AMIES 2026 SYMPOSIUM — DATA ENGINEERING

Recursive Order-Based Compression for Permutable Lists

Bernardo Machado Ferreira, Nuno Escudeiro

Instituto Superior de Engenharia do Porto (ISEP)

ISEP, Porto · 24th of September 2026

OVERVIEW

Agenda

01
Motivation & research question

02
State of the art & gaps

03
Proposed method & theory

04
Implementation

05
Experimental results

06
Conclusions & future work

MOTIVATION

Permutable data hides free capacity in its ordering

Database rows, image collections, JSON records, file lists

their order is often semantically meaningless

A permutable list of *N* distinct elements admits

$$\log_2(N!) \text{ bits}$$

of free, unused encoding capacity in its arrangement alone

Standard compressors assume order is fixed and meaningful. This dissertation asks: what if we used that ordering itself as an information channel?

CENTRAL RESEARCH QUESTION

How can order information in permutable lists be used to compress them efficiently, at scale, while keeping random access?

Scalability

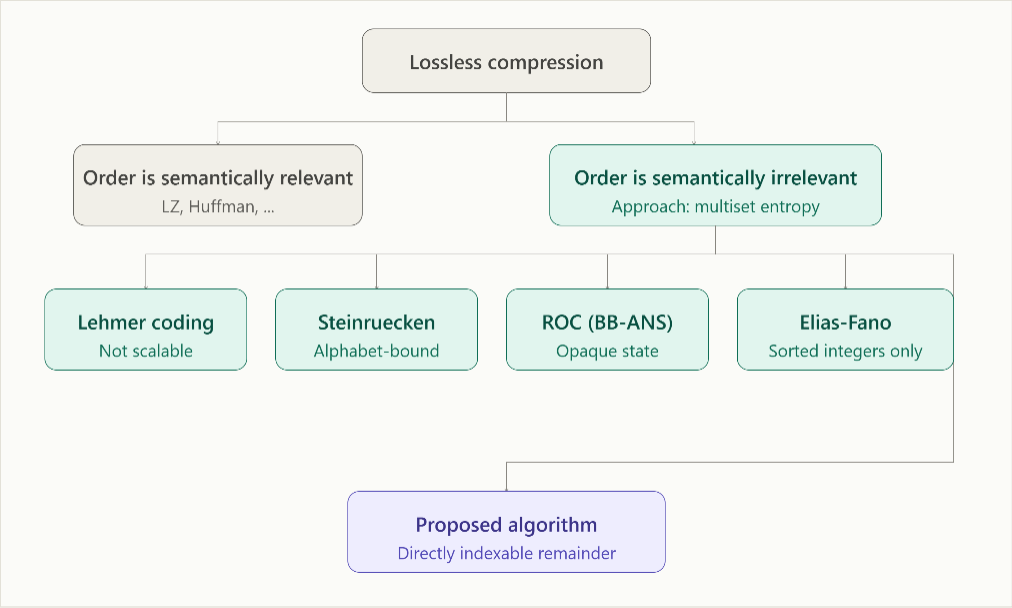
O(N log K) time, K = distinct elements

Optimality

Asymptotically converges to the entropy bound

Access

O(1) random access to the remainder



POSITIONING

Several prior methods already exploit order

Lehmer coding, Steinruecken's arithmetic trees, ROC, and Elias–Fano each trade off scalability, random access, or generality differently.

This method combines a specific configuration of different properties, namely its practicality

Highly compatible with standard compressors like *zstd* or *gzip*

THE PROPOSED APPROACH

A list's order is just another way to write the same list

Take the same 8 letters, arranged three different ways - still the same letters:

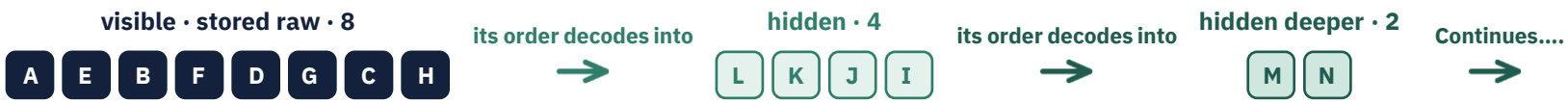
D A G H B C E F

H E F C A B G D

H C B E F G D A

Which arrangement we pick is free to choose - and that choice itself can carry information.

We can spend that freedom: the list we keep stays visible - its order quietly hides a smaller list, whose order hides a smaller list, and so on...



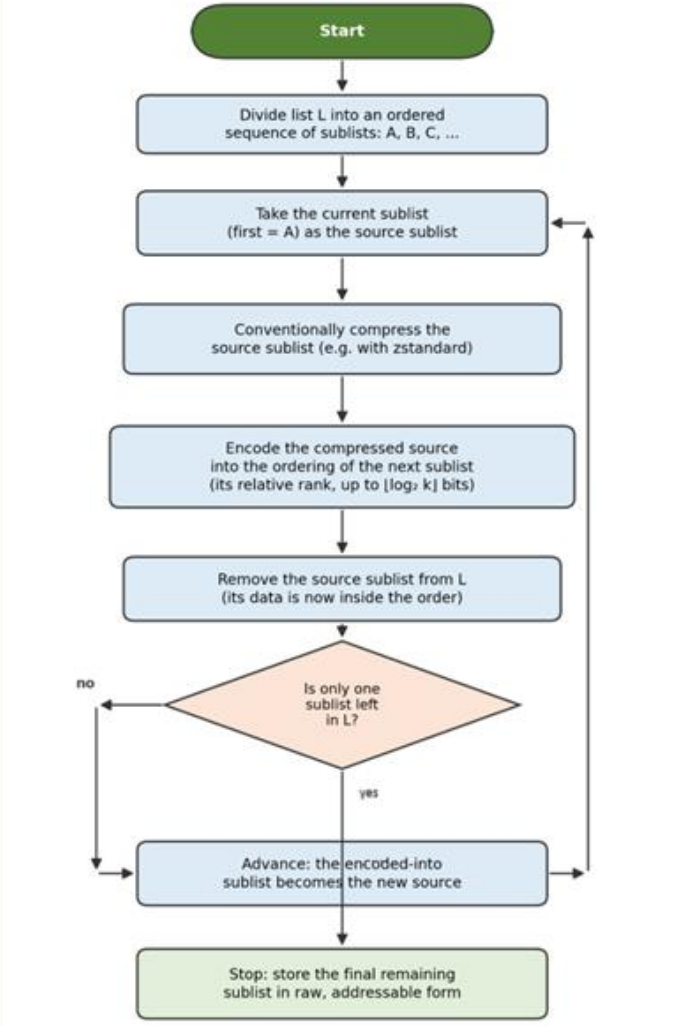
**Illustrative only, not a literal worked example*

MECHANICS

The encoding loop

- Split L into an ordered sequence of sublists
- Compress the source sublist conventionally
- Encode it into the next sublist's ordering — $\lfloor \log_2(\text{alive_count}) \rfloor$ bits at a time
- Remove the source sublist; advance
- Repeat until one sublist remains, stored raw

Decoding simply reverses this walk.



THEORETICAL RESULT

The recursive savings converge to the entropy bound

Geometric decay of information per level

$$a_{(k+1)} \sim a_k / c \quad \text{as } n \to \infty$$

Savings vs. theoretical optimum

$$\text{Savings} / \text{Optimal} \to 1$$

Proven analytically for lists of unique elements; Experiments shows the same asymptotic optimality holds empirically for multisets, where elements repeat.

IMPLEMENTATION DETAIL

Two ways to size each sublist

Worst-case (guaranteed)

$\lfloor \log_2 \rfloor$ accumulation per frequency level

No leftover bits, no split accepted unless provably decodable

More conservative: larger first parts, lower compression ratio

Average-case (multinomial)

$\log_2$ of the multinomial coefficient, via lgamma

Holds asymptotically — some leftover bits stored separately

Closer to optimal — the preferred default in practice

Both use binary search over prefix lengths, O(log N) evaluations per split; only the capacity function differs.

IMPLEMENTATION

Python 3.12, a Fenwick tree, and O(N log K)

Stack

Standard library only; optional zstandard binding; NumPy for benchmarking

Fenwick tree

Maps the k-th alive symbol to its index in O(log K); point-update on exhaustion

Complexity

O(N log K) overall; O(N log N) when every element is unique (K = N)

Three modules: **encode.py**, **decode.py**, **encodefiles.py**. Each sublist except the last is conventionally compressed, then folded into the next sublist's order via the Fenwick-backed encoder.

EXPERIMENTAL EVALUATION

Setup

Hardware

13th Gen Intel Core i7-1355U, 16 GB RAM, Windows 11, single-threaded

Methodology

10 runs per configuration; mean and standard deviation reported per step

Datasets

MNIST images, synthetic 32-bit integers (1K–1M rows), JSON records

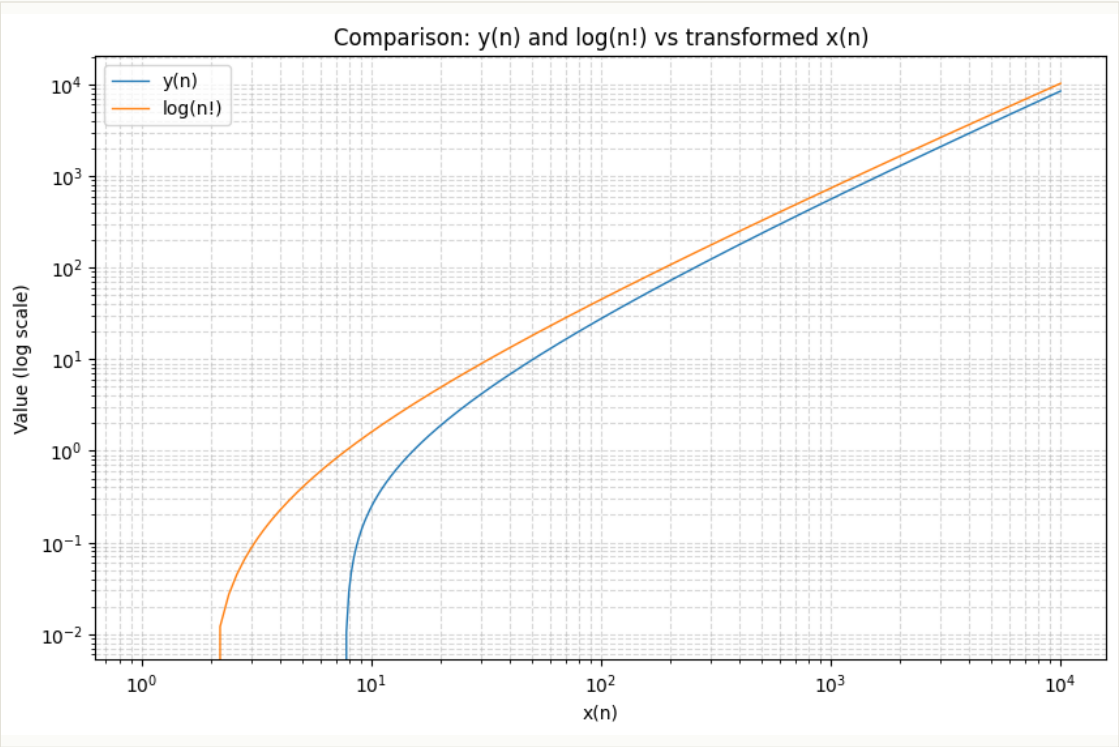
Measured savings are compared against the theoretical optimal savings to validate asymptotic optimality — including for multisets, where the analytical proof does not directly apply.

RESULTS

Savings approach the theoretical limit as N grows

Elements (N)	Total size	Measured savings	% of optimal
1,000	4,000 B	879 B	82.8%
10,000	40,000 B	12,187 B	85.4%
100,000	400,000 B	144,412 B	89.4%
1,000,000	4,000,000 B	1,523,686 B	92.1%

Time complexity scales close to linearly beyond ~50,000 rows, once the number of distinct elements (K ≈ 10,000) stabilises — matching the O(N log K) prediction.



CASE STUDY

MNIST: 160 images removed, 9,840 remain addressable

~12 KB recovered vs. **~14 KB** theoretical capacity

Random access to any of the 9,840 remaining images: **O(1)** random access

WRAPPING UP

Conclusions & future work

What was delivered

- A scalable, O(N log K) recursive permutation encoder
- Asymptotically optimal savings, proven and measured
- O(1) random access preserved on the remainder
- Fully composable with zstandard, gzip and similar

Future work

- Close the efficiency gap for small sequences
- Integration with columnar engines (DuckDB, Parquet)
- Data-type-specific tuning beyond arrays and file folders
- Tighten the worst-case strategy toward average-case

The central question — can order information compress permutable lists efficiently and at scale — is answered affirmatively.

THANK YOU

Questions & Discussion

Recursive Order-Based Compression for Permutable Lists

Bernardo Machado Ferreira, Nuno Escudeiro ISEP, Porto