

4. The data will then be gathered from each bucket and put back into a list.

**The final list will be 15, 17, 18, 28, 32, 41, 54, 77.**

**Comparison of sorts**

It is important to understand that different algorithms might be best used in different situations. For example, sometimes an algorithm won’t work with a particular set of data, and in some instances one algorithm will be much quicker or more efficient than another.

**Bubble sort**
One of the main advantages of a bubble sort is that it is a very simple algorithm to describe to a computer. There is only really one task to perform (compare two values and, if needed, swap them). This makes for a very small and simple computer program.

The biggest problem with a bubble sort is that it takes a very long time to run. For example, if there are 100 values to sort, each pass through the list will take 99 comparisons – and you might have to repeat it 99 times.

**Bucket sort**

One of the main advantages of a bucket sort is that is quicker to run than a bubble sort. Putting data into small buckets that can be sorted individually reduces the number of comparisons that need to be carried out.

The biggest problem with a bucket sort is that the algorithm is a bit more complicated than the bubble sort to describe for a computer. It might be easy to decide to sort numbers in groups of 10, but what about a list of names? Should there be a bucket for each letter of the alphabet, or perhaps one bucket for all of the X, Y and Z names because there aren’t many of those?

**More sorting algorithms**

There are lots and lots of different sorting algorithms. Many of these use clever ideas to make sorting lists much quicker, although the algorithms can get very complicated at times.

Other sorting algorithms include the:

- merge sort
- insertion sort
- shell sort
- quick sort
Week 6: Logical reasoning

What is logical reasoning?

Logical reasoning is the process of applying rules to problem solving. Algorithms are designed as a set of steps to follow to solve a problem. At the same time, a set of rules is determined. For example, selection is based on rules:

1. if a condition is met, do this
2. if not, do that

These rules govern the path that is followed through the algorithm.

Rules are built using logical reasoning to ensure that the algorithm performs correctly.

Why use logical reasoning?

Algorithms are step-by-step instructions to solve a problem, created along with rules that decide which path to take when a decision is faced.
When trying to solve a problem, it may be that more than one solution is found. A different algorithm can be built from each solution. Logical reasoning determines if algorithms will work by predicting what happens when the algorithm’s steps - and the rules they consist of - are followed.

**Predictions from each algorithm can be used to compare solutions and decide on the best one.**

**Logical reasoning in practice**

**Example: Are you old enough to drive?**

Look at this simple pseudocode algorithm for working out if you are old enough to drive a car:

```plaintext
OUTPUT 'How old are you?
INPUT user inputs their age
STORE the user's input in the age variable
IF age > 17 THEN
    OUTPUT 'You are old enough to drive a car!
ELSE IF age < 17 THEN
    OUTPUT 'You are too young to drive a car.'
```

This algorithm has a fault.

Logical reasoning can be used to determine what the fault is.

There are two conditions in this algorithm. The first checks if age is greater than 17. The second checks if age is less than 17.

We can use logical reasoning to predict the outcomes of this algorithm. Logically, it is clear that a person’s age could be greater than 17, less than 17, or could actually be 17. From this, it is possible to use these values in the algorithm to make predictions:

- A rule exists to check if age is greater than 17. If age is greater than 17, the algorithm will output ‘You are old enough to drive a car!’
- A rule exists to check if age is less than 17. If age is less than 17, the algorithm will output ‘You are too young to drive a car.’
- No rule exists to check if age is equal to 17. If age is equal to 17, there will be no output.

Logic tells us that either a rule needs to be added, or the existing rules need to be changed. The algorithm can be fixed by either:

- adding a new rule – ELSE IF age = 17 OUTPUT ‘You are old enough to drive a car!’
- or amending the first rule – IF age >= 17 OUTPUT ‘You are old enough to drive a car!’

Either solution would work.

**Comparing algorithms**

Logical reasoning can be used to compare how effective and efficient different solutions are.

Look at this simple pseudocode algorithm for a cinema’s ticketing system. It works out if you are young enough to get a discount:
There are three conditions in this algorithm. The first checks if age is less than 17. The second checks if age is greater than 17. The third checks if age is equal to 17.

Logically, it is clear that a person’s age could be greater than 17, less than 17 or could actually be 17. From this we use these values in our algorithm to make predictions:

- if age is less than 17, the algorithm will output ‘You qualify for the student discount!’
- if age is greater than 17, the algorithm will output ‘You’re too old for the student discount!’
- if age is equal to 17, the algorithm will output ‘You qualify for the student discount!’

The predictions show that this algorithm will work, because logically all possibilities have been covered.

The following is a different solution:

```plaintext
OUTPUT 'How old are you?'
INPUT User inputs their age
STORE the user's input in the age variable
IF age < 17 THEN
    OUTPUT 'You qualify for the student discount!'
ELSE IF age > 17 THEN
    OUTPUT 'You're too old for the student discount!'
ELSE IF age = 17 THEN
    OUTPUT 'You qualify for the student discount!
```

There is just one condition in this algorithm. It checks if age is less than or equal to 17.

Logically, it is clear that a person’s age could be greater than 17, less than 17 or could actually be 17. From this we use these values in our algorithm to make predictions:

- if age is less than or equal to 17, the algorithm will output ‘You qualify for the student discount!’
- otherwise (ie if age is greater than 17) it will output ‘You are too old for the student discount!’

Both algorithms work well. However, if we look at them both we can see that the first algorithm contains three rules (conditions), but the second contains just one.

The second algorithm is shorter, more efficient and easier to program and, therefore, logically would be a better choice.
Computers use binary to process data. There are simple techniques to convert between binary and denary and to add two binary numbers together.

1. Please enter your name (first name and last name)

Enter your answer

2. What is binary?

- A useless code that is no longer used
- The way a computer speaks
- A sequence of 1s and 0s

3. What kind of 'base' system is binary known as?

- Base 10
- Base 2
- Base 16

4. What kind of number system is used in everyday life?

- Denary
- A simple number system
- Binary
5. What would the denary number 199 be in binary?
   - 1100111
   - 10100111
   - 11000110

6. What would the denary number 55 be in binary?
   - 00101111
   - 00110111
   - 00101111

7. What would the denary number 222 be in binary?
   - 11011110
   - 10111110
   - 11010110

8. What would 10110010 be as a denary number?
   - 168
   - 178
   - 188

9. What would 00101110 be as a denary number?
   - 46
   - 56
10. What would 00100101 + 01000100 be as a binary number?

- 01101001
- 01101011
- 10101001

11. What is an overflow error?

- When the computer program has been given the wrong command
- When the result of a binary calculation is too long for a computer to process
- When something is spelt incorrectly when programming

Submit
Y9 SUM2 Week 2: Representing text, images and sound

Learn how text, images and sound are converted into binary so they can be processed by a computer and how images and sound are compressed to create smaller files.

Required

1. Please enter your name (first name and last name)

2. What form must data be in for a computer to be able to process it?
   - Denary
   - Binary
   - Binary or denary

3. What code can be used to convert text into binary?
   - ASCII code
   - Python
   - Binary code

4. How many characters can the ASCII code store?
   - 128
   - 129
5. What is included in the ASCII code?
   - Just letters
   - Any key on a normal keyboard
   - Just letters and numbers

6. What is metadata?
   - Data that gives information about other data
   - Data that isn’t needed
   - Data that is saved for later use

7. What does an analogue to digital converter do?
   - Assigns binary code to text
   - Makes an image smaller
   - Samples a sound wave at regular intervals then converts these to binary

8. Why would an image be compressed?
   - To increase its quality
   - To make it smaller to store
   - To change the colours it contains

9. What does lossy compression do?
   - Increases the quality of a file
   - Compresses by discarding data that is thought to be unnecessary
10. What does lossless compression do?

- Compresses by discarding data that is thought to be unnecessary
- Increases the quality of a file
- Compresses by breaking the file down into smaller parts for storage then putting them back together again when opened

11. What is a popular method for lossy compression of images?

- MPEG
- JPEG
- TIF

Submit
Y9 SUM2 Week 3: Designing an algorithm

Before designing an algorithm it is important to first understand what the problem is. Algorithms can be designed using pseudocode or a flowchart, and the standard notations of each should be known.

Required

1. Please enter your name (first name and last name)
   Enter your answer

2. What should be considered when designing an algorithm?
   - If the correct software is being used
   - If the correct hardware is being used
   - If there is more than one way of solving the problem

3. What are the two main ways that algorithms can be designed?
   - In pseudocode or as a flowchart
   - By hardware or software
   - Images or videos

4. What is pseudocode?
   - A specific programming language that all computers use
5. What is a flowchart?
   - A flowchart is a specific programming language
   - A flowchart is a text-based way of designing an algorithm
   - A flowchart is a diagram that represents a set of instructions

6. What is the symbol for a decision in a flowchart?
   - A diamond
   - A parallelogram
   - A circle

7. Which of these is the correct symbol for a process in a flowchart?
   - A parallelogram
   - A rectangle
   - A diamond

8. How are symbols connected together in a flowchart?
   - Symbols do not get connected together in a flowchart
   - With lines and an arrow to show the direction of flow
   - By numbers

9. Which symbol is used for a decision in a flowchart?
   - A circle
10. How would a condition loop be created when writing pseudocode?

- With a FOR loop
- With a WHILE loop
- With a WHILE loop or a REPEAT-UNTIL loop

11. When can algorithms be used?

- To design a solution to any problem
- Only with computers
- For programming
Y9 SUM2 Week 4: Searching

Searching for data can be very difficult. Searching algorithms, such as serial search and binary search, make the process of searching for data much easier.

Required

1. Please enter your name (first name and last name)

   Enter your answer

2. What does a searching algorithm do?
   - Search through a set of data
   - Save a set of data
   - Help to organise data

3. What would be needed if searching algorithms didn’t exist?
   - A new set of data to look at would be needed
   - Each item of data would need to be looked at one by one, until the searched for data was found
   - The data would need to be saved

4. Which of these is a type of searching algorithm?
   - Serial search
   - Search engine
5. **What does a serial search do?**
   - Organises the data into alphabetical order
   - Looks at the first item of data, then each one in turn, until it finds the data item requested
   - Splits the data until the requested data is found

6. **What is an advantage of a serial search?**
   - It is very quick
   - It only works with ordered lists
   - It is a simple algorithm

7. **Which of the following is NOT an advantage of a serial search?**
   - It is a simple algorithm, so it is easy to write a computer program to carry it out
   - It is the quickest search to use
   - It can be used on any set of data regardless of type

8. **What does a binary search do?**
   - Looks at the first item of data, then each one in turn, until it finds the data item requested
   - Converts all the data into binary
   - Takes the data and splits it in half repeatedly until it finds the data item requested

9. **Which search algorithm would be best to use with ordered data?**
10. What is an advantage of a binary search

- It is very quick
- It only works with ordered lists
- It is a simple algorithm

11. What is the biggest disadvantage of a binary search?

- It can only be used if the data is sorted into an order
- It is slow
- It takes the data and keeps dividing it in half until it finds the item it is looking for

Submit
Y9 SUM2 Week 5: Sorting

Putting data into order can be difficult and time consuming. Sorting algorithms, such as bubble sort and bucket sort can help with this.

Required

1. Please enter your name (first name and last name)

   Enter your answer

2. What does a sorting algorithm do?
   - Saves a set of data
   - Finds an item of data in a set of data
   - Puts a list of items into order

3. What does a bubble sort do?
   - Separates a list of data into different collections of data, before sorting and gathering back into a list
   - Sorts a list by comparing two items that are side by side, to see which is out of order
   - Finds an item of data in a list

4. How many passes will a bubble sort go through?
   - Only one pass
5. Why does a bubble sort do a final pass even when the data is in the correct order?

- To save the data
- It needs to do this to put the data back into a list
- It does not recognise that the data is in order until the final pass requires no changes

6. Which of the following is an advantage of a bubble sort?

- It takes a very long time to run
- It is a very small and simple computer program
- It is very quick

7. Which of the following is NOT an advantage of a bubble sort?

- There is only one task to perform
- It is very quick
- It is a very small and simple computer program

8. What does a bucket sort do?

- Separates a list of data into different collections of data which are sorted and gathered back into a list
- Finds an item of data in a list
- Goes through a list of data a number of times and compares two items that are side by side to each other to see which is out of order
9. Which of the following is a disadvantage of a bucket sort?
   - It takes a long time to run
   - There is only one task to perform
   - It is more complicated than a bubble sort

10. Which of the following is an advantage of a bucket sort?
    - It is simpler than a bubble sort
    - It takes a long time to run
    - It is quicker to run than a bubble sort

11. Which of the following is not a type of sorting algorithm?
    - Merge sort
    - Insertion sort
    - Long sort

Submit
Y9 SUM2 Week 6: logical reasoning

There is almost always more than one solution to a problem. Logical reasoning is used to predict the outcomes of the algorithms that are designed to solve a problem, to help select the best solution.

Required

1. Please enter your name (first name and last name)

   Enter your answer

2. What is logical reasoning?
   - Using rules to solve problems
   - Counting rules in an algorithm
   - Using steps to solve problems

3. What is logical reasoning used for?
   - To predict the outcome of an algorithm
   - To make decisions
   - To iterate steps

4. Why is logical reasoning used?
   - To compare the effectiveness of different algorithms to solve a problem
   - To determine where to start in an algorithm
5. Which symbol means 'greater than'?

- [ ] 5
- [ ] 6
- [ ] 7

6. Which symbol means 'greater than or equal to'?

- [ ] 5
- [ ] 6
- [ ] 7

7. Which symbol means 'equal to'?

- [ ] 5
- [ ] 6
- [ ] 7

8. Which condition would confirm that someone 17 or over is able to drive a car?

- [ ] age >= 17
- [ ] age > 17
- [ ] age = 17

9. Which condition would confirm that someone 12 or over is able to watch a 12 rated film?

- [ ] age >= 12
10. Which condition would confirm that only someone 21 and over can drive a mini-bus?

- age > 21
- age = 22
- age >= 21

11. Which condition would say that only those older than 14 can watch a 15 rated film?

- age > 14
- age = 15
- age >= 14

Submit