

Data

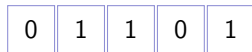
AP Computer Science Principles ■ Unit 2

AP Computer Science Principles

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

What we will cover

- 1 Binary numbers
- 2 Data compression
- 3 Extracting information
- 4 Using programs with data



bits



Everything is bits

A **bit** is a single 0 or 1. **Binary** (base 2) uses place values that are **powers of two** 2 .

128	64	32	16	8	4	2	1	
0	0	1	0	1	1	0	1	= 45

$32 + 8 + 4 + 1 = 45$. Read binary $\rightarrow$ decimal by adding the place values where a bit is 1.

place value	128	64	32	16	8	4	2	1
bits	1	0	0	1	0	1	1	0

$$150 = 128 + 16 + 4 + 2 \Rightarrow 10010110$$

Fixed bits, overflow & sampling

Convert & overflow

1011_2 : place values 8, 4, 2, 1; bits at 8, 2, 1 give $8 + 2 + 1 = 11$.

With n bits you can make 2^n values, 0 to $2^n - 1$. Exceeding it is an **overflow error** (4 bits max 15; $15 + 1$ overflows).

Analog signals become **digital** by **sampling** – many quick measurements.
Eight bits group into one **byte** .

A finite number of bits has two consequences

the cause

a computer has a **finite** number of bits, so it can represent only a limited range of values

overflow error

a number **too large** for the available bits cannot be stored correctly

round-off error

numbers with **decimals** can only be **approximated**, because infinitely many real values must map onto finitely many bit patterns

why it matters

$0.1 + 0.2$ does not print as 0.3 in most languages – and that is not a bug in the language

Both are consequences of the same fact. Say “finite bits” first and the explanation writes itself.

Where a number loses precision

Overflow

A value too large for the bits allotted wraps around or errors – the count that goes negative.

Round-off error

Round-off (rounding) error : a real number stored in finite bits is stored *approximately*, so $0.1 + 0.2 \neq 0.3$ exactly.

Accumulation

Round the same way a million times and the error compounds – which is why money is stored in integer cents, not floats.

The shared cause

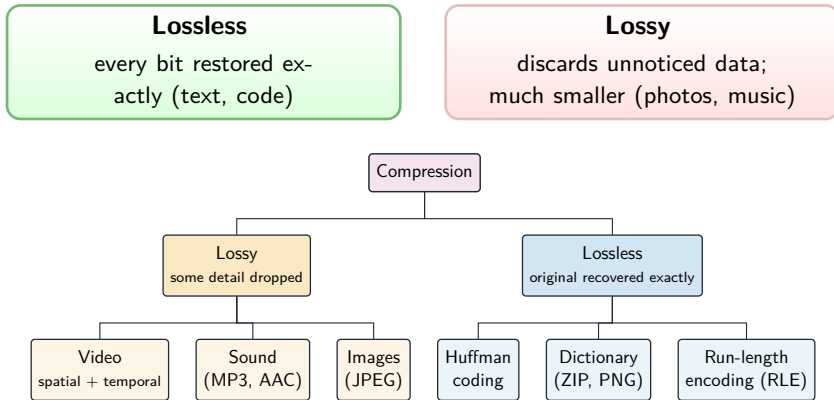
Both are the same limit: a fixed number of bits cannot represent unlimited values.

An answer that names the limit – “only 2^{32} distinct values exist” – explains both errors at once.



Data compression

Compression stores the same data in **fewer bits** – less space, faster transmission.



Lossy trades **size** against **quality**. Many methods remove **redundancy** – e.g. “100 white pixels” → “white, 100 times”.

Two kinds of compression, and when each is right

compression

reducing the number of bits needed to store or send data

lossless compression

the original is **perfectly recoverable** – PNG, ZIP, FLAC. Use it for text, code and anything where a changed bit matters

lossy compression

discards data judged unnoticeable, and it **cannot be recovered** – JPEG, MP3, MP4

the trade

lossy achieves far smaller files; lossless guarantees fidelity

the rule

never use lossy on a program's source, a spreadsheet, or a medical scan

The exam asks which is appropriate, not which is better. Match the choice to **whether losing detail is acceptable**.



Extracting information from data

Programs turn raw **data sets** into **information** ,
finding **patterns** and **trends** .

Filtering

keep the rows you want

Cleaning

fix duplicates, blanks, errors

Beware: a **correlation** never proves **causation** , and **bias** in how data was collected can mislead. **Metadata** & **visualization** help.



*Hard-disk platters and head: data compressed
and stored as magnetic patterns*

From data to information

data into information

data becomes useful when patterns, trends and answers are extracted from it

filtering

keeping only the rows that meet a condition

cleaning

removing errors, duplicates and incomplete records before analysing

visualizing

a chart makes a pattern visible that a table hides

the caution

large data sets reveal **correlation**, which is not causation – and they raise **privacy** questions about the people in them

Programs process data at scales no human can. That is the benefit *and* the reason the privacy question is on this paper.



Using programs with data

A program **processes** data: records in, new information out.

Iterate to summarise

A loop uses **iteration** to visit each **element** : summing 70, 85, 90, 60, 95 gives 400;
 $\div 5 = 80$ (the average).

A **filter** keeps a **subset** that meets a condition; programs also **combine** sources, **transform** values, and **clean** first – dirty data gives a wrong answer.

Worked example – choosing a compression

**Which compression for a source-code file?
For a holiday photo?**

Code → **lossless**. Every bit matters – a single changed character breaks the program. Restoring the **exact** original is non-negotiable. **Photo** → **lossy**. A small quality loss is invisible, and the file gets **much** smaller.

The trade is **size** against **fidelity**. Lossy loss is **permanent** – you can never get the discarded data back.

Key points

Binary place values are powers of two; watch **overflow**

Lossless keeps every bit; **lossy** trades quality for size

Correlation is not **causation**; clean data first

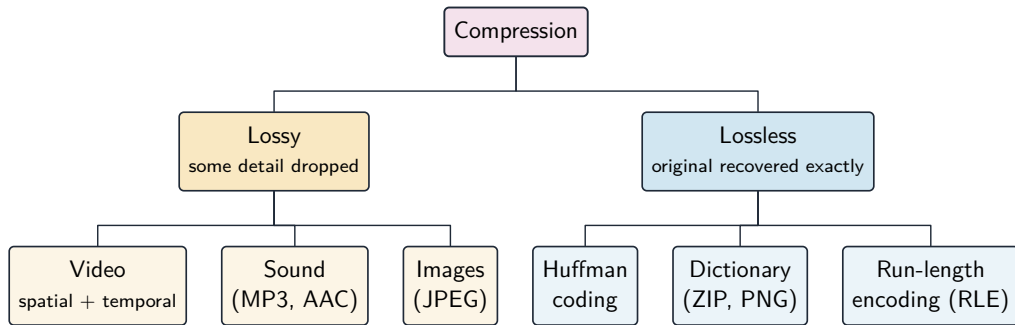
Programs **iterate**, **filter** & transform data

Binary Numbers



Binary digits on a display – all digital data is ultimately stored as 0s and 1s

Check yourself, in detail



Compression methods: lossless versus lossy, with common examples

Exam tips

- Convert confidently between **binary**, **decimal**, and (where asked) hexadecimal – practise until it is quick.
- Remember a bit is one binary digit and a byte is 8 bits; n bits represent 2^n values.
- Explain that all data – numbers, text, images, sound – is stored as binary, and that finite bits cause **overflow** and **round-off**.
- Distinguish **lossless** from **lossy** compression and when each is appropriate.
- Show the analog-to-digital idea: **sampling** turns a continuous signal into discrete values.



Unit 2 – key takeaways

1

Bases

binary, decimal, hex – until it is quick

2

Bits

n bits $\Rightarrow 2^n$ values; byte = 8 bits

3

Limits

overflow and round-off from finite bits

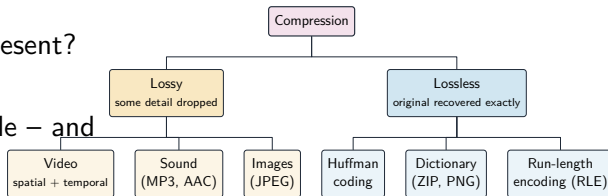
4

Compression

lossless vs lossy; sampling analog to digital

Check yourself

- 1 Convert 1011_2 to decimal.
- 2 How many values can n bits represent?
- 3 Lossless or lossy for a program file – and why?
- 4 Does a correlation in the data prove causation?



Answers 1. 11 2. 2^n (from 0 to $2^n - 1$) 3. lossless – every bit must be restored exactly 4. no