Naively sorting evolving data is optimal and robust

George Giakkoupis[1], Marcos Kiwi[2], Dimitrios Los[1]

[1]INRIA, France [2]Universidad de Chile

Background

Sorting with evolving data

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.

Sorting with evolving data

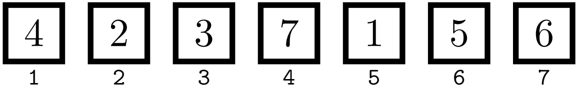
- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.

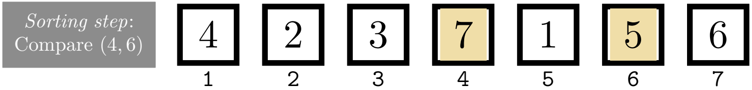
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.



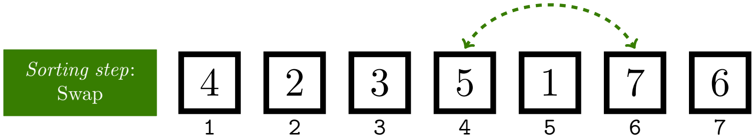
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.



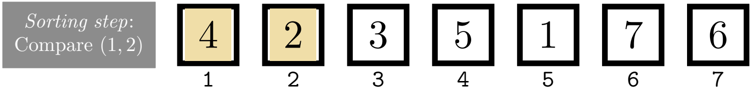
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.



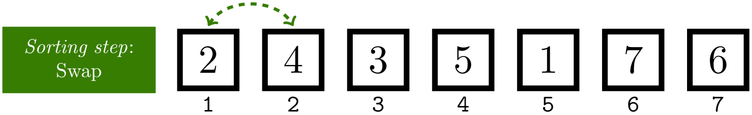
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.



Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.



Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.

Sorting with evolving data

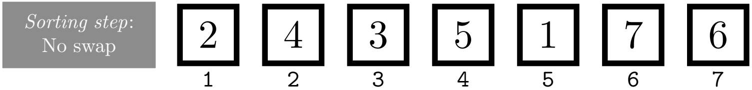
- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.

Sorting step:
No swap

2	4	3	5	1	7	6
1	2	3	4	5	6	7

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped.



Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.

Sorting step:
No swap

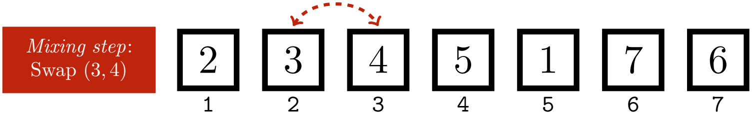
2	4	3	5	1	7	6
1	2	3	4	5	6	7

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM$^+$12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.

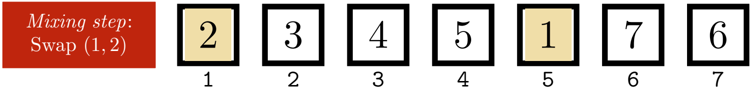
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



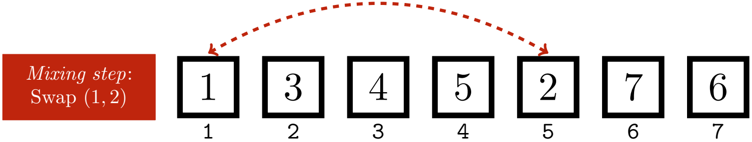
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM⁺12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



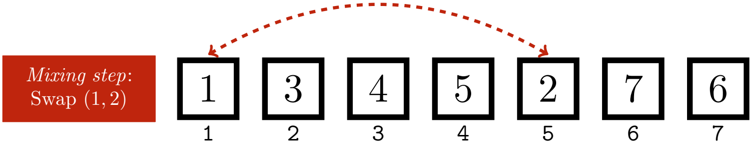
Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



Sorting with evolving data

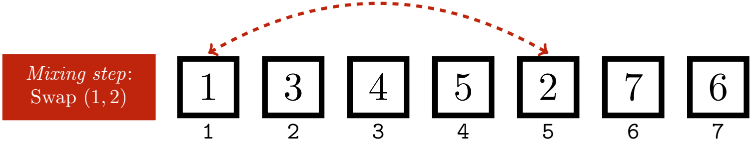
- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM⁺12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



- **Aim:** maintain an ordering π_t which is close to the true ordering

Sorting with evolving data

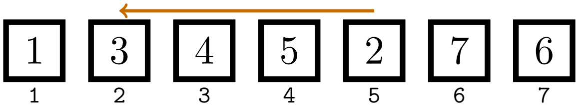
- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



- **Aim:** maintain an ordering π_t which is close to the true ordering
  - Maximum deviation: $\mathrm{mdev}(\pi_t) = \max_{i \in [n]} |\pi_t(i) - i|$

Sorting with evolving data

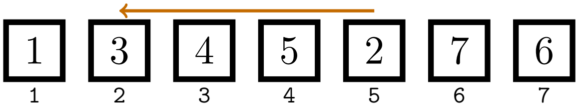
- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



- **Aim:** maintain an ordering π_t which is close to the true ordering
  - Maximum deviation: $\mathrm{mdev}(\pi_t) = \max_{i \in [n]} |\pi_t(i) - i|$

Sorting with evolving data

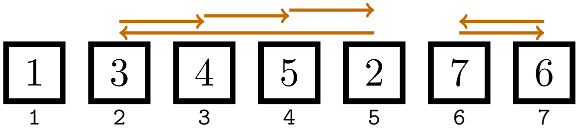
- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



- **Aim:** maintain an ordering π_t which is close to the true ordering
  - Maximum deviation: $\mathrm{mdev}(\pi_t) = \max_{i \in [n]} |\pi_t(i) - i|$
  - Total deviation: $\mathrm{tdev}(\pi_t) = \sum_{i \in [n]} |\pi_t(i) - i|$

Sorting with evolving data

- Introduced by Anagnostopoulos, Kumar, Mahdian, Upfal and Vandin [AKM+12].
- As the sorting algorithm is *executing*, the input is *evolving*.
- More concretely, every *sorting* step is followed by b *mixing* steps.
  - Sorting step: algorithm chooses a pair of indices and compares their values.
  - Mixing step: a pair of items of adjacent *rank* is sampled uniformly at random, and swapped. We don't learn which pair was swapped.



- **Aim:** maintain an ordering π_t which is close to the true ordering
  - Maximum deviation: $\mathrm{mdev}(\pi_t) = \max_{i\in[n]} |\pi_t(i) - i|$
  - Total deviation: $\mathrm{tdev}(\pi_t) = \sum_{i\in[n]} |\pi_t(i) - i|$ (within factor 2 of Kendall tau distance)

Motivation

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly.

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data:

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM+12],

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM+12], *top-k* [HLSZ17],

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM$^+$12], *top-k* [HLSZ17], *minimum spanning trees* [AKM$^+$12], ...

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM$^+$12], *top-k* [HLSZ17], *minimum spanning trees* [AKM$^+$12], ...

- Further problems studied in the evolving data model:

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM$^+$12], *top-k* [HLSZ17], *minimum spanning trees* [AKM$^+$12], ...

- Further problems studied in the evolving data model: *matching* [KLM16],

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM$^+$12], *top-k* [HLSZ17], *minimum spanning trees* [AKM$^+$12], ...

- Further problems studied in the evolving data model: *matching* [KLM16], *PageRank computation* [BKMU12, OMK15, ML21],

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM+12], *top-k* [HLSZ17], *minimum spanning trees* [AKM+12], ...

- Further problems studied in the evolving data model: *matching* [KLM16], *PageRank computation* [BKMU12, OMK15, ML21], *community detection* [ALL+16],

Motivation

- This problem can be used to model several rankings where comparisons are expensive/costly. For instance, among:
  - political candidates (comparison via debates/polls)
  - different systems/features (comparison via A/B testing)
  - tennis/chess players (comparison via H2H games)
  - websites/songs/movies

- Sorting is a critical component in other problems with evolving data: *selection* [AKM+12], *top-k* [HLSZ17], *minimum spanning trees* [AKM+12], ...

- Further problems studied in the evolving data model: *matching* [KLM16], *PageRank computation* [BKMU12, OMK15, ML21], *community detection* [ALL+16], *shortest paths* [ZZW+16], *densest subgraph computation* [ELS15], *tracking labels* [AM22], ...

Previous results

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QuickSort achieves w.h.p. $\text{mdev} = O(\log n)$ and $\text{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\text{tdev} = O(n \log \log n)$.

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QUICKSORT achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QUICKSORT achieves w.h.p. $\mathrm{tdev} = O(n \log \log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QuickSort achieves w.h.p. $\text{mdev} = O(\log n)$ and $\text{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\text{tdev} = O(n \log \log n)$.
  - For any sorting algorithm, $\text{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.

Previous results

- Anagnostopoulos et al. [AKM$^+$12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\mathrm{tdev} = O(n \log\log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.
- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE$^+$18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. $\mathrm{tdev} = O(n)$ in $O(n^2)$ steps.

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\mathrm{tdev} = O(n \log \log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.
- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE+18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. $\mathrm{tdev} = O(n)$ in $O(n^2)$ steps.

  $\rightsquigarrow$ Quadratic time algorithms are optimal.

Previous results

- Anagnostopoulos et al. [AKM$^+$12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\mathrm{tdev} = O(n \log\log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.

- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE$^+$18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. $\mathrm{tdev} = O(n)$ in $O(n^2)$ steps.

  $\rightsquigarrow$ Quadratic time algorithms are optimal.
  - Interleaved with QuickSort converges in $O(n \log n)$ steps.

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\mathrm{tdev} = O(n \log\log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.

- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE+18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. $\mathrm{tdev} = O(n)$ in $O(n^2)$ steps.

  $\rightsquigarrow$ Quadratic time algorithms are optimal.
  - Interleaved with QuickSort converges in $O(n \log n)$ steps.
  - Proof tailored for $b = 1$. *Open problem:* What happens for $b > 1$?

Previous results

- Anagnostopoulos et al. [AKM$^+$12] proved that QuickSort achieves w.h.p. mdev $= O(\log n)$ and tdev $= O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. tdev $= O(n \log\log n)$.
  - For any sorting algorithm, tdev $= \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.

- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE$^+$18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. tdev $= O(n)$ in $O(n^2)$ steps.

  $\rightsquigarrow$ Quadratic time algorithms are optimal.
  - Interleaved with QuickSort converges in $O(n \log n)$ steps.
  - Proof tailored for $b = 1$. *Open problem:* What happens for $b > 1$?
  - Besa Vial et al. [BVDE$^+$18b] *observed* bounds extend to more powerful mixing steps (e.g. *hot-spot adversary*)?

Previous results

- Anagnostopoulos et al. [AKM$^+$12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\mathrm{tdev} = O(n \log \log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.

- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE$^+$18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. $\mathrm{tdev} = O(n)$ in $O(n^2)$ steps.

  $\rightsquigarrow$ Quadratic time algorithms are optimal.
  - Interleaved with QuickSort converges in $O(n \log n)$ steps.
  - Proof tailored for $b = 1$. *Open problem:* What happens for $b > 1$?
  - Besa Vial et al. [BVDE$^+$18b] *observed* bounds extend to more powerful mixing steps (e.g. *hot-spot adversary*)?

- Mahdian [Mah14] proposed Naive-Sort which in each step samples a pair of *adjacent* items; and sort them if they are out of order.

Previous results

- Anagnostopoulos et al. [AKM+12] proved that QuickSort achieves w.h.p. $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$.
  - A more *refined version* of QuickSort achieves w.h.p. $\mathrm{tdev} = O(n \log \log n)$.
  - For any sorting algorithm, $\mathrm{tdev} = \Omega(n)$.
  - *Conjecture:* For any const $b \geq 1$, there exists an algorithms which matches this.

- Besa Vial, Devanny, Eppstein, Goodrich and Johnson [BVDE+18a] showed that Insertion-Sort for $b = 1$ achieves w.h.p. $\mathrm{tdev} = O(n)$ in $O(n^2)$ steps.

  $\rightsquigarrow$ Quadratic time algorithms are optimal.
  - Interleaved with QuickSort converges in $O(n \log n)$ steps.
  - Proof tailored for $b = 1$. *Open problem:* What happens for $b > 1$?
  - Besa Vial et al. [BVDE+18b] *observed* bounds extend to more powerful mixing steps (e.g. *hot-spot adversary*)?

- Mahdian [Mah14] proposed Naive-Sort which in each step samples a pair of *adjacent* items; and sort them if they are out of order.
  - *Open problem:* Does Naive-Sort achieve the optimal mdev and tdev?

Our results

Our results

- Main result:

Our results

- Main result:
  - Naive-Sort achieves optimal $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n)$

Our results

- Main result:
  - NAIVE-SORT achieves optimal $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.

Our results

- Main result:
  - Naive-Sort achieves optimal $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].

Our results

- Main result:
  - Naive-Sort achieves optimal $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM$^+$12, BVDE$^+$18a, Mah14].
- These bounds hold in generalised settings:

Our results

- Main result:
  - NAIVE-SORT achieves optimal $\text{mdev} = O(\log n)$ and $\text{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.

Our results

- Main result:
  - NAIVE-SORT achieves optimal mdev $= O(\log n)$ and tdev $= O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM$^+$12, BVDE$^+$18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.

Our results

- Main result:
  - Naive-Sort achieves optimal $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.
  - For *non-constant* b, $\mathrm{mdev} = \Theta(b \log n)$.

Our results

- Main result:
  - NAIVE-SORT achieves optimal $\text{mdev} = O(\log n)$ and $\text{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.
  - For *non-constant* b, $\text{mdev} = \Theta(b \log n)$.
- Starting from an ordering with $\text{mdev}(\pi_0) = \Delta$, it converges in $O(n \cdot \Delta)$ steps (when $\Delta = \Omega(\log n)$).

Our results

- Main result:
  - NAIVE-SORT achieves optimal mdev $= O(\log n)$ and tdev $= O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.
  - For *non-constant* b, mdev $= \Theta(b \log n)$.
- Starting from an ordering with $\mathrm{mdev}(\pi_0) = \Delta$, it converges in $O(n \cdot \Delta)$ steps (when $\Delta = \Omega(\log n)$).
- By interleaving with QUICKSORT, it converges in $O(n \log n)$ steps.

Our results

- Main result:
  - NAIVE-SORT achieves optimal $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.
  - For *non-constant* b, $\mathrm{mdev} = \Theta(b \log n)$.
- Starting from an ordering with $\mathrm{mdev}(\pi_0) = \Delta$, it converges in $O(n \cdot \Delta)$ steps (when $\Delta = \Omega(\log n)$).
- By interleaving with QUICKSORT, it converges in $O(n \log n)$ steps.
- Analysis techniques based on:

Our results

- Main result:
  - Naive-Sort achieves optimal $\text{mdev} = O(\log n)$ and $\text{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM$^+$12, BVDE$^+$18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.
  - For *non-constant* b, $\text{mdev} = \Theta(b \log n)$.
- Starting from an ordering with $\text{mdev}(\pi_0) = \Delta$, it converges in $O(n \cdot \Delta)$ steps (when $\Delta = \Omega(\log n)$).
- By interleaving with QuickSort, it converges in $O(n \log n)$ steps.
- Analysis techniques based on:
  - An exponential potential function with gaps.

Our results

- Main result:
  - NAIVE-SORT achieves optimal $\text{mdev} = O(\log n)$ and $\text{tdev} = O(n)$ for any const $b \geq 1$ in $t = O(n^2)$ steps.
  - Resolves *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- These bounds hold in generalised settings:
  - Mixing steps where dispacement follows a distribution with finite MGF.
  - Requiring only that a $1/(b+1)$ fraction of sorting steps, every n steps.
  - For *non-constant* b, $\text{mdev} = \Theta(b \log n)$.
- Starting from an ordering with $\text{mdev}(\pi_0) = \Delta$, it converges in $O(n \cdot \Delta)$ steps (when $\Delta = \Omega(\log n)$).
- By interleaving with QUICKSORT, it converges in $O(n \log n)$ steps.
- Analysis techniques based on:
  - An exponential potential function with gaps.
  - A technique for decoupling sorting and mixing steps.

Analysis

Analysis outline

Analysis outline

- **Part A**: Outline for the $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$ bounds.

Analysis outline

- **Part A**: Outline for the $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$ bounds.
  - Analysing *sorting steps*

Analysis outline

- **Part A**: Outline for the mdev $= O(\log n)$ and tdev $= O(n \log n)$ bounds.
  - Analysing *sorting steps*
    - Why the normal exponential potential does not work

Analysis outline

- **Part A**: Outline for the mdev $= O(\log n)$ and tdev $= O(n \log n)$ bounds.
  - Analysing *sorting steps*
    - Why the normal exponential potential does not work
    - Why a *variant* using gaps does work

Analysis outline

- **Part A**: Outline for the mdev $= O(\log n)$ and tdev $= O(n \log n)$ bounds.
  - Analysing *sorting steps*
    - Why the normal exponential potential does not work
    - Why a *variant* using gaps does work
  - Analysing *mixing steps*

Analysis outline

- **Part A**: Outline for the mdev $= O(\log n)$ and tdev $= O(n \log n)$ bounds.
  - Analysing *sorting steps*
    - Why the normal exponential potential does not work
    - Why a *variant* using gaps does work
  - Analysing *mixing steps*
  - Combining sorting and mixing steps

Analysis outline

- **Part A**: Outline for the mdev $= O(\log n)$ and tdev $= O(n \log n)$ bounds.
  - Analysing *sorting steps*
    - Why the normal exponential potential does not work
    - Why a *variant* using gaps does work
  - Analysing *mixing steps*
  - Combining sorting and mixing steps
- **Part B**: (Brief) outline for the tdev $= O(n)$ bound.

Part A: The $\mathrm{mdev} = O(\log n)$ and $\mathrm{tdev} = O(n \log n)$ bounds

Why the normal exponential potential does not work

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps.

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps. Recall that NAIVE-SORT tries to sort items in adjacent positions.

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i\in[n]} \left(e^{\alpha\cdot|i-\pi_t(i)|} - 1\right).$$

- Assume *only* sorting steps. Recall that NAIVE-SORT tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs.

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps. Recall that Naive-Sort tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs. For example,

$$\pi_t = (4, 5, 6, \mathbf{7}, \mathbf{1}, 2, 3)$$

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps. Recall that NAIVE-SORT tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs. For example,

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- *All* items in the first sorted block contribute the same to Φ_t.

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i\in[n]} \left(e^{\alpha\cdot|i-\pi_t(i)|} - 1\right).$$

- Assume *only* sorting steps. Recall that NAIVE-SORT tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs. For example,

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- *All* items in the first sorted block contribute the same to Φ_t.
- BUT *only* the head of a block decreases in expectation.

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps. Recall that Naive-Sort tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs. For example,

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- *All* items in the first sorted block contribute the same to Φ_t.
- BUT *only* the head of a block decreases in expectation.
- So, the potential satisfies

$$\mathbf{E}\left[\, \Phi_{t+1} | \Phi_t \,\right] = \Phi_t \cdot \left(1 - \Theta\left(\frac{1}{n^2} \right) \right).$$

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps. Recall that NAIVE-SORT tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs. For example,

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- *All* items in the first sorted block contribute the same to Φ_t.
- BUT *only* the head of a block decreases in expectation.
- So, the potential satisfies

$$\mathbf{E}\left[\, \Phi_{t+1} | \Phi_t \,\right] = \Phi_t \cdot \left(1 - \Theta\left(\frac{1}{n^2} \right) \right).$$

- Can only be used to prove sorting in $O(n^3)$ steps ...

Why the normal exponential potential does not work

- The exponential potential function with smoothing parameter α is defined as

$$\Phi_t = \sum_{i \in [n]} \left(e^{\alpha \cdot |i - \pi_t(i)|} - 1 \right).$$

- Assume *only* sorting steps. Recall that NAIVE-SORT tries to sort items in adjacent positions.
- **Problem:** There could be very *few* sortable pairs. For example,

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- *All* items in the first sorted block contribute the same to Φ_t.
- BUT *only* the head of a block decreases in expectation.
- So, the potential satisfies

$$\mathbf{E}\left[\, \Phi_{t+1} | \Phi_t \,\right] = \Phi_t \cdot \left(1 - \Theta\left(\frac{1}{n^2} \right) \right).$$

- Can only be used to prove sorting in $O(n^3)$ steps ... and does not work for mixing steps.

Adding gaps (I)

Adding gaps (I)

- So, how can we handle *sorted blocks*?

Adding gaps (I)

- So, how can we handle *sorted blocks*?

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

Adding gaps (I)

- So, how can we handle *sorted blocks*?

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- **Main idea:** Add $n \cdot (d-1)$ *gaps* to make distances non-uniform.

$$\pi_t = (6, 2, 5, 7, 1, 4, 3) \quad \rightarrow \quad \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{6}, 2, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right),$$

Adding gaps (I)

- So, how can we handle *sorted blocks*?

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- **Main idea:** Add $n \cdot (d-1)$ *gaps* to make distances non-uniform.

$$\pi_t = (6, 2, 5, 7, 1, 4, 3) \quad \rightarrow \quad \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{6}, 2, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right),$$

so that no item has a gap to its *left/right* if its target is to the *left/right*.

Adding gaps (I)

- So, how can we handle *sorted blocks*?

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- **Main idea:** Add $n \cdot (d-1)$ *gaps* to make distances non-uniform.

$$\pi_t = (6, 2, 5, 7, 1, 4, 3) \quad \to \quad \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{6}, 2, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right),$$

  so that no item has a gap to its *left/right* if its target is to the *left/right*.

- Sorted state:

$$\left(\underset{1}{1}, \perp, \underset{2}{2}, \perp, \underset{3}{3}, \perp, \underset{4}{4}, \perp, \underset{5}{5}, \perp, \underset{6}{6}, \perp, \underset{7}{7}\right).$$

Adding gaps (I)

- So, how can we handle *sorted blocks*?

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- **Main idea:** Add $n \cdot (d-1)$ *gaps* to make distances non-uniform.

$$\pi_t = (6, 2, 5, 7, 1, 4, 3) \quad \rightarrow \quad \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{6}, 2, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right),$$

  so that no item has a gap to its *left/right* if its target is to the *left/right*.

- Sorted state:

$$\left(\underset{1}{1}, \perp, \underset{2}{2}, \perp, \underset{3}{3}, \perp, \underset{4}{4}, \perp, \underset{5}{5}, \perp, \underset{6}{6}, \perp, \underset{7}{7}\right).$$

- Swapping values $(2, 6)$ gives

$$l_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{\mathbf{6}}, \mathbf{2}, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right)$$

Adding gaps (I)

- So, how can we handle *sorted blocks*?

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- **Main idea:** Add $n \cdot (d-1)$ *gaps* to make distances non-uniform.

$$\pi_t = (6, 2, 5, 7, 1, 4, 3) \quad \to \quad \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{6}, 2, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right),$$

so that no item has a gap to its *left/right* if its target is to the *left/right*.

- Sorted state:

$$\left(\underset{1}{1}, \perp, \underset{2}{2}, \perp, \underset{3}{3}, \perp, \underset{4}{4}, \perp, \underset{5}{5}, \perp, \underset{6}{6}, \perp, \underset{7}{7}\right).$$

- Swapping values $(2, 6)$ gives

$$l_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \perp, \underset{3}{\mathbf{6}}, \mathbf{2}, \underset{4}{\perp}, \perp, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right) \to l_{t+1} = \left(\underset{1}{\perp}, \perp, \underset{2}{\mathbf{2}}, \perp, \underset{3}{\perp}, \perp, \underset{4}{\perp}, \mathbf{6}, \underset{5}{5}, 7, \underset{6}{1}, 4, \underset{7}{3}\right)$$

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- Now, with the gaps, we have that

$$\ell_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, 4, \underset{3}{5}, 6, \underset{4}{7}, 1, \underset{5}{2}, 3, \underset{6}{\perp}, \perp, \underset{7}{\perp}\right)$$

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- Now, with the gaps, we have that

$$\ell_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, 4, \underset{3}{5}, 6, \underset{4}{7}, 1, \underset{5}{2}, 3, \underset{6}{\perp}, \perp, \underset{7}{\perp}\right)$$

- and all distances are increasing in a *sorted block.*

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- Now, with the gaps, we have that

$$\ell_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, 4, \underset{3}{5}, 6, \underset{4}{7}, 1, \underset{5}{2}, 3, \underset{6}{\perp}, \perp, \underset{7}{\perp} \right)$$

- and all distances are increasing in a *sorted block.*
- Since Φ_t is exponential,

$$\Phi_t = \sum_{j:\ell_t(j)\neq\perp} e^{\alpha|d\cdot\ell_t(j)-j|},$$

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- Now, with the gaps, we have that

$$\ell_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, 4, \underset{3}{5}, 6, \underset{4}{7}, 1, \underset{5}{2}, 3, \underset{6}{\perp}, \perp, \underset{7}{\perp} \right)$$

- and all distances are increasing in a *sorted block*.
- Since Φ_t is exponential,

$$\Phi_t = \sum_{j:\ell_t(j)\neq\perp} e^{\alpha|d\cdot\ell_t(j)-j|},$$

the contribution of the *head* of the block dominates.

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- Now, with the gaps, we have that

$$\ell_t = \Big(\underset{1}{\perp}, \perp, \underset{2}{\perp}, \underset{3}{4}, \underset{4}{5}, 6, \underset{5}{7}, 1, \underset{6}{2}, 3, \perp, \perp, \underset{7}{\perp}\Big)$$

- and all distances are increasing in a *sorted block*.
- Since Φ_t is exponential,

$$\Phi_t = \sum_{j:\ell_t(j)\neq\perp} e^{\alpha|d\cdot\ell_t(j)-j|},$$

the contribution of the *head* of the block dominates. $\rightsquigarrow$ overall decrease

Adding gaps (II)

- Returning to the previous example

$$\pi_t = (4, 5, 6, 7, 1, 2, 3)$$

- Now, with the gaps, we have that

$$\ell_t = \left(\underset{1}{\perp}, \perp, \underset{2}{\perp}, 4, \underset{3}{5}, 6, \underset{4}{7}, 1, \underset{5}{2}, 3, \underset{6}{\perp}, \perp, \underset{7}{\perp}\right)$$

- and all distances are increasing in a *sorted block.*
- Since Φ_t is exponential,

$$\Phi_t = \sum_{j:\ell_t(j)\neq\perp} e^{\alpha|d\cdot\ell_t(j)-j|},$$

  the contribution of the *head* of the block dominates. $\rightsquigarrow$ overall decrease
- So, in a sorting step, we can show that

$$\mathbf{E}\left[\,\Phi_{t+1}|\Phi_t\,\right] \leq \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right).$$

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \texttt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

- We enforce this by performing successive swaps.

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \texttt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

- We enforce this by performing successive swaps. Also *do not increase* Φ_t.

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

- We enforce this by performing successive swaps. Also *do not increase* Φ_t.
- Then, we bound

$$|\pi_t(i) - i| \le |\pi_t(i) - \tau_t(\pi_t(i))| + |\tau_t(\pi_t(i)) - i|.$$

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

- We enforce this by performing successive swaps. Also *do not increase* Φ_t.
- Then, we bound

$$|\pi_t(i) - i| \leq |\pi_t(i) - \tau_t(\pi_t(i))| + |\tau_t(\pi_t(i)) - i|.$$

- On the RHS, we bound
  - the first term using the analysis involving Φ_t.

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

- We enforce this by performing successive swaps. Also *do not increase* Φ_t.
- Then, we bound

$$|\pi_t(i) - i| \leq |\pi_t(i) - \tau_t(\pi_t(i))| + |\tau_t(\pi_t(i)) - i|.$$

- On the RHS, we bound
  - the first term using the analysis involving Φ_t.
  - the second term by analysing n *weakly-dependent random walks*,

Handling mixing steps

- Our aim is to decouple sorting and mixing steps.
- **Main idea:** Maintain a *target array* τ_t (initially $\tau_0 = \mathtt{id}_n$)

$$\pi_t = (4, 1, 3, 5, 6, 2, 7) \qquad \tau_t = (1, 2, 3, 4, 5, 6, 7).$$

  And, in a mixing step, swap the target positions of the items, i.e.,

$$\pi_{t+1} = (\mathbf{3}, 1, \mathbf{4}, 5, 6, 2, 7) \quad \tau_{t+1} = (1, 2, \mathbf{4}, \mathbf{3}, 5, 6, 7).$$

- This way, the potential *does not change*, i.e., $\Phi_{t+1} = \Phi_t$.
- To preserve increasing distances needed for the sorting analysis, we require that

$$\pi_t(i) < \pi_t(i+1) \quad \Rightarrow \quad \tau_t(\pi_t(i)) < \tau_t(\pi_t(i+1)).$$

- We enforce this by performing successive swaps. Also *do not increase* Φ_t.
- Then, we bound

$$|\pi_t(i) - i| \leq |\pi_t(i) - \tau_t(\pi_t(i))| + |\tau_t(\pi_t(i)) - i|.$$

- On the RHS, we bound
  - the first term using the analysis involving Φ_t.
  - the second term by analysing n *weakly-dependent random walks*, e.g., using $\Psi_t = \sum_{i \in [n]} e^{\beta \cdot |i - \tau_t^{-1}(i)|}$.

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \le \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn}$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \leq \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \leq \Phi_t \cdot e^{-\Omega(k)}$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \le \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \le \Phi_t \cdot e^{-\Omega(k)} \le n \cdot e^{\alpha \Delta_t} \cdot e^{-\Omega(k)},$$

where $\Delta_t = \mathrm{mdev}(\pi_t)$.

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\,\Phi'_{t+kn} \middle| \Phi_t\,\right] \le \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \le \Phi_t \cdot e^{-\Omega(k)} \le n \cdot e^{\alpha \Delta_t} \cdot e^{-\Omega(k)},$$

where $\Delta_t = \mathrm{mdev}(\pi_t)$.

- By "resetting" the targets, for $k = \Omega(\Delta_t + \log n)$ we have that

$$\mathbf{E}\left[\,\Phi_{t+kn} | \Phi_t\,\right] \le n \cdot e^{O(\sqrt{(\Delta_t + \log n) \cdot \log n})},$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \leq \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \leq \Phi_t \cdot e^{-\Omega(k)} \leq n \cdot e^{\alpha \Delta_t} \cdot e^{-\Omega(k)},$$

  where $\Delta_t = \mathrm{mdev}(\pi_t)$.
- By "resetting" the targets, for $k = \Omega(\Delta_t + \log n)$ we have that

$$\mathbf{E}\left[\, \Phi_{t+kn} \middle| \Phi_t \,\right] \leq n \cdot e^{O(\sqrt{(\Delta_t + \log n) \cdot \log n})},$$

  and so

$$\mathrm{mdev}(\pi_{t+kn}) = O(\sqrt{(\Delta_t + \log n) \cdot \log n}).$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \leq \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \leq \Phi_t \cdot e^{-\Omega(k)} \leq n \cdot e^{\alpha \Delta_t} \cdot e^{-\Omega(k)},$$

where $\Delta_t = \mathrm{mdev}(\pi_t)$.

- By "resetting" the targets, for $k = \Omega(\Delta_t + \log n)$ we have that

$$\mathbf{E}\left[\, \Phi_{t+kn} | \Phi_t \,\right] \leq n \cdot e^{O(\sqrt{(\Delta_t + \log n) \cdot \log n})},$$

and so

$$\mathrm{mdev}(\pi_{t+kn}) = O(\sqrt{(\Delta_t + \log n) \cdot \log n}).$$

- By applying *iteratively*, after $m = \Omega(n \cdot (\Delta_0 + \log n))$ steps w.h.p.

$$\mathrm{mdev}(\pi_m) = O(\log n) \quad \text{and} \quad \mathrm{tdev}(\pi_m) = O(n \log n).$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that
$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$
- In these steps, the potential changes by
$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \le \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \le \Phi_t \cdot e^{-\Omega(k)} \le n \cdot e^{\alpha \Delta_t} \cdot e^{-\Omega(k)},$$
where $\Delta_t = \mathrm{mdev}(\pi_t)$.
- By “resetting” the targets, for $k = \Omega(\Delta_t + \log n)$ we have that
$$\mathbf{E}\left[\, \Phi_{t+kn} | \Phi_t \,\right] \le n \cdot e^{O(\sqrt{(\Delta_t + \log n) \cdot \log n})},$$
and so
$$\mathrm{mdev}(\pi_{t+kn}) = O(\sqrt{(\Delta_t + \log n) \cdot \log n}).$$

Convergence time bounds are tight.

- By applying *iteratively*, after $m = \Omega(n \cdot (\Delta_0 + \log n))$ steps w.h.p.
$$\mathrm{mdev}(\pi_m) = O(\log n) \quad \text{and} \quad \mathrm{tdev}(\pi_m) = O(n \log n).$$

Combining the two analyses

- In kn steps (for $k = \Omega(\log n)$), we have that

$$\max_{i \in [n]} |\tau_t(i) - i| = O\left(\sqrt{k \log n}\right).$$

- In these steps, the potential changes by

$$\mathbf{E}\left[\, \Phi'_{t+kn} \middle| \Phi_t \,\right] \leq \Phi_t \cdot \left(1 - \Omega\left(\frac{1}{n}\right)\right)^{kn} \leq \Phi_t \cdot e^{-\Omega(k)} \leq n \cdot e^{\alpha \Delta_t} \cdot e^{-\Omega(k)},$$

where $\Delta_t = \operatorname{mdev}(\pi_t)$.

- By "resetting" the targets, for $k = \Omega(\Delta_t + \log n)$ we have that

$$\mathbf{E}\left[\, \Phi_{t+kn} \middle| \Phi_t \,\right] \leq n \cdot e^{O(\sqrt{(\Delta_t + \log n) \cdot \log n})},$$

and so

$$\operatorname{mdev}(\pi_{t+kn}) = O(\sqrt{(\Delta_t + \log n) \cdot \log n}).$$

Convergence time bounds are tight.

Bounds hold for relaxed mixing steps.

- By applying *iteratively*, after $m = \Omega(n \cdot (\Delta_0 + \log n))$ steps w.h.p.

$$\operatorname{mdev}(\pi_m) = O(\log n) \quad \text{and} \quad \operatorname{tdev}(\pi_m) = O(n \log n).$$

Part B: The $\mathrm{tdev} = O(n)$ bound

Why the previous analysis does not extend

- The previous analysis uses *long* intervals of length kn.
- **Problem:** When $k = o(\log n)$, then $\mathrm{mdev}(\tau) = \omega(k)$.
- For example, when $k = \Theta(1)$, it follows that $\mathrm{mdev}(\tau) = \Theta(\log n / \log\log n)$, and so the previous upper bound *does not* guarantee a decrease

$$\mathbf{E}\left[\,\Phi_{t+kn} | \Phi_t\,\right] \leq n \cdot e^{-\Theta(1)} \cdot e^{\Theta(\log n/\log\log n)} = n \cdot e^{\Theta(\log n/\log\log n)}.$$

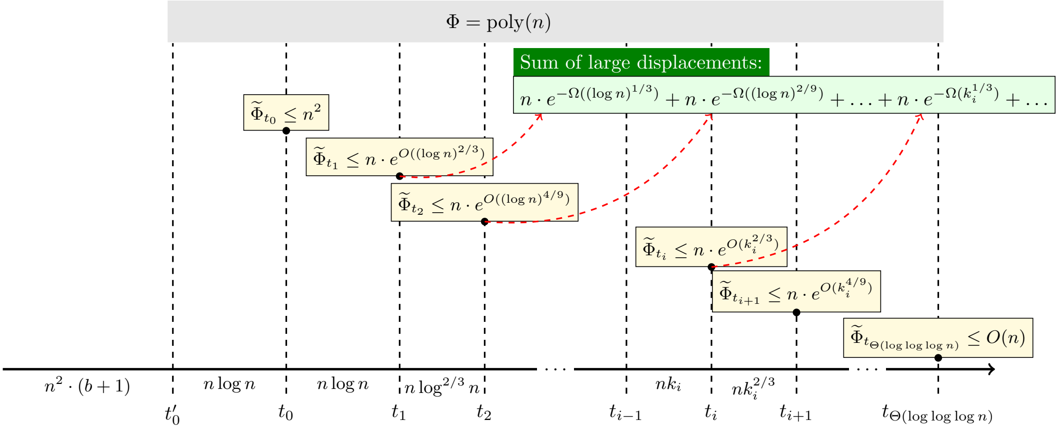
- **Obsrvation:** We are assuming worst-case bound on displacement for *all* items in τ.
- **Solution:** *Reset* targets only for items with small displacements (those below $O(k^{2/3})$).
  - *Small displacements*: handle as in previous analysis.
  - *Large displacements*: show that on aggregate they don't contribute much.
- We prove a concentration inequality (c.f. [LS22]) showing that

$$\mathbf{Pr}\left[\Phi_t \leq n \cdot e^{O(k^{2/3})}\right] \geq 1 - n^{-2}.$$

which implies that *aggregate* contribution of large displacements is $n \cdot e^{-\Omega(k^{1/3})}$.

Putting it all together

- We repeat for $\Theta(\log\log\log n)$ iterations.
- In iteration i, we set $k_{i+1} = (k_i)^{2/3}$.



Conclusions

Summary and open problems

Summary and open problems

In this work, we have shown

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.
  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM$^+$12, BVDE$^+$18a, Mah14].

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.
  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- The analysis works for relaxed conditions:

Summary and open problems

In this work, we have shown

- Naive-Sort achieves the *optimal* mdev and tdev for any const $b \geq 1$.

  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.
  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF and (ii) fraction of sorting steps is $1/(b+1)$.

Summary and open problems

In this work, we have shown

- Naive-Sort achieves the *optimal* mdev and tdev for any const $b \geq 1$.
  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM$^+$12, BVDE$^+$18a, Mah14].
- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF and (ii) fraction of sorting steps is $1/(b+1)$.
  $\rightsquigarrow$ Naive-Sort is *robust*.

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.

  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].

- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF and (ii) fraction of sorting steps is $1/(b+1)$.

  $\rightsquigarrow$ NAIVE-SORT is *robust*.

Several directions for future work:

Summary and open problems

In this work, we have shown

- Naive-Sort achieves the *optimal* mdev and tdev for any const $b \geq 1$.

  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM$^+$12, BVDE$^+$18a, Mah14].

- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF and (ii) fraction of sorting steps is $1/(b+1)$.

  $\rightsquigarrow$ Naive-Sort is *robust*.

Several directions for future work:

- Handle erroneous comparisons (see *biased card shuffling* [DR00, BBHM05]).

Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.
  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF and (ii) fraction of sorting steps is $1/(b+1)$.
  $\rightsquigarrow$ NAIVE-SORT is *robust*.

Several directions for future work:

- Handle erroneous comparisons (see *biased card shuffling* [DR00, BBHM05]).
- Use NAIVE-SORT to other problems with evolving data.

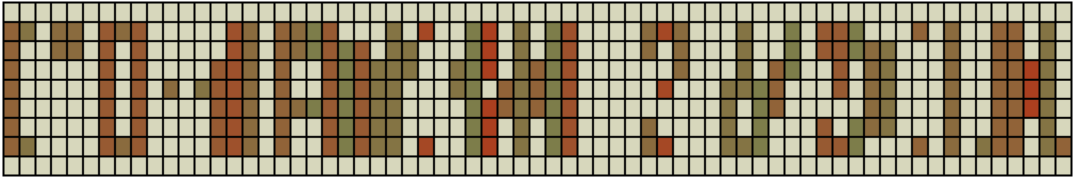
Summary and open problems

In this work, we have shown

- NAIVE-SORT achieves the *optimal* mdev and tdev for any const $b \geq 1$.
  $\rightsquigarrow$ Resolving *conjectures/open problems* in [AKM+12, BVDE+18a, Mah14].
- The analysis works for relaxed conditions: (i) mixing steps with a perturbation distribution with finite MGF and (ii) fraction of sorting steps is $1/(b+1)$.
  $\rightsquigarrow$ NAIVE-SORT is *robust*.

Several directions for future work:

- Handle erroneous comparisons (see *biased card shuffling* [DR00, BBHM05]).
- Use NAIVE-SORT to other problems with evolving data.
- Apply *analysis techniques* to related problems.



For more visualisations, see: `team.inria.fr/wide/papers/focs24`

Bibliography I

- Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin, *Algorithms on evolving graphs*, 3rd Innovations in Theoretical Computer Science Conference (ITCS'12), 2012, pp. 149–160.
- Aris Anagnostopoulos, Jakub Lacki, Silvio Lattanzi, Stefano Leonardi, and Mohammad Mahdian, *Community detection on evolving graphs*, Annual Conference on Neural Information Processing Systems (NeurIPS'16), 2016, pp. 3522–3530.
- Aditya Acharya and David M. Mount, *Optimally tracking labels on an evolving tree*, 34th Canadian Conference on Computational Geometry (CCCG'22), 2022, pp. 1–8.
- Itai Benjamini, Noam Berger, Christopher Hoffman, and Elchanan Mossel, *Mixing times of the biased card shuffling and the asymmetric exclusion process*, Trans. American Mathematical Society **357** (2005), no. 8, 3013–3029.
- Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal, *Pagerank on an evolving graph*, 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'12), 2012, pp. 24–32.

Bibliography II

- Juan José Besa Vial, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson, *Optimally sorting evolving data*, 45th International Colloquium on Automata, Languages, and Programming (ICALP'18), 2018, pp. 81:1–81:13.
- ———, *Quadratic time algorithms appear to be optimal for sorting evolving data*, 20th Workshop on Algorithm Engineering and Experiments (ALENEX'18), 2018, pp. 87–96.
- Persi Diaconis and Arun Ram, *Analysis of systematic scan Metropolis algorithms using Iwahori-Hecke algebra techniques*, Michigan Mathematical Journal **48** (2000), no. 1, 157 – 190.
- Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio, *Efficient densest subgraph computation in evolving graphs*, 24th International Conference on World Wide Web (WWW'15), ACM, 2015, pp. 300–310.
- Qin Huang, Xingwu Liu, Xiaoming Sun, and Jialin Zhang, *Partial sorting problem on evolving data*, Algorithmica **79** (2017), no. 3, 960–983.

Bibliography III

- Varun Kanade, Nikos Leonardos, and Frédéric Magniez, *Stable matching with evolving preferences*, Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, (APPROX/RANDOM'16), LIPIcs, vol. 60, 2016, pp. 36:1–36:13.
- Dimitrios Los and Thomas Sauerwald, *Balanced allocations with incomplete information: The power of two queries*, Proc. 13th Innovations in Theoretical Computer Science Conference, ITCS, 2022, pp. 103:1–103:23.
- Mohammad Mahdian, *Algorithms on evolving data sets*, Talk given at ICERM/Brown Workshop on Stochastic Graph Models, March 17–21, 2014.
- Dingheng Mo and Siqiang Luo, *Agenda: Robust personalized pageranks in evolving graphs*, 30th ACM International Conference on Information and Knowledge Management (CIKM '21), 2021, pp. 1315–1324.

Bibliography IV

- Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi, *Efficient pagerank tracking in evolving networks*, Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15), 2015, pp. 875–884.
- Yiming Zou, Gang Zeng, Yuyi Wang, Xingwu Liu, Xiaoming Sun, Jialin Zhang, and Qiang Li, *Shortest paths on evolving graphs*, 5th International Conference on Computational Social Networks (CSoNet'16), Springer, 2016, pp. 1–13.