ROBOPROX

An Optimal Algorithm for Sorting Pattern-Avoiding Sequences

Michal Opler
Czech Technical University in Prague

July 7, 2025

Graphs
Games
Optimization
Algorithms
Theoretical
Computer Science

Co-funded by the European Union

MŠMT
MINISTRY OF EDUCATION, YOUTH AND SPORTS

Robotics and Advanced Industrial Production
CZ.02.01.01/00/22_008/0004590

Comparison-Based Sorting

- Input: Sequence $x_1, x_2, \ldots, x_n$ (of distinct values).
- Task: Rearrange input to increasing order.
- **Access to input is available only through comparing pairs of elements.**

Comparison-Based Sorting

- Input: Sequence $x_1, x_2, \ldots, x_n$ (of distinct values).
- Task: Rearrange input to increasing order.
- **Access to input is available only through comparing pairs of elements.**

Theorem

Any deterministic comparison-based sorting algorithm must perform $\Omega(n \log n)$ comparisons in the worst case to sort n elements.

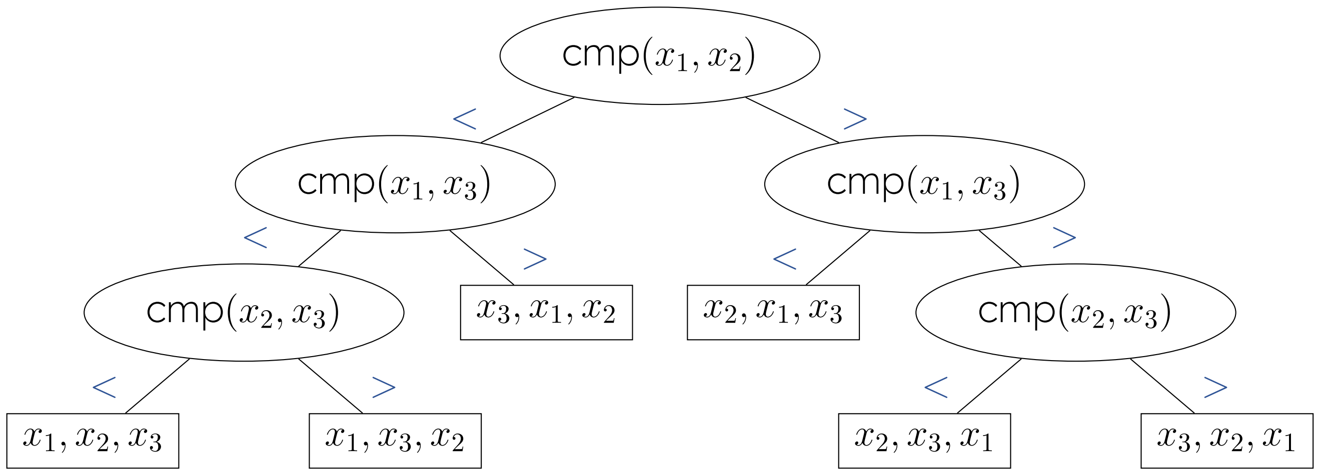
Comparison-Based Sorting

- Input: Sequence $x_1, x_2, \ldots, x_n$ (of distinct values).
- Task: Rearrange input to increasing order.
- **Access to input is available only through comparing pairs of elements.**

Theorem

Any deterministic comparison-based sorting algorithm must perform $\Omega(n \log n)$ comparisons in the worst case to sort n elements.

Proof sketch:



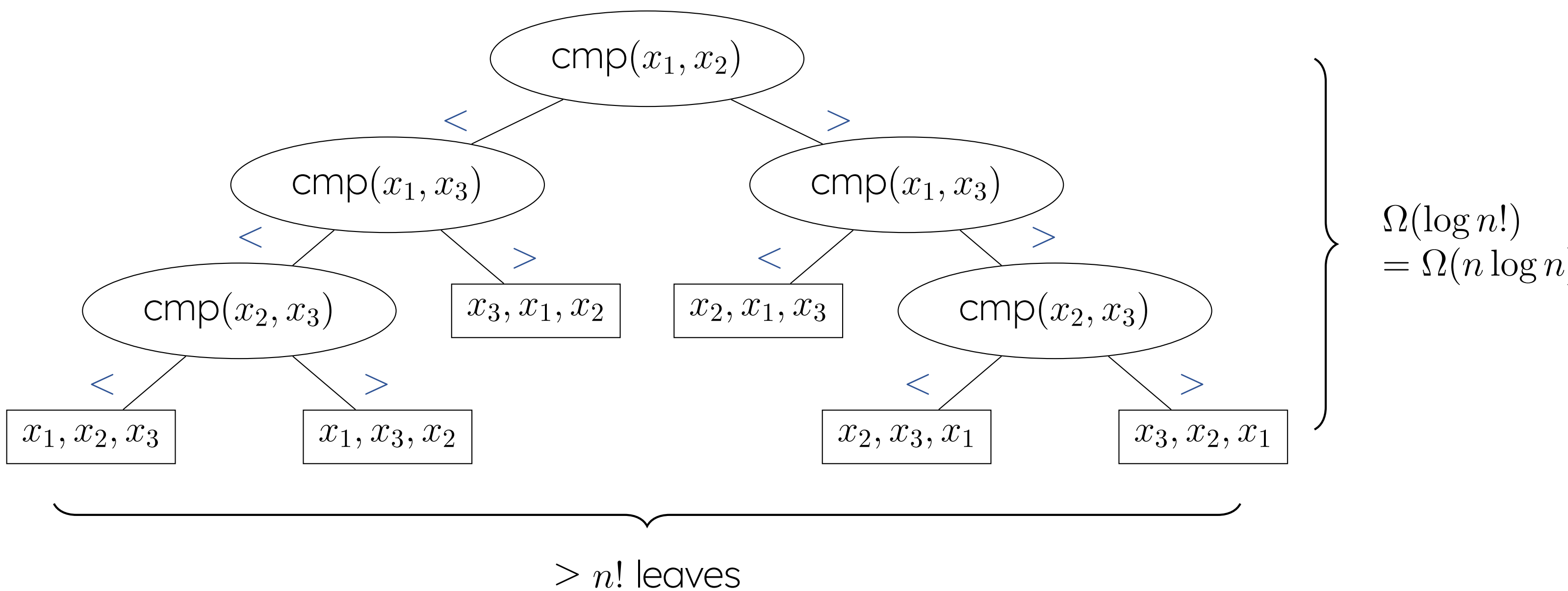
Comparison-Based Sorting

- Input: Sequence $x_1, x_2, \ldots, x_n$ (of distinct values).
- Task: Rearrange input to increasing order.
- **Access to input is available only through comparing pairs of elements.**

Theorem

Any deterministic comparison-based sorting algorithm must perform $\Omega(n \log n)$ comparisons in the worst case to sort n elements.

Proof sketch:



$\square$

Restricted Inputs

- A family of inputs Γ of length n cannot be sorted in $o(\log_2 |\Gamma|)$ time.
- When $|\Gamma| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Restricted Inputs

- A family of inputs Γ of length n cannot be sorted in $o(\log_2 |\Gamma|)$ time.
- When $|\Gamma| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Examples

- Sequences with k runs

1, 4, 6, 9, 2, 5, 7, 3, 8, 10

Restricted Inputs

- A family of inputs $\mathit{\Gamma}$ of length n cannot be sorted in $o(\log_2 |\mathit{\Gamma}|)$ time.
- When $|\mathit{\Gamma}| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Examples

- Sequences with k runs $\mathcal{O}(\log k \cdot n)$

1, 4, 6, 9, 2, 5, 7, 3, 8, 10

Restricted Inputs

- A family of inputs Γ of length n cannot be sorted in $o(\log_2 |\Gamma|)$ time.
- When $|\Gamma| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Examples

- Sequences with k runs $\mathcal{O}(\log k \cdot n)$

  1, 4, 6, 9, 2, 5, 7, 3, 8, 10

- k-increasing sequences

  1, 4, 2, 3, 6, 5, 8, 10, 9, 7

Restricted Inputs

- A family of inputs Γ of length n cannot be sorted in $o(\log_2 |\Gamma|)$ time.
- When $|\Gamma| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Examples

- Sequences with k runs $\mathcal{O}(\log k \cdot n)$

  1, 4, 6, 9, 2, 5, 7, 3, 8, 10

- k-increasing sequences $\mathcal{O}(\log k \cdot n)$

  1, 4, 2, 3, 6, 5, 8, 10, 9, 7

Restricted Inputs

- A family of inputs Γ of length n cannot be sorted in $o(\log_2 |\Gamma|)$ time.
- When $|\Gamma| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Examples

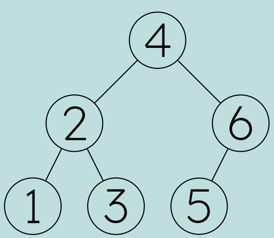
- Sequences with k runs $\qquad \mathcal{O}(\log k \cdot n)$

  1, 4, 6, 9, 2, 5, 7, 3, 8, 10

- k-increasing sequences $\qquad \mathcal{O}(\log k \cdot n)$

  1, 4, 2, 3, 6, 5, 8, 10, 9, 7

- Preorder (traversal) sequences



$\rightarrow$ **4, 2, 1, 3, 6, 5**

Restricted Inputs

- A family of inputs Γ of length n cannot be sorted in $o(\log_2 |\Gamma|)$ time.
- When $|\Gamma| \in 2^{\mathcal{O}(n)}$, there is a possibility of sorting in $\mathcal{O}(n)$ time!

Examples

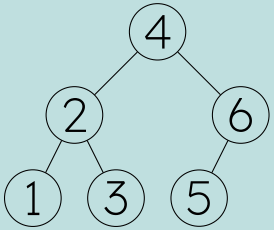
- Sequences with k runs $\quad \mathcal{O}(\log k \cdot n)$

  1, 4, 6, 9, 2, 5, 7, 3, 8, 10

- k-increasing sequences $\quad \mathcal{O}(\log k \cdot n)$

  1, 4, 2, 3, 6, 5, 8, 10, 9, 7

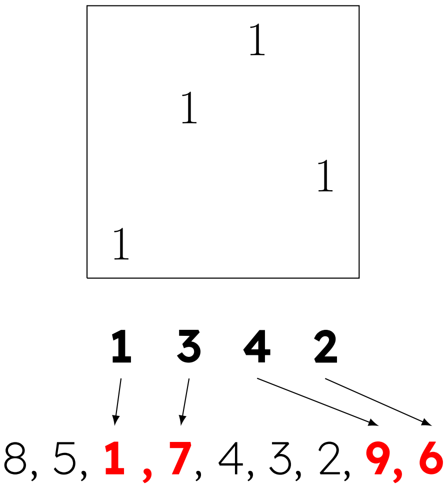
- Preorder (traversal) sequences $\quad \mathcal{O}(n)$



$\rightarrow$ **4, 2, 1, 3, 6, 5**

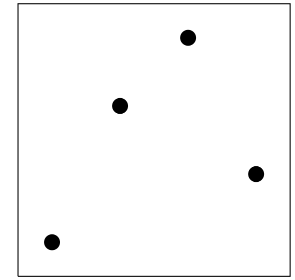
Pattern-Avoiding Inputs

- A permutation π is a sequence $\pi_1, \ldots, \pi_n$ of distinct values from $[n] = \{1, \ldots, n\}$.
- We can also represent π as an $n \times n$ 0-1 matrix M_π.
- A sequence contains a permutation pattern π if it has a subsequence order-isomorphic to π. Otherwise, it avoids π.



Pattern-Avoiding Inputs

- A permutation π is a sequence $\pi_1, \ldots, \pi_n$ of distinct values from $[n] = \{1, \ldots, n\}$.
- We can also represent π as an $n \times n$ 0-1 matrix M_π.
- A sequence contains a permutation pattern π if it has a subsequence order-isomorphic to π. Otherwise, it avoids π.

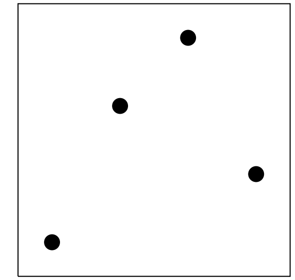


1 3 4 2

8, 5, **1 , 7**, 4, 3, 2, **9, 6**

Pattern-Avoiding Inputs

- A permutation π is a sequence $\pi_1, \ldots, \pi_n$ of distinct values from $[n] = \{1, \ldots, n\}$.
- We can also represent π as an $n \times n$ 0-1 matrix M_π.
- A sequence contains a permutation pattern π if it has a subsequence order-isomorphic to π. Otherwise, it avoids π.



1 3 4 2

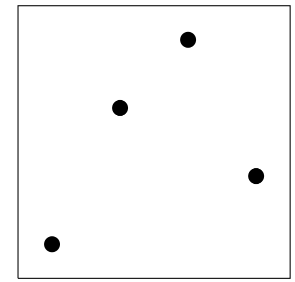
8, 5, **1**, **7**, 4, 3, 2, **9**, **6**

Examples

- k-increasing $\leftrightarrow$ $(k+1, k, \ldots, 1)$-avoiding
- preorder $\leftrightarrow$ stack-sortable $\leftrightarrow$ 231-avoiding [Knuth 1968]

Pattern-Avoiding Inputs

- A permutation π is a sequence $\pi_1, \ldots, \pi_n$ of distinct values from $[n] = \{1, \ldots, n\}$.
- We can also represent π as an $n \times n$ 0-1 matrix M_π.
- A sequence contains a permutation pattern π if it has a subsequence order-isomorphic to π. Otherwise, it avoids π.



1 3 4 2

8, 5, **1**, **7**, 4, 3, 2, **9**, **6**

Examples

- k-increasing $\leftrightarrow$ $(k+1, k, \ldots, 1)$-avoiding
- preorder $\leftrightarrow$ stack-sortable $\leftrightarrow$ 231-avoiding [Knuth 1968]

Theorem (Stanley-Wilf conjecture) [Klazar 2000; Marcus, Tardos 2004]

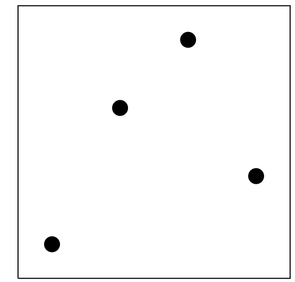
For every pattern π, the number of π-avoiding permutations of length n is $2^{\mathcal{O}_\pi(n)}$.

In fact, the limit $s_\pi = \lim_{n\to\infty} \sqrt[n]{|\{\sigma \text{ avoids } \pi \mid \sigma \in \mathcal{S}_n\}|}$ exists [Arratia 1999]

Stanley-Wilf limit of π

Pattern-Avoiding Inputs

- A permutation π is a sequence $\pi_1, \ldots, \pi_n$ of distinct values from $[n] = \{1, \ldots, n\}$.
- We can also represent π as an $n \times n$ 0-1 matrix M_π.
- A sequence contains a permutation pattern π if it has a subsequence order-isomorphic to π. Otherwise, it avoids π.



1 3 4 2

8, 5, **1**, **7**, 4, 3, 2, **9**, **6**

Examples

- k-increasing $\leftrightarrow$ $(k+1, k, \ldots, 1)$-avoiding
- preorder $\leftrightarrow$ stack-sortable $\leftrightarrow$ 231-avoiding [Knuth 1968]

Theorem (Stanley-Wilf conjecture) [Klazar 2000; Marcus, Tardos 2004]

For every pattern π, the number of π-avoiding permutations of length n is $2^{\mathcal{O}_\pi(n)}$.

In fact, the limit $s_\pi = \lim_{n\to\infty} \sqrt[n]{|\{\sigma \text{ avoids } \pi \mid \sigma \in \mathcal{S}_n\}|}$ exists [Arratia 1999]

Stanley-Wilf limit of π

Question: Can π-avoiding sequences be sorted in linear time for arbitrary π?

Known Results

Can be done "non-uniformly":

Theorem [Fredman 1976]

For any set Γ of permutations of length n, there exists a decision tree of depth $\log_2 |\Gamma| + 2n$ that sorts every input from Γ.

Known Results

Can be done “non-uniformly”:

Theorem [Fredman 1976]

For any set Γ of permutations of length n, there exists a decision tree of depth $\log_2 |\Gamma| + 2n$ that sorts every input from Γ.

Pattern-specific approach: exploit the specific structure of π-avoiding permutations for a fixed π

- k-increasing in $\mathcal{O}(\log k \cdot n)$ time
- 231-avoiding in $\mathcal{O}(n)$ time [Knuth 1968]
- 1234-, 1243- and 2143-avoiding in $\mathcal{O}(n)$ time [Arthur 2007]
- 1324-, 1342-, 1423- and 1432-avoiding in $\mathcal{O}(n \log\log\log n)$ time [Arthur 2007]
- no pattern-specific $o(n \log n)$ algorithm known for 2413-avoiding!

Known Results

Can be done "non-uniformly":

Theorem [Fredman 1976]

For any set Γ of permutations of length n, there exists a decision tree of depth $\log_2 |\Gamma| + 2n$ that sorts every input from Γ.

Pattern-specific approach: exploit the specific structure of π-avoiding permutations for a fixed π

- k-increasing in $\mathcal{O}(\log k \cdot n)$ time
- 231-avoiding in $\mathcal{O}(n)$ time [Knuth 1968]
- 1234-, 1243- and 2143-avoiding in $\mathcal{O}(n)$ time [Arthur 2007]
- 1324-, 1342-, 1423- and 1432-avoiding in $\mathcal{O}(n \log\log\log n)$ time [Arthur 2007]
- no pattern-specific $o(n \log n)$ algorithm known for 2413-avoiding!

Pattern-agnostic approach: Use general-purpose sorting algorithm and analyze it on π-avoiding inputs. In particular, insertion sort into self-adjusting BSTs:

- $n\, 2^{(\alpha(n))^{\mathcal{O}(|\pi|)}}$ time [Chalermsook, Goswami, Kozma, Mehlhorn, Saranurak 2015]
- $n\, 2^{\mathcal{O}(\alpha(n) + |\pi|^2)}$ time [Chalermsook, Gupta, Jiamjitrak, Acosta, Yingch. 2023]

where $\alpha(\cdot)$ is the inverse-Ackermann function

Main Result

Theorem

There is a comparison-based algorithm that sorts π-avoiding sequences of length n in $\mathcal{O}((\log s_\pi + 1) \cdot n)$ time even if π is a priori unknown.

Matches the information-theoretic lower bound!

Overview of the algorithm

Two ingredients:

(I) Merging $n/\log n$ presorted sequences of length $\log n$ in $\mathcal{O}((\log s_\pi + 1) \cdot n)$ time.

(II) Sorting n/k sequences of length k in $\mathcal{O}((\log s_\pi + 1) \cdot n)$ time for any $k \in \mathcal{O}(\log \log \log n)$.

The algorithm

1. Cut up the input sequence into parts of length $\log \log \log n$.
2. Sort all of them using (II). $\mathcal{O}((\log s_\pi + 1) \cdot n)$
3. Three layers of bottom-up MergeSort using (I). $\mathcal{O}((\log s_\pi + 1) \cdot n)$

(I) Efficient Merging

The Combinatorial Hammer

- A 0-1 matrix M contains a pattern π if M_π can be obtained from M by removing rows, columns and turning some 1-entries to 0-entries.
- $\mathrm{ex}_\pi(n) \leftarrow$ a maximum number of 1-entries in a π-avoiding $n \times n$ 0-1 matrix.

$$\begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & 1 \end{pmatrix}$$

contains 132

The Combinatorial Hammer

- A 0-1 matrix M contains a pattern π if M_π can be obtained from M by removing rows, columns and turning some 1-entries to 0-entries.
- $\mathrm{ex}_\pi(n) \leftarrow$ a maximum number of 1-entries in a π-avoiding $n \times n$ 0-1 matrix.

$$\begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & 1 \end{pmatrix}$$

contains 132

The Combinatorial Hammer

- A 0-1 matrix M contains a pattern π if M_π can be obtained from M by removing rows, columns and turning some 1-entries to 0-entries.
- $\mathrm{ex}_\pi(n) \leftarrow$ a maximum number of 1-entries in a π-avoiding $n \times n$ 0-1 matrix.

$$\begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & 1 \end{pmatrix}$$

contains 132

Theorem (Füredi-Hajnal conjecture) [Marcus, Tardos 2004]

For each π, we have $\mathrm{ex}_\pi \in \mathcal{O}_\pi(n)$.

Moreover, the limit $c_\pi = \lim_{n\to\infty} \frac{1}{n}\mathrm{ex}_\pi(n)$ exists and $\mathrm{ex}_\pi(n) \le c_\pi \cdot n$ for all n. [Cibulka 2009]

Füredi-Hajnal limit of π

The Combinatorial Hammer

- A 0-1 matrix M contains a pattern π if M_π can be obtained from M by removing rows, columns and turning some 1-entries to 0-entries.
- $\mathrm{ex}_\pi(n) \leftarrow$ a maximum number of 1-entries in a π-avoiding $n \times n$ 0-1 matrix.

$$\begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & 1 \end{pmatrix}$$

contains 132

Theorem (Füredi-Hajnal conjecture) [Marcus, Tardos 2004]

For each π, we have $\mathrm{ex}_\pi \in \mathcal{O}_\pi(n)$.

Moreover, the limit $c_\pi = \lim_{n\to\infty} \frac{1}{n}\mathrm{ex}_\pi(n)$ exists and $\mathrm{ex}_\pi(n) \le c_\pi \cdot n$ for all n. [Cibulka 2009]

Füredi-Hajnal limit of π

Lemma (The actual hammer)

Every $m \times n$ 0-1 matrix with strictly more than $c_\pi \cdot \max(m, n)$ 1-entries contains π.

The Combinatorial Hammer

- A 0-1 matrix M contains a pattern π if M_π can be obtained from M by removing rows, columns and turning some 1-entries to 0-entries.
- $\mathrm{ex}_\pi(n) \leftarrow$ a maximum number of 1-entries in a π-avoiding $n \times n$ 0-1 matrix.

$$\begin{pmatrix} 0 & 0 & \mathbf{1} & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & 0 & 1 \end{pmatrix}$$ contains 132

Theorem (Füredi-Hajnal conjecture) [Marcus, Tardos 2004]

For each π, we have $\mathrm{ex}_\pi \in \mathcal{O}_\pi(n)$.

Moreover, the limit $c_\pi = \lim_{n\to\infty} \frac{1}{n} \mathrm{ex}_\pi(n)$ exists and $\mathrm{ex}_\pi(n) \leq c_\pi \cdot n$ for all n. [Cibulka 2009]

Füredi-Hajnal limit of π

Lemma (The actual hammer)

Every $m \times n$ 0-1 matrix with strictly more than $c_\pi \cdot \max(m, n)$ 1-entries contains π.

Additionally, s_π and c_π are polynomially related – $s_\pi \in \Omega(c_\pi^{2/9}) \cap \mathcal{O}(c_\pi^2)$ [Cibulka 2009]

$\Rightarrow \log s_\pi$ and $\log c_\pi$ are interchangeable within $\mathcal{O}$-notation.

Efficient Pattern-Avoiding Merge

Simplified version: the algorithm knows π and its Füredi-Hajnal limit c_π.

Let $d \leftarrow \lceil 2c_\pi \rceil,\ m \leftarrow n/\log n$

Input: π-avoiding sequence S partitioned into m presorted sequences $S_1, \ldots, S_m$

1. $S'_1, \ldots, S'_{\lfloor m/d \rfloor} \leftarrow$ merge consecutive d-tuples of sequences
2. while there are some non-exhausted sequences:
3. $S'_{i_1}, \ldots, S'_{i_{d+1}} \leftarrow d+1$ sequences with smallest initial elements
4. $x \leftarrow$ initial element of $S'_{i_{d+1}}$ (largest out of these)
5. output merge of $S'_{i_1}, \ldots, S'_{i_d}$ while smaller than x

Round = One execution of the loop (lines 3.–5.)

Efficient Pattern-Avoiding Merge

Simplified version: the algorithm knows π and its Füredi-Hajnal limit c_π.

Let $d \leftarrow \lceil 2c_\pi \rceil$, $m \leftarrow n/\log n$

Input: π-avoiding sequence S partitioned into m presorted sequences $S_1, \ldots, S_m$

1. $S'_1, \ldots, S'_{\lfloor m/d \rfloor} \leftarrow$ merge consecutive d-tuples of sequences
2. while there are some non-exhausted sequences:
3. $\quad S'_{i_1}, \ldots, S'_{i_{d+1}} \leftarrow d+1$ sequences with smallest initial elements
4. $\quad x \leftarrow$ initial element of $S'_{i_{d+1}}$ (largest out of these)
5. $\quad$ output merge of $S'_{i_1}, \ldots, S'_{i_d}$ while smaller than x

Round = One execution of the loop (lines 3.–5.)

Implementation:

- "large" heap for storing $S'_1, \ldots, S'_{\lfloor m/d \rfloor}$ where the key of S'_i is its initial element
  - $(d+1)\times$ExtractMin in step 3.
  - $(d+1)\times$Insert after step 5. (returning non-empty sequences back)
- "small" heaps for performing d-way merges in steps 1. and 5.

Efficient Pattern-Avoiding Merge

Simplified version: the algorithm knows π and its Füredi-Hajnal limit c_π.

Let $d \leftarrow \lceil 2c_\pi \rceil$, $m \leftarrow n/\log n$

Input: π-avoiding sequence S partitioned into m presorted sequences $S_1, \ldots, S_m$

1. $S'_1, \ldots, S'_{\lfloor m/d \rfloor} \leftarrow$ merge consecutive d-tuples of sequences $\mathcal{O}(\log d \cdot n)$
2. while there are some non-exhausted sequences:
3. $\quad S'_{i_1}, \ldots, S'_{i_{d+1}} \leftarrow d+1$ sequences with smallest initial elements $\mathcal{O}(d \cdot \log n)$
4. $\quad x \leftarrow$ initial element of $S'_{i_{d+1}}$ (largest out of these) $\mathcal{O}(\log n)$
5. $\quad$ output merge of $S'_{i_1}, \ldots, S'_{i_d}$ while smaller than x $\mathcal{O}(\log d \cdot n)$ overall

Round = One execution of the loop (lines 3.–5.)

Implementation:

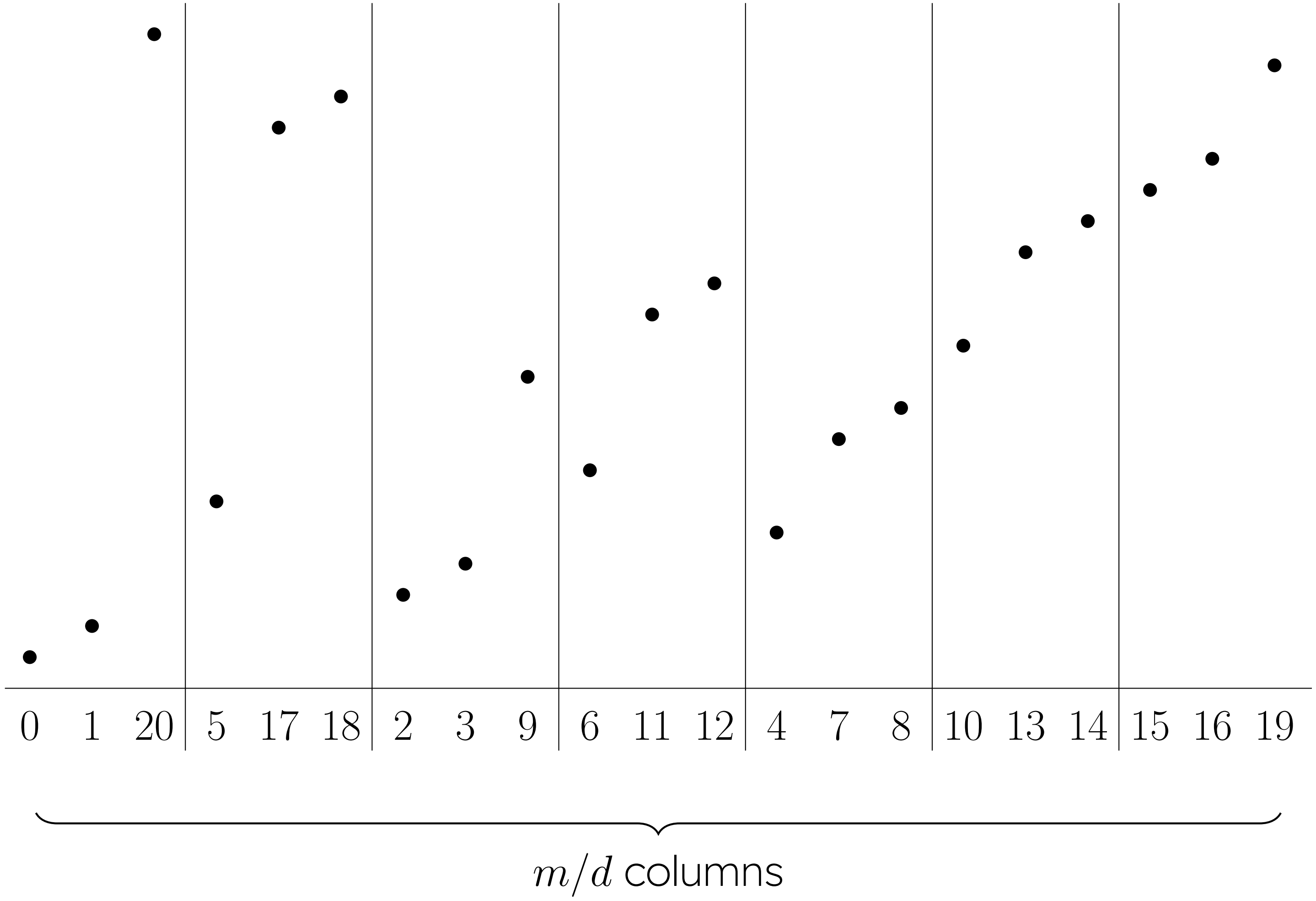
- "large" heap for storing $S'_1, \ldots, S'_{\lfloor m/d \rfloor}$ where the key of S'_i is its initial element
  - $(d+1)\times$ExtractMin in step 3.
  - $(d+1)\times$Insert after step 5. (returning non-empty sequences back)
- "small" heaps for performing d-way merges in steps 1. and 5.

Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.

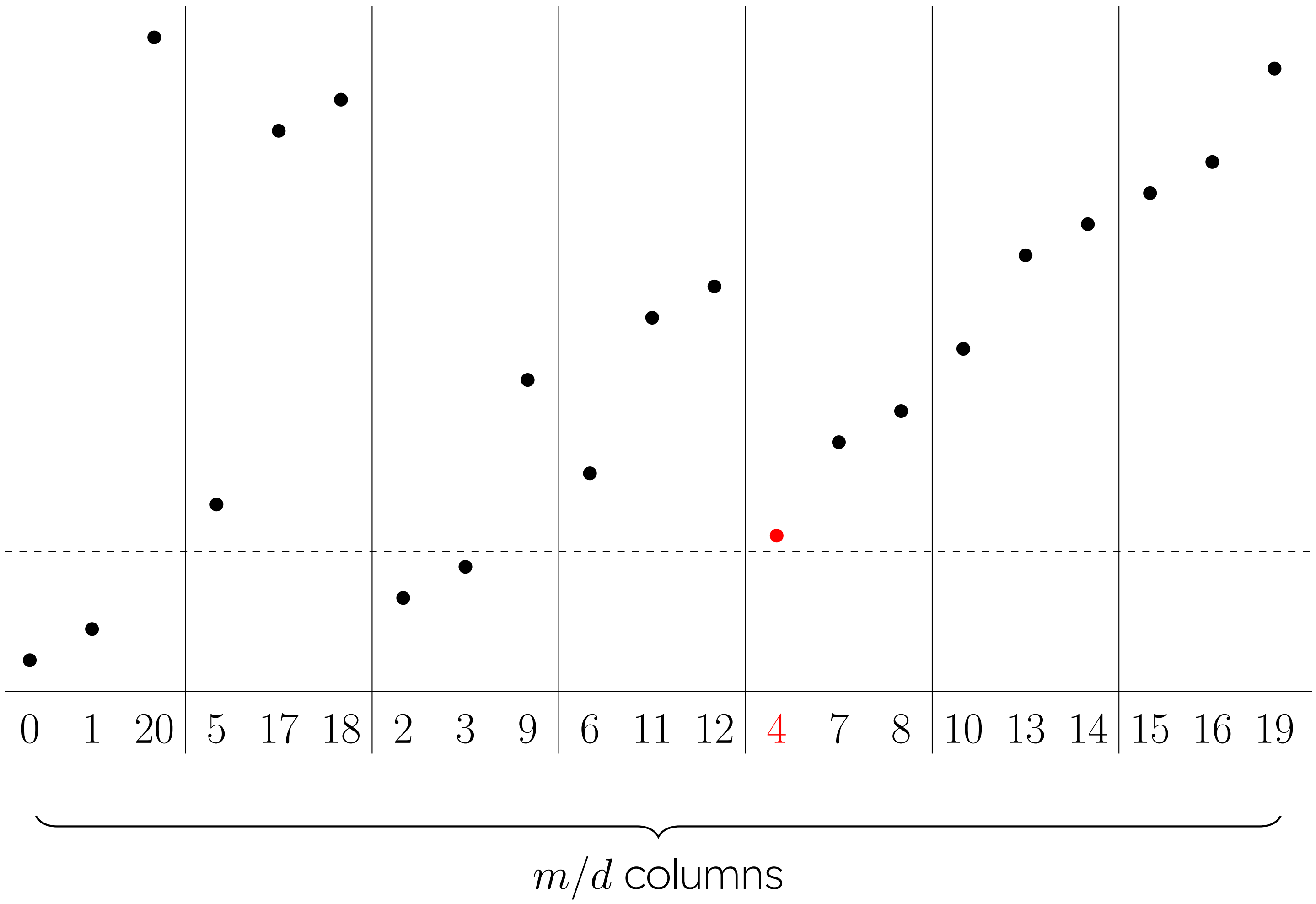
$\Rightarrow \mathcal{O}(\log d \cdot n)$ runtime.

Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.

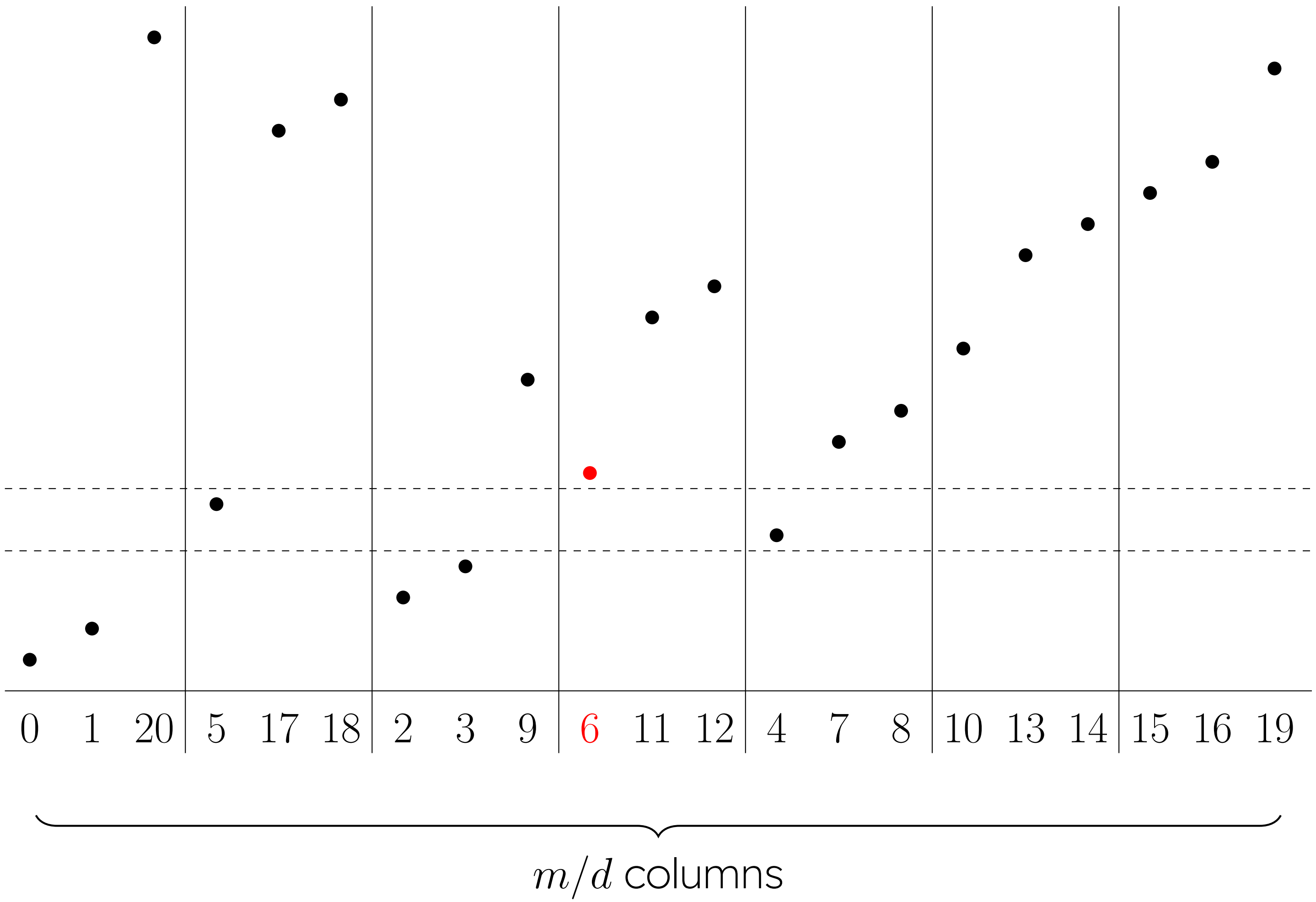
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



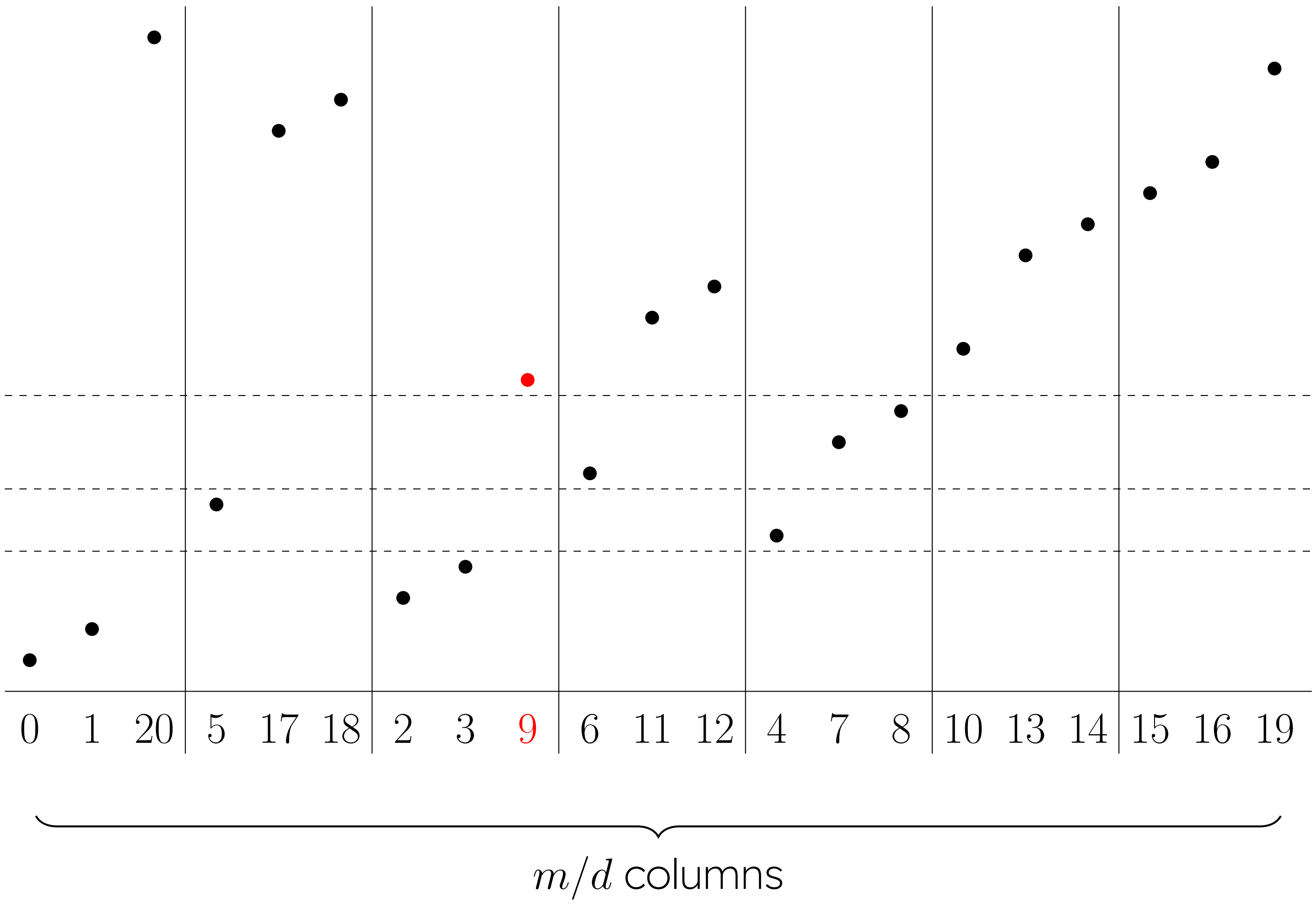
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



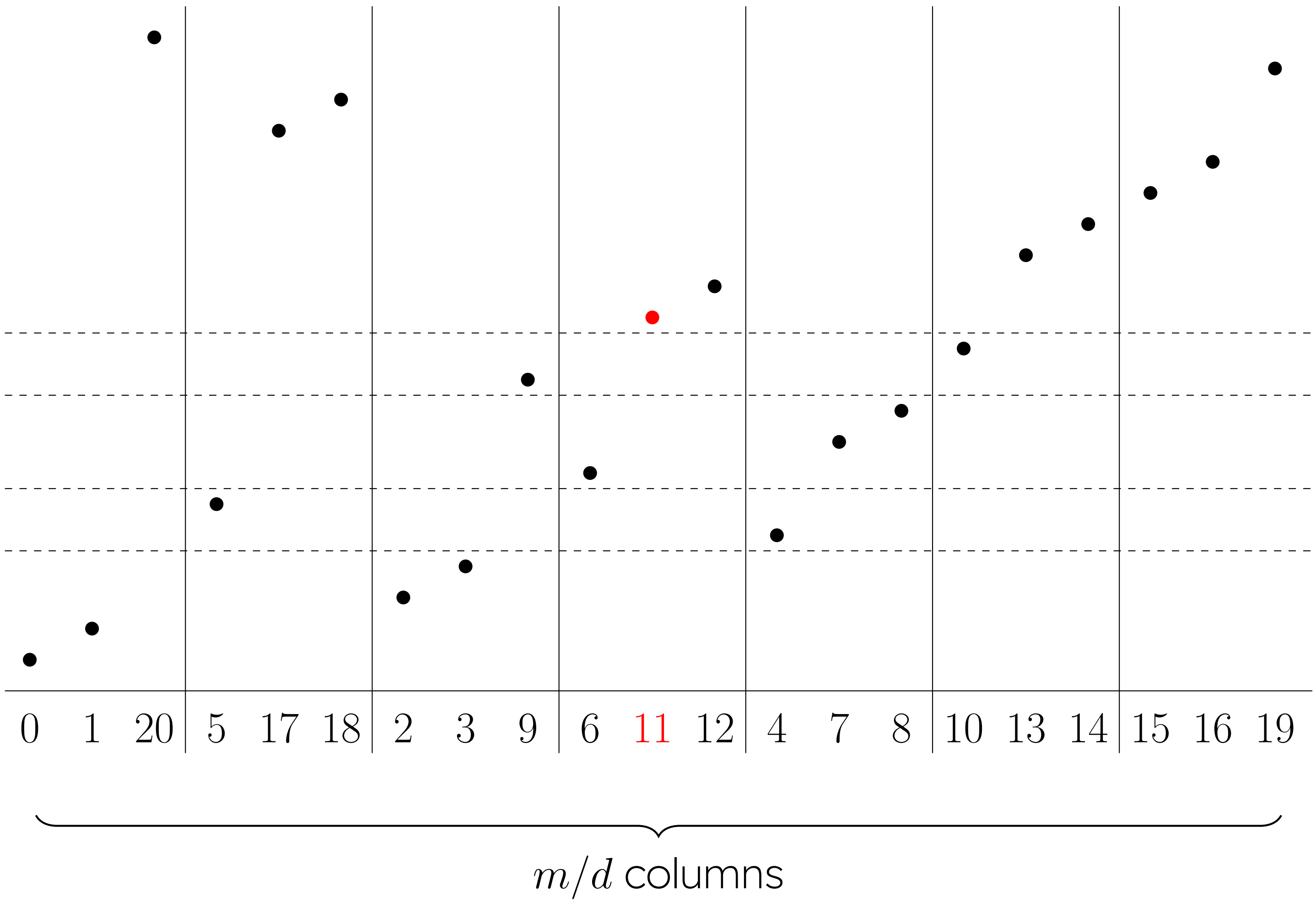
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



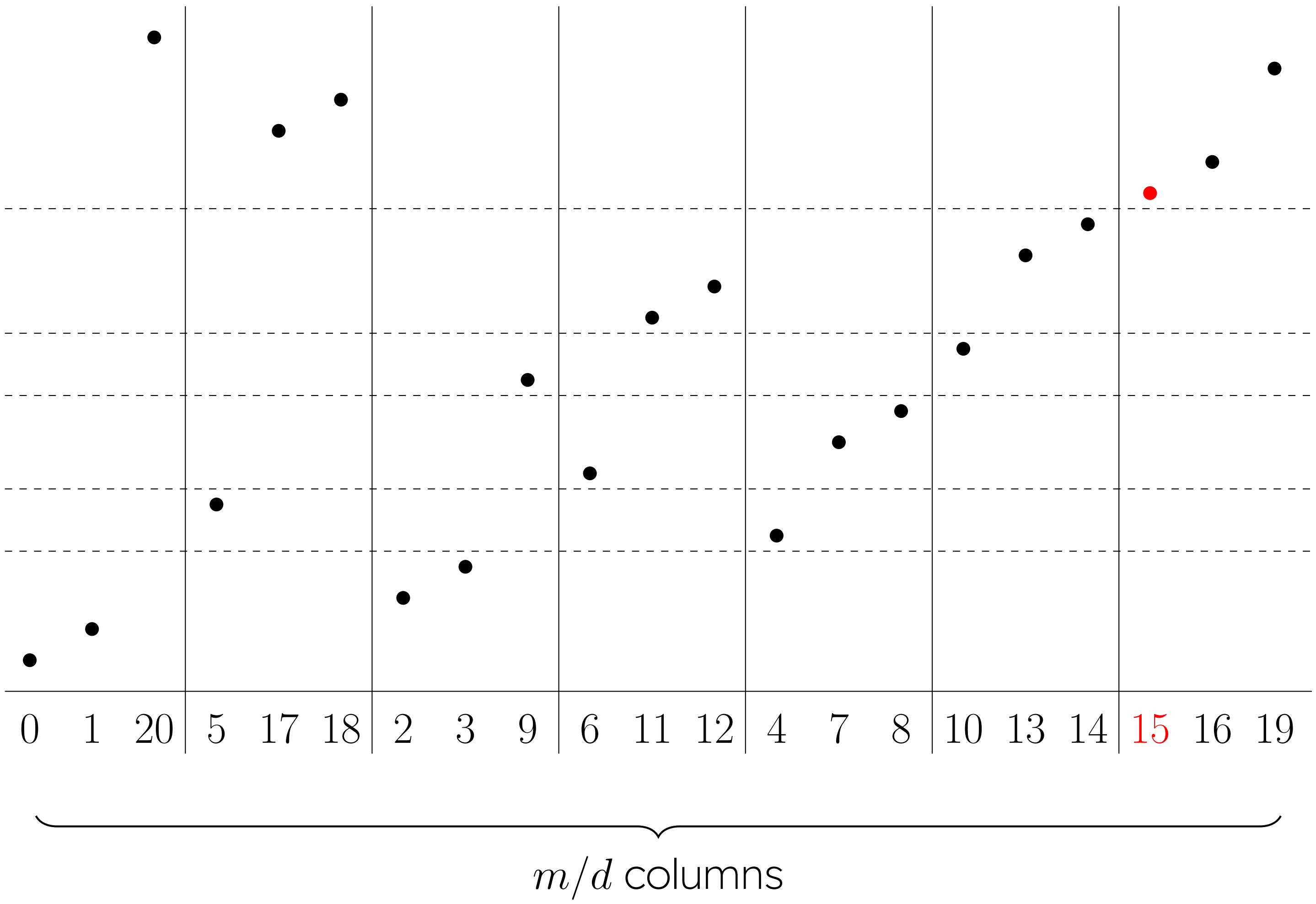
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



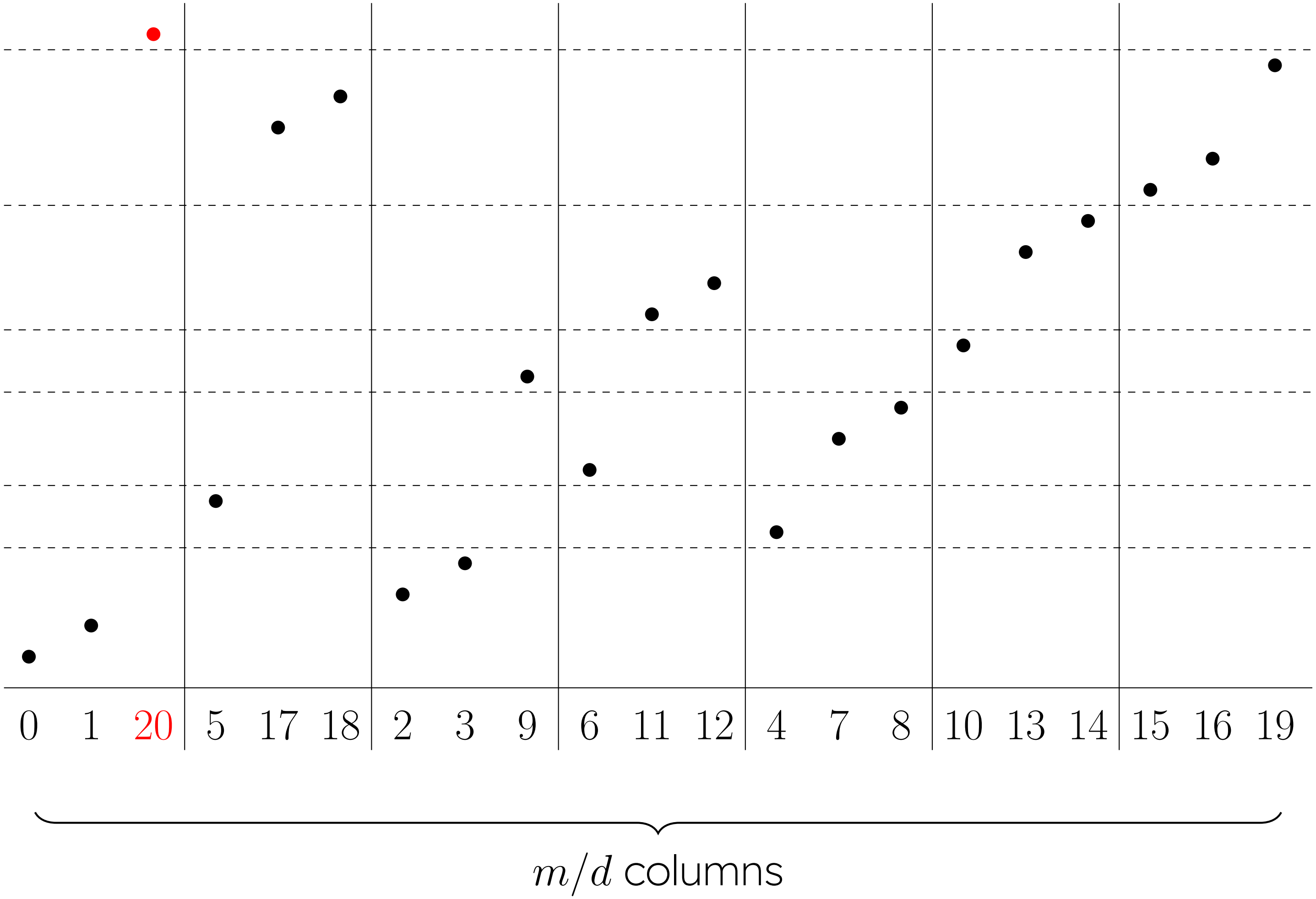
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



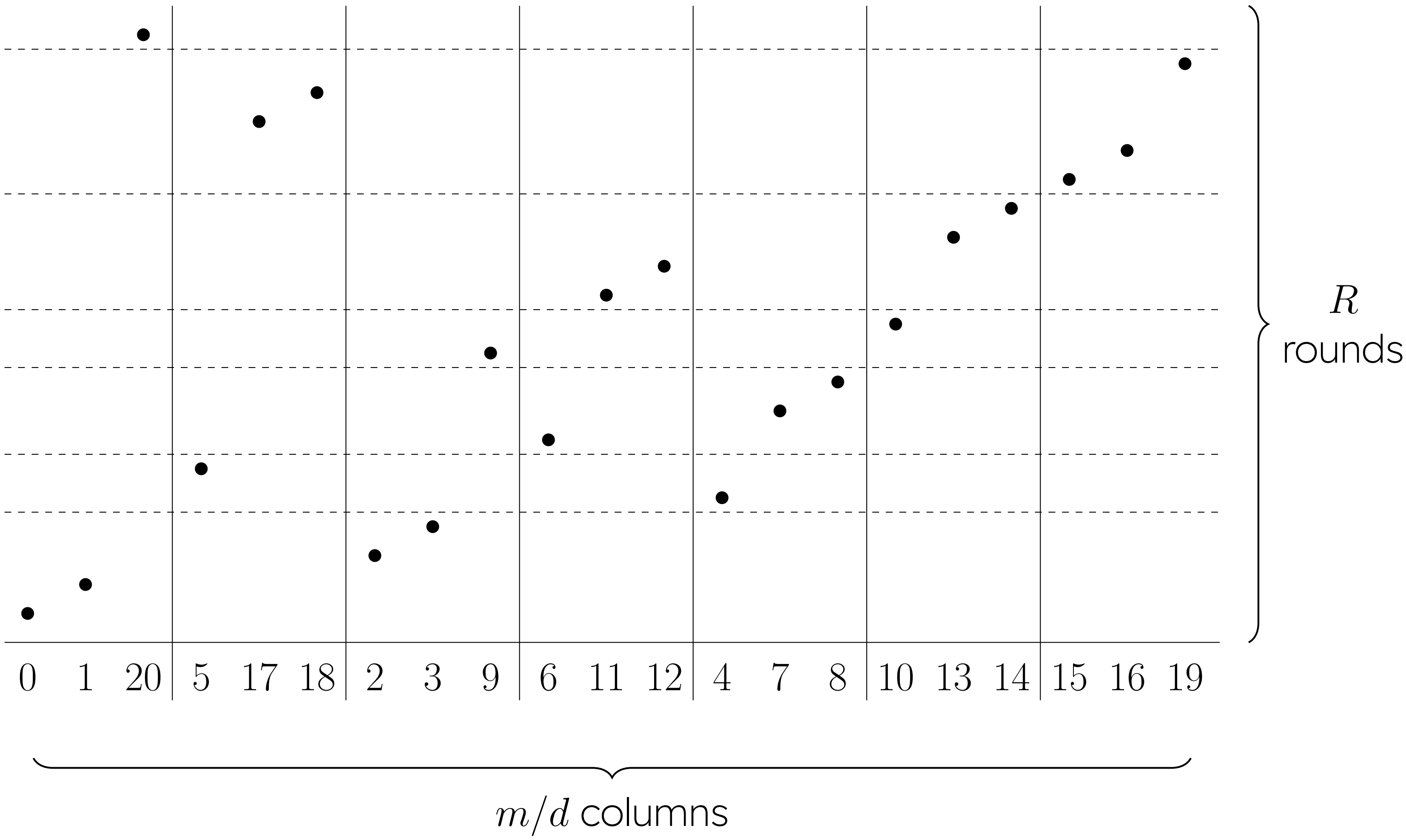
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



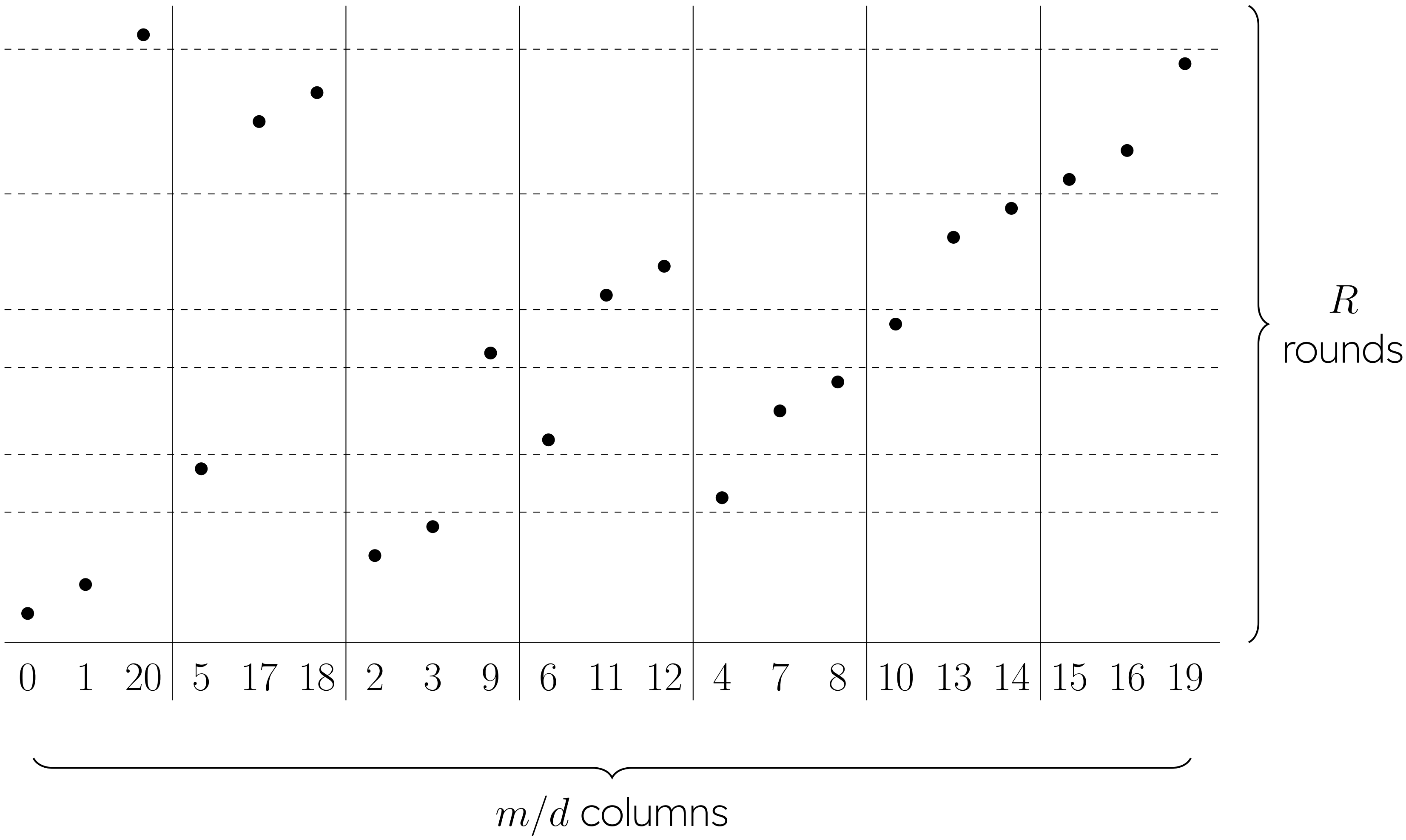
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



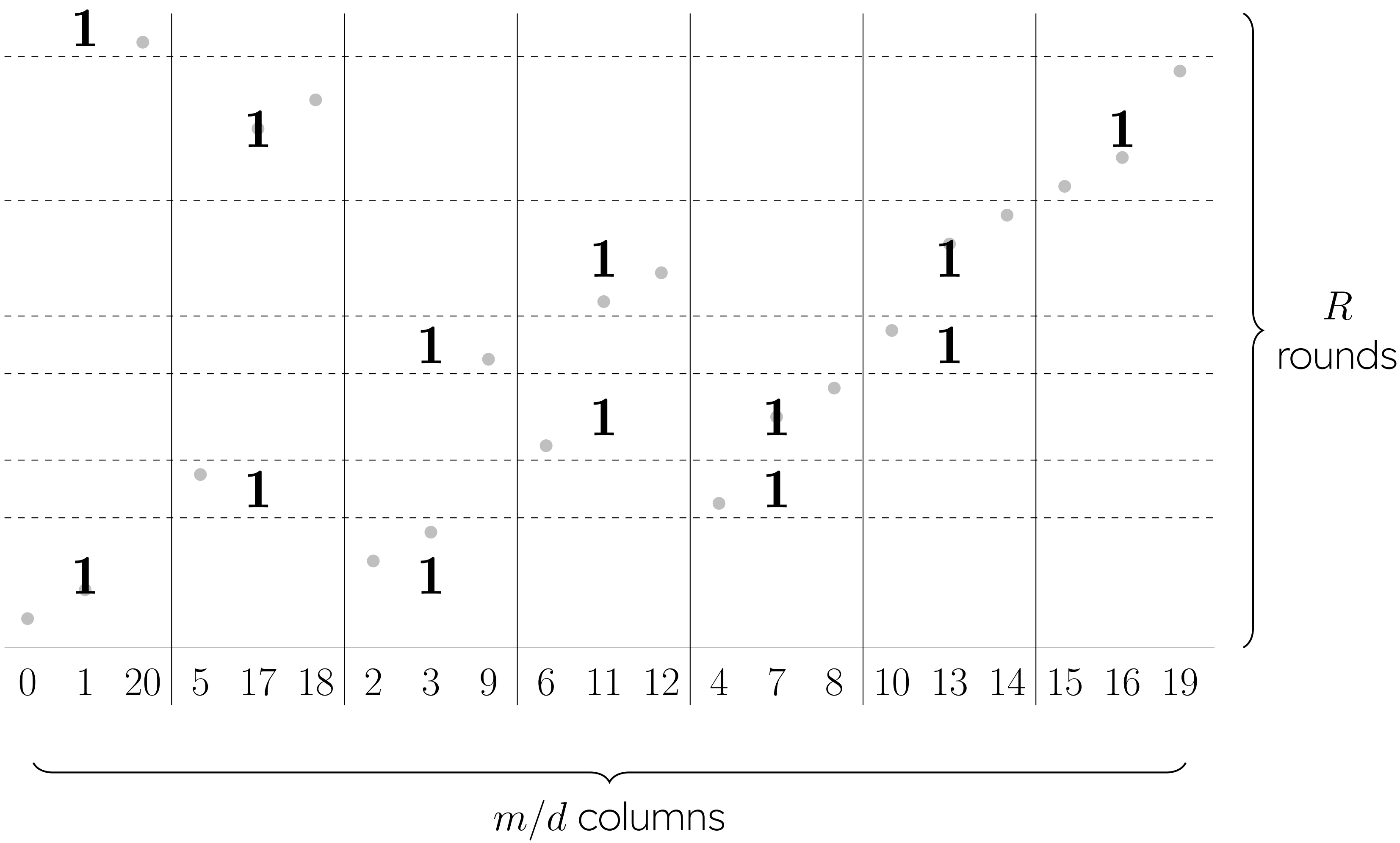
Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.



Claim: The algorithm terminates after at most $\frac{m}{d} = \frac{n}{d \log n}$ rounds.

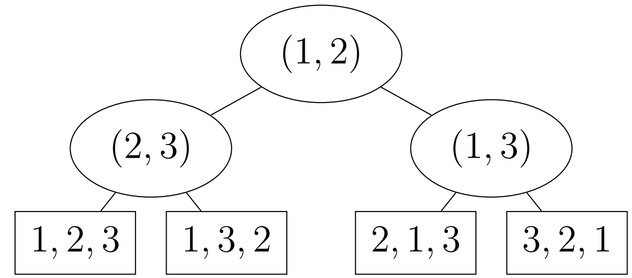


- Assume for a contradiction that $R > m/d$.
- # 1-entries $\geq d \cdot (R-1) \geq 2c_\pi \cdot (R-1) \geq c_\pi \cdot R = c_\pi \cdot \max(m/d, R)$
  $\Rightarrow$ the matrix contains π $\Rightarrow$ the input sequence S contains π.

(II) Sorting Many Short Sequences

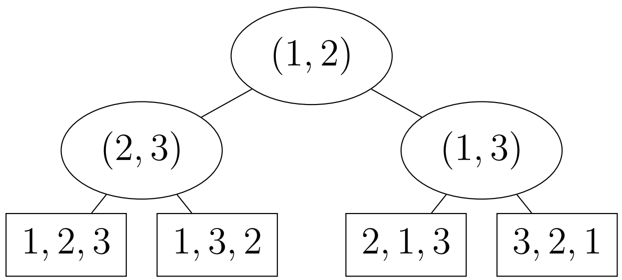
Brute Force with Decision Trees

- A decision tree T for input length k is a full binary tree with internal nodes labeled by elements of $[k]^2$ and leaves labeled by permutations of length k.



Brute Force with Decision Trees

- A decision tree T for input length k is a full binary tree with internal nodes labeled by elements of $[k]^2$ and leaves labeled by permutations of length k.



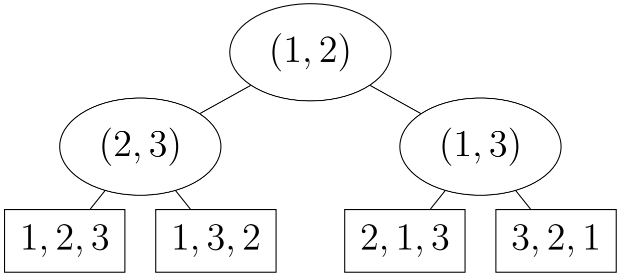
Lemma

There are at most $2^{2^{h+o(k)+o(h)}}$ decision trees of height at most h for input length k.

In our case $k = \log\log\log n$ and thus, there are $o(n)$ trees to consider.

Brute Force with Decision Trees

- A decision tree T for input length k is a full binary tree with internal nodes labeled by elements of $[k]^2$ and leaves labeled by permutations of length k.



Lemma

There are at most $2^{2^{h+o(k)+o(h)}}$ decision trees of height at most h for input length k.

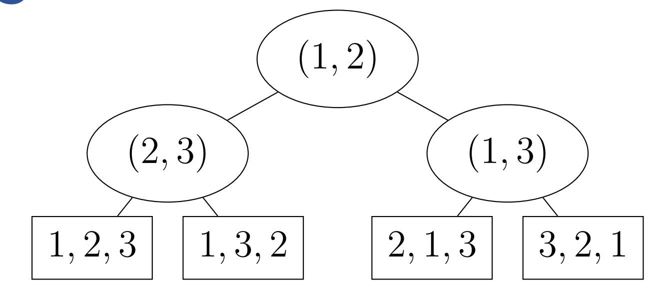
In our case $k = \log\log\log n$ and thus, there are $o(n)$ trees to consider.

Theorem [Fredman 1976] [Marcus, Tardos 2004]

For every pattern π, there exists a decision tree of depth $\mathcal{O}((\log s_\pi + 1) \cdot k)$ that sorts every π-avoiding permutation of length k.

Brute Force with Decision Trees

- A decision tree T for input length k is a full binary tree with internal nodes labeled by elements of $[k]^2$ and leaves labeled by permutations of length k.



Lemma

There are at most $2^{2^{h+o(k)+o(h)}}$ decision trees of height at most h for input length k.

In our case $k = \log\log\log n$ and thus, there are $o(n)$ trees to consider.

Theorem [Fredman 1976] [Marcus, Tardos 2004]

For every pattern π, there exists a decision tree of depth $\mathcal{O}((\log s_\pi + 1)\cdot k)$ that sorts every π-avoiding permutation of length k.

What if the algorithm knows π?

$P \leftarrow$ generate all π-avoiding permutations of length k
for every decision tree T of depth at most $\mathcal{O}((\log s_\pi + 1)\cdot k)$:
    if T sorts every permutation from P:
        $T_{\text{opt}} \leftarrow T$ and break
sort every sequence on input using T_{opt}.

$o(n)$

$\mathcal{O}((\log s_\pi + 1)\cdot k \cdot \frac{n}{k})$

Conclusion

Theorem

There is a comparison-based algorithm that sorts π-avoiding sequences of length n in $\mathcal{O}((\log s_\pi + 1) \cdot n)$ time even if π is a priori unknown.

Matches the information-theoretic lower bound!

Questions:

- Can we sort π-avoiding sequences with
  - $(\log_2 s_\pi + \mathcal{O}(1)) \cdot n$ comparisons, or even
  - $(\log_2 s_\pi + o(1)) \cdot n$ comparisons?
- Are there other problems where we can exploit pattern-avoidance algorithmically?

Conclusion

Theorem

There is a comparison-based algorithm that sorts π-avoiding sequences of length n in $\mathcal{O}((\log s_\pi + 1) \cdot n)$ time even if π is a priori unknown.

Matches the information-theoretic lower bound!

Questions:

- Can we sort π-avoiding sequences with
  - $(\log_2 s_\pi + \mathcal{O}(1)) \cdot n$ comparisons, or even
  - $(\log_2 s_\pi + o(1)) \cdot n$ comparisons?
- Are there other problems where we can exploit pattern-avoidance algorithmically?

Thank you!

Step I without Prior Knowledge of π

Input: π-avoiding sequence S partitioned into m presorted sequences $S_1, \ldots, S_m$

1. $d \leftarrow 1$
2. while there are some elements remaining:
3. $\quad S_1, \ldots, S_{\lfloor m/2 \rfloor} \leftarrow$merge consecutive pairs of sequences $\qquad \mathcal{O}(n)$
4. $\quad d \leftarrow 2 \cdot d, \quad m \leftarrow \lfloor m/2 \rfloor$
5. $\quad$ repeat m times or until all sequences are exhausted:
6. $\quad\quad S_{i_1}, \ldots, S_{i_{d+1}} \leftarrow d+1$ sequences with smallest initial elements
7. $\quad\quad x \leftarrow$ initial element of $S_{i_{d+1}}$ (largest out of these)

$\Big\}\ \mathcal{O}(n)$

8. $\quad\quad$ output merge $S_{i_1}, \ldots, S_{i_d}$ while smaller than x $\qquad \mathcal{O}(\log d_{max} \cdot n)$ overall

Phase = One execution of the outer loop (lines 2.–8.)

Step I without Prior Knowledge of π

Input: π-avoiding sequence S partitioned into m presorted sequences $S_1, \dots, S_m$

1. $d \leftarrow 1$
2. while there are some elements remaining:
3. $\quad S_1, \dots, S_{\lfloor m/2 \rfloor} \leftarrow$merge consecutive pairs of sequences $\qquad \mathcal{O}(n)$
4. $\quad d \leftarrow 2 \cdot d, \quad m \leftarrow \lfloor m/2 \rfloor$
5. $\quad$ repeat m times or until all sequences are exhausted:
6. $\qquad S_{i_1}, \dots, S_{i_{d+1}} \leftarrow d+1$ sequences with smallest initial elements $\qquad \mathcal{O}(n)$
7. $\qquad x \leftarrow$ initial element of $S_{i_{d+1}}$ (largest out of these)
8. $\qquad$ output merge $S_{i_1}, \dots, S_{i_d}$ while smaller than x $\qquad \mathcal{O}(\log d_{max} \cdot n)$ overall

Phase = One execution of the outer loop (lines 2.–8.)

Claim: The algorithm terminates after at most $\lceil \log c_\pi \rceil + 1$ phases.

Proof:

- Let d_i and m_i be values of d and m in the i^{th} phase.
- Observe that $d_i = 2^i$ and $m_i \leq \frac{n}{d_i \log n}$
- Assume that the algorithm reaches the phase $\lceil \log c_\pi \rceil + 1$.
- For $i = \lceil \log c_\pi \rceil + 1$, we have $d_i \geq 2c_\pi$ and the same argument as before shows that m_i rounds must suffice to finish merging all elements. $\square$

Step II without Prior Knowledge of π

Fix an enumeration of decision trees in the increasing order by their depth.

Input: sequences $S_1, \dots, S_{n/k}$, each of length k

1. $T \leftarrow$ first decision tree in the enumeration sequence
2. for $i \in \{1, \dots, n/k\}$:
3. $\quad S_i' \leftarrow$ rearrange S_i using T
4. $\quad$ while S_i' is not sorted:
5. $\quad\quad T \leftarrow$ next decision tree in the enumeration sequence
6. $\quad\quad S_i' \leftarrow$ rearrange S_i using T
7. $\quad$ output S_i'

Step II without Prior Knowledge of π

Fix an enumeration of decision trees in the increasing order by their depth.

Input: sequences $S_1, \ldots, S_{n/k}$, each of length k

1. $T \leftarrow$ first decision tree in the enumeration sequence
2. for $i \in \{1, \ldots, n/k\}$:
3. $S'_i \leftarrow$ rearrange S_i using T
4. while S'_i is not sorted:
5. $T \leftarrow$ next decision tree in the enumeration sequence
6. $S'_i \leftarrow$ rearrange S_i using T
7. output S'_i

Observation: The depth of the decision tree never exceeds $\mathcal{O}((\log s_\pi + 1) \cdot k)$.

We distinguish successful and unsuccessful applications of T on S_i (lines 3. and 6.).

Claim: In total, there are exactly n/k successful and $o(n)$ unsuccessful applications.
Proof:

- The sequence S_i is sorted after a successful application of $T \Rightarrow 1$ per sequence.
- Each unsuccessful application causes an advance in the enumeration of decision trees $\Rightarrow 1$ per decision tree, $o(n)$ in total. □