Solutions

Each answer shows the steps, then the **result**.

2.1

- lossless: the original data is recovered **exactly**
- lossy: some detail is **permanently removed** to make the file smaller

2.2

- lossless — e.g. a program / document / spreadsheet, because every bit must be exact
- lossy — e.g. a streamed photo / song / video, because a much smaller file matters more than perfect detail

2.3

- group identical pixels into runs
- **5W 3B 1W**

2.4

- a photograph has detail that changes pixel to pixel, so runs are only 1 long — storing (count, value) is no shorter

3.1

- video is a huge amount of data and must be sent in **real time** over limited **bandwidth**
- lossless would not shrink it enough, so it would keep buffering/freezing
- lossy makes it small enough to stream with only minor quality loss

3.2

(a) each row is split into runs of the same colour; each run is stored as a (count, colour) pair instead of every pixel

- long runs of one colour take far less space

(b) e.g. a simple logo / icon / large areas of one colour

3.3

- use a lossless method: a dictionary replaces repeated character sequences with a short reference, **or** Huffman coding gives frequent characters shorter codes
- the exact text can still be rebuilt

3.4

(a) songs are large and streamed over mobile data, so a much smaller file downloads faster and uses less data

- the listener barely notices the dropped detail

(b) e.g. some audio quality is lost permanently

3.5

- spatial compression reduces the data **within a single frame** (like a JPEG image)
- temporal compression stores only the **differences between frames**
- most of one frame is the same as the previous one

3.6

(a) fewer samples are taken each second, so there is less data and the file is **smaller**

- the higher frequencies can no longer be represented, so the sound is duller and less like the original
- data has been thrown away permanently, so this is **lossy** compression

(b) a scanned document must stay **exactly** readable — losing detail can turn one character into another

- the image is mostly flat white, which lossless methods compress very well anyway
- a photograph has continuous tones where small changes are invisible to the eye, so lossy compression saves far more space at no noticeable cost

(c) `1011010` has four 1s, already even, so the parity bit is `0`

- `1110111` has six 1s, also even, so the parity bit is `0`

(d) even parity by row: byte 1 has three 1s → `1`; byte 2 has two 1s → `0`; byte 3 has three 1s → `1`

- the parity byte holds the column parities: `1,0,1` → `0`; `0,1,1` → `0`; `1,1,0` → `0`; `1,0,1` → `0`

(e) a single byte's parity bit only says that the number of 1s is wrong — any one of the bits could be the culprit

- in a block check the flipped bit breaks the parity of **both** its row and its column
- the row and column intersect at exactly one position, so the faulty bit can be found and corrected