Analysis of Algorithms via Extremal Combinatorics

László Kozma
Freie Universität Berlin

Analysis of Algorithms, Bath, UK
June 2024
Extremal Combinatorics

Typical question (informally):
How large/dense/frequent can $X$ be if it avoids $Y$?
Typical question (informally):

How large/dense/frequent can $X$ be if it avoids $Y$?
How many edges can a graph of $n$ vertices have if it avoids $k$-cliques?

Answer: At most $\left(1 - \frac{1}{k} - \frac{1}{k^2}\right)n^2$. [Turán, 1941]
How many edges can a graph of $n$ vertices have if it avoids $k$-cliques?
How many edges can a graph of \( n \) vertices have if it avoids \( k \)-cliques?

**Answer:** At most \( \left( 1 - \frac{1}{k-1} \right) \frac{n^2}{2} \). [Turán, 1941]
How many edges can a graph of \( n \) vertices have if it avoids \( k \)-cliques?

**Answer:** At most \( \left(1 - \frac{1}{k-1}\right) \frac{n^2}{2} \). [Turán, 1941]
How many edges can a graph of $n$ vertices have if it avoids $k$-cliques?

**Answer:** At most \( \left( 1 - \frac{1}{k-1} \right) \frac{n^2}{2} \). [Turán, 1941]

This bound is sharp.
How many edges can a graph of $n$ vertices have if it avoids $k$-cliques?

**Answer**: At most $\left(1 - \frac{1}{k-1}\right) \frac{n^2}{2}$. [Turán, 1941]

This bound is sharp.
How many edges can a graph of \( n \) vertices have if it avoids \( k \)-cliques?

**Answer:** At most \( \left( 1 - \frac{1}{k-1} \right) \frac{n^2}{2} \). [Turán, 1941]
Forbidden matrices

How many •’s can an \( n \times n \) matrix have if it avoids the pattern \( P \)?
Forbidden matrices

How many ⬤′s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:
How many ⦿’s can an \( n \text{-by-} n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:
Forbidden matrices

How many ●’s can an \( n\)-by-\( n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:
Forbidden matrices

How many •’s can an \( n \times n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:

\[
\begin{array}{c}
\begin{array}{ccc}
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\end{array}
\end{array}
\]

\( \rightarrow \)

\( \left( \begin{array}{ccc}
\cdot & \cdot & \cdot \\
\end{array} \right), n \)

\( = \Theta \left( n^{3} \right) \)

\( \rightarrow \)

\( \left( \begin{array}{ccc}
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot \\
\end{array} \right), n \)

\( = \Theta \left( n \cdot \alpha \left( n \right) \right) \)

\( \rightarrow \)

related to Davenport-Schinzel sequences
Forbidden matrices

How many ●’s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:
Forbidden matrices

How many •’s can an \( n \times n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:
Forbidden matrices

How many •’s can an \( n \)-by-\( n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:
Forbidden matrices

How many •’s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:

- Matrix containing the pattern $\begin{array}{cc}
\bullet & \bullet \\
\end{array}$ with the pattern $P$
- Matrix containing the pattern $\begin{array}{ccc}
\bullet & \bullet & \\
\end{array}$ with the pattern $P$

- Matrix $\begin{array}{cccc}
\bullet & \bullet & \bullet & \\
\end{array}$ does not avoid $P$

Matrix pattern containment: $\begin{array}{cc}
\bullet & \bullet \\
\end{array}$

$\begin{array}{ccc}
\bullet & \bullet & \\
\end{array}$

$\begin{array}{cccc}
\bullet & \bullet & \bullet & \\
\end{array}$

$\begin{array}{cccc}
\bullet & \bullet & \bullet & \\
\end{array}$
Forbidden matrices

How many •’s can an \( n \)-by-\( n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:

\[
\begin{array}{c}
\text{ex}(P, n): \text{max number of •’s, while avoiding } P \ (0 \leq \text{ex}(P, n) \leq n^2)
\end{array}
\]
Forbidden matrices

How many ●’s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:

$\text{ex}(P, n)$: max number of ●’s, while avoiding $P$ ($0 \leq \text{ex}(P, n) \leq n^2$)

$\text{ex}\left(\left(\begin{array}{ll} ● & ● \\ ● & ● \end{array}\right), n\right) =$
Forbidden matrices

How many ●’s can an \( n \times n \) matrix have if it avoids the pattern \( P \)?

Matrix pattern containment:

\[
\begin{array}{c}
\text{ex}(P, n) : \text{max number of ●’s, while avoiding } P \ (0 \leq \text{ex}(P, n) \leq n^2) \\
\text{ex} \left( \left( \begin{array}{ll} \\
\end{array} \right), n \right) = \Theta(n^{3/2}) \quad \rightarrow \text{Zarankiewicz problem: bipartite graph avoiding } C_4
\end{array}
\]
Forbidden matrices

How many ⋅’s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:

ex($P, n$): max number of ⋅’s, while avoiding $P$ ($0 \leq \text{ex}(P, n) \leq n^2$)

ex $\left( \begin{pmatrix} ⋅ \cdot \\ ⋅ \cdot \end{pmatrix}, n \right) = \Theta(n^{3/2})$ → Zarankiewicz problem: bipartite graph avoiding $C_4$

ex $\left( \begin{pmatrix} ⋅ \\ ⋅ \end{pmatrix}, n \right) =$
Forbidden matrices

How many ⋄’s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:

ex$(P, n)$: max number of ⋄’s, while avoiding $P$ ($0 \leq \text{ex}(P, n) \leq n^2$)

$\text{ex}\left(\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}, n\right) = \Theta(n^{3/2}) \rightarrow \text{Zarankiewicz problem: bipartite graph avoiding } C_4$

$\text{ex}\left(\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}, n\right) = \Theta(n)$
How many ⋄'s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:

\[
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix},
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix},
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix}
\]

ex$(P, n)$: max number of ⋄'s, while avoiding $P$ ($0 \leq \text{ex}(P, n) \leq n^2$)

\[
\text{ex}
\left(
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix}, n
\right) = \Theta(n^{3/2}) \quad \rightarrow \text{Zarankiewicz problem: bipartite graph avoiding } C_4
\]

\[
\text{ex}
\left(
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix}, n
\right) = \Theta(n)
\]

\[
\text{ex}
\left(
\begin{bmatrix}
\begin{array}{ccc}
\bullet & \bullet & \bullet \\
\end{array}
\end{bmatrix}, n
\right) = \Theta(n)
\]
Forbidden matrices

How many ⋄’s can an $n$-by-$n$ matrix have if it avoids the pattern $P$?

Matrix pattern containment:

$\text{ex}(P, n)$: max number of ⋄’s, while avoiding $P$ ($0 \leq \text{ex}(P, n) \leq n^2$)

$\text{ex} \left( \left( \begin{array}{cc} \cdot & \cdot \\ \cdot & \cdot \end{array} \right), n \right) = \Theta(n^{3/2}) \rightarrow$ Zarankiewicz problem: bipartite graph avoiding $C_4$

$\text{ex} \left( \left( \begin{array}{c} \cdot \\ \cdot \end{array} \right), n \right) = \Theta(n)$

$\text{ex} \left( \left( \begin{array}{c} \cdot \\ \cdot \end{array} \right), n \right) = \Theta(n \cdot \alpha(n)) \rightarrow$ related to Davenport-Schinzel sequences
A broad class: permutation patterns, e.g., $P_\pi = \begin{pmatrix} \bullet \\ \bullet \end{pmatrix}$.

**Conjecture**  [Füredi-Hajnal 1992]

conjecture
A broad class: permutation patterns, e.g., \( P_\pi = \begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix} \).

**Conjecture** [Füredi-Hajnal 1992]

\[ \text{ex}(P_\pi, n) \in O_{\pi}(n), \text{ for any permutation } \pi. \]

\[ \rightarrow \text{linear in } n, \text{ for any fixed } \pi \]
A broad class: permutation patterns, e.g., $P_{\pi} = \begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}$.

**Conjecture** [Füredi-Hajnal 1992]

$\text{ex}(P_{\pi}, n) \in O_{\pi}(n)$, for any permutation $\pi$.  $ightarrow$ linear in $n$, for any fixed $\pi$

**The conjecture is true!** [Marcus, Tardos, 2004]
A broad class: permutation patterns, e.g., $P_{\pi} = \begin{pmatrix} \bullet & \bullet \end{pmatrix}$.

**Conjecture** [Füredi-Hajnal 1992]

$\text{ex}(P_{\pi}, n) \in O_{\pi}(n)$, for any permutation $\pi$. → linear in $n$, for any fixed $\pi$

**The conjecture is true!** [Marcus, Tardos, 2004]

$\text{ex}(P, n)$ also characterized for many other patterns:

→ can be

\[
\begin{aligned}
& n \\
& n \cdot \text{polylog}(n) \\
& n \cdot 2^{\alpha(n)} \\
& n^{1+\varepsilon} \\
& \ldots
\end{aligned}
\]
Use extremal combinatorics to:

I. Analyse algorithms

II. Model input structure
Use extremal combinatorics to:

I. Analyse algorithms

II. Model input structure
Example: Union-Find with path compression [Pettie, 2010]
Example: Union-Find with path compression [Pettie, 2010]

→ Collection of disjoint sets:

\text{find}(x): \text{return set containing } x
\text{union}(A, B): \text{merge } A \text{ and } B

→ Initially all singletons
Example: Union-Find with path compression [Pettie, 2010]

→ Collection of disjoint sets:

\[ \text{find}(x) \text{: return set containing } x \]
\[ \text{union}(A, B) \text{: merge } A \text{ and } B \]

→ Initially all singletons
Example: Union-Find with path compression [Pettie, 2010]

→ Collection of disjoint sets:

$\text{find}(x)$: return set containing $x$

$\text{union}(A, B)$: merge $A$ and $B$

→ Initially all singletons
**Example: Union-Find with path compression** [Pettie, 2010]

\[ \text{union}(B, C): \]

make the root of one tree the child of the other (arbitrarily)
**Example: Union-Find with path compression** [Pettie, 2010]

union($B, C$):
make the root of one tree the child of the other (arbitrarily)
**Example: Union-Find with path compression** [Pettie, 2010]

**union**$(B, C)$:
make the root of one tree the child of the other (arbitrarily)
**Example: Union-Find with path compression** [Pettie, 2010]

\[ \textbf{find}(x): \]
compress path from \( x \) to root

\[
\text{BuC}
\]
**Example: Union-Find with path compression** [Pettie, 2010]

\[ \text{find}(x): \]
compress path from \( x \) to root

![Diagram]

- \( v_0 \)
- \( v_1 \)
- \( v_2 \)
- \( v_3 \)
- \( x \)
Example: Union-Find with path compression [Pettie, 2010]

\textbf{find}(x):

compress path from $x$ to root
What is the (amortized) cost of operations?
→ View operations in single tree $T$
   (suppose all \textbf{unions} done upfront)

→ General path compression:
   $x \rightarrow y$ where $y$ is ancestor of $x$

→ Analyze cost of $n$ general path compressions in $T$
Example: Union-Find with path compression [Pettie, 2010]

→ Encode entire execution as an $n \times n$ matrix $M$
Example: Union-Find with path compression [Pettie, 2010]

→ Encode entire execution as an $n \times n$ matrix $M$
Example: Union-Find with path compression [Pettie, 2010]

→ Encode entire execution as an $n \times n$ matrix $M$

$M_{xj} = \bullet \iff$ node $x$ touched during $j$-th compression

**Total cost** = number of $\bullet$'s in $M$
Example: Union-Find with path compression [Pettie, 2010]

→ Encode entire execution as an $n \times n$ matrix $M$

Lemma: $M$ avoids $P = \left( \begin{array}{ccc} \bullet & \bullet & \bullet \end{array} \right)$. 
**Lemma:** \( M \) avoids \( P = (\bullet \bullet) \).

**Proof:** suppose not, then \( x \to y \to z \) on a path (because of postorder and as nodes cannot gains ancestors.

After \( i \)-th compress, \( x \) and \( y \) become unrelated, cannot be on \( j \)-th compress path together. \( \square \)
**Example: Union-Find with path compression** [Pettie, 2010]

**Lemma:** $M$ avoids $P = \begin{pmatrix} \bullet \bullet \\ \bullet \bullet \end{pmatrix}$.

**Proof:** suppose not, then $x \rightarrow y \rightarrow z$ on a path (because of postorder and as nodes cannot gain ancestors).

After $i$-th compress, $x$ and $y$ become unrelated, cannot be on $j$-th compress path together. □

\[
\implies \text{Cost} \leq \text{ex} \left( \begin{pmatrix} \bullet \bullet \\ \bullet \bullet \end{pmatrix}, n \right) \\
= n \log_2 n + O(n)
\]

[Tardos, 2005]
**Example: Union-Find with path compression** [Pettie, 2010]

**Lemma:** $M$ avoids $P = (\bullet \bullet \bullet \bullet)$.  

**Proof:** suppose not, then $x \rightarrow y \rightarrow z$ on a path (because of postorder and as nodes cannot gain ancestors).

After $i$-th compress, $x$ and $y$ become unrelated, cannot be on $j$-th compress path together. □

\[
\Rightarrow \text{Cost} \leq \text{ex} \left( \left( \bullet \bullet \bullet \bullet, n \right) \right)
\]

\[
= n \log_2 n + O(n)
\]

[Tardos, 2005]
Algorithm A → encode execution → 0/1 Matrix M

- Show that avoids some pattern P
- Bound on density of M = Bound on cost of A
Use extremal combinatorics to:

I. Analyse algorithms

II. Model input structure
Use extremal combinatorics to:

I. Analyse algorithms

II. Model input structure
Use extremal combinatorics to:

I. Analyse algorithms

II. Model input structure
Permutation patterns

Permutation $\tau$ contains permutation $\pi$: $\tau$ has a subsequence with the same ordering as $\pi$. (Otherwise $\tau$ avoids $\pi$.)

Example:

3 2 4 5 1 7 8 6 contains 1 2 4 3 because 2 5 7 6 $\sim$ 1 2 4 3 (both are like • • • •)

3 2 4 5 1 7 8 6 contains 1 2 3 4 because 2 4 5 7 $\sim$ 1 2 3 4 (both are like • • • •)

3 2 4 5 1 7 8 6 avoids 4 3 2 1
Permutation $\tau$ contains permutation $\pi$:

$\tau$ has a subsequence with the same ordering as $\pi$. 

Example:

$3 \ 2 \ 4 \ 5 \ 1 \ 7 \ 8 \ 6$ contains $1 \ 2 \ 4 \ 3$ because $2 \ 5 \ 7 \ 6 \ ∼ \ 1 \ 2 \ 4 \ 3$ (both are like • • • •)
Permutation \( \tau \) contains permutation \( \pi \): 
\( \tau \) has a subsequence with the same ordering as \( \pi \).

(Otherwise \( \tau \) avoids \( \pi \).)
Permutation patterns

Permutation $\tau$ contains permutation $\pi$: $\tau$ has a subsequence with the same ordering as $\pi$. (Otherwise $\tau$ avoids $\pi$.)

Example:
Permutation patterns

Permutation \( \tau \) contains permutation \( \pi \):
\( \tau \) has a subsequence with the same ordering as \( \pi \).

(Otherwise \( \tau \) avoids \( \pi \).)

Example:

\[
\begin{array}{cccccccc}
3 & 2 & 4 & 5 & 1 & 7 & 8 & 9 & 6 \\
\end{array}
\]
contains

\[
\begin{array}{cccc}
1 & 2 & 4 & 3 \\
\end{array}
\]
Permutation patterns

Permutation \( \tau \) contains permutation \( \pi \):
\( \tau \) has a subsequence with the same ordering as \( \pi \).
(Otherwise \( \tau \) avoids \( \pi \).)

Example:

\( 3 \, 2 \, 4 \, 5 \, 1 \, 7 \, 8 \, 6 \) contains \( 1 \, 2 \, 4 \, 3 \) because \( 2 \, 5 \, 7 \, 6 \) \sim \( 1 \, 2 \, 4 \, 3 \)
Permutation τ **contains** permutation π:
τ has a subsequence with the same ordering as π.
(Otherwise τ **avoids** π.)

Example:

3 2 4 5 1 7 8 6  **contains**  1 2 4 3  because 2 5 7 6 \sim 1 2 4 3
(both are like  \cdot \cdot )
Permutation pattern

Permutation \( \tau \) contains permutation \( \pi \):

\( \tau \) has a subsequence with the same ordering as \( \pi \).

(Otherwise \( \tau \) avoids \( \pi \).)

Example:

\[
3 \ 2 \ 4 \ 5 \ 1 \ 7 \ 8 \ 6 \ \text{contains} \ 1 \ 2 \ 4 \ 3 \ \text{because} \ 2 \ 5 \ 7 \ 6 \ \sim \ 1 \ 2 \ 4 \ 3 \\
\text{(both are like} \cdot \cdot \\
3 \ 2 \ 4 \ 5 \ 1 \ 7 \ 8 \ 6 \ \text{contains} \ 1 \ 2 \ 3 \ 4
\]
Permutation \( \tau \) **contains** permutation \( \pi \):
\( \tau \) has a subsequence with the same ordering as \( \pi \).

(Otherwise \( \tau \) **avoids** \( \pi \).)

**Example:**

3 2 4 5 1 7 8 6 **contains** 1 2 4 3 because 2 5 7 6 \( \sim \) 1 2 4 3

(both are like • • • •)

3 2 4 5 1 7 8 6 **contains** 1 2 3 4 because 2 4 5 7 \( \sim \) 1 2 3 4

(both are like • • • •)
Permutation $\tau$ contains permutation $\pi$: $\tau$ has a subsequence with the same ordering as $\pi$. (Otherwise $\tau$ avoids $\pi$.)

Example:

$3\ 2\ 4\ 5\ 1\ 7\ 8\ 6$ contains $1\ 2\ 4\ 3$ because $2\ 5\ 7\ 6 \sim 1\ 2\ 4\ 3$

(both are like \cdot \cdot \cdot)

$3\ 2\ 4\ 5\ 1\ 7\ 8\ 6$ contains $1\ 2\ 3\ 4$ because $2\ 4\ 5\ 7 \sim 1\ 2\ 3\ 4$

(both are like \cdot \cdot \cdot)

$3\ 2\ 4\ 5\ 1\ 7\ 8\ 6$ avoids $4\ 3\ 2\ 1$
Permutation patterns

\begin{array}{ccccccc}
2 & 7 & 4 & 1 & 5 & 3 & 6 \\
\end{array}

\text{contains} \quad 2 & 3 & 1
Permutation patterns

contains 2 3 1
Permutation patterns

contains 2 3 1
Examples

\[ \tau \text{ avoids } (2, 1) \iff \]

\[ \tau \text{ avoids } (k + 1, k, \ldots, 1) \iff \tau \text{ is } k \text{-increasing} \]

\[ \tau \text{ avoids } (2, 3, 1) \iff \tau \text{ is sortable with a stack} \]

\[ \tau \text{ avoids } (1, 3, 2) \text{ and } (3, 1, 2) \iff \text{every entry is a left-to-right min or max} \]
Examples

τ avoids (2, 1) ⇐⇒ τ is increasing

τ avoids (2, 3, 1) ⇐⇒ τ is sortable with a stack

τ avoids (1, 3, 2) and (3, 1, 2) ⇐⇒ every entry is a left-to-right min or max
Examples

\(\tau\) avoids \((2, 1)\) \iff \(\tau\) is increasing

\(\tau\) avoids \((k + 1, k, \ldots, 1)\) \iff \(\tau\) is \(k\)-increasing
Examples

\[ \tau \text{ avoids } (2, 1) \iff \tau \text{ is increasing} \]

\[ \tau \text{ avoids } (k + 1, k, \ldots, 1) \iff \tau \text{ is } k\text{-increasing} \]
Examples

\[ \tau \text{ avoids } (2, 1) \iff \tau \text{ is increasing} \]

\[ \tau \text{ avoids } (k + 1, k, \ldots, 1) \iff \tau \text{ is } k\text{-increasing} \]
Examples

\( \tau \) avoids \((2, 1)\) \iff \( \tau \) is increasing

\( \tau \) avoids \((k + 1, k, \ldots, 1)\) \iff \( \tau \) is \(k\)-increasing
Examples

\[ \tau \text{ avoids } (2, 1) \iff \tau \text{ is increasing} \]

\[ \tau \text{ avoids } (k + 1, k, \ldots, 1) \iff \tau \text{ is } k\text{-increasing} \]

\[ \tau \text{ avoids } (2, 3, 1) \iff \]
Examples

\( \tau \) avoids \((2, 1)\) \iff \( \tau \) is increasing

\( \tau \) avoids \((k + 1, k, \ldots, 1)\) \iff \( \tau \) is \(k\)-increasing

\( \tau \) avoids \((2, 3, 1)\) \iff \( \tau \) is sortable with a stack
Examples

\( \tau \) avoids \((2, 1)\) \iff \( \tau \) is increasing

\( \tau \) avoids \((k + 1, k, \ldots, 1)\) \iff \( \tau \) is \(k\)-increasing

\( \tau \) avoids \((2, 3, 1)\) \iff \( \tau \) is sortable with a stack

5. [M28] Show that it is possible to obtain a permutation \(p_1 p_2 \ldots p_n\) from \(12\ldots n\) using a stack if and only if there are no indices \(i < j < k\) such that \(p_j < p_k < p_i\).
Examples

\( \tau \) avoids \((2, 1) \) \iff \( \tau \) is increasing

\( \tau \) avoids \((k + 1, k, \ldots, 1) \) \iff \( \tau \) is \( k \)-increasing

\( \tau \) avoids \((2, 3, 1) \) \iff \( \tau \) is sortable with a stack

\begin{itemize}
\item 5. [M28] Show that it is possible to obtain a permutation \( p_1 p_2 \ldots p_n \) from 12\ldots n using a stack if and only if there are no indices \( i < j < k \) such that \( p_j < p_k < p_i \).  
\end{itemize}
Examples

\( \tau \) avoids \((2, 1)\) \iff \( \tau \) is increasing

\( \tau \) avoids \((k + 1, k, \ldots, 1)\) \iff \( \tau \) is \(k\)-increasing

\( \tau \) avoids \((2, 3, 1)\) \iff \( \tau \) is sortable with a stack
Examples

\[ \tau \text{ avoids } (2, 1) \iff \tau \text{ is increasing} \]

\[ \tau \text{ avoids } (k + 1, k, \ldots, 1) \iff \tau \text{ is } k\text{-increasing} \]

\[ \tau \text{ avoids } (2, 3, 1) \iff \tau \text{ is sortable with a stack} \]

\[ \tau \text{ avoids } (1, 3, 2) \text{ and } (3, 1, 2) \iff \]
Examples

\[ \tau \text{ avoids } (2, 1) \iff \tau \text{ is increasing} \]

\[ \tau \text{ avoids } (k + 1, k, \ldots, 1) \iff \tau \text{ is } k\text{-increasing} \]

\[ \tau \text{ avoids } (2, 3, 1) \iff \tau \text{ is sortable with a stack} \]

\[ \tau \text{ avoids } (1, 3, 2) \text{ and } (3, 1, 2) \iff \text{ every entry is a left-to-right min or max} \]
$\tau$ avoids $(2, 1) \iff \tau$ is increasing

$\tau$ avoids $(k + 1, k, \ldots, 1) \iff \tau$ is $k$-increasing

$\tau$ avoids $(2, 3, 1) \iff \tau$ is sortable with a stack

$\tau$ avoids $(1, 3, 2)$ and $(3, 1, 2) \iff$ every entry is a left-to-right min or max
How many permutations of length $n$ avoid a pattern $\pi$?
How many permutations of length $n$ avoid a pattern $\pi$?

**Conjecture**  [Stanley, Wilf, 1980s]
How many permutations of length $n$ avoid a pattern $\pi$?

**Conjecture** [Stanley, Wilf, 1980s]

At most $2^{O_\pi(n)}$. 

$\rightarrow$ single-exponential in $n$ ($\ll n!$)

$\rightarrow$ equivalent with the F"uredi-Hajnal conjecture on matrix density [Klazar, 2000]

Both conjectures are true! [Marcus, Tardos, 2004]
How many permutations of length $n$ avoid a pattern $\pi$?

**Conjecture** [Stanley, Wilf, 1980s]

At most $2^{O_\pi(n)}$. → single-exponential in $n$ ($\ll n!$)
How many permutations of length $n$ avoid a pattern $\pi$?

**Conjecture** [Stanley, Wilf, 1980s]
At most $2^{O_\pi(n)}$. → single-exponential in $n$ ($\ll n!$)

→ equivalent with the Füredi-Hajnal conjecture on matrix density

[Klazar, 2000]
How many permutations of length $n$ avoid a pattern $\pi$?

**Conjecture** [Stanley, Wilf, 1980s]

At most $2^{O_{\pi}(n)}$. → single-exponential in $n$ ($\ll n!$)

→ equivalent with the Füredi-Hajnal conjecture on matrix density [Klazar, 2000]

**Both conjectures are true!** [Marcus, Tardos, 2004]
The two results:

- At most \((s_\pi)^n\) permutations of length \(n\) that avoid \(\pi\).
  
  [Stanley-Wilf]

- The density of an \(n \times n\) matrix that avoids \(P_\pi\) is at most \(c_\pi \cdot n\).
  
  [Füredi-Hajnal]
The two results:

- At most \((s_\pi)^n\) permutations of length \(n\) that avoid \(\pi\).

  [Stanley-Wilf]

- The density of an \(n \times n\) matrix that avoids \(P_\pi\) is at most \(c_\pi \cdot n\).

  [Füredi-Hajnal]

\(\rightarrow\) \(c_\pi\) and \(s_\pi\) are polynomially related
The two results:

- At most \((s_\pi)^n\) permutations of length \(n\) that avoid \(\pi\).
  
  [Stanley-Wilf]

- The density of an \(n \times n\) matrix that avoids \(P_\pi\) is at most \(c_\pi \cdot n\).
  
  [Füredi-Hajnal]

\(\rightarrow\) \(c_\pi\) and \(s_\pi\) are polynomially related
A new algorithmic question:
A new algorithmic question:

- Consider some algorithmic problem with sequential input

Example: Sorting $n$ items via comparisons

Complexity: $\Theta(n \log n)$

Can we sort faster if input avoids some (arbitrary) fixed $\pi$?

Sort with $O(n)$ comparisons?

Sort in $O(n)$ time?

Want a general-purpose algorithm that is not tailored to each pattern $\pi$
A new algorithmic question:

- Consider some algorithmic problem with sequential input
- Does its complexity change if input avoids pattern $\pi$?
A new algorithmic question:

- Consider some algorithmic problem with sequential input
- Does its complexity change if input avoids pattern $\pi$?

Example: Sorting $n$ items via comparisons
A new algorithmic question:

- Consider some algorithmic problem with sequential input
- Does its complexity change if input avoids pattern $\pi$?

Example: Sorting $n$ items via comparisons

- Complexity: $\Theta(n \log n)$
A new algorithmic question:

- Consider some algorithmic problem with sequential input
- Does its complexity change if input avoids pattern $\pi$?

**Example: Sorting $n$ items via comparisons**

- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed $\pi$?
A new algorithmic question:

- Consider some algorithmic problem with sequential input
- **Does its complexity change if input avoids pattern** \( \pi \)?

Example: **Sorting** \( n \) **items via comparisons**

- Complexity: \( \Theta(n \log n) \)
- Can we sort faster if input avoids some (arbitrary) fixed \( \pi \)?
  - Sort with \( O(n) \) comparisons?
A new algorithmic question:

- Consider some algorithmic problem with sequential input
- **Does its complexity change if input avoids pattern** $\pi$?

Example: **Sorting** $n$ items via comparisons

- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed $\pi$?
  - Sort with $O(n)$ comparisons?
  - Sort in $O(n)$ time?
A new algorithmic question:
- Consider some algorithmic problem with sequential input
- **Does its complexity change if input avoids pattern** $\pi$?

**Example: Sorting $n$ items via comparisons**
- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed $\pi$?
  - Sort with $O(n)$ comparisons?
  - Sort in $O(n)$ time?
- Want a **general-purpose** algorithm that is not tailored to each pattern $\pi$
Why study this?
Why study this?

- Generalizes known easy inputs: 21-avoiding, 231-avoiding, etc.
Why study this?

- Generalizes known easy inputs: 21-avoiding, 231-avoiding, etc.
- There are \(2^{O(n)}\) such inputs \(\implies\) sorting lower bound just \(O(n)\)
Why study this?

- Generalizes known easy inputs: 21-avoiding, 231-avoiding, etc.
- There are $2^{O(n)}$ such inputs $\implies$ sorting lower bound just $O(n)$
- Make [Marcus-Tardos] results “algorithmic”
Why study this?

- Generalizes known easy inputs: 21-avoiding, 231-avoiding, etc.
- There are $2^{O(n)}$ such inputs $\implies$ sorting lower bound just $O(n)$
- Make [Marcus-Tardos] results “algorithmic”
- Analogy to avoided minors in graphs
  $\rightarrow$ sparsity, decompositions, efficient algorithms
Why study this?

- Generalizes known easy inputs: 21-avoiding, 231-avoiding, etc.
- There are $2^{O(n)}$ such inputs $\implies$ sorting lower bound just $O(n)$
- Make [Marcus-Tardos] results “algorithmic”
- Analogy to avoided minors in graphs
  $\rightarrow$ sparsity, decompositions, efficient algorithms
Example: Sorting

Idea:
Sort by inserting into a binary search tree (BST)
→ Use some dynamically balanced tree?
\[ O(\log n) \] per operation
\[ \Rightarrow \ O(n \log n) \] cost for sorting (too much!)
→ To achieve \( O(n) \), we need some adaptive BST, like Splay tree
**Example: Sorting**

**Idea:** Sort by inserting into a binary search tree (BST)
**Example: Sorting**

**Idea:** Sort by inserting into a binary search tree (BST)

→ Use some dynamically balanced tree?

\[ \text{Cost for sorting} \approx O(n \log n) \text{ (too much!)} \]

To achieve \( O(n) \), we need some adaptive BST, like Splay tree.
Example: Sorting

**Idea:** Sort by inserting into a binary search tree (BST)

→ Use some dynamically *balanced* tree?

\[ O(\log n) \text{ per operation} \iff O(n \log n) \text{ cost for sorting (too much!)} \]
Example: Sorting

**Idea:** Sort by inserting into a binary search tree (BST)

→ Use some dynamically balanced tree?

\[ O(\log n) \] per operation \( \Rightarrow \) \[ O(n \log n) \] cost for sorting (too much!)

→ To achieve \( O(n) \), we need some **adaptive** BST, like Splay tree
Self-adjusting tree: **Splay tree**  [Sleator, Tarjan, 1983]
Self-adjusting tree: **Splay tree** [Sleator, Tarjan, 1983]

When searching or inserting, rotate* the accessed element up, until it becomes the root.
Self-adjusting tree: Splay tree [Sleator, Tarjan, 1983]

When searching or inserting, rotate* the accessed element up, until it becomes the root.

* in a funny way.
Self-adjusting tree: **Splay tree** [Sleator, Tarjan, 1983]

When searching or inserting, rotate* the accessed element up, until it becomes the root.

* in a funny way.
Self-adjusting tree: **Splay tree**  
[Sleator, Tarjan, 1983]

When searching or inserting, rotate* the accessed element up, until it becomes the root.

* in a funny way.
Self-adjusting tree: **Splay tree**  [Sleator, Tarjan, 1983]

**III.6.1.2. Splay-Bäume**  249

![Diagram of Splay tree operations](image)

**Abb. 95. 4 Schütte-Operationen.**

[K. Mehlhorn: Data structures and algorithms (German ed. 1986)]
A matrix view of BSTs

[Demaine, Harmon, Iacono, Kane, Pătraşcu, SODA’09]
access sequence $X$
eq 4, 5, 6, 1, 2, 3
access sequence $X$
eq, 4, 5, 6, 1, 2, 3

$\rightarrow$ point set $X$
**access sequence** $X$
eq, 4, 5, 6, 1, 2, 3
→ point set $X$
access sequence $X$
eq 4, 5, 6, 1, 2, 3
→ point set $X$

dynamic BST serving $X$
**access sequence** $X$
eq e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set $X$

dynamic **BST** serving $X$

$\rightarrow$ point set $Y \supseteq X$
**access sequence** $X$
eq 4, 5, 6, 1, 2, 3

→ point set $X$

**dynamic BST** serving $X$

→ point set $Y \supseteq X$

↑ nodes touched by pointer
moves and rotations
**access sequence**  \( X \)
e.g., 4, 5, 6, 1, 2, 3
→ point set  \( X \)

**dynamic BST** serving  \( X \)
→ point set  \( Y \supseteq X \)
↑

nodes touched by pointer
moves and rotations

\( Y \) **is a BST execution of**  \( X \)  \iff \( Y \) **is a satisfied superset of**  \( X \)
**access sequence** $X$
eq 4, 5, 6, 1, 2, 3

→ point set $X$

**dynamic BST** serving $X$

→ point set $Y \supseteq X$

↑ nodes touched by pointer
moves and rotations

$Y$ is a **BST execution of** $X$ $\iff$ $Y$ is a **satisfied superset of** $X$
access sequence \( X \)
e.g., 4, 5, 6, 1, 2, 3
\( \rightarrow \) point set \( X \)

**dynamic BST** serving \( X \)
\( \rightarrow \) point set \( Y \supseteq X \)

nodes touched by pointer
moves and rotations

\( Y \) is a BST execution of \( X \) \iff \( Y \) is a satisfied superset of \( X \)

no \( a, b \in Y \) form an empty rectangle
**access sequence** $X$
eq 4, 5, 6, 1, 2, 3
→ point set $X$

dynamic BST serving $X$
→ point set $Y \supseteq X$

↑ nodes touched by pointer
moves and rotations

$Y$ is a BST execution of $X$ $\iff$ $Y$ is a satisfied superset of $X$

no $a, b \in Y$ form an empty rectangle

**access sequence** $X$
eq 4, 5, 6, 1, 2, 3
→ point set $X$

**dynamic BST** serving $X$
→ point set $Y \supseteq X$

nodes touched by pointer

moves and rotations

$Y$ is a BST execution of $X$ $\iff$ $Y$ is a satisfied superset of $X$

no $a, b \in Y$ form an empty rectangle
access sequence $X$
eq 4, 5, 6, 1, 2, 3
→ point set $X$

dynamic BST serving $X$
→ point set $Y \supseteq X$
↑

nodes touched by pointer
moves and rotations

$Y$ is a BST execution of $X$ \iff $Y$ is a satisfied superset of $X$

no $a, b \in Y$ form an empty rectangle

\[ \begin{array}{ccc}
\text{a} & \text{X} & \text{b} \\
\text{b} & \text{a} & \text{X} \\
\text{a} & \text{X} & \text{b} \\
\text{b} & \text{a} & \text{X}
\end{array} \]
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline

bottom-up.

Can be implemented as a

BST, similar to splay trees.

This describes an

insertion-sort

execution.

Task:

Bound the cost ↓

number of dots in matrix
A matrix view of BSTs

Suggests a natural algorithm:

...
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.
Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

- Geometric sweepline bottom-up.
- Can be implemented as a BST, similar to splay trees.

This describes an insertion-sort execution.

Task: Bound the cost ↓ number of dots in matrix.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.

This describes an **insertion-sort** execution.
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline
bottom-up.

Can be implemented as a
BST, similar to splay trees.

This describes an **insertion-sort** execution.

**Task:** Bound the cost
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.

This describes an insertion-sort execution.

**Task:** Bound the cost

down

number of dots in matrix
A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

Can be implemented as a BST, similar to splay trees.

This describes an insertion-sort execution.

**Task:** Bound the cost

\[ \downarrow \]

number of dots in matrix
Encode execution as a matrix
- $\bullet$ = input sequence (avoids $\pi$)
- $\bullet$ = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.
$\Rightarrow$
for various special cases $O(n)$ can be shown, e.g., for $\pi = k, \ldots, 1$. 
Encode execution as a matrix

- = input sequence (avoids π)
- = data structure operations
number of points (● + ●) = total cost
Encode execution as a matrix
- = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.
→ input-revealing gadget

- O($n$) can be shown, e.g., for $\pi = k, \ldots, 1$. 

- touched node
- input
- keys
Encode execution as a matrix

• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost
**Encode execution as a matrix**
- $\bullet =$ input sequence (avoids $\pi$)
- $\cdot =$ data structure operations
number of points ($\bullet + \cdot$) = total cost

**Key Lemma**
Follows from sweepline.

$\Rightarrow$ execution avoids

$$\begin{bmatrix}
\bullet \\
\cdot \\
\cdot \\
\cdot \\
\cdot
\end{bmatrix}$$
→ for various special cases $O(n)$ can be shown, e.g., for $\pi = k, \ldots, 1$. 
Encode execution as a matrix

- = input sequence (avoids \( \pi \))
- = data structure operations

number of points (● + ●) = total cost

Key Lemma

Follows from sweepline.

\[
\begin{bmatrix}
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\
\end{bmatrix}
\]

→ for various special cases \( O(n) \) can be shown, e.g., for \( \pi = k, \ldots, 1 \).
**Encode execution as a matrix**

- $\bullet = $ input sequence (avoids $\pi$)
- $\bullet = $ data structure operations
- number of points ($\bullet + \bullet$) = total cost

*Key Lemma*

If execution contains the pattern:

If execution contains the pattern:
Encode execution as a matrix

• = input sequence (avoids $\pi$)

• = data structure operations

number of points ($\bullet + \bullet$) = total cost

does not exist

Key Lemma
Encode execution as a matrix
• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Encode execution as a matrix
• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Encode execution as a matrix

• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.
Encode execution as a matrix

- = input sequence (avoids π)
- = data structure operations
number of points (- + -) = total cost

Key Lemma
Follows from sweepline.

→ input-revealing gadget
Encode execution as a matrix

• = input sequence (avoids $\pi$)
• = data structure operations

number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.

→ input-revealing gadget
Encode execution as a matrix
• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.

→ input-revealing gadget
Encode execution as a matrix
• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.

→ input-revealing gadget
Encode execution as a matrix

- input sequence (avoids $\pi$)
- data structure operations

number of points ($\bullet + \bullet$) = total cost

Key Lemma

Follows from sweepline.

$\rightarrow$ input-revealing gadget
Encode execution as a matrix
• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.

→ input-revealing gadget
input $X$ avoids $\left(\bullet, \bullet\right)$
Encode execution as a matrix

• = input sequence (avoids π)
• = data structure operations
number of points (• + •) = total cost

Key Lemma
Follows from sweepline.

→ input-revealing gadget
input X avoids \( \left( \begin{array}{c} • \\ • \end{array} \right) \)

\[ \Rightarrow \text{execution avoids} \left( \begin{array}{c} • \\ • \\ • \\ • \\ • \\ • \end{array} \right) \]
Encode execution as a matrix

- = input sequence (avoids $\pi$)
- = data structure operations

Number of points ($\bullet + \bullet$) = total cost

Key Lemma

Follows from sweepline.

\[
\text{input } X \text{ avoids } \left( \begin{array}{c} \bullet \\ \bullet \\ \bullet \end{array} \right) \\
\implies \text{execution avoids } \left( \begin{array}{c} \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{array} \right) \\
\implies \text{cost of sorting } X \text{ is } \leq n \cdot 2^{\text{poly}(\alpha(n))} \quad [\text{CGKMS'15}]
\]

Using [Klazar '00] [Keszegh '09]
Encode execution as a matrix
• = input sequence (avoids $\pi$)
• = data structure operations
number of points ($\bullet + \bullet$) = total cost

Key Lemma
Follows from sweepline.

input $X$ avoids $\pi$

$\Longrightarrow$ execution avoids $\pi \otimes (\bullet \bullet)$

$\Longrightarrow$ cost of sorting $X$ is $n \cdot 2^{\alpha(n)O(|\pi|)}$ [CGKMS’15]

using [Klazar ’00] [Keszegh ’09]
Encode execution as a matrix
• = input sequence (avoids \( \pi \))
• = data structure operations
number of points \((\bullet + \bullet)\) = total cost

Key Lemma
Follows from sweepline.

input \(X\) avoids \(\pi\)
\[ \implies \text{execution avoids } \pi \otimes (\bullet \bullet \bullet) \]
\[ \implies \text{cost of sorting } X \text{ is } n \cdot 2^{\alpha(n)O(|\pi|)} \quad \text{[CGKMS'15]} \]
using [Klazar '00] [Keszegh '09]

→ for various special cases \(O(n)\) can be shown, e.g., for \(\pi = k, \ldots, 1\).
Remarks:

Result obtained via general-purpose BST; pattern-avoidance only used in the analysis.

Can also do it via selection-sort with an adaptive heap.

Result relies on extremal function $\text{ex}(\pi \otimes (\ldots \ldots \ldots), n)$.

Recent improvement:

$n \cdot 2^\alpha(n) \rightarrow n \cdot 2^\alpha(n) + O(|\pi|^2)$ (tight)

[Chalermsook, Pettie, Ying.'23]

Can we get to $O(n^2)$? [BK'24] [Opler'24+]
Remarks:

- Result obtained via general-purpose BST; pattern-avoidance only used in the analysis
Remarks:

- Result obtained via general-purpose BST; pattern-avoidance only used in the analysis
- Can also do it via selection-sort with an adaptive heap [KS'18]
Remarks:

- Result obtained via general-purpose BST; pattern-avoidance only used in the analysis.
- Can also do it via selection-sort with an adaptive heap $[KS'18]$.
- Result relies on extremal function $\text{ex} (\pi \otimes (\ldots \cdot \cdot \cdot), n)$. 

$\text{Chalermsook, Pettie, Ying.'23}$
Remarks:

- Result obtained via general-purpose BST;
  pattern-avoidance only used in the analysis

- Can also do it via selection-sort with an adaptive heap [KS’18]

- Result relies on extremal function \( \text{ex} (\pi \otimes (\bullet \bullet \bullet), n) \)

**Recent improvement:** \( n \cdot 2^\alpha(n)^{O(|\pi|)} \rightarrow n \cdot 2^\alpha(n) + O(|\pi|^2) \) (tight)

[Chalermsook, Pettie, Ying.’23]
Remarks:

- Result obtained via general-purpose BST;
  pattern-avoidance only used in the analysis

- Can also do it via selection-sort with an adaptive heap [KS’18]

- Result relies on extremal function $\text{ex}(\pi \otimes (\cdot \cdot \cdot), n)$

**Recent improvement:** $n \cdot 2^{\alpha(n)O(|\pi|)} \rightarrow n \cdot 2^{\alpha(n)+O(|\pi|^2)}$ (tight)

[Chalermsook, Pettie, Ying.’23]
Remarks:

- Result obtained via general-purpose BST; pattern-avoidance only used in the analysis
- Can also do it via selection-sort with an adaptive heap [KS’18]
- Result relies on extremal function $\text{ex} \left( \pi \otimes \left( \bullet \cdot \bullet \right), n \right)$

**Recent improvement:** $n \cdot 2^{\alpha(n)O(|\pi|)} \rightarrow n \cdot 2^{\alpha(n)+O(|\pi|^2)}$ (tight) [Chalermsook, Pettie, Ying.’23]

**Can we get to $O(n)$?**
Remarks:

- Result obtained via general-purpose BST;
  pattern-avoidance only used in the analysis

- Can also do it via selection-sort with an adaptive heap [KS’18]

- Result relies on extremal function $\text{ex} \left( \pi \otimes (\cdot \cdot \cdot), n \right)$

  **Recent improvement:** $n \cdot 2^\alpha(n)O(|\pi|) \rightarrow n \cdot 2^\alpha(n)+O(|\pi|^2)$ (tight)
  [Chalermsook, Pettie, Ying.’23]

  **Can we get to $O(n)$?** yes: [BKO’24] [Opler’24+]
Example: TSP
Given $n$ points in $[0, 1]^2$, find TSP-tour of min length.
Given \( n \) points in \([0, 1]^2\), find TSP-tour of min length.
Example: TSP

Given \( n \) points in \([0, 1]^2\), find TSP-tour of min length.
Example: TSP

Given $n$ points in $[0, 1]^2$, find TSP-tour of min length.

Worst-case OPT length =
Example: TSP

Given \( n \) points in \([0, 1]^2\), find TSP-tour of min length.

Worst-case OPT length = \( O(\sqrt{n}) \).
Example: TSP

Given $n$ points in $[0, 1]^2$, find TSP-tour of min length.

Worst-case OPT length $= O(\sqrt{n})$. 
Example: TSP

Given $n$ points in $[0, 1]^2$, find TSP-tour of min length.

Worst-case OPT length = $O(\sqrt{n})$.

(0,0)  (1,1)

1/\sqrt{n}

(this bound is tight)
Given \( n \) points in \([0, 1]^2\), find TSP-tour of min length.
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.
Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

e.g. $\pi = (3, 2, 1)$ \hspace{1em} \text{cost} \in O(1)$
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

e.g. $\pi = (3, 2, 1)$ \hspace{1cm} cost $\in O(1)$
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

For arbitrary $\pi$? 
Example: TSP

Given \( n \) points in \([0, 1]^2\) avoiding \( \pi \), find TSP-tour of min length.
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.
Example: TSP

Given $n$ points in $[0, 1]^2$, avoiding $\pi$, find TSP-tour of min length.

Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])

Cost $f(n) \leq$ (tour between cells) + (tours within cells).
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])

Cost $f(n) \leq$ (tour between cells) + (tours within cells).

\[
f(n) \leq f(c_\pi \cdot \sqrt{n}) + c_\pi \cdot f(\sqrt{n}/c_\pi)
\]
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])

Cost $f(n) \leq (\text{tour between cells}) + (\text{tours within cells})$.

$$f(n) \leq f(c_\pi \cdot \sqrt{n}) + c_\pi \cdot f(\sqrt{n}/c_\pi)$$

$$\leq (\log n)^O(\log c_\pi) \ll \sqrt{n}.$$

[BKO'24]
Example: TSP

Given $n$ points in $[0, 1]^2$ avoiding $\pi$, find TSP-tour of min length.

Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])

Cost $f(n) \leq$ (tour between cells) + (tours within cells).

$$f(n) \leq f(c_\pi \cdot \sqrt{n}) + c_\pi \cdot f(\sqrt{n}/c_\pi)$$
$$\leq (\log n)^O(\log c_\pi) \ll \sqrt{n}. \quad [BKO'24]$$

$\rightarrow$ with more work, we can reduce to $O(c_\pi \cdot \log n)$. 
X is $\pi$-avoiding
$X$ is $\pi$-avoiding

$\implies$ twin-width$(X) \leq c_\pi$
$X$ is $\pi$-avoiding

$\implies$ twin-width$(X) \leq c_\pi \iff X$ has a $c_\pi$-wide merge-sequence

[Guillemot, Marx '14]
$X$ is $\pi$-avoiding
\[ \implies \quad \text{twin-width}(X) \leq c_\pi \quad \iff \quad X \text{ has a } c_\pi\text{-wide merge-sequence} \]
[Guillemot, Marx '14]

\[ \rightarrow \quad \text{Can use merge-sequence to construct } O_\pi(\log n) \text{ cost TSP tour. } \quad \text{[BKO '24]} \]
$X$ is $\pi$-avoiding

\[ \implies \text{twin-width}(X) \leq c\pi \iff X \text{ has a } c\pi \text{-wide merge-sequence} \]

[Guillemot, Marx ’14]

\[ \rightarrow \text{Can use merge-sequence to construct } O_\pi(\log n) \text{ cost TSP tour.} \quad [\text{BKO '24}] \]

\[ \rightarrow \text{Can use merge-sequence to construct } O_\pi(n) \text{ cost BST execution.} \quad [\text{BKO '24}] \]
$X$ is $\pi$-avoiding

$$\implies \text{twin-width}(X) \leq c_\pi \iff X \text{ has a } c_\pi\text{-wide merge-sequence}$$

[Guillemot, Marx ’14]

→ Can use merge-sequence to construct $O_\pi(\log n)$ cost TSP tour. [BKO ’24]

→ Can use merge-sequence to construct $O_\pi(n)$ cost BST execution. [BKO ’24]
Merge sequences

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge**: Replace two points/rectangles by their bounding box

**Merge sequence**: Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge**: Replace two points/rectangles by their bounding box

**Merge sequence**: Sequence of rectangle/point families obtained by successive merges
**Merge sequences**

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge**: Replace two points/rectangles by their bounding box

**Merge sequence**: Sequence of rectangle/point families obtained by successive merges
**Merge sequences**

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
Merge sequences

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
**Merge sequences**

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges
**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges

Two rectangles/points see each other if their projections on the x- or y-axis overlap.
Merge sequences

**Merge**: Replace two points/rectangles by their bounding box

**Merge sequence**: Sequence of rectangle/point families obtained by successive merges

Two rectangles/points **see each other** if their projections on the x- or y-axis overlap.

A rectangle family is **d-wide** if no rectangle/point sees more than \( d \) other rectangles/points.
Merge sequences

**Merge:** Replace two points/rectangles by their bounding box

**Merge sequence:** Sequence of rectangle/point families obtained by successive merges

Two rectangles/points **see each other** if their projections on the x- or y-axis overlap.

A rectangle family is **$d$-wide** if no rectangle/point sees more than $d$ other rectangles/points.

A merge sequence is **$d$-wide** if each of its rectangle families is $d$-wide.
$X$ is $\pi$-avoiding

\[ \implies \quad \text{twin-width}(X) \leq c_\pi \iff X \text{ has a } c_\pi\text{-wide merge-sequence} \]

[Guillemot, Marx '14]

\[ \rightarrow \text{Can use merge-sequence to construct } O_\pi(\log n) \text{ cost TSP tour.} \quad [\text{BKO '24}] \]

\[ \rightarrow \text{Can use merge-sequence to construct } O_\pi(n) \text{ cost BST execution.} \quad [\text{BKO '24}] \]
Let $X$ be $\pi$-avoiding.

\[ \text{twin-width}(X) \leq c_\pi \iff X \text{ has a } c_\pi\text{-wide merge-sequence} \]

[Guillemot, Marx '14]

\[ \rightarrow \text{Can use merge-sequence to construct } O_\pi(\log n) \text{ cost TSP tour.} \quad [\text{BKO '24}] \]

\[ \rightarrow \text{Can use merge-sequence to construct } O_\pi(n) \text{ cost BST execution.} \quad [\text{BKO '24}] \]

\[ \Rightarrow \text{Sorting can be implemented by } O(n)\text{-cost insertion-sort in BST...} \]
$X$ is $\pi$-avoiding

$\implies$ twin-width($X$) $\leq c\pi$ $\iff$ $X$ has a $c\pi$-wide merge-sequence

[Guillemot, Marx '14]

$\rightarrow$ Can use merge-sequence to construct $O_\pi(\log n)$ cost TSP tour. [BKO '24]

$\rightarrow$ Can use merge-sequence to construct $O_\pi(n)$ cost BST execution. [BKO '24]

$\implies$ Sorting can be implemented by $O(n)$-cost insertion-sort in BST...

...but this is computed offline in $O(n \log n)$ time, so no $O(n)$-time sort yet.
\[ X \text{ is } \pi\text{-avoiding} \]

\[ \implies \text{twin-width}(X) \leq c_\pi \iff X \text{ has a } c_\pi\text{-wide merge-sequence} \]

[Guillemot, Marx '14]

→ Can use merge-sequence to construct \( O_\pi(\log n) \) cost TSP tour.  [BKO '24]

→ Can use merge-sequence to construct \( O_\pi(n) \) cost BST execution.  [BKO '24]

\[ \implies \text{Sorting can be implemented by } O(n)\text{-cost insertion-sort in BST...} \]

...but this is computed offline in \( O(n \log n) \) time, so no \( O(n) \)-time sort yet.

\[ \implies \text{Splay tree achieves } O(n)\text{-time sorting assuming dynamic optimality conjecture} \]
\( X \) is \( \pi \)-avoiding

\[ \implies \text{twin-width}(X) \leq c_\pi \iff X \text{ has a } c_\pi \text{-wide merge-sequence} \]

[Guillemot, Marx '14]

\( \rightarrow \) Can use merge-sequence to construct \( O_\pi(\log n) \) cost TSP tour.  

[BKÖ '24]

\( \rightarrow \) Can use merge-sequence to construct \( O_\pi(n) \) cost BST execution.  

[BKÖ '24]

\( \implies \) Sorting can be implemented by \( O(n) \)-cost insertion-sort in BST...

...but this is computed offline in \( O(n \log n) \) time, so no \( O(n) \)-time sort yet.

\( \implies \) Splay tree achieves \( O(n) \)-time sorting assuming dynamic optimality conjecture

Very recent: \( O(n) \)-time sort of pattern-avoiding input via careful mergesort + forbidden submatrix analysis.  

[Opler '24+].
Conclusions

1. Extremal combinatorics used to analyse algorithms
   Examples mostly from data structures
   TODO: find more examples, when does it work?

2. Pattern-avoidance reduces complexity
data structures, sorting
   [CGKMS'15] [KS'18] [BK'24] [Opler'24]
geometric problems: TSP, MST, Steiner-tree, Manhattan-netw.
   [BK'24]
online problems: \(k\)-server
   [BK'24]
Matching lower bounds + stronger bounds for families of patterns
   TODO: find more examples, when does it work?
#1. Extremal combinatorics used to analyse algorithms
Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures

TODO: find more examples, when does it work?
#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- **TODO:** find more examples, when does it work?
Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- TODO: find more examples, when does it work?

#2. Pattern-avoidance reduces complexity
Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- **TODO:** find more examples, when does it work?

#2. Pattern-avoidance reduces complexity

- data structures, sorting [CGKMS'15] [KS'18] [BKO'24] [Opler'24]
#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- **TODO:** find more examples, when does it work?

#2. Pattern-avoidance reduces complexity

- data structures, sorting [CGKMS'15] [KS'18] [BKO'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [BKO'24]
Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- TODO: find more examples, when does it work?

#2. Pattern-avoidance reduces complexity

- data structures, sorting [CGKMS’15] [KS’18] [BKO’24] [Opler’24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [BKO’24]
- online problems: $k$-server [BKO’24]
Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- **TODO:** find more examples, when does it work?

#2. Pattern-avoidance reduces complexity

- data structures, sorting [CGKMS'15] [KS'18] [BKO'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [BKO'24]
- online problems: $k$-server [BKO'24]

- Matching lower bounds + stronger bounds for families of patterns $\pi$
Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- **TODO:** find more examples, when does it work?

#2. Pattern-avoidance reduces complexity

- data structures, sorting [CGKMS’15] [KS’18] [BKO’24] [Opler’24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [BKO’24]
- online problems: $k$-server [BKO’24]
- Matching lower bounds + stronger bounds for families of patterns $\pi$
- **TODO:** find more examples, when does it work?
#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures
- **TODO:** find more examples, when does it work?

#2. Pattern-avoidance reduces complexity

- data structures, sorting [CGKMS’15] [KS’18] [BKO’24] [Opler’24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [BKO’24]
- online problems: $k$-server [BKO’24]
- Matching lower bounds + stronger bounds for families of patterns $\pi$
- **TODO:** find more examples, when does it work?
References

*Optimization with pattern-avoiding input*
B. A. Berendsohn, L. Kozma, M. Opler. [STOC 2024]

*Smooth heaps and a dual view of self-adjusting data structures*
L. Kozma, T. Saranurak. [STOC 2018]

*Pattern-avoiding access in binary search trees*
P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, T. Saranurak. [FOCS 2015]
Optimization with pattern-avoiding input  
B. A. Berendsohn, L. Kozma, M. Opler. [STOC 2024]

Smooth heaps and a dual view of self-adjusting data structures  
L. Kozma, T. Saranurak. [STOC 2018]

Pattern-avoiding access in binary search trees  
P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, T. Saranurak. [FOCS 2015]

On an extremal problem in graph theory  
P. Turán. [Matematikai és Fizikai Lapok 1941]

Davenport-Schinzel theory of matrices  
Z. Füredi, P. Hajnal. [Discrete Mathematics 1992]

Excluded permutation matrices and the Stanley-Wilf conjecture  

On 0-1 matrices and small excluded submatrices  

Applications of Forbidden 0-1 Matrices to Search Tree and Path Compression-Based Data Structures  
S. Pettie. [SODA 2010]

Finding small patterns in permutations in linear time  
S. Guillemot, D. Marx. [SODA 2014]
Optimization with pattern-avoiding input
Benjamin Aram Berendsohn, L. Kozma, Michal Opler. [STOC 2024]
Optimization with pattern-avoiding input
Benjamin Aram Berendsohn, L. Kozma, Michal Opler. [STOC 2024]

...thanks...
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
   (repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
   (repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)

2. Execute next merge

3. Add all grid points in new rectangle

(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
   (repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
   (repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle

(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \(d\)-wide merge sequence of \(P\)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
   (repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: \( d \)-wide merge sequence of \( P \)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
   (repeat)
Arborially satisfied superset with merge sequences

Given: \(d\)-wide merge sequence of \(P\)

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)
Arborially satisfied superset with merge sequences

Given: $d$-wide merge sequence of $P$

1. Form grid from all points (and rectangle sides)
2. Execute next merge
3. Add all grid points in new rectangle
(repeat)

Claim:
(a) Result is arborally satisfied
(b) # of added points is $O(d^2 \cdot n)$ (proof now)
Claim: In every step, we add $O(d^2)$ points.
Claim: In every step, we add $O(d^2)$ points.

Proof idea: Rectangle sees $\leq d$ other rectangles/points

\[ \implies \leq 4d \text{ grid lines} \]
**Claim:** In every step, we add $O(d^2)$ points.

**Proof idea:** Rectangle sees $\leq d$ other rectangles/points

$\implies \leq 4d$ grid lines

**Conclusion:** The optimum for every input sequence of twin-width $d$ is $O(d^2 \cdot n)$.

$\implies$ The optimum for every $\pi$-avoiding input sequence is $O(c_\pi^2 \cdot n)$. 
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the \( i \)-th rectangle is at most \( \mathcal{O}(1/(n - i)) \). [NEW]

Every \( \pi \)-avoiding point set has a \( \mathcal{O}(c_\pi) \)-wide **distance-balanced** merge sequence.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $\Theta(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $\Theta(c_\pi)$-wide distance-balanced merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a distance-balanced merge sequence the width and height of the $i$-th rectangle is at most $O(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $O(c_\pi)$-wide distance-balanced merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $\mathcal{O}(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $\mathcal{O}(c_{\pi})$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $O(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $O(c_\pi)$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $\mathcal{O}(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $\mathcal{O}(c_\pi)$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a distance-balanced merge sequence the width and height of the \(i\)-th rectangle is at most \(O(1/(n - i))\). [NEW]

Every \(\pi\)-avoiding point set has a \(O(c_\pi)\)-wide distance-balanced merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the \(i\)-th rectangle is at most \(O\left(\frac{1}{n - i}\right)\). [NEW]

Every \(\pi\)-avoiding point set has a \(O(c_\pi)\)-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a distance-balanced merge sequence the width and height of the $i$-th rectangle is at most $O(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $O(c_\pi)$-wide distance-balanced merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $\mathcal{O}(1/(n-i))$. [NEW]

Every $\pi$-avoiding point set has a $\mathcal{O}(c_\pi)$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $O(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $O(c_{\pi})$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a distance-balanced merge sequence the width and height of the $i$-th rectangle is at most $O(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $O(c_\pi)$-wide distance-balanced merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $\mathcal{O}(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $\mathcal{O}(c_\pi)$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the \(i\)-th rectangle is at most \(\mathcal{O}\left(\frac{1}{n - i}\right)\). [NEW]

Every \(\pi\)-avoiding point set has a \(\mathcal{O}(c_\pi)\)-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.
Distance-balanced merge sequences for MST

In a **distance-balanced** merge sequence the width and height of the $i$-th rectangle is at most $\mathcal{O}(1/(n - i))$. [NEW]

Every $\pi$-avoiding point set has a $\mathcal{O}(c_\pi)$-wide **distance-balanced** merge sequence.

**Spanning tree construction:** Whenever we merge two rectangles, connect arbitrary points within them.

\[
\text{Length: } \sum_{i=1}^{n-1} \frac{1}{n-i} \approx \log n
\]