A Polylogarithmic Competitive Algorithm for Stochastic Online Sorting and TSP

Thanos Tolias
NTUA and Archimedes/Athena RC

Joint work with Andreas Kalavas and Charalampos Platanos

Online Sorting

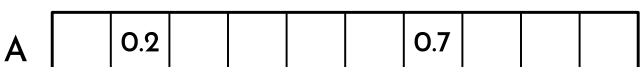
There is an array A with n initially empty cells $A[1], A[2], \ldots, A[n]$.

At each timestep t, a real number $x(t)$ in $[0, 1]$ arrives, and **we must irrevocably place the number into a cell of the array.**

Online Sorting

There is an array A with n initially empty cells $A[1], A[2], \ldots, A[n]$.

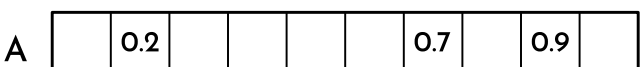
At each timestep t, a real number $x(t)$ in $[0, 1]$ arrives, and **we must irrevocably place the number into a cell of the array.**



Online Sorting

There is an array A with n initially empty cells $A[1], A[2], \ldots, A[n]$.

At each timestep t, a real number $x(t)$ in $[0, 1]$ arrives, and **we must irrevocably place the number into a cell of the array.**



Online Sorting

The objective is to minimize the sum of absolute differences between the numbers occupying adjacent array cells.

$$\text{Cost}(A) = \sum_{i=2}^{n} |A[i] - A[i-1]|$$

Online Sorting

The objective is to minimize the sum of absolute differences between the numbers occupying adjacent array cells.

$$\text{Cost}(A) = \sum_{i=2}^{n} |A[i] - A[i-1]|$$

A	0.1	0.2	0.4	0.3	0.3	0.5	0.7	0.6	0.9	1.0

Cost(A) = 1.3

OPT	0.1	0.2	0.3	0.3	0.4	0.5	0.6	0.7	0.9	1.0

Cost(OPT) = 0.9

Previous Work

Adversarial Online Sorting: Numbers are selected by an adaptive adversary.

- Aamand et al. [SODA23] presented an $O(\sqrt{n})$-competitive algorithm.

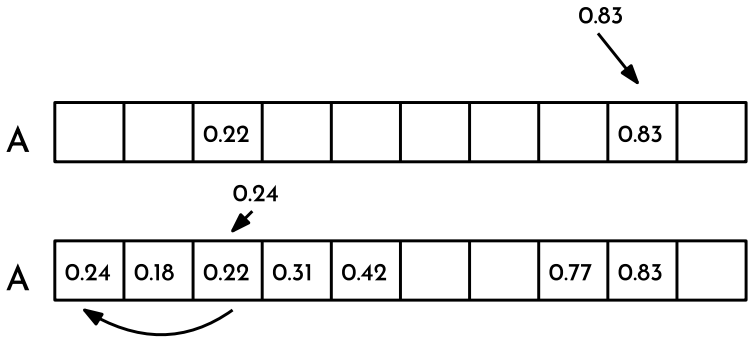
Stochastic Online Sorting: Numbers are drawn i.i.d. from $U(0, 1)$.

- Abrahamsen et al. [ESA24] presented an $\tilde{O}(n^{1/4})$-competitive algorithm with high probability.
- Recently, independently to our work, Hu [SODA26] provided a $\log n \cdot 2^{O(\log^* n)}$-competitive on expectation algorithm and a logarithmic lower bound.

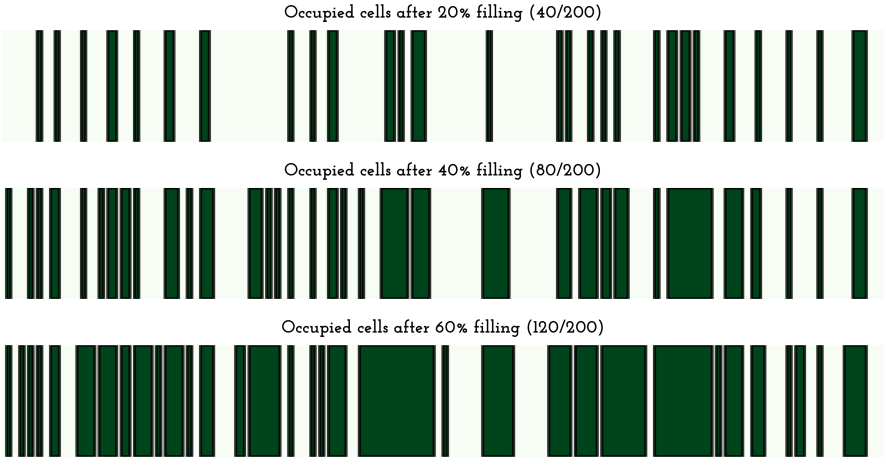
A Naive Algorithm

For each timestep $t \in [n]$:

- Try to place $x(t)$ to its percentile cell $A[\lceil x(t) \cdot n \rceil]$
- If occupied, place $x(t)$ to the nearest available cell.



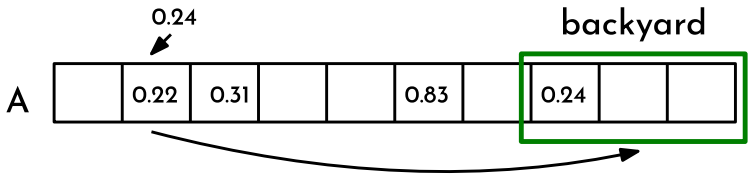
Why Naive Placement Fails



Occupied cells (green) at different points of the run.

Backyard

What if instead of placing the element to the nearest available cell we kept a final part of the array to handle conflicts? Lets call this final part "backyard".



Backyard

What if instead of placing the element to the nearest available cell we kept a final part of the array to handle conflicts? Lets call this final part "backyard".

The main part of the array is now perfectly sorted and we can place elements on the backyard using the adversarial algorithm.

Problem: The backyard is too big.

Blocks

Idea: Treat numbers that are close to each other as being the same.

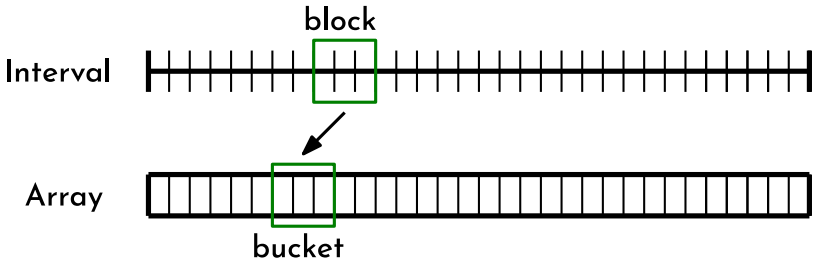
Partition $[0, 1]$ into subintervals called "blocks". Numbers in the same block are considered close to each other.

If an algorithm managed to place the numbers so that the corresponding blocks were perfectly sorted, then

$$\text{Cost}(A) \leq n \cdot (\text{length of a block})$$

Buckets

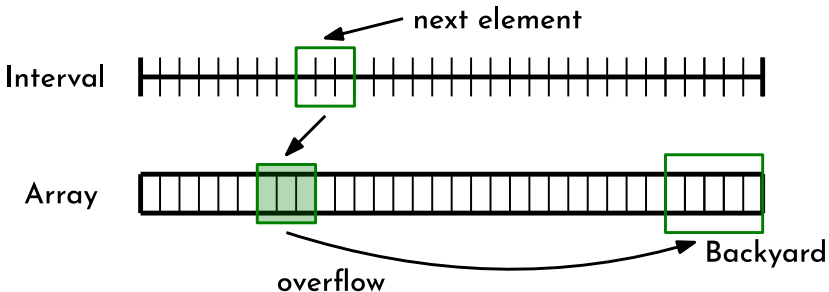
Map each block to a continuous chunk of the array, called "bucket".



The number of elements drawn in each block is a random variable. How big should the corresponding buckets be?

Backyard

Approach: Make buckets small enough so they are filled with high probability and let backyard handle the overflows.



Cost of the Algorithm

Using the adversarial algorithm of Aamand et al. on a single bucket with "valid" input:

$$\sqrt{\text{bucket size}} \cdot (\text{block length})$$

Since the blocks form a partition of [0,1] and the buckets have similar size, the total cost for all buckets is:

$$\sqrt{\text{bucket size}}$$

Finally, the backyard receives numbers from all [0,1] in the worst case and has cost:

$$\sqrt{\text{backyard size}} \cdot (1 - 0)$$

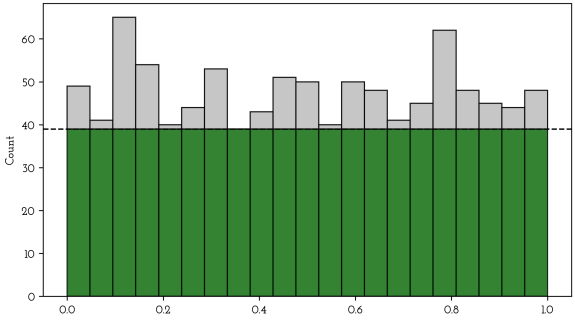
Putting everything together:

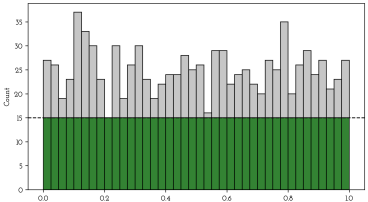
$$\text{total cost} = \sqrt{\text{bucket size}} + \sqrt{\text{backyard size}}$$

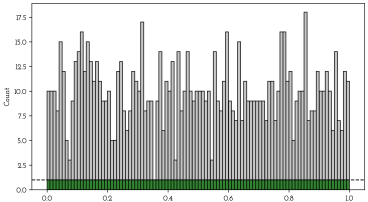
Backyard Size VS In-Bucket Cost

Tension arises.

- "Big" buckets lead to high in-bucket cost.
- "Small" buckets demand a big backyard.







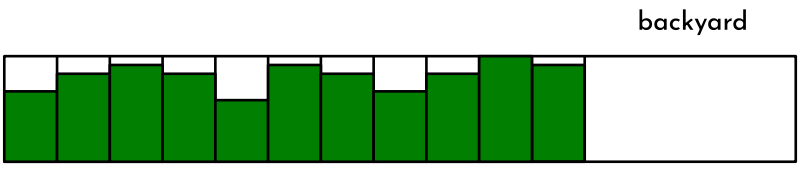
Our Insight

Numbers placed in the backyard are not truly adversarial. We can try to use buckets there!

Our Insight

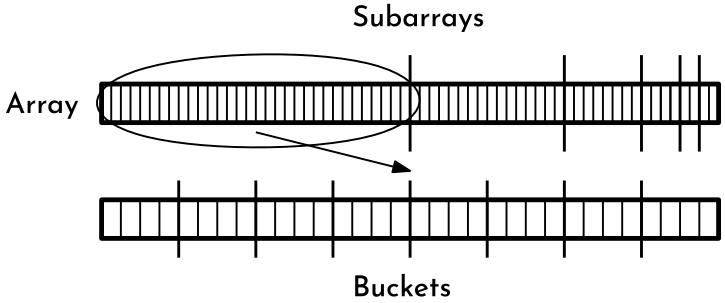
Numbers placed in the backyard are not truly adversarial. We can try to use buckets there!

It doesn't help to commit to bucket sizes at the beginning. We wait until the very last minute.



Subarrays

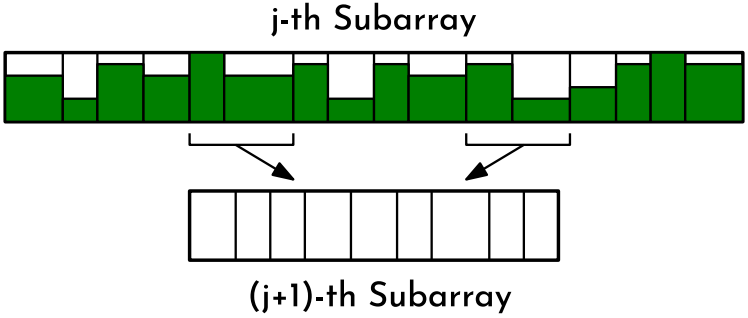
We subdivide the array into subarrays of geometrically decreasing size.
Subarrays are further subdivided to buckets.
Commit to the sizes of the buckets only for the first subarray.



Dynamic Bucket Sizes

Once a bucket of a subarray overflows, the sizes of the buckets of the next subarray are specified.

Next subarray compensates for the imbalances of the previous one.



Main Invariant

Adaptivity allows for significantly smaller buckets. Buckets of size $O(\log^2 n)$ suffice to maintain the following invariant.

Lemma (Informal)

The following invariant holds with high probability: Each subarray j is fully filled before a single bucket of subarray $j + 1$ overflows.

Corollary

Each bucket is filled with "valid" numbers.

Putting Everything Together

Cost of a single bucket using the adversarial algorithm of Aamand et al.

$$O(\log n) \cdot (\text{mapped interval length})$$

Since the blocks mapped to the buckets of a subarray form a partition of $[0, 1]$ and the buckets are of similar size, the in-bucket cost of each subarray is $O(\log n)$.

Since there are $O(\log(n))$ subarrays the total cost of the algorithm is $O(\log^2 n)$ under the condition that the invariant holds.

Online TSP

Elements are points in the d-dimensional unit cube.

At timestep t, point $x(t)$ arrives and must be placed in an empty cell of the array.

The objective is to minimize the sum of the Euclidean distances between consecutive points in the array:

$$\text{Cost}(A) = \sum_{i=2}^{n} \|A[i] - A[i-1]\|$$

Online TSP

Adversarial: Introduced by Abrahamsen et al. [ESA24], later improved to a tight $O(\sqrt{n})$ by Bertram [ESA25].

Stochastic: We introduce Stochastic Online TSP as a multidimensional extension of stochastic online sorting. Points are drawn i.i.d. from the uniform distribution on the d-dimensional unit cube.

What Changes?

Challenge: For any partition of the hypercube into blocks, the optimal solution might not visit points block by block.

Partitioning into blocks: While the number of blocks is less than $n / \log^2 n$ we use round robin over the dimensions. Each time we split each block in half across the selected dimension.

Approach Outline

We leverage existing results regarding the concentration of TSP tour lengths over randomly drawn points to prove the following theorem.

Theorem (Informal)

The optimal tour that visits the points of any subarray block-by-block has length within a constant factor of OPT with high probability.

The rest of the analysis closely follows the one-dimensional setting.

Open Problems

- Random-order Online Sorting. Adversarial set of points presented in uniformly random order. (Semi-random models? Adversarial corruptions?)
- "Prophet" Online Sorting. Points are sampled independently from (different) known distributions.

Thank you!