

Exercise walk-through

1.1

Data Representation

A-Level Computer Science

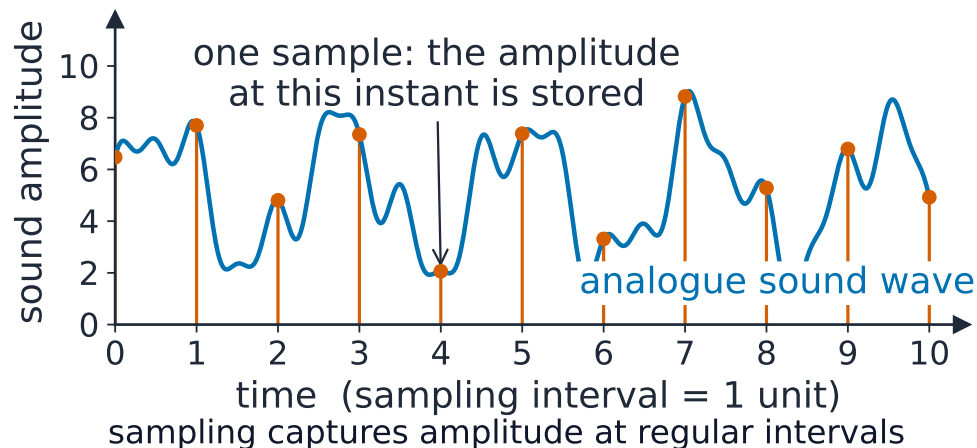
Sampling sound

Analogue data →
digital data by
sampling :

- **sampling rate** (CD 44.1 kHz)
- **sampling resolution** – bits per **amplitude quantisation**, set by

10 s stereo CD

$44100 \times 16 \times 10 \times 2 \approx 1.68 \text{ MiB}$



Run-length encoding

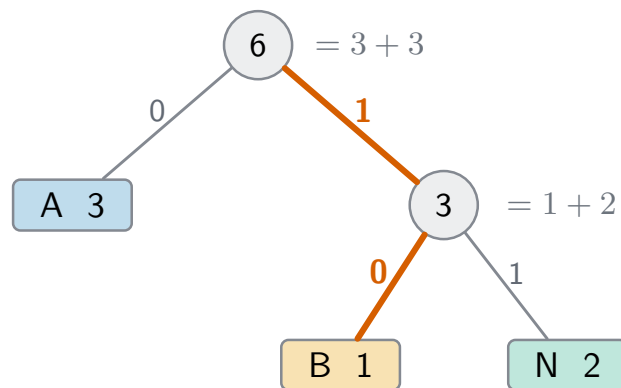
Store each **run** of identical values once, as a **(count, value)** pair – **lossless**, so the original comes back exactly. Best on **flat areas**; useless on **noisy** data, where a run of one costs more than the value it replaces.

8×8 bitmap	binary	RLE code
	00000000	8W
	01111110	1W 6B 1W
	01100000	1W 2B 5W
	01111100	1W 5B 2W
	01100000	1W 2B 5W
	01100000	1W 2B 5W
	01100000	1W 2B 5W
	00000000	8W

Key: 1 = black (B), 0 = white (W). RLE stores *count + colour*: e.g. 01111110 → 1W 6B 1W

Huffman coding

Give the **commonest** symbol the **shortest** code – a **variable-length code**, still **lossless**. No code starts another, so the decoder needs no separator, and the average code length approaches the data's **entropy**.



Walk **down from the root**, writing a bit at each branch: left = 0, right = 1.

A = 0

B = 10 right, then left

N = 11

each circle joins the two smallest counts

BANANA becomes 100110110 — **10 bits**, against 12 for fixed 2-bit codes: the commonest letter is nearest the root.

Exercise 2.1

2.1 Convert binary 01101010 to denary.

[1]

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

Exercise 2.1

2.1 Convert binary 01101010 to denary.

[1]

0	1	1	0	1	0	1	0
---	---	---	---	---	---	---	---

2.1

- binary $\rightarrow$ denary: multiply each 1-bit by 2^n (position n counts from 0 at the right)
- 1-bits at positions 6, 5, 3, 1: $2^6 + 2^5 + 2^3 + 2^1$
- $64 + 32 + 8 + 2 = \mathbf{106}$

Exercise 2.2

2.2 Convert denary 200 to 8-bit binary.

[2]

200 →

--	--	--	--	--	--	--	--

--

Exercise 2.2

2.2 Convert denary 200 to 8-bit binary.

[2]

200 →

--	--	--	--	--	--	--	--

2.2

- denary → binary: divide by 2, record remainders, read **bottom to top**

```
200 ÷ 2 = 100 rem 0
100 ÷ 2 = 50  rem 0
 50 ÷ 2 = 25  rem 0
 25 ÷ 2 = 12  rem 1
 12 ÷ 2 = 6   rem 0
 6 ÷ 2 = 3    rem 0
 3 ÷ 2 = 1    rem 1
 1 ÷ 2 = 0    rem 1 ← left-most bit
```

- remainders bottom → top: 11001000
- check: $128 + 64 + 8 = 200$

Exercises 2.3·2.4

2.3 Convert binary 11110001 to (a) denary and (b) hexadecimal.

[2]

1	1	1	1	0	0	0	1
---	---	---	---	---	---	---	---

2.4 State the **minimum number of bits** needed to store the denary value 500. Show how you decided.

[2]

500	→	<table border="1"><tr><td></td><td></td><td></td></tr></table>				...	<table border="1"><tr><td></td><td></td></tr></table>		

Exercises 2.3·2.4

2.3 Convert binary 11110001 to (a) denary and (b) hexadecimal.

[2]

1	1	1	1	0	0	0	1
---	---	---	---	---	---	---	---

2.3

(a) binary $\rightarrow$ denary

- 1-bits at positions 7, 6, 5, 4, 0: $128 + 64 + 32 + 16 + 1 = \mathbf{241}$

(b) binary $\rightarrow$ hex: split into nibbles from the right

- $1111 = 15 = \mathbf{F}$ and $0001 = 1$, so **F1**

2.4 State the **minimum number of bits** needed to store the denary value 500. Show how you decided.

[2]

500

 $\rightarrow$

--	--	--

 ...

--	--

2.4

- need the smallest b with $2^b > 500$
- $2^8 = 256$ is too small; $2^9 = 512 > 500$
- **9 bits**

Exercise 2.5

2.5 Represent the denary number 59 in **BCD**.

[2]

59 →

--	--	--	--	--	--	--	--

Exercise 2.5

2.5 Represent the denary number 59 in **BCD**.

[2]

59 →

--	--	--	--	--	--	--	--

2.5

- BCD stores **each denary digit** as its own 4-bit code
- 5 → 0101, 9 → 1001
- 59 in BCD = 0101 1001

Exercise 2.6

2.6 Add the 8-bit numbers 01011100 and 00110110. State whether overflow occurs. [2]

$$\begin{array}{r} \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline \end{array} \\ + \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ \hline \end{array} \end{array}$$

Exercise 2.6

2.6 Add the 8-bit numbers 01011100 and 00110110. State whether overflow occurs. [2]

$$\begin{array}{r}
 \boxed{0} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{1} \boxed{0} \boxed{0} \\
 + \boxed{0} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0}
 \end{array}$$

2.6

- add one column at a time from the right

$$\begin{array}{r}
 0101 \ 1100 \quad (92) \\
 + 0011 \ 0110 \quad (54) \\
 \hline
 1001 \ 0010 \quad (146)
 \end{array}$$

- $146 \leq 255$ (no carry out of bit 7), so **no overflow**

Exercise 3.1

3.1 Add the 8-bit binary numbers 10110100 and 01101101. Show your working, then **name and describe** the error that occurs. [4]

$$\begin{array}{r} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{0} \\ + \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \\ \hline \end{array}$$

Exercise 3.1

3.1 Add the 8-bit binary numbers 10110100 and 01101101. Show your working, then **name and describe** the error that occurs. [4]

$$\begin{array}{r} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{0} \boxed{0} \\ + \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \boxed{1} \boxed{0} \boxed{1} \end{array}$$

3.1

- add the columns from the right

1011 0100 (180)
Full solution on next slide

Full solution 3.1

3.1

- add the columns from the right

```
    1011 0100    (180)
+   0110 1101    (109)
-----
1 0010 0001    (carry out of bit 7)
```

- stored 8-bit result = 00100001
- $180 + 109 = 289 > 255$, so a 1 is lost from bit 7
- the error is **overflow**: the result needs 9 bits but only 8 are stored, so the stored answer is wrong

Exercise 3.2

3.2 (a) Convert the hexadecimal number 3D to denary. Show your working. [2]

3D hexadecimal

(b) Convert the denary number 158 to hexadecimal. Show your working. [2]

Exercise 3.2

3.2 (a) Convert the hexadecimal number 3D to denary. Show your working. [2]

3D hexadecimal

(b) Convert the denary number 158 to hexadecimal. Show your working. [2]

3.2

(a) hex $\rightarrow$ denary: powers of 16

- D = 13
- $3 \times 16^1 + 13 \times 16^0 = 48 + 13 = \mathbf{61}$

(b) denary $\rightarrow$ hex: divide by 16

- $158 \div 16 = 9$ remainder 14
- remainder 14 = E, so **9E**

Exercise 3.3

3.3 (a) Write the denary number -45 as an 8-bit two's complement binary number. Show your working. [2]

-45 →

(b) The 8-bit two's complement number 11101110 is stored. Calculate the denary value it represents. [2]

Exercise 3.3

3.3 (a) Write the denary number -45 as an 8-bit two's complement binary number. Show your working. [2]

-45 →

(b) The 8-bit two's complement number 11101110 is stored. Calculate the denary value it represents. [2]

3.3

(a) write -45 in 8-bit two's complement

Full solution 3.3

3.3

(a) write -45 in 8-bit two's complement

- $+45 = 32 + 8 + 4 + 1 \rightarrow 00101101$
- invert every bit $\rightarrow 11010010$
- add 1 $\rightarrow 11010011$

(b) read 11101110

- sign bit is 1, so the value is **negative**
- invert $\rightarrow 00010001$
- add 1 $\rightarrow 00010010 = 18$
- denary value = **-18**

Exercise 3.4 (a)

3.4 A computer stores text using a character set.

(a) The text is stored using the **Unicode** character set instead of **ASCII**. Give **two** advantages of using Unicode. [2]

text:

H	i	!
---	---	---

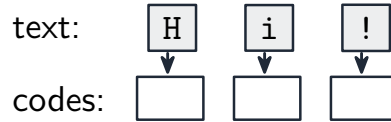
codes:

--	--	--

Exercise 3.4 (a)

3.4 A computer stores text using a character set.

(a) The text is stored using the **Unicode** character set instead of **ASCII**. Give **two** advantages of using Unicode. [2]



(a) any **two** of:

- Unicode can represent characters from (almost) every language / far more characters than ASCII
- it includes symbols and emoji
- it avoids the regional clashes of extended ASCII

Exercise 3.4 (b)

3.4 A computer stores text using a character set.

(b) A character has the binary code 0010 0111. Convert this code to denary. [1]

Exercise 3.4 (b)

3.4 A computer stores text using a character set.

(b) A character has the binary code 0010 0111. Convert this code to denary. [1]

(b) binary $\rightarrow$ denary

- 1-bits at positions 5, 2, 1, 0: $32 + 4 + 2 + 1 = \mathbf{39}$

Exercise 3.5 (a)

3.5 A program stores currency amounts using **BCD** rather than ordinary binary.

(a) State what **BCD** stands for.

[1]

Exercise 3.5 (a)

3.5 A program stores currency amounts using **BCD** rather than ordinary binary.

(a) State what **BCD** stands for.

[1]

(a) **Binary Coded Decimal**

Exercise 3.5 (b)

3.5 A program stores currency amounts using **BCD** rather than ordinary binary.

(b) Give one **advantage** of using BCD for this purpose. [1]

2.49 ◦

Exercise 3.5 (b)

3.5 A program stores currency amounts using **BCD** rather than ordinary binary.

(b) Give one **advantage** of using BCD for this purpose. [1]

2.49 ◦

(b) any **one** of:

- each denary digit is stored exactly, so money values are not changed by rounding errors when converting fractions to binary
- BCD digits map directly onto a seven-segment display