Optimal Bounds for Open Addressing Without Reordering

Martín Farach-Colton*, Andrew Krapivin†, William Kuszmaul‡

Abstract

In this paper, we revisit one of the simplest problems in data structures: the task of inserting elements into an open-addressed hash table so that elements can later be retrieved with as few probes as possible. We show that, even without reordering elements over time, it is possible to construct a hash table that achieves far better expected search complexities (both amortized and worst-case) than were previously thought possible. Along the way, we disprove the central conjecture left by Yao in his seminal paper *"Uniform Hashing is Optimal"*. All of our results come with matching lower bounds.

1 Introduction

In this paper, we revisit one of the simplest problems in data structures: the task of inserting elements into an open-addressed hash table so that elements can later be retrieved with as few probes as possible. We show that, even without reordering elements over time, it is possible to construct a hash table that achieves far better expected probe complexities (both amortized and worst-case) than were previously thought possible. Along the way, we disprove the central conjecture left by Yao in his seminal paper *"Uniform Hashing is Optimal"* [21].

Background. Consider the following basic problem of constructing an ***open-addressed hash table without reordering***. A sequence $x_1, x_2, \ldots, x_{(1-\delta)n}$ of keys are inserted, one after another, into an array of size n. Each x_i comes with a ***probe sequence*** $h_1(x_i), h_2(x_i), \ldots \in [n]^\infty$ drawn independently from some distribution $\mathcal{P}$. To insert an element x_i, an ***insertion algorithm*** $\mathcal{A}$ must choose some not-yet-occupied position $h_j(x)$ in which to place the element. Note that insertions cannot reorder (i.e., move around) the elements that were inserted in the past, so the only job of an insertion is to select which unoccupied slot to use. The full specification of the hash table is given by the pair $(\mathcal{P}, \mathcal{A})$.

If x_i is placed in position $h_j(x_i)$, then x_i is said to have ***probe complexity*** j. This refers to the fact that a query could find x_i by making j probes to positions $h_1(x), \ldots h_j(x)$. The goal is to design the hash table $(\mathcal{P}, \mathcal{A})$ in a way that minimizes the ***amortized expected probe complexity***, i.e., the expected value of the average probe complexity across all of the keys $x_1, x_2, \ldots, x_{(1-\delta)n}$.

The classic solution to this problem is to use ***uniform probing*** [13]: the probe sequence for each key is a random permutation of $\{1, 2, \ldots, n\}$, and each insertion x_i greedily uses the first unoccupied position from its probe sequence. It is a straightforward calculation to see that random probing has amortized expected probe complexity $\Theta(\log \delta^{-1})$.

*NYU. martin@farach-colton.com
†Cambridge University. andrew@krapivin.net
‡CMU. kuszmaul@cmu.edu

Ullman conjectured in 1972 [19] that the amortized expected probe complexity of $\Theta(\log \delta^{-1})$ should be optimal across all ***greedy*** algorithms, i.e., any algorithm in which each element uses the first unoccupied position in its probe sequence. This conjecture remained open for more than a decade before it was proven by Yao in 1985 [21] in a celebrated paper titled "*Uniform Hashing is Optimal*".

The classical way to get around Yao's lower bound is to consider a relaxation of the problem in which the insertion algorithm is permitted to perform *reordering*, i.e., moving elements around after they're inserted. In this relaxed setting, it is possible to achieve $O(1)$ amortized expected probe complexity even when the hash table is completely full [6, 10, 17]. What is not clear is whether this relaxation is necessary. Could a non-greedy algorithm potentially achieve $o(\log \delta^{-1})$ amortized expected probe complexity, without reordering? Or is reordering fundamentally necessary to achieve small amortized probe complexity?

Question 1. Can an open-addressed hash table achieve amortized expected probe complexity $o(\log \delta^{-1})$ without reordering elements after they are inserted?

A closely related problem is that of minimizing ***worst-case expected probe complexity***. A *worst-case* bound on expected probe complexity must apply to each insertion individually—even to the insertions that are performed when the hash table is very full. Uniform probing achieves a worst-case expected probe complexity of $O(\delta^{-1})$. It has remained an open question, however, whether this bound is asymptotically optimal without the use of reordering.

Question 2. Can an open-addressed hash table achieve worst-case expected probe complexity $o(\delta^{-1})$ without reordering?

This second question, somewhat notoriously [8, 14–16, 19, 21], remains open even for *greedy* open-addressed hash tables. Yao conjectured in 1985 [21] that uniform probing should be nearly optimal in this setting, that is, that any greedy open-addressed hash table must have worst-case expected probe complexity at least $(1 - o(1))\delta^{-1}$. Despite its simplicity, Yao's conjecture has never been settled.

This paper: Tight bounds for open addressing without reordering. In Section 2, we give a single hash table that answers both of the above questions in the affirmative. Specifically, we show how to achieve an amortized bound of $O(1)$ and a worst-case bound of $O(\log \delta^{-1})$ on the expected probe complexity in an open-addressed hash table that does not make use of reordering.

Theorem 1. Let $n \in \mathbb{N}$ and $\delta \in (0, 1)$ be parameters such that $\delta > O(1/n)$ and δ^{-1} is a power of two. It is possible to construct an open-addressing hash table that supports $n - \lfloor \delta n \rfloor$ insertions in an array of size n, that does not reorder items after they are inserted, and that offers amortized expected probe complexity $O(1)$, worst-case expected probe complexity $O(\log \delta^{-1})$, and worst-case expected insertion time $O(\log \delta^{-1})$.

We refer to our insertion strategy as ***elastic hashing***, because of the way that the hash table often probes much further down the probe sequence before snapping back to the position it ends up using. That is, in the process of deciding which slot $h_i(x)$ to put a key x in, the algorithm will first examine many slots $h_j(x)$ satisfying $j > i$. This non-greedy behavior is essential, as it is the only possible way that one could hope to avoid Yao's lower bound [21] without reordering.

Our bound of $O(1)$ on amortized expected probe complexity is, of course, optimal. But what about the bound of $O(\log \delta^{-1})$ on worst-case expected probe complexity? We prove that this bound

is also optimal: any open-addressing hash table that does not use reordering must have worst-case expected probe complexity at least $\Omega(\log \delta^{-1})$.

Next, in Section 3, we turn our attention to *greedy* open-addressed hash tables. Recall that, in this setting, Question 1 has already been resolved – it has been known for decades that uniform probing is asymptotically optimal [21]. Question 2, on the other hand, remains open – this is the setting where Yao conjectured uniform probing to be optimal [8, 14–16, 19, 21]. Our second result is a simple greedy open-addressed strategy, which we call ***funnel hashing***, that achieves $O(\log^2 \delta^{-1})$ worst-case expected probe complexity:

Theorem 2. Let $n \in \mathbb{N}$ and $\delta \in (0, 1)$ be parameters such that $\delta > O(1/n^{o(1)})$. There is a greedy open-addressing strategy that supports $n - \lfloor \delta n \rfloor$ insertions in an array of size n, and that offers worst-case expected probe complexity (and insertion time) $O(\log^2 \delta^{-1})$. Furthermore, the strategy guarantees that, with probability $1 - 1/\operatorname{poly}(n)$, the worst-case probe complexity over all insertions is $O(\log^2 \delta^{-1} + \log \log n)$. Finally, the amortized expected probe complexity is $O(\log \delta^{-1})$.

The bound of $O(\log^2 \delta^{-1}) = o(\delta^{-1})$ on worst-case expected probe complexity is asymptotically smaller than the $\Theta(\delta^{-1})$ bound that Yao conjectured to be optimal. This means that Yao's conjecture is false, and that there is a sense in which uniform probing is *sub-optimal* even among greedy open-addressed strategies.

Despite their somewhat unusual shape, the bounds in Theorem 2 turn out to be optimal. For worst-case expected probe complexity, we prove a matching lower bound of $\Omega(\log^2 \delta^{-1})$ that applies to any greedy open-addressing scheme. For high-probability worst-case probe complexity, we prove a matching lower bound $\Omega(\log^2 \delta^{-1} + \log \log n)$ that applies to any open-addressing algorithm that does not perform reorderings. Perhaps surprisingly, this second lower bound holds even to non-greedy strategies.

The basic structure of funnel hashing, the hash table that we use to prove Theorem 2, is quite simple, and subsequent to the initial version of this paper, the authors have also learned of several other hash tables that make use of the same high-level idea in different settings [7, 9]. Multi-level adaptive hashing [7] uses a similar structure (but only at low load factors) to obtain a hash table with $O(\log \log n)$ levels that supports high parallelism in its queries – this idea was also applied subsequently to the design of contention-resolution schemes [4]. Filter hashing [9] applied the structure to high load factors to get an alternative to d-ary cuckoo hashing that, unlike known analyses of standard d-ary cuckoo hashing, can be implemented with constant-time polynomial hash functions. Filter hashing achieves the same $O(\log^2 \delta^{-1})$-style bound as funnel hashing, and indeed, one way to think about funnel hashing is as a variation of filter hashing that is modified to be an instance of greedy open-addressing and to have optimal worst-case probe complexity, thereby leading to Theorem 2 and disproving Yao's conjecture [21].

Additional problem history and related work. We conclude the introduction by briefly giving some additional discussion of related work and of the history of the problems and models studied in this paper.

The idea of studying amortized expected probe complexity appears to have been first due to Knuth in his 1963 paper on linear probing [11]. Knuth observed that, when a linear-probing hash table is $1 - \delta$ full, then even though the expected insertion time is $O(\delta^{-2})$, the amortized expected probe complexity is $O(\delta^{-1})$. Knuth would later pose a weaker version of Ullman's conjecture [12], namely that uniform probing is optimal out of a restricted set of greedy strategies known as *single-hashing* strategies. This weaker conjecture was subsequently proven by Ajtai [1], whose techniques ended up serving as the eventual basis for Yao's proof of the full conjecture [21]. As noted earlier,

Yao conjectured that it should be possible to obtain a stronger result, namely that the *worst-case* expected probe complexity in any greedy open-addressing hash table is $\Omega(\delta^{-1})$. This conjecture remained open [8, 14–16] until the current paper, which disproves it in Theorem 2.

Although we do not discuss key-value pairs in this paper, most applications of open-addressing associate a value with each key [13]. In these settings, the job of a query is not necessarily to determine whether the key is present (it is often already known to be), but instead to recover the corresponding value. This distinction is important because both probe complexity and amortized probe complexity are notions that apply only to keys that *are present*. Minimizing amortized expected probe complexity, in particular, corresponds to minimizing the expected time to query a *random element out of those present*. There is no similar notion for negative queries—when querying an element not present, there is no interesting difference between querying an arbitrary element versus a random one.

For *worst-case* expected probe complexity, on the other hand, one can in some cases hope to extend one's results to negative queries. For greedy algorithms, in particular, negative query time is the same as insertion time (both stop when they encounter a free slot) [13]. Thus the guarantee in Theorem 2 extends to imply an $O(\log^2 \delta^{-1})$ expected time bound for negative queries.

One can also extend the study of open-addressing without reordering to settings that support both insertions and deletions over an infinite time horizon [2, 3, 18]. In this setting, even very basic schemes such as linear probing [18] and uniform probing [3] have resisted analysis—it is not known whether either scheme achieves expected insertion times, probe complexities, or amortized probe complexities that even bounded as a function of δ^{-1}. It is known, however, that the optimal amortized expected probe complexity in this setting is $\delta^{-\Omega(1)}$ (see Theorem 3 in [3]), meaning that results such as Theorems 1 and 2 are not possible.

2 Elastic Hashing

In this section, we construct elastic hashing, an open-addressed hash table (without reordering) that achieves $O(1)$ amortized expected probe complexity and $O(\log \delta^{-1})$ worst-case expected probe complexity.

Theorem 1. Let $n \in \mathbb{N}$ and $\delta \in (0, 1)$ be parameters such that $\delta > O(1/n)$ and δ^{-1} is a power of two. It is possible to construct an open-addressing hash table that supports $n - \lfloor \delta n \rfloor$ insertions in an array of size n, that does not reorder items after they are inserted, and that offers amortized expected probe complexity $O(1)$, worst-case expected probe complexity $O(\log \delta^{-1})$, and worst-case expected insertion time $O(\log \delta^{-1})$.

Our construction will make use of a specific injection $\phi : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$.

Lemma 1. There exists an injection $\phi : \mathbb{Z}^+ \times \mathbb{Z}^+ \to \mathbb{Z}^+$ such that $\phi(i, j) \leq O(i \cdot j^2)$.

Proof. Take i's binary representation $a_1 \circ a_2 \circ \cdots \circ a_p$ and j's binary representation $b_1 \circ b_2 \circ \cdots \circ b_q$ (here, a_1 and b_1 are the most significant bits of i and j, respectively), and construct $\phi(i, j)$ to have binary representation

$$1 \circ b_1 \circ 1 \circ b_2 \circ 1 \circ b_3 \circ \cdots \circ 1 \circ b_1 \circ 0 \circ a_1 \circ a_2 \circ \ldots \circ a_p,$$

where again the digits read from most significant bit to least significant bit. The map $\phi(i, j)$ is an injection by design since one can straightforwardly recover i and j from $\phi(i, j)$'s binary representation. On the other hand,

$$\log_2 \phi(i, j) \leq \log_2 i + 2 \log_2 j + O(1),$$

which means that $\phi(i,j) \leq O(i \cdot j^2)$, as desired. $\square$

The algorithm. We now describe our insertion algorithm. Break the array A of size n into disjoint arrays $A_1, A_2, \ldots, A_{\lceil \log n \rceil}$ satisfying $|A_{i+1}| = |A_i|/2 \pm 1$.[1]

We will simulate a two-dimensional probe sequence $\{h_{i,j}\}$, where probe $h_{i,j}(x)$ is a random slot in array A_i. In particular, we map the entries of the two-dimensional sequence $\{h_{i,j}\}$ to those of a one-dimensional sequence $\{h_i\}$ by defining

$$h_{\phi(i,j)}(x) := h_{i,j}(x),$$

where ϕ is the map from Lemma 1. This means that the probe complexity of an element x placed in slot $h_{i,j}(x)$ is $O(i \cdot j^2)$.

We break the $n - \lfloor \delta n \rfloor$ insertions into ***batches*** $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \ldots$. Batch $\mathcal{B}_0$ fills array A_1 to have $\lceil 0.75|A_1| \rceil$ elements, where each element x is inserted using the first available slot in the probe sequence $h_{1,1}(x), h_{1,2}(x), h_{1,3}(x), \ldots$. For $i \geq 1$, batch $\mathcal{B}_i$ consists of

$$|A_i| - \lfloor \delta |A_i|/2 \rfloor - \lceil 0.75 \cdot |A_i| \rceil + \lceil 0.75 \cdot |A_{i+1}| \rceil \tag{1}$$

insertions, all of which are placed in arrays A_i and A_{i+1}. (The final batch may not finish, since we run out of insertions.) For $i \geq 0$, the guarantee at the end of the batch $\mathcal{B}_i$ is that each A_j satisfying $j \in \{1, \ldots, i\}$ contains exactly $|A_j| - \lfloor \delta |A_j|/2 \rfloor$ elements, and that A_{i+1} contains exactly $\lceil 0.75 \cdot |A_{i+1}| \rceil$ elements. Note that this guarantee forces the batch size to be given by Eq. (1). Additionally, because the total number of insertions is $n - \lfloor \delta n \rfloor$, and because each batch $\mathcal{B}_i$ leaves at most $O(n/2^i) + \delta n/2$ remaining free slots in the full array A, the insertion sequence is guaranteed to finish within $O(\log \delta^{-1})$ batches.

Let c be a parameter that we will later set to be a large positive constant, and define the function

$$f(\epsilon) = c \cdot \min(\log^2 \epsilon^{-1}, \log \delta^{-1}).$$

We now describe how to implement the insertion of an element x during a batch $\mathcal{B}_i$, $i \geq 1$. Suppose that, when the insertion occurs, A_i is $1 - \epsilon_1$ full and A_{i+1} is $1 - \epsilon_2$ full. There are three cases:

1. If $\epsilon_1 > \delta/2$ and $\epsilon_2 > 0.25$, then x can go in either of A_i or A_{i+1} and is placed as follows: if any of the positions
$$h_{i,1}(x), h_{i,2}(x), \ldots, h_{i,f(\epsilon_1)}(x)$$
are free in A_i, then x is placed in the first such free slot; and, otherwise, x is placed in the first free slot from the sequence of positions
$$h_{i+1,1}(x), h_{i+1,2}(x), h_{i+1,3}(x), \ldots.$$

2. If $\epsilon_1 \leq \delta/2$, then x must be placed in A_{i+1}, and x is placed in the first free slot from the sequence of positions
$$h_{i+1,1}(x), h_{i+1,2}(x), h_{i+1,3}(x), \ldots.$$

3. Finally, if $\epsilon_2 \leq 0.25$, then x must be placed in A_i, and x is placed in the first free slot from the sequence of positions
$$h_{i,1}(x), h_{i,2}(x), h_{i,3}(x), \ldots.$$

[1]The ± 1's are needed so that it is possible to satisfy $|A_1| + |A_2| + \cdots + A_{\lceil \log n \rceil} = n$. Note that, in this context, $a \pm 1$ means one of $a - 1, a, a + 1$.

We refer to the final case as the ***expensive case*** since x is inserted into a potentially very full array A_i using uniform probing. We shall see later, however, that this case is very rare: with probability $1 - O(1/|A_i|^2)$, the case never occurs during batch $\mathcal{B}_i$.

Note that Cases 2 and 3 are disjoint (only one of the two cases can ever occur in a given batch) by virtue of the fact that, once $\epsilon_1 \leq \delta/2$ and $\epsilon_2 \leq 0.25$ hold simultaneously, then the batch is over.

Bypassing the coupon-collector bottleneck. Before we dive into the analysis, it is helpful to understand at a very high level how our algorithm is able to bypass the "coupon-collector" bottleneck faced by uniform probing. In uniform probing, each probe can be viewed as sampling a random coupon (i.e., slot); and standard coupon-collector lower bounds say that at least $\Omega(n \log \delta^{-1})$ probes need to be made if a $(1-\delta)$-fraction of coupons are to be collected. This prohibits uniform probing (or anything like uniform probing) from achieving an amortized expected probe complexity better than $O(\log \delta^{-1})$.

A critical feature of our algorithm is the way in which it decouples each key's *insertion probe complexity* (i.e., the number of probes made while inserting the key) from its *search probe complexity* (i.e., the number of probes needed to find the key). The latter quantity, of course, is what we typically refer to simply as *probe complexity*, but to avoid ambiguity in this section, we will sometimes call it *search probe complexity*.

The insertion algorithm will often probe much further down the probe sequence than the position it ends up using. It might seem unintutive at first glance that such insertion probes could be useful, but as we shall see, they are the key to avoiding the coupon-collecting bottleneck—the result is that most coupons contribute *only* to the insertion probe complexity, and not to the search probe complexity.

To see the decoupling in action, consider an insertion in batch $\mathcal{B}_1$ in which A_1 is, say, a $(1-2\delta^{-1})$-fraction full, and A_2 is, say, a 0.6-fraction full. The insertion makes $\Theta(f(\delta^{-1})) = \Theta(\log \delta^{-1})$ probes to A_1, but most likely they all fail (each probe has only an $O(\delta)$ probability of success). The insertion then looks in A_2 for a free slot, and most likely ends up using a position of the form $h_{\phi(2,j)}$ for some $j = O(1)$, resulting in search probe complexity $O(\phi(2,j)) = O(1)$. So, in this example, even though the insertion probe complexity is $\Theta(\log \delta^{-1})$, the search probe complexity is $O(1)$.

The coupon-collector bottleneck is also what makes *worst-case* expected insertion (and search) bounds difficult to achieve. We know that $\Theta(n \log \delta)$ total coupons must be collected, and intuitively it is the final insertions (i.e., those that take place a high load factors) that must do most of the collecting. After all, how can insertions at low load factors make productive use of more than a few coupons? This is what dooms algorithms such as uniform probing to have a worst-case expected insertion time of $O(\delta^{-1})$.

Our algorithm also circumvents this bottleneck: even though $\Theta(n \log \delta^{-1})$ total coupons are collected, no insertion has an expected contribution of more than $O(\log \delta^{-1})$. This means that even insertions that take place at low load factors need to be capable of 'productively' making use of $\Theta(\log \delta^{-1})$ probes/coupons. How is this possible? The key is that a constant-fraction of insertions x have the following experience: when x is inserted (in some batch $\mathcal{B}_i$), the array A_i is *already* almost full (so x can productively sample $O(\log \delta^{-1})$ probes/coupons in that array), but the next array A_{i+1} is not very full (so x can go there in the likely event that none of the $O(\log \delta^{-1})$ probes/coupons in A_i pay off). This is how the algorithm is able to spread the coupon collecting (almost evenly!) across $\Theta(n)$ operations.

Algorithm Analysis. We begin by analyzing the probability of a given batch containing insertions in the expensive case.

Lemma 2. With probability at least $1 - O(1/|A_i|^2)$, none of the insertions in batch $\mathcal{B}_i$ are in the expensive case (i.e., Case 3).

Proof. Let m denote the size of array A_i. We may assume that $m = \omega(1)$, since otherwise the lemma is trivial. For $j \in \{2, 3, \ldots, \lceil \log \delta^{-1} \rceil\}$, let T_j be the time window during batch $\mathcal{B}_i$ in which A_i goes from having $\lfloor m/2^j \rfloor$ free slots to $\max(\lfloor m/2^{j+1} \rfloor, \lfloor \delta m/2 \rfloor)$ free slots.

Each insertion during T_j is guaranteed to be in one of Cases 1 or 3, so the insertion makes at least $f(2^{-j})$ probe attempts in A_i, each of which has at least $2^{-(j+1)}$ probability of succeeding. If $f(2^{-j}) > 100 \cdot 2^j$, then each insertion during time window T_j has probability at least $1 - (1 - 1/2^{j+1})^{100 \cdot 2^j} > 0.99$ of being placed in array A_i. Otherwise, if $f(2^{-j}) < 100 \cdot 2^j$, then the insertion has a $\Theta(f(2^{-j})/2^j)$ probability of being placed in array A_i. Thus, in general, each insertion in T_j uses array A_i with probability at least

$$\min(0.99, \Theta(f(2^{-j})/2^j)). \tag{2}$$

It follows that

$$\mathbb{E}[|T_j|] \leq \frac{m/2^{j+1}}{\min(0.99, \Theta(f(2^{-j})/2^j))} + O(1) \leq \Theta\left(\frac{m}{f(2^{-j})}\right) + 1.02 \cdot m/2^{j+1} + O(1).$$

Since Eq. (2) holds for each insertion in T_i independently of how the previous insertions behave, we can apply a Chernoff bound to conclude that, with probability at least $1 - 1/m^3$,

$$|T_j| \leq \mathbb{E}[|T_j|] + O(\sqrt{m \log m}) \leq O\left(\frac{m}{f(2^{-j})}\right) + 1.02 \cdot m/2^{j+1} + O(\sqrt{m \log m}). \tag{3}$$

With probability at least $1 - O(1/m^2)$, Eq. (3) holds for every window T_j.

Thus, treating c as a parameter (rather than a constant), we have that

$$\begin{aligned}
\sum_j |T_j| &\leq \sum_{j=2}^{\lceil \log \delta^{-1} \rceil} \left(O\left(\frac{m}{f(2^{-j})}\right) + 1.02 \cdot m/2^{j+1} + O(\sqrt{m \log m}) \right) \\
&\leq 0.26 \cdot m + m \cdot O\left(\sum_{j=2}^{\lceil \log \delta^{-1} \rceil} \frac{1}{f(2^{-j})} \right) + O(1) \\
&= 0.26 \cdot m + m \cdot O\left(\sum_{j=2}^{\lceil \log \delta^{-1} \rceil} \frac{1}{c \min(j^2, \log \delta^{-1})} \right) + O(1) \\
&\leq 0.26 \cdot m + \frac{m}{c} \cdot O\left(\sum_{j=2}^{\infty} \frac{1}{j^2} + \sum_{j=2}^{\lceil \log \delta^{-1} \rceil} \frac{1}{\log \delta^{-1}} \right) + O(1) \\
&\leq 0.26 \cdot m + \frac{m}{c} \cdot O(1) + O(1).
\end{aligned}$$

If we set c to be a sufficiently large positive constant, then it follows that $\sum_j |T_j| < 0.27 \cdot m + O(1)$. However, the first $0.27 \cdot m$ insertions during batch $\mathcal{B}_i$ can fill array A_{i+2} to at most a $0.54 + o(1) < 0.75$ fraction full (recall, in particular, that we have $m = \omega(1)$ without loss of generality). This means that none of the insertions during time windows $T_1, T_2, \ldots$ are in Case 3 (the expensive case). On the other hand, after time windows $T_1, T_2, \ldots$ are complete, the remaining insertions in the batch are all in Case 2. Thus, with probability at least $1 - O(1/m^2)$, none of the insertions in the batch are in Case 3. □

Next, we bound the expected search probe complexity for a given insertion within a given batch.

Lemma 3. The expected search probe complexity for an insertion in batch $\mathcal{B}_i$ is $O(1+i)$.

Proof. Insertions in batch 0 have search probe complexities of the form $\phi(1,j) = O(j^2)$ where j is a geometric random variable with mean $O(1)$. They therefore have expected probe complexities of $O(1)$. For the rest of the proof, let us assume that $i > 0$.

Let x be the element being inserted. Let C_j, $j \in \{1,2,3\}$, be the indicator random variable for the event that x's insertion is in Case j. Let D_k, $k \in \{1,2\}$ be the indicator random variable for the event that x ends up in array A_{i+k-1}. Finally, let Q be the search probe complexity of x. We can break $\mathbb{E}[Q]$ into

$$\begin{aligned}\mathbb{E}[Q] &= \mathbb{E}[QC_1D_1] + \mathbb{E}[QC_1D_2] + \mathbb{E}[QC_2] + \mathbb{E}[QC_3] &(4)\\ &\le \mathbb{E}[QC_1D_1] + \mathbb{E}[QD_2] + \mathbb{E}[QC_3], &(5)\end{aligned}$$

where the final inequality uses the fact that C_2 implies D_2 and thus that $\mathbb{E}[QC_1D_2] + \mathbb{E}[QC_2] \le \mathbb{E}[QD_2]$.

To bound $\mathbb{E}[QC_1D_1]$, observe that

$$\begin{aligned}\mathbb{E}[QC_1D_1] &\le \mathbb{E}[QD_1 \mid C_1] &(6)\\ &= \mathbb{E}[Q \mid D_1, C_1] \cdot \Pr[D_1 \mid C_1] &(7)\end{aligned}$$

Suppose that, when x is inserted, array A_i is $1-\epsilon$ full and that x uses Case 1. Then the only positions that x considers in A_i are $h_{i,1}, \ldots, h_{i,f(\epsilon^{-1})}$. The probability that any of these positions are free is at most $O(f(\epsilon^{-1}) \cdot \epsilon)$. And, if one is free, then the resulting search probe complexity Q will be at most $\phi(i, f(\epsilon^{-1}) \le O(if(\epsilon^{-1})^2)$. Thus Eq. (7) satisfies

$$\begin{aligned}&\mathbb{E}[Q \mid D_1, C_1] \cdot \Pr[D_1 \mid C_1]\\ &\le O(if(\epsilon^{-1})^2) \cdot O(f(\epsilon^{-1})\epsilon)\\ &\le O(i\epsilon f(\epsilon^{-1})^3)\\ &\le O(i\epsilon \log^6 \epsilon^{-1})\\ &\le O(i).\end{aligned}$$

To bound $\mathbb{E}[QD_2]$, recall that D_2 can only occur if A_{i+1} is at most a 0.75 fraction full. Thus, if D_2 occurs, then x will have search probe complexity $\phi(i+1, j)$ where j is at most a geometric random variable with mean $O(1)$. We can therefore bound $\mathbb{E}[QD_2]$ by

$$\mathbb{E}[QD_2] \le \mathbb{E}[Q \mid D_2] \le \mathbb{E}[\phi(i+1, j)],$$

where j is a geometric random variable with mean $O(1)$. This, in turn, is at most

$$O(\mathbb{E}[i \cdot j^2]) = O(i).$$

Finally, to bound $\mathbb{E}[QC_3]$, observe that

$$\mathbb{E}[QC_3] = \mathbb{E}[Q \mid C_3] \cdot \Pr[C_3]. \tag{8}$$

By Lemma 2, we have $\Pr[C_3] = O(1/|A_i|^2)$. Since Case 3 inserts x into A_i using the probe sequence $h_{\phi(i,1)}, h_{\phi(i,2)}, \ldots$, the search probe complexity of x will end up being given by $\phi(i,j) = O(i \cdot j^2)$ where

j is a geometric random variable with mean $O(|A_i|)$. This implies a bound on $\mathbb{E}[Q \mid C_3]$ of $O(i \cdot |A_i|^2)$. Thus, we can bound Eq. (8) by

$$\mathbb{E}[Q \mid C_3] \cdot \Pr[C_3] \leq O(i \cdot |A_i|^2/|A_i|^2) = O(i).$$

Having bounded each of the terms in Eq. (5) by $O(i)$, we can conclude that $\mathbb{E}[Q] = O(i)$, as desired. □

Finally, we bound the worst-case expected insertion time by $O(\log \delta^{-1})$.

Lemma 4. The worst-case expected time for an insertion is $O(\log \delta^{-1})$.

Proof. Insertions in batch 0 take expected time $O(1)$, since they make $O(1)$ expected probes into A_1. Now consider an insertion in some batch $\mathcal{B}_i$, $i \geq 1$. If the insertion is in either of Cases 1 or 2, then the insertion makes at most $f(\delta^{-1})$ probes in A_i and at most $O(1)$ expected probes in A_{i+1}. The expected insertion time in each of these cases is therefore at most $f(\delta^{-1}) = O(\log \delta^{-1})$. Finally, each insertion has probability at most $1/|A_i|^2$ of being in Case 3 (by Lemma 2), and the expected insertion time in Case 3 can be bounded by $O(|A_i|)$ (since we probe repeatedly in A_i to find a free slot). Therefore, the contribution of Case 3 to the expected insertion time is at most $O(1/|A_i|) = O(1)$. □

Putting the pieces together, we prove Theorem 1

Proof. By Lemmas 3, the insertions in $\mathcal{B}_i$ each have expected search probe complexity $O(i)$. Since there are $O(\log \delta^{-1})$ batches, this implies a worst-case expected search probe complexity of $O(\log \delta^{-1})$. And, since the $|\mathcal{B}_i|$s are geometrically decreasing, the amortized expected search probe complexity overall is $O(1)$. Finally, by Lemma 4, the worst-case expected time per insertion is $O(\log \delta^{-1})$. This completes the proof of the theorem. □

3 Funnel Hashing

In this section, we construct a *greedy* open-addressing scheme that achieves $O(\log^2 \delta)$ worst-case expected probe complexity, and high-probability worst-case probe complexity $O(\log^2 \delta + \log \log n)$. As we shall see, the high-probability worst-case bound is optimal.

Theorem 2. Let $n \in \mathbb{N}$ and $\delta \in (0, 1)$ be parameters such that $\delta > O(1/n^{o(1)})$. There is a greedy open-addressing strategy that supports $n - \lfloor \delta n \rfloor$ insertions in an array of size n, and that offers worst-case expected probe complexity (and insertion time) $O(\log^2 \delta^{-1})$. Furthermore, the strategy guarantees that, with probability $1 - 1/\operatorname{poly}(n)$, the worst-case probe complexity over all insertions is $O(\log^2 \delta^{-1} + \log \log n)$. Finally, the amortized expected probe complexity is $O(\log \delta^{-1})$.

Proof. Throughout the section, we assume without loss of generality that $\delta \leq 1/8$. Let $\alpha = \lceil 4 \log \delta^{-1} + 10 \rceil$ and $\beta = \lceil 2 \log \delta^{-1} \rceil$.

The greedy open-addressing strategy that we will use in this section is as follows. First, we split array A into two arrays, A' and a ***special array*** denoted $A_{\alpha+1}$, where $\lfloor 3\delta n/4 \rfloor \geq |A_{\alpha+1}| \geq \lceil \delta n/2 \rceil$, with the exact size chosen so that $|A'|$ is divisible by β. Then, split A' into α arrays $A_1, \ldots, A_\alpha$ such that $|A_i| = \beta a_i$, satisfying $a_{i+1} = 3a_i/4 \pm 1$. That is, the size of each array is a multiple of β and they are (roughly) geometrically decreasing in size. Note that, for $i \in [\alpha - 10]$,

$$\sum_{j>i} |A_j| \geq ((3/4) + (3/4)^2 + \cdots + (3/4)^{10}) \cdot |A_i| > 2.5|A_i|.$$

Each array A_i with $i \in [\alpha]$ is further subdivided into arrays $A_{i,j}$, each of size β. We define an ***attempted insertion*** of a key k into A_i (for $i \in [\alpha]$) as follows:

1. Hash k to obtain a subarray index $j \in \left[\frac{|A_i|}{\beta}\right]$.
2. Check each slot in $A_{i,j}$ to see if any are empty.
3. If there is an empty slot, insert into the first one seen, and return success. Otherwise, return fail.

To insert a key k into the overall data structure, we perform attempted insertions on each of $A_1, A_2, \ldots, A_\alpha$, one after another, stopping upon a successful attempt. Each of these $\alpha = O(\log \delta^{-1})$ attempts probes up to $\beta = O(\log \delta^{-1})$ slots. If none of the attempts succeed, then we insert k into the special array $A_{\alpha+1}$. The special array $A_{\alpha+1}$ will follow a different procedure than the one described above—assuming it is at a load factor of at most $1/4$, it will ensure $O(1)$ expected probe complexity and $O(\log \log n)$ worst-case probe complexity. Before we present the implementation of $A_{\alpha+1}$, we will first analyze the behaviors of $A_1, A_2, \ldots, A_\alpha$.

At a high level, we want to show that each A_i fills up to be almost full over the course of the insertions. Critically, A_i does not need to give any guarantees on the probability of any specific insertion succeeding. All we want is that, after the insertions are complete, A_i has fewer than, say, $\delta|A_i|/64$ free slots.

Lemma 5. For a given $i \in [\alpha]$, we have with probability $1 - n^{-\omega(1)}$ that, after $2|A_i|$ insertion attempts have been made in A_i, fewer than $\delta|A_i|/64$ slots in A_i remain unfilled.

Proof. Since each insertion attempt selects a uniformly random $A_{i,j}$ to use, out of $|A_i|/\beta$ options, the expected number of times that a given $A_{i,j}$ is used is 2β. Letting the number of attempts made to insert into $A_{i,j}$ be $X_{i,j}$, we have by a Chernoff bound that

$$\Pr[X_{i,j} < \beta] = \Pr[X_{i,j} < (1 - 1/2)\mathbb{E}[X_{i,j}]] = e^{-2^2\beta/2} \leq e^{-4\log\delta^{-1}} = \delta^4 \leq \frac{1}{128}\delta.$$

Note that, since we always insert into the subarray we choose if it has empty slots, the only scenario in which $A_{i,j}$ remains unfilled is if $X_{i,j} < \beta$. Therefore, the expected number of subarrays that remain unfilled is at most $\frac{1}{128}\delta\left(\frac{|A_i|}{\beta}\right)$, and, consequently, the expected number of subarrays that become full is $\left(1 - \frac{1}{128}\delta\right)\left(\frac{|A_i|}{\beta}\right)$.

Define $Y_{i,k}$ to be the random number in $[|A_i|/\beta]$ such that the kth insertion attempt into A_i uses subarray $A_{i,Y_{i,k}}$. Let $f(Y_{i,1}, \ldots, Y_{i,2|A_i|})$ denote how many subarrays $A_{i,j}$ remain unfilled after $2|A_i|$ insertion attempts have been made. Changing the outcome of a single $Y_{i,k}$ changes f by at most 2—one subarray may become unfilled and one may become filled. Also, by the above, $\mathbb{E}[f(Y_{i,1}, \ldots, Y_{i,2|A_i|})] = \frac{1}{128}\delta\left(\frac{|A_i|}{\beta}\right)$. Therefore, by McDiarmid's inequality,

$$\Pr\left[f(Y_{i,1}, \ldots, Y_{i,2|A_i|}) \geq \frac{1}{64}\delta\left(\frac{|A_i|}{\beta}\right)\right] \leq \exp\left(-\frac{2\left(\frac{1}{128}\delta\frac{|A_i|}{\beta}\right)^2}{2|A_i|}\right) = \exp\left(-|A_i|O(\beta^2\delta^2)\right).$$

Since $|A_i| = n\,\mathrm{poly}(\delta)$ and $\delta = n^{o(1)}$, we have that

$$|A_i|O(\beta^2\delta^2) = n^{1-o(1)},$$

so the probability that more than a $\frac{\delta}{64}$-fraction of the subarrays in A_i remain unfilled is $\exp(-n^{1-o(1)}) = 1/n^{-\omega(1)}$. All subarrays are the same size, so, even if these unfilled subarrays remain completely empty, we still have that $\frac{1}{64}\delta$ of the total slots remain unfilled, as desired.

□

As a corollary, we can obtain the following statement about $A_{\alpha+1}$:

Lemma 6. With probability $1 - n^{-\omega(1)}$, the number of keys inserted into $A_{\alpha+1}$ is fewer than $\frac{\delta}{8}n$.

Proof. Call A_i *fully explored* if at least $2|A_i|$ insertion attempts are made to A_i. By Lemma 5, we have with probability $1 - n^{-\omega(1)}$ that every fully-explored A_i is at least $(1 - \delta/64)$ full. We will condition on this property for the rest of the lemma.

Let $\lambda \in [\alpha]$ be the largest index such that A_λ receives fewer than $2|A_\lambda|$ insertion attempts (or $\lambda = \texttt{null}$ if no such index exists). We will handle three cases for λ.

First, suppose that $\lambda \leq \alpha - 10$. By definition, we know that, for all $\lambda < i \in [\alpha]$, A_i is fully explored, and therefore that A_i contains at least $|A_i|(1 - \delta/64)$ keys. The total number of keys in A_i, $i > \lambda$, is therefore at least

$$(1 - \delta/64) \sum_{i=\lambda+1}^{\alpha} |A_i| \geq 2.5(1 - \delta/64)|A_\lambda|,$$

contradicting the fact that at most $2|A_\lambda|$ insertions are made in total for all arrays A_i with $i \geq \lambda$ (recall by the construction of our algorithm that we must first try [and fail] to insert into A_λ before inserting into A_i for any $i \geq \lambda$). This case is thus impossible, and we are done.

Next, suppose that $\alpha - 10 < \lambda \leq \alpha$. In this case, fewer than $2|A_{\alpha-10}| < n\delta/8$ keys are attempted to be inserted into any A_i with $i \geq \lambda$, including $i = \alpha + 1$, and we are done.

Finally, suppose that $\lambda = \texttt{null}$. In this case, each A_i, $i \in [\alpha]$, has at most $\delta|A_i|/64$ empty slots. Therefore, the total number of empty slots at the end of all insertions is at most

$$|A_{\alpha+1}| + \sum_{i=1}^{\alpha} \frac{\delta|A_i|}{64} = |A_{\alpha+1}| + \frac{\delta|A'|}{64} \leq \frac{3n\delta}{4} + \frac{n\delta}{64} < n\delta,$$

which contradicts the fact that, after $n(1 - \delta)$ insertions, there are at least $n\delta$ slots empty. This concludes the proof. □

Now, the only part left is to implement the $\leq \delta n/8$ insertions that reach $A_{\alpha+1}$. We must do so with $O(1)$ expected probe complexity and with $O(\log \log n)$ worst-case probe complexity, while incurring at most a $1/\operatorname{poly}(n)$ probability of hash-table failure.

We implement $A_{\alpha+1}$ in two parts. That is, split $A_{\alpha+1}$ into two subarrays, B and C, of equal (± 1) size. To insert, we first try to insert into B, and, upon failure, we insert into C (an insertion into C is guaranteed to succeed with high probability). B is implemented as a uniform probing table, and we give up searching through B after $\log \log n$ attempts. C is implemented as a two-choice table with buckets of size $2 \log \log n$.

Since B has size $|A_{\alpha+1}|/2 \geq \delta n/4$, its load factor never exceeds $1/2$. Each insertion into $A_{\alpha+1}$ makes $\log \log n$ random probes in B, each of which has at least a $1/2$ probability of succeeding. The expected number of probes that a given insertion makes in B is therefore $O(1)$, and the probability that a given insertion tries to use B but fails (therefore moving on to C) is at most $1/2^{\log \log n} \leq 1/\log n$.

On the other hand, C is implemented as a two choice table with buckets of size $2 \log \log n$. Each insertion hashes to two buckets a and b uniformly at random, and uses a probe sequence in which it tries the first slot of a, the first slot of b, the second slot of a, the second slot of b, and so on. The effect of this is that the insertion ends up using the emptier of the two buckets (with ties broken towards a). If both buckets are full, our table fails. However, with high probability, this does not happen, by the following classical power-of-two-choices result [5]:

Theorem 3. If m balls are placed into n bins by choosing two bins uniformly at random for each ball and placing the ball into the emptier of the two bins, then the maximum load of any bin is $m/n + \log \log n + O(1)$ with high probability in n.

Applying this theorem to our setting, we can conclude that, with high probability in $|A_{\alpha+1}|/\log \log n$, and therefore with high probability in n, no bucket in C ever overflows. This ensures the correctness of our implementation of $A_{\alpha+1}$.

Since each insertion in $A_{\alpha+1}$ uses C with probability at most $1/\log n$, and there are at most $2 \log \log n$ slots checked in C, the expected time that each insertion spends in C is at most $o(1)$. Thus, insertions that reach $A_{\alpha+1}$ take expected time $O(1)$ and worst-case time $O(\log \log n)$.

Since we only attempt to insert into β slots for each A_i (a single bucket), the probe complexity of a given insertion is at most $\beta\alpha + f(A_{\alpha+1}) = O(\log^2 \delta^{-1} + f(A_{\alpha+1}))$, where $f(A_{\alpha+1})$ is the number of probes made in $A_{\alpha+1}$. This implies a worst-case expected probe complexity of $O(\log^2 \delta^{-1})$ and a high-probability worst-case probe complexity of $O(\log^2 \delta^{-1} + \log \log n)$.

We now only have left to prove the amortized expected probe complexity. The expected number of probes we make into each subarray is at most $c \log \delta^{-1}$ for some constant c (including for $A_{\alpha+1}$), and we first insert into A_1, then A_2, and so on. Thus, the total expected probe complexity across all keys is at most

$$|A_1| \cdot c \log \delta^{-1} + |A_2| \cdot 2c \log \delta^{-1} + |A_3| \cdot 3c \log \delta^{-1} + \cdots + |A_{\alpha+1}| \cdot (\alpha + 1) \log \delta^{-1}.$$

Since the A_i's are geometrically decreasing in size with the exception of $A_{\alpha+1}$, which itself is only $O(n\delta)$ in size, the above sum is dominated (up to a constant factor) by its first term. The total expected probe complexity across all keys is thus $O(|A_1| \log \delta^{-1}) = O(n \log \delta^{-1})$, implying that the amortized expected probe complexity is $O(n \log \delta^{-1}/(n(1-\delta))) = O(\log \delta^{-1})$, as desired. This completes the proof of Theorem 2. □

4 A Lower Bound for Greedy Algorithms

In this section we prove that the $O(\log^2 \delta^{-1})$ expected-cost bound from Theorem 2 is optimal across all greedy open-addressed hash tables.

Theorem 4. Let $n \in \mathbb{N}$ and $\delta \in (0, 1)$ be parameters, where δ is an inverse power of two. Consider a greedy open-addressed hash table with capacity n. If $(1-\delta)n$ elements are inserted into the hash table, then the final insertion must take expected time $\Omega(\log^2 \delta^{-1})$.

Our proof of Theorem 4 will make use of Yao's lower bound on *amortized* insertion time [21]:

Proposition 7 (Yao's Theorem [21]). Let $n \in \mathbb{N}$ and $\delta \in (0, 1)$ be parameters. Consider a greedy open-addressed hash table with capacity n. If $(1-\delta)n$ elements are inserted into the hash table, then the amortized expected time per insertion must be $\Omega(\log \delta^{-1})$.

Building on Proposition 7, we can obtain the following critical lemma:

Lemma 8. There exists a universal positive constant $c > 0$ such that the following is true: For any values of δ, n, there exists some integer $1 \le i \le \log \delta^{-1}$ such that the $(1 - 1/2^i)n$-th insertion has expected cost at least $ci \log \delta^{-1}$.

Proof. Let c be a sufficiently small positive constant, and suppose for contradiction that the lemma does not hold. Let q_j be the expected cost of the j-th insertion. Because the hash table uses greedy open addressing, we know that the q_js are monotonically increasing. It follows that

$$\mathbb{E}[\sum q_j] \ge \sum_{i \in [1, \log \delta^{-1}]} \frac{n}{2^i} \cdot q_{(1-1/2^i)n},$$

which by assumption is at most

$$\sum_{i \in [1, \log \delta^{-1}]} \frac{n}{2^i} \cdot c \cdot i \log \delta^{-1} \le cn \cdot O(\log \delta^{-1}).$$

Setting c to be a sufficiently small positive constant contradicts Proposition 7. □

Lemma 9. Let c be the positive constant from Lemma 8. Then, the final insertion takes expected time at least

$$\sum_{j=1}^{\log \delta^{-1}} cj.$$

Proof. By Lemma 8, there exists an integer $1 \le i \le \log \delta^{-1}$ such that the $(1 - 1/2^i)n$-th insertion has expected cost at least $ci \log \delta^{-1}$. If $i = \log \delta^{-1}$, then we are done. Otherwise, we can complete the proof by strong induction on n, as follows.

Let S denote the set of occupied positions after the $(1 - 1/2^i)n$-th insertion. Condition on some outcome for S, and define the ***second-layer cost*** for future insertions to be the expected number of probes that the insertion makes to slots $[n] \setminus S$. To analyze the second-layer costs, we can imagine that the slots $[n] \setminus S$ are the only slots in the hash table, and that the slots in S are removed from each element's probe sequence. This new "compressed" hash table has size $n/2^i$ and is set to receive $n/2^i - \delta n = (n/2^i) \cdot (1 - \delta 2^i)$ insertions that are each implemented with greedy open addressing. It follows by induction that the final insertion in the "compressed" hash table has expected cost at least

$$\sum_{j=1}^{\log \delta^{-1} - i} cj. \tag{9}$$

This is equivalent to saying that the final insertion in the full hash table has expected *second-layer cost* at least Eq. (9). Moreover, although Eq. (9) was established conditioned on some specific outcome for S, since it holds for each individual outcome, it also holds without any conditioning at all.

Finally, in addition to the second-layer cost, the final insertion must also perform at least $ci \log n$ expected probes even just to find any slots that are not in S. (This is due to our earlier application of Lemma 8.) It follows that the total expected cost incurred by the final insertion is at least

$$ci \log \delta^{-1} + \sum_{j=1}^{\log \delta^{-1} - i} cj \ge \sum_{j=1}^{\log \delta^{-1}} cj,$$

as desired. □

Finally, we can prove Theorem 4 as a corollary of Lemma 9.

Proof of Theorem 4. By Lemma 9, there exists a positive constant c such that the expected cost incurred by the final insertion is at least

$$\sum_{j=1}^{\log \delta^{-1}} cj = \Omega(\log^2 \delta^{-1}).$$

□

5 Lower Bounds For Open Addressing Without Reordering

In this section, we give two lower bounds that apply not just to greedy open-addressed hash tables but to any open-addressed hash table that does not perform reordering. Our first result is a lower bound of $\Omega(\log \delta^{-1})$ on worst-case expected probe complexity (matching the upper bound from Theorem 1). Our second result is a lower bound of $\Omega(\log^2 \delta^{-1} + \log \log n)$ on (high-probability) worst-case probe complexity (matching the upper bound from Theorem 2, which is achieved by a greedy scheme).

For the following proofs, we assume that the probe sequences for keys are iid random variables. This is equivalent to assuming that the universe size is a large polynomial and then sampling the keys at random (with replacement); with high probability, such a sampling procedure will not sample any key twice.

5.1 Common Definitions

Both lower bounds will make use of a shared set of definitions:

- Let $m = n(1 - \delta)$.
- Let $k_1, k_2, \ldots, k_m$ be the set of keys to be inserted.
- Let $H_i(k_j)$ be i-th entry in the probe sequence for k_j. Because the distribution of $H_i(k_j)$ is the same for all k_j, we will sometimes use H_i as a shorthand. We will also use h_i to refer to a (non-random) specific outcome for H_i.
- Let $\mathcal{H}_c(k_j) = \{H_i(k_j) : i \in [c]\}$ denote the set consisting of the first c probes made by k_j. Again, because $\mathcal{H}_c(k_j)$ has the same distribution for all k_j, we will sometimes use $\mathcal{H}_c$ as a shorthand.
- For $i \in [m]$, let $S_i \subset [n], |S_i| = n - i$ be a random variable denoting the set of unfilled slots in the array after i keys have been inserted (with the distributed derived from the hashing scheme).
- For $i \in [m]$ and $j \in \mathbb{N}$, let $X_{i,j}$ be a random variable indicating whether the slot indexed by $H_j(k_i)$ is empty at the time that k_i is inserted.
- Let Y_i be the position in the probe sequence that k_i uses. In other words, k_i is placed in the slot $H_{Y_i}(k_i)$. We will also use y_i to refer to a (non-random) specific outcome for Y_i. Note that the slot must be empty, so $Y_i \in \{r : X_{i,r} = 1\}$ (in a greedy algorithm, the first available slot is taken—in this case, $Y_i = \min\{r : X_{i,r} = 1\}$—but we make no assumptions as to whether the algorithm is greedy or not).
- Let L_i be the random variable denoting the location in the array in which the ith key is inserted.

5.2 Worst Case Expected Probe Complexity

In this section, we prove the following theorem:

Theorem 5. In any open-addressing scheme achieving load factor $1-\delta$ without reordering, the worst case expected probe complexity must be $\Omega(\log \delta^{-1})$. In particular, there exists some $i \in [m]$ for which $\mathbb{E}[Y_i] = \Omega(\log \delta^{-1})$.

At a high level, we want to reverse the idea of the upper bound algorithms. Rather than partitioning our array into subarrays with exponentially decreasing size, we show that, at minimum, such a construction arises naturally. Given an upper bound c on worst-case expected probe complexity, we will show that there must exist disjoint groups of slots $v_1, v_2, \ldots, v_{\Theta(\log \delta^{-1})}$, of exponentially decreasing sizes, with the property that, for each i, $\mathbb{E}[\mathcal{H}_{2c} \cap v_i] \geq \Omega(1)$. This, in turn, implies that $2c \geq \mathbb{E}[|\mathcal{H}_{2c}|] \geq \Omega(\log \delta^{-1})$. As we shall see, the tricky part is defining the v_is in a way that guarantees this property.

Proof. Let c be any upper bound for $E[Y_i]$ that holds for all i. We want to prove that $c = \Omega(\log \delta^{-1})$. Note that, by Markov's inequality, $\Pr[Y_i \leq 2c] \geq \frac{1}{2}$ for any $i \in [m]$.

Let $\alpha = \left\lfloor \frac{\log \delta^{-1}}{3} \right\rfloor \in \Omega(\log \delta^{-1})$. For $i \in [\alpha]$, let $a_i = n\left(1 - \frac{1}{2^{3i}}\right)$. Note that $a_i \leq a_{\log \delta^{-1}/3} \leq n\left(1 - \frac{1}{2^{3 \log \delta^{-1}/3}}\right) = n(1-\delta) = m$. Further note that $|S_{a_i}| = n - a_i = \frac{n}{2^{3i}}$, since S_i represents the slots still unfilled; and note that the sizes $|S_{a_i}|$, for $i = 1, 2, \ldots$ form a geometric sequence with ratio $1/2^3 = 1/8$. It follows that, for any $s_{a_i} \leftarrow S_{a_i}$, $s_{a_{i+1}} \leftarrow S_{a_{i+1}}, \ldots, s_{a_\alpha} \leftarrow S_{a_\alpha}$, even if the s_{a_i}'s are *not compatible with each other* (i.e., even if $s_{a_{i+1}} \not\subseteq s_{a_i}$), we have

$$\left|s_{a_{i+1}} \cup s_{a_{i+2}} \cup \cdots \cup s_{a_\alpha}\right| \leq \sum_{j \geq i+1} |s_{a_j}| \leq |s_{a_i}|/7.$$

Since $\Pr[Y_i \leq 2c] \geq \frac{1}{2}$, we have that, for any $t < 2m - n = n(1-2\delta)$,

$$\mathbb{E}[|\{i : Y_i \leq 2c \text{ and } t < i \leq m\}|] \geq \frac{m-t}{2} \geq \frac{n-t}{4} = \frac{|S_t|}{4}.$$

Therefore, for each $j \in [\alpha - 1]$, there is some $s_{a_j} \subseteq [n]$ such that

$$\mathbb{E}\left[|\{i : Y_i \leq 2c \text{ and } a_j < i \leq m\}| \,\middle|\, S_{a_j} = s_{a_j}\right] \geq \frac{|s_{a_j}|}{4}. \tag{10}$$

That is, we find some concrete instance s_{a_j} of the random variable S_{a_j} that sets the number of expected "small" values of Y_i, with $i > a_j$ and given $S_{a_j} = s_{a_j}$, to at least the overall expected number. It is important to note that the $s_{a_1}, s_{a_2}, \ldots$ may have an arbitrary relationship to one another; they need not be mutually compatible as values for $S_{a_1}, S_{a_2}, \ldots$. Perhaps surprisingly, even despite this, we will still be able to reason about the relationships between the s_{a_i}s. In particular, we will show by the end of the proof that, for each j and for each insertion, the expected number of probes out of the first $2c$ that probe a position in $s_{a_j} \setminus \bigcup_{k>j} s_{a_k}$ is $\Omega(1)$. This will allow us to deduce that, in expectation, the first $2c$ entries the probe sequence contain at least $\Omega(\log \delta^{-1})$ distinct values, implying that $c = \Omega(\log \delta^{-1})$.

Let

$$\mathcal{L}_j = \{L_i : m \geq i > a_j \text{ and } Y_i \leq 2c\} \,\middle|\, S_{a_j} = s_{a_j}$$

be the (random variable) set of positions that are used by the "fast" insertions (i.e., those satisfying $Y_i \le 2c$) that take place at times $i > a_j$, given that the set of set of unfilled slots (at time a_j) is s_{a_j}. Note that

$$\mathbb{E}[|\mathcal{L}_j|] \ge \frac{|s_{a_j}|}{4},$$

by Eq. (10). Observe that $\mathcal{L}_i \subseteq s_{a_j}$, since all slots filled starting with s_{a_j} as the set of empty slots must come from s_{a_j}. We will now argue that, because $\mathbb{E}[|\mathcal{L}_j|]$ is so large, we are guaranteed that $\mathbb{E}[|\mathcal{L}_j \setminus \bigcup_{k>j} s_{a_k}|]$ is also large, namely, $\Omega(|s_{a_j}|)$.

Define

$$t_j = \bigcup_{k>j} s_{a_k},$$
$$v_j = s_{a_j} \setminus t_j.$$

and note that the v_js are disjoint:

Claim 10. $v_j \cap v_k = \emptyset$ for all $j \neq k$. That is, all the v_j's are mutually disjoint.

Proof. Without loss of generality, suppose $j < k$. By the definition of t_j, $s_{a_k} \subseteq t_j$. By the definition of v_k, $v_k \subseteq s_{a_k}$. Finally, by the definition of v_j, $v_j \cap t_j = \emptyset$. Therefore, $v_j \cap v_k = \emptyset$. □

As we saw earlier, $|t_j| = |s_{a_j+1} \cup \cdots \cup s_{a_\alpha}| \le |s_{a_j}|/7$. Since $\mathcal{L}_i \subseteq s_{a_j} \subseteq v_j \cup t_j$, we have that

$$\frac{|s_{a_j}|}{4} \le \mathbb{E}\left[|\mathcal{L}_i|\right] = \mathbb{E}\left[|\mathcal{L}_i \cap v_j|\right] + \mathbb{E}\left[|\mathcal{L}_i \cap t_j|\right] \le \mathbb{E}\left[|\mathcal{L}_i \cap v_j|\right] + \frac{|s_{a_j}|}{7}.$$

Subtracting, we get that

$$\mathbb{E}\left[|\mathcal{L}_i \cap v_j|\right] \ge \frac{|s_{a_j}|}{4} - \frac{|s_{a_j}|}{7} \ge \frac{|s_{a_j}|}{16}. \tag{11}$$

The high-level idea for the rest of the proof is as follows. We want to argue that the v_j's are disjoint sets that each have a reasonably large (i.e., $\Omega(1)$) probability of having an element appear in the first $2c$ probes $\mathcal{H}_{2c}$ of a given probe sequence. From this, we will be able to deduce that c is asymptotically at least as large as the number of v_j's, which is $\Omega(\log \delta^{-1})$.

Let

$$p_{i,j} = \Pr[Y_i \le 2c \text{ and } L_i \in v_j],$$
$$q_j = \Pr[\mathcal{H}_{2c} \cap v_j \neq \emptyset].$$

We necessarily have $p_{i,j} \le q_j$, since, for $Y_i \le 2c$ and $L_i \in v_j$, we must have that at least some hash function in the first $2c$ outputted an index in v_j. We thus have that

$$\frac{|s_{a_j}|}{16} \le \mathbb{E}[|\mathcal{L}_j \cap v_j|] = \sum_{i=a_j+1}^{m} p_{i,j} \le \sum_{i=a_j+1}^{m} q_j = q_j(m - a_j) \le q_j(n - a_j) = q_j|s_{a_j}|.$$

From this, we conclude that $q_j \geq \frac{1}{16}$ for all $j \in [\alpha - 1]$. Therefore,

$$\begin{aligned} 2c = |\{H_i : i \in [2c]\}| &= |\{H_i : i \in [2c]\} \cap [n]| \\ &= \mathbb{E}[|\{H_i : i \in [2c]\} \cap [n]|] \\ &\geq \sum_{j=1}^{\alpha-1} \mathbb{E}[|\{H_i : i \in [2c]\} \cap v_j|] \\ &\geq \sum_{j=1}^{\alpha-1} q_j \geq \frac{1}{16}(\alpha - 1) = \Omega(\log \delta^{-1}), \end{aligned}$$

and we are done.

□

5.3 High-Probability Worst-Case Probe Complexity

To prove the high probability lower bounds, we will use a similar set construction to that in the previous proof. The one difference is that we now have a cap on the maximum probe complexity. In terms of the variables used in the proof of Theorem 5, this one extra constraint allows us to obtain a stronger bound on $\mathbb{E}[\mathcal{H}_{2c} \cap v_j]$—namely our bound on this quantity will increase from $\Omega(1)$ to $\Omega(\log \delta^{-1})$.

The main idea is that, since we now have a *worst-case* upper bound c on how many probes an insertion can use, we can more explicitly analyze the actual probability that a particular slot *ever* gets probed. As will show, for a slot to be seen with probability greater than $1 - \delta$ (which is necessary for a $(1-\delta)$-fraction of slots to get filled), it must appear in $\mathcal{H}_c$ with probability at least $\Omega(\log \delta^{-1}/n)$. Integrating this into our analysis, we will be able to pick up an extra $\Omega(\log \delta^{-1})$ factor compared to the proof of Theorem 5.

Theorem 6. In any open-addressing scheme that does not perform reordering, with probability greater than $1/2$, there must be some key whose probe complexity ends up as $\Omega(\log^2 \delta^{-1})$. In other words,

$$\Pr\left[Y_i \leq c \, \forall i \in [m]\right] \leq \frac{1}{2},$$

for all $c \in o(\log^2 \delta^{-1})$.

Proof. Suppose that some open-addressing scheme that does not perform reordering exists such that, for some $c \in \mathbb{N}$, the probe complexity of all keys is at most c with probability greater than $1/2$. We will show that $c = \Omega(\log^2 \delta^{-1})$.

By definition, we have that

$$\Pr[Y_i \leq c \, \forall i \in [m]] > \frac{1}{2}. \tag{12}$$

Therefore, for each $i \in \{0, 1, \ldots, m\}$, there must be some $s_i \subseteq [n]$ of size $n - i$ such that

$$\Pr\left[Y_i \leq c \, \forall i \in [m] \,\middle|\, S_i = s_i\right] > \frac{1}{2}.$$

Otherwise, by the definition of conditional probability, we would contradict Eq. (12).

Claim 11. For any $i < n(1-256\delta)$, there must be some set $r_i \subseteq s_i \subseteq [n]$ with $|r_i| > \frac{n-i}{2} = \frac{|s_i|}{2}$ such that, for any $x \in r_i$,

$$P[x \in \mathcal{H}_c] > \frac{1}{32}\frac{\log\left(\frac{|s_i|}{n\delta}\right)}{|s_i|}.$$

Proof. A necessary condition for a table to succeed (that is, for all keys to have a probe complexity of at most c) with a load factor of $1-\delta$ is that

$$s_i \setminus \cup_{j>i}\mathcal{H}_c(k_j)$$

has size at most δn. Indeed, these are the set of slots that are empty after the insertion of k_i, and that never get probed by (the first c probes) of any of the remaining insertions.

Therefore,

$$\Pr\left[\left|s_i \cap \left(\bigcup_{j=i+1}^{m}\mathcal{H}_c(k_j)\right)\right| > |s_i| - \delta n : S_i = s_i\right] > \frac{1}{2}. \tag{13}$$

Note that the conditioning on $S_i = s_i$ is unnecessary, as the random variables $\mathcal{H}_c(k_j)$, $j > i$, are independent of the event $S_i = s_i$.

Let $p = \frac{\log\left(\frac{|s_i|}{n\delta}\right)}{|s_i|}$, and let t_i be the set of all slots $x \in s_i$ such that

$$\Pr[x \in \mathcal{H}_c] \leq \frac{p}{32}.$$

We will complete the proof of the claim by showing that $|t_i| < |s_i|/2$. To do so, we will calculate the number of elements in t_i that are expected to appear in some $H_c(k_j)$ with $j > i$:

$$\begin{aligned}
\mathbb{E}\left[\left|t_i \cap \left(\bigcup_{j=i+1}^{m}\mathcal{H}_c(k_j)\right)\right|\right] &= \sum_{x\in t_i}\Pr\left[x \in \bigcup_{j=i+1}^{m}\mathcal{H}_c(k_j)\right] \\
&= \sum_{x\in t_i} 1-(1-\Pr\left[x\in\mathcal{H}_c\right])^{m-i} \text{ (since } \mathcal{H}_c(k_j) \text{ is iid across } j) \\
&\leq \sum_{x\in t_i} 1-\left(1-\frac{p}{32}\right)^{|s_i|-n\delta} \\
&\leq \sum_{x\in t_i} 1-\left(1-\frac{p}{32}\right)^{|s_i|/2} \qquad \text{(since } i < n(1-2\delta) \text{ by assumption)} \\
&\leq |t_i| - |t_i|\left(1-\frac{1}{|s_i|}\right)^{\log\left(\frac{|s_i|}{n\delta}\right)|s_i|/64} \qquad \text{(since } (1-x/t) \leq (1-1/t)^x \text{ if } x,t \geq 1) \\
&< |t_i| - |t_i|(1/2)^{\log\left(\frac{|s_i|}{n\delta}\right)/8} \\
&< |t_i| - |t_i|\left(\frac{n\delta}{|s_i|}\right)^{1/8}.
\end{aligned}$$

By assumption, $i < n(1-256\delta)$, so $|s_i| > n - n(1-256\delta) = 256n\delta$. Since $|s_i| > 256n\delta$, we have that $\left(\frac{n\delta}{|s_i|}\right)^{1/8} < \left(\frac{1}{256}\right)^{1/8} = \frac{1}{2}$. We thus have that

$$|t_i| - |t_i|\left(\frac{n\delta}{|s_i|}\right)^{1/8} < \frac{|t_i|}{2}.$$

For the table insertion process to succeed, we must have that

$$\mathbb{E}\left[\left|t_i \cap \left(\bigcup_{j=i+1}^{m} \mathcal{H}_c(k_j)\right)\right|\right] \geq |t_i| - n\delta,$$

as otherwise more than $n\delta$ slots are never part of any hash function output and therefore are guaranteed to be unfilled at the end. It follows that $|t_i| < 2n\delta < |s_i|/2$, as desired.

□

Let $a_i = n\left(1 - \frac{1}{4^i}\right)$ for $i \in [\log \delta^{-1}/4]$. Observe that, for any $i \in [\log \delta^{-1}/4]$,

$$\left|\bigcup_{j=i+1}^{\log \delta^{-1}/4} s_{a_j}\right| \leq \sum_{j=1}^{\log \delta^{-1}/4-i} \frac{|s_{a_i}|}{4^j} \leq \frac{|s_{a_i}|}{3} \leq \frac{3|s_{a_i}|}{8}.$$

Let

$$v_i = r_{a_i} \setminus \left(\bigcup_{j=i+1}^{\log \delta^{-1}/4} s_{a_j}\right),$$

for $i \in [\log \delta^{-1}/4]$. By Claim 11, we have that $|r_{a_i}| \geq \frac{|s_{a_j}|}{2}$, so

$$|v_i| \geq \frac{|s_{a_i}|}{2} - \frac{3\,|s_{a_i}|}{8} = \frac{|s_{a_i}|}{8}.$$

Note that $v_i \cap v_j = \emptyset$ for all $i \neq j$ with a similar proof to Claim 10 in the previous subsection. Also, note that

$$|s_{a_i}| \geq n\left(\frac{1}{4}\right)^{\log \delta^{-1}/4} = n\left(\frac{1}{2}\right)^{\log \delta^{-1}/2} = n\sqrt{\delta}.$$

We now obtain a lower bound on $|\mathcal{H}_c| \leq c$ by unrolling the definition of each v_i. In particular,

assuming without loss of generality that $\log \delta^{-1}/4 > 256$, we have

$$\begin{aligned}
c \geq \mathbb{E}[|\mathcal{H}_c|] &\geq \sum_{i=1}^{\log \delta^{-1}/4} \mathbb{E}[|\mathcal{H}_c \cap v_i|] \\
&= \sum_{i=1}^{\log \delta^{-1}/4} \sum_{x \in v_i} \mathbb{E}[|\{x\} \cap \mathcal{H}_c|] \\
&= \sum_{i=1}^{\log \delta^{-1}/4} \sum_{x \in v_i} \Pr[x \in \mathcal{H}_c] \\
&\geq \sum_{i=1}^{\log \delta^{-1}/4} \sum_{x \in v_i} \frac{1}{32} \frac{\log\left(\frac{|s_{a_i}|}{n\delta}\right)}{|s_{a_i}|} \\
&= \frac{1}{32} \sum_{i=1}^{\log \delta^{-1}/4} |v_i| \frac{\log\left(\frac{|s_{a_i}|}{n\delta}\right)}{|s_{a_i}|} \\
&\geq \frac{1}{32} \sum_{i=1}^{\log \delta^{-1}/4} \frac{|s_{a_i}|}{8} \frac{\log\left(\frac{n\sqrt{\delta}}{n\delta}\right)}{|s_{a_i}|} \\
&= \frac{1}{32} \sum_{i=1}^{\log \delta^{-1}/4} \frac{\log \delta^{-1}}{16} && (\text{since } \log(1/\sqrt{\delta}) = \log \delta^{-1}/2) \\
&= \frac{1}{32} \cdot \frac{1}{16} \cdot \frac{1}{4} \log^2 \delta^{-1} = \Omega(\log^2 \delta^{-1}),
\end{aligned}$$

as desired.

□

We now combine our result above with a known result to obtain our full lower bound:

Theorem 7. In any open-addressing scheme that does not support reordering, there must be some key whose probe complexity ends up as $\Omega(\log \log n + \log^2 \delta^{-1})$ with probability greater than $1/2$, assuming that $1 - \delta = \Omega(1)$.

Proof. We only need to prove that some key has probe complexity $\Omega(\log \log n)$, as we already proved there is some key with probe complexity $\Omega(\log^2 \delta^{-1})$ in Theorem 6. Our proof mirrors the proof of Theorem 5.2 in [3], which in turn is primarily based on the following theorem in [20]:

Theorem 8 (Theorem 2 in [20]). Suppose m balls are sequentially placed into m bins using an arbitrary mechanism, with the sole restriction that each ball chooses between d bins according to some arbitrary distribution on $[m]^d$. Then the fullest bin has $\Omega(\log \log n/d)$ balls at the end of the process with high probability.

Now, suppose we have some arbitrary open-addressing scheme that does not perform reordering, where, with probability greater than $1/2$, all keys have probe complexity at most d. We modify our hash table scheme into a balls and bins process that chooses between at most d bins as follows.

Suppose key k_i is inserted into location $l_i = h_j(k_i)$. If $j \leq d$, then place ball i into bin $h_j(k_i) \bmod m$. Otherwise, place ball i into bin $h_d(k_i) \bmod m$, in this way ensuring that the scheme chooses between

at most d bins; the set of possible choices is $\{H_j(k_i) \bmod m : j \leq d\}$, a set of size (at most) d. This process also ensures that the fullest bin most likely has few balls land in it:

Lemma 12. With probability greater than 1/2, the fullest bin has $O(1)$ balls at the end of the process.

Proof. Suppose that ball i lands in bin B_i. The indices of the balls that land in the jth bin are thus $\{i : B_i = j\}$.

Now, suppose that $B_i = L_i \bmod m$ for all $i \in [m]$, and note that this happens with probability greater than 1/2. Since each slot can only store one key, $L_i \neq L_j$ for any $i \neq j$. Therefore, $\{i : B_i = j\} \subseteq \{i \in [n] : i \bmod m = j\}$. Since $1 - \delta = \Omega(1)$, we have that $m = \Omega(n)$, or $n = O(m)$. Thus, $|\{i \in [n] : i \bmod m = j\}| = O(1)$ for all $i \in [m]$, and the fullest bin has at most $O(1)$ balls, as desired. □

By Theorem 8, the fullest bin has $\Omega(\log \log n/d)$ balls at the end of the process with high probability. If $d = o(\log \log n)$, then the fullest bin has $\omega(1)$ balls in the fullest bin with high probability, contradicting Lemma 12. Therefore, $d = \Omega(\log \log n)$, as desired. □

6 Acknowledgments and Funding

The authors would like to thank Mikkel Thorup for several useful conversations, including feedback on an earlier version of this paper.

William Kuszmaul was partially supported by a Harvard Rabin Postdoctoral Fellowship and by a Harvard FODSI fellowship under NSF grant DMS-2023528. Martín Farach-Colton was partially supported by the Leanard J. Shustek Professorship and NSF grants CCF-2106999, NSF-BSF CCF-2247576, CCF-2423105, and CCF-2420942.

References

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. There is no fast single hashing algorithm. *Information Processing Letters*, 7(6):270–273, 1978.

[2] Michael A Bender, Alex Conway, Martín Farach-Colton, William Kuszmaul, and Guido Tagliavini. Iceberg hashing: Optimizing many hash-table criteria at once. *Journal of the ACM*, 70(6):1–51, 2023.

[3] Michael A. Bender, Alex Conway, Martín Farach-Colton, William Kuszmaul, and Guido Tagliavini. *Tiny Pointers*, pages 477–508. 2023. doi:10.1137/1.9781611977554.ch21.

[4] Michael A. Bender, Martin Farach-Colton, Simai He, Bradley C. Kuszmaul, and Charles E. Leiserson. Adversarial contention resolution for simple channels. In *SPAA*, pages 325–332. ACM, 2005.

[5] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: the heavily loaded case. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 745–754. Association for Computing Machinery, 2000. doi:10.1145/335305.335411.

[6] Richard P Brent. Reducing the retrieval time of scatter storage techniques. *Communications of the ACM*, 16(2):105–109, 1973.

[7] Andrei Z Broder and Anna R Karlin. Multilevel adaptive hashing. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 43–53, 1990.

[8] Walter A Burkhard. External double hashing with choice. In *8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05)*, pages 8–pp. IEEE, 2005.

[9] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, 2005.

[10] Gaston H Gonnet and J Ian Munro. Efficient ordering of hash tables. *SIAM Journal on Computing*, 8(3):463–478, 1979.

[11] Donald E Knuth. Notes on "open" addressing. *Unpublished memorandum*, pages 11–97, 1963.

[12] Donald E Knuth. Computer science and its relation to mathematics. *The American Mathematical Monthly*, 81(4):323–343, 1974.

[13] Donald Ervin Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 2nd edition, 1998. URL: `https://www.worldcat.org/oclc/312994415`.

[14] George Lueker and Mariko Molodowitch. More analysis of double hashing. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 354–359, 1988.

[15] Paul M Martini and Walter A Burkhard. Double hashing with multiple passbits. *International Journal of Foundations of Computer Science*, 14(06):1165–1182, 2003.

[16] Mariko Molodowitch. *Analysis and design of algorithms: double hashing and parallel graph searching*. University of California, Irvine, 1990.

[17] J Ian Munro and Pedro Celis. Techniques for collision resolution in hash tables with open addressing. In *Proceedings of 1986 ACM Fall joint computer conference*, pages 601–610, 1986.

[18] Peter Sanders. Hashing with linear probing and referential integrity. *arXiv preprint arXiv:1808.04602*, 2018.

[19] Jeffrey D Ullman. A note on the efficiency of hashing functions. *Journal of the ACM (JACM)*, 19(3):569–575, 1972.

[20] Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM (JACM)*, 50(4):568–589, 2003.

[21] Andrew C Yao. Uniform hashing is optimal. *Journal of the ACM (JACM)*, 32(3):687–693, 1985.