THE LIMITS OF BUFFERING: A TIGHT LOWER BOUND FOR DYNAMIC MEMBERSHIP IN THE EXTERNAL MEMORY MODEL[*]

ELAD VERBIN[†] AND QIN ZHANG[‡]

Abstract. We study the dynamic membership (or dynamic dictionary) problem, which is one of the most fundamental problems in data structures. We study the problem in the external memory model with cell size b bits and cache size m bits. We prove that if the amortized cost of updates is at most 0.9 (or any other constant < 1), then the query cost must be $\Omega(\log_{b \log n}(n/m))$, where n is the number of elements in the dictionary. In contrast, when the update time is allowed to be $1 + o(1)$, then a bit vector or hash table give query time $O(1)$. Thus, this is a *threshold phenomenon for data structures.*

This lower bound answers a folklore conjecture of the external memory community. Since almost any data structure task can solve membership, our lower bound implies a dichotomy between two alternatives: (*i*) make the amortized update time at least 1 (so the data structure does not buffer, and we lose one of the main potential advantages of the cache), or (*ii*) make the query time at least roughly logarithmic in n. Our result holds even when the updates and queries are chosen uniformly at random and there are no deletions; it holds for randomized data structures, holds when the universe size is $O(n)$, and does not make any restrictive assumptions such as *indivisibility.* All of the lower bounds we prove hold regardless of the space consumption of the data structure, while the upper bounds only need linear space.

The lower bound has some striking implications for external memory data structures. It shows that the query complexities of many problems such as 1D-range counting, predecessor, rank-select, and many others, are all the same in the regime where the amortized update time is less than 1, as long as the cell size is large enough ($b = \text{polylog}(n)$ suffices).

The proof of our lower bound is based on a new combinatorial lemma called the Lemma of Surprising Intersections (LOSI) which allows us to use a proof methodology where we first analyze the intersection structure of the positive queries by using encoding arguments, and then use statistical arguments to deduce properties of the intersection structure of *all* queries, even the negative ones. In most other data structure arguments that we know, it is difficult to argue anything about the negative queries. Therefore we believe that the LOSI and this proof methodology might find future uses for other problems.

1. Introduction.

Dynamic membership is one of the most fundamental problems in data structures. In this problem, we must design a data structure for maintaining a dynamic set $S \subset U$ ($|U| = u$) with $|S| = n \leq u$.

- *Insert:* Insert an item $x \in U$ into S.
- *Delete:* Delete an item $x \in S$ from S.
- *Query:* Given any particular $x \in U$, decide whether $x \in S$.

A closely related problem is the dynamic dictionary problem, in which the insert operation gets two arguments: an item $x \in U$ and a data element y, which is to be associated with x; when querying x, if the answer is "yes", the data structure should also return x's associated data.

We study the problem in the *external memory* model (EM model) of Aggarwal and Vitter [1]. In this model, the data structure consists of a disk containing a collection

[*]A preliminary version of this article appeared in Proceedings of the 42th ACM Symposium on Theory of Computing (STOC 2010) .

[†]This work was done while Elad Verbin was a postdoctoral researcher in ITCS, Tsinghua University. He was supported in part by the National Natural Science Foundation of China Grant 60553001, and the National Basic Research Program of China Grants 2007CB807900 and 2007CB807901. elad.verbin@gmail.com

[‡]IBM Research Almaden. Corresponding author. Most of this work was done while Qin Zhang was a Ph.D. student in the Hong Kong University of Science and Technology (HKUST). He was supported by Hong Kong Direct Allocation Grant (DAG 07/08) and Hong Kong CERG Grant 613507. Some of this work was done when he was visiting ITCS. zhangqi@us.ibm.com

of b-bit cells (or cells of $B = b/\log u$ words [1]) and a cache of m bits (or $M = m/\log u$ words). The cost of an operation is measured by the number of cells that are *probed* (= read or written). Accesses to the cache and changes to it are free. Thus, the EM model is similar to Yao's cell probe model [33] with an additional free-to-access cache. The cache often drastically changes the behavior of the model, since it enables update operations to potentially take $o(1)$ amortized time, by writing the inserted elements to the cache (for free) and then, every few operations, writing them to the disk all at once using only one probe; this effect is called *buffering* or *batching* and it is one of the most interesting features of the EM model [2]. Another difference between the EM model and the cell probe model is that in the cell probe model we are typically interested in cells of size $b \approx \log u$ bits, while in the EM model we typically care about $b = \text{polylog}(u)$ or even larger values of b. For more information on the EM model, see e.g. the book of Vitter [30]. About data structures in the EM model, also see e.g. [21, 3]. Recall that, in both the cell probe model and the EM model, computation is *free*; the only charge is for probing cells. Lower bounds in these models often have matching upper bounds in more realistic models (e.g. RAM) even though they are significantly stronger than real-world machines in their computational ability. This elucidates the point that in data structures, the bottleneck is often in collecting information for the computation, rather than in performing the computation itself.

It is interesting to note that buffering is the feature that captures the benefit of real-world caching (such as when using RAM as a cache for the hard drive, or using L2 cache as a cache for RAM, etc). In such real-world cases, the cost of accessing the cache is so small compared to accesses to the higher level memory, that it is essentially free. In some types of real-world caching, once a page becomes full we write it all back to higher-level memory, which is exactly what happens in the theoretical concept of *buffering*. Often, the goal of real-world caching is to enable faster data access and data modification. In this sense, the membership problem in the external memory model is a fundamental problem in the context of data access in real-world computing. It is well known that much of the machine time of databases, routers, search engines, and many other applications is taken by hashing and other types of membership searching, and that most of this time is spent on copying data from higher levels in the memory hierarchy. Thus, any improvement on membership in the external memory model would mean immediate and significant speedups in many applications. (And it is therefore somewhat tragic that in this paper we show that dramatic improvements on the current state of the art are impossible, at least under the EM theoretical model).

So, how fast can we solve membership in the EM model? When looking at the EM literature, it quickly becomes clear that there are two radically different approaches to solving the membership problem, which are disconnected from one another: hashing and buffering. In the hashing-type approaches (e.g. [30, chap. 10], [21, sec. 4]), a hash table is used, and the goal is to make the operations as efficient as possible, but it is always implicit that they cannot take less than 1 probe each (hence, no buffering is achieved). The query times are typically $O(1)$ with a very small constant in the big-Oh, or even $1 + o(1)$. The goal is to make the constants as good as possible, under various measures (worst case, average case, etc). On the other hand, in the

[1] It is convenient to use *bit* in our lower bound proof, while it is more convenient to use *words* when presenting various upper bounds in the literature. They are essentially the same – only differ by a factor of $\log u$.

[2] To get an idea of this, the reader can think of the way paging is done, in which we always try to put the best set of pages into the cache so that we can access them for free in the future as long as they have not been replaced.

buffering-type approaches (e.g. [30, chap. 11], [21, sec. 3.5]), various ways to achieve buffering are explored, and typical solutions such as the ubiquitous *buffer tree* (see Section 1.2) achieve an amortized update time of $O(1/B^{1-\epsilon})$ (for some small constant ϵ), which is much smaller than 1 if B is large enough. However, all of these buffering-type approaches take query time roughly $\log_B n$ [3] (In order to simplify and compare various bounds, in the introduction of this paper we always assume that $M \geq B^{1+\epsilon}$ (the tall cache assumption), $b \geq \log u$ and $n \geq m^{1+\epsilon}$ where ϵ is some arbitrary small constant. But we do not need any of these assumptions in our lower bound proof).

Few texts discuss why these two types of approaches are so different. Is there a way to get the best of both worlds: a data structure with amortized update time $o(1)$ and query time $O(1)$? This fundamental question has often been asked informally, and was also asked explicitly by Jensen and Pagh [14]. In this paper we give a very strong negative answer to this question: we prove a threshold result – that if the amortized update time is any constant bounded below 1 (for example, 0.9), then the query time must be roughly at least logarithmic in n, namely $\geq \Omega(\log_b n)$. See Theorem 1.1. This is an unusually delicate threshold behavior for data structures problems, and seems to say something deep about the EM model.

Since membership is one of the easiest data structures problems (in the sense that it is reducible to most other data structure problems) it follows that this lower bound applies for most other data structure problems as well (such as dictionary, predecessor, prefix sum, range counting, union find, rank-select, and many others; some reductions will be shown in the Section 3.5).

In the remainder of this section we present our results, describe the buffer tree, and discuss related work. In Section 2 we try to convey the intuition of our lower bound via a fake proof, which is quite simple (but not formally true) and captures many of the ideas behind our result. In Section 3 we present the real, formal, proof of our lower bound. Finally we conclude the paper in Section 4.

1.1. Our results.

The statement of our lower bound is:

THEOREM 1.1. *Suppose we insert a sequence of n elements chosen uniformly at random, into any initially empty randomized data structure in the EM model with cell size b and cache size m. If the expected total cost of these insertions is $n \cdot t_u$, and the data structure is able to answer a membership query with expected average t_q probes,*[4] *then we have the following tradeoff: If $t_u \leq 0.9$, then $t_q \geq \Omega(\log_{b \log n}(n/m))$. This holds provided that $u \geq c \cdot n$ for some large enough constant c. It holds even when we measure t_u only by the number of cells written.*

This theorem settles a folklore conjecture (explicitly proposed by Jensen and Pagh [14]), thereby completing the line of research of [31] and [35].

It is important to note that our bound does not assume indivisibility (that is, each element should be stored atomically in the data structure). The ideas behind it might be useful for further bounds in the EM model that do not assume indivisibility. Also, our lower bound holds even if we only count writes. This is also a notable factor since typically writes are much slower and cost more energy than reads.

As discussed before, the constant 0.9 in Theorem 1.1 is arbitrary, and can be replaced by any constant < 1. The constant c and the constants hidden in the big-Ω

[3] Here we require that the queries should be answered immediately. If queries can be delayed, then the amortized cost can be made the same as that of updates. See more discussions in Section 1.2.

[4] That is, at any time during the updated sequence, if we query uniformly at random an element from the universe, then the expected cost of the query should be no more than t_q. All expectations are taken over all possible update sequences.

notation are quite wild, as is common with lower bounds. We did not make any effort to optimize them. We are reasonably sure that the real constants are quite tame, and since this makes a big difference in practice, it might be interesting to perform more careful analysis in order to pinpoint the right constants.

Theorem 1.1 is proved in Section 3, but only for deterministic data structures under an input distribution. The theorem can be generalized to hold for randomized data structures using a Minimax principle similar to Yao's [32]; see the discussion in [35].

Another contribution of this paper is the *Lemma of Surprising Intersections* (LOSI) which might be of independent interest. It is first discussed informally in Section 2, then discussed and proved in Section 3.4. The data-structural threshold phenomenon that we prove in fact arises from and corresponds to a threshold phenomenon in the LOSI.

1.2. The Buffer Tree. We describe here the ubiquitous buffer tree which supports very strong buffering in the EM model. It is relevant here for two reasons: first of all, since we are claiming to prove tight lower bounds, it makes sense to show how the corresponding upper bounds were obtained; secondly, our intuition for the lower bound can be seen to *match* the upper bound in some sense (this is quite common with tight lower bounds), so the upper bound actually helps to get intuition for the lower bound.

Here we sketch a construction of a buffer tree, which is a variant of the one in [4]. λ ($2 \leq \lambda \leq B$) is an adjustable parameter used to adjust the tradeoff between t_u and t_q. A buffer tree is a tree where the degrees of all nodes except the root are between $\lambda/4$ and λ. The height of the tree is $\Theta(\log_\lambda n)$. The root and each internal node consists of a buffer containing M universe elements. The buffer of the root always resides in the cache, and the rest resides in the disk.

Intuitively, the buffer tree processes updates in a "lazy" fashion. For example, when we try to insert an item, we do not trace all the way down the tree and find the right place to perform the insertion. Instead, we just put it in the root. We do this until the root is half-full, and then we insert all of its items into the buffers of its children; the elements are distributed among the children according to their natural order, just like in a regular B-Tree. Similarly, whenever an internal node is half-full, we push all element in its buffer to its children. Deletions are done similarly (pushing them down until they encounter the element they are meant to delete). Queries can be done similarly, by lazily pushing them down the tree, if we are content with the data structure returning the answer to queries at a later time. Rebalance operations need to be done like in a B-Tree, but also lazily. In this way all the operations can be performed in a batched fashion, thus the amortized cost of each operation is $O(\frac{\lambda}{B}\log_\lambda n)$. However, if we want a query to be answered immediately, as is required in our problem, then we have to at least walk down a root-to-leaf path for each query in order to find the answer, which takes $\Omega(\log_\lambda n)$ time.

It is thus interesting to note that the lower bound in this paper crucially relies on the need to answer queries immediately. If answers to queries can be delayed, a buffer tree can support both updates and queries in time $o(1)$ (when B is large enough).

By storing some additional data, the buffer tree can also be made to solve many other data structure problems with the same update times, see e.g. [4].

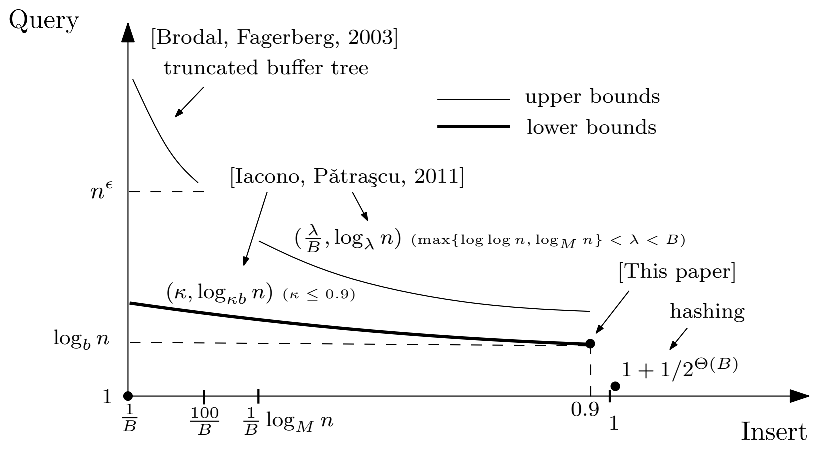
1.3. Related Work. Dynamic membership has a long history of research. Here we review just the works that are most relevant to ours. On the upper bound side, Knuth [15] already showed that using hashing with standard collision resolution strategies (e.g. linear probing or chaining) gives a data structure for dynamic membership with both average query time (both successful and unsuccessful) and amortized update time only slightly larger than 1, namely $t_q = t_u = 1 + 1/2^{\Omega(B)}$. In a follow up work, Jensen and Pagh [14] showed how to design a hash table that uses close to $n \log u$ bits (which is roughly the optimal space) and has update time and query time both $1 + O(1/\sqrt{B})$. If t_q is required to be worst case, Pagh and Rodler [23] proposed an interesting hashing method called *cuckoo hashing*; in this method $t_q = 2$ and t_u is still a constant in expectation. All these upper bounds work for both dynamic dictionary and dynamic membership. Intuitively, membership may have better upper bounds than dictionary, but we are not aware of any data structure that solves membership strictly better than dictionary. The *Bloom filter* [7] achieves better space utilization than all data structures that solve dictionary, but it has query and update cost larger than some constant> 1, and also contains false positives. All these are hashing-type data structures, and more related works on external hashing can be found in [11, 17, 16].

Another class of data structures that solve the dynamic membership problem is the buffering-type data structure, for example, the buffer tree and several of its variants developed by Brodal and Fagerberg [8]. All of these buffering-type data structures can achieve $t_u = o(1)$ (when B is large enough), but $t_q = \Omega(\log_B n)$.

As for lower bounds, one of the previous works that is conceptually closest to ours is Brodal and Fagerberg [8]. They prove a lower bound of $t_q \geq \log_B n$, regardless of what t_u is, in the *comparison model*, which assumes that elements are indivisible and the only operation allowed on them is comparison. This model obviously does not support hashing, and is in general too weak to indicate what is possible on a real machine. One of the other lower bounds that they prove is very interesting – they prove that $t_q \geq n \big/ M^{O(Bt_u)}$ also in the comparison model. For $t_u = O(1/B)$ this gives $t_q \geq n^{\Omega(1)}$ for sufficient large n. We believe that this is essentially true in the general EM model as well, but proving it might be beyond the reach of current technology.

What is known outside of the comparison model? If we consider the worst case operation cost, Pagh [22] proved that t_q must be at least 2, even for static membership, meaning that cuckoo hashing is optimal in terms of worst-case query time. In [9, 18, 29] some super-constant lower bounds on $\max\{t_q, t_u\}$ are shown but they are proved in models more restrictive than (or incomparable to) the EM model. There are very few lower bounds for dynamic membership in the RAM model. The reason might be that if there is no cache, then each update has to be committed immediately to the disk, thus t_u must be at least 1. Therefore Knuth's upper bound is already tight up to an $o(1)$ additive term. In the EM model, t_u does not necessarily have to be at least 1 if the size of a cell is large, so there was some hope (dispelled by our lower bound) to get update time less than 1 while maintaining good query time. To prove this is impossible, [31, 35] gave a set of tradeoff curves between t_q and t_u. Concretely, they showed that the amortized update cost is at least $\Omega(1)$ if the average query cost is required to be no more than $1 + \delta$ for any constant $\delta < 1/2$. This result already ruled out the possibility of achieving $t_q = 1 + o(1)$ and $t_u = o(1)$ simultaneously. [31] also studied the radically different case where we only measure the average query time for positive queries.[5] In this case, [31] show a data structure achieving both $t_q = 1 + o(1)$ and $t_u = o(1)$. Therefore, there is a gap between the case of supporting

[5]That is, t_q is defined as the expected query cost of a random $x \in S$, instead of $x \in U$.



B is the size of a cell. M is the size of the cache. Both in words. $b = B \log u$.

FIG. 1.1. *Some of the known upper and lower bounds on membership data structures in the external memory model. In order to provide a unified presentation of all the bounds, we assume for simplicity that $M \ge B^{1+\epsilon}$, $b \ge \log u$ and $n \ge m^{1+\epsilon}$.*

only successful queries and the case of also supporting unsuccessful ones.

Very recently, after the conference version of this paper, Iacono and Pătraşcu [13] showed that for any $\max\{\log\log n, \log_M n\} \le \lambda \le B$, there exists a data structure achieving $t_u = O(\lambda/B)$ and $t_q = O(\log_\lambda n)$. Their data structure is not comparison based and outperforms the buffer tree for $t_u = O(\frac{1}{B}\log_M n)$. In the same paper they also proved a matching lower bound. Their lower bound is essentially the same as ours except that it slightly improves for very small $t_u = O(\frac{1}{B}\log_M n)$. The ideas of their proof are also quite similar to ours. Their proof is done by pure information theoretic arguments and shorter than ours, but we feel that it somehow hides the intuition behind. Therefore we think our combinatorial proof (plus a few information theoretic arguments) is still interesting in that it exposes the underlying intuition better.

Figure 1.1 depicts some of the known upper bounds and lower bounds.

The techniques used in this paper are somewhat related to the recent lower bounds on succinct data structures in the cell probe model, of Golynski [12] and of Pătraşcu and Viola [27]. Their bounds can also be thought of as performing a "cache elimination" step (like we do in Section 2) interleaved with a "probe elimination" step. For the "probe elimination" step we use the LOSI, and their bounds use other ideas.

2. Intuition of the Lower Bound via a Fake Proof .

In this section we provide an exposition of our results that is not strictly true, but captures many of our ideas in an easy-to-understand setting, and in fact represents the way that we came up with the solution in the first place. We believe that this section not only helps to explain our bound, but actually might be useful for novices to the field, by showing how very simple (but technically false) arguments give the "right" lower bounds, as well as the right intuition, for data structure lower bounds. These arguments can then be replaced by a correct (but more complicated) proof, which will be our task in Section 3 of the paper.

In the presentation in this section we cheat constantly, with the goal of getting across most of our ideas, without the required hard work. Thus, we warn in advance that nothing in this section should be assumed to be strictly correct, and the whole section should be treated as just intuition.

We begin by discussing the cache elimination principle. Let n, u, m, t_q, t_u denote the same quantities as defined in the introduction: the number of items, universe size, cache size (in bits), query time and update time, respectively.

The Cache Elimination Principle. For most reasonable problems P, if there is a data structure for problem P with parameters (n, u, m, t_q, t_u) then there exists a data structure for problem P with parameters
$(n/10m, u/10m, 0, t_q, t_u)$.

Fix $t_u = 0.9$ (any other constant less than 1 would work as well). The framework of the fake proof is to use the cache elimination principle alternately with the following probe elimination step:

LOSI-Based Cache Augmentation. For the membership problem, in the case where there is no cache, if there is a data structure with parameters $(n, u, 0, t_q, t_u)$ then there is a data structure with parameters $(n, u, t_q b, t_q - 0.2, t_u)$.

By using these two steps alternately, one can start with a structure with parameters (n, u, m, t_q, t_u) and end up with a structure with parameters

$$(n/m(10t_q b)^{5t_q}, u/m(10t_q b)^{5t_q}, 0, 0, t_u) ,$$

which implies that $n/m(10t_q b)^{5t_q} \leq 1$, and this gives

$$t_q = \Omega(\log_{bt_q}(n/m)) = \Omega(\log_{b \log n}(n/m)) ,$$

which is the lower bound that we desire.

It is interesting to observe that this string of "reductions" of parameters matches more or less the structure of the buffer tree. The LOSI-based cache augmentation step corresponds, in some sense, to putting the root of the tree in the cache, thus saving one probe in each query.

We now give more details on the fake proof, in particular on the cache elimination principle and on the LOSI-based cache augmentation step.

The cache elimination principle is analogous to the round elimination lemma of Miltersen et al. [19] or Sen [28] or the notion of *published bits* found in various works of Pătraşcu and his collaborators (see for example [26, 27]). We omit here a more comprehensive discussion of the cache elimination principle, and defer it to Section 2.2. The principle will be easier to understand after we show how we use it.

The fake proof works only for non-adaptive queries, i.e. for the case where the set of cells $q(x)$ which are probed by query(x) depends only on x and not on the state of the cache or of the disk. Also, it works only when t_u is defined as the expected cost of a random update, rather than as the amortized cost. Both of these restrictions can be lifted in our formal proof.

We now describe the LOSI-based cache augmentation step. Let us start with a data structure for dynamic membership with parameters $(n, u, 0, t_q, t_u)$. We are going to heavily exploit the fact that there is no cache. Consider two uniformly random insertion operations, ins(x) and ins(y), which we assume are processed together, at a total expected cost of $2t_u$ probes. Since there is no cache, these operations must make at least one probe (in total). Denote by p the probability that they make exactly one probe in total. We know that their expected total cost is $\geq p + (1-p) \cdot 2$. Thus $2t_u \geq p + 2(1-p)$, which gives $p \geq 2(1 - t_u)$. Therefore, with probability at least $2(1 - t_u)$, these two updates make only one probe. From now on consider the case that they make only one probe in total. Denote by c the common cell that they probe. Consider the query operations query(x) and query(y). Since there is no cache, both of these operations must probe cell c, because it contains the only indication in the data structure that x and/or y were inserted.

We thus see that for two random elements x and y, with probability $\geq 2(1-t_u)$, the queries query(x) and query(y) probe a common cell.[6] We claim that this is a "surprising intersection" (or "surprising collision"). To see that this statistical behavior is surprising, consider hashing. In hashing, the queries are roughly equally split among the cells, and the collision probability of two random queries is roughly $1/n$. In our case, when $t_u = 0.9$, the collision probability is 0.2 – a constant!

Our main technical contribution in this paper is the Lemma Of Surprising Intersections (LOSI) which shows that if such surprising statistical behavior occurs, that must be because the queries have some rigid structure; namely, it shows there must be a set C that consists of only a few cells, such that C intersects a constant fraction of all query-paths $\{q(x) \mid x \in U\}$. Here is a trivial version of the LOSI, which suffices for our purposes in this fake proof. The full (and considerably more involved) LOSI can be found in Lemma 3.5.

Lemma 2.1 (LOSI – Very Simple Version). *Let $\mathcal{F} = \{S_1, \ldots, S_u\}$ be a family of sets with the following property: When choosing two random elements $x, y \in U$, $\Pr_{x,y}[S_x \cap S_y \neq \emptyset] \geq 0.2$. Then if we pick $x \in U$ at random, we get that $\mathbf{E}_x[|\{y \in U \mid S_x \cap S_y \neq \emptyset\}|] \geq 0.2u$.*

Proof. Choose x uniformly from U. Then by the condition, with probability 0.2 over the choices of y, it holds that $S_x \cap S_y \neq \emptyset$. The conclusion follows by using linearity of expectation. $\square$

By substituting $S_x = q(x)$ (recall that the queries are non-adaptive), we see that when we choose x randomly, $q(x)$ intersects $0.2u$ $q(y)$'s. Also, since the average query time is t_q, the set $q(x)$ consists of roughly t_q elements.

Now, to get the LOSI-based cache augmentation step, we take the elements of $q(x)$ and place them into the cache. This saves one probe for roughly a 0.2 fraction of the universe, because any probe that used to read one of the elements of $q(x)$ can now read the cache instead. Thus, from our data structure that had parameters $(n, u, 0, t_q, t_u)$ we get a data structure with parameters $(n, u, t_q b, t_q - 0.2, t_u)$. (The $t_q b$ appears because we put t_q cells in the cache, and each cell consists of b bits).

After applying both types of reductions $5t_q$ times, we would get a data structure with parameters

$$(n/m(10t_q b)^{5t_q}, u/m(10t_q b)^{5t_q}, 0, 0, t_u) \ .$$

This is an absurdity – we get correct answers to queries, without having any cache and without any probes. Therefore, this must mean that the data structure maintains less than one element, so that $n/m(10t_q b)^{5t_q} \leq 1$. This gives $t_q = \Omega(\log_{bt_q}(n/m)) = \Omega(\log_{b \log n}(n/m))$, which is the lower bound that we desired.

In Section 3 we turn this fake proof into a correct one. In particular, we develop a much stronger version of the LOSI, and we replace the cache elimination arguments (which are intuitive but incorrect) by encoding-based arguments (which are both intuitive *and* correct, but make the other arguments more complicated). We comment

[6] Note that if $t_u < 1/2$ then this probability is > 1 and this argument explodes. This is one way to see that we are cheating here. The strange behavior here comes from the fact that we ignored the error term created by cache elimination. However, we will just persist, and refer the suspicious reader to the full proof in Section 3.

that such a formalization is not a direct translation. First, in the fake proof we perform reductions of parameters to obtain an impossible data structure, while in the real proof we directly count the average number of queries. Second, in the fake proof we eliminate query probes by increasing the error (see the discussion in Section 2.2), while in the real proof we eliminate query probes by keeping adding cells in the disk to the cache (note that probing the cache is free of charge).

2.1. Discussion of the LOSI. One interesting thing to note about the LOSI is that it allows us to argue about the structure of negative queries (ones whose answer is "false") from statistical information that we know about (much fewer) positive queries. In a sense, the LOSI allows us to argue about the structure of queries *that we know nothing about*, from statistical properties of probe-sequences of queries that we do know something about.[7]

The LOSI might be of special importance in the dynamic membership problem, as explained by the fact that there is a data structure for dynamic membership in the EM model that uses update time $o(1)$ and average query time $1 + o(1)$ when the average is taken only over positive queries [31], so to get our lower bound we really need to argue about negative queries as well. This is a difficult task when using only traditional encoding-based techniques [8], since negative queries do not correspond to any inserts that were done, and thus do not correspond to any change to the data structure.

We thus suspect that the LOSI and the idea behind it (first proving statistical properties of certain queries using encoding arguments, and then using statistical reasoning to get implications about the set of all queries) might be of future use, particularly for problems where positive queries are easier to answer than negative queries. It seems like an interesting question to find data structure problems that exhibit this property, and in general to find further applications of our lower bound scheme.

2.2. Discussion of the Cache Elimination Principle. The intuition behind the cache elimination principle is that if we divide the domain into $10m$ parts then the cache contains, on average, 0.1 bits of information about each part. We then restrict our operations only to one part, chosen randomly among all $10m$ parts. Then, since we are working only in one part, the cache provides only roughly 0.1 bits of information, so it can be completely eliminated with a small increase in error (for example by using the average encoding lemma, see Sen [28]).

To make the cache elimination principle formally true, one has to introduce error, and to make some (not too restrictive) explicit demands on the problem P. Then one can prove the cache elimination principle for *static* data structure problems quite easily; one way to do so is to use an analogous proof to Sen's proof of the round elimination lemma [28]. However, for dynamic problems no formally correct proof of anything like cache elimination is known, due to various technical difficulties that we do not detail here. It would be very interesting if a proof was found for something like the cache elimination principle, since this has the potential to simplify many lower bound proofs.

[7] Of course, it seems odd to discuss at such length a lemma with a two-line proof; however, the strong LOSI (Lemma 3.5) is more involved, and it seems necessary when one wishes to get a non-fake proof of our results.

[8] We noticed that after the conference version of this paper, Iacono and Pătraşcu [13] did translate our arguments to an encoding-based proof, but we feel that it somehow hides the original intuition.

3. The Lower Bound. In this section we prove Theorem 1.1. First we describe the general framework of the proof. Following that we highlight two lemmas that express two key ideas, and they together give the lower bound. Finally we prove the two lemmas.

3.1. The framework of the proof. Let $\gamma = 1/400000$. Let l be chosen large enough such that $l \geq 80000 t_q b/\gamma$. We also assume that $t_q = O(\log n)$, since otherwise we are already done.

For any $x \in U$, let $q(x)$ be the query path of x, that is, the set of cells that are probed when querying x. $q(x)$ is actually determined by the cache state and disk state at the time of the query. For an arbitrary cache state M and disk state D, we write $q_{M,D}(x)$ explicitly whenever M and D are not clear from context. It is possible that for some combination of M and D, $q_{M,D}(x)$ is not well defined – certain combinations of cache and disk states cannot simply arise. In such cases, we simply set $q_{M,D}(x) = \emptyset$. When we talk about $q(x)$ *at some time snapshot*, we mean $q(x)$ under the cache state and disk state at that time snapshot, and sometimes we denote this by $q_{\texttt{snapshot}}(x)$ for convenience.

We consider an update sequence of n insertions, each of which is chosen uniformly at random from U with replacement. We think of these n insertions as being performed on n subsequent "time steps", from time 1 to time n. In the following, we assume for simplicity that all insertions are different. It is quite easy to adapt the argument to the case where the inserted elements can have repetitions, but we do not show this here. The suspicious reader can choose the universe size to be at least n^3, in which case the birthday paradox guarantees that with probability $1 - O(1/n)$ there are indeed no repetitions and the rest of the argument goes through easily.

We pick a time snapshot, denoted by `END` uniformly at random between time $n/1000$ and n.[9] Most of our analysis will focus on the time `END`. We partition the insertions before `END` into exponentially growing epochs: The first epoch contains the set of insertions Y_1 ($|Y_1| = m/\gamma$) immediately before `END` and epoch i consists of the set of insertions Y_i ($|Y_i| = ml^{i-1}/\gamma$) immediately before epoch $i-1$. Note that we number the epochs in the reverse order: the first epoch consists of the insertions that are done last. The total number of epochs created is $d = \Theta(\log_l(n/m)) = \Omega(\log_{b \log n}(n/m))$.

For epoch i, let C_i^u be the set of cells that are probed during the insertions of Y_i. Let $\texttt{BEGIN}_i$ be the snapshot just before the i-th epoch. Let Y_i' be a set of elements of size $|Y_i|$ chosen uniformly at random from U. Let $\texttt{END}_i'$ (sometimes just denoted $\texttt{END}'$) be the snapshot at the same time as `END` created by inserting Y_i' instead of Y_i at epoch i. Let $C_i'^u$ be the set of cells probed during the insertions of Y_i', and let $C_i'^q$ be the set of cells probed when querying all elements $y \in Y_i'$ in the state $(M_{\texttt{END}}, D_{\texttt{BEGIN}_i})$ [10]. Let $C_i^* = C_i^u \cup C_i'^q$ and $C_{<i}^* = \bigcup_{j<i} C_j^*$. The lower bound follows from the following lemma:

LEMMA 3.1. *Suppose that $t_u \leq 0.9$ and pick $x \in U$ uniformly at random. Let $\mathcal{E}_i$ be the indicator random variable which is 1 if $q_{\texttt{END}}(x)$ intersects $C_i^* \backslash C_{<i}^*$. Then $\mathbf{E}[\mathcal{E}_i] \geq \Omega(1)$ for all $i = 1, 2, \ldots, d$.* Lemma 3.1 directly implies by linearity of

[9] This is done for technical reasons, in order to get the lower bound to apply for amortized update time. If one just wants t_u to denote the average time for a randomly-selected insert, then `END` can be selected to simply be time n.

[10] One may wonder why we use the combination of the `END` state of M and the $\texttt{BEGIN}_i$ state of D. This is done for technical reasons. In particular, $q_{\texttt{END}}(x) = q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(x)$ if $q_{\texttt{END}}(x)$ does not intersect those updated cells in the disk. We believe one can get a better understanding after reading the last paragraph of Section 3.4.

expectation that for a random x, $\mathbf{E}\left[|q_{\texttt{END}}(x)|\right] \geq \Omega(d)$ and this immediately gives Theorem 1.1. Thus, we just need to prove Lemma 3.1. From now on we concentrate on one epoch i and prove that $\mathbf{E}[\mathcal{E}_i] \geq \Omega(1)$.

3.2. Proof of Lemma 3.1. In this section we prove Lemma 3.1. We start by giving two lemmas. The first lemma (Lemma 3.2) will actually be used to prove the second lemma (Lemma 3.3). We highlight it here since it expresses one of our key ideas: Those query-paths of newly inserted elements will intersect a small set of cells if the cache is small. And then we can use LOSI to generalize this to all query-paths of universe elements. The proofs of the two lemmas will be given in the next two subsections.

LEMMA 3.2. **(the encoding lemma)** *Let m be the cache size and suppose Y is the set of the last $|Y|$ random insertions. Let C^u be the set of cells probed by the insertions in Y. Also assume that $m \leq \gamma |Y|$. Let* `END` *be the time after inserting Y and* `BEGIN` *be the time before inserting Y. Then with probability at least* 0.999 *over the choices of Y, both the following hold:*

1. $|\{y \in Y \mid q_{\texttt{END}}(y) \cap C^u \neq \emptyset\}| \geq 0.99\,|Y|$
2. $\left|\{y \in Y \mid q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(y) \cap C^u \neq \emptyset\}\right| \geq 0.99\,|Y|$.

LEMMA 3.3. **(application of LOSI)** *Under the same conditions as Lemma 3.2, let Y' be another randomly-drawn set of $|Y|$ insertions. Let C'^u be the set of cells probed during the insertions of Y' and $C'^q = \bigcup_{y \in Y'} q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(y)$. Let $C^* = C^u \cup C'^q$. If $|C'^u| \leq 0.91\,|Y|$ and $m \leq \gamma |Y|$, then with probability at least* 0.999 *over the choices of Y and Y', it holds that $|\{x \in U \mid q_{\texttt{END}}(x) \cap C^* \neq \emptyset\}| \geq \Omega(u)$.*

Lemma 3.3 above implies the correctness of Lemma 3.1 for the first epoch by substituting $Y = Y_1$, $C = C_1$, etc. To prove Lemma 3.1 for epoch $i > 1$, we make the following changes to the data structure:[11] We put all cells $C^*_{<i}$ into the cache. Then, every time that a probe wishes to read or change a cell that we put in the cache, we redirect it to the corresponding location in the cache. This new data structure can simulate the old one, and notice that for this new data structure, C^*_i is $C^*_i \backslash C^*_{<i}$. Furthermore, the following two properties are immediate consequences of our conceptual operation. They will be used in various places in the proofs of Lemma 3.2 and Lemma 3.3.

- The state of the disk at the end of epoch i is identical to that in `END`.
- The size of the cache in this modified data structure is $m_i = m + \left|C^*_{<i}\right|$.

We say epoch i is *good* if: (1) $|C'^u_i| \leq 0.91\,|Y_i|$, and (2) $m_i \leq \gamma |Y_i|$ after the conceptual operation. Note that Lemma 3.3 actually implies Lemma 3.1 provided that every epoch i ($1 \leq i \leq d$) is good with probability at least 0.002, which indeed holds by the following lemma.

LEMMA 3.4. *Every epoch i ($1 \leq i \leq d$) is good with probability at least* 0.002 *over the random choices of* `END`*, and the choice of the insertion sequences.*

Proof. Consider requirement (1). The expected update cost between time $n/1000$ and n is at most $0.9n$. Averaging over our choice of `END`, the total update cost of an epoch of length $|Y_i|$ is $\frac{1}{0.999} \cdot 0.9\,|Y_i|$ in expectation. By Markov's inequality, the probability that this update cost is more than $0.91\,|Y_i|$ is at most 0.006. As for requirement (2), we union bound over the events that any of the sets C^*_j ($j < i$) is too large. The expected total size of these sets is $\mathbf{E}[\left|C^*_{<i}\right|] \leq \left(\frac{t_u}{0.999} + t_q\right) \sum_{j<i} |Y_j| \leq \gamma |Y_i|/20000b$. Again by Markov's inequality, with probability at least 0.999 over the

[11]We can change the data structure since we just need to prove Lemma 3.1 for epoch i. This is why it was important to use linearity of expectation.

choice of END, the input sequence and the random queries, it holds that $\left|C^*_{<i}\right| \leq \gamma |Y_i| /2b$. Consequently, $m_i = m + |Y_i| \cdot b \leq \gamma |Y_i|$. Combining (1) and (2), we have that with probability at least $0.006 - (1 - 0.999) \geq 0.002$, epoch i is good. □

We give the proofs of Lemma 3.2 and 3.3 in Section 3.3 and 3.4, respectively. The proof of Lemma 3.2 uses an information theoretic argument and the proof of Lemma 3.3 explores the underlying combinatorial structures of the set of query-paths $\{q(x) \mid x \in U\}$.

3.3. Proof of Lemma 3.2. *Proof.* It is easy to see that if the first item of the lemma holds, then the second item holds, since the only changes to the disk between $D_{\texttt{BEGIN}}$ and $D_{\texttt{END}}$ are in the cells that are in C^u, so the set in item 1 and the set in item 2 are actually the same set. From now on we shall just prove the first item holds.

We consider all query-paths $\{q(y) \mid y \in Y\}$. The idea behind the proof is to show how to "compress" (= to encode) the set Y by just specifying the elements $y \in Y$ that *do* touch C^u, and specifying the state of the cache at the end of the insertions. Then, since Y cannot be "compressed" beyond the standard information-theoretic limit, we get that the size of the encoding must be large, which means that many elements $y \in Y$ indeed *do* touch C^u, which is what we needed to prove.

The trick is to get convinced that this "compression algorithm" indeed compresses Y in a way that Y can be decoded from the compressed representation. Following is the full proof.

Let Z be the set of queries y in Y such that the query path $q(y)$ as it is performed at END *does* intersect C^u. We try to prove that with probability at least 0.999 over the choices of Y, it holds that $|Z| \geq 0.99 |Y|$.

In the encoding, we condition on the elements that were previously inserted. By "conditioning" we mean that both the compressor and the decompressor know which elements were inserted and in what order they were inserted. We call this the "shared information" (since it is shared between the compressor and the decompressor). It is easy to see that given the shared information, the decompressor can compute the state of the cache and of the disk at the beginning of the insertions of Y. (This uses the fact that the data structure is deterministic). The encoding of Y consists of the following.

1. The state of the cache at the end of the insertions of Y. This takes m bits.
2. The elements of Z. This takes at most $\log \binom{u}{|Z|}$ bits, which is at most $|Z| \log(eu/|Z|)$.[12]

We now observe that given the encoding of Y and the shared information, the decompressor can answer any query(x) $(x \in U)$ correctly, as follows: (i) if x is already inserted before Y, answer "yes". (ii) If $x \in Z$, answer "yes". (iii) If both (i) and (ii) did not return "yes", then run the query algorithm with the state of the cache at END and with the state of the disk as it is *before* the insertions of Y. By the definition of Z, and since $x \notin Z$, this returns the right answer. Therefore, we managed to encode Y into $m + |Z| \log(eu/|Z|)$ bits.

Now, suppose in contradiction that with probability more than 0.001, $|Z| \leq 0.99 |Y|$. With the rest of the probability, $|Z|$ can be as large as $|Y|$. Then the

[12]Technically speaking, we also have to encode the cardinality $|Z|$ itself, which takes another $\log u$ bits. This does not change the computation significantly, so we ignore it.

expected size of the encoding is at most

$$m + 0.999\,|Y| \cdot \log\frac{eu}{|Y|} + 0.001 \cdot 0.99\,|Y| \log\frac{eu}{0.99\,|Y|}$$
$$\le m + 0.99999\,|Y| \log\frac{eu}{0.99\,|Y|} .$$

On the other hand, there are at least $\binom{u-n}{|Y|} \ge \binom{u/2}{|Y|} \ge 2^{|Y|\log\frac{u/2}{|Y|}}$ possibilities to choose Y. Since we can never encode x bits into $x-1$ bits, we get that:

$$m + 0.99999\,|Y| \log\frac{eu}{0.99\,|Y|} \ge |Y| \log\frac{u/2}{|Y|}$$
$$\Rightarrow m \ge |Y| \log\left(\left(\frac{u}{|Y|}\right)^{0.00001} \cdot \frac{1}{4e}\right) > 0.000005 \cdot |Y| \; .$$

The last inequality holds since $u > c \cdot n \ge c\,|Y|$ for some sufficiently large constant c. This contradicts the fact that $m \le \gamma\,|Y| < 0.000005\,|Y|$. ◻

3.4. Proof of Lemma 3.3. In this section we first state and prove the LOSI. It is essentially an involved combinatorial argument that deduces a global behavior from an observed local behavior. We then prove Lemma 3.3, which is an application of the LOSI to our setting.

LEMMA 3.5. *[Lemma Of Surprising Intersections (LOSI)] Let k be an integer, and let $p = 2^{-k/40000}$. Let $\mathcal{F} = \{S_1, \ldots, S_u\}$ be a family of sets. Suppose that when we choose $Y \subseteq [u]$, $|Y| = k$, out of all size-k sets uniformly at random, with probability at least p the following holds:*

$$\begin{array}{c} \textit{there exists a set } S \textit{ such that:} \\ (i)\ |S| \le 0.91k, \textit{ and} \\ (ii)\ |\{y \in Y \mid S_y \cap S \ne \emptyset\}| \ge 0.99k \; . \end{array} \tag{3.1}$$

Then it follows that when we pick a family Z of k random sets from the collection, their union intersects an $\Omega(1)$ fraction of the sets in expectation,[13] *that is,*

$$\mathbf{E}_{Z \subseteq U, |Z|=k}[|\{x \in U \mid \exists z \in Z, S_x \cap S_z \ne \emptyset\}|] \ge \Omega(u) \; .$$

The LOSI is tight, up to constants, in most of its parameters. In particular, it is not true that one can always choose Z to be of cardinality much smaller than k. To get one example for this, construct a family $\mathcal{F}$ such that $S_1 = S_2 = \ldots = S_{10^6 u/k} = \{1\}$, $S_{10^6 u/k+1} = S_{10^6 u/k+2} = \ldots = S_{2 \cdot 10^6 u/k} = \{2\}$, and so on. Then the condition of the LOSI holds, but to get the conclusion of the LOSI, Z really needs to be of size $\Omega(k)$.

What makes the LOSI non-trivial to prove is the fact that p can be exponentially small in k. If p was constant, then the LOSI would be easy to prove using a simple averaging argument, similar to the proof of the simple LOSI in Section 2; but since p can be exponentially small, we need to carefully use a concentration bound, which makes the LOSI somewhat more complicated to prove.

[13]Notice that both the condition of the LOSI and the conclusion of the LOSI are deterministic statements, that use probability only to replace counting. We could formulate it as an entirely deterministic statement that uses no probabilistic notions. It is important to note that the LOSI is a combinatorial statement, not a probabilistic one.

We need two structural lemmas, Lemma 3.6 and Lemma 3.7, in the proof. For a graph $G = (U, E)$ and a subset $Z \subseteq U$, let $G[Z]$ denote the induced subgraph of G on Z, let $\mathsf{MAX\text{-}IS}(G[Z])$ denote the size of a maximum independent set in $G[Z]$, and let $\deg_G(x)$ denote the degree of vertex x in G.

Lemma 3.6. *Let $G = (U, E)$ be a simple graph such that*

$$\Pr_{Z \subseteq U, |Z|=k}[\mathsf{MAX\text{-}IS}(G[Z]) \geq 0.97k] \leq 1 - e^{-k/40000} .$$

Then $|\{x \in U \mid deg_G(x) \leq u/100k\}| \leq 0.99u$.

Proof. Let W be the set of vertices in G that have degree at most $u/100k$, and let $S = U \backslash W$. We prove the contrapositive of the lemma: that if $|W| > 0.99u$, then $\Pr_{Z \subseteq U, |Z|=k}[\mathsf{MAX\text{-}IS}(G[Z]) \geq 0.97k] > 1 - e^{-k/40000}$.

Consider a random variable which represents one of the vertices of Z, say the one chosen first. Call this random variable v. The probability that v has at least one neighbor in Z is at most

$$\Pr[v \in W] \cdot (k-1) \cdot 1/100k + \Pr[x \in S] . \qquad (3.2)$$

This arises from the fact that if $v \in W$, then it has degree at most $u/100k$, and therefore each other vertex in Z has a probability of at most $1/100k$ to be v's neighbor. And we union bound on this over all $k-1$ neighbors.

An easy computation shows that (3.2) simplifies to $\leq 0.99 \cdot (k-1) \cdot 1/100k + (1 - 0.99) < 0.02$. Now write, for each vertex v in Z, an indicator random variable that is 1 if v has no neighbors in Z. By linearity of expectation, the expectation of the sum of these variables is $\geq 0.98k$, therefore $\mathbf{E}[|\mathsf{MAX\text{-}IS}(G[Z])|] \geq 0.98k$.

To finish the proof of the lemma, we show that $\mathsf{MAX\text{-}IS}(G[Z])$ is a highly concentrated random variable. To see this, define a martingale $X_0, X_1, \ldots, X_k$. X_0 is equal to $\mathbf{E}[\mathsf{MAX\text{-}IS}(G[Z])]$ before any vertex of Z was chosen, thus X_0 is a deterministic value at least $0.98k$. X_1 is equal to $\mathbf{E}[\mathsf{MAX\text{-}IS}(G[Z])]$ conditioning on the first vertex that is chosen. X_2 is the same value, conditioning on the first two vertices chosen, and so on. This martingale is thus the vertex-exposure martingale, where we expose vertices of Z one by one. It is easy to see that $|X_{i+1} - X_i| \leq 1$ for all $0 \leq i < k$. Then by Azuma's inequality (see, e.g., [2, chap. 7.2]), we have that with probability larger than $1 - e^{-k/40000}$ over the choices of Z, $X_k > (0.98 - 0.01)k = 0.97k$. $\square$

We also have the following lemma.

Lemma 3.7. *Let $G = (U, E)$ be a simple graph with the property that at least a 0.01 fraction of vertices have degrees at least $u/100k$. If we pick a subset $Z \subseteq U$ of size k uniformly at random, then the expected size of $N_G(Z)$ is at least $\Omega(u)$.*

Proof. For any $v \in U$ whose degree is at least $u/100k$, the probability that it has a neighbor in Z is at least $1 - (1 - 1/100k)^k \geq 1 - e^{-1/100}$. Therefore we have

$$\mathbf{E}[N_G(Z)] \geq 0.01 \cdot \left(1 - e^{-1/100}\right) u \geq \Omega(u).$$

$\square$

Proof. (of Lemma 3.5). We first define an intersection graph that captures the combinatorial intersection structure of the family. The graph is $G = G_{\mathcal{F}} = (U, E)$, where two vertices $x, y \in U$ are connected by an edge if $S_x \cap S_y \neq \emptyset$. We now see that to prove the LOSI we just need to prove that a random set Z of k vertices will have $\mathbf{E}_Z[|N_G(Z)|] \geq \Omega(u)$, where $N_G(Z)$ is the set of vertices in G that have at least one neighbor in Z. In the following, whenever we say *k-set* we mean "a set of cardinality k".

We show that if for a specific Y condition (3.1) holds, then $\mathsf{MAX\text{-}IS}(G[Y]) \leq 0.96k$. To see this, observe that from (3.1) and from the pigeonhole principle, it follows that there exist y, y' such that S_y and $S_{y'}$ contain a common element from S. Thus y and y' are connected by an edge in the graph. Eliminate these two vertices, and repeat this argument. We can repeat the argument at least $(0.99k - 0.91k)/2 = 0.04k$ times, therefore there is a matching in $G[Y]$ of cardinality $0.04k$, and therefore $\mathsf{MAX\text{-}IS}(G[Y]) \leq 0.96k$.

The condition (3.1) holds for a random k-set Y with probability p, so for a randomly selected k-set Y, it holds that $\Pr_Y[\mathsf{MAX\text{-}IS}(G[Y]) \leq 0.96k] \geq p$. We thus see that many induced subgraphs do not have large independent sets. By applying Lemma 3.6, we get that $|\{x \in U \mid \deg_G(x) \leq u/100k\}| \leq 0.99u$. And then by applying Lemma 3.7, we get exactly the conclusion of the LOSI. $\square$

Now we give the proof of Lemma 3.3.

Proof. (of Lemma 3.3). Recall that Y is a set of the last $|Y| = k$ insertions done, and Y' is a set of random elements, also of cardinality k, chosen from U. Let `BEGIN` be the time before inserting Y, and recall that `END` is the snapshot after inserting Y and that `END`$'$ is the snapshot after inserting Y' instead of Y. Let $M_{\texttt{BEGIN}}, M_{\texttt{END}}$ and $M_{\texttt{END}'}$ be the states of the cache at `BEGIN`, `END` and `END`$'$ respectively. Likewise, let $D_{\texttt{BEGIN}}, D_{\texttt{END}}$ and $D_{\texttt{END}'}$ be the states of the disk at `BEGIN`, `END` and `END`$'$.

Now, the second item of Lemma 3.2 gives us the following:

Claim 1.

With probability at least 0.999 *over the choices of* Y'*, at least a* 0.99 *fraction of* Y' *will have query-paths that intersect* C'^u *under* $(M_{\texttt{END}'}, D_{\texttt{BEGIN}})$.

To prove Lemma 3.3 it is enough to prove that

$$\left| \left\{ x \in U \;\middle|\; q_{\texttt{END}}(x) \bigcap \left(\bigcup_{y \in Y'} q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(y) \bigcup C^u \right) \neq \emptyset \right\} \right| \geq \Omega(u) \; . \tag{3.3}$$

Consider some $x \in U$. If $q_{\texttt{END}}(x)$ intersects C^u, then x is counted in the left-hand side of equation (3.3). Otherwise, $q_{\texttt{END}}(x) = q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(x)$. Now, recall that the set Y' is chosen uniformly at random. Therefore, we just need to show that $\{q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(y)\}_{y \in Y'}$ satisfies the condition of the LOSI with probability $\geq p$. Here $\{q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(x) \mid x \in U\}$ correspond to $\{S_1, \ldots, S_u\}$ in LOSI, C'^u corresponds to S in LOSI (recall that $|C'^u| \leq 0.91k$ by the assumption of Lemma 3.3), $k = |Y'| \geq |Y_1| = m/\gamma = 400000m$, and $p = 2^{-k/40000} \leq 2^{-10m}$. Claim 1 tells us that $\{q_{(M_{\texttt{END}'}, D_{\texttt{BEGIN}})}(y)\}_{y \in Y'}$ satisfies the condition of the LOSI with probability ≥ 0.999. Thus, we just need to estimate the probability that $M_{\texttt{END}'} = M_{\texttt{END}}$. The probability of this is at least $1/2^m$, since $M_{\texttt{END}'}$ and $M_{\texttt{END}}$ are both drawn from the same distribution over a universe of cardinality 2^m. Therefore, the probability that $\{q_{(M_{\texttt{END}}, D_{\texttt{BEGIN}})}(y)\}_{y \in Y'}$ satisfies the condition of the LOSI is $0.999/2^m > 2^{-10m} \geq p$, and we are done. $\square$

3.5. Direct Applications. Our lower bound for membership directly applies for a wide range of other data structure problems, such as predecessor, prefix-sum, range counting, union find, rank-select, etc. Many of these problems can be solved by the buffer tree; thus, our lower bound is also tight. In this section we include definitions of some of these problem and the corresponding reductions, for the purpose of facilitating future research in this area.

Here is a partial list of problems that our lower bound applies.

1. Predecessor: Maintain a set $S \subseteq U$ with $|S| \leq n$ under insertions and deletions. Given an $x \in U$, return $\max\{y \mid y \leq x, y \in S\}$.
2. 1D range counting: Maintain a set $S \subseteq U$ with $|S| \leq n$ under insertions and deletions. Given a range $[a, b]$, return $|\{x \mid a \leq x \leq b, x \in S\}|$.
3. 1D range reporting: Maintain a set $S \subseteq U$ with $|S| \leq n$ under insertions and deletions. Given a range $[a, b]$, return $\{x \mid a \leq x \leq b, x \in S\}$. The cost is often expressed as $f(n) + k/B$ where k is the total number of items reported. We usually only care about the $f(n)$ term.
4. Partial sum: Maintain an array $A = \{a_1, a_1, \ldots, a_n\}$ to support two operations. Update(i, j): set $a_i = j$. Query(i): return $\sum_{k=1}^{i} a_k$.
5. Union-find: Maintain a set of elements U ($|U| = u$) partitioned into a number of disjoint subsets to support two operations. Union(x, y): merge the two subsets containing x and y respectively into a single subset. Find(x): return which subset a particular element x is in.
6. Rank-select: Maintain an ordered set S under insertions and deletions to support another two operations. Rank(x, S): return the number of elements in S which are no greater than x. Select(i, S): return the i-th smallest element in S.

Since membership is a special case of the first four problems, the reductions are obvious.

For the union-find problem, we maintain u initial subsets containing $1, 2, \ldots, u$, respectively, as well as a special subset containing a special symbol, say $*$. When x is inserted into S in the dynamic membership problem, we perform union $(x, *)$. Querying whether $x \in S$ is equivalent to testing whether Find$(x) =$ Find$(*)$. Note that our lower bound for the dynamic membership holds even when there is no deletion.

For the rank-select problem, querying whether $x \in S$ is equivalent to testing whether Select(Rank(x, S),S) $= x$.

4. Conclusions. In this paper we have resolved the complexity of dynamic membership in the external memory model, for the regime where t_u is just below 1. Consider the case $b \geq \log^{1+\epsilon} u$ and $n \geq m^{1+\epsilon}$ for some arbitrary constant $\epsilon > 0$. For this case, our bounds are tight all the way down to $t_u = \log n \Big/ b^{1-\epsilon'}$ for some constant $\epsilon' > 1/(1+\epsilon)$. Namely, the lower bound shows that for such values of b, $t_q \geq \Omega(\log_b n)$ whenever $t_u \leq 0.9$, and the upper bound shows that for such b, we can achieve $t_q = O(\log_b n)$ and $t_u = O\left(\log n \Big/ b^{1-\epsilon'}\right)$ for some constant $\epsilon' > 1/(1+\epsilon)$. For t_u smaller than this, the lower bound does not match the known upper bounds, and better lower bounds might be possible. Indeed, very recently Iacono and Pătraşcu [13] further extended the lower bound curve to smaller t_u. But it is still open when $t_u \in [\log u/b, \log\log n \cdot \log u/b]$. See the discussion in Section 1.3.

There are still a few regimes where our results are not tight:

1. For b which is smaller than $\log^{1+\epsilon} u$ for every $\epsilon > 0$, our lower bound is loose (in fact, for $b = 1$ it seems extremely loose) partly due to inherent limitations of the LOSI, and partly because we did not concentrate on such b. It is interesting whether the bound can be made tighter, and whether a variant of the LOSI suffices or another technique is needed.
2. For $t_u = O(\log u/b)$, we believe that the right behavior of the query time is $t_q \geq \Omega(n^{\Omega(1)})$. It seems very interesting to try to prove this, but it does not seem that the cell probe lower bound community has sufficiently advanced technology to achieve this.

These two questions seem to not only require new ideas, but to be close to the boundaries of current lower bound technology. We are especially interested in the second one.

Acknowledgments. The authors would like to thank Mihai Pătraşcu, Ke Yi and Zeyuan Allen Zhu for helpful discussions. We would like to thank the STOC and SICOMP anonymous reviewers for their insightful reviews.

REFERENCES

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.
[2] N. Alon and J. H. Spencer. The Probabilistic Method (second edition). *Wiley-Interscience series in discrete mathematics and optimization.* John Wiley & Sons, 2004.
[3] L. Arge. External Memory Data Structures. *In Handbook of Massive Data Sets, J. Abello, P.M. Pardalos, M.G.C. Resende (Eds.),* Kluwer Academic Publishers, 2002.
[4] L. Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003.
[5] L. Arge, M. Bender, E. Demaine, B. Holland-Minkley, and J. I. Munro. Cache-oblivious priority-queue and graph algorithms. In *Proc. ACM Symposium on Theory of Computing*, pages 268–276, 2002.
[6] P. Beame and F. E. Fich. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.
[7] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[8] G. S. Brodal and R. Fagerberg. Lower bounds for external memory dictionaries. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 546–554, 2003.
[9] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: upper and lower bounds. *SIAM Journal on Computing*, 23:738–761, 1994.
[10] R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. Heaps and heapsort on secondary storage. *Theoretical Computer Science*, 220(2):345–362, 1999.
[11] R. Fagin, J. Nievergelt, N. Pippenger, and H. Strong. Extendible hashing—a fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.
[12] A. Golynski. Cell probe lower bounds for succinct data structures. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 625–634, 2009.
[13] J. Iacono and M. Pătraşcu. Using Hashing to Solve the Dictionary Problem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 570–582, 2012.
[14] M. S. Jensen and R. Pagh. Optimality in external memory hashing. *Algorithmica*, 52(3):403–411, 2008.
[15] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, Reading MA, second edition, 1998.
[16] P. A. Larson. Performance analysis of linear hashing with partial expansions. In *ACM Transactions on Database Systems*, 7(4):566–587, 1982.
[17] W. Litwin. Linear hashing: a new tool for file and table addressing. In *Proc. International Conference on Very Large Databases*, pages 212–223, 1980.
[18] K. Mehlhorn, S. Näher, and M. Rauch. On the complexity of a game related to the dictionary problem. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 546–548, 1989.
[19] P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. In *Journal of Computer and System Sciences*, 57(1):37–49 1998.
[20] C. W. Mortensen, R. Pagh, and M. Pătraşcu. On dynamic range reporting in one dimension. In *Proc. ACM Symposium on Theory of Computing*, pages 104–111, 2005.
[21] R. Pagh. Basic External Memory Data Structures. In *Algorithms for Memory Hierarchies, U. Meyer, P. Sanders, J. Sibeyn (Eds.).* Lecture Notes in Computer Science, volume 2625, pages 14–35. Springer, 2003.
[22] R. Pagh. On the cell probe complexity of membership and perfect hashing. In *Proc. ACM Symposium on Theory of Computing*, pages 425–432, 2001.
[23] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51:122–144, 2004.

[24] M. Pătraşcu. Personal communication.

[25] M. Pătraşcu and E. D. Demaine. Tight bounds for the partial-sums problem. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 20–29, 2004.

[26] M. Pătraşcu and M. Thorup. Time-space trade-offs for predecessor search. In *Proc. ACM Symposium on Theory of Computing*, pages 232–240, 2006.

[27] M. Pătraşcu and E. Viola. Cell-probe lower bounds for succinct partial sums. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 117–122, 2010.

[28] P. Sen. Lower bounds for predecessor searching in the cell probe model. In *Journal of Computer and System Sciences*, 74(3):364–385 2003.

[29] R. Sundar. A lower bound for the dictionary problem under a hashing model. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 612–621, 1991.

[30] J. S. Vitter. *Algorithms and Data Structures for External Memory.* Now Publishers, 2008.

[31] Z. Wei, K. Yi, and Q. Zhang. Dynamic external hashing: The limit of buffering. In *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, pages 253–259, 2009.

[32] A. C. Yao. Probabilistic computations: Towards a united measure of complexity. *Proc. IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[33] A. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

[34] K. Yi. Personal communication.

[35] K. Yi and Q. Zhang. On the cell probe complexity of dynamic membership. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 123–133, 2010.

[36] Z. A. Zhu. Personal communication.