Optimal Stochastic Online Sorting

Daniel Anker Hermansen
Aarhus University, Denmark

Abstract

In the online sorting problem that was introduced by Aamand, Abrahamsen, Beretta and Kleist [SODA 2023], n items of real numbers arrive in an online fashion. Each item has to be placed irrevocably into an array of size n before the next item is revealed. The cost is the sum of the absolute differences between adjacent items. We study the stochastic online sorting problem where the items are sampled uniformly at random from the interval $[0, 1]$, a problem first studied by Abrahamsen, Bercea, Klausen and Kozma [ESA 2024], who presented an algorithm achieving an expected competitive ratio of $O\big((n \log n)^{1/4}\big)$. Later Hu [SODA 2026] achieved an improved expected competitive ratio of $\log n \cdot 2^{O(\log^* n)}$. Hu also showed a lower bound of an expected competitive ratio of $\Omega(\log n)$. In this paper, we present a simple algorithm achieving an expected competitive ratio of $O(\log n)$, thus settling the complexity. In the variant where the array has size $\lceil (1+\varepsilon)n \rceil$, our algorithm achieves an expected competitive ratio of $O(1 + \log \varepsilon^{-1})$, improving the previous best known of $O(1 + \varepsilon^{-1})$ by Abrahamsen et al. We generalise the algorithm to higher dimensions (stochastic online Euclidean traveling salesman problem), where an expected competitive ratio of $O(1)$ is achieved. This improves a previous competitive ratio of $O(\log^2 n)$ by Kalavas, Platanos and Tolias [STACS 2026].

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithm, sorting, expected analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.142

Acknowledgements I want to thank Gerth Stølting Brodal for introducing me to the problem and providing valuable feedback on drafts of the paper.

1 Introduction

We consider the online sorting problem, which is a special case of the online Euclidean traveling salesman problem, where you have to visit n points in n time slots. The points are given in an online fashion, and have to be irrevocably assigned a time slot before the next point is revealed. In the online sorting problem, the points are in one dimension, i.e., on a line. We show that if the points are distributed uniformly in the interval $[0, 1]$, we can achieve an expected competitive ratio of $O(\log n)$, that is, the length of the path is a factor $O(\log n)$ worse than the optimal path in the offline case.

Formally, in the online sorting problem, you are given an initially empty array of size n, and n items, which are real numbers, arrive in an online fashion. Each item has to be placed in the array before the next item is revealed. Once an item is placed, it cannot be moved. There can only be one item per slot. Ideally, the goal would be to end up with a sorted array. However, this is, of course, in general not possible in the online case. Instead, the goal is to minimize the sum of absolute differences of adjacent items, i.e., $\sum_{i=1}^{n-1} |A_{i+1} - A_i|$. In the offline version of this problem, where all n items are known before placing the items, an optimal assignment can be determined by sorting the items. Then the cost is the difference between the largest and smallest item. We define the competitive ratio of an online algorithm to be the ratio between the achieved cost and the optimal cost in the offline case. This means that the competitive ratio is always at least 1 and no more than $n - 1$, as the difference between any two adjacent cells is at most the difference between the largest and the smallest. This problem was introduced by Aamand, Abrahamsen, Beretta and Kleist [1], and they presented an algorithm achieving an asymptotically tight competitive ratio of $\Theta(\sqrt{n})$.

A natural question to consider is if the items are chosen randomly rather than adversarially, can one then achieve a better competitive ratio in expectation? In the stochastic variant, the items are sampled independently from the uniform distributions in the interval $[0, 1]$. Abrahamsen, Bercea, Beretta, Klausen and Kozma [2] introduced this variant and showed a slight improvement from the adversarial case to a competitive ratio of $O\big((n \log n)^{1/4}\big)$ with high probability, which Hu [8] later improved to an expected competitive ratio of $\log n \cdot 2^{O(\log^* n)}$. Hu also showed a lower bound for the expected cost of $\Omega(\log n)$. A variant of the problem allows the array to have size $\lceil (1+\varepsilon)n \rceil$, i.e., $\lceil \varepsilon n \rceil$ cells are empty when all the items have been placed. The cost is defined by ignoring the empty cells. Abrahamsen, Bercea, Beretta, Klausen and Kozma [2] introduced the variant and showed how to achieve an upper bound of expected competitive ratio of $O(1 + \varepsilon^{-1})$.

We present an algorithm that, for an array of size n and uniformly distributed inputs in $[0, 1]$, achieves an expected competitive ratio of $O(\log n)$, thus settling the complexity for the stochastic online sorting problem. For the variant with $\lceil (1+\varepsilon)n \rceil$ cells, the same algorithm without any modifications achieves an expected competitive ratio of $O(1 + \log \varepsilon^{-1})$. We also generalise the algorithm to higher dimensions where we show how the algorithm achieves an expected competitive ratio fo $O(1)$ for the stochastic online Euclidean traveling salesman problem for dimension $d \geq 2$.

1.1 Previous Work

Online sorting was originally introduced as a tool in working with online geometric packing of convex polygons in the plane [1], where tight asymptotic bounds were shown on the competitive ratio of $\Theta(\sqrt{n})$ for the adversarial case for deterministic algorithms. The algorithm presented in [1] works by segmenting the array into equal-sized buckets and the universe, assumed to be $[0, 1]$, into equal-sized disjoint spans. The main idea is to have twice the number of buckets as spans. Each bucket is either empty or has a span associated with it. If there is a bucket whose associated span covers the next item, the item is placed in the first empty cell of that bucket. If that bucket is full, the item is placed in an empty bucket, and the span of the full bucket is now associated with that bucket. If there is no empty bucket, the algorithm recurses on all the empty cells, viewed as an array through concatenation. This ensures that before recursion, at least half of the items are placed. By setting the number of buckets $\approx \sqrt{n}$, the bound is achieved. Later, a lower bound was shown for the competitive ratio of $\Omega(\sqrt{n})$ for randomized algorithms against an oblivious adversary, i.e., an adversary who does not know the coin flips of the algorithm [2].

Due to the high competitive ratio for the adversarial case, the stochastic variant, where the items are drawn independently and identically from the uniform distribution, was introduced by Abrahamsen et. al [2]. They showed a slight improvement with high probability compared to the adversarial case, with an expected competitive ratio of $O\big((n \log n)^{1/4}\big)\big)$. Their algorithm is similar to [1] and uses buckets; however, only one per span, and uses concentration bounds to show that the buckets will be filled with high probability. Additionally, they allocate an extra bucket, called the backyard, which deals with all items to overflowing buckets. Their best result comes from recursing in each bucket, which they can do as each bucket receives items uniformly random in a subinterval. Hu [8] studied the problem and showed nearly tight bounds with a lower bound of expected $\Omega(\log n)$ cost and an algorithm achieving expected competitive ratio of $\log n \cdot 2^{O(\log^* n)}$. The main innovation was to use $O(\log \log n)$ phases of buckets with geometrically decreasing number of buckets. This makes the recursion much more complicated, as they do not follow a uniform distribution. Concurrently, [9] achieved a competitive ratio of $O(\log^2 n)$ with high probability. The algorithm presented

in [9] also uses phases, but $O(\log n)$ phases instead, and instead of recursing in each bucket, the deterministic algorithm from [1] is used in each bucket. Using other parameters, the algorithm in [9] can achieve a competitive ratio of $O(\log^{3/2+\varepsilon} n)$ with high probability.

A natural attempt at solving the stochastic problem may be to try to place items according to their expected rank. For an item x, we can place it into cell $\lceil xn \rceil$, and if this cell is full, we can use linear probing to find the next free cell. This technique is used in hash tables, which admit $O(1)$ insertion in expectation [10]. The key to this result is that the array is slightly larger than the number of items. When filling the array completely, this does not work very well as the items cluster, and so some items have to be placed far away. The expected distance an item is probed, when filling an array fully, is $\Theta(\sqrt{n})$ [10]. Using a larger array with linear probing for stochastic online sorting was explored in [2], and it was shown that if the array has size $\lceil (1+\varepsilon)n \rceil$ we can use this technique along with a buffer at the end to avoid wraparound to achieve an expected competitive ratio of $O(1+\varepsilon^{-1})$ for stochastic online sorting.

The variant with the larger array has also been studied in the adversarial case. This was introduced in [2] where a deterministic competitive ratio of $2^{O\left(\sqrt{\log n}\sqrt{\log\log n+\log \varepsilon^{-1}}\right)}$ was achieved for $\varepsilon = \Omega\left(\frac{\log n}{n}\right)$ and a lower bound for the competitive ratio of $\Omega\left(\frac{\log n}{(1+\varepsilon)\log\log n}\right)$. This was recently improved significantly in [3] to a competitive ratio of $O\left(\frac{\log^2 n}{\varepsilon}\right)$ for $\varepsilon = \Omega\left(\frac{\log n}{n}\right)$ and $O\left(\frac{\log^2 n}{1+\varepsilon}\right)$ for $\varepsilon \in [\Omega(1), O(\log^2 n)]$. An implication of this is that $O(1)$ cost is achievable with $O(n \log^2 n)$ space. For $\varepsilon \leq 3$ an algorithm with competitive ratio of $(\varepsilon^{-1}\log n)^{O(\log\log n)}$ exists [12].

Another generalization of online sorting is in higher dimensions. Here, the items are points in d dimensions, and the cost is the Euclidean distance. This can be seen as an online version of the traveling salesman problem. As online sorting is the special case where $d = 1$ of online TSP, all lower bounds apply. Initially an algorithm achieving $O\left(2^d\sqrt{dn\log n}\right)$ was presented in [2]. Since then, online TSP has been improved to $O(\sqrt{n})$, thus matching the lower bound [6]. Online TSP has also been studied as a stochastic variant by [9], achieving a competitive ratio of $O(\log^2 n)$ with high probability.

1.2 Our Results

In this paper, we close the gap between the upper and lower bounds for stochastic online sorting, and thus answer an open question stated in [8]. Concretely, we present a deterministic online algorithm achieving the following theorem:

▶ **Theorem 1.** *There exists an algorithm for stochastic online sorting achieving an expected competitive ratio of $O(\log n)$.*

Our algorithm also implies an improvement for the case where the size of the array is $\lceil (1+\varepsilon)n \rceil$, where we achieve the following theorem:

▶ **Theorem 2.** *For all $\varepsilon > 0$, there exists an algorithm for stochastic online sorting with an array with $\lceil (1+\varepsilon)n \rceil$ cells achieving an expected competitive ratio of $O(1+\log \varepsilon^{-1})$.*

Note that for small additional space, say $\varepsilon \leq \frac{1}{2}$, Theorem 2 improves the previous best expected competitive ratio of [2] from $O(\varepsilon^{-1})$ to $O(\log \varepsilon^{-1})$.

We generalise the algorithm to the stochastic online Euclidean traveling salesman problem, where a better competitive ratio is possible for $d \geq 2$ dimensions.

▶ **Theorem 3.** *There exists an algorithm for the stochastic online Euclidean traveling salesman problem in d dimensions achieving an expected competitive ratio of $O(1)$ for $d \geq 2$.*

1.3 Technical overview

We reuse ideas from [1, 9]. Our algorithm is illustrated in Figure 1. Our algorithm runs in $\log_2 n - O(1)$ phases, consisting of geometrically decreasing number of cells and geometrically decreasing number of buckets, both decreasing by a factor 2. The buckets of a phase each have an associated span forming a partition of the universe. For each item, we attempt to place it into the bucket whose span contains the item. We call this bucket the *canonical bucket* for the item. If this is not possible, we instead insert it into a bucket whose span is close to the item. This incurs an extra cost, but as long as the items are not inserted too far away and it does not happen too often, the expected extra cost is low. In previous work, a lot of consideration was given to how to place items inside the buckets. We avoid this by setting the bucket capacity to $\Theta(1)$. Then the order of items inside any bucket does not matter asymptotically, so we simply place the items in the first free cell.

If only a few cells remain free in a phase, items may be inserted very far away from their canonical buckets. Therefore, we only partially fill each phase. Concretely, we fill $\frac{3}{4}$ of the available cells in each phase. This leaves us with some extra cells, which we reuse in the next phase. As there are half as many buckets, the span of each bucket in the next phase covers two buckets from the previous phase. Therefore, we merge the buckets pairwise. The buckets may then be too large. We use the same procedure as for placing items to place the excessive reused cells in the new buckets.

By considering the buckets as leaves in a binary tree, we insert the item into the bucket closest to the canonical bucket in the tree, in terms of tree edges, not in terms of their associated spans. We show that this ensures that the expected misplacement of items and cells only multiplies the cost by $O(1)$.

1.4 Paper Organization

In Section 2, we describe our algorithm achieving Theorems 1 and 2. In Section 3, we show the analysis for the case where the array has size n. In Section 4, we extend the analysis to the case where the array is larger than n. In Section 5 we show how the algorithm can be generalised to higher dimensions. In Section 6 we give an experimental evaluation of the competitive ratio achieved by the algorithm.

2 The Algorithm

Our algorithm runs in $k = \log_2 n - O(1)$ phases $1, 2, \ldots, k$ and is illustrated in Figure 1. In phase i, the algorithm places $\frac{n}{2^i}$ (ignoring roundings for now) items in 2^{k-i} buckets $1, 2, \ldots, 2^{k-1}$ with associated spans, where all buckets have equal capacity and whose spans partition the universe. The buckets of phase i have a total capacity of $\frac{4}{3} \cdot \frac{n}{2^i}$ cells. A cell containing no item is considered *free*. A cell containing an item is considered *filled*. Initially, all cells are free. From now on, we write all roundings explicitly. The output array A is partitioned into subarrays $C_1, C_2, \ldots, C_k$ contiguously from left to right. Items in phase i are placed in C_i and free cells in $C_1, C_2, \ldots, C_{i-1}$ at the end of phase $i-1$. The capacity of C_i is set such that the number of available cells at the beginning of phase i, including the free cells from phase $i-1$, is as precisely described below.

Before we can discuss how to allocate buckets and place items, we introduce the *universe tree*, which is a hierarchical segmentation of $[0, 1]$, where a node spans an interval, such that each level of the tree forms a partition of $[0, 1]$, with each node having equal length span.

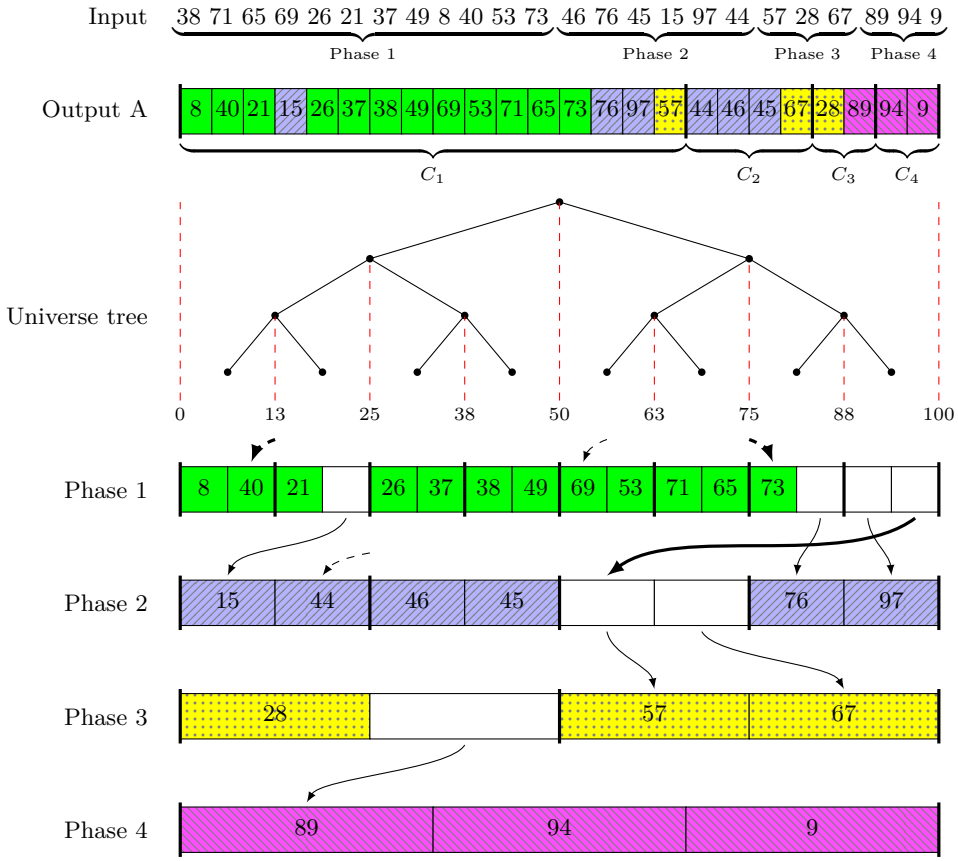


Figure 1 An example of the algorithm with $n = 24$ items and $k = 4$ phases. We use the distribution of integers from 0 to 99 instead of $[0, 1]$, for example's sake only. The colors indicate the phase an item was placed in. For each node in the universe tree, its span is the interval between the two dashed lines on either side. In the first phase, the buckets correspond to the lowest level in the universe tree, thus spanning intervals $[0, 13), [13, 25), \ldots, [88, 100)$. A dashed arrow above an item indicates that the item was not placed in its canonical bucket. The thickness indicates how far away in tree distance it is placed in terms of the number of levels in the tree traversed. For example, 44 only had to go up one level in the tree, being placed in the $[0, 25)$ bucket instead of its canonical $[25, 50)$ bucket, and thus is thin. However, 40 had to go up two levels in the tree, being placed in the $[0, 13)$ bucket instead of its canonical $[38, 50)$ bucket, and is therefore thick (we allow 40 to be placed in any bucket in the span $[0, 25)$, not necessarily the closest to $[38, 50)$, with a free cell). The arrows between phases indicate the cells that were free at the end of a phase and, therefore, reused in the next phase. The thickness indicates how far from its canonical bucket it was placed. For example, the last cell in phase 1 is not placed in its canonical bucket, which is $[75, 100)$, but in its neighboring bucket, which is $[50, 75)$ and is therefore thick. The remaining cells are taken from C_i from left to right.

The tree has k levels and is a perfect binary tree. In a level, the nodes have "increasing" spans from left to right. The buckets in a phase correspond to a level in the universe tree. In phase 1, the buckets correspond to the lowest level.

In phase 1 we split C_1 into 2^{k-1} buckets of equal capacity. For an item x, we call the bucket whose span contains x the *canonical* bucket. This is bucket $\lceil 2^{k-1}x \rceil$ from the left[1]. For each item in phase 1, if there is a free cell in the canonical bucket, the item is placed in an arbitrary free cell, which then becomes filled. Otherwise, we find a bucket that has the shortest distance in terms of tree edges in the universe tree from the canonical bucket. In case there are multiple such buckets, an arbitrary one is chosen.

After placing the items of a phase, we still have free cells that are reused in the next phase. The next phase has half as many buckets, corresponding to going up one level in the universe tree. All new buckets have equal capacity. It would be natural to reuse a cell in the bucket in the parent node, as its span covers the bucket it is currently in. This, however, may lead to some buckets being too large. We solve this exactly the same way as we solved placing the items. Each cell has a canonical bucket that it ideally would be a part of. If the bucket already has the number of cells needed, we find a closest bucket in the universe tree that still reuqires more free cells. After having allocated all free cells at the end of phase $i-1$ to the buckets of phase i, the buckets are filled with fresh cells from C_i, from left to right. Note that a cell may be reused in multiple phases and, in the extreme case, in all phases.

For the remaining phases, we place items just as we did in phase 1. In phase i, we calculate the canonical bucket as $\lceil 2^{k-i}x \rceil$. In the last phase, we only have one bucket and all remaining items are placed in this bucket.

As each phase reuses free cells from the previous phase, we set the capacity of C_i appropriately. Let s_i denote the ideal capacity of C_i (not necessarily an integer). For the first phase, we have no cells to reuse and therefore, $s_1 = \frac{4}{3} \cdot \frac{n}{2} = \frac{2}{3} \cdot n$. For phase $2 \leq i < k$, a forth of the cells available in the last phase are reused, therefore $s_i = \frac{4}{3} \cdot \frac{n}{2^i} - \frac{1}{4} \cdot \frac{4}{3} \cdot \frac{n}{2^{i-1}} = \frac{2}{3} \cdot \frac{n}{2^i}$. For the last phase we use all remaining cells therefore $s_k = n - \sum_{i=1}^{k-1} s_i = n - \frac{2}{3} \cdot n - \sum_{i=2}^{k-1} \frac{2}{3} \cdot \frac{n}{2^i} = n\left(\frac{1}{3} - \frac{2}{3}\sum_{i=2}^{k-1} \frac{1}{2^i}\right) = n\left(\frac{1}{3} - \frac{2}{3}\left(\frac{1}{2} - \frac{1}{2^{k-1}}\right)\right) = \frac{2}{3} \cdot \frac{n}{2^{k-1}}$.

Unfortunately, s_i may not be an integer. Imagine allocating the cells anyway by cutting some cells. A cell that is cut between two subarrays is allocated to the latter. Since each C_i gains less than one cell from C_{i-1} and gives less than one cell to C_{i+1}, the actual capacity of C_i is s_i rounded either up or down. When allocating buckets, they also may not all be able to have the same capacity. Therefore, we calculate the average bucket capacity for the phase and make some that are rounded down, and some rounded up. We decide which buckets have which capacity arbitrarily.

Concretely, by setting $k = \lfloor \log_2 n - 5 \rfloor$, the capacity of the buckets is $\Theta(1)$. The exact number of items placed in phase $i < k$ is $\left\lfloor \frac{3}{4} m_i \right\rfloor$, where m_i is the number of free cells at the beginning of phase i.

3 Analysis

In the analysis, we place temporary items in the cells. At the start of phase i, we place temporary items in all cells in all buckets in phase i that equal the minimum of the bucket's span. When real items are placed by the algorithm, we replace these temporary items with the real items in the analysis. The idea is that if we replace an item by another with absolute

[1] In case $x = 0$, which happens with probability 0, the algorithm chooses bucket 1.

difference d, the increase in cost is at most $2d$. For the free cells reused from the previous phase, we replace the temporary items with new temporary items of absolute difference d, which we also count as a cost of $2d$. We bound the expected cost of our analysis, which is an upper bound of the cost of the algorithm. Cost comes from the following four sources:

① The cost between different C_i.

② The cost for placing temporary items in C_i.

③ The cost for placing real items.

④ The cost for replacing temporary items when reusing cells from $C_1, \ldots, C_{i-1}$.

The difference between two items in adjacent cells is at most 1. For $2 \leq i \leq k$, there is one boundary between C_{i-1} and C_i. As there are $k-1$ such values of i, the cost ① is at most $k-1$.

Within each phase, we place temporary items as the minimum value of the bucket's span in each of the bucket's cells. As the temporary items in C_i are allocated into buckets left to right, their temporary items are ordered ascendingly, and thus the cost ② is at most 1 per phase, i.e., k for all phases.

For placing a real item, what matters is the difference from the temporary value in the cell. The further up the universe tree the placement goes, the larger that difference is. Consider the highest node on the path from the canonical bucket to the bucket in which the item was placed in. We call this the *placing* node. The span of this node spans both the item and the temporary item in the cell. Therefore, this span is an upper bound on the difference. For each phase individually, consider marking the placing node for each item. Then the cost is bounded by twice the sum over the products of the number of marks on a node and the length of its span. Consider all nodes that are marked, but where no ancestor is marked. We say such a node is *top-marked*. Every item placed in the subtree of a top-marked node has a canonical bucket in this subtree, so multiplying the length of the span by the number of cells in this subtree is an upper bound on the cost for all those items. All items are also in a subtree of a node that is top-marked. A node can only be top-marked if a child overflows based only on the items that have their canonical buckets in the child's subtree. We call this type of overflow *local overflow*. This is because no ancestor is marked, no items with a canonical bucket outside of the subtree are placed in the subtree, and one of the children must overflow first, thus not based on any items whose canonical bucket is in the other child's subtree. By counting all nodes with at least one overflowing child, we upper bound the cost. We do this by upper bounding the probability of overflow and then calculating the expected cost for a node. We use linearity of expectation over all nodes and phases to get an upper bound on the expected cost ③.

As reusing cells uses the same procedure as placing items, the analysis for ④ is the same as for ③, except that calculating the overflow probability is slightly different.

The central argument for the analysis of ③ and ④ is that when we go up one level into the tree, the cost for a node quadruples as the length of the span is twice as long and the number of cells in its subtree is twice as many. On the other hand, there are only half as many nodes. We show that the probability of overflow geometrically decreases by a factor of four using a Chernoff bound (Lemma 7 and Lemma 8 below), thus leading the expected cost per level to decrease by a factor of two. As the buckets have a capacity bounded by a constant (Lemma 4), the cost is at most constant for the level corresponding to the buckets. As the cost geometrically decreases with increasing level in the tree, the cost overall is $O(1)$ per phase.

We now give a detailed analysis. We first show that buckets have a capacity bounded by a constant. Concretely, we show that all buckets have capacity B or $B+1$, where $B = \lfloor \frac{n}{3 \cdot 2^{k-2}} \rfloor$, and that B is a constant.[2]

▶ **Lemma 4** (B is a constant). *$42 \leq B \leq 85$ and therefore $B = \Theta(1)$.*

Proof. From the definition of $B = \lfloor \frac{n}{3 \cdot 2^{k-2}} \rfloor$ and $k = \lfloor \log_2 n - 5 \rfloor$:

$$B = \left\lfloor \frac{n}{3 \cdot 2^{\lfloor \log_2 n - 7 \rfloor}} \right\rfloor \geq \left\lfloor \frac{n}{3 \cdot 2^{\log_2 n - 7}} \right\rfloor \geq 42 \, .$$
$$B = \left\lfloor \frac{n}{3 \cdot 2^{\lfloor \log_2 n - 7 \rfloor}} \right\rfloor \leq \left\lfloor \frac{n}{3 \cdot 2^{\log_2 n - 8}} \right\rfloor \leq 85 \, .$$ ◀

▶ **Lemma 5** (Cells in a phase). *Let L_i be the cells available in phase i. For $1 \leq i < k$, the number of cells available in phase i is bounded by $2^{k-i} \cdot B \leq |L_i| \leq 2^{k-i} \cdot (B+1)$.*

Proof. The proof is by induction. For phase 1 we have $L_1 = C_1$, and by definition of C_1, $\lfloor \frac{2}{3} \cdot n \rfloor \leq |C_1| \leq \lceil \frac{2}{3} \cdot n \rceil$. Note that $2^{k-1} \cdot B = 2^{k-1} \lfloor \frac{n}{3 \cdot 2^{k-2}} \rfloor \leq \lfloor 2^{k-1} \frac{n}{3 \cdot 2^{k-2}} \rfloor = \lfloor \frac{2}{3} \cdot n \rfloor$ for $k \geq 1$. Similarly, $2^{k-1} \cdot (B+1) = 2^{k-1} \left(\lfloor \frac{n}{3 \cdot 2^{k-2}} \rfloor + 1 \right) \geq 2^{k-1} \lceil \frac{n}{3 \cdot 2^{k-2}} \rceil \geq \lceil \frac{2}{3} \cdot n \rceil$ for $k \geq 1$. Therefore,

$$2^{k-1} \cdot B \leq |L_1| \leq 2^{k-1} \cdot (B+1) \, . \tag{1}$$

For the induction step, recall that in phase $i-1$, $|L_{i-1}|$ cells are available and $\lfloor \frac{3}{4} |L_{i-1}| \rfloor$ items are placed. Therefore, there are $\lceil \frac{1}{4} |L_{i-1}| \rceil$ free cells that are reused in phase i. Therefore, $|L_i| = \lceil \frac{1}{4} |L_{i-1}| \rceil + |C_i|$. By definition of C_i for $2 \leq i \leq k-1$, $\lfloor \frac{2}{3} \cdot \frac{n}{2^i} \rfloor \leq |C_i| \leq \lceil \frac{2}{3} \cdot \frac{n}{2^i} \rceil$ we have, similarly to Equation 1,

$$2^{k-i-1} \cdot B \leq |C_i| \leq 2^{k-i-1} \cdot (B+1) \, . \tag{2}$$

By the induction hypothesis

$$2^{k-i+1} \cdot B \leq |L_{i-1}| \leq 2^{k-i+1} \cdot (B+1) \tag{3}$$
$$2^{k-i-1} \cdot B \leq \tfrac{1}{4} |L_{i-1}| \leq 2^{k-i-1} \cdot (B+1) \tag{4}$$
$$2^{k-i-1} \cdot B \leq \lceil \tfrac{1}{4} |L_{i-1}| \rceil \leq 2^{k-i-1} \cdot (B+1) \, . \tag{5}$$

As $i \leq k-1$, both bounds are integers, and therefore, after rounding, the inequalities still hold. Combining Equation 2 and Equation 5 completes the proof. ◀

▶ **Corollary 6** (Bucket capacity). *Every bucket in phase $i < k$ has capacity B or $B+1$.*

Proof. Following Lemma 5 the average bucket capacity for a phase is between B and $B+1$. As there is at least one bucket smaller than or equal to the average and at least one larger than or equal to, no bucket can be further from the average than 1. ◀

We now show that for both placing items and reusing cells, the probability that a node overflows is geometrically decreasing when going up the tree.

▶ **Lemma 7** (Probability of overflow when placing items). *For a node $j \geq 0$ levels above the buckets in the universe tree, the node locally overflows with probability at most $\left(\frac{1}{4}\right)^j$ when placing items.*

[2] The parameter k could be chosen such that $B = 1$, but results in a slightly more complicated analysis. For the evaluation, we do use such parameters.

Proof. Recall that local overflow means that the bucket overflows only based on items with a canonical bucket in the subtree of the node. We therefore bound the probability that too many items were sampled in this node's span during the current phase.

We use the following Chernoff bound [11, Corollary 4.6] for a binomial distribution X with mean μ:

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2\mu}{3}} .$$

As each bucket has capacity at least B, the node never locally overflows if at most $2^j B$ items are sampled in this node's span. Therefore, we bound the probability that more were sampled in the phase. Since the expected number of items placed per bucket in a phase is between $\frac{3}{4}B$ and $\frac{3}{4}(B+1)$, we know that $2^j \cdot \frac{3}{4}B \leq \mu \leq 2^j \cdot \frac{3}{4}(B+1)$. By setting $\delta = \frac{4}{3}\frac{B}{B+1} - 1 \geq \frac{4}{3} \cdot \frac{42}{43} - 1 = \frac{39}{129}$ using $B \geq 42$ from Lemma 4 we get:

$$\Pr[X > 2^j B] \;\leq\; \Pr[X \geq (1+\delta)\mu] \;\leq\; e^{-\frac{39^2 \cdot 2^j \cdot 3 \cdot B}{129^2 \cdot 3 \cdot 4}} \;\leq\; e^{-\frac{39^2 \cdot 3 \cdot 42}{129^2 \cdot 3 \cdot 4} 2^j} \;\leq\; \left(\tfrac{1}{4}\right)^j . \qquad \blacktriangleleft$$

▶ **Lemma 8** (Probability of overflow when reusing cells). *For a node $j \geq 0$ levels above the buckets in the universe tree, the node locally overflows with probability at most $\left(\frac{1}{4}\right)^j$ when reusing cells.*

Proof. Recall that local overflow means that the bucket overflows only based on items with a canonical bucket in the subtree of the node. For a node to locally overflow when reusing cells, this means that there were many free cells in its span in the previous phase. This can be seen as an underflow in the previous phase. We therefore bound the probability that too few items were sampled in the node's span during the previous phase.

We use the following Chernoff bound [11, Theorem 4.5] for a binomial distribution X with mean μ:

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2\mu}{2}} .$$

As each bucket has capacity at least B, if there are fewer than $2^j B$ free cells in this subtree, this node never locally overflows. This means that if more than $2^j(\frac{B}{2}+1)$ items were sampled in this node's span in the previous phase, this node never locally overflows. Therefore, we bound the probability that fewer items were sampled in this node's span. The expected number of items per bucket in the previous phase is at least $\frac{3}{4}B$; therefore, $\mu \geq 2^{j+1}\frac{3}{4}B$. By setting $\delta = \frac{1}{3} - \frac{4}{3B} \geq \frac{19}{63}$, using $B \geq 42$ from Lemma 4, we get,

$$\Pr\left[X \leq 2^j\left(\tfrac{B}{2}+1\right)\right] \;\leq\; \Pr[X \leq (1-\delta)\mu] \;\leq\; e^{-\frac{19^2 \cdot 2^{j+1} \cdot 3 \cdot B}{63^2 \cdot 2 \cdot 4}} \;\leq\; e^{-\frac{19^2 \cdot 3 \cdot 42}{63^2 \cdot 2 \cdot 4} 2^{j+1}} \;\leq\; \left(\tfrac{1}{4}\right)^j . \blacktriangleleft$$

We can now bound the expected cost per phase.

▶ **Lemma 9** (Expected cost per phase). *The expected cost is $O(1)$ per phase.*

Proof. As discussed earlier the cost ① and ② is $O(1)$ per phase.

For phase $j < k$, consider a node j levels above the buckets. By a union bound, Lemma 7 and Lemma 8, the probability that a node is top-placing is at most twice the probability that a child locally overflows, that is, a node in level $j-1$ above the buckets. The probability is thus $2 \cdot \left(\frac{1}{4}\right)^{j-1}$, for $j \geq 1$. The formula also holds for $j = 0$, as it claims a probability greater than 1. Let ℓ be the length of the span of this node. The cost this node incurs if it is top-placing is at most the number of cells in its subtree times 2ℓ. As there are

2^j buckets in its subtree and each has at most $B+1$ cells, the expected cost is at most $2^j(B+1)\cdot 2\cdot\left(\frac{1}{4}\right)^{j-1}\ell = \frac{1}{2^j}8(B+1)\ell$. Consider all nodes j levels above the buckets. The sum of their span-lengths is 1; therefore, the cost of all nodes in this level using linearity of expectation is at most $\frac{1}{2^j}8(B+1)$. Summing over all levels forms a geometrically decreasing sum such that the expected cost ③ and ④ is $O(1)$.

For phase k using Lemma 4 and Lemma 5, the number of cells in phase k is $|C_k| + \left\lceil\frac{1}{4}|L_{k-1}|\right\rceil \le |C_k| + \lceil|L_{k-1}|\rceil \le \left\lceil\frac{2}{3}\cdot\frac{n}{2^{k-1}}\right\rceil + 2(B+1) = O(1)$. As each item can at most differ by 1 compared to the temporary item in both the case of placing items and reusing free cells, the cost ③ and ④ is $O(1)$. ◀

We show that the cost is $O(\log n)$. But it is actually the competitive ratio that is the goal. We show that those are essentially the same thing.

▶ **Lemma 10.** *An algorithm that achieves expected cost c for a uniform input distribution achieves expected competitive ratio at most $2c + o(1)$.*

Proof. Let X be a continuous random variable denoting the cost of the algorithm and Y a continuous random variable denoting the optimal cost in the offline case. Note that X and Y are dependent variables. Let x_{min} and x_{max} be random variables for the minimum and maximum items, respectively. Note that $Y = x_{max} - x_{min}$. The competitive ratio can never be more than $n-1$, as the difference between two adjacent items cannot be more than $Y = x_{max} - x_{min}$. Below we use that $\Pr[x_{min} \ge \frac{1}{4}] = \Pr[x_{max} \le \frac{3}{4}] = \left(\frac{3}{4}\right)^n$. By the law of the unconscious statistician [13, Equation 5.19], the expected competitive ratio can be calculated as

$$\begin{aligned}
\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\tfrac{x}{y}f_{X,Y}(x,y)dydx &= \int_{-\infty}^{\infty}\int_{\frac{1}{2}}^{\infty}\tfrac{x}{y}f_{X,Y}(x,y)dydx + \int_{-\infty}^{\infty}\int_{-\infty}^{\frac{1}{2}}\tfrac{x}{y}f_{X,Y}(x,y)dydx\\
&\le 2\int_{-\infty}^{\infty}x\int_{\frac{1}{2}}^{\infty}f_{X,Y}(x,y)dydx + n\int_{-\infty}^{\frac{1}{2}}\int_{-\infty}^{\infty}f_{X,Y}(x,y)dxdy\\
&\le 2\int_{-\infty}^{\infty}x\int_{-\infty}^{\infty}f_{X,Y}(x,y)dydx + n\int_{-\infty}^{\frac{1}{2}}f_Y(y)dy\\
&= 2\int_{-\infty}^{\infty}xf_X(x)dx + n\Pr\left[Y\le\tfrac{1}{2}\right]\\
&\le 2c + n\Pr\left[x_{min}\ge\tfrac{1}{4}\ \vee\ x_{max}\le\tfrac{3}{4}\right]\\
&\le 2c + n\left(\Pr\left[x_{min}\ge\tfrac{1}{4}\right] + \Pr\left[x_{max}\le\tfrac{3}{4}\right]\right)\\
&= 2c + n\cdot 2\left(\tfrac{3}{4}\right)^n\\
&= 2c + o(1)\ .
\end{aligned}$$

◀

We can now prove the first theorem of the paper.

▶ **Theorem 1.** *There exists an algorithm for stochastic online sorting achieving an expected competitive ratio of $O(\log n)$.*

Proof. As each phase has expected cost $O(1)$ by Lemma 9, and there are $\log_2 n - O(1)$ phases, the expected cost is $O(\log n)$. Lemma 10 completes the proof. ◀

4 Larger Array

We now consider the case where the array may be larger than n. The algorithm is exactly the same except that it terminates early when there are no more items. Note that ε need not be known ahead of time; only the size of the output array A is needed.

▶ **Theorem 2.** *For all $\varepsilon > 0$, there exists an algorithm for stochastic online sorting with an array with $\lceil (1+\varepsilon)n \rceil$ cells achieving an expected competitive ratio of $O(1 + \log \varepsilon^{-1})$.*

Proof. Let $m = \lceil (1+\varepsilon)n \rceil$. The expected cost per phase is $O(1)$ by Lemma 9. Therefore, we only need to count the number of phases before all items are placed to bound the expected cost. If $\varepsilon \geq 2$, then $|C_1| \geq 2n$ and the number of items that can be placed in the first phase is at least $\lfloor \frac{3}{4} \cdot 2n \rfloor \geq n$. There will therefore be one phase. Otherwise, using Lemma 5, Lemma 4 and assuming $t < k$, the number of items that can be placed after t phases is

$$\begin{aligned}
\sum_{i=1}^{t} |C_i| - \left\lceil \tfrac{|L_t|}{4} \right\rceil &\geq \left\lfloor \tfrac{m}{3} \left(2 + \sum_{i=2}^{t} \tfrac{1}{2^{i-1}} \right) \right\rfloor - \left\lceil 2^{k-t-2}(B+1) \right\rceil \\
&\geq \tfrac{m}{3} \left(2 + \sum_{i=2}^{t} \tfrac{1}{2^{i-1}} \right) - 2^{k-t-2}(B+1) - 2 \\
&\geq \tfrac{m}{3} \left(3 - \tfrac{1}{2^{t-1}} \right) - 2^{k-t-2}(B+5) && t < k \Rightarrow 2^{k-t-2} \geq \tfrac{1}{2} \\
&\geq m - \tfrac{m}{3 \cdot 2^{t-1}} - 2^{k-t-1} B && B > 42 \text{ by Lemma 4} \\
&\geq m - \tfrac{m}{3 \cdot 2^{t-1}} - 2^{k-t-1} \tfrac{m}{3 \cdot 2^{k-2}} \\
&= m - \tfrac{m}{3 \cdot 2^{t-1}} - \tfrac{m}{3 \cdot 2^{t-1}} \\
&= m - \tfrac{m}{3 \cdot 2^{t-2}} \, .
\end{aligned}$$

We solve for when $n = \frac{m}{1+\varepsilon}$ items are placed:

$$\begin{aligned}
m - \tfrac{m}{3 \cdot 2^{t-2}} &= \tfrac{m}{1+\varepsilon} \\
1 - \tfrac{1}{3 \cdot 2^{t-2}} &= \tfrac{1}{1+\varepsilon} \\
-\tfrac{1}{3 \cdot 2^{t-2}} &= -\tfrac{\varepsilon}{1+\varepsilon} \\
t - 2 + \log_2 3 &= \log_2(1+\varepsilon) - \log_2 \varepsilon \, .
\end{aligned}$$

Therefore, $t = O\left(1 + \log \varepsilon^{-1}\right)$, where $\varepsilon < 2$. Lemma 10 completes the proof. ◀

5 Higher Dimensions

In this section we sketch how to generalise the algorithm to higher dimensions, i.e., the stochistic online Euclidean traveling salesman problem. The algorithm is the same except that we need to define how to assign cannonical buckets to points in the d-dimensional unit cube, $[0,1]^d$. We therefore define a new universe tree. The root consists of the entire universe, the d-dimensional unit cube, $[0,1]^d$. The two children of a node split the parent halfway along a dimension. Which dimension is split along is chosen cyclically by level in the universe tree. This is the same way as a k-d tree splits the universe [5].

▶ **Theorem 3.** *There exists an algorithm for the stochastic online Euclidean traveling salesman problem in d dimensions achieving an expected competitive ratio of $O(1)$ for $d \geq 2$.*

Proof. The analysis is done in the same way as in Section 3. In one dimension, we had the length of the span as an upper bound on how far apart items can be. We now have a d-dimensional rectangle instead. The furthest two points can be from each other is the distance between two opposite corners. For a node j levels below the root in the universe tree, the volume is 2^{-j}. As each side length are roughly equally long, they may have a difference of a factor of two; the side lengths are approximately $2^{-\frac{j}{d}}$ long. Using the Pythagorean theorem two opposite corners have distance $O(\sqrt{d} \cdot 2^{-\frac{j}{d}})$. As in Lemma 9 we need the sum over the length of all spans in a level of the universe tree, which is $O\left(\sqrt{d} \cdot \left(2^{1-\frac{1}{d}}\right)^j\right)$.

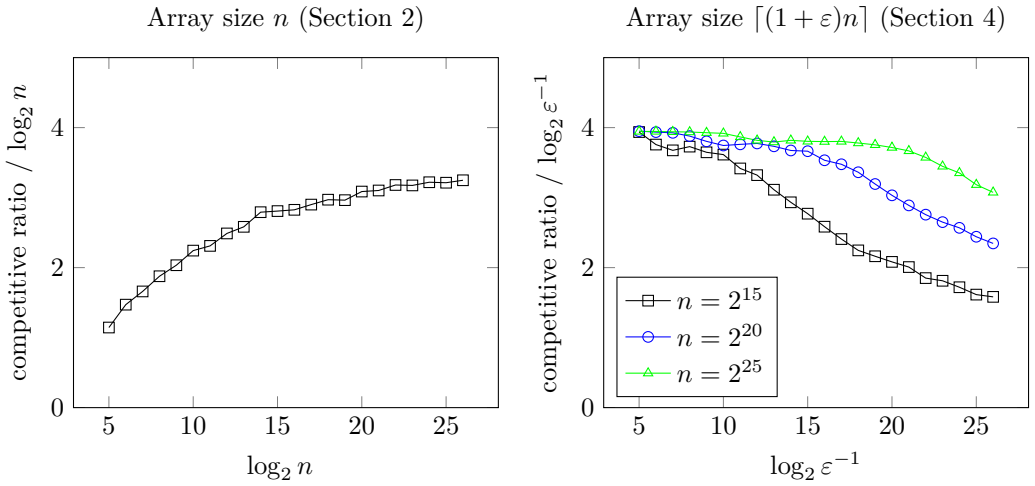
For phase i the buckets are on level $k-i$ and therefore the cost is $O\left(\sqrt{d}\cdot\left(2^{1-\frac{1}{d}}\right)^{k-i}\right)$. Note that $2^{1-\frac{1}{d}} \geq \sqrt{2}$ for $d \geq 2$. Therefore, the sum over all phases forms a geometrically decreasing sum.

$$\sum_{i=1}^{k} O\left(\sqrt{d}\cdot\left(2^{1-\frac{1}{d}}\right)^{k-i}\right) = O\left(\sqrt{d}\cdot\left(2^{1-\frac{1}{d}}\right)^{k}\right) = O\left(\sqrt{d}\cdot n^{1-\frac{1}{d}}\right).$$

Comparing this with bounds on the expected optimal cost of $\Theta(\sqrt{d}\cdot n^{1-\frac{1}{d}})$ for a tour of uniformly random generated points in the d-dimensional unit cube [4, 14] results in an expected competitive ratio of $O(1)$. More details can be found in [7]. ◀

6 Experimental Evaluation

We experimentally evaluated the algorithm from Section 2 and Section 4 with different n and ε.



For the case of $\varepsilon = 0$ (left), we see that the constant factor seems to converge towards 3.5. For the case of $\varepsilon > 0$ (right), it seems that the constant is around 4, except when $n < \varepsilon^{-1}$, where $\log n$ is less than $\log \varepsilon^{-1}$ and therefore the cost decreases compared to $\log \varepsilon^{-1}$.

7 Conclusion

In this paper, we showed that for the stochastic online sorting problem there is an upper bound for the expected competitive ratio of $O(\log n)$, thus matching the lower bound [8], and $O(\log \varepsilon^{-1})$ for the case with the larger array. The implicit constant hidden in our analysis of Theorem 1 is around 6000. As seen in our experimental evaluation, the true competitive ratio appears to be around $3.5 \log n$. Our algorithm has no high probability guarantees. We leave it as an open question if a competitive ratio of $O(\log n)$ is also possible with high probability.

For the case of the larger array, the competitive ratio appears to be around $4 \log \varepsilon^{-1}$, when $\varepsilon^{-1} < n$. There is no lower bound for the case with the larger array. It would be interesting to either find a matching lower bound or an even better algorithm.

We show that the algorithm can be generalised for the stochastic online Euclidean traveling salesman problem, where the competitive ratio improves to $O(1)$ for $d \geq 2$ dimensions.

In the worst case, our algorithm has a competitive ratio of $\Theta(n)$. This can be achieved by sending 0 until the left half of the universe tree has no more cells, then alternating 0 and 1. We leave it an open question if there exists an algorithm that is optimal both in the stochastic case and the adversarial case simultaneously.

We only consider the uniform distribution. For other distribution the algorithm presented here can be run by adjusting the spans according to the distribution, and it seems that the competitive ratio is at least close to $O(\log n)$, when we tested with a normal distribution; however, we were not able to prove it.

References

1 Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1806–1833. SIAM, 2023. doi:10.1137/1.9781611977554.CH69.

2 Mikkel Abrahamsen, Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, and László Kozma. Online sorting and online TSP: randomized, stochastic, and high-dimensional. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, Royal Holloway, London, United Kingdom, September 2-4, 2024*, LIPIcs, pages 5:1–5:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ESA.2024.5.

3 Yossi Azar, Debmalya Panigrahi, and Or Vardi. Nearly tight bounds for the online sorting problem. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 6642–6658. SIAM, 2026. doi:10.1137/1.9781611978971.237.

4 Jillian Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959. doi:10.1017/S0305004100034095.

5 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. doi:10.1145/361002.361007.

6 Christian Bertram. Online metric TSP. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *33rd Annual European Symposium on Algorithms, ESA 2025, Warsaw, Poland, September 15-17, 2025*, LIPIcs, pages 80:1–80:9. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.80.

7 Daniel Anker Hermansen. Stochastic online euclidean tsp. Master's thesis, Aarhus University, 2026. To appear on ArXiv.

8 Yang Hu. Nearly optimal bounds for stochastic online sorting. In Kasper Green Larsen and Barna Saha, editors, *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, BC, Canada, January 11-14, 2026*, pages 4969–4995. SIAM, 2026. doi:10.1137/1.9781611978971.180.

9 Andreas Kalavas, Charalampos Platanos, and Thanos Tolias. A polylogarithmic competitive algorithm for stochastic online sorting and TSP. In Meena Mahajan, Florin Manea, Annabelle McIver, and Kim Thang Nguyen, editors, *43rd International Symposium on Theoretical Aspects of Computer Science, STACS 2026, Grenoble, France, March 9-13, 2026*, LIPIcs, pages 58:1–58:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.STACS.2026.58.

10 Don Knuth. Notes on "open" addressing, 1963. URL: https://jeffe.cs.illinois.edu/teaching/datastructures/2011/notes/knuth-OALP.pdf.

11 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CBO9780511813603.

12 Jubayer Nirjhor and Nicole Wein. Improved online sorting. In Jannik Matuschke and José Verschae, editors, *Approximation and Online Algorithms - 23rd International Workshop, WAOA 2025, Warsaw, Poland, September 18-19, 2025, Proceedings*, Lecture Notes in Computer Science, pages 187–197. Springer, 2025. doi:10.1007/978-3-032-06706-7_13.

13 Hossein Pishro-Nik. *Introduction to probability, statistics, and random processes*. Kappa Research LLC, 2014. URL: https://www.probabilitycourse.com.

14 J. Michael Steele. *Probability Theory and Combinatorial Optimization*. Society for Industrial and Applied Mathematics, 1997. doi:10.1137/1.9781611970029.