## Sound

**Sample** - Measure of the analogue signal at a given point in time

**Sample rate -** number of samples taken per second and is measured in Hertz.

**Sample resolution** - number of bits used to represent each sample

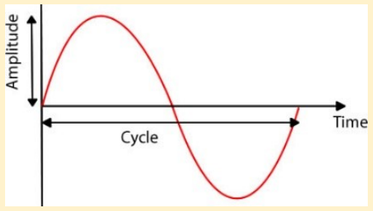
The size of sound files can be calculated using:

*size of file = length (seconds) x sample rate x sampling resolution*

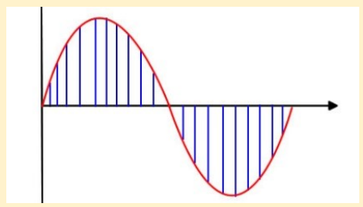
For sound to be stored digitally on a computer it needs to be converted from its continuous analogue form into a discrete binary values. The steps are:

1. Microphone detects the sound wave and converts it into an electrical (analogue) signal
2. The analogue signal is sampled at regular intervals
3. The samples are approximated to the nearest integer (quantised)
4. Each integer is encoded in binary with a fixed number of bits

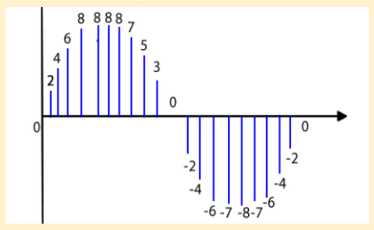
*Original analogue signal*




*Sample signal at regular intervals*



*Integer values give to each sample*




*Encode as binary*

0 2 4 6 8 8 8 8 7 5 3 0 ->

00000 00010 00100 01000

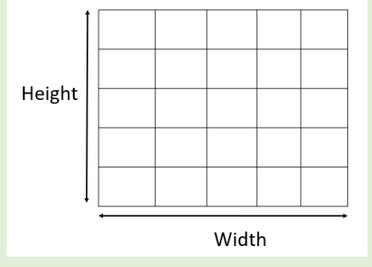
01000 01000 01000 00111

00101 00011 …

## Images

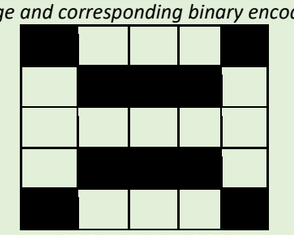
Bitmap images are made up from tiny dots called **pixels**. Each pixel will have a colour associated with it. An image can then be constructed from many of pixels which will have different colours arranged in rows and columns.

*Total number of pixels in image = width in pixels x height in pixels*




**Colour depth** is the number of bits used to represent each pixel in an image. If we have a black and white image it has two colours. Each pixel can be represented by a single pixel because a bit value of 0 is black and 1 is white.

*Image and corresponding binary encoding*



0111010001111111000101110

To represent more colours we can use more bits. For instance if we have 2-bits per pixel we can represent 4 colours because we know have 4 binary code combinations (00, 01, 10 11) where each code represents a different colour

**Pixilation** occurs when the image is overstretched. In these situations, the image looses quality and has a blocky and blurred appearance. This arises when the image is presented at too large a size and there are not enough pixels to reproduce the details in the image at this larger size.

**Calculating the size of a bitmap image**

*File size in bits = width in pixels x height in pixels x colour depth*

*File size in bytes = width in pixels x height in pixels x colour depth / 8*

## Data Compression

The purpose of data compression is to make the files smaller which means that:

- Less time / less bandwidth to transfer data
- Take up less space on the disk

Given that there are 7 bits per ASCII character, the uncompressed size of an ASCII phrase is:

*size = number of characters (including spaces) x 7*

**Run Length Encoding (RLE)** is a compression method where sequences of the same values are stored in pairs of the value and the number of those values. For instance, the sequence:

0 0 0 1 1 0 1 1 1 1 0 1 1 1 1

would be represented as:

3 0 2 1 1 0 4 1 1 0 4 1

**Huffman coding** is a form of compression that allows us to use fewer bits for higher frequency data. More common letters are represented using fewer bits than less common letters. For instance, “a” and “e”, which occur in many words would be represented with fewer bit than “z” which occurs rarely.
This allows for much more effective compression than RLE.

The steps involved in Huffman encoding as are follows:

1. Do frequency table
2. Order table
3. Create the tree
4. Add 1, 0 to the branches
5. Encode letters
6. Encode message

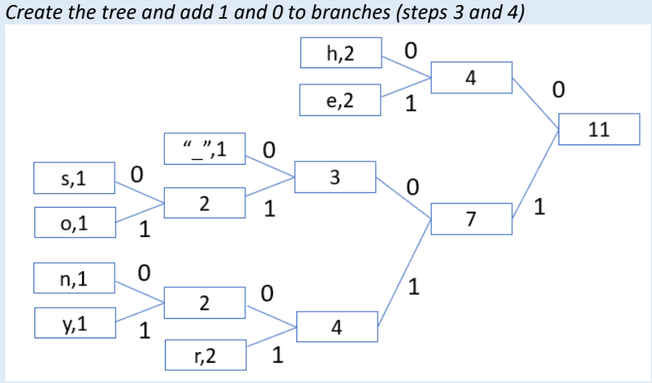
*Worked Example: How much smaller is the phrase henry horse encoded using Huffman encoding compared with its uncompressed size.*

*Calculate the uncompressed size*
In the phrase *henry horse* there are 11 characters (including the space). Therefore the uncompressed size is 11 x 7 = <u>77 bits</u>

*Generate ordered frequency table (steps 1 and 2)*

| letter | frequency |
|---|---|
| e | 2 |
| h | 2 |
| r | 2 |
| <space> | 1 |
| o | 1 |
| s | 1 |
| y | 1 |
| n | 1 |

*Create the tree and add 1 and 0 to branches (steps 3 and 4)*




*Encode letters*

| Letter | encoding |
|---|---|
| e | 01 |
| h | 00 |
| r | 111 |
| <space> | 100 |
| o | 1011 |
| s | 1000 |
| n | 1100 |
| y | 1101 |

*Encode message*
00 01 1100 111 1101 100 00 1011 111 1000 01 = <u>33 bits</u>

Therefore by using compression we have reduced the size from 77 bits to 33 bits a saving of <u>44 bits</u>.