Optimally Sorting Evolving Data

Juan Jose Besa
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
jjbesavi@uci.edu
https://orcid.org/0000-0002-5676-7011

William E. Devanny[1]
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
wdevanny@uci.edu

David Eppstein
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
eppstein@uci.edu

Michael T. Goodrich
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
goodrich@uci.edu

Timothy Johnson
Dept. of Computer Science, Univ. of California, Irvine, Irvine, CA 92697 USA
tujohnso@uci.edu

Abstract

We give optimal sorting algorithms in the *evolving data* framework, where an algorithm's input data is changing while the algorithm is executing. In this framework, instead of producing a final output, an algorithm attempts to maintain an output close to the correct output for the current state of the data, repeatedly updating its best estimate of a correct output over time. We show that a simple repeated insertion-sort algorithm can maintain an $O(n)$ Kendall tau distance, with high probability, between a maintained list and an underlying total order of n items in an evolving data model where each comparison is followed by a swap between a random consecutive pair of items in the underlying total order. This result is asymptotically optimal, since there is an $\Omega(n)$ lower bound for Kendall tau distance for this problem. Our result closes the gap between this lower bound and the previous best algorithm for this problem, which maintains a Kendall tau distance of $O(n \log \log n)$ with high probability. It also confirms previous experimental results that suggested that insertion sort tends to perform better than quicksort in practice.

2012 ACM Subject Classification Theory of computation → Sorting and searching

Keywords and phrases Sorting, Evolving data, Insertion sort

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.81

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.03350`.

Funding This article reports on work supported by the DARPA under agreement no. AFRL FA8750-15-2-0092. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. This work was also supported in part from NSF grants 1228639, 1526631, 1217322, 1618301, and 1616248.

[1] Supported by an NSF Graduate Research Fellowship under grant DGE-1321846.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 81; pp. 81:1–81:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the classic version of the sorting problem, we are given a set, S, of n comparable items coming from a fixed total order and asked to compute a permutation that places the items from S into non-decreasing order, and it is well-known that this can be done using $O(n \log n)$ comparisons, which is asymptotically optimal (e.g., see [6, 8, 14]). However, there are a number of interesting applications where this classic version of the sorting problem doesn't apply.

For instance, consider the problem of maintaining a ranking of a set of sports teams based on the results of head-to-head matches. A typical approach to this sorting problem is to assume there is a fixed underlying total order for the teams, but that the outcomes of head-to-head matches (i.e., comparisons) are "noisy" in some way. In this formulation, the ranking problem becomes a one-shot optimization problem of finding the most-likely fixed total order given the outcomes of the matches (e.g., see [5, 7, 9, 10, 15]). In this paper, we study an alternative, complementary motivating scenario, however, where instead of there being a fixed total order and noisy comparisons we have a scenario where comparisons are accurate but the underlying total order is evolving. This scenario, for instance, captures the real-world phenomenon where sports teams make mid-season changes to their player rosters and/or coaching staffs that result in improved or degraded competitiveness relative to other teams. That is, we are interested in the sorting problem for *evolving data*.

1.1 Related Prior Work for Evolving Data

Anagnostopoulos *et al.* [1] introduce the *evolving data* framework, where an input data set is changing while an algorithm is processing it. In this framework, instead of an algorithm taking a single input and producing a single output, an algorithm attempts to maintain an output close to the correct output for the current state of the data, repeatedly updating its best estimate of the correct output over time. For instance, Anagnostopoulos *et al.* [1] mention the motivation of maintaining an Internet ranking website that displays an ordering of entities, such as political candidates, movies, or vacation spots, based on evolving preferences.

Researchers have subsequently studied other interesting problems in the evolving data framework, including the work of Kanade *et al.* [13] on stable matching with evolving preferences, the work of Huang *et al.* [12] on selecting top-k elements with evolving rankings, the work of Zhang and Li [18] on shortest paths in evolving graphs, the work of Anagnostopoulos *et al.* [2] on st-connectivity and minimum spanning trees in evolving graphs, and the work of Bahmani *et al.* [3] on PageRank in evolving graphs. In each case, the goal is to maintain an output close to the correct one even as the underlying data is changing at a rate commensurate to the speed of the algorithm. By way of analogy, classical algorithms are to evolving-data algorithms as throwing is to juggling.

1.2 Problem Formulation for Sorting Evolving Data

With respect to the sorting problem for evolving data, following the formulation of Anagnostopoulos *et al.* [1], we assume that we have a set, S, of n distinct items that are properly ordered according to a total order relation, "$<$". In any given time step, we are allowed to compare any pair of items, x and y, in S according to the "$<$" relation and we learn the correct outcome of this comparison. After we perform such a comparison, α pairs of items that are currently consecutive according to the "$<$" relation are chosen uniformly at random and their relative order is swapped. As in previous work [1], we focus on the case

where $\alpha = 1$, but one can also consider versions of the problem where the ratio between comparisons and random consecutive swaps is something other than one-to-one. Still, this simplified version with a one-to-one ratio already raises some interesting questions.

Since it is impossible in this scenario to maintain a list that always reflects a strict ordering according to the "$<$" relation, our goal is to maintain a list with small *Kendall tau* distance, which counts the number of inversions, relative to the correct order.[2] Anagnostopoulos *et al.* [1] show that, for $\alpha = 1$, the Kendall tau distance between the maintained list and the underlying total order is $\Omega(n)$ in both expectation and with high probability. They also show how to maintain this distance to be $O(n \log \log n)$, with high probability, by performing a multiplexed batch of quicksort algorithms on small overlapping intervals of the list. Recently, Besa Vial *et al.* [4] empirically show that repeated versions of quadratic-time algorithms such as bubble sort and insertion sort seem to maintain an asymptotically optimal distance of $O(n)$. In fact, this linear upper bound seems to hold even if we allow α, the number of random swaps at each step, to be a much larger constant.

1.3 Our Contributions

The main contribution of the present paper is to prove that repeated insertion sort maintains an asymptotically optimal Kendall tau distance, with high probability, for sorting evolving data. This algorithm repeatedly makes in-place insertion-sort passes (e.g., see [6, 8]) over the list, l_t, maintained by our algorithm at each step t. Each such pass moves the item at position j to an earlier position in the list so long as it is bigger than its predecessor in the list. With each comparison done by this repeated insertion-sort algorithm, we assume that a consecutive pair of elements in the underlying ordered list, l'_t, are chosen uniformly at random and swapped. In spite of the uncertainty involved in sorting evolving data in this way, we prove the following theorem, which is the main result of this paper.

▶ **Theorem 1.** *Running repeated insertion-sorts algorithm, for every step $t = \Omega(n^2)$, the Kendall tau distance between the maintained list, l_t, and the underlying ordered list, l'_t, is $O(n)$ with exponentially high probability.*

That is, after an initialization period of $\Theta(n^2)$ steps, the repeated insertion-sort algorithm converges to a steady state having an asymptotically optimal Kendall tau distance between the maintained list and the underlying total order, with exponentially high probability. We also show how to reduce this initialization period to be $\Theta(n \log n)$ steps, with high probability, by first performing a quicksort algorithm and then following that with the repeated insertion-sort algorithm.

Intuitively, our proof of Theorem 1 relies on two ideas: the adaptivity of insertion sort and that, as time progresses, a constant fraction of the random swaps fix inversions. Ignoring the random swaps for now, when there are k inversions, a complete execution of insertion sort performs roughly $k + n$ comparisons and fixes the k inversions (e.g., see [6, 8]). If an ϵ fraction of the random swaps fix inversions, then during insertion sort $\epsilon(k + n)$ inversions are fixed by the random swaps and $(1 - \epsilon)(k + n)$ are introduced. Naively the total change in the number of inversions is then $(1 - 2\epsilon)(k + n) - k$ and when $k > \frac{1-2\epsilon}{2\epsilon}n$, the number of inversions decreases. So the number of inversions will decrease until $k = O(n)$.

This simplistic intuition ignores two competing forces involved in the heavy interplay between the random swaps and insertion sort's runtime, however, in the evolving data model,

[2] Recall that an *inversion* is a pair of items u and v such that u comes before v in a list but $u > v$. An *inversion* in a permutation π is a pair of elements $x \neq y$ with $x < y$ and $\pi(x) > \pi(y)$.

Algorithm 1 Repeated insertion sort pseudocode

```
function REPEATED_INSERTION_SORT(l)
    while true do
        for i ← 1 to n − 1 do
            j ← i
            while j > 0 and l[j] < l[j − 1] do
                swap l[j] and l[j − 1]
                j ← j − 1
```

which necessarily complicates our proof. First, random swaps can cause an insertion-sort pass to end too early, thereby causing insertion sort to fix fewer inversions than normal. Second, as insertion sort progresses, it decreases the chance for a random swap to fix an inversion. Analyzing these two interactions comprises the majority of our proof of Theorem 1.

In Section 3, we present a complete proof of Theorem 1. The most difficult component of Theorem 1's proof is Lemma 6, which lower bounds the runtime of insertion sort in the evolving data model. The proof of Lemma 6 is presented separately in Section 4.

Due to space requirements, some proofs are left to the Arxiv version of the paper.

2 Preliminaries

The sorting algorithm we analyze in this paper for the evolving data model is the repeated insertion-sort algorithm whose pseudocode is shown in Algorithm 1.

Formally, at time t, we denote the sorting algorithms' list as l_t and we denote the underlying total order as l'_t. Together these two lists define a permutation, σ_t, of the indices, where $\sigma_t(x) = y$ if the element at index x in l_t is at position y in l'_t. We define the *simulated final state at time t* to be the state of l obtained by freezing the current underlying total order, l'_t, (i.e., no more random swaps) and simulating the rest of the current round of insertion sort (we refer to each iteration of the **while-true** loop in Algorithm 1 as a *round*). We then define a *frozen-state* permutation, $\hat{\sigma}_t$, where $\hat{\sigma}_t(x) = y$ if the element at index x in the simulated final state at time t as at index y in l'_t.

Let us denote the number of inversions at time t, in σ_t, with I_t. Throughout the paper, we may choose to drop time subscripts if our meaning is clear. The Kendall tau distance between two permutations π_1 and π_2 is the number of pairs of elements $x \neq y$ such that $\pi_1(x) < \pi_1(y)$ and $\pi_2(x) > \pi_2(y)$. That is, the Kendall tau distance between l_t and l'_t is equal to I_t, the number of inversions in σ_t. Figure 1 shows the state of l, l', I, and σ for two steps of an insertion sort (but not in the same round).

As the inner **while**-loop of Algorithm 1 executes, we can view l as being divided into three sets: the set containing just the *active* element, $l[j]$ (which we view as moving to the left, starting from position i, as it is participating in comparisons and swaps), the *semi-sorted* portion, $l[0 : i]$, not including $l[j]$, and the *unsorted* portion, $l[i + 1 : n - 1]$. Note that if no random adjacent swaps were occurring in l' (that is, if we were executing insertion-sort in the classical algorithmic model), then the semi-sorted portion would be in sorted order.

To understand the nature of the inversions in the semi-sorted portion, we will use the *Cartesian tree* [17]. Given a list, L, of m numbers with no two equal numbers, the Cartesian tree of L is a binary rooted tree on the numbers where the root is the minimum element $L[k]$, the left subtree of the root is the Cartesian tree of $L[0 : k - 1]$, and the right subtree of the root is the Cartesian tree of $L[k + 1 : m]$. In our analysis, we will primarily consider the Cartesian tree of the simulated final state at time t where $L[k] = \hat{\sigma}_t(k)$ in the frozen-state

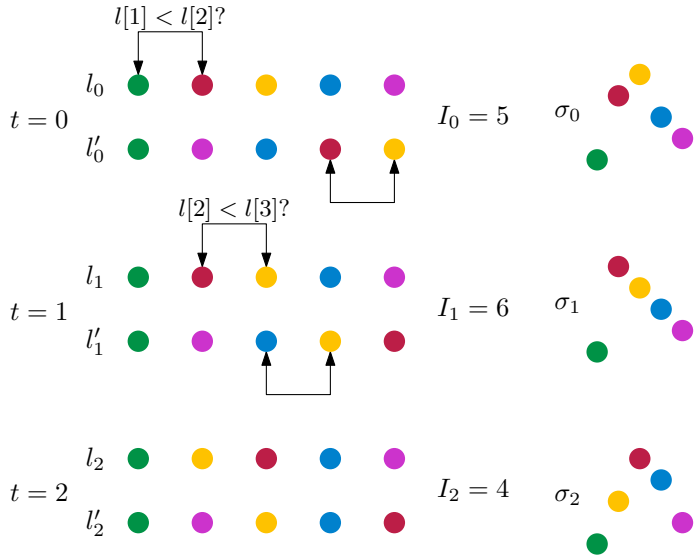

Figure 1 Examples of l, l', I, and σ over two steps of an algorithm. In the first step the green and red elements are compared in l and the red and yellow elements are swapped in l'. In the second step the red and yellow elements are compared and swapped in l and the blue and yellow elements are swapped in l'.

permutation $\hat{\sigma}_t$. We also choose to include two additional elements, $L[-1] = -1$ and $L[n] = n$, for boundary cases.

We call the path from the root to the rightmost leaf of the Cartesian tree the (right-to-left) minima path as the elements on this path are the right-to-left minima in the list. For a minimum, $l[k]$, denote with $M(k)$ the index of the element in the left subtree of $l[k]$ that maximizes $\hat{\sigma}(k)$, i.e., the index of the largest element in the left subtree.

We use the phrase *with high probability* to indicate when an event occurs with probability that tends towards 1 as $n \to \infty$. When an event occurs with probability of the form $1 - e^{-poly(n)}$, we say it occurs with *exponentially high probability*. During our analysis, we will make use of the following facts.

▶ **Lemma 2** (Poisson approximation (Corollary 5.9 in [16])). *Let $X_1^{(m)}, \ldots, X_n^{(m)}$ be the number of balls in each bin when m balls are thrown uniformly at random into n bins. Let $Y_1^{(m)}, \ldots, Y_n^{(m)}$ be independent Poisson random variables with $\lambda = m/n$. Then for any event $\varepsilon(x_1, \ldots, x_n)$:*

$$\Pr\left[\varepsilon\left(X_1^{(m)}, \ldots, X_n^{(m)}\right)\right] \leq e\sqrt{m}\Pr\left[\varepsilon\left(Y_1^{(m)}, \ldots, Y_n^{(m)}\right)\right].$$

▶ **Lemma 3** (Hoeffding's inequality (Theorem 2 in [11])). *If $X_1, \ldots, X_n$ are independent random variables and $a_k \leq X_k \leq b_k$ for $k = 1, \ldots, n$, then for $t > 0$:*

$$\Pr\left[\sum_k X_k - E\left[\sum_k X_k\right] \geq tn\right] \leq e^{-2n^2t^2/(\sum_k (b_k - a_k)^2)}.$$

3 Sorting Evolving Data with Repeated Insertion Sort

Let us begin with some simple bounds with respect to a single round of insertion sort.

▶ **Lemma 4.** *If a round of insertion sort starts at time t_s and finishes at time t_e, then*

1. *$t_e - t_s = F + n - 1$, where F is the number of inversions fixed (at the time of a comparison in the inner **while**-loop) by this round of insertion sort.*
2. *$t_e - t_s < n^2/2$*
3. *for any $t_s \leq t \leq t_e$, $I_t - I_{t_s} < n$.*

Proof. (1): For each iteration of the outer **for**-loop, each comparison in the inner **while**-loop either fixes an inversion (at the time of that comparison) or fails to fix an inversion and completes the inner **while**-loop. Note that this "failed" comparison may not have compared elements of l, but may have short circuited due to $j \leq 0$. Nevertheless, every comparison that doesn't fail fixes an inversion (at the time of that comparison); hence, each non-failing comparison is counted in F.

(2): In any round, there are at most $n(n-1)/2$ comparisons, by the formulations of the outer **for**-loop and inner **while**-loop.

(3): At time t, the round of insertion sort will have executed $t - t_s$ steps. Of those steps, at least $t - t_s - (n-1)$ comparisons resulted in a swap that removed an inversion and at most $n-1$ comparisons did not result in a change to l. The random swaps occurring during these comparisons introduced at most $t - t_s$ inversions. So $I_t - I_{t_s} \leq t - t_s - \big(t - t_s - (n-1)\big) = n - 1$. ◀

We next assert the following two lemmas, which are used in the next section.

▶ **Lemma 5.** *There exists a constant, $0 < \epsilon < 1$, such that, for a round of insertion sort that takes time t^*, at least ϵt^* of the random adjacent swaps in l' decrease I during the round, with exponentially high probability.*

Proof. Proof omitted due to space requirements. ◀

▶ **Lemma 6.** *If a round of insertion sort starts at time t_s with $I_{t_s} \geq (12c^2 + 2c)n$ and finishes at time t_e, then, with exponentially high probability, $t_e - t_s \geq cn$, i.e., the insertion sort round takes at least cn steps.*

Proof. See Section 4. ◀

3.1 Proof of Theorem 1

Armed with the above lemmas (albeit postponing the proofs of Lemma 5 and Lemma 6), let us prove our main theorem.

Theorem 1. *There exists a constant, $0 < \epsilon < 1$, such that, when running the repeated insertion-sort algorithm, for every step $t > (1 + 1/\epsilon)n^2$, the Kendall tau distance between the maintained list, l_t, and the underlying ordered list, l'_t, is $O(n)$, with exponentially high probability.*

Proof. By Lemma 5, there exists a constant $0 < \epsilon < 1$ such that at least an ϵ fraction of all of the random swaps during a round of insertion sort fix inversions. Consider an epoch of the last $(1 + 1/\epsilon)n^2$ steps of the repeated insertion-sort algorithm, that is, from time $t' = t - (1 + 1/\epsilon)n^2$ to t. During this epoch, some number, $m \geq 1$, of complete rounds of insertion sort are performed from start to end (by Lemma 4). Denote with t_k the time at which insertion-sort round k ends (and round $k+1$ begins), and let t_m denote the end time of the final complete round, during this epoch. By construction, observe that $t' \leq t_0$ and

$t_m \leq t$. Furthermore, because the insertion-sort rounds running before t_0 and after t_m take fewer than $n^2/2$ steps (by Lemma 4), $t_m - t_0 \geq n^2/\epsilon$.

The remainder of the proof consists of two parts. In the first part, we show that for some complete round of insertion sort ending at time $t_k \leq t$, I_{t_k} is $O(n)$, with exponentially high probability. In the second part, we show that once we achieve I_{t_k} being $O(n)$, for $t_k \leq t$, then I_t is $O(n)$, with exponentially high probability.

For the first part, suppose, for the sake of a contradiction, $I_{t_k} > \left(12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon}\right)n$, for all $0 \leq k \leq m$. Then, by a union bound over the polynomial number of rounds, Lemma 6 applies to every such round of insertion sort. So, with exponentially high probability, each round takes at least n/ϵ steps. Moreover, by Lemma 5, with exponential probability, an ϵ fraction of the random swaps from t_m to t_0 will decrease the number of inversions. That is, these random swaps increase the number of inversions by at most

$$(1-\epsilon)(t_m - t_0) - \epsilon(t_m - t_0) = (1-2\epsilon)(t_m - t_0),$$

with exponentially high probability. Furthermore, by Lemma 4, at least a $\frac{(1/\epsilon)-1}{1/\epsilon} = 1-\epsilon$ fraction of the insertion-sort steps fix inversions (at the time of a comparison). Therefore, with exponentially high probability, we have the following:

$$\begin{aligned} I_{t_m} &\leq I_{t_0} - (1-\epsilon)(t_m - t_0) + (1-2\epsilon)(t_m - t_0) \\ &= I_{t_0} - \epsilon(t_m - t_0) \\ &\leq I_{t_0} - n^2. \end{aligned}$$

But, since $I_{t_0} < n^2$, the above bound implies that $I_{t_m} < 0$, which is a contradiction. Therefore, with exponentially high probability, there is a $k \leq m$ such that $I_{t_k} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon})n$.

For the second part, we show that the probability for a round $\ell > k$ to have $I_{t_\ell} > (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$ is exponentially small, by considering two cases (and their implied union-bound argument):

- If $I_{t_{\ell-1}} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon})n$, then Lemma 4 implies $I_{t_\ell} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$.
- If $(12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon})n \leq I_{t_{\ell-1}} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$, then, similar to the argument given above, during a round of insertion sort, ℓ, at least a $1-\epsilon$ fraction of the steps fix an inversion, and an ϵ fraction of the steps do nothing. Also at least an ϵ fraction of the random swaps fix inversions, while a $1-\epsilon$ fraction add inversions. Finally, the total length of the round is $t_\ell - t_{\ell-1}$. Thus, with exponentially high probability, the total change in inversions is at most $-\epsilon(t_\ell - t_{\ell-1})$ and $I_{t_\ell} < I_{t_{\ell-1}}$.

Therefore, by a union bound over the polynomial number of insertion-sort rounds, the probability that any $I_{t_\ell} > (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n$ for $k < \ell \leq m$ is exponentially small. By Lemma 4, $I_t \leq I_{t_m} + n$. So, with exponentially high probability, $I_{t_m} \leq (12(\frac{1}{\epsilon})^2 + \frac{2}{\epsilon} + 1)n = O(n)$ and $I_t = O(n)$, completing the proof. ◀

3.2 Improved Convergence Rate

In this subsection, we provide an algorithm that converges to $O(n)$ inversions more quickly. To achieve the steady state of $O(n)$ inversions, repeated insertion sort performs $\Theta(n^2)$ comparisons. But this running time to reach a steady state is a worst-case based on the fact that the running time of insertion sort is $O(n+I)$, where I is the number of initial inversions in the list, and, in the worst case, I is $\Theta(n^2)$. By simply running a round of quicksort on l first, we can achieve a steady state of $O(n)$ inversions after just $\Theta(n \log n)$ comparisons. See Algorithm 2. That is, we have the following.

Algorithm 2 Quicksort followed by repeated insertion sort pseudocode

```
function QUICK_THEN_INSERTION_SORT(l)
    quicksort(l)
    while true do
        for i ← 1 to n − 1 do
            j ← i
            while j > 0 and l[j] < l[j − 1] do
                swap l[j] and l[j − 1]
                j ← j − 1
```

▶ **Theorem 7.** *When running Algorithm 2, for every $t = \Omega(n \log n)$, I_t is $O(n)$ with high probability.*

Proof. By the results of Anagnostopoulos *et al.* [1], the initial round of quicksort takes $\Theta(n \log n)$ comparisons and afterwards the number of inversions (that is, the Kendall tau distance between the maintained list and the true total order) is $O(n \log n)$, with high probability. Using a nearly identical argument to the proof of Theorem 1, and the fact that an insertion-sort round takes $O(I + n)$ time to resolve I inversions, the repeated insertion-sort algorithm will, with high probability, achieve $O(n)$ inversions in an additional $O(n \log n)$ steps. From that point on, it will maintain a Kendall tau distance of $O(n)$, with high probability. ◀

4 Proof of Lemma 6

Recall Lemma 6, which establishes a lower bound for the running time of an insertion-sort round, given a sufficiently large amount of inversions relative to the underlying total order.

Lemma 6. *If a round of insertion sort starts at time t_s with $I_{t_s} \geq (12c^2 + 2c)n$ and finishes at time t_e, then, with exponentially high probability, $t_e - t_s \geq cn$, i.e., the insertion sort round takes at least cn steps.*

The main difficulty in proving Lemma 6 is understanding how the adjacent random swaps in l' affect the runtime of the current round of insertion sort on l. Let S_t be the number of steps left to perform in the current round of insertion sort if there were no more random adjacent swaps in l'. In essence, S can be thought of as an estimate of the remaining time in the current insertion sort round. If a new round of insertion sort is started at time t_s, then $S_{t_s - 1} = 1$ and $I_{t_s} \leq S_{t_s} \leq I_{t_s} + n - 1$. Each step of an insertion sort round decreases S by one and the following random swap may increase or decrease S by some amount. Figure 2 illustrates an example where one random adjacent swap in l' decreases S by a non-constant amount (relative to n).

A random adjacent swap in l' involving two elements in the unsorted portion of l will either increase or decrease S by one depending on whether it introduces or removes an inversion. Random adjacent swaps involving elements in the semi-sorted portion have more complex effects on S.

An inversion currently in the list $\big(l[a], l[b]\big)$ will be fixed by insertion sort if $l[a]$ and $l[b]$ will be compared and the two are swapped. Because $a < b$, $l[b]$ must be the active element during this comparison. An inversion $\big(l[a], l[b]\big)$ will not be fixed by insertion sort if $l[b]$ was already inserted into the semi-sorted portion or there is some element $l[c]$ in the semi-sorted portion with $a < c < b$ and $\sigma(c) < \sigma(b)$. We call an inversion with $l[b]$ in the semi-sorted

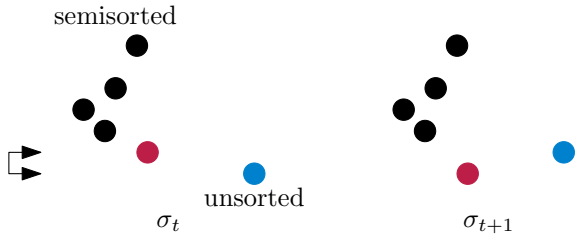

Figure 2 An example where swapping the ordering of the red and blue elements in l' creates multiple blocked inversions between the blue element and the black elements. Recall that our list is partitioned into the semisorted region, which contains elements that have already been compared in this round, and the unsorted region.

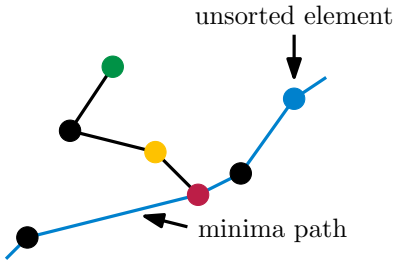

Figure 3 In this Cartesian tree, the green-blue pair is a blocked inversion and the green-yellow pair is a stuck inversion. Both pairs of inversions blame the red element.

portion a *stuck* inversion and an inversion with a smaller semi-sorted element between the pair a *blocked* inversion. We say an element $l[c]$ in the semi-sorted portion of l *blocks* an inversion $\big(l[a], l[b]\big)$ with $a \leq i$ and $l[b]$ either the active element or in the unsorted portion of l, if $l[c]$ is in the semi-sorted portion of l with $a < c < b$ and $\sigma(c) < \sigma(b)$. Note that there may be multiple elements blocking a particular inversion. Figure 3 shows examples of these two types of inversions.

We denote the number of "bad" inversions at time t that will not be fixed with B_t. That is, B_t is the sum of the blocked and stuck inversions. At the end of an insertion-sort round every inversion present at the start was either fixed by the insertion sort, fixed by a random adjacent swap in l', or is currently stuck. No elements can be blocked at the end of an insertion-sort round, because the semi-sorted portion is the entire list. Stuck inversions are either created by random adjacent swaps in l' or were blocked inversions and insertion sort finished inserting the right element of the pair. Blocked inversions are only introduced by the random adjacent swaps in l'. Thus B_t is unaffected by the steps of insertion sort.

Every inversion present at the start must be fixed by a step of insertion sort, be fixed by a random swap, or it will end up "bad". Therefore, for any given time, t, by using naive upper bounds based on the facts that every insertion sort step can fix an inversion and every random adjacent swap can remove an inversion, we can immediately derive the following:

▶ **Lemma 8.** *For an insertion sort round that starts at time t_s and ends at time t_e, if $t_s \leq t \leq t_e$, then $S_t \geq I_{t_s} - 2(t - t_s) - B_t$.*

Since, when an insertion sort round finishes, $S_{t_e - 1} = 1$, Lemma 8 implies $2(t_e - t_s - 1) + B_{t_e} + 1 \geq I_{t_s}$. If we understand how B changes with each random adjacent swap in l', then we can bound how long insertion sort needs to run for this inequality to be true.

We associate the blocked and stuck inversions with elements that we say are *blamed* for the inversions. A blocked inversion $\big(l[a], l[b]\big)$ blames the element $l[c]$ with $a < c < b$ and minimum $\sigma(c)$. Note than $l[c]$ is on the minima path of the modified Cartesian tree, and $l[a]$ is in the left subtree of $l[c]$. A stuck inversion either blames the element on the minima path whose subtree contains both $l[a]$ and $l[b]$ or if they appear in different subtrees, the inversion blames the element $l[c]$ with $a < c < b$ and minimum $\sigma(c)$. Again note that the blamed element is on the minima path and $l[a]$ is in the blamed element's left subtree. The bad inversions in Figure 3 blame the red element.

Whether stuck or blocked, every inversion blames an element on the minima path and the left element of the inverted pair appears in that minimum's subtree. If $l[k]$ is on the minima path, $M(k)$ is the index of the element in $l[k]$'s subtree with maximum $\sigma(M(k))$, and an inversion $\big(l[a], l[b]\big)$ has $l[a]$ in $l[k]$'s subtree, then both $l[a]$ and $l[b]$ are in the range $\sigma(k)$ to $\sigma(M(k))$. So we can upper bound B_t by $\sum_{k=0}^{n-1}(\sigma(M(k)) - \sigma(k))^2$, where we extend M to non-minima indices with $M(k) = k$ if k is not the index of a minima in l.

4.1 Bounding the Number of Blocked and Stuck Inversions with Counters

For the purposes of bounding B_t, we conceptually associate two counters, $Inc(x)$ and $Dec(x)$, with each element, x. The counters are initialized to zero at the start of an insertion sort round. When an element x is increased by a random swap in l', we increment $Inc(x)$ and when x is decreased by a random swap in l', we increment $Dec(x)$. After the random swap occurs, we may choose to exchange some of the counters between pairs of elements, but we will always maintain the following invariant:

Invariant 1. *For an element, $l[k]$, on the minima path,*

$$Inc\big(l[M(k)]\big) + Dec\big(l[k]\big) \geq \sigma\big(M(k)\big) - \sigma(k).$$

This invariant allows us to prove the following Lemma:

▶ **Lemma 9.** *If $\sum_{k=0}^{n-1} Inc\big(l[k]\big)^2 < \kappa$ and $\sum_{k=0}^{n-1} Dec\big(l[k]\big)^2 < \kappa$, then $B_t \leq 4\kappa$.*

Proof.

$$\begin{aligned} B_t &\leq \sum_{k=0}^{n-1} \Big(\sigma(M(k)) - \sigma(k)\Big)^2 \\ &\leq \sum_{k=0}^{n-1} \Big(Inc\big(M(k)\big) + Dec(k)\Big)^2 \qquad \text{By Invariant 1} \qquad (1) \end{aligned}$$

By the assumptions of this lemma, interpreting Inc and Dec as two n-dimensional vectors, we know their lengths are both less than $\sqrt{\kappa}$. Equation 1 is the squared length of the sum of the Dec and Inc vectors with the entries of Inc permuted by the function M. By the triangle inequality, the length of their sum is at most $2\sqrt{\kappa}$ and so the squared length of their sum is at most 4κ.

Therefore, $B_t \leq 4\kappa$. ◀

▶ **Lemma 10.** *There is a counter maintenance strategy that maintains Invariant 1 such that after each random adjacent swap in l', the corresponding counters are incremented and then some counters are exchanged between pairs of elements.*

Proof. Proof omitted due to space requirements. ◀

4.2 Bounding the Counters with Balls and Bins

We model the *Inc* and *Dec* counters each with a balls and bins process and analyze the sum of squares of balls in each bin. Each element in l is associated with one of n bins. When an element's *Inc* counter is increased, throw a ball into the corresponding bin. If a pair of *Inc* counters are exchanged, exchange the set of balls in the two corresponding bins. The *Dec* counters can be modeled similarly.

This process is almost identical to throwing balls into n bins uniformly at random. Note that the exchanging of balls in pairs of bins takes place after a ball has been placed in a chosen bin, effectively permuting two bin labels in between steps. If every bin was equally likely to be hit at each time step, then permuting the bin labels in this way would not change the final sum of squares and the exchanging of counters could be ignored entirely. Unfortunately the bin for the element at $l[n-1]$ in the case of *Inc* counters or $l[0]$ in the case of *Dec* counters cannot be hit, i.e., there is a forbidden bin controlled by the counter swapping strategy. However, even when in each round the forbidden bin is adversarially chosen, the sum of squares of the number of balls in each bin will be stochastically dominated by a strategy of always forbidding the bin with the lowest number of balls. Therefore, the sum of squares of m balls being thrown uniformly at random into $n-1$ bins stochastically dominates the sum of squares of the *Inc* (or *Dec*) counters after m steps.

▶ **Theorem 11.** *If cn balls are each thrown uniformly at random into n bins with $c > e$, then the sum over the bins of the square of the number of balls in each bin is at most $3c^2n$ with exponentially high probability.*

Proof. Let $X_1, \ldots, X_n$ be random variables where X_k is the number of balls in bin k and let $Y_1, \ldots, Y_n$ be independent Poisson random variables with $\lambda = c$.

By the Poisson approximation, Lemma 2,

$$\Pr\left[\sum_k X_k^2 \geq 3c^2n\right] \leq e\sqrt{cn}\Pr\left[\sum_k Y_k^2 \geq 3c^2n\right].$$

Let Z_k be the event that $Y_k \geq ecn^{1/6}$ and Z be the event that at least one Z_k occurs.

$$\Pr[Z] \leq n\Pr[Z_1] \quad \text{by a union bound.}$$

$$\begin{aligned}\Pr[Z_1] &= e^{-c}\sum_{k=ecn^{1/6}}^{\infty}\frac{c^k}{k!} \leq e^{-c}\sum_{k=ecn^{1/6}}^{\infty}\frac{c^k}{e\left(\frac{k}{e}\right)^k}\\ &= e^{-c-1}\sum_{k=ecn^{1/6}}^{\infty}\left(\frac{ec}{k}\right)^k \leq e^{-c-1}\sum_{k=ecn^{1/6}}^{\infty}\left(\frac{1}{n^{1/6}}\right)^k\\ &= e^{-c-1}(n^{1/6})^{-ecn^{1/6}}\sum_{k=0}^{\infty}\frac{1}{n^{1/6}}^k \leq e^{-c}n^{-\frac{ec}{6}n^{1/6}}.\end{aligned}$$

$$\Rightarrow \Pr[Z] \leq \frac{n}{e^c n^{\frac{ec}{6}n^{1/6}}} \leq e^{-\Omega(n^{1/6})}.$$

Letting $Y = \sum_k Y_k^2$:

$$E[Y|\neg Z] \leq E[Y] = nE[Y_1^2] = n\left(c + c^2\right) \leq 2c^2n.$$

Given $\neg Z$, $(Y_k)^2 \in [0, ecn^{1/3}]$. So we can apply Hoeffding's inequality, Lemma 3, to get:

$$\Pr\left[Y - E\left[Y|\neg Z\right] \geq tn|\neg Z\right] \leq e^{-2t^2n^2/\left(n\left(ecn^{1/3}\right)^2\right)}.$$

Setting $t = c^2$, we have:

$$\begin{aligned}\Pr\left[Y - E\left[Y|\neg Z\right] \geq c^2 n|\neg Z\right] &\leq e^{\left(-2c^4n^2\right)/\left(n\left(ecn^{1/3}\right)^2\right)} \\ &\leq e^{-2n^{1/3}}.\end{aligned}$$

Because $E\left[Y|\neg Z\right] \leq 2c^2 n$, we have $\Pr[Y \geq 3c^2 n|\neg Z] \leq e^{-\Omega(n^{1/3})}$.

$$\begin{aligned}\Pr\left[Y \geq 3c^2 n\right] &= \Pr\left[Y \leq 3c^2 n \text{ and } Z\right] + \Pr\left[Y \leq c^2 n \text{ and } \neg Z\right] \\ &\leq \Pr[Z] + \Pr\left[Y \leq 3c^2 n|\neg Z\right] \\ &\leq e^{-\Omega(n^{1/6})} + \Pr[Y - E[Y|\neg Z] \geq c^2 n|\neg Z] \\ &\leq e^{-\Omega(n^{1/6})} + e^{-\Omega(n^{1/3})} \leq 2e^{-\Omega(n^{1/6})}.\end{aligned}$$

Thus, we can conclude $\Pr[\sum_k X_k^2 \geq 3c^2 n] \leq \frac{2e\sqrt{cn}}{e^{\Omega(n^{1/6})}} \leq e^{-\,poly(n)}$. ◀

Recall that by Lemma 8, if an insertion-sort round ends at time t, then $I_{t_s} \leq 2(t - t_s) + B_t + 1$. Theorem 11 and a simple union bound tell us that if $t \leq t_s + cn$, then $\sum_{k=0}^{n-1} Inc\big(l[k]\big)^2 \leq 3c^2(n-1)$ and $\sum_{k=0}^{n-1} Dec\big(l[k]\big)^2 \leq 3c^2(n-1)$ with exponentially high probability. So by Lemma 9, $B_t \leq 12c^2 n$.

Recall that when the insertion sort round finishes, $2(t_e - t_s - 1) + B_{t_e} + 1 \geq I_{t_s}$. If fewer than cn steps have been performed, the left hand side of this inequality is less than $(12c^2+2c)n$ with exponentially high probability. Therefore, if we started with $(12c^2 + 2c)n$ inversions, the current round of insertion sort must perform at least cn steps with exponentially high probability; otherwise, there are unfixed but still "good" inversions. This completes the proof of Lemma 6.

5 Conclusion

We have shown that, although it is much simpler than quicksort and only fixes at most one inversion in each step, repeated insertion sort leads to the asymptotically optimal number of inversions in the evolving data model. We have also shown that by using a single round of quicksort before our repeated insertion sort, we can get to this steady state after an initial phase of $O(n \log n)$ steps, which is also asymptotically optimal.

For future work, it would be interesting to explore whether our results can be composed with other problems involving algorithms for evolving data, where sorting is a subcomponent. In addition, our analysis in this paper is specific to insertion sort, and only applies when exactly one random swap is performed after each comparison. We would like to extend this to other sorting algorithms that have been shown to perform well in practice and to the case in which the number of random swaps per comparison is a larger constant. Finally, it would also be interesting to explore whether one can derive a much better ϵ value than we derived in the proof of Lemma 5.

References

1 Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011. Special issue on selected papers from 36th International Colloquium on Automata, Languages and Programming (ICALP 2009). doi:10.1016/j.tcs.2010.10.003.

2 Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *3rd ACM Innovations in Theoretical Computer Science Conference (ITCS)*, pages 149–160, 2012. doi:10.1145/2090236.2090249.

3 Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 24–32, 2012. doi:10.1145/2339530.2339539.

4 Juan Jose Besa Vial, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Quadratic time algorithms appear to be optimal for sorting evolving data. In *Proc. Algorithm Engineering & Experiments (ALENEX 2018)*, pages 87–96, 2018. doi:10.1137/1.9781611975055.8.

5 Mark Braverman and Elchanan Mossel. Noisy sorting without resampling. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 268–276, 2008.

6 Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

7 Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. doi:10.1137/S0097539791195877.

8 Michael T. Goodrich and Roberto Tamassia. *Algorithm Design and Applications*. Wiley Publishing, 1st edition, 2014.

9 Benoit Groz and Tova Milo. Skyline queries with noisy comparisons. In *34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 185–198, 2015. doi:10.1145/2745754.2745775.

10 Dorit S. Hochbaum. Ranking sports teams and the inverse equal paths problem. In Paul Spirakis, Marios Mavronicolas, and Spyros Kontogiannis, editors, *2nd Int. Workshop on Internet and Network Economics (WINE)*, volume 4286 of *Lecture Notes in Computer Science*, pages 307–318, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11944874_28.

11 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1080/01621459.1963.10500830.

12 Qin Huang, Xingwu Liu, Xiaoming Sun, and Jialin Zhang. Partial sorting problem on evolving data. *Algorithmica*, 79(3):1–24, 2017. doi:10.1007/s00453-017-0295-3.

13 Varun Kanade, Nikos Leonardos, and Frédéric Magniez. Stable Matching with Evolving Preferences. In Klaus Jansen, Claire Mathieu, José D. P. Rolim, and Chris Umans, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 60 of *LIPIcs*, pages 36:1–36:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2016.36.

14 Donald Ervin Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Pearson Education, 2nd edition, 1998.

15 Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Sorting noisy data with partial information. In *4th ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 515–528, 2013. doi:10.1145/2422436.2422492.

16 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.

17 Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

18 Jialin Zhang and Qiang Li. Shortest paths on evolving graphs. In H. Nguyen and V. Snasel, editors, *5th Int. Conf. on Computational Social Networks (CSoNet)*, volume 9795 of *Lecture Notes in Computer Science*, pages 1–13, Berlin, Heidelberg, 2016. Springer. doi:10.1007/978-3-319-42345-6_1.