

AQA Computer Science A-Level
**4.5.6 Representing images,
sound and other data**
Advanced Notes



Specification:

4.5.6.1 Bit patterns, images, sound and other data:

Describe how bit patterns may represent other forms of data, including graphics and sound.

4.5.6.2 Analogue and digital:

Understand the difference between analogue and digital:

- data
- signals

4.5.6.3 Analogue/digital conversion:

Describe the principles of operation of:

- an analogue to digital converter (ADC)
- a digital to analogue converter (DAC)

Know that ADCs are used with analogue sensors.

Know that the most common use for a DAC is to convert a digital audio signal to an analogue signal.

4.5.6.4 Bitmapped graphics:

Explain how bitmaps are represented.

Explain the following for bitmaps:

- resolution
- colour depth
- size in pixels

Calculate storage requirements for bitmapped images and be aware that bitmap image files may also contain metadata.

Be familiar with typical metadata.

4.5.6.5 Vector graphics:

Explain how vector graphics represents images using lists of objects.

Give examples of typical properties of objects.

Use vector graphic primitives to create a simple vector graphic.



4.5.6.6 Vector graphics versus bitmapped graphics:

Compare the vector graphics approach with the bitmapped graphics approach and understand the advantages and disadvantages of each.

Be aware of appropriate uses of each approach.

4.5.6.7 Digital representation of sound:

Describe the digital representation of sound in terms of:

- sample resolution
- sampling rate and the Nyquist theorem

Calculate sound sample sizes in bytes.

4.5.6.8 Musical Instrument Digital Interface (MIDI):

Describe the purpose of MIDI and the use of event messages in MIDI.

Describe the advantages of using MIDI files for representing music.

4.5.6.9 Data compression:

Know why images and sound files are often compressed and that other files, such as text files, can also be compressed.

Understand the difference between lossless and lossy compression and explain the advantages and disadvantages of each.

Explain the principles behind the following techniques for lossless compression:

- run length encoding (RLE)
- dictionary-based methods

4.5.6.10 Encryption:

Understand what is meant by encryption and be able to define it.

Be familiar with Caesar cipher and be able to apply it to encrypt a plaintext message and decrypt a ciphertext. Be able to explain why it is easily cracked.

Be familiar with Vernam cipher or one-time pad and be able to apply it to encrypt a plaintext message and decrypt a ciphertext. Explain why Vernam cipher is considered as a cypher with perfect security.

Compare Vernam cipher with ciphers that depend on computational security.



Bit patterns, images, sound and other data

So far, we've only seen bit patterns used to represent **numbers**. However, computers also use bit patterns to represent **all other forms of data**, including **graphics** and **sound**.

Analogue and digital

Analogue data is **continuous**, there are **no limits** to the values that the data can take. In contrast, digital data is **discrete**, meaning that it can only take **particular values**.

Analogue and digital signals vary in a similar way. An analogue signal can take **any values** and can change **as frequently as required** whereas a digital signal must always take **one of a specified range of values** and can only change value **at specified intervals**.



Analogue signal



Digital signal

The trace of a digital signal's value against time is characterised by **sharp edges** and **vertical drops** as the signal changes value. While analogue signals tend to be **smooth curves**, they can sometimes feature **sharp peaks**.

Analogue/digital conversion

Digital to analogue conversion

When converting from digital to analogue, a device called a **digital to analogue converter** (or **DAC** for short) is used. The device reads a bit pattern representing an analogue signal and outputs an alternating, analogue, electrical **current**.

The most common use for a DAC is to convert a digital **audio signal** to an **analogue signal**.

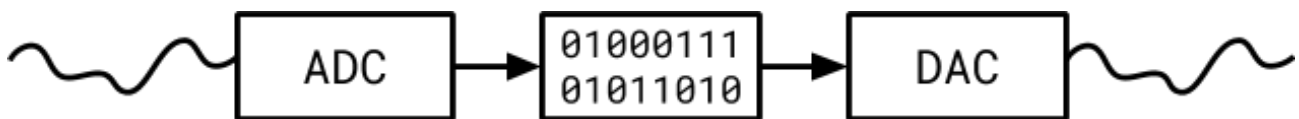


Analogue to digital conversion

Many sensors such as [temperature sensors](#) and [microphones](#) output an analogue signal. When a computer needs to make use of these, they use an [analogue to digital converter](#) (ADC for short) to convert the analogue signal to a digital bit pattern. The device works by taking a [reading](#) of an analogue signal at [regular intervals](#) and recording the value in a process called [sampling](#).

Samples are taken at a specific [frequency](#), given in [Hertz](#), which determines the [number of samples taken per second](#). This is usually a [high number](#) as greater sampling frequencies result in a [better reproduction](#) of the analogue signal.

Once the value of the analogue signal has been recorded, it can be stored [digitally](#) as a bit pattern.

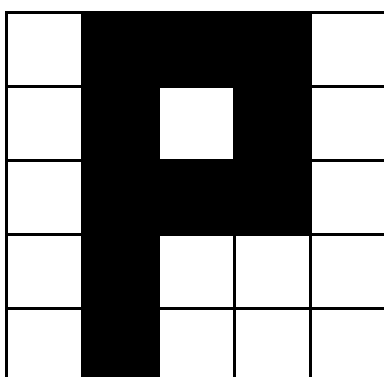


Bitmapped graphics

Computers represent [images](#) in two different ways, one of which is by using [bitmap graphics](#). In bitmap graphics, an image is broken down into [pixels](#), each of which has a [binary](#) value assigned to it.

The [resolution](#) of an image is often expressed as a [number of dots per square inch](#) in an image, where a dot is a [pixel](#). However, resolution can also refer to the [number of pixels](#) in an image, for example, the image below could be said to have a resolution of 5×5 pixels.

The [value assigned](#) to a pixel [determines the colour](#) of the pixel. The example below shows the [binary representation](#) of a simple bitmap image in which a 1 represents a black pixel and a 0 represents a white pixel.



```

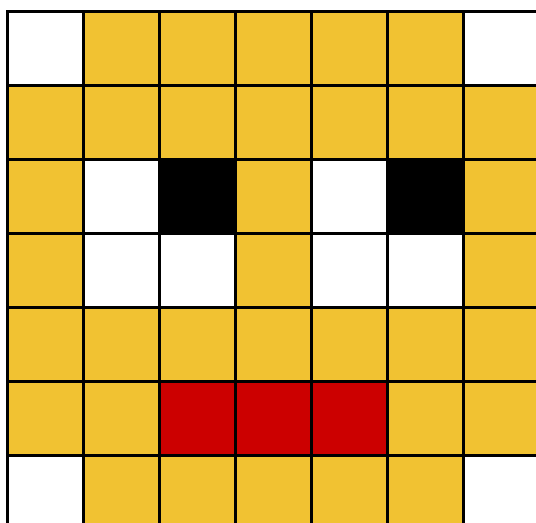
0 1 1 1 0
0 1 0 1 0
0 1 1 1 0
0 1 0 0 0
0 1 0 0 0
  
```

Pixel

Picture element. The smallest addressable part of a bitmap image.



The **number of bits** assigned to a pixel in an image is called its **colour depth**. In the example above, each pixel has been assigned **one bit**, allowing for 2 (2^1) different colours to be represented. If a colour depth of **two bits** were used, there would be **four** (2^2) different colours that each pixel could take, represented by the bit patterns 00, 01, 10 and 11.



```

00 11 11 11 11 11 00
11 11 11 11 11 11 11
11 00 01 11 00 01 11
11 00 00 11 00 00 11
11 11 11 11 11 11 11
11 11 10 10 10 11 11
00 11 11 11 11 11 00
  
```

In order to calculate the **storage required** to represent a bitmap image, multiply the **number of pixels** (width \times height) by the **bit depth**.

The picture of the face has $7 \times 7 = 49$ pixels, each of which is assigned **two bits**, so it requires **98 bits** to be represented.

$$7 \times 7 \times 2 = 98 \text{ bits}$$

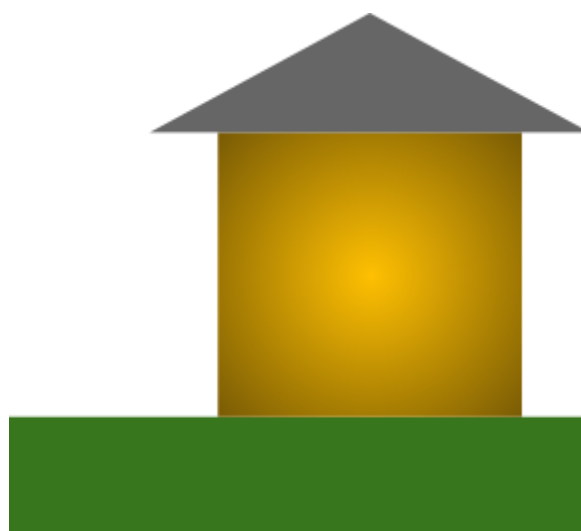
This method of calculating the storage requirements for bitmapped images produces a **minimum value**. This is because bitmap image files may also contain **metadata**, typical examples of which include the image's **width**, **height**, **date created** and **colour depth**.



Vector graphics

Vector graphics represent images using **geometric objects** and **shapes** such as rectangles, circles and lines. The **properties** (such as fill colour, fill style and dimensions) of each geometric object or shape in the image are stored in a **list**.

shape	properties
rectangle	fill-colour: green fill-style: solid height: 2 width: 10 start-position: (0, 0)
square	fill-colour: yellow fill-style: vignette width: 6 start-position: (4, 2)
triangle	fill-colour: grey fill-style: solid width: 7 start-position: (3, 8)



Vector graphics versus bitmapped graphics

Because vector graphics use shapes rather than pixels, they can be scaled **without losing quality**. Enlarging a bitmap image results in a **blurry** or even **pixelated** image whereas enlarging a vector graphic results in **no loss of clarity**.

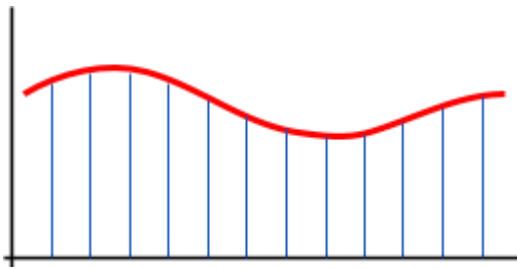
Vector graphics are well suited to simple images which use shapes, like **company logos**, but they're **no use for photographs**. Bitmapped graphics are used for storing photographs.

Vector graphics frequently use **less storage space** than bitmapped graphics, as information is stored for each shape, rather than for every single pixel in an image.

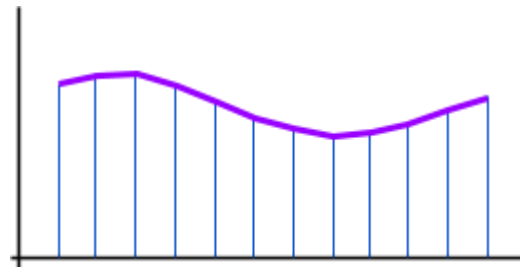


Digital representation of sound

Computers represent sound as a **sequence of samples**, each of which takes a **discrete digital value**. The number of samples per second is called the **sampling rate** and is expressed in **Hertz**.



Analogue signal sampled



Samples used to recreate signal digitally

The **number of bits** allocated to each sample is referred to as the **sample resolution**. Higher sample resolutions result in **greater audio quality** but also **increased file size**.

The size of a sound sample can be calculated by **multiplying together** the **duration of the sample in seconds**, the **sampling rate in Hertz** and the **sample resolution**.

For example, a **minute-long** audio file sampled at **44 kHz** with a sample resolution of **24 bits** would require almost **8 megabytes** of storage.

$$60 \times 44,000 \times 24 = 63360000 \text{ bits}$$
$$63360000 \div 8 = 7920000 \text{ bytes}$$

As with images, **metadata** can be stored in an audio file. This will **increase the space required** to store the file.

The Nyquist Theorem

The Nyquist theorem states that the sampling rate of a digital audio file must be **at least twice** the frequency of the sound. If the sampling rate is below this, the sound may not be accurately represented.



Musical Instrument Digital Interface (MIDI)

Musical instrument digital interface, or **MIDI**, is used with **electronic musical instruments** which can be **connected to computers**. Rather than storing samples of sound, MIDI stores sound as a series of **event messages**, each of which represents an **event** in a piece of music. These can be thought of as a **series of instructions** which could be used to recreate a piece of music.

Event messages could contain information such as:

- The **duration** of a note
- The **instrument** with which a note is played
- How **loud** a note is (its **volume**)
- If a note should be **sustained**

There are numerous advantages to using MIDI over a sampled recording of a piece of music. Using MIDI allows **easy manipulation** of music **without loss of quality**. The instruments on which notes sound can be changed, notes can be **transposed** and the duration of notes can be altered.

Furthermore, MIDI files are often **smaller in size** than sampled audio files and are **lossless** so there's **no information lost** when music is stored using MIDI.

However, there are disadvantages to using MIDI. MIDI **can't be used for storing speech** and sometimes results in a **less realistic sound** than sampled recordings.



Data compression

Files are compressed in order to **reduce their size**. Smaller files can be **transferred faster** between storage devices or over the internet.

Images are often compressed, but sound files and text files can also be compressed.

There are two categories of compression, **lossy** and **lossless**.

Lossy compression

When using lossy compression, **some information is lost** in the process of reducing the file's size. This could be **reducing the resolution** of an image or **lowering the sample resolution** of a sampled audio file.

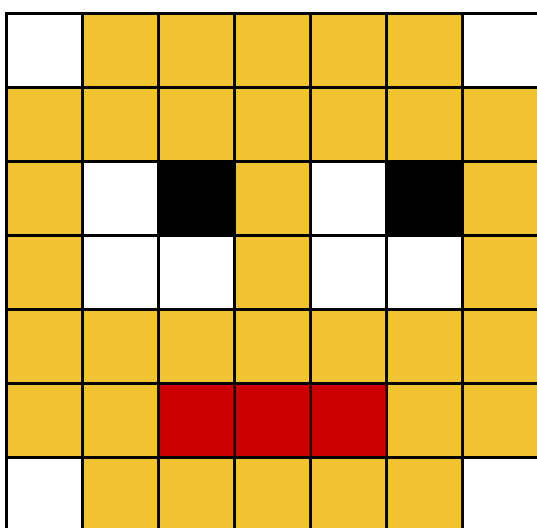
Lossless compression

In contrast to lossy compression, there is **no loss of information** when using lossless compression. The size of a file can be reduced **without decreasing its quality**.

Two methods of lossless compression are **run length encoding** and **dictionary-based methods**.

Run length encoding (RLE)

Run length encoding (**RLE** for short) **reduces the size** of a file by removing **repeated information** and replacing it with **one occurrence** of the repeated information followed by the **number of times** it is to be repeated.



```

00 115 00
      117
11 00 01 11 00 01 11
      11 002 11 002 11
          117
112 103 112
      00 115 00
  
```



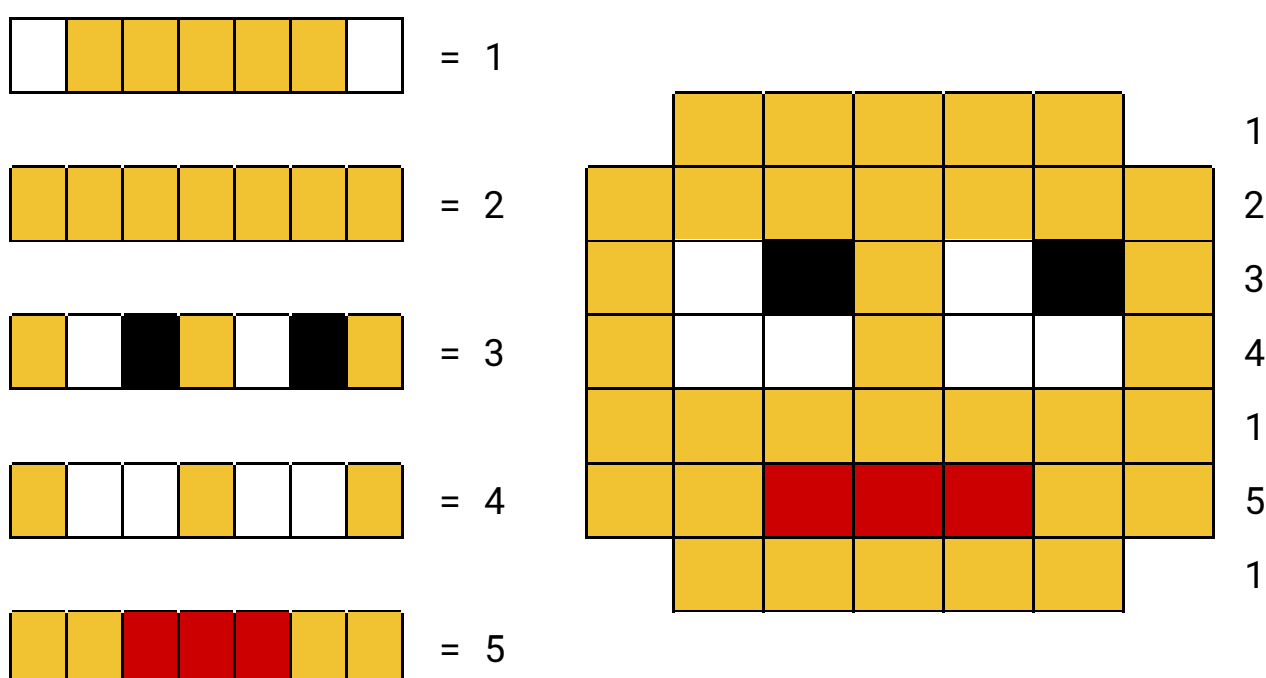
The example uses the image of a face that was represented as a bitmap image earlier in these notes. Using RLE to **replace repeated pixels** with one pixel value and a **number or repetitions** has **reduced the storage space** required to represent the image.

The third row of pixels in the image has **no repeated values** and as such, **couldn't be compressed** by RLE. This highlights the fact that **not all data is suitable for compression by run length encoding**.

Dictionary-based methods

When a file is compressed with a dictionary-based method, a **dictionary** containing **repeated data** is **appended** to the file.

For the picture of the face, the dictionary on the left could be used.



Using the dictionary, the file could be represented using just the data 1234151, as shown on the right.

This method results in a **significant reduction** in size, but don't forget that the dictionary used to compress the data **has to be present in the file** in order for the image to be reproduced. This will **increase the size** of the file.

Both run length encoding and dictionary-based methods are **most effective** on files that contain **a lot of repeated data**.



Lossy Compression	Lossless Compression
Some information is lost in the compression process	No loss of information
Quality of file is reduced	No loss of quality
The extent to which file size can be reduced is not limited	There is a limit to how much a file can be compressed

Encryption

Encryption is the process of scrambling data so that it cannot be understood if intercepted in order to keep it secure during transmission. Unencrypted information is referred to as plaintext and encrypted information is called ciphertext. A cipher is a type of encryption method.

In order to decrypt ciphertext, you must know the encryption method used and the key used to encrypt the information.



Caesar ciphers


Caesar ciphers encrypt information by **replacing characters**. One character is **always** replaced by the **same character**. There are two types of Caesar cipher that you need to be aware of. **Shift ciphers** and **substitution ciphers**.

Shift ciphers

When encrypting using a shift cipher, all of the letters in the alphabet are **shifted by the same amount**. The amount by which characters are shifted forms the key.

Plaintext

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z



X Y Z A B C D E F G H I J K L M N O P Q R S T U V W

Ciphertext

The example above uses a **shift** of three characters, so the key is **three**. Using the key three, the plaintext “TURING” could be encrypted as the ciphertext “QROFKD”.

Substitution ciphers

Substitution ciphers are a type of Caesar cipher in which letters are **randomly replaced**.

Plaintext

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

F J E D M K B I C H L S A T U R V W G Y Q N P Z X O

Ciphertext

Using the cipher in the example, the plaintext “FLOWERS” would be encrypted as “KSUPMWG”.

Caesar ciphers can be **easily cracked**. The **frequency** at which each character occurs can provide a clue as to which letter has been replaced with which. The **most commonly occurring letter** in English is likely to be an E. Once you discover just **one character**, a shift cipher can be **completely cracked**, as the **key** can be found. Substitution ciphers are a little better but are still **relatively easy** to crack.



Vernam ciphers

The Vernam cipher is an example of a one-time pad cipher. This means that **each key should only ever be used once**. Additionally, the Vernam cipher requires the key to be **random** and **at least as long as the plaintext** that is to be encrypted.

The Vernam cipher works by:

1. **Aligning** the characters of the **plaintext** and the **key**
2. Converting each character to **binary** (using an **information coding system**)
3. Applying a logical **XOR** operation to the two bit patterns
4. Converting the result back to a character

Synoptic Link

Computers use **information coding systems** to represent characters.

Information coding systems are covered under **Fundamentals of Data Representation**.

XOR

Logical operation which outputs true when exactly one of its inputs is true.

Example. encrypting:

Plaintext	H	E	L	L	O
Plaintext binary	01001000	01000101	01001100	01001100	01001111
Key	u	r	n	w	u
Key binary	01110101	01110010	01101110	01110111	01110101
Plaintext binary XOR key binary	00111101	00110111	00100010	00111011	00110101
Ciphertext	=	7	"	;	5

In the example above, each of the characters in the plaintext and the key are converted to binary, then XORed before being converted back to characters.

As the example shows, the plaintext HELLO is encrypted by a Vernam cipher with the key urnwu as the ciphertext = "7";5.



Example, decrypting:

When decrypting using a Vernam cipher, the key that was used to encrypt the plaintext is used again.

Ciphertext	=	7	"	;	5	
Ciphertext binary		00111101	00110111	00100010	00111011	00110101
Key		u	r	n	w	u
Key binary		01110101	01110010	01101110	01110111	01110101
Ciphertext binary XOR key binary		01001000	01000101	01001100	01001100	01001111
Plaintext		H	E	L	L	O

Since the key used with a Vernam cipher is chosen at random, the ciphertext is also random and so the cipher is considered **completely secure**. In fact, the Vernam cipher is the only cipher mathematically proven to be completely secure.

Computational security

All ciphers other than the Vernam cipher are, in theory, crackable, but not within a reasonable timeframe given current computing power. Ciphers that use this form of security are said to rely on **computational security**.

