Lossless Data Compression

Entropy coding vs. dictionary-based algorithms

CSCI 201 / ALGORITHM DEEP DIVE

Decoding recovers the original bytes exactly.

James Abarca, Cyril Lorenzo, Andrew Villorente

What is lossless compression?

Compression represents data in less space when the data has patterns.

ORIGINAL		HUFFMAN PAYLOAD		RESTORED
BANANA	encode →	**110100100**	decode →	**BANANA**

The payload is 9 bits. The decoder also needs the codebook, which maps bit strings to letters.

Lossless

The restored data matches the original byte for byte. Example: a zipped document.

Lossy

The restored data can differ because encoding discards some detail. Example: a typical JPEG photo.

A bit is a 0 or 1. A byte is eight bits. A codec encodes and decodes data.

Why we use the word BANANA

One word contains the two patterns this project explores.

ENTROPY CODING

Some letters occur more often

A appears three times, N twice, and B once. A common letter can get a shorter code.

```
A: 3    N: 2    B: 1
```

DICTIONARY CODING

Some sequences repeat

ANA appears twice. A compressor can represent a repeat by pointing to earlier data.

```
B [A N A] N A
B A N [A N A]
```

WHAT ENTROPY MEASURES

Entropy is the average uncertainty per symbol, given a probability model.

$$H = -\sum p(x) \log_2 p(x)$$

p(x) is the probability of symbol x. We estimate it from the letter counts in BANANA.

How we compare codecs

Suppose a 100 MiB input compresses to 10 MiB and takes 2 seconds to restore.

COMPRESSED SIZE

10% of the original

How much space the compressed file takes.

DECOMPRESSION SPEED

50 MiB/s

How much original data the decoder restores per second.

PEAK DECODER MEMORY

Highest RAM use

The most working memory the decoder process used during the run.

A MiB is 1,048,576 bytes. RAM is working memory. The benchmark measures file size and RAM separately.

Huffman coding

In a prefix code, no complete codeword begins another codeword.
Huffman uses symbol frequencies to minimize average length among binary prefix codes.

FREQUENCIES IN BANANA

`A × 3` `N × 2` `B × 1`

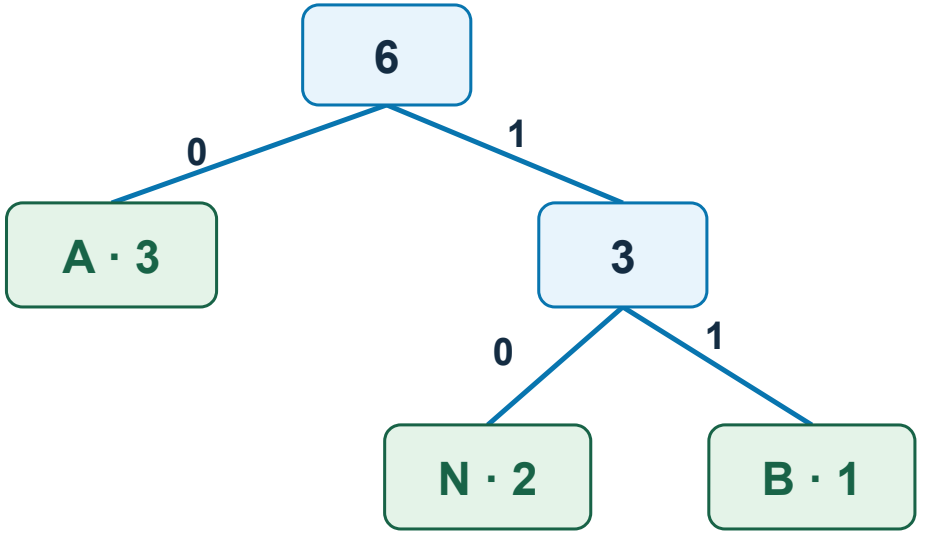
Merge B(1) and N(2), then merge the new node with A(3).

ENCODED PAYLOAD

`11  0  10  0  10  0`

9 bits, excluding the codebook and framing

PREFIX TREE



A = 0 N = 10 B = 11

48-bit byte representation / 12-bit fixed-width payload / 9-bit Huffman payload

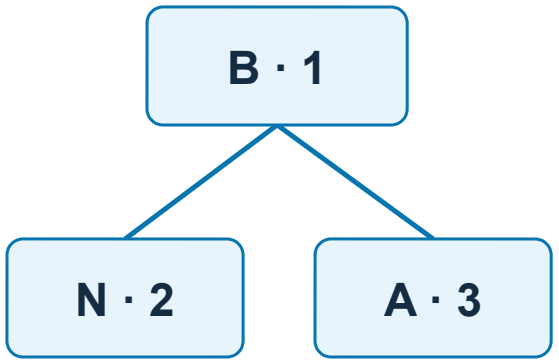
Huffman averages 1.50 bits per letter here. Letter-frequency entropy is 1.46 bits per letter (about 8.75 bits across six letters).
The fixed-width code assumes three known symbols. Payload sizes exclude codebook and framing overhead.

How the Huffman min-heap changes

A min-heap keeps the lightest node at the root for each merge.

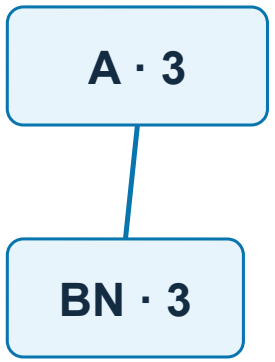
INITIAL HEAP

Weight = letter count



FIRST MERGE

Pop B and N. Push BN (3).



FINAL MERGE

Pop A and BN. Push root (6).



The heap selects the two nodes to merge. Either child can take branch 0 in the resulting code tree.

Our tree assigns N to branch 0 within BN. Swapping branches changes bit patterns while preserving code lengths.

Arithmetic coding and ANS

Both methods use symbol probabilities to encode sequences efficiently.

Arithmetic coding

Refine an interval for every symbol. A binary tag inside the final interval identifies the message when the decoder knows its length.

ARITHMETIC CODING FOR AB

A narrows the interval to [0, 0.75).
B narrows it to [0.5625, 0.75).
The tag $0.101_2 = 0.625$ lies in that interval.
The decoder knows to recover two symbols.

Asymmetric numeral systems

Update an integer state for each symbol. In tabled ANS, the decoder uses the state to look up a symbol and an update rule.

TOY rANS STEP FOR A

$M = 4$, $f(A) = 3$, $f(B) = 1$. A starts at 0.
Write $5 = 1 \times 3 + 2$.
Encoding A gives $1 \times 4 + 2 = 6$.
The decoder finds $6 \bmod 4 = 2$ in A's range [0, 3).
It recovers state $1 \times 3 + 2 = 5$.

Both use $P(A) = 0.75$ and $P(B) = 0.25$. M is total frequency. The toy rANS step precedes renormalization.

LZ77 sliding window

A match stores a length and a backward distance into the decoded output.

ENCODER OUTPUT

```
B   A   N   +   copy(length=3, distance=2)
```

DECODER SNAPSHOTS

```
B A N   →   B A N A   →   B A N A N   →   B A N A N A
```

The third copy reads the A written by the first copy.

Decoders check that the backward distance falls within the available history.

This LZSS-style variant emits literals or (length, distance) matches. LZ77 (1977) used (offset, length, next character) triples.

How an LZ77 hash chain finds a match

A prefix hash narrows the search to earlier positions in one bucket.

TOY INPUT AND ZERO-BASED POSITIONS

```
0: ABX      3: ABY      6: ABZ      9: ABXQ
```



Points to newest | Collision: skip | Collision: skip | Match: length 3

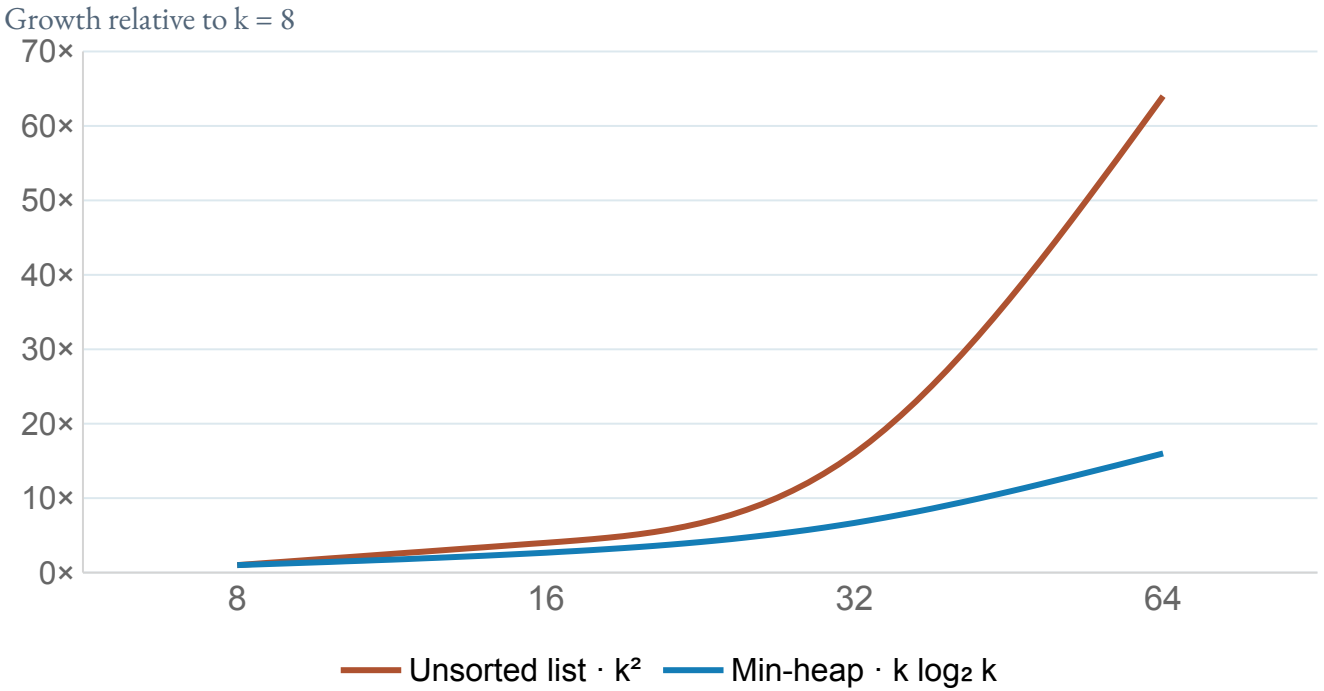
Emit distance 9 and length 3. A two-candidate cap would miss this match.

The toy hash puts ABX, ABY, and ABZ in one bucket. Position 0 must still be inside the window.

Data structures and search costs

Big O describes how work or memory grows as the input gets larger.

THEORETICAL HUFFMAN TREE CONSTRUCTION COST



Distinct symbols (k). From 8 to 64 symbols, the quadratic model grows 64× and the heap model grows 16×.

Counting input takes O(n), where n is input length. The tree and heap use O(k) space.

LZ77 bounds are per search. W = window size, L = maximum comparison length, C = candidate cap.

LZ77 MATCH SEARCH

Scan the window: O(WL)
Cap the chain: O(CL)
Window and index: O(W) space

LZ78 / LZW PHRASE LOOKUP

A trie shares phrase prefixes and stores their codes.

LZ78 sends a phrase index plus the next symbol. LZW sends a code for the whole matched phrase.

Implementation challenges

A compressor needs a bitstream format and a decoder that checks its inputs.

CANONICAL HUFFMAN

Dynamic DEFLATE sends code lengths. The decoder rebuilds canonical codes in a fixed order.

A bit buffer handles codes that cross byte boundaries. Literal/length and distance codes have a 15-bit cap.

LZ77 SEARCH AND COPY

Cap candidate visits and check collisions. Lazy matching probes the next position for a longer match.

With three bytes of available history, distance four is invalid. A distance-two match can overlap. Copy forward so each new byte can feed the next step.

rANS RENORMALIZATION

The encoder emits low bits to keep its state bounded. The decoder refills the state from those bits.

Sources: DEFLATE specification (RFC 1951, sections 3–4) and Duda's ANS paper.

How modern codecs combine the methods

The dictionary stage finds matches. Entropy coding represents the resulting symbols.

FORMAT	MATCH REPRESENTATION	ENTROPY CODING
DEFLATE	LZ77 window matches	Huffman codes for literals, lengths, and distances
Zstandard	LZ77-style sequences	Huffman for compressed literals and FSE for sequence symbols
Brotli	LZ77 + built-in dictionary	Context-dependent Huffman code selection

FSE means Finite State Entropy. The formats also define other block modes.

ONE RAW FIXED-HUFFMAN DEFLATE BLOCK FOR BANANA

LZ77 emits B, A, N, and a match of length 3 at distance 2. Fixed codes represent those tokens and end of block.

With the match, the block takes 46 bits before padding, or 6 bytes. Six literals take 58 bits, or 8 bytes.

These raw block sizes include the block header and end marker. The input itself is 6 bytes.

Compression in deployment

Different workloads make transfer size, memory, or recovery time more important.

FILE ARCHIVERS

.tar.gz and .tar.zst

Balance archive size and restore time.

HTTP TEXT DELIVERY

Brotli and Zstandard content codings

Reduce bytes sent while keeping client decoding fast.

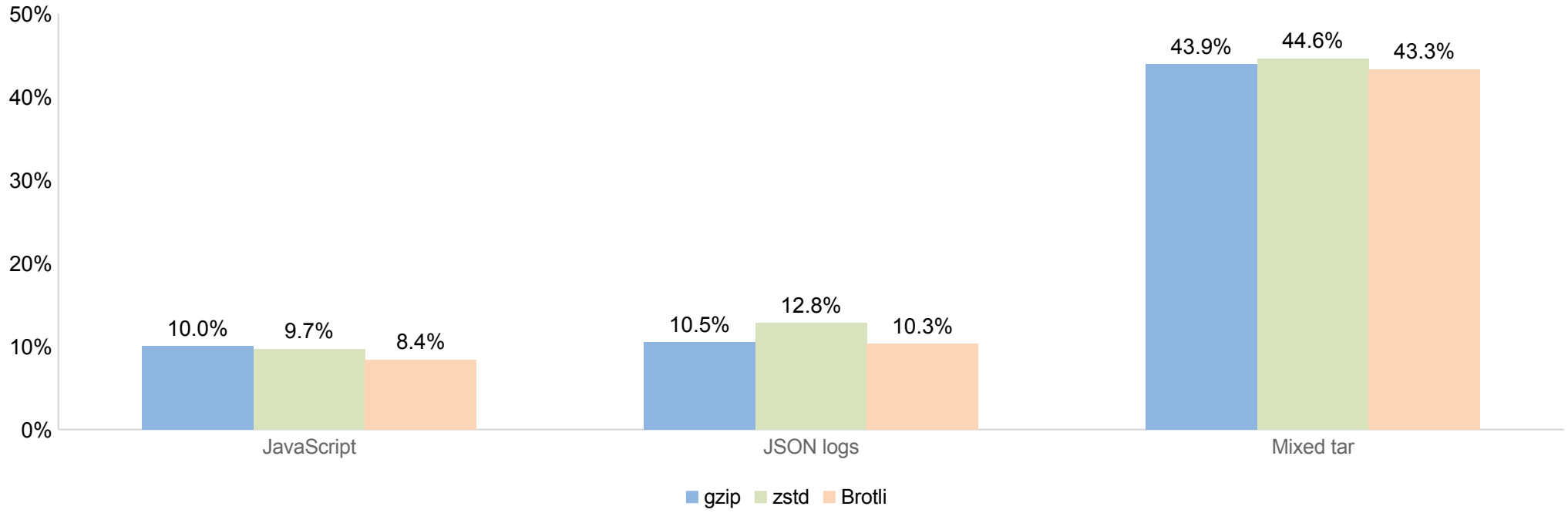
CLOUD DATA PIPELINES

Zstandard-compressed Parquet pages

Balance storage cost, read speed, and memory.

Brotli used the fewest bytes at these settings

Compressed size as a percentage of original bytes. Lower is better.



Generated JavaScript, JSON logs, and mixed tar (7.2–7.6 MiB). About 37% of the tar was random bytes. gzip level 6, zstd level 3, and Brotli quality 5. All nine combinations recovered the original bytes.

Decode speed varied by workload

Cells show median throughput in MiB/s and peak decoder process memory in MiB.

Input	gzip (−6)	zstd (−3)	Brotli (q5)
JavaScript	605 / 1.3	1,001 / 5.2	609 / 9.9
JSON logs	1,426 / 1.3	792 / 5.5	892 / 10.0
Mixed tar	2,049 / 1.3	1,267 / 6.0	551 / 10.3

JavaScript compressed to 8.4% with Brotli and 9.7% with Zstandard.

Zstandard decoded at 1,001 vs. 609 MiB/s and used 5.2 vs. 9.9 MiB peak process memory.

Five runs per case on one Apple Silicon Mac. Timing includes startup and compressed file reads.

Peak RSS covers the entire decoder process. Each codec uses its own setting scale.

Apple gzip 479, zstd 1.5.7, Brotli 1.2.0.

Takeaway and Q&A

Evaluate a codec on the input and resource limits of its deployment.

Smaller files can cost more to decode.

If you restore this data many times on a memory-limited server, which result matters most?
What would you measure on your real workload?

Tutorial mirrors

[Author 1 URL] [Author 2 URL] [Author 3 URL]

References

Huffman · Minimum-redundancy codes

Witten et al. · Arithmetic coding

Duda · Asymmetric numeral systems

Ziv & Lempel · LZ77

Ziv & Lempel · LZ78

Welch · LZW

Shannon · A Mathematical Theory of Communication

RFC 1951 · DEFLATE

RFC 8878 · Zstandard

RFC 7932 · Brotli

RFC 9659 · HTTP zstd window

GNU tar · Compressed archives

Apache Parquet · Page compression