The Price of Order in the Logarithmic Method

Sichen Wang
wsc@smbu.edu.cn
Shenzhen MSU-BIT University

Zhipeng Lu
zhipeng.lu@smbu.edu.cn
Shenzhen MSU-BIT University

Jingbang Chen*
chenjb@cuhk.edu.cn
CUHK-Shenzhen & SLAI

Abstract

The logarithmic method is a classical static-to-dynamic transformation: it stores one dynamic ordered set as several immutable static components and rebuilds them by merges. The same component-and-merge discipline underlies write-optimized ordered indexes, where cheap insertions must be reconciled with exact ordered queries. In this paper, we study the insertion-only version after n insertions, over abstract keys, in a strongly materialized merge-stack model with sequential component merges and one forward scan of the live components per query. We bound the product between the total amount of data written during the n insertions and the worst-case amount of data read by a single query, known as the *write-read product*. The optimal bounds are as follows:

- Membership and local certificates: $\Theta(n \log^2 n)$.
- Order and range queries with named keys or endpoints: $\Theta(n \log^3 n)$.
- Select: $\Theta(n^2)$.

Thus, the logarithmic method does not impose a universal dynamic overhead: under materialized one-way access, the optimum depends on what information the query reveals before the scan starts. This pinpoints the access-model obstruction behind the extra logarithm for exact order and range queries, and the quadratic barrier for select.

1 Introduction

A classical theme in dynamic data structures is to obtain dynamism by rebuilding static structures in batches. The logarithmic method of Bentley and Saxe keeps several static structures of different ages and merges them as updates arrive [BS80]; lower-bound and structural versions go back to Mehlhorn and Overmars [Meh81, Ove83]. This is also a natural abstraction of write-optimized ordered storage. The LSM-tree was introduced to make high-rate updates sequential and cheap [OCGO96]; sorted-run architectures appear in large-scale storage systems and key-value stores such as Bigtable, Cassandra, MyRocks/RocksDB, WiscKey, and PebblesDB [CDG+08, LM10, MDL20, LPG+17, RKCA17], and in cache-oblivious write-optimized search structures [BFCF+07].

The broad area is active because write optimization creates a persistent tension between update cost, point lookup cost, range-query cost, and space. The systems literature formulates this tension explicitly through the RUM tradeoff, optimizes merge policies and filter allocation for point lookups, and designs range filters or range-query mechanisms for LSM-style stores [AKM+16, DAI17, DI18, MLI25, ZLL+18, LCK+20, ZCWJ21]. From a theory point of view, however, these systems raise a more basic question: what does the logarithmic method itself do to the complexity of ordered-search queries?

*Corresponding author.

At a fixed time, a logarithmic-method structure is not one search tree. It is a set of independently built sorted components. A point lookup can often be certified locally in each component, for instance by a hash table or filter. An exact ordered query is different: predecessor, rank, and exact range queries must know where the query key, or the interval endpoints, fall inside the relevant components. Thus the issue is not only the number of components, but also what kind of information the query must recover inside each component.

Question. For each query family Q, what is the optimal product of amortized write cost and worst-case read cost for a logarithmic-method ordered set? Equivalently, if E is the total amount of data written and $\widehat{R}_Q$ is the worst read cost for Q, we study

$$P_Q(n) = \inf_{\mathcal{A}} \left(\frac{E_{\mathcal{A}}(n)}{n} \right) \widehat{R}_{\mathcal{A},Q}(n),$$

and throughout the paper we report the unnormalized product $E\widehat{R}_Q$.

This question is not answered by the closest existing lines of work. Classical static-to-dynamic transformations analyze the overhead of maintaining many static structures, but do not separate query families that all have logarithmic static search cost [BS80, Meh81, Ove83, MRYY21]. LSM and write-optimized-storage work optimizes concrete tradeoffs for point and range queries, but does not give a matching query-by-query lower-bound classification under a single merge history [AKM+16, DAI17, DI18, ZLL+18, LCK+20, ZCWJ21]. Fractional cascading explains how repeated searches across related catalogs can be shared [CG86, MN90, Afs21], but assumes that the cross-catalog links needed by the search are actually available. Lower bounds in cell-probe and external-memory models study different access regimes, where memory cells or blocks can be probed by address [FS89, BF03, Lar12, Păt11]. Our focus is the restricted but natural merge-stack regime in which components are written by sequential merges and later read once in chronological order.

The main obstruction is easy to see in a small example. Suppose the current set is stored in three sorted components C_1, C_2, C_3, scanned in that order. For membership of a key x, each component only has to certify whether x occurs there. For predecessor or rank of x, the query must learn the local gap of x inside each C_i. A search in C_1 does not reveal the gap in C_2 or C_3 unless some cross-component search shortcut has already been materialized. The natural algorithmic objection is fractional cascading: repeated searches in many sorted catalogs are precisely what fractional cascading was designed to remove [CG86], and dynamic variants maintain such links under updates [MN90, Afs21].

In a materialized one-way merge stack, however, the shortcut has the wrong time direction. The query scans older components before younger ones. A bridge into a younger component would have to be encountered in an older component, but the younger component did not exist when the older one was written. Rewriting the older component later changes its birth time and moves it behind the younger one in the scan. Thus the components remain independent sorted catalogs for the purpose of exact order localization.

We formalize this obstruction in a merge-stack model. An insertion-only ordered set is stored as immutable sorted *components*. Updates append singleton components and rebuild suffixes by sequential merges. Queries make one forward pass over the live components, from old to young, and cannot return to a component after passing it. Our main theorem is for *abstract keys*, which can be compared and hashed but carry no word-RAM coordinate revealing their rank, and for *strongly materialized* components, which are self-contained encodings of their own keys rather than directories for other components. These assumptions are the clean setting in which the missing fractional

cascade is exposed. The select query gives the forward scan even less information: it supplies only a rank K, not a target key value, and the threshold key may be in a component that has already passed under the head.

1.1 Our Results

The main result is a tight, query-sensitive classification of the logarithmic method in this model.

Theorem 1.1 (Main Classification)**.** *In the strongly materialized merge-stack model over abstract keys, with oblivious merge schedules and bounded-error randomized query evaluation, the optimal write-read product satisfies*

$$\min E\widehat{R}_Q = \begin{cases} \Theta(n \log^2 n), & Q \text{ is locally certifiable in each component,} \\ \Theta(n \log^3 n), & Q \text{ is a target-key order or exact range query,} \\ \Theta(n^2), & Q \text{ is select.} \end{cases}$$

Here the first line includes membership, point-emptiness, and global minimum/maximum. The second line includes predecessor, successor, rank, range-count, exact range-sum with unit weights, range-minimum, range-maximum, and bounded-interval emptiness.

The minimum ranges over all algorithms in the model. The upper bounds are explicit merge schedules with forward-readable static indexes. The lower bounds are worst-case over insertion sequences and query instances.

family	examples	information available before the scan	product
local certificates	membership, point-emptiness, global min/max	a key or constant-size certificate checked component by component	$\Theta(n \log^2 n)$
target-key order/range	predecessor, rank, range-count, exact range aggregates	key values or interval endpoints whose local gaps must be found	$\Theta(n \log^3 n)$
selection	select	only a rank K, with no target key value	$\Theta(n^2)$

Theorem 1.1 is not an answer-size separation. Membership, rank, and select all have $\Theta(\log n)$-bit answers on the hard instances, and rank and select are inverse operations. The difference is what the scan knows before it starts. Membership names a key and equality can be checked locally. Predecessor, rank, and range queries name key values, but exactness requires locating those values inside each component. Select names no key at all; the answer value is unknown while early components pass under the one-way scan.

The assumptions in Theorem 1.1 are also close to the boundary of the phenomenon. Integer keys reduce the cost of searching within one component; random access removes the no-backseek difficulty behind select; and non-materialized cross-component directories can reintroduce fractional-cascading information. We treat these variants after the main proof, in Section 7 and Appendix A. They are not part of the headline model, but they help separate which restriction is responsible for which line of the classification.

1.2 Overview

The proof has a simple architecture. For each query family we prove a read barrier at a fixed state, and then combine it with an accounting lemma for all suffix-merge histories.

For membership, the fixed-state barrier is the number of live components. Let $\widehat{W}$ be the maximum number of live components, and let $\widetilde{D}$ be the average number of times an item is written, including its initial write. The edit floor and the merge-forest counting lemma give

$$E = \Omega(n\widetilde{D}), \qquad \widetilde{D}\widehat{W} = \Omega(\log^2 n).$$

Some membership query must inspect all live components, so $\widehat{R} = \Omega(\widehat{W})$. This gives $E\widehat{R} = \Omega(n \log^2 n)$, matched by the balanced Bentley–Saxe schedule with local hashes.

For target-key order and range queries, the read barrier is the summed static search cost in the live components. At a state with component sizes m_i, we prove

$$\widehat{R}_{\mathrm{ord}} = \Omega\left(\sum_i \log(m_i + 1)\right).$$

The terms add because strong materialization forbids a read in one component from revealing the private search position of the query key in another component. The merge-history side then proves the size-diversity law

$$E \cdot \max_t \sum_{C \text{ live at } t} \log(|C| + 1) = \Omega(n \log^3 n).$$

This is the only extra logarithm in the middle line of Theorem 1.1. Once the predecessor lower bound is established, direct-sum arguments transfer it to rank, range-count, exact range-sum, range-minimum, range-maximum, and bounded-interval emptiness.

For select, the query is target-free. Consider a boundary in the scan with P stored keys before it and U stored keys after it. We construct a rank query whose answer is one of $\min\{P, U\}$ possible prefix keys, but the index of the right key is determined only by suffix information. Since the scan cannot go back, it must read

$$\Omega(\min\{P, U\})$$

words before crossing the boundary. If the read budget is small, nearly all live keys must repeatedly sit in one huge component; immutability then forces that component to be rebuilt many times. This yields $E = \Omega(n^2/\widehat{R})$ and hence $E\widehat{R} = \Omega(n^2)$.

Section 2 defines the model, costs, query families, and key assumptions. Section 3 proves the membership line. Section 4 proves the no-cascade read floor, the size-diversity law, and the order/range line. Section 5 proves the cut lower bound and the quadratic select line. Section 6 gives a general static-to-dynamic lifting principle for additive queries. Section 7 records access-model variants, and Appendix A studies cascades beyond strong materialization.

1.3 Related Works

The closest prior work explains pieces of the landscape: dynamization by merging, write-optimized storage, fractional cascading, and lower bounds in richer memory models. The following table summarizes why none of these directions already implies Theorem 1.1.

line of work	what it explains	difference from this paper
logarithmic method	how to maintain many rebuilt static structures	no classification separating query families with similar static costs
LSM and write-optimized storage	practical read/write/space tradeoffs for sorted runs	not a matching lower-bound theorem for membership, order/range, and select under one merge history
fractional cascading	how to share repeated searches across related catalogs	the useful bridge has the wrong chronological direction in a materialized one-way merge stack
cell-probe and external-memory lower bounds	tradeoffs under random access to cells or blocks	our restriction is not cell access but the order in which materialized information can be read and rewritten

Static-to-Dynamic Transformations. Bentley and Saxe introduced the logarithmic method for decomposable searching problems [BS80]. Mehlhorn proved lower bounds for such transformations [Meh81], and Overmars developed a systematic theory of dynamic structures built from static ones [Ove83]. Recent competitive dynamization studies merge policies under nonuniform inputs and read/write ratios [MRYY21]. These works explain the overhead of maintaining many static structures. They do not distinguish query types whose static problems are all logarithmic; the extra logarithm in Theorem 1.1 is caused by exact order localization across materialized components.

Write-Optimized Storage. The LSM-tree was introduced to make high-rate updates sequential and cheap [OCGO96]. LSM-style sorted runs appear in large-scale storage systems and key-value stores [CDG+08, LM10, MDL20, LPG+17, RKCA17]. The systems literature maps the read/update/storage tradeoff [AKM+16], optimizes merge policies and filter allocation for point lookups [DAI17, DI18, MLI25], and builds succinct or probabilistic range filters and range-query mechanisms [ZLL+18, LCK+20, ZCWJ21]. This work motivates the access discipline, but it does not prove a tight query-by-query product classification under the same merge histories.

Fractional Cascading. Fractional cascading removes repeated binary searches across related catalogs in static structures [CG86]; dynamic fractional cascading studies how to maintain such links under updates [MN90, Afs21]. Our obstruction is orthogonal. The issue is not only the cost of updating a cascade, but whether the bridge can appear before the component it accelerates is created. In the strongly materialized one-way model it cannot; Appendix A examines what changes when such cross-component data are allowed.

Lower Bounds and Restricted Access. External-memory dictionary lower bounds study update/query tradeoffs under random block access [BF03, DIN22]. Cell-probe lower bounds for dynamic data structures, including dynamic range counting, use a more powerful random-access memory model [FS89, Lar12, Păt11, Ko25]. Our bounds are orthogonal: component encodings are static and transparent, but the read and rewrite order is restricted. The lifting theorem in Section 6 connects our merge-stack bounds to static predecessor and range-searching lower bounds [PT06, PT07, Păt07, CW13]. The merge-history accounting is closest in spirit to list labeling and file maintenance [IKR81, BKS12, BCFC+22, BCFC+24]; the select lower bound is related to selection under restricted access [MP80, FJ84, KKZZ19, BDHK23].

2 Preliminaries

This section fixes the model used in the main classification. The objects are merge histories of immutable sorted components, the costs are total rewritten mass and prefix-worst read cost, and the key assumptions isolate the absence of cross-component order information.

2.1 Merge Histories and Components

A *merge stack* maintains a sequence of immutable sorted components $C_1, \ldots, C_W$, oldest at the bottom. Each component stores a sorted block of key-value samples together with any static index built from that block alone. Three operations maintain it:

- *insert*(x, y) pushes a new singleton component $\{(x, y)\}$ on top;
- *merge* pops a suffix $C_i, \ldots, C_W$ and pushes a single component holding the union of their samples in sorted order;
- *query* makes one forward pass over $C_1, C_2, \ldots$ in stack order, reading component data; after an $O(1)$ header it may skip a component's body, but it never returns to an earlier component.

Inserts and merges interleave according to a schedule of the algorithm's choice. The main lower bounds consider schedules that are oblivious to the key values; this condition is part of strong materialization below.

2.2 Cost Measures

The *write cost* E is the total component data created or rewritten over the execution. The read cost of a query is the data it reads, and $\widehat{R}$ is its prefix-worst value: the maximum, over every time t and every query, of the cost of answering that query against the state after step t. Making $\widehat{R}$ prefix-worst keeps the read cost from being amortized away by periodic global rebuilds, since every intermediate state must remain queryable. For a randomized evaluator the cost of a query is its expected number of charged reads, and $\widehat{R}$ is again prefix-worst. The complexity measure throughout is the product $E\widehat{R}$.

A token is a machine word of $w = \Theta(\log n)$ bits, and keys and values are drawn from universes of size $\mathrm{poly}(n)$, so a sample occupies $O(1)$ tokens and a component of m samples occupies $\Theta(m)$. We use the following notation.

symbol	meaning
E	total rewritten component data
$\widehat{R}_Q$	prefix-worst read cost for query family Q
$\widehat{W}$	maximum number of live components over the execution
$\widetilde{D}$	average item write depth, including the initial write
L	$\log_2 n$
λ	$\log_2 \log_2 n$

2.3 Query Families

Three query types drive the paper. *Membership* is the exact-match point query: given x, return its value if x is live and a default otherwise. It is locally certifiable because equality can be settled inside each component by a hash or perfect dictionary. *Predecessor* is the basic target-key order

query: given x, return the value of the largest live key at most x. Successor, rank, and exact range queries – count, sum, minimum, maximum, and emptiness over an interval – are target-key order or range queries of the same kind, since each must locate a key value or endpoint in the live order. Membership and these order queries are *decomposable*: an answer combines fixed per-component contributions. *Select* is different. It is given a rank K and returns the K-th smallest live key; no target key value is supplied to the scan. Inserted keys are distinct, and the lower bounds use distinct-key instances.

2.4 Key Models and Materialization

Keys come in two models, and the order separation depends on which one is used. An *abstract key* is the model for string, tuple, or opaque record keys: it is hashable and comparable, but carries no integer structure. We read it through an oracle. A key has an opaque equality label and an unknown position in the total order, and neither the label nor its hash reveals that position. A single read returns at most one stored key handle together with $O(1)$ control words, and the handle's rank among those already read is learned only by comparison. The encoder cannot manufacture an order-bearing separator it has not read, and an answer must be a key or value decoded from the words read, not a pointer into an unread cell. The $O(1)$ control words therefore cannot stand in for the order of a whole block. An *integer key* is an integer in a poly(n) universe, with the word-RAM available: a van Emde Boas tree or y-fast trie locates the predecessor inside a component in $O(\lambda)$ reads, independent of its size, and the order gap narrows accordingly (Section 7).

The main theorems hold in executions realized by size-based log-structured merging. Ordinary materialization, which constrains only a component's contents, is not enough: once the merge schedule may depend on the data, the partition into components itself carries order information across them, and an unread block is no longer free for an adversary to complete. We use the following stronger condition.

Definition 2.1 (Strong Materialization)**.** An execution is *strongly materialized* if it satisfies the following.

- *Canonical updates*: after each insertion the active components partition the inserted samples into contiguous blocks of consecutive insertion times, and each insertion performs at most one merge, replacing a suffix of the active components and the new singleton by a single component.
- *Oblivious schedule*: the merged suffix at each step depends only on the step index and the current component sizes, not on the keys, values, query answers, or any coins.
- *Exact layout*: a component of m samples occupies a publicly fixed length $\ell(m) \in [c_0 m, c_1 m]$, so component boundaries, addresses, and the merge structure are functions of the size sequence alone.
- *Local encoding*: a component's tokens depend only on its own samples, their insertion order and values, and comparisons among its own keys; inputs that agree on every component's internal order, labels, values, and insertion history, and differ only in how keys of distinct components interleave, induce identical component encodings.
- *No external state*: no sample-dependent state persists outside the components, the query evaluator starts from a data-independent configuration with coins independent of the data, and every dependence on the data passes through reads of component tokens.

This is what an LSM run already is, a self-contained sorted file; Appendix A removes the condition and lets a young component share or copy order information from older ones.

The per-component arguments use one consequence. Call an execution *no-cascade* if a read in one component cannot shrink another component's set of candidate answers. Strong materialization supplies it: changing one block leaves the partition, every other component's encoding, and the layout untouched, so a read elsewhere says nothing about that block. A non-materialized global key order, searchable by random access, would violate this, the components then admitting a joint search in $O(\log n)$ reads; random access alone, with the components still materialized, does not. This locality is what Appendix A gives up.

The canonical-updates clause is without loss of generality: the merges performed between two insertions act on nested suffixes of the stack and coalesce into their final suffix merge, and collapsing them removes the intermediate writes and queryable states, so neither E nor $\widehat{R}$ grows. Under exact layout the write cost is structural rather than informational: building a component of m samples costs $\Theta(m)$ tokens whatever it holds, a fact the edit floor of Section 3 reads off directly.

2.5 Serialization and Forward Reads

The lower bounds are cleaner in a low-level view of the same execution. Serialize the components in stack order into a single token string, each component a contiguous block preceded by a length-prefix header. Edits become tail operations: an insert pushes one block at the tail, and a merge of a stack suffix pops the most recently pushed blocks and pushes one new block. Writing LCP for the longest common prefix, the edit $s_{t-1} \to s_t$ costs

$$d_{\mathrm{LIFO}}(s_{t-1}, s_t) = \big(|s_{t-1}| - |\mathrm{LCP}|\big) + \big(|s_t| - |\mathrm{LCP}|\big),$$

the number of tokens popped and pushed at the tail, so $E = \sum_t d_{\mathrm{LIFO}}(s_{t-1}, s_t)$; we call such tail-only edits *LIFO*. A query is run by a *forward, no-backseek* head: it starts at position 0 and at each step either reads the current token, at unit cost, advancing one place, or skips forward at no cost by an amount it computes from its finite control, the query, and the tokens read so far. It never moves backward, working memory is free, and only reads are charged.

Lemma 2.2 (Serialization Equivalence)**.** *For strongly materialized executions, the merge-stack model and the LIFO-token, forward-no-backseek model agree up to constant factors in* $(E, \widehat{R})$.

Proof. Serialize as above. An insert pushes a block of $\Theta(1)$ tokens at the tail, and a merge pops exactly the most recently pushed blocks and pushes one new block, so every edit is tail-only; the pushed token mass equals the rewritten component data up to the matching pops, a constant factor, so $E_{\mathrm{token}} = \Theta(E)$. The components occupy the string in stack order, which is birth order and the order a forward head reads them, and skipping a body after its header is a legal no-backseek pass, so $\widehat{R}_{\mathrm{token}} = \Theta(\widehat{R})$. Conversely, a tail-only token execution whose pushed blocks are local encodings of the unions of the popped blocks splits into contiguous blocks in birth order, since the tokens pushed at one step are consecutive and share a birth time, and is a merge-stack execution with the same merge structure. □

One may therefore keep the merge-stack picture throughout and read the lower-bound proofs as statements about the token string.

2.6 Baseline Upper Bounds

The serial Bentley–Saxe transform [BS80] supplies the upper bounds the lower bounds will match.

Proposition 2.3 (Bentley–Saxe Baseline)**.** *A single schedule has $E = O(n \log n)$ with at most $\lceil \log_2(n+1) \rceil$ active components at all times, and answers membership in $\widehat{R} = O(\log n)$, abstract-key predecessor in $\widehat{R} = O(\log^2 n)$, and integer predecessor in $\widehat{R} = O(\lambda \log n)$. The corresponding products are*

$$E\widehat{R} = O(n \log^2 n), \qquad O(n \log^3 n), \qquad O(n \log^2 n\, \lambda).$$

Proof. Keep the components whose sizes are the set bits of the insert count, merging equal-sized ones on each insert. A component of size 2^k is built at most $n/2^k$ times, each at $\Theta(2^k)$ tokens, so $E = O(n \log n)$, and at most $\lceil \log_2(n+1) \rceil$ components are live at any time. Each component carries a forward-readable static structure, length-prefixed so the head skips its body after one header read. A static perfect hash answers membership in $O(1)$ reads per component, even against an adversarial query, so $\widehat{R} = O(\log n)$. For abstract-key predecessor, a balanced search tree laid out in search-round order, all depth-one nodes, then all depth-two, and so on, is traversed root to leaf as a forward scan of $O(\log m_k)$ comparison reads in component k, summing to $O(\log^2 n)$ over the active components. For integer predecessor, a y-fast trie answers in $O(\lambda)$ reads per component under the same round-by-round layout, giving $\widehat{R} = O(\lambda \log n)$. □

The membership and predecessor products are the first the lower bounds match (Sections 3 and 4); select needs a different schedule and is polynomially larger (Section 5). How the predecessor bound responds when these conditions are relaxed, and why the relaxations are not interchangeable, is the subject of Section 7.

3 Membership

The Bentley–Saxe schedule of Section 2 answers membership with product $n \log^2 n$. We show this is best possible, and in doing so establish the three structural facts the later lower bounds reuse: a level decomposition that the tail-only discipline forces, an edit floor on the write cost, and a counting bound that trades depth against frontier. Together they expose the mechanism the paper turns on, that $E\widehat{R}$ factors into a write budget and a sum of per-level read charges. Membership is the case where a hash holds every charge at $O(1)$; order and rank raise it.

3.1 Merge Forests

Fix an execution. A live token in s_t has a *birth step*, the step at which it was last pushed, and tokens with a common birth step form a *level*; by Lemma 2.2 a level is just a component of Section 2 in token-string form. Write $W(t)$ for the number of live levels at time t.

Lemma 3.1 (Level Decomposition)**.** *At every time t the live levels are contiguous and ordered by birth, $s_t = L_{t_1} \cdots L_{t_W}$ with $t_1 < \cdots < t_W$, and the edit $s_{t-1} \to s_t$ pops a suffix of them and pushes one new top level.*

Proof. A canonical update pops a suffix of the active components and pushes one merged component (Section 2). The tokens pushed at one step are consecutive at the tail and share a birth step, so each level is contiguous and older levels lie earlier; an edit pops a suffix of whole levels and makes the pushed tokens the new top level. □

The decomposition makes the execution a *merge forest*: an internal node for each step that merges, its children the levels popped there and the singleton inserted then, and a leaf for each insert. Leaf j takes part in d_j merges, one for each of its internal ancestors. The rewritten sample mass is therefore

$$M := \sum_j d_j = \sum_P |P|,$$

$|P|$ being the number of leaves below the internal node P. The forest is fixed by the schedule alone, whatever the tokens mean, and the lower bounds read off it.

The hard instances carry their entropy in the values: each inserted value is independent and uniform over a $\mathrm{poly}(n)$ universe, so it is $\Theta(\log n)$ bits, the content of one token.

Lemma 3.2 (Incompressibility)**.** *On the i.i.d.-value instance, a materialized level that is the sole source for m of its samples occupies $\Omega(m)$ tokens.*

Proof. The state must determine those m values, whose joint entropy is $\Theta(m \log n)$ bits, or $\Theta(m)$ tokens. Materialization places them in this level alone, so it occupies $\Omega(m)$ tokens. □

Corollary 3.3 (Edit Floor)**.** *With* augmented depth $\widetilde{D} := (M+n)/n$*, the write cost is $E = \Omega(n\widetilde{D})$.*

Proof. Under exact layout a level of m samples costs $\Theta(m)$ tokens, so $E = \Theta(n+M) = \Theta(n\widetilde{D})$. With no layout fixed, Lemma 3.2 applies to each level when it is built, the sole source for its samples at that moment, and summing over the levels built, of total size $n+M$, gives $E \geq c(n+M) = cn\widetilde{D}$ for a constant $c \in (0,1]$. □

The third fact ties depth to frontier. Its counting argument is the online-labeling lemma of Bulánek, Koucký, and Saks [BKS12], recast in merge-forest terms. Write $\widehat{W} := \max_t W(t)$ for the *frontier width*.

Lemma 3.4 (Frontier Width)**.** *Every merge forest on n leaves with augmented depth $\widetilde{D}$ and frontier width $\widehat{W}$ has $\widetilde{D}\widehat{W} = \Omega(\log^2 n)$.*

Proof. With a dummy root above the final frontier, a leaf in d_j merges has augmented depth $d_j + 1$, of mean $\widetilde{D}$. Let $N(h, W)$ be the most leaves an ordered suffix-merge tree of height at most h can have while its live frontier never exceeds W. Its root pops a suffix of $r \leq W$ children, each a completed subtree that holds a frontier slot until the merge, together with the singleton inserted then, a height-0 child holding no slot. With $N(0, W) = 1$,

$$N(h,W) \;\leq\; 1 + \sum_{q=1}^{W} N(h-1,q) \;\leq\; \sum_{q=0}^{W} \binom{h-1+q}{q} \;=\; \binom{h+W}{W},$$

the singleton being the $q = 0$ term and the last step the hockey-stick identity. For $a \leq b$, $\log_2 \binom{a+b}{a} \leq a \log_2 \frac{e(a+b)}{a} = O(\sqrt{ab})$ using $\log_2 u \leq 2\sqrt{u}$. By Markov at least $n/2$ leaves have augmented depth at most $2\widetilde{D}$; keeping them and suppressing empty internal nodes leaves an ordered tree with at least $n/2$ leaves, height at most $\lceil 2\widetilde{D} \rceil$, and frontier at most $\widehat{W}$, so

$$\log_2 \tfrac{n}{2} \;\leq\; \log_2 \binom{\lceil 2\widetilde{D} \rceil + \widehat{W}}{\widehat{W}} \;=\; O\big(\sqrt{\widetilde{D}\widehat{W}}\big),$$

that is $\widetilde{D}\widehat{W} = \Omega(\log^2 n)$. □

3.2 The Membership Bound

The read side needs one fact: a forward head must visit every live level.

Lemma 3.5 (Sequential Access)**.** *For an oblivious schedule, some strongly materialized realization forces $\widehat{R} \geq \widehat{W}$ on a fresh membership query against a deterministic evaluator, and $\widehat{R} \geq \widehat{W}/3$ against a bounded-error one.*

Proof. At a time of maximum frontier, take the realization in which every live level straddles a fresh key x matching none of its samples. Each live level must be decided on its own, that x is absent there, and under strong materialization no other level constrains it, so a level left unread can be filled with x by the adversary, flipping the answer. A deterministic evaluator therefore reads all $\widehat{W}$ levels, so $\widehat{R} \geq \widehat{W}$. For a bounded-error one, couple this input with the one in which only level i gains x: the transcripts agree until level i is read, so skipping it makes the evaluator answer both alike and err on one, and the two error probabilities sum to at least the probability of skipping level i. Bounded error keeps that sum below $\frac{2}{3}$, so level i is read with probability at least $\frac{1}{3}$ and $\widehat{R} \geq \widehat{W}/3$. A length header does not escape the charge, that read being the one counted. □

Theorem 3.6 (Tight Membership Bound)**.** *Strongly materialized membership has $E\widehat{R} = \Theta(n \log^2 n)$, attained at $(E, \widehat{R}) = (\Theta(n \log n), \Theta(\log n))$ by hashing on the Bentley–Saxe schedule.*

Proof. The upper bound is Proposition 2.3. For the lower bound, Lemma 3.5 gives $\widehat{R} \geq \widehat{W}$, Corollary 3.3 gives $E = \Omega(n\widetilde{D})$, and Lemma 3.4 gives $\widetilde{D}\widehat{W} = \Omega(\log^2 n)$, so $E\widehat{R} = \Omega(n\widetilde{D}\widehat{W}) = \Omega(n \log^2 n)$. □

Nothing in the lower bound used what membership computes, only that every live level must be decided, so $n \log^2 n$ is a floor for the whole model, reached by membership. It is robust to randomization through Lemma 3.5, and it constrains the tradeoff and not just the product: $\widehat{R} = O(\log n)$ forces $\widehat{W} = O(\log n)$, hence $\widetilde{D} = \Omega(\log n)$ and $E = \Omega(n \log n)$. Membership is the hashable extreme, each level settled by one probe. The rest of the paper keeps this write budget and asks whether the read side can stay at $O(1)$ per level; once the query needs order, it cannot.

4 Order Queries

This section proves the middle line of Theorem 1.1. The proof has three steps. First, strong materialization and one-way access rule out the cross-component shortcut that fractional cascading would need. Second, once the components cannot share searches, a fixed state with component sizes m_i forces $\Omega(\sum_i \log(m_i + 1))$ reads for a hard predecessor query. Third, every cheap suffix-merge history exposes some state where this summed log-size is large, giving the product lower bound $\Omega(n \log^3 n)$. The final subsection transfers predecessor to rank and exact range queries.

4.1 No Cascading Across Components

Membership spends $O(1)$ per level because a hash names where the answer sits. Order has no such shortcut and cannot build one by linking levels. Call a *forward bridge* from a level L_i to a younger level L_j any data stored in L_i that lets the head, once it has read L_i, reach the query's predecessor in L_j within $O(1)$ further reads.

Lemma 4.1 (Causal-Bridge Obstruction)**.** *Under forward, no-backseek access with LIFO edits, no forward bridge can be consulted when it would help.*

Proof. Levels are read oldest first (Lemma 3.1). A bridge that speeds up a younger level L_j must be read before L_j, hence stored in an older level L_i; but its content depends on L_j, fixed only when L_j is built, so its carrier must be younger than L_j. The two demands collide: when L_i was built L_j did not yet exist, and on adversarial independent keys no guess survives. Rebuilding L_i after L_j only pushes its birth past L_j, moving it behind in read order, and a younger level pointing into an older one is reached only once the older target has been passed. □

This is the obstruction behind Afshani's dynamic fractional-cascading lower bound [Afs21], sharpened from a tax to an impossibility: the bridge a cascade needs would have to be written before its target exists. The other route, a young level carrying a directory into the older ones, is closed not by access but by strong materialization, each component encoding only its own samples (Section 2); Appendix A drops that assumption. What remains is the cost of searching one level alone.

Within a level a hash still names one specific key, while the predecessor is another, fixed by the level's order, that the hash of x does not reach. Locating it is a comparison search, and on a level of m keys an adversarial query forces $\Omega(\log m)$ reads, made precise against a hard gap distribution in Lemma 4.2.

4.2 Per-State Search Lower Bound

Fix a time and write $\Phi(t) = \sum_{\text{active } L} \log_2 |L|$ for the summed log-size of the live levels. Since the per-level searches do not combine (Lemma 4.1), the read cost is their sum. This sum is not forced at every state: with the live levels in disjoint key ranges a single header settles all but one. What does hold is that each size skeleton, the live levels' sizes alone, admits an adversarial key assignment making the sum necessary, which under an oblivious schedule is all the lower bound needs.

Lemma 4.2 (Zero-Side Ordered Gap)**.** *Fix stored keys $k_1 < \cdots < k_m$ and handles $b < x$. For $0 \le j \le m$ let Z_j place $k_1, \dots, k_j < b < x < k_{j+1}, \dots, k_m$, and for $0 \le j < m$ let H_j place $k_1, \dots, k_j < b < k_{j+1} < x < k_{j+2}, \dots, k_m$, so the predecessor of x is b on Z_j and k_{j+1} on H_j. Any randomized comparison tree reads $\Omega(\log(m+1))$ keys in expectation on Z_J, J uniform, if it uses the known internal order of the k_r, learns the order of an unread key only by reading it, and errs with probability at most $\frac{1}{3}$ over $\{Z_j, H_j\}$.*

Proof. Take a deterministic tree first. A read key has three relevant outcomes, $k_r < b$, $b < k_r < x$, and $x < k_r$. Let E_0 count the zero instances not answered b, E_1 the hot instances answered b, and Λ the distinct b-leaves reached by correctly answered zero instances. The zero instances reaching one leaf form an interval: if Z_a and Z_c ($a < c$) share a b-leaf then no k_r with $a < r \le c$ was read, being above x in Z_a and below b in Z_c, so every H_h with $a \le h < c$ follows the same transcript and is wrongly answered b. A leaf covering z zero instances thus forces $z - 1$ hot errors, whence $\Lambda \ge (m+1) - E_0 - E_1$. Average error at most $\frac{5}{12}$ over the uniform mixture of the $2m+1$ instances gives $E_0 + E_1 \le \frac{5}{12}(2m+1)$ and $\Lambda \ge (m+1)/6$. With one correct zero instance per leaf, at depths $d_1, \dots, d_\Lambda$, the ternary Kraft inequality $\sum_\ell 3^{-d_\ell} \le 1$ forces mean depth $\ge \log_3 \Lambda$, so

$$\mathbb{E}_J[\text{reads on } Z_J] \;\ge\; \tfrac{\Lambda}{m+1} \log_3 \Lambda \;=\; \Omega(\log(m+1)).$$

For a randomized tree the coin-fixed error averages at most $\frac{1}{3}$, so by Markov a $\frac{1}{5}$ fraction of the fixings have error at most $\frac{5}{12}$, and averaging over them keeps $\Omega(\log(m+1))$. □

Lemma 4.3 (Anchor-and-Zones Floor). *Fix a public size vector $(m_1, \dots, m_W)$ of an oblivious schedule and an evaluator of error at most $\frac{1}{3}$. Some strongly materialized state with these sizes admits a query x on which the evaluator has $\widehat{R} \geq c \sum_{i=1}^{W} \log_2(m_i + 1)$, for an absolute $c > 0$.*

Proof. We average over a distribution of states, so some realization meets the expectation, and obliviousness makes it carry the prescribed sizes. Take level 1 as anchor: fix its internal order, draw the query gap uniformly among the m_1 gaps with a stored lower endpoint, and put x there with b that endpoint. Locating b is a randomized predecessor search among equally likely gaps, one ternary split per read, so a bounded-error evaluator spends $\Omega(\log(m_1 + 1))$ reads, and an $O(1)$ header holds only constantly many handles and cannot help. For $i \geq 2$ fix three private zones, a low slab below every anchor key, a high slab above x, and a hot subzone inside (b, x), all ordered publicly by level index. Draw J_i uniform in $\{0, \dots, m_i\}$; the zero placement puts $k_{i,1}, \dots, k_{i,J_i}$ in the low slab and the rest in the high slab, and the hot alternative, when $J_i < m_i$, moves k_{i,J_i+1} into the hot subzone. Expose all outside information for free, b included. Reading $k_{i,r}$ then reveals only its zone, comparisons within the level giving the known index order and comparisons across levels only the public zone order, so nothing finer about J_i is learned without reading level i. The task is Lemma 4.2 with $m = m_i$ and inherits its error guarantee, costing $\Omega(\log(m_i + 1))$. Summing over the anchor and the levels gives the bound. □

The floor is existential over the states realizing a skeleton, not a property of every state; specialized to one read per level it is the randomized membership floor $\widehat{R} \geq \widehat{W}/3$ of Theorem 3.6. On the hard realization $\widehat{R} \geq c \sum_k \log_2(m_k + 1) \geq c \max\{\widehat{W}, \Phi(t)\}$, and the separation turns on whether a schedule can hold Φ down throughout while editing for only $O(n \log n)$. It cannot.

4.3 Merge-History Size Diversity

A logarithm above membership needs $\Phi(t) = \Omega(\log^2 n)$ at some moment, and no schedule holds Φ at the membership scale $O(\log n)$ throughout. The size-diversity law prices the time-averaged potential from below by the edit budget. It is an entropy-prefix argument on the merge forest, budget-free, the budget entering only through the augmented depth $\widetilde{D}$.

Add a dummy root above the final frontier, with $S_{v_0} = n$. For an internal node v write S_v for the leaves below it, $q_v = S_v/n$, and $z_v = \log_2 S_v$; its children in birth order have sizes $s_1, \dots, s_r$, weights $p_i = s_i/S_v$, entropy $h_v = \sum_i p_i \log_2(1/p_i)$, and *ordered prefix load* $a_v = \sum_j p_j \sum_{i<j} \log_2 s_i$. A merge coalesces an active suffix, so $r \leq \widehat{W} + 1$.

Lemma 4.4 (Four Identities). *For every merge forest, exactly and with no budget hypothesis,*

$$\text{(I) } \frac{1}{n}\int_0^n \Phi = \sum_v q_v a_v, \quad \text{(II) } \sum_v q_v h_v = L, \quad \text{(III) } \sum_v q_v h_v z_v \geq \tfrac{1}{2} L^2, \quad \text{(IV) } \sum_v q_v = \widetilde{D}.$$

Proof. For (I), read the integral as a sum over the intervals between insertions; it is the lifetime identity $\int_0^n \Phi = \sum_L \tau_L \log_2 |L|$, where τ_L, the inserts during L's life, equals the mass of L's younger siblings at its parent, since a LIFO merge carries off exactly the levels younger than L. As $S_v a_v = \sum_i (\sum_{j>i} s_j) \log_2 s_i = \sum_i \tau_{v_i} \log_2 s_i$, summing over v and dividing by n gives (I). For (II) and (III), sample a leaf uniformly and let $Z_0 = L > Z_1 > \cdots \geq 0$ be the log-sizes along its root-to-leaf path, with $X_k = Z_k - Z_{k+1}$ and $\sum_k X_k = L$. At a node v on the path $\mathbb{E}[X_k \mid v] = h_v$ and $\Pr[v \text{ on path}] = q_v$, giving (II); and $\sum_k Z_k X_k = \frac{1}{2}(\sum_k X_k)^2 + \frac{1}{2}\sum_k X_k^2 \geq \frac{1}{2} L^2$ with $\mathbb{E}[Z_k X_k \mid v] = z_v h_v$, giving (III). For (IV), $\sum_v q_v = \frac{1}{n} \sum_{\text{leaves}} (d_j + 1) = \widetilde{D}$, since each leaf has $d_j + 1$ internal ancestors. □

Identity (III) is the engine: entropy weighted by current log-size unavoidably reaches $\Omega(L^2)$, exactly. Turning it into a bound on (I) is a local computation.

Lemma 4.5 (Local Prefix Bound)**.** *Fix $\varepsilon \in (0, \frac{1}{4})$, $\gamma = \varepsilon^2$, $c_0 = L^{-7}$, and write $\mu = G^{-1}$ for $G(m) = (m+1)\log_2(m+1) - m\log_2 m$. Uniformly over internal nodes with $z_v \geq \varepsilon L$, $h_v \geq c_0$, and $r \leq n^\gamma + 1$,*

$$a_v \;\geq\; (1-\kappa)\, z_v\, \mu(h_v), \qquad \kappa \leq \varepsilon + o(1).$$

Proof. Put $\theta = (\log_2 r + 9\log_2 L)/(\varepsilon L)$, so $\theta = \varepsilon + o(1)$ under the hypotheses. Call a child *good* if $p_i \geq 2^{-\theta z_v}$, so $\log_2 s_i = z_v + \log_2 p_i \geq (1-\theta) z_v$. A bad child has $p_i < 2^{-\theta\varepsilon L} = (rL^9)^{-1}$, so the bad mass is at most L^{-9}, and renormalizing onto the good children shifts the entropy by $O(L^{-8})$ and hence μ by a factor $1 - o(1)$: since $(\ln \mu)' = O(1 + 1/h)$ and $h_v \geq c_0$, the shift in $\ln \mu$ is $O((1+L^7)L^{-8}) = O(L^{-1})$. The maximum-entropy bound, that a law on $\{0, 1, 2, \dots\}$ of entropy h has mean at least $\mu(h)$, applied to a child drawn by p_j, gives at least $(1 - o(1))\mu(h_v)$ expected earlier good children; each adds $\geq (1-\theta) z_v$ to a_v, so $a_v \geq (1 - \theta - o(1)) z_v \mu(h_v)$. □

Theorem 4.6 (Size-Diversity Law)**.** *Every LIFO merge forest with $\widehat{W} \leq n^\gamma$ and $\widetilde{D} \leq L^6$ satisfies*

$$\tfrac{1}{n}\int_0^n \Phi(t)\, dt \;\geq\; (1 - \varepsilon - o(1))\, \widetilde{D}\, L\, \mu\Big(\tfrac{\beta L}{\widetilde{D}}\Big), \qquad \beta = \tfrac{1}{2} - \varepsilon - o(1),$$

so $\max_t \Phi(t)$ is at least the same value.

Proof. Discard from (III) the nodes with $z_v < \varepsilon L$, contributing at most $\varepsilon L \sum_v q_v h_v = \varepsilon L^2$ by (II), and those with $h_v < c_0$, contributing at most $c_0 L \widetilde{D} = o(L^2)$ by (IV). The survivors have $z_v \geq \varepsilon L$, $h_v \geq c_0$, $r \leq \widehat{W} + 1 \leq n^\gamma + 1$, and carry $\sum^{\mathrm{s}} q_v h_v z_v \geq \beta L^2$. Write $Y_v = q_v h_v z_v / L^2$, so $m_Y := \sum^{\mathrm{s}} Y_v \geq \beta$ and $\sum^{\mathrm{s}} Y_v / h_v = L^{-2} \sum^{\mathrm{s}} q_v z_v \leq \widetilde{D}/L$ by (IV). Set $\varphi(u) = u\,\mu(1/u)$, convex and decreasing, and note $q_v z_v \mu(h_v) = L^2 Y_v \varphi(1/h_v)$. Identity (I) and the local prefix bound give

$$\tfrac{1}{nL^2}\int \Phi \;\geq\; (1-\kappa)\, m_Y\, \mathbb{E}[\varphi(1/h)],$$

the expectation under Y/m_Y. Jensen and then the monotonicity of φ at $\mathbb{E}[1/h] \leq \widetilde{D}/(L m_Y)$ give

$$m_Y\, \mathbb{E}[\varphi(1/h)] \;\geq\; m_Y\, \varphi\Big(\tfrac{\widetilde{D}}{L m_Y}\Big) \;=\; \tfrac{\widetilde{D}}{L}\, \mu\Big(\tfrac{m_Y L}{\widetilde{D}}\Big) \;\geq\; \tfrac{\widetilde{D}}{L}\, \mu\Big(\tfrac{\beta L}{\widetilde{D}}\Big),$$

using $m_Y \geq \beta$ and μ increasing. Multiply by L^2. □

The frontier cap is necessary for the law, since the never-merge schedule has $\Phi \equiv 0$, but it does not constrain the predecessor bound: the assembly below routes wide-frontier executions through the read floors instead. The cap aside, the law is the right general form rather than a relaxation, asymptotically tight along the lazy base-q counter, so the uniform reading $\max_t \Phi \geq c \log^2 n$ for all schedules is false, that family sending $\max_t \Phi / \log^2 n \to 0$ as q grows. Since Φ depends only on the forest, the bound holds for every realization of a randomized algorithm.

Lemma 4.7 (Calculus Floor)**.** $\min_{h>0} \mu(h)/h^2 = \frac{1}{4}$*, attained only at $h = 2$; equivalently $G(m) \leq 2\sqrt{m}$ with equality iff $m = 1$. Hence $\widetilde{D}^2\, \mu(\beta L/\widetilde{D}) \geq (\beta^2/4) L^2$.*

Proof. Substituting $m = \mu(h)$, the claim $\mu(h) \geq h^2/4$ reads $G(m) \leq 2\sqrt{m}$. Let $F(m) = 2\sqrt{m} - G(m)$, so $F(0^+) = F(1) = 0$ and $F'(m) = m^{-1/2} - \log_2(1 + 1/m)$, of the sign of $1 - u(m)$ for $u(m) = \sqrt{m}\log_2(1 + 1/m)$. Here u is unimodal, with $u'(m) = 0$ at a single point since $(1 + 1/m)^{m+1}$ decreases strictly to e, and with $u \to 0$ at both ends and $u(1) = 1$ it exceeds 1 on exactly one interval $(m_1, 1)$. So F rises on $(0, m_1)$, returns to $F(1) = 0$, then rises, giving $F \geq 0$ with equality only at $m \in \{0, 1\}$. The lone interior contact $m = 1$ gives $h = G(1) = 2$ and $\mu(2) = 1$, so the minimum is $\frac{1}{4}$; substituting $h = \beta L/\widetilde{D}$ gives the last inequality. □

4.4 The Predecessor Bound

Theorem 4.8 (Tight Predecessor Bound)**.** *Every strongly materialized algorithm with a bounded-error evaluator, under no hypothesis on the frontier $\widehat{W}$ or the budget M, has predecessor $E\widehat{R} = \Omega(n \log^3 n)$, which binary Bentley–Saxe attains. Predecessor is $\Theta(n \log^3 n)$, one clean logarithm above membership.*

Proof. The upper bound is Proposition 2.3. Fix the algorithm; its oblivious schedule fixes the size trajectory and E independently of the keys. Fix $\varepsilon = 10^{-2}$, $\gamma = \varepsilon^2$, $\widetilde{D} = (M + n)/n$, and split on the schedule.

(i) $\widehat{W} > n^\gamma$. Lemma 4.3 at a maximum-frontier skeleton gives $\widehat{R} = \Omega(\widehat{W}) > \Omega(n^\gamma)$, and $E \geq n$, so $E\widehat{R} = \Omega(n^{1+\gamma}) \gg n \log^3 n$.

(ii) $\widehat{W} \leq n^\gamma$, $\widetilde{D} > L^6$. The edit floor gives $E = \Omega(n\widetilde{D}) > \Omega(nL^6)$ and Lemma 4.3 at the final skeleton, where $\sum_k \log_2(m_k + 1) \geq \log_2(n + 1)$, gives $\widehat{R} = \Omega(L)$, so $E\widehat{R} = \Omega(nL^7) \gg n \log^3 n$.

(iii) $\widehat{W} \leq n^\gamma$, $\widetilde{D} \leq L^6$. The size-diversity law gives a time $t^\star$ with $\Phi(t^\star) \geq (1 - \varepsilon - o(1))\widetilde{D}L\,\mu(\beta L/\widetilde{D})$; Lemma 4.3 there gives $\widehat{R} = \Omega(\Phi(t^\star))$, the edit floor gives $E = \Omega(n\widetilde{D})$, and the calculus floor closes it:

$$E\widehat{R} \;\geq\; \Omega\big(nL \cdot \widetilde{D}^2 \mu(\beta L/\widetilde{D})\big) \;\geq\; \Omega\big(\tfrac{\beta^2}{4}\, nL^3\big) = \Omega(nL^3).$$

The cases are exhaustive. Obliviousness makes the size trajectory the same on every input, so the skeleton each case invokes occurs along the execution, and Lemma 4.3 exhibits a hard realization there; the product is the algorithm's worst case over inputs, in expectation over its coins. □

The bound $\widehat{R} \geq \widehat{W}$ alone yields only $n \log^2 n$, one logarithm short; the extra logarithm needs $\widehat{R} = \Omega(\Phi)$ at a high-Φ time, which the size-diversity law supplies. Predecessor pays for size diversity, not merely frontier width: membership is priced by $\widehat{W}$, order by Φ.

4.5 Extensions to Rank and Range Queries

Predecessor is one order query among many. Whether a decomposable query pays the extra logarithm is read off the combiner, from how much of a single block's contribution can reach the global answer, and for natural exact queries this takes one of two sizes. It is $O(1)$ when an $O(1)$-read witness settles a block's contribution, as for membership, point-emptiness, and the global minimum or maximum. It is $\Theta(\log m)$ when the contribution turns on q's position among a block's m sorted keys, as for predecessor, successor, rank, range-count, range-sum, range-minimum, range-maximum, and bounded-interval emptiness. The bridge to a read floor in the second case is a per-level direct sum, supplied for predecessor and successor by Lemma 4.3 and for the rest by the two lemmas below.

Call Q *additive* if $Q(S, q) = \sum_k \varphi(L_k, q)$ in an abelian group, each contribution $\varphi(B, q)$ a function of q's position in B; rank, range-count, and unit-weight range-sum qualify. Let $g_Q(m)$ be the worst-case

reads to compute $\varphi(B, q)$ on one static block of m samples under a hard distribution $\mathcal{D}_m$, and call the family $\{\mathcal{D}_m\}$ *embeddable* when every active size vector is realized by independent private blocks and one shared query whose marginal at level i is $\mathcal{D}_{m_i}$.

Lemma 4.9 (Additive Direct Sum)**.** *For an additive Q with an embeddable hard family of per-level complexity g_Q under an oblivious schedule, the hard realization at time t forces $\widehat{R}_Q \geq \Omega\big(\sum_k g_Q(m_k)\big)$, deterministically and, with g_Q the randomized static complexity, against bounded-error evaluators.*

Proof. Draw the embedded product instance and fix a level i. Run the evaluator from B_i's draw alone, charging its reads inside L_i and answering each read outside L_i for free from a block drawn on independent coins from its law given q, which strong materialization and no-cascade make independent of B_i. The evaluator returns $Q(S, q)$, and subtracting the $\sum_{j \neq i} \varphi(L_j, q)$ just sampled leaves $\varphi(L_i, q)$ by the group inverse. This is a single-block evaluator for $\mathcal{D}_{m_i}$ whose outside answers are advice independent of B_i, so it cannot read L_i below the static floor $g_Q(m_i)$. The blocks are disjoint, so the reads sum. □

For abstract keys $g_Q(m) = \Theta(\log(m+1))$: computing q's position in a block is the comparison search of Lemma 4.2, and a balanced tree matches it. The lemma then gives $\widehat{R}_Q \geq c \sum_k \log_2(m_k + 1)$ for rank, range-count, and unit-weight range-sum, the floor predecessor pays; Section 6 reuses it with the integer and two-dimensional complexities.

Lemma 4.10 (Idempotent Direct Sum)**.** *Fix a public size vector of an oblivious schedule and a bounded-error evaluator. For bounded-interval emptiness, range-minimum, and range-maximum, each with a fixed empty-interval default, some strongly materialized state with these sizes admits an interval I, empty in every level, on which the evaluator has $\widehat{R}_Q \geq c \sum_i \log_2(m_i + 1)$, for an absolute $c > 0$.*

Proof. Give each level i a low slab below I, a high slab above I, and a hot subinterval inside I, the zones disjoint and ordered publicly by level index. Draw J_i uniform in $\{0, \ldots, m_i\}$, placing the first J_i keys of level i in its low slab and the rest in its high slab, so I holds no key and Q returns its default; the hot alternative, when $J_i < m_i$, moves k_{i,J_i+1} into the hot subinterval, making I nonempty and changing the answer while every other encoding stays fixed. Expose every level but i in its empty state for free; deciding the global answer is then deciding whether level i leaves I empty, the task of Lemma 4.2 with $m = m_i$ and the endpoints of I as the two handles, at $\Omega(\log(m_i + 1))$ reads. By no-cascade, reads elsewhere do not help, the per-level costs sum, and some all-empty realization meets the bound. □

Theorem 4.11 (Order/Range Dichotomy)**.** *A decomposable query has $E\widehat{R}_Q = \Theta(n \log^2 n)$ when an $O(1)$-read per-level certificate settles it and some unread live level can change its answer, as with membership, point-emptiness, and the global extrema. When instead a level's contribution turns on q's position among its keys, as for predecessor and successor, the additive rank, range-count, and range-sum, and the idempotent range-minimum, range-maximum, and bounded-interval emptiness, the cost is $E\widehat{R}_Q = \Theta(n \log^3 n)$, with no hypothesis on frontier or budget.*

Proof. On the cheap side an $O(1)$-read witness per level answers Q in $\widehat{R} = O(W) = O(\log n)$, hence $E\widehat{R} = O(n \log^2 n)$, matched below by the membership floor of Section 3, whose argument needs only that an unread live level can flip the answer. On the order-localizing side the direct sums give $\widehat{R}_Q \geq c \sum_k \log_2(m_k + 1) \geq c \max\{\widehat{W}, \Phi(t)\}$ on the hard skeleton, predecessor's own floor: Lemma 4.3 for predecessor and, reversing the order, successor; Lemma 4.9 for rank, range-count, and unit-weight

range-sum; and Lemma 4.10 for the idempotent range queries, where an empty query interval keeps every level pivotal. The three-case assembly of Theorem 4.8 then gives $\Omega(n \log^3 n)$, attained by binary Bentley–Saxe carrying the matching static structure, a balanced tree with subtree counts, prefix sums, or subtree extrema, at $O(\log m_k)$ reads per level. □

The line is hashing against within-level order localization: membership and point-emptiness fall to a hash and the global extrema to one stored value per level, all staying at the membership scale, while predecessor, successor, rank, and the range queries turn on q's position in each block and pay the logarithm. Total count is cheaper still, fixed by the public size sequence at $\Theta(n)$ and lying outside the trichotomy. What the logarithm buys is exactness of the order. The idempotent range queries carry no group inverse, yet an empty interval makes every level pivotal, a single hidden key overturning the empty default (Lemma 4.10). The logarithm is specific to abstract keys, and integer keys shrink it (Section 7).

Remark 4.12. Approximation refunds the logarithm. A constant additive error $\pm\varepsilon n$ in rank or range-count, or a constant-precision quantile, separates only $O(1)$ buckets and returns the query to $\Theta(n \log^2 n)$; a multiplicative error keeps the $\Theta(\log n)$ scales and does not.

5 Select

Select proves the third line of Theorem 1.1. Rank is a key-to-count query: the query names a key, and each component contributes a count. Select is the inverse, count-to-key query: the query names only a rank K. In a one-way scan the key that will become the answer is not known while the early components are being read, and those components cannot be revisited once later components reveal the threshold. The proof first turns this target-free difficulty into a cut lower bound, and then shows that any structure with small read cost must repeatedly rebuild a large component.

5.1 The Cut Lower Bound

Comparisons among keys already read are free, so the lower bound is on token reads and the adversary attacks the unread keys. Two lemmas carry it: a bound on what a single read can learn, and a cut floor built from it.

Lemma 5.1 (Layout Channel)**.** *Fix the query q, its coins ρ, and the data-independent initial configuration σ_0. Condition on the public layout G, the size sequence together with the boundaries and addresses it determines. Let X be the stored keys and values. For one run of the forward head let T be its number of charged reads and Π its transcript, the words it reads together with its working memory. Then*

$$I(X; \Pi \mid G, q, \rho, \sigma_0) \;\le\; b\,\mathbb{E}[T \mid G, q, \rho, \sigma_0] \;=\; O\big(\mathbb{E}[T] \log n\big), \qquad b = \Theta(\log n).$$

Proof. Assume $\mathbb{E}[T] < \infty$, the bound being vacuous otherwise, and condition throughout on G, q, ρ, σ_0. Record the k-th read as $V_k = (A_k, Y_k)$ when $T \ge k$ and $V_k = \perp$ otherwise, where A_k is the address read and Y_k the returned word together with the rank of its handle among those already read. A query reads at most the $O(n)$ live tokens, so that rank fits in $O(\log n)$ bits and Y_k carries $b = \Theta(\log n)$ bits. The head is deterministic given the conditioning and the reads so far, so $\{T \ge k\}$ is a function of $V_1, \dots, V_{k-1}$, and on that event so is A_k, by Definition 2.1; hence $I(X; A_k \mid V_{<k}) = 0$ and

$$I(X; V_k \mid V_{<k}) \;\le\; H(Y_k \mid V_{<k}, A_k) \;\le\; b.$$

Since $V_k = \perp$ when $T < k$, chain rule over $k \le K$ gives $I(X; V_1, \dots, V_K) \le b\,\mathbb{E}[\min\{T, K\}]$, and $K \to \infty$ gives $I(X; V^\infty) \le b\,\mathbb{E}[T]$. The transcript Π is a function of the conditioning and V^∞ by the no-external-state clause, so data processing closes it. □

For a randomized evaluator the coins ρ are independent of the data, so the bound holds with Π replaced by (Π, ρ) and averaged over ρ.

Lemma 5.2 (Cut Floor)**.** *At a boundary between two active levels holding P samples before it and U after, with $r = \min\{P, U\}$, some input family with the fixed rank $K = r$ forces every bounded-error forward evaluator to read $\Omega(r)$ words before the head crosses the boundary. Hence $\widehat{R}(t) \ge c \min\{P, U\}$ for an absolute $c > 0$.*

Proof. Place r disjoint rank slabs $S_1 < \cdots < S_r$ in the prefix, separated by fixed guards, each slab a public range of $Q = \mathrm{poly}(n)$ candidate positions. Draw a_i uniformly and independently from S_i and let Z_i name which candidate it is, so the a_i carry $\Theta(\log n)$ independent bits while their order is fixed by the slabs; put the remaining $P - r$ prefix keys in a guard above S_r. The suffix encodes an index j: place $r - j$ of its keys in a guard below S_1 and the rest in a guard above every slab. Then exactly $r - j$ suffix keys and the $j - 1$ slab keys $a_1, \dots, a_{j-1}$ lie below a_j, so a_j is the r-th smallest of the union, while the prefix and the rank $K = r$ are the same for every j.

Draw J uniform on $\{1, \dots, r\}$, independent of $Z = (Z_1, \dots, Z_r)$, and let Π be the transcript and coins at the crossing. The prefix is read before the crossing and does not depend on J, so (Z, Π) is independent of J, while the suffix determines J and carries nothing about Z. A correct answer is a_J, which the head must decode from Π and J since it cannot return to the prefix; so by Fano $I(Z_j; \Pi \mid J = j) = \Omega(\log Q)$, equal to $I(Z_j; \Pi)$ by that independence. As the Z_i are independent, $I(Z; \Pi) \ge \sum_i I(Z_i; \Pi) = \Omega(r \log n)$. Since Z is a function of X and Π records the T prefix reads, Lemma 5.1 gives $\Omega(r \log n) \le I(X; \Pi) \le O(\mathbb{E}[T] \log n)$, whence $\mathbb{E}[T] = \Omega(r)$ and $\widehat{R}(t) \ge \mathbb{E}[T] = \Omega(r)$. □

The guards keep the suffix clear of Z: its keys sit in fixed slabs, never at a data-dependent value such as $a_1 - 1$, so reading them reveals only j. The floor rests on a single boundary rather than a sum over levels, but that is enough: a low read cost forces almost all the live mass into one level, and building such a level again and again is the write cost select cannot escape.

5.2 The Quadratic Product Bound

Theorem 5.3 (Quadratic Select Bound)**.** *Strongly materialized select has $\min E\widehat{R} = \Theta(n^2)$, with no hypothesis on the frontier or the budget, and the lower bound holds against bounded-error query randomization.*

Proof. Write $R = \widehat{R}$ and $s = R/c$ with c the constant of Lemma 5.2. The live mass at time t is t, and at every boundary $\min\{P, U\} \le R/c = s$, so each boundary mass is $\le s$ or $\ge t - s$ and the level spanning $(s, t - s)$ has size $\ge t - 2s$. Suppose $R < cn/8$, so $s < n/8$ and this level exceeds $n/4$ throughout $t \in [n/2, n]$. Take times $t_0 < \cdots < t_K$ spaced by $\Delta = \lceil 2s \rceil + 1$ across $[n/2, n]$, so $K = \Omega(n/(R+1))$, and let B_i be such a level at t_i. Being materialized, B_i is static after birth, so $|B_i| \le t_i$, while $|B_{i+1}| \ge t_{i+1} - 2s > t_i \ge |B_i|$; the B_i are therefore distinct components, each costing $\Omega(n)$ tokens. Hence $E = \Omega(n^2/(R+1))$, and as $R \ge 1$, $E\widehat{R} = \Omega(n^2)$. If instead $R \ge cn/8$, then $E \ge n$ gives $E\widehat{R} \ge nR = \Omega(n^2)$.

For the upper bound, keep one large level together with a suffix of at most s singletons, merging the whole stack into a fresh large level every s inserts. Rebuilding the large level at sizes $s, 2s, \ldots, n$ costs $E = \Theta(n^2/s)$. The large level is a sorted array at the bottom of the stack, so the head jumps by a computed offset to its rank window $[K - s, K]$, reads those $O(s)$ keys and the $\leq s$ singletons above it, and selects the global K-th among these candidates; the suffix shifts the answer's rank in the large level by at most s, so the window holds it, and $\widehat{R} = O(s)$. The product is $\Theta(n^2)$ at every s. □

The three query classes now stand in a line, governed by what the query hands the forward head:

$$\underbrace{\Theta(n \log^2 n)}_{\text{hashable witness}} \;<\; \underbrace{\Theta(n \log^3 n)}_{\text{known key value}} \;\ll\; \underbrace{\Theta(n^2)}_{\text{rank, no value}} \quad .$$

The first gap is one logarithm, order against hashing; the second is a polynomial. Rank and select are inverse operations, value to count and count to value, and select's answer is itself only $O(\log n)$ bits, yet the two lie polynomially apart: the gap is made by the access discipline, not by the information in the answer. Select gives the head no value to aim at, and that is what carries it past predecessor. The cost is a property of the forward head, not of the key model, since the lower bound forces its reads by completing cells the head has not visited and never inspects a key's representation; it therefore persists for integer keys, where the word-RAM that narrows the order gap (Section 7) still leaves select at $\Theta(n^2)$.

Remark 5.4. Constant-precision approximate select returns to $\Theta(n \log^2 n)$: $O(1)$ samples at evenly spaced ranks per level form a forward quantile sketch that needs no refinement of the exact threshold. Exactness is again what the cost buys.

Remark 5.5. A cross-level rank catalog would answer select in $O(1)$ reads, but maintaining one under LIFO appears to cost $\Omega(n^2)$ edits, which would carry the $\Theta(n^2)$ past strong materialization, as Appendix A carries the predecessor bound. We do not pursue this here.

6 Static-to-Dynamic Lifting

Every query that pays the order logarithm lands at the same $n \log^3 n$, and a single reduction explains why. Each pays, per level, the static cost of locating the query there; the model forbids sharing those costs across levels, so the read cost is their sum. For an additive query this turns the dynamization optimum into the edit budget times a sum of static per-level complexities, a dynamic lower bound read off a static one. The logarithm of order is then no artifact of maintenance but the faithful lift of a static search cost.

Recall the additive queries of Section 4: $Q(S, q) = \sum_k \varphi(L_k, q)$ in an abelian group, with per-level static complexity $g_Q(m)$, the reads to compute one block's contribution $\varphi(B, q)$, and embeddable hard families that plant an independent instance in every level (Lemma 4.9).

Theorem 6.1 (Static-to-Dynamic Lifting)**.** *Let Q be additive with an embeddable hard family and per-level static complexity g_Q: a forward-readable static structure attains $O(g_Q(m))$ reads on a block of m samples, and the hard family forces $\Omega(g_Q(m))$. Suppose g_Q is nondecreasing with $g_Q(1) \geq 1$ and of linear rate $g_Q(2^k) \geq \zeta_Q k$ for some $\zeta_Q \in (0, 1]$, and $\Sigma_Q := \sum_{k=0}^{L} g_Q(2^k) = \Theta(\zeta_Q L^2)$. Then every strongly materialized algorithm with a bounded-error evaluator has $E\widehat{R}_Q = \Omega(nL\,\Sigma_Q)$, which binary Bentley–Saxe attains, so* $\min E\widehat{R}_Q = \Theta(nL\,\Sigma_Q)$.

Proof. For the upper bound, the schedule of Proposition 2.3 carries that static structure at each level, a balanced search tree with subtree aggregates for the additive queries. Then $E = O(nL)$, and the active sizes being distinct powers of two, $\widehat{R} = \sum_{\text{active}} g_Q(m_k) \le \Sigma_Q$.

For the lower bound, Lemma 4.9 gives $\widehat{R}_Q(t) = \Omega(\sum_k g_Q(m_k))$; the linear rate and the floor $g_Q(1) \ge 1$ make $g_Q(m) = \Omega(\zeta_Q \log_2(m+1))$, so $\widehat{R}_Q(t) = \Omega(\zeta_Q \sum_k \log_2(m_k+1))$. Since $\sum_k \log_2(m_k+1)$ is at least $\max\{W(t), \Phi(t)\}$ at all times and at least $\log_2(n+1)$ at the final skeleton, this is the read floor of Theorem 4.8 scaled by ζ_Q. Its three-case assembly then applies: the wide-frontier and deep cases exceed $nL\,\Sigma_Q$, while the size-diversity case multiplies $E = \Omega(n\widetilde{D})$ by $\widehat{R}_Q = \Omega(\zeta_Q \Phi(t^\star))$ and closes with the calculus floor, giving $E\widehat{R}_Q = \Omega(\zeta_Q n L^3)$. By $\Sigma_Q = \Theta(\zeta_Q L^2)$ this is $\Omega(nL\,\Sigma_Q)$. □

The middle row of the dichotomy is now one line: abstract-key rank, range-count, and range-sum have $g_Q(m) = \Theta(\log(m+1))$, hence $\zeta_Q = \Theta(1)$, $\Sigma_Q = \Theta(L^2)$, and $E\widehat{R}_Q = \Theta(nL^3)$. Beyond recovering the additive row of Theorem 4.11, the lift names the structure of the cost. The read bound is a sum of per-level static search complexities, one term per live level, because maintenance does not inflate a single level and cascading cannot amortize across them. That this sum cannot stay below L^2 is the size-diversity of Section 4: that section forces the sum up to L^2, and this one explains why the cost is that sum to begin with. The two are the lower-bound face and the structural face of the same $n \log^3 n$.

Remark 6.2. Because g_Q is a static complexity, the lift inherits its robustness. Integer keys shrink the per-level cost and carry rank, range-count, and range-sum to $\Theta(n \log^2 n\,\lambda)$, while in two dimensions the separation survives the word-RAM, with range-counting lifting to $\Theta(n \log^3 n/\lambda)$ and no van Emde Boas collapse. Both are taken up in Section 7; the integer bounds there are deterministic and Las Vegas cell-probe bounds, and a bounded-error version would need a randomized static one. Integer predecessor, not an additive query, stays open between 1 and λ.

7 Access-Model Variants

The classification of Theorem 1.1 is stated for abstract keys, strong materialization, and one-way scans. This section records what happens when one restriction at a time is relaxed. These variants are not needed for the main lower bounds; their role is to identify which assumption supports which separation. Integer keys reduce the cost of searching inside one component, random access removes the no-backseek obstruction for select, and non-materialized cascade data can remove the abstract-key predecessor gap only when it can be placed in the right scan direction.

7.1 Integer Keys

The order logarithm is a property of abstract keys. An integer key in a poly(n) universe is searchable by the word-RAM: a y-fast trie [Wil83] finds a run's predecessor in $O(\lambda)$ reads and a fusion tree [FW93] in $O(\log_w m)$, both forward-readable under the round-by-round layout of Proposition 2.3. Predecessor then costs $O(n \log^2 n\,\lambda)$, its gap over membership shrinking from $\log n$ to λ, the way radix sort undercuts the comparison-sorting bound without refuting it.

The collapse is not uniform, and the reason is what makes abstract keys hard. An additive query reads a contribution from every run, so each must still be searched, now at integer cost: a rank among m integers is a predecessor search of static complexity $g_Q(m) = \Theta(\max\{1, \min\{\log_w m, \lambda\}\})$, the y-fast or fusion layout above, matched from below by Pătraşcu and Thorup [PT06, PT07],

deterministic and Las Vegas alike. Its binary sum is $\Sigma_Q = \Theta(L\lambda)$, so Theorem 6.1 places rank, range-count, and range-sum at $\Theta(n \log^2 n\, \lambda)$, exactly $\Theta(\lambda)$ above membership.

Predecessor behaves differently, because it is settled by one run and the rest need only be ruled out. Ruling run i out asks whether its keys meet the open interval between the running best and the query, a range-emptiness test the word-RAM answers in $O(1)$ [ABR01] but abstract keys answer only by localizing the query inside the run, the $\Omega(\log m)$ step behind the separation. The per-run floor the additive queries enjoy has no integer counterpart here, so the exact integer cost of predecessor stays open between $\Theta(n \log^2 n)$ and $\Theta(n \log^2 n\, \lambda)$: a lower bound would need a per-run test that stays $\Omega(\lambda)$-hard under a shared query, and the natural one, whether a run holds the global predecessor, is the range-emptiness integers trivialize. The key models part exactly here.

The collapse is one-dimensional. Two-dimensional range-counting has no van Emde Boas speedup, its cell-probe complexity $\Omega(\log m/\lambda)$ at linear space [Păt07] matched by a range tree [CW13], so lifting $g_Q(m) = \Theta(\max\{1, \log m/\lambda\})$ gives $\Theta(n \log^3 n/\lambda)$, above membership by $\Theta(\log n/\log\log n)$. The separation is no artifact of unstructured keys; it survives the word-RAM as soon as the queries are two-dimensional, the regime of multi-attribute range search.

7.2 Random Access

Select's polynomial cost is the signature of the no-backseek head, and it vanishes the moment the head may return. The obstacle of Section 5 was an answer threshold fixed by all runs jointly, unknown while the early runs are read and out of reach once they are passed. Random access dissolves it: selecting the K-th smallest among W sorted runs is classical, $O(W + \sum_i \log(k_i + 1))$ comparisons with k_i run i's share of the K smallest [FJ84, KKZZ19], which is $O(\log^2 n)$ at a diverse frontier. Random access brings select to $O(n \log^3 n)$, the value-query scale, with the exact optimum open.

It does no more. Predecessor stays at $n \log^3 n$ under random access, since every run must still be searched and no run bridges another; what random access removes is not the per-run search but the forward head's inability to revisit an internal threshold. The polynomial gap was never in the answer, only $O(\log n)$ bits, but in that head. Integer keys touch only the order gap and random access only select's, each isolating the restriction its cost depends on.

7.3 Deque Edits

Two restrictions guard the cascade together: a merge stack writes only at the tail, behind the forward head, and its runs are strongly materialized, so no run holds order information about another. Relax both, writing at the head's entrance and letting a young run carry a bridge into an older one, and the separation falls. Model the relaxation by *deque* edits, popping and pushing at both ends of the token string while keeping the same forward, no-backseek head, so a component may be placed at position 0, ahead of all the head will read; dropping materialization lets a young component store forward handles into the older ones, charged to itself and leaving them unedited.

Theorem 7.1 (Deque Collapse)**.** *Under deque edits with non-materialized runs, abstract-key predecessor admits $E = O(n \log n)$ and $\widehat{R} = O(\log n)$, so $E\widehat{R} = O(n \log^2 n)$ and the order separation does not survive.*

Proof. A useful bridge from U to V needs U read before V and V born before U (Lemma 4.1), and tail-only editing forces birth order to follow read order and kills it. Deque editing breaks the

coupling: lay the active components youngest first, $C_1, \dots, C_W$ read from the entrance with sizes increasing, so C_i is read before C_{i+1} while C_{i+1} is the older of the two, the very order a cascade needs.

Run the Bentley–Saxe counter mirrored at the entrance: each insert pushes a singleton at the front and, while the two front components share a size, pops both, merges, and pushes one new front component. A component of m_i keys stores its block as a forward-readable balanced tree together with a sparse cascade into the next-older C_{i+1}, cutting C_{i+1}'s order into $O(m_i)$ intervals of span $O(m_{i+1}/m_i)$ and keeping, per gap, its own predecessor and a forward handle to the precomputed subtree root for that interval, a node already inside C_{i+1}'s own tree once C_{i+1} is serialized. Each older component is serialized once with its roots in place, so a younger one adds only its $O(m_i)$ handles and never edits it; hence $|C_i| = O(m_i)$, and a size-2^k component is built $O(n/2^k)$ times at $O(2^k)$ tokens, giving $E = O(n \log n)$.

A query searches C_1 in $O(\log m_1)$ reads, then at each step follows the stored handle into a span-$O(1 + m_{i+1}/m_i)$ interval of C_{i+1}, searches its precomputed subtree, updates the best candidate, and moves on without a backseek. The reads telescope,

$$\widehat{R} = O\Big(W + \log m_1 + \sum_{i \geq 2} \log \tfrac{m_i}{m_{i-1}}\Big) = O(W + \log m_W) = O(\log n). \qquad \square$$

The collapse is one of orientation, not of two-endedness. A queue, pushing at the tail and popping at the head, still builds its components at the scan's exit and does not collapse; what predecessor pays for under tail-only editing is exactly that new connectivity must be written behind the forward scan. The gap sits at the missing cascade, and the cascade the deque builds is non-materialized, a young run pointing into an older one, so the collapse leaves the materialized model and the tail-only one together. This is why random access alone, the runs still materialized, does not collapse predecessor. The matching $\Omega(n \log^2 n)$ would follow from a membership floor under deque edits, a minor open point since the universal floor of Section 3 uses the tail-only frontier-width bound; the collapse out of the $n \log^3 n$ class is unconditional. Afshani's dynamic fractional-cascading lower bound [Afs21] forces either fully dynamic updates or worst-case update time, while the construction here is insertion-only with amortized rebuilding, and escapes both.

7.4 LSM Read Amplification

The read gap reaches past edit histories to the runs in place at query time, though only under a hypothesis. A run can hold its median and successor in an $O(1)$ header and certify a query's predecessor between them in two reads, so overlap by itself does not force $\Omega(\log m)$. What forces it is a query distribution that, after all outside information is exposed, still leaves $\Omega(\log m_i)$ of conditional gap entropy in each overlapped run, exactly the hard family of Lemma 4.3.

Proposition 7.2 (Range Read Amplification)**.** *Let $R_1, \dots, R_W$ be active sorted runs, pairwise no-cascade and carrying no cross-run order catalog. Under a query distribution leaving $\Omega(\log m_i)$ conditional gap entropy in each run it crosses, an exact range or order query has read cost $\Omega(\sum_{i \text{ crossed}} \log m_i)$, while a point query is settled in $O(1)$ reads per run.*

Proof. The runs being mutually no-cascade, a read in one cannot shrink another's candidate set, so the per-run argument of Lemma 4.3 applies to each crossed run on its own: the hypothesis leaves $\Omega(\log m_i)$ gap entropy there, and localizing the endpoint costs $\Omega(\log m_i)$ reads. Carrying no

cross-run catalog, the runs do not share the cost, so it sums. A point query is hashable, settled per run by a filter in $O(1)$. □

The bound is on the runs in place, however built or read, so it holds in any leveled sorted-run store. In a random-access store the per-run filters let a point query skip the runs that miss it, leaving $O(1)$; a range or order query has no such escape and localizes in every run it spans, $\Theta(\log_T n)$ at size-ratio T, each localization irreducible by Proposition 7.2. This range-over-point read amplification closes only two ways: a word-RAM integer index, which serves one dimension but not string keys or multi-attribute ranges, where the two-dimensional separation persists; or a cross-run order catalog, which appears to need $\Omega(n^2)$ edits to maintain under append-only merging (Section 5). For write-optimized, string-keyed, multi-attribute engines in the insertion-only regime the model captures, the range read amplification is intrinsic, not an implementation artifact [OCGO96, DAI17, DI18].

7.5 Intermediate Queries

Order at $n \log^3 n$ and select at n^2 are the two ends of one scale, set by how much the query reveals about its answer's value. A w-windowed select, the K-th smallest under the promise that the answer lies in a bracket of at most w live keys, interpolates between them: at $w = O(1)$ the bracket pins the value and the query is an order query, at $w = N$ the promise is empty and it is select. The cut floor restricted to the bracket gives $\widehat{R} \geq c \min\{w, P, U\}$, only the $\leq w$ keys inside it carrying the ambiguity. The matching write tradeoff across the interior is open, and we leave it.

A Cascades Beyond Materialization

Strong materialization (Definition 2.1) makes each component a self-contained sorted file, the locality the lower bounds above turn on: a read in one component carries no order information about another. This section removes exactly that. A young component may now keep a directory, copied separators, or a search summary for older keys. The rest of the serialized model stays: LIFO edits, exact layout, no sample-dependent state outside the token string, oracle abstract keys, and a forward no-backseek head.

Here the separation turns genuinely dynamic. The earlier proof priced the write budget against the frontier width and the live search potential, but a non-materialized suffix can fuse many old searches into one directory lookup, so the frontier is no longer the right primitive. What survives is sharper: while a token sits in the longest common prefix of two consecutive states, the transcript by which a query reaches it is fixed. This yields a canonical alphabetic search tree laid out in stack order, and the reduction it supports proves the $n \log^3 n$ bound at near-optimal read cost and isolates the rest as a single suffix-recourse conjecture.

Write s_t for the token string after update t. A LIFO edit has the form

$$s_t = P_t A_t, \qquad s_{t+1} = P_t B_t,$$

with P_t the longest common prefix and charged cost $|A_t| + |B_t|$; set $R = \lceil \widehat{R} \rceil$.

Lemma A.1 (Prefix-Transcript Invariance)**.** *Fix a deterministic evaluator and a query q. On the states $s = PA$ and $s' = PB$, the two executions agree until the head first leaves P.*

Proof. Both start in the same control state, query, and head position. While the head stays in P, each step reads the same token at the same position, exact layout included, runs the same

computation, sees the same comparison and equality outcomes, moves to the same control state, and computes the same forward skip, so the executions agree inductively. A token of the new suffix B is not reached before the head leaves P, so it cannot alter any earlier branch. □

This is stronger than the causal-bridge obstruction of Lemma 4.1: not only can a later directory not point backward in time, a later suffix cannot change the transcript by which a query reaches any retained token.

Lemma A.2 (Alphabetic-Tree Normal Form)**.** *Fix a state of N distinct abstract keys with distinct values and a deterministic exact-predecessor evaluator of read cost at most R. Placing a single nonmember handle in each of the $N+1$ open gaps in turn, the evaluator induces an alphabetic comparison tree of $N+1$ leaves and height at most R, each internal separator witnessed by a token occurrence that reveals the corresponding stored key, with the witness positions strictly increasing along every root-to-leaf path.*

Proof. Let the $N+1$ alternatives be one nonmember handle ξ with a fixed equality label, distinct from every stored one, that the oracle drops into each gap in turn; across the alternatives ξ's equality tests stay false and its hash address stays fixed, so only a comparison against a revealed stored key can branch by gap. Unfold the evaluator into its decision tree and normalize it to compare ξ with each handle the moment that handle is read, which adds no charged reads and puts the order-sensitive comparisons in read order. Order-independent reads then give the same outcome in every gap and are quotiented out. Each gap's answer is the value of its predecessor, distinct across the $N+1$ gaps, so their transcripts carry distinct leaves. For the two gaps flanking a stored key x the predecessors differ; with ξ placed close enough to x, every comparison against a $y \neq x$ agrees on both sides and no hash or equality label reveals an order position, so the first order-sensitive divergence is a comparison against a token revealing x, taken as the canonical witness of the boundary at x. These comparisons separate contiguous intervals of query order, so after suppressing unary nodes they form an alphabetic tree of $N+1$ leaves; each charged read reveals at most one stored handle, so a root-to-leaf path holds at most R of them; and the handles being compared in read order under a head that never moves back, the witness positions strictly increase along each path. □

The oracle-key hypothesis carries this. A directory entry can say where to skip, and a young component can copy an old handle, but a token that reveals no stored key cannot act as an order-bearing separator: an old handle denotes the same abstract key, its hash leaks no position, and by Lemma A.1 the transcript reaching its token is fixed, so a later directory cannot reinterpret it as a fresh separator before it is read. This shuts the loophole by which metadata might manufacture separators for free.

A retained canonical witness keeps its address in the tree. If its token lies in P_t, so does every earlier witness on its query path, and Lemma A.1 fixes the incoming transcript and the ancestor chain; a suffix witness occurs later in the string, so it may refine a retained leaf but cannot become an ancestor of a retained witness.

Theorem A.3 (Labeling Reduction)**.** *Every deterministic non-materialized exact-predecessor structure with write cost E and prefix-worst read cost $\widehat{R}$ induces an online order-preserving labeling of the inserted keys, with label range at most 2^{R+1} and total relabeling at most E, in which changing an old label means deleting that key's canonical witness from the physical suffix and appending replacements.*

Proof. At each time take the canonical tree of Lemma A.2 and give a separator at depth d with binary address a the dyadic label $(2a+1)2^{R-d}$, the midpoint of its depth-d interval refined to depth R. These are strictly ordered by the inorder of the separators and lie in $\{1,\dots,2^{R+1}-1\}$, a range for the reduction and not a constraint on the keys. Under one edit $P_tA_t \to P_tB_t$, an old key whose witness stays in P_t keeps its depth, address, and label, so every old key that is relabeled has its witness in the popped A_t; distinct keys have distinct witnesses, so at most $|A_t|$ keys are relabeled, and the new key draws its first label from B_t. The update costs at most $|A_t|+|B_t|$, and the sum is at most E. The retained witnesses are a prefix of the physical witness order and the replacements are written in the appended suffix, the recourse the statement names. □

Proposition A.4 (Near-Static Regime)**.** *A deterministic non-materialized exact-predecessor structure with $\widehat{R} \le \log_2 n + O(1)$ has $E\widehat{R} = \Omega(n \log^3 n)$.*

Proof. The labeling lower bound is adaptive over a totally ordered key set, so against a deterministic structure the adversary fixes a sequence of n keys with distinct values whose order alone is read through the oracle, realized by distinct ranks in the poly(n) universe. With $\widehat{R} \le L + O(1)$ the induced label range $2^{R+1} = O(n)$ stays well inside that universe, so the adversary can refine a label interval past exhaustion, and the linear-range labeling lower bound of Bulánek, Koucký, and Saks [BKS12] forces $\Omega(nL^2)$ relabelings, giving $E = \Omega(nL^2)$. Exact predecessor search needs $\widehat{R} = \Omega(L)$, so $E\widehat{R} = \Omega(nL^3)$. □

The reduction reaches no further. The $n \log^3 n$ bound rests on the linear-range labeling lower bound, hence on a label range $O(n)$ and $\widehat{R} \le \log_2 n + O(1)$. For larger read cost the range outgrows the linear regime and, once it passes the poly(n) key universe, leaves ordinary labeling with no relabelings to force. What the reduction keeps and ordinary labeling discards is sharper: an old label changes only through a physical suffix rebuild of the alphabetic tree. Extending the lower bound to all read costs would amount to proving that suffix-recourse alphabetic trees must delete $\Omega(nL^3/R)$ old separators over some insertion sequence when their height is bounded by R. We record this as the residual non-materialized kernel, not as a claim needed by the main theorem.

Remark A.5. A sparse directory genuinely escapes the materialized proof. For an old sorted set A and a younger batch B, the young suffix can store, for each nonempty gap g of A, a static predecessor structure over $B_g = \{x \in B : x \text{ falls in } g\}$, of total size $O(|B|)$; having searched A the evaluator knows g and searches only B_g, fusing many localization scales without copying A. What it cannot escape is temporal renewal: once several batches enter one gap, a late directory names the newest bucket but the forward head cannot turn back to an older body, so making the newest resolver forward-readable forces a suffix rebuild. Whether this renewal can be paid once for all nested scales is the residual question. Three views sharpen it. Witnesses sit parent before descendant, so a popped witness pops its descendants and the deletions are disjoint complete subtrees, not arbitrary relabelings. A direct-sum potential suffices while a prefix skeleton survives, the local recourse potentials adding over the external cells; what is missing is the renewal inequality once the skeleton itself is popped, when one rebuilt subtree can refresh every nested resolver beneath it at a stroke. And in rectangle terms each witness is a block laminar in both time and query order, a laminarity broken by path copying, backward pointers, or entrance writes, which is why random-access cascades, the cache-oblivious lookahead array [BFCF+07], and dynamic fractional cascading [Afs21] sit beside this problem rather than settle it.

Materialization is the last assumption the model sheds, and the one this paper leaves as a frontier rather than a crutch: the predecessor bound of Section 4 needs none of this section, the non-

materialized separation is unconditional at near-optimal read cost, and the residue is a suffix-recourse alphabetic-tree problem. The section is deterministic. A randomized extension cannot follow from the labeling reduction alone, since randomized online labeling beats the deterministic $\log^2 n$ barrier in the linear range [BCFC+22]; any randomized proof would have to use the suffix-renewal constraint itself, not label density.

Acknowledgments

The authors used OpenAI's ChatGPT 5.5 Pro to help organize and polish the exposition of this paper, based on drafts written by the authors. All mathematical statements, proofs, and references were checked, revised, and finalized by the authors. The authors assume responsibility for all content of the submission.

References

[ABR01] Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. Optimal static range reporting in one dimension. In *STOC*, pages 476–482, 2001.

[Afs21] Peyman Afshani. A lower bound for dynamic fractional cascading. In *SODA*, pages 2229–2248, 2021. arXiv:2011.00503.

[AKM+16] Manos Athanassoulis, Michael S. Kester, Lukas M. Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki, and Mark Callaghan. Designing access methods: The RUM conjecture. In *EDBT*, pages 461–466, 2016.

[BCFC+22] Michael A. Bender, Alex Conway, Martín Farach-Colton, Hanna Komlós, William Kuszmaul, and Nicole Wein. Online list labeling: Breaking the $\log^2 n$ barrier. In *FOCS*, pages 980–990, 2022. arXiv:2203.02763.

[BCFC+24] Michael A. Bender, Alex Conway, Martín Farach-Colton, Hanna Komlós, Michal Koucký, William Kuszmaul, and Michael Saks. Nearly optimal list labeling. In *FOCS*, pages 2253–2274, 2024. arXiv:2405.00807.

[BDHK23] Sander Borst, Daniel Dadush, Sophie Huiberts, and Danish Kashaev. A nearly optimal randomized algorithm for explorable heap selection. In *IPCO*, volume 13904 of *Lecture Notes in Computer Science*, pages 29–43, 2023. arXiv:2210.05982.

[BF03] Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *SODA*, pages 546–554, 2003.

[BFCF+07] Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming B-trees. In *SPAA*, pages 81–92, 2007.

[BKS12] Jan Bulánek, Michal Koucký, and Michael Saks. Tight lower bounds for the online labeling problem. In *STOC*, pages 1185–1198, 2012. arXiv:1112.5636.

[BS80] Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems*, 26(2):1–26, 2008.

[CG86] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[CW13] Timothy M. Chan and Bryan T. Wilkinson. Adaptive and approximate orthogonal range counting. In *SODA*, pages 241–251, 2013.

[DAI17] Niv Dayan, Manos Athanassoulis, and Stratos Idreos. Monkey: Optimal navigable key-value store. In *SIGMOD*, pages 79–94, 2017.

[DI18] Niv Dayan and Stratos Idreos. Dostoevsky: Better space-time trade-offs for LSM-tree based key-value stores via adaptive removal of superfluous merging. In *SIGMOD*, 2018.

[DIN22] Rathish Das, John Iacono, and Yakov Nekrich. External-memory dictionaries with worst-case update cost. In *ISAAC*, volume 248 of *LIPIcs*, pages 21:1–21:13, 2022. arXiv:2211.06044.

[FJ84] Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984.

[FS89] Michael L. Fredman and Michael E. Saks. The cell probe complexity of dynamic data structures. In *STOC*, pages 345–354, 1989.

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.

[IKR81] Alon Itai, Alan G. Konheim, and Michael Rodeh. A sparse table implementation of priority queues. In *ICALP*, volume 115 of *Lecture Notes in Computer Science*, pages 417–431, 1981.

[KKZZ19] Haim Kaplan, László Kozma, Or Zamir, and Uri Zwick. Selection from heaps, row-sorted matrices, and $X + Y$ using soft heaps. In *SOSA*, volume 69 of *OASIcs*, pages 5:1–5:21, 2019.

[Ko25] Young Kun Ko. Unifying the landscape of super-logarithmic dynamic cell-probe lower bounds, 2025. arXiv:2510.17717; ECCC TR25-156.

[Lar12] Kasper Green Larsen. The cell probe complexity of dynamic range counting. In *STOC*, pages 85–94, 2012. arXiv:1105.5933.

[LCK+20] Siqiang Luo, Subarna Chatterjee, Rafael Ketsetsidis, Niv Dayan, Wilson Qin, and Stratos Idreos. Rosetta: A robust space-time optimized range filter for key-value stores. In *SIGMOD*, pages 2071–2086, 2020.

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[LPG+17] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Hariharan Gopalakrishnan, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. WiscKey: Separating keys from values in SSD-conscious storage. *ACM Transactions on Storage*, 13(1):1–28, 2017.

[MDL20] Yoshinori Matsunobu, Siying Dong, and Herman Lee. MyRocks: LSM-tree database storage engine serving facebook's social graph. *Proceedings of the VLDB Endowment*, 13(12):3217–3230, 2020.

[Meh81] Kurt Mehlhorn. Lower bounds on the efficiency of transforming static data structures into dynamic structures. *Mathematical Systems Theory*, 15(1):1–16, 1981.

[MLI25] Dingheng Mo, Siqiang Luo, and Stratos Idreos. How to grow an LSM-tree? Towards bridging the gap between theory and practice. *Proceedings of the ACM on Management of Data*, 3(3):173:1–173:25, 2025. arXiv:2504.17178.

[MN90] Kurt Mehlhorn and Stefan Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990.

[MP80] J. Ian Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315–323, 1980.

[MRYY21] Claire Mathieu, Rajmohan Rajaraman, Neal E. Young, and Arman Yousefi. Competitive data-structure dynamization. In *SODA*, pages 2269–2287, 2021. arXiv:2011.02615.

[OCGO96] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.

[Ove83] Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983.

[Păt07] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. In *STOC*, pages 40–46, 2007.

[Păt11] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011.

[PT06] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *STOC*, pages 232–240, 2006. arXiv:cs/0603043.

[PT07] Mihai Pătraşcu and Mikkel Thorup. Randomization does not help searching predecessors. In *SODA*, pages 555–564, 2007.

[RKCA17] Pandian Raju, Rohan Kadekodi, Vijay Chidambaram, and Ittai Abraham. PebblesDB: Building key-value stores using fragmented log-structured merge trees. In *SOSP*, pages 497–514, 2017.

[Wil83] Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, 1983.

[ZCWJ21] Wenshao Zhong, Chen Chen, Xingbo Wu, and Song Jiang. REMIX: Efficient range query for LSM-trees. In *USENIX FAST*, pages 51–64, 2021. arXiv:2010.12734.

[ZLL+18] Huanchen Zhang, Hyeontaek Lim, Viktor Leis, David G. Andersen, Michael Kaminsky, Kimberly Keeton, and Andrew Pavlo. SuRF: Practical range query filtering with fast succinct tries. In *SIGMOD*, pages 323–336, 2018.