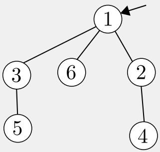
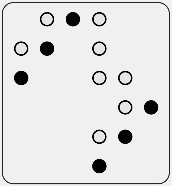
Analysis of Algorithms via Extremal Combinatorics

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?

Extremal Combinatorics

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]

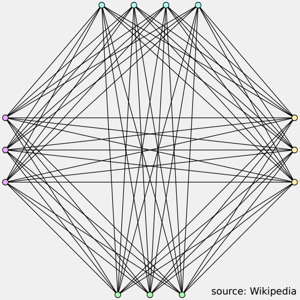
This bound is sharp.

Extremal Combinatorics

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.



source: Wikipedia

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]

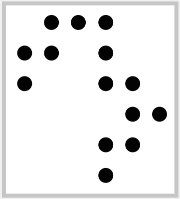
Forbidden matrices

How many •'s can an $\underline{n\text{-by-}n\text{ matrix}}$ have if it avoids the $\underline{\text{pattern } P}$?

Forbidden matrices

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

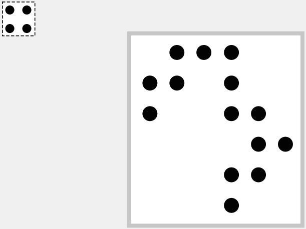
Matrix pattern containment:



Forbidden matrices

How many •'s can an $\underline{n\text{-by-}n \text{ matrix}}$ have if it avoids the $\underline{\text{pattern } P}$?

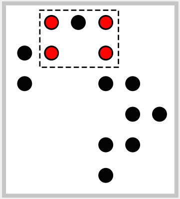
Matrix pattern containment:



Forbidden matrices

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

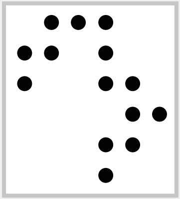
Matrix pattern containment:



Forbidden matrices

How many $\bullet$'s can an $\underline{n\text{-by-}n \text{ matrix}}$ have if it avoids the $\underline{\text{pattern } P}$?

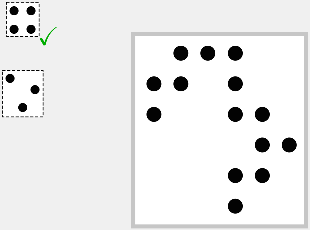
Matrix pattern containment:



Forbidden matrices

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

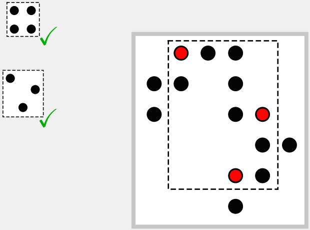
Matrix pattern containment:



Forbidden matrices

How many $\bullet$'s can an n-by-n matrix have if it avoids the pattern P?

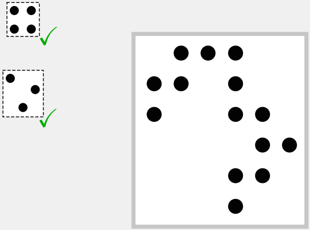
Matrix pattern containment:



Forbidden matrices

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

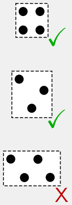
Matrix pattern containment:

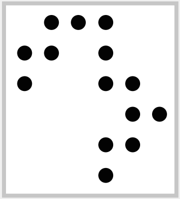


Forbidden matrices

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

Matrix pattern containment:

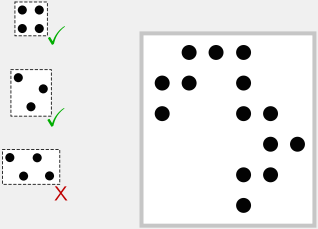




Forbidden matrices

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

Matrix pattern containment:

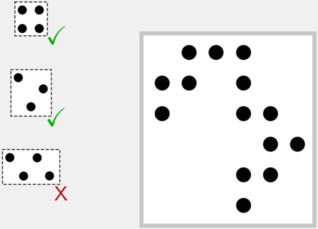


$\mathrm{ex}(P,n)$: max number of •'s, while avoiding P $(0 \leq \mathrm{ex}(P,n) \leq n^2)$

Forbidden matrices

How many $\bullet$'s can an n-by-n matrix have if it avoids the pattern P?

Matrix pattern containment:



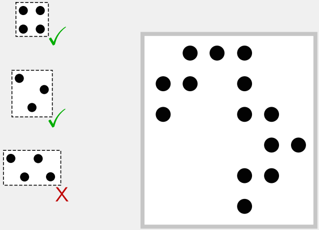
$\mathrm{ex}(P, n)$: max number of $\bullet$'s, while avoiding P $(0 \leq \mathrm{ex}(P, n) \leq n^2)$

$$\mathrm{ex}\left(\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}, n\right) =$$

Forbidden matrices

How many $\bullet$'s can an n-by-n matrix have if it avoids the pattern P?

Matrix pattern containment:



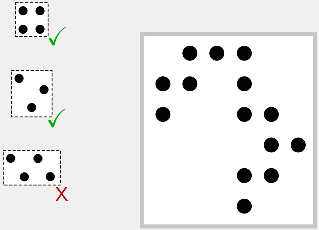
$\mathrm{ex}(P, n)$: max number of $\bullet$'s, while avoiding P $(0 \leq \mathrm{ex}(P, n) \leq n^2)$

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

Forbidden matrices

How many •'s can an $\underline{n\text{-by-}n \text{ matrix}}$ have if it avoids the $\underline{\text{pattern } P}$?

Matrix pattern containment:



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

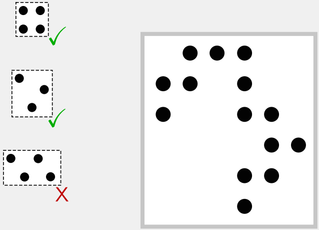
$\mathrm{ex}\left(\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}, n\right) = \Theta(n^{3/2})$ $\rightarrow$ Zarankiewicz problem: bipartite graph avoiding C_4

$\mathrm{ex}\left(\begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}, n\right) =$

Forbidden matrices

How many $\bullet$'s can an n-by-n matrix have if it avoids the pattern P?

Matrix pattern containment:



$\mathrm{ex}(P,n)$: max number of $\bullet$'s, while avoiding P $(0 \le \mathrm{ex}(P,n) \le n^2)$

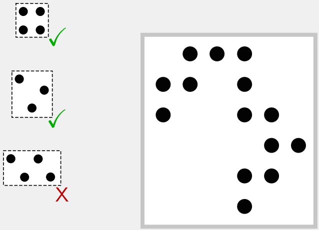
$\mathrm{ex}\left(\begin{pmatrix} \bullet & \bullet \\ \bullet & \bullet \end{pmatrix}, n\right) = \Theta(n^{3/2})$ $\rightarrow$ Zarankiewicz problem: bipartite graph avoiding C_4

$\mathrm{ex}\left(\begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}, n\right) = \Theta(n)$

Forbidden matrices

How many $\bullet$'s can an n-by-n matrix have if it avoids the pattern P?

Matrix pattern containment:



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

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

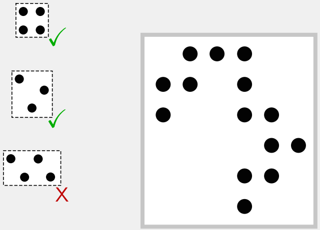
$\mathrm{ex}\left(\begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}, n\right) = \Theta(n)$

$\mathrm{ex}\left(\begin{pmatrix} \bullet & & \bullet & \\ & \bullet & & \bullet \end{pmatrix}, n\right) =$

Forbidden matrices

How many $\bullet$'s can an n-by-n matrix have if it avoids the pattern P?

Matrix pattern containment:



$\mathrm{ex}(P, n)$: max number of $\bullet$'s, while avoiding P $(0 \le \mathrm{ex}(P, n) \le n^2)$

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

$\mathrm{ex}\left(\begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}, n\right) = \Theta(n)$

$\mathrm{ex}\left(\begin{pmatrix} \bullet & & \bullet & \\ & \bullet & & \bullet \end{pmatrix}, n\right) = \Theta(n \cdot \alpha(n))$ $\rightarrow$ related to Davenport-Schinzel sequences

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

Conjecture [Füredi-Hajnal 1992]

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

Conjecture [Füredi-Hajnal 1992]

$\mathrm{ex}(P_\pi, n) \in O_\pi(n)$, for any permutation π. $\rightarrow$ linear in n, for any fixed π

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

Conjecture [Füredi-Hajnal 1992]

$\mathrm{ex}(P_\pi, n) \in O_\pi(n)$, for any permutation π. $\rightarrow$ linear in n, for any fixed π

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

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

Conjecture [Füredi-Hajnal 1992]

$\mathrm{ex}(P_\pi, n) \in O_\pi(n)$, for any permutation π. $\rightarrow$ linear in n, for any fixed π

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

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

$$\rightarrow \text{can be} \begin{cases} n \\ n \cdot \mathrm{polylog}(n) \\ n \cdot 2^{\alpha(n)} \\ n^{1+\varepsilon} \\ \dots \end{cases}$$

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]

$\rightarrow$ Collection of disjoint sets:

find(x): return set containing x
union(A, B): merge A and B

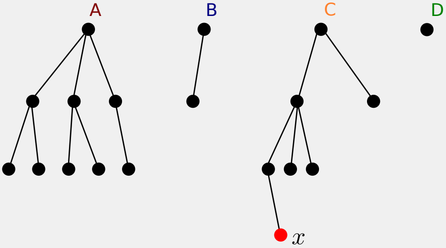
$\rightarrow$ Initially all singletons

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

→ Collection of disjoint sets:

find(x): return set containing x
union(A, B): merge A and B

→ Initially all singletons

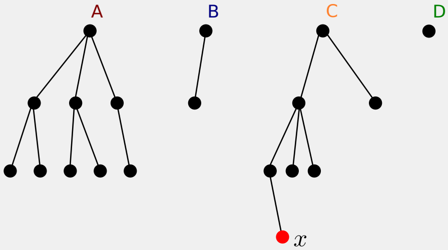


Example: Union-Find with path compression [Pettie, 2010]

$\rightarrow$ Collection of disjoint sets:

find(x): return set containing x
union(A, B): merge A and B

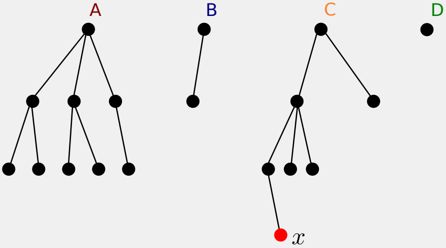
$\rightarrow$ Initially all singletons



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

union(B, C):

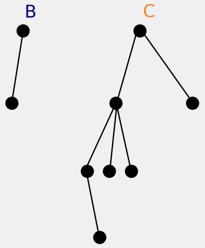
make the root of one tree the child of the other (arbitrarily)



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

union(B, C):

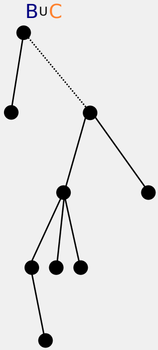
make the root of one tree the child of the other (arbitrarily)



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

union(B, C):

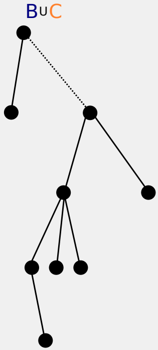
make the root of one tree the child of the other (arbitrarily)



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

find(x):

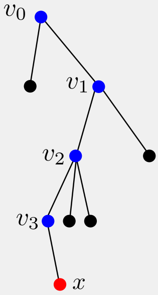
compress path from x to root



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

find(x):

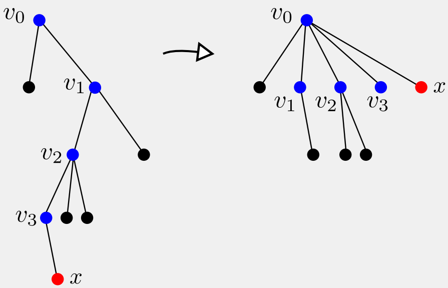
compress path from x to root



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

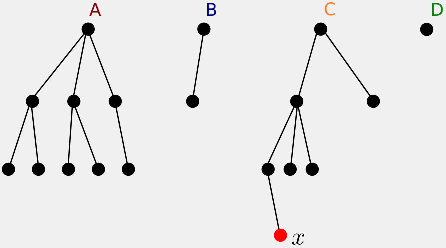
find(x):

compress path from x to root



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

What is the (amortized) cost of operations?

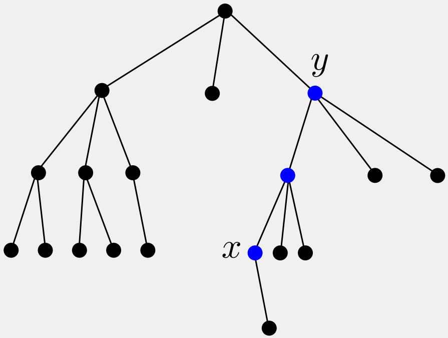


Example: Union-Find with path compression [Pettie, 2010]

$\rightarrow$ View operations in single tree T
(suppose all **union**s done upfront)

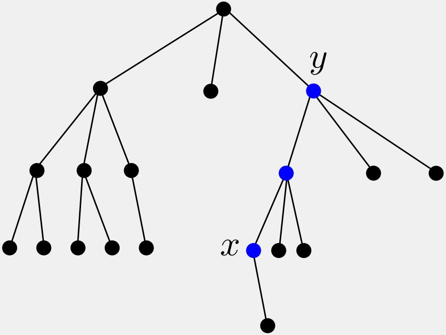
$\rightarrow$ General path compression:
$x \rightarrow y$ where y is ancestor of x

$\rightarrow$ Analyze cost of n general path compressions in T



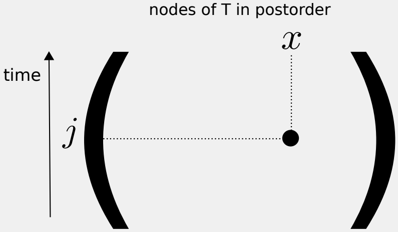
Example: Union-Find with path compression [Pettie, 2010]

→ 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

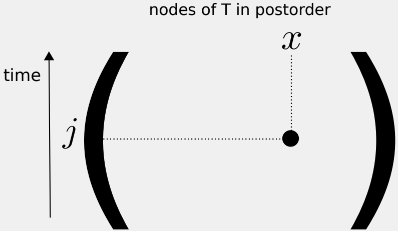


Example: Union-Find with path compression [Pettie, 2010]

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

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

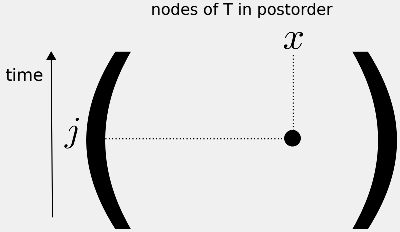
total cost $=$ number of $\bullet$'s in M



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

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

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

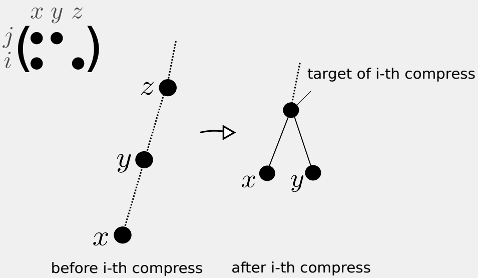


Example: Union-Find with path compression [Pettie, 2010]

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

Proof: suppose not, then $x \to y \to z$ 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. $\square$



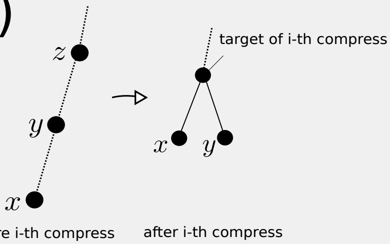
Example: Union-Find with path compression [Pettie, 2010]

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

Proof: suppose not, then $x \to y \to z$ 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. $\square$

$$\begin{array}{cccc} & x & y & z \\ j & \bullet & \bullet & \\ i & \bullet & & \bullet \end{array}$$



$$\implies \text{Cost} \le \mathrm{ex}\left(\begin{pmatrix} \bullet & \bullet & \\ \bullet & & \bullet \end{pmatrix}, n\right)$$
$$= n \log_2 n + O(n)$$

[Tardos, 2005]

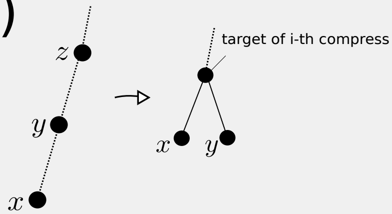
Example: Union-Find with path compression [Pettie, 2010]

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

Proof: suppose not, then $x \to y \to z$ 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. $\square$

$$\begin{array}{cc} & \begin{matrix} x & y & z \end{matrix} \\ \begin{matrix} j \\ i \end{matrix} & \begin{pmatrix} \bullet & \bullet & \\ \bullet & & \bullet \end{pmatrix} \end{array}$$

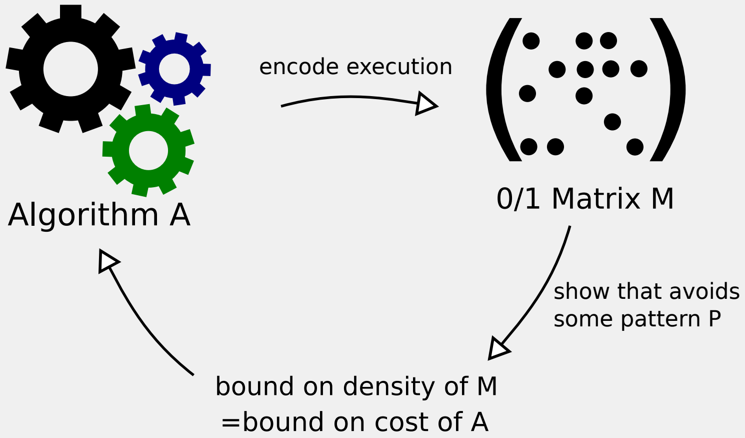


before i-th compress

after i-th compress

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

[Tardos, 2005]


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 patterns

Permutation τ **contains** permutation π:

τ has a subsequence with the same ordering as π.

Permutation patterns

Permutation τ **contains** permutation π:

τ has a subsequence with the same ordering as π.

(Otherwise τ **avoids** π.)

Permutation patterns

Permutation τ **contains** permutation π:

τ has a subsequence with the same ordering as π.

(Otherwise τ **avoids** π.)

Example:

Permutation patterns

Permutation τ **contains** permutation π:

τ has a subsequence with the same ordering as π.

(Otherwise τ **avoids** π.)

Example:

$3\ 2\ 4\ 5\ 1\ 7\ 8\ 9\ 6$ **contains** $1\ 2\ 4\ 3$

Permutation patterns

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 $2\ 5\ 7\ 6 \sim 1\ 2\ 4\ 3$

Permutation patterns

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 )

Permutation patterns

Permutation τ **contains** permutation π:

τ has a subsequence with the same ordering as π.

(Otherwise τ **avoids** π.)

Example:

$3\ {\color{red}2}\ 4\ {\color{red}5}\ 1\ {\color{red}7}\ 8\ {\color{red}6}$ **contains** ${\color{red}1\ 2\ 4\ 3}$ because ${\color{red}2\ 5\ 7\ 6} \sim {\color{red}1\ 2\ 4\ 3}$

(both are like )

$3\ {\color{red}2}\ {\color{red}4}\ {\color{red}5}\ 1\ {\color{red}7}\ 8\ 6$ **contains** ${\color{red}1\ 2\ 3\ 4}$

Permutation patterns

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 )

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

(both are like )

Permutation patterns

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 )

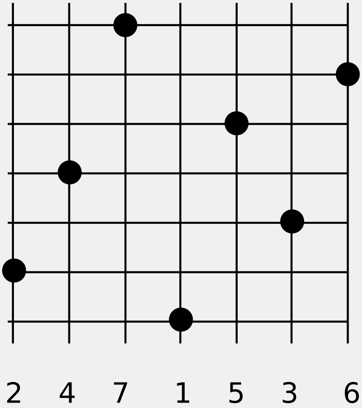
$3\ 2\ 4\ 5\ 1\ 7\ 8\ 6$ **contains** $1\ 2\ 3\ 4$ because $2\ 4\ 5\ 7 \sim 1\ 2\ 3\ 4$

(both are like )

$3\ 2\ 4\ 5\ 1\ 7\ 8\ 6$ **avoids** $4\ 3\ 2\ 1$

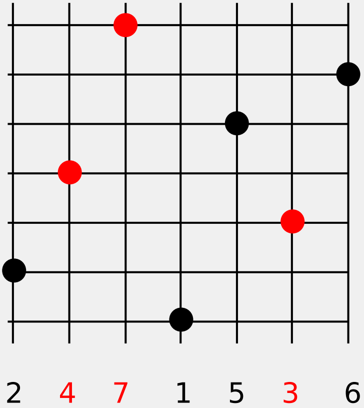
Permutation patterns

Permutation patterns



contains 2 3 1

Permutation patterns



contains 2 3 1

Examples

τ avoids $(2,1)$ $\iff$

Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

τ avoids $(k+1,k,\ldots,1)$ $\iff$

Examples

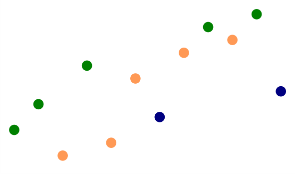
τ avoids $(2, 1)$ $\iff$ τ is increasing

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

Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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



Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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

Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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

τ avoids $(2,3,1)$ $\iff$

Examples

τ avoids $(2, 1)$ $\iff$ τ is increasing

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

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

Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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

τ avoids $(2,3,1)$ $\iff$ τ 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 $p_j < p_k < p_i$.

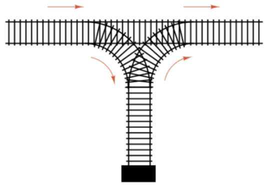
Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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

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

▶ **5.** [*M28*] Show that it is possible to obtain a permutation $p_1 p_2 \ldots p_n$ from $1\,2 \ldots n$ using a stack if and only if there are no indices $i < j < k$ such that $p_j < p_k < p_i$.



Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

τ avoids $(k+1,k,\dots,1)$ $\iff$ τ is k-increasing

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

Examples

τ avoids $(2, 1)$ $\iff$ τ is increasing

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

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

τ avoids $(1, 3, 2)$ and $(3, 1, 2)$ $\iff$

Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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

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

τ avoids $(1,3,2)$ and $(3,1,2)$ $\iff$ every entry is a left-to-right min or max

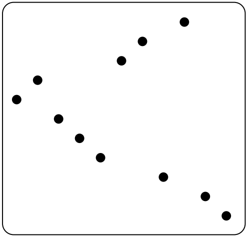
Examples

τ avoids $(2,1)$ $\iff$ τ is increasing

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

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

τ 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 π?

How many permutations of length n avoid a pattern π?

Conjecture [Stanley, Wilf, 1980s]

How many permutations of length n avoid a pattern π?

Conjecture [Stanley, Wilf, 1980s]

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

How many permutations of length n avoid a pattern π?

Conjecture [Stanley, Wilf, 1980s]

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

How many permutations of length n avoid a pattern π?

Conjecture [Stanley, Wilf, 1980s]

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

$\rightarrow$ equivalent with the Füredi-Hajnal conjecture on matrix density

[Klazar, 2000]

How many permutations of length n avoid a pattern π?

Conjecture [Stanley, Wilf, 1980s]

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

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

$\rightarrow$ 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 π.
  [Stanley-Wilf]

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

The two results:

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

  [Stanley-Wilf]

- The density of an $n \times n$ matrix that avoids P_π is at most $c_\pi \cdot n$.

  [Füredi-Hajnal]

$\rightarrow c_\pi$ and s_π are polynomially related

The two results:

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

  [Stanley-Wilf]

- The density of an $n \times n$ matrix that avoids P_π is at most $c_\pi \cdot n$.

  [Füredi-Hajnal]

$\rightarrow c_\pi$ and s_π are polynomially related

A new algorithmic question:

A new algorithmic question:

- Consider some algorithmic problem with sequential input

A new algorithmic question:

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

A new algorithmic question:

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

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 π?**

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 π?**

Example: Sorting n items via comparisons

- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed π?

A new algorithmic question:

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

Example: Sorting n items via comparisons

- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed π?
  - Sort with $O(n)$ comparisons?

A new algorithmic question:

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

Example: Sorting n items via comparisons

- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed π?
  - 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 π?**

Example: Sorting n items via comparisons

- Complexity: $\Theta(n \log n)$
- Can we sort faster if input avoids some (arbitrary) fixed π?
  - Sort with $O(n)$ comparisons?
  - Sort in $O(n)$ time?
- Want a general-purpose algorithm that is not tailored to each pattern π

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
  $\to$ 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
  $\to$ sparsity, decompositions, efficient algorithms

Example: Sorting

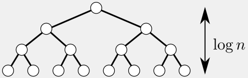
Example: Sorting

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

Example: Sorting

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

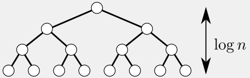
$\rightarrow$ Use some dynamically balanced tree?



Example: Sorting

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

$\rightarrow$ Use some dynamically balanced tree?

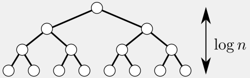


$O(\log n)$ per operation $\implies$ $O(n \log n)$ cost for sorting (**too much!**)

Example: Sorting

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

$\rightarrow$ Use some dynamically balanced tree?



$\boxed{O(\log n)}$ per operation $\implies$ $\boxed{O(n \log n)}$ cost for sorting (**too much!**)

$\rightarrow$ 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.

---

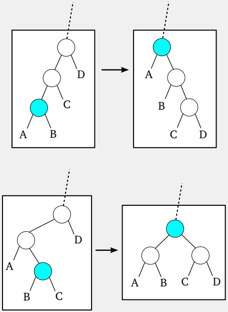
in a funny way.

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**

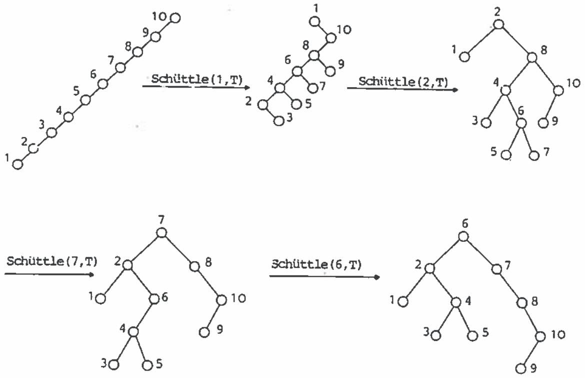


Abb. 95. 4 Schüttle-Operationen.

[K. Mehlhorn: Data structures and algorithms (German ed. 1986)]

A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

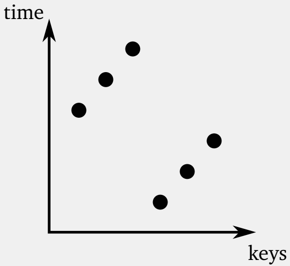
access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

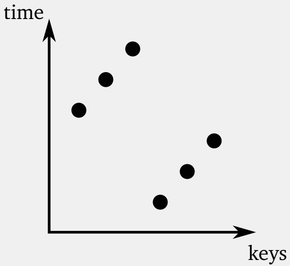


A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătraşcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

dynamic BST serving X



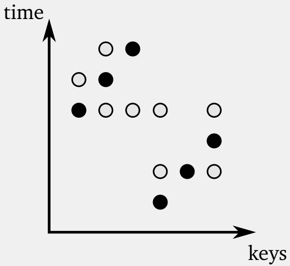
A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătraşcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$



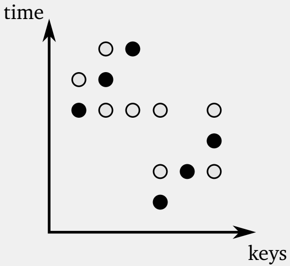
A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer
moves and rotations



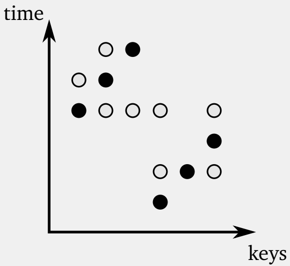
A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer
moves and rotations



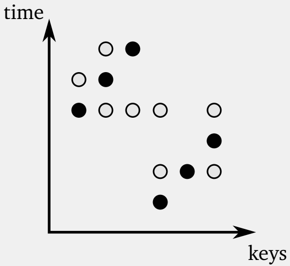
Y is a BST execution of X $\iff$ Y is a satisfied superset of X

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer moves and rotations



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

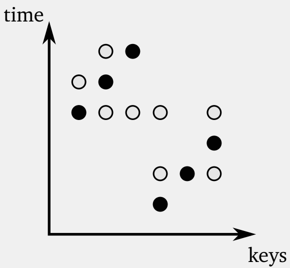
A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer moves and rotations



Y **is a BST execution of** X $\iff$ Y **is a satisfied superset of** X
$\downarrow$
no $a, b \in Y$ form an empty rectangle

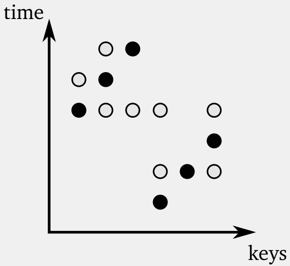
A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătrașcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

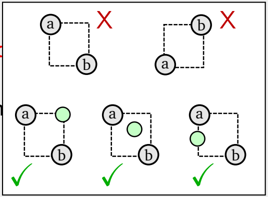
$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer
moves and rotations



Y **is a BST execution of** X $\iff$ Y **is a satisfied**
$\downarrow$
no $a, b \in Y$ form an em



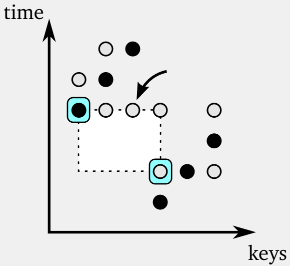
A matrix view of BSTs [Demaine, Harmon, Iacono, Kane, Pătraşcu, SODA'09]

access sequence X
e.g., 4, 5, 6, 1, 2, 3

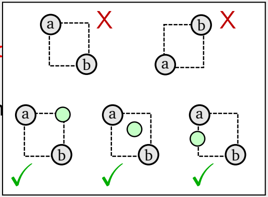
$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer moves and rotations



Y **is a BST execution of** X $\iff$ Y **is a satisfie**
$\downarrow$
no $a, b \in Y$ form an em

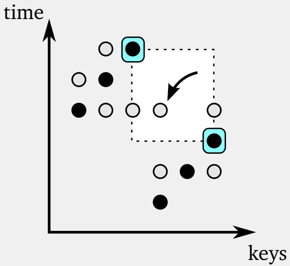


access sequence X
e.g., 4, 5, 6, 1, 2, 3

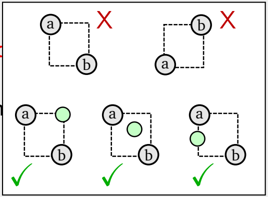
$\rightarrow$ point set X

dynamic BST serving X

$\rightarrow$ point set $Y \supseteq X$
$\uparrow$
nodes touched by pointer moves and rotations



Y is a BST execution of X $\iff$ Y is a satisfie
$\downarrow$
no $a, b \in Y$ form an en



A matrix view of BSTs

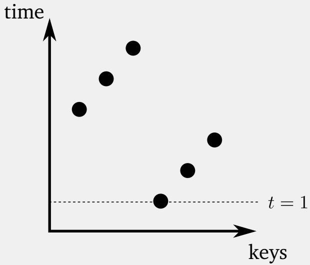
A matrix view of BSTs

Suggests a natural algorithm:

A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

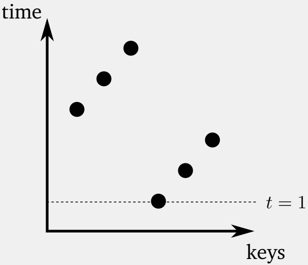


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

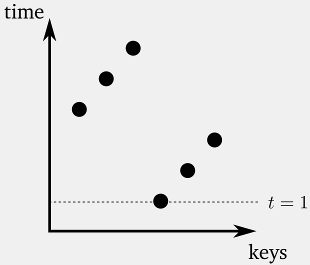


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

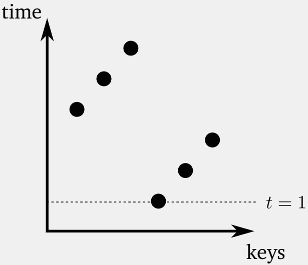


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

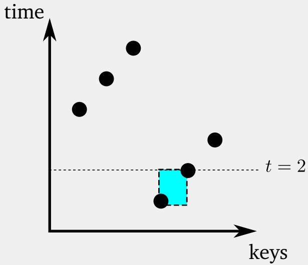


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

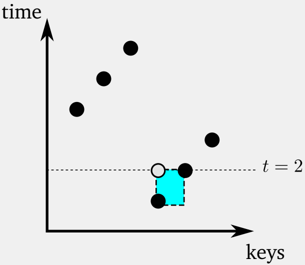


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

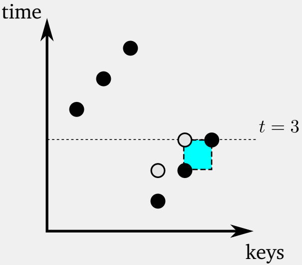


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

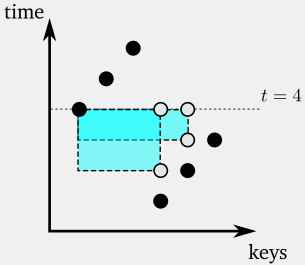
Can be implemented as a BST, similar to splay trees.



Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

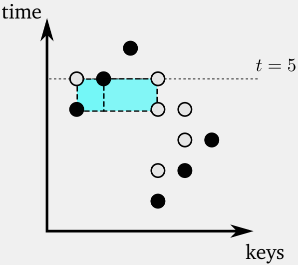


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

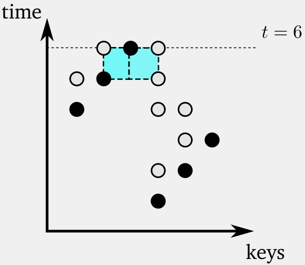


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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

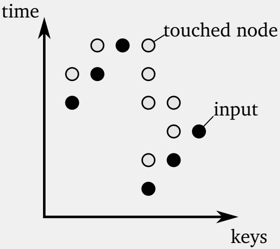


A matrix view of BSTs

Suggests a natural algorithm:

Geometric sweepline bottom-up.

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



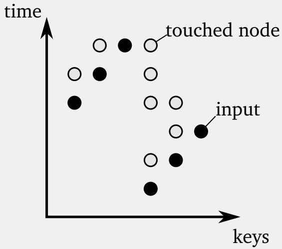
This describes an **insertion-sort** execution.

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.

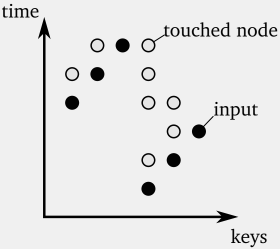
Task: Bound the cost

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

$\downarrow$

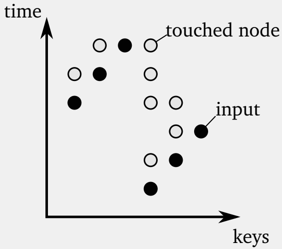
number of dots in matrix

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 π)

$\bullet$ = data structure operations

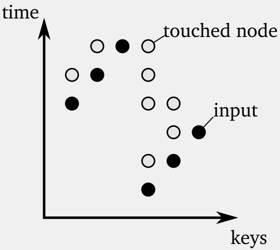
number of points ($\bullet + \bullet$) = total cost

Encode execution as a matrix

• = input sequence (avoids π)

• = data structure operations

number of points (• + •) = total cost

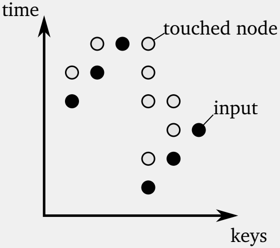


Encode execution as a matrix

• = input sequence (avoids π)

• = data structure operations

number of points (• + •) = total cost

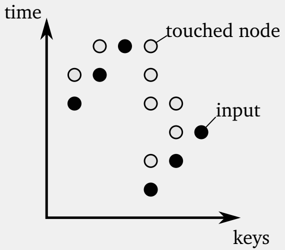


Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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

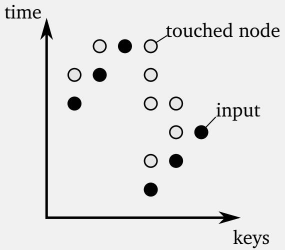


Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\color{gray}\bullet$ = data structure operations

number of points ($\bullet + \color{gray}\bullet$) = total cost

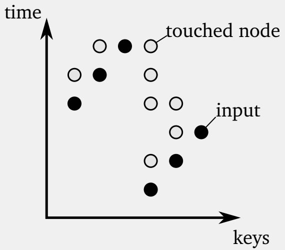


Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



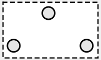
Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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

If execution contains the pattern:



Key Lemma

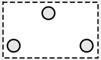
Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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

there must be an input point inside



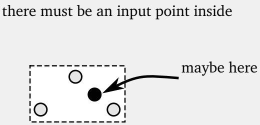
Key Lemma

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



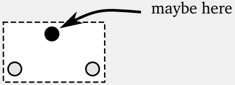
Key Lemma

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



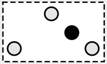
Key Lemma

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

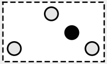
Follows from sweepline.

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

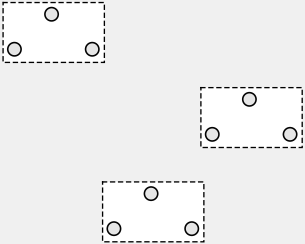
$\rightarrow$ **input-revealing gadget**

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

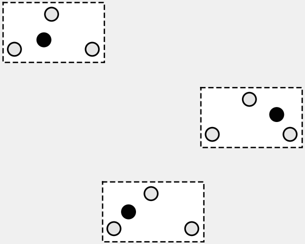
$\rightarrow$ **input-revealing gadget**

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

$\rightarrow$ **input-revealing gadget**

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

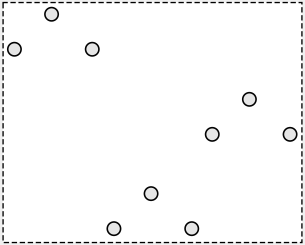
$\bullet$ = data structure operations

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

Key Lemma

Follows from sweepline.

$\rightarrow$ **input-revealing gadget**



Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

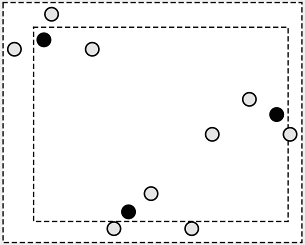
$\bullet$ = data structure operations

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

Key Lemma

Follows from sweepline.

$\rightarrow$ **input-revealing gadget**

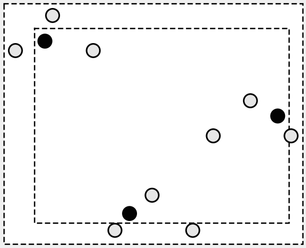


Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\color{gray}{\bullet}$ = data structure operations

number of points ($\bullet + \color{gray}{\bullet}$) = total cost



Key Lemma

Follows from sweepline.

$\rightarrow$ **input-revealing gadget**

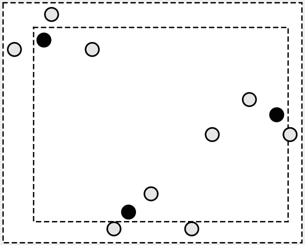
input X avoids $\begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}$

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

$\rightarrow$ **input-revealing gadget**

input X avoids $\begin{pmatrix}\bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}$

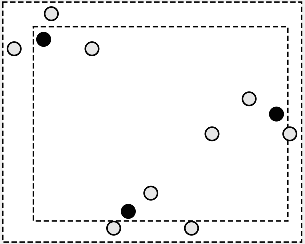
$$\implies \text{execution avoids} \begin{pmatrix} & \bullet & & & & & & & \\ \bullet & & \bullet & & & & & & \\ & & & & & & & \bullet & \\ & & & & & & \bullet & & \bullet \\ & & & & \bullet & & & & \\ & & & \bullet & & \bullet & & & \end{pmatrix}$$

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

input X avoids $\begin{pmatrix} \bullet & & \\ & & \bullet \\ & \bullet & \end{pmatrix}$

$\implies$ execution avoids $\begin{pmatrix} & \bullet & & & & & & & \\ \bullet & & \bullet & & & & & & \\ & & & & & & & \bullet & \\ & & & & & & \bullet & & \bullet \\ & & & & \bullet & & & & \\ & & & \bullet & & \bullet & & & \end{pmatrix}$

$\implies$ cost of sorting X is $\leq n \cdot 2^{\text{poly}(\alpha(n))}$ [CGKMS'15]

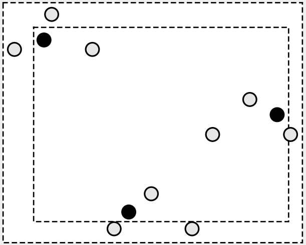
using [Klazar '00] [Keszegh '09]

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

input X avoids π

$\implies$ execution avoids $\pi \otimes ({}_{\bullet}\,{}^{\bullet}\,{}_{\bullet})$

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

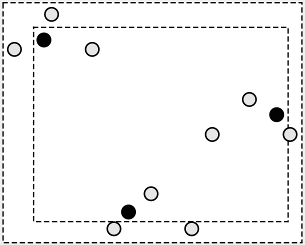
using [Klazar '00] [Keszegh '09]

Encode execution as a matrix

$\bullet$ = input sequence (avoids π)

$\bullet$ = data structure operations

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



Key Lemma

Follows from sweepline.

input X avoids π

$\implies$ execution avoids $\pi \otimes \left({}_{\bullet}{}^{\bullet}{}_{\bullet} \right)$

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

using [Klazar '00] [Keszegh '09]

$\rightarrow$ for various **special cases** $O(n)$ can be shown, e.g., for $\pi = k, \ldots, 1$.

Remarks:

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 $\mathrm{ex}\left(\pi \otimes \left({}_{\bullet}{}^{\bullet}{}_{\bullet} \right), n\right)$

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 $\operatorname{ex}\left(\pi \otimes \left(\quad\right), n\right)$

  Recent improvement: $n \cdot 2^{\alpha(n)^{O(|\pi|)}} \to n \cdot 2^{\alpha(n)+O(|\pi|^2)}$ (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 $\mathrm{ex}\left(\pi \otimes \left({}_{\bullet}{}^{\bullet}{}_{\bullet}\right), n\right)$

Recent improvement: $n \cdot 2^{\alpha(n)^{O(|\pi|)}} \to n \cdot 2^{\alpha(n)+O(|\pi|^2)}$ (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 $\mathrm{ex}\left(\pi \otimes \left(\begin{smallmatrix} & \bullet & \\ \bullet & & \bullet \end{smallmatrix}\right), n\right)$

Recent improvement: $n \cdot 2^{\alpha(n)^{O(|\pi|)}} \to n \cdot 2^{\alpha(n)+O(|\pi|^2)}$ (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 $\mathrm{ex}\left(\pi \otimes \left(\right), n\right)$

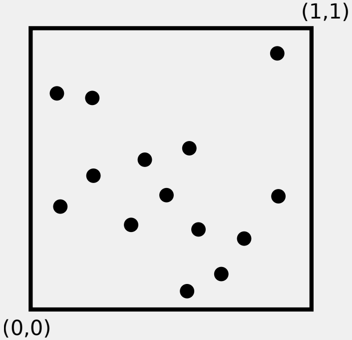
Recent improvement: $n \cdot 2^{\alpha(n)^{O(|\pi|)}} \to n \cdot 2^{\alpha(n)+O(|\pi|^2)}$ (tight)
[Chalermsook, Pettie, Ying.'23]

Can we get to $O(n)$? yes: [BKO'24] [Opler'24+]

Example: TSP

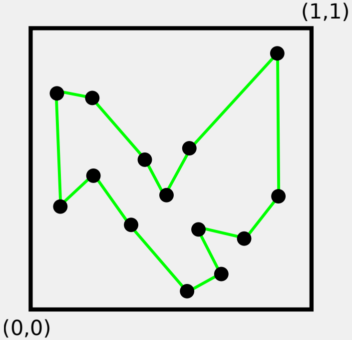
Example: TSP

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



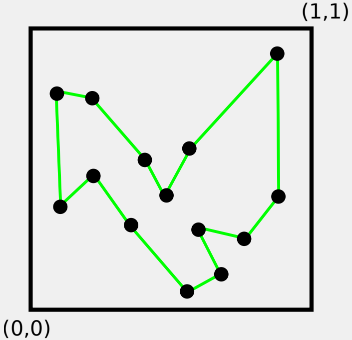
Example: TSP

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



Example: TSP

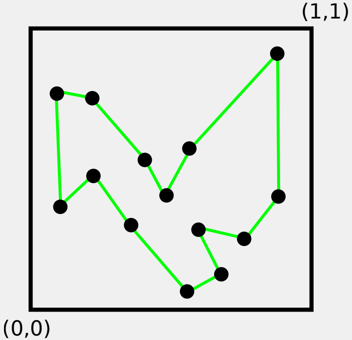
Given n points in $[0,1]^2$, find TSP-tour of min length.



Example: TSP

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

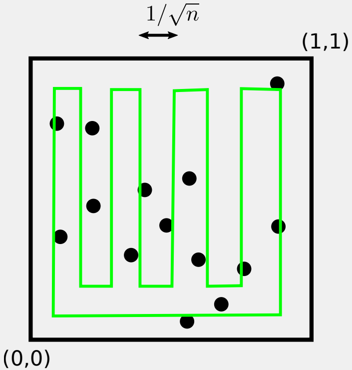
Worst-case OPT length $=$



Example: TSP

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

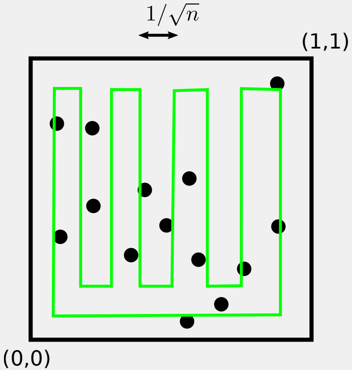
Worst-case OPT length $= O(\sqrt{n})$.



Example: TSP

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

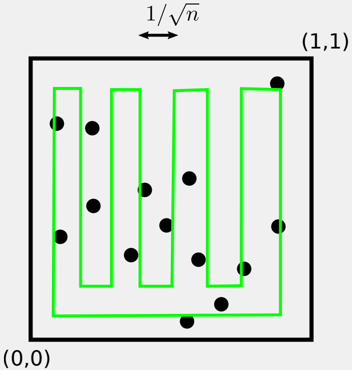
Worst-case OPT length $= O(\sqrt{n})$.



Example: TSP

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

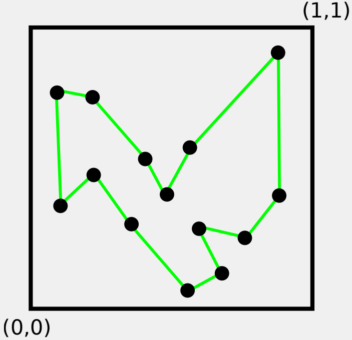
Worst-case OPT length $= O(\sqrt{n})$.



(this bound is tight)

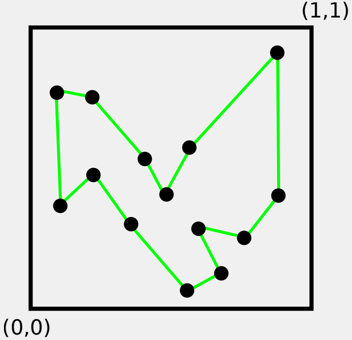
Example: TSP

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



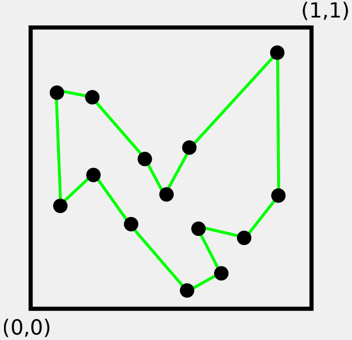
Example: TSP

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



Example: TSP

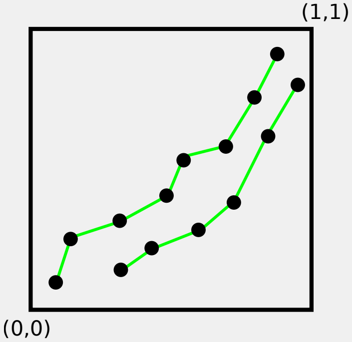
Given n points in $[0,1]^2$ avoiding π, find TSP-tour of min length.



Example: TSP

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

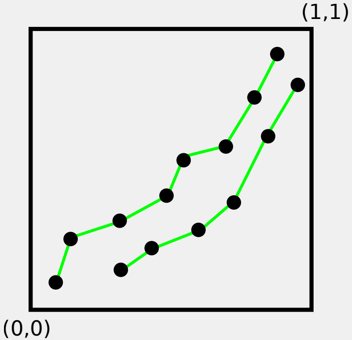
e.g. $\pi = (3,2,1)$ cost $\in O(1)$



Example: TSP

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

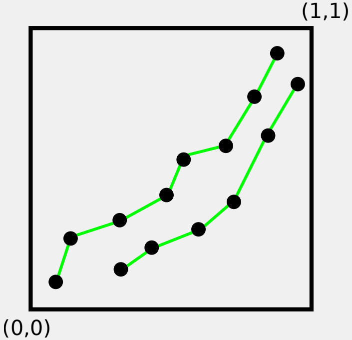
e.g. $\pi = (3,2,1)$ cost $\in O(1)$



Example: TSP

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

For arbitrary π?

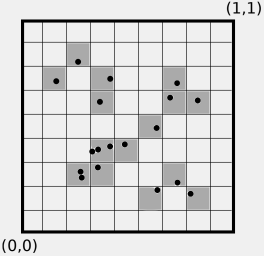


Example: TSP

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

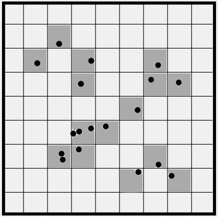
Example: TSP

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



Example: TSP

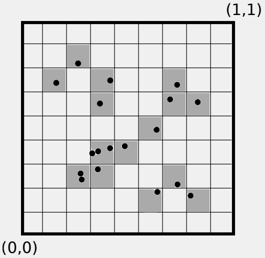
Given n points in $[0,1]^2$ avoiding π, find TSP-tour of min length.



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 π, find TSP-tour of min length.

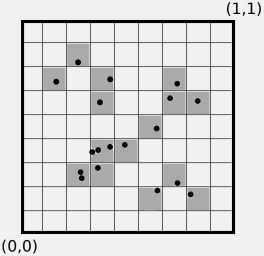


Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])

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

Example: TSP

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



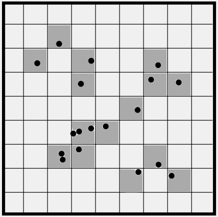
Consider $\sqrt{n} \times \sqrt{n}$ grid. Only $c_\pi \cdot \sqrt{n}$ cells touched. (by [Marcus-Tardos])

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 π, find TSP-tour of min length.



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

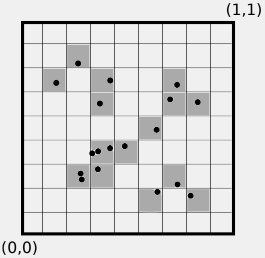
Cost $f(n) \leq$ (tour between cells) + (tours within cells).

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

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

Example: TSP

Given n points in $[0,1]^2$ avoiding π, 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).

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

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

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

X is π-avoiding

X is π-avoiding

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

X is π-avoiding

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

[Guillemot, Marx '14]

X is π-avoiding

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

[Guillemot, Marx '14]

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

X is π-avoiding

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

[Guillemot, Marx '14]

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

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

X is π-avoiding

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

[Guillemot, Marx '14]

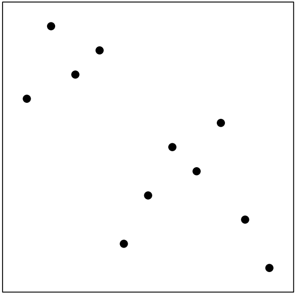
$\rightarrow$ Can use merge-sequence to construct $O_\pi(\log n)$ cost TSP tour. [BKO '24]

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

Merge sequences

Merge: Replace two points/rectangles by their bounding box

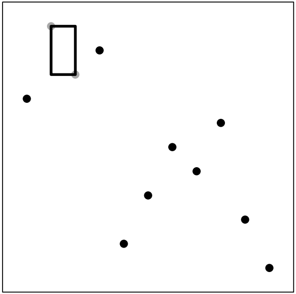
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

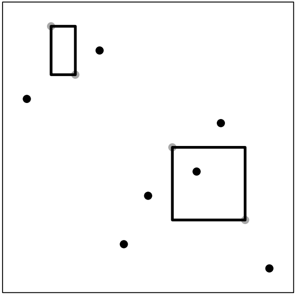
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

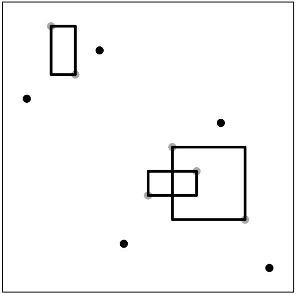
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

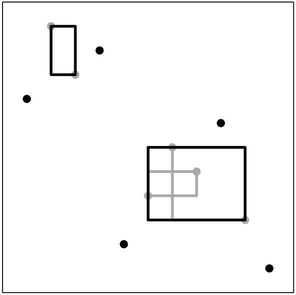
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

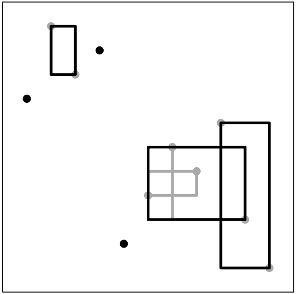
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

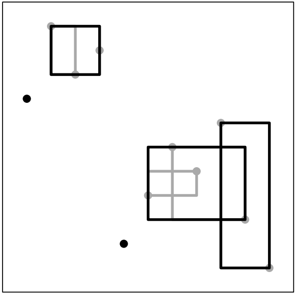
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

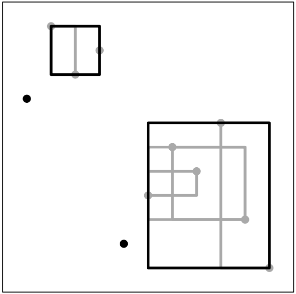
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

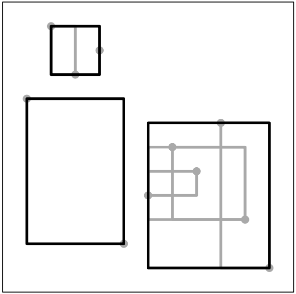
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

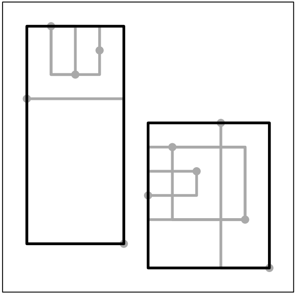
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

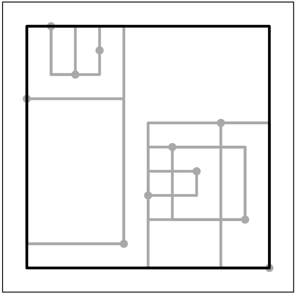
Merge sequence: Sequence of rectangle/point families obtained by successive merges



Merge sequences

Merge: Replace two points/rectangles by their bounding box

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

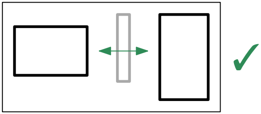


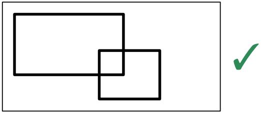
Merge sequences

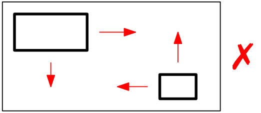
Merge: Replace two points/rectangles by their bounding box

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.







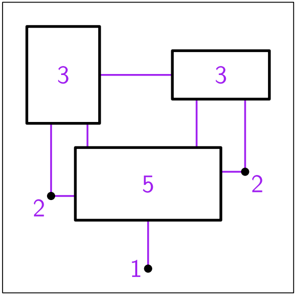
Merge sequences

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

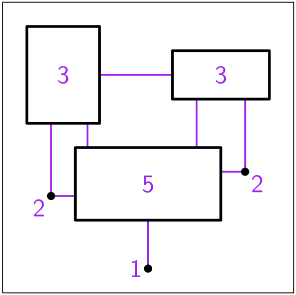
Merge: Replace two points/rectangles by their bounding box

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 π-avoiding

$\implies$ twin-width$(X) \leq c_\pi$ $\iff$ X has a c_π-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]

X is π-avoiding

$\implies$ twin-width$(X) \leq c_\pi$ $\iff$ X has a c_π-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...

X is π-avoiding

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

[Guillemot, Marx '14]

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

$\to$ 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 is π-avoiding

$\implies$ twin-width$(X) \leq c_\pi$ $\iff$ X has a c_π-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.

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

X is π-avoiding

$\implies$ twin-width$(X) \leq c_\pi$ $\iff$ X has a c_π-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.

$\implies$ 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

Conclusions

#1. Extremal combinatorics used to analyse algorithms

Conclusions

#1. Extremal combinatorics used to analyse algorithms

- Examples mostly from data structures

Conclusions

#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 [CG**K**MS'15] [**K**S'18] [B**K**O'24] [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 [CG**K**MS'15] [**K**S'18] [B**K**O'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [B**K**O'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 [CG**K**MS'15] [**K**S'18] [B**K**O'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [B**K**O'24]
- online problems: k-server [B**K**O'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 [CG**K**MS'15] [**K**S'18] [B**K**O'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [B**K**O'24]
- online problems: k-server [B**K**O'24]
- Matching lower bounds + stronger bounds for families of patterns π

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 [CG**K**MS'15] [**K**S'18] [B**K**O'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [B**K**O'24]
- online problems: k-server [B**K**O'24]

- Matching lower bounds + stronger bounds for families of patterns π
- **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

- data structures, sorting [CG**K**MS'15] [**K**S'18] [B**K**O'24] [Opler'24]
- geometric problems: TSP, MST, Steiner-tree, Manhattan-netw. [B**K**O'24]
- online problems: k-server [B**K**O'24]

- Matching lower bounds + stronger bounds for families of patterns π
- **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]

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]

. . . .

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
A. Marcus, G. Tardos. [J. Combin. Theory Ser. A 2004]

On 0-1 matrices and small excluded submatrices
G. Tardos. [J. Combin. Theory Ser. A 2005]

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]

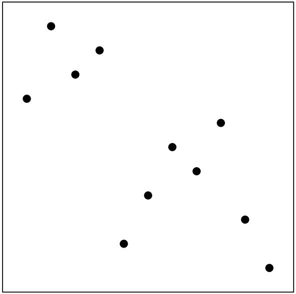
...thanks...

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)

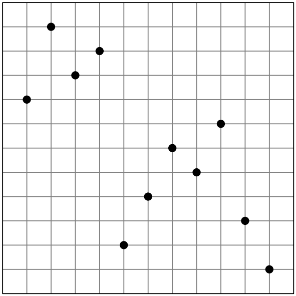


Arborially satisfied superset with merge sequences

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)

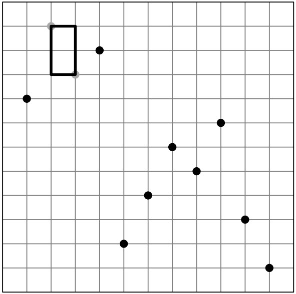


Arborially satisfied superset with merge sequences

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)

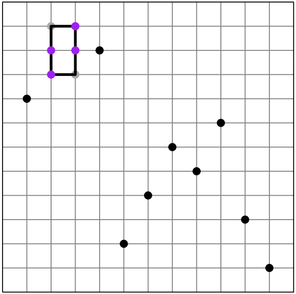


Arborially satisfied superset with merge sequences

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)

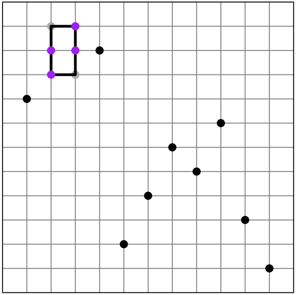


Arborially satisfied superset with merge sequences

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)

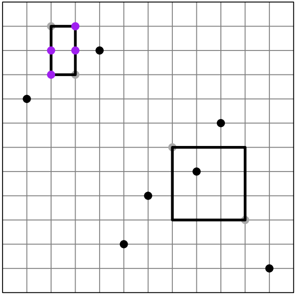


Arborially satisfied superset with merge sequences

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)

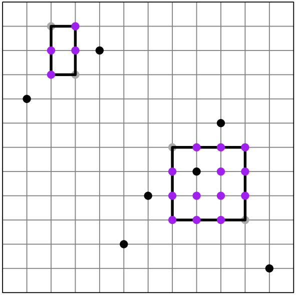


Arborially satisfied superset with merge sequences

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)

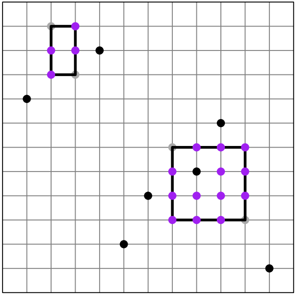


Arborially satisfied superset with merge sequences

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)

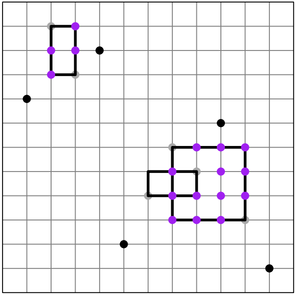


Arborially satisfied superset with merge sequences

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)

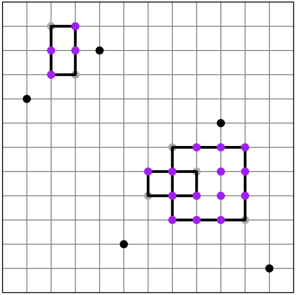


Arborially satisfied superset with merge sequences

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

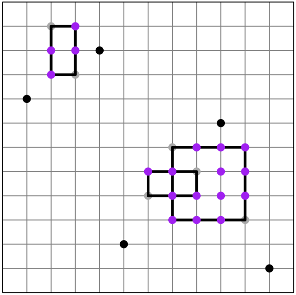
Given: d-wide merge sequence of P

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

2. Execute next merge

3. Add all grid points in new rectangle

(repeat)

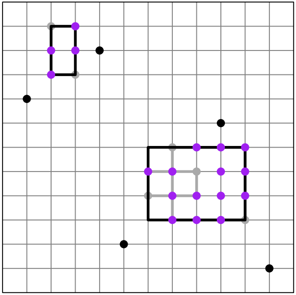


Arborially satisfied superset with merge sequences

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)

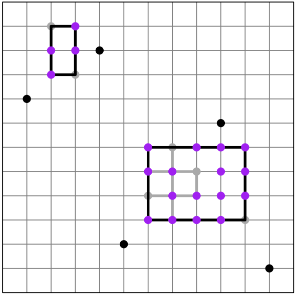


Arborially satisfied superset with merge sequences

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

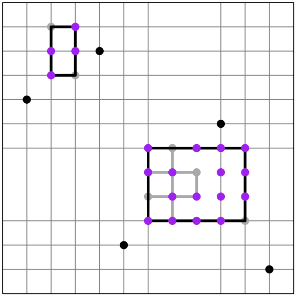
Given: d-wide merge sequence of P

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

2. Execute next merge

3. Add all grid points in new rectangle

(repeat)

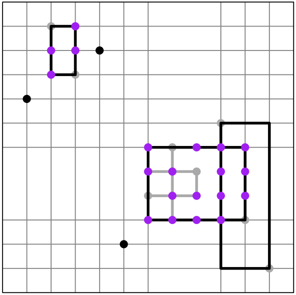


Arborially satisfied superset with merge sequences

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)

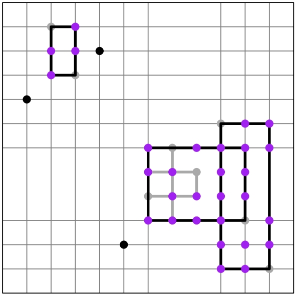


Arborially satisfied superset with merge sequences

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)

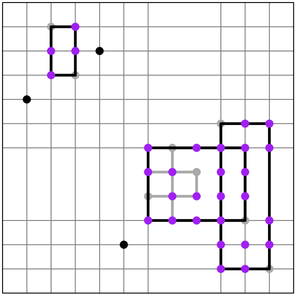


Arborially satisfied superset with merge sequences

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)

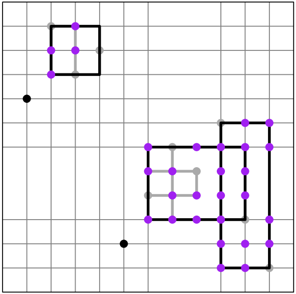


Arborially satisfied superset with merge sequences

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)

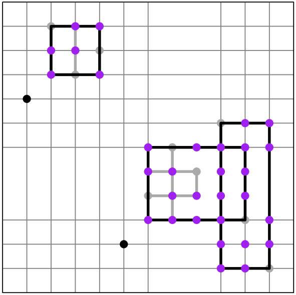


Arborially satisfied superset with merge sequences

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)

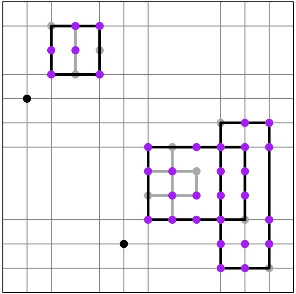


Arborially satisfied superset with merge sequences

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)

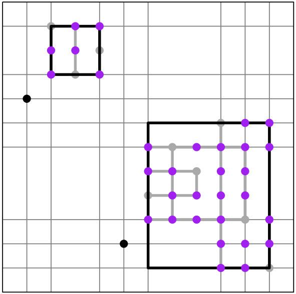


Arborially satisfied superset with merge sequences

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)

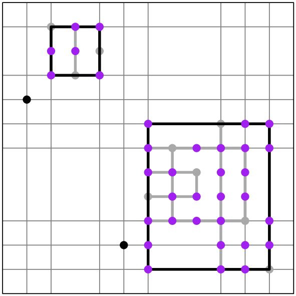


Arborially satisfied superset with merge sequences

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)

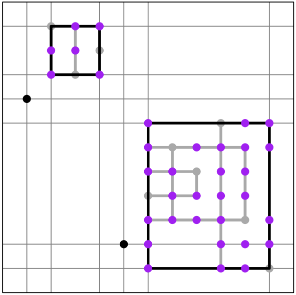


Arborially satisfied superset with merge sequences

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)

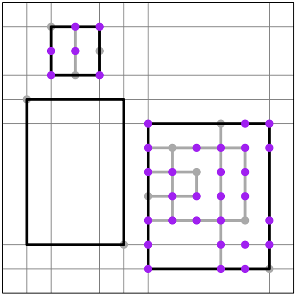


Arborially satisfied superset with merge sequences

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)

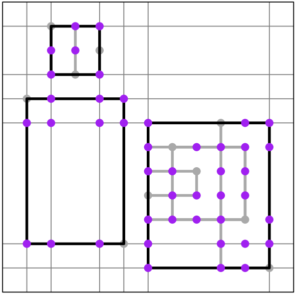


Arborially satisfied superset with merge sequences

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)

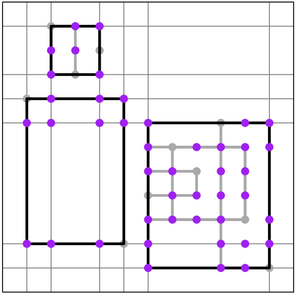


Arborially satisfied superset with merge sequences

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)

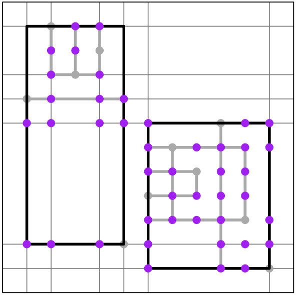


Arborially satisfied superset with merge sequences

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)

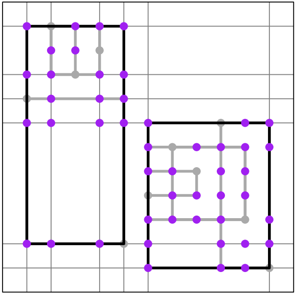


Arborially satisfied superset with merge sequences

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)

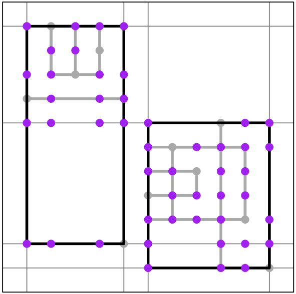


Arborially satisfied superset with merge sequences

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)

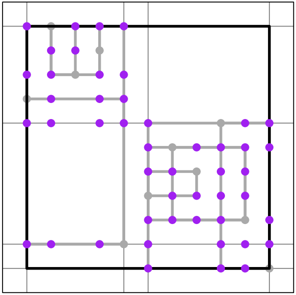


Arborially satisfied superset with merge sequences

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)

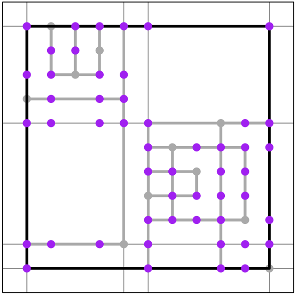


Arborially satisfied superset with merge sequences

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)

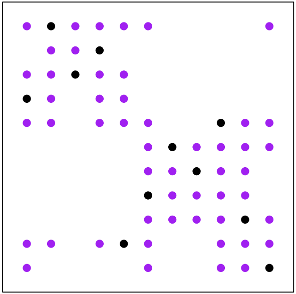


Arborially satisfied superset with merge sequences

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)

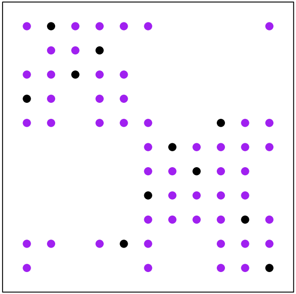


Arborially satisfied superset with merge sequences

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)



Claim:
(a) Result is arborally satisfied
(b) $\#$ of added points is $\mathcal{O}(d^2 \cdot n)$ **(proof now)**

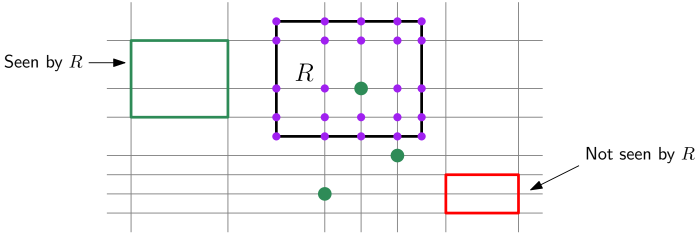
of added points

Claim: In every step, we add $\mathcal{O}(d^2)$ points.

$\#$ of added points

Claim: In every step, we add $\mathcal{O}(d^2)$ points.

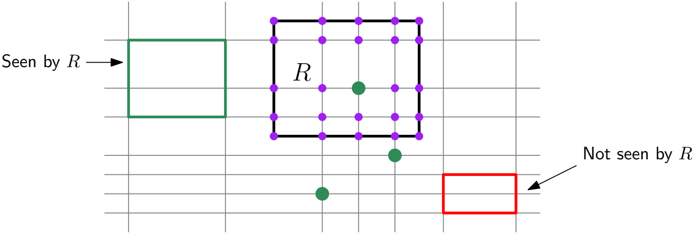
Proof idea: Rectangle sees $\leq d$ other rectangles/points
$\implies \leq 4d$ grid lines



$\#$ of added points

Claim: In every step, we add $\mathcal{O}(d^2)$ 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 $\mathcal{O}(d^2 \cdot n)$.

$\implies$ The optimum for every π-avoiding input sequence is $\mathcal{O}(c_\pi^2 \cdot n)$.

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 π-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 $\mathcal{O}(1/(n-i))$. **[NEW]**

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

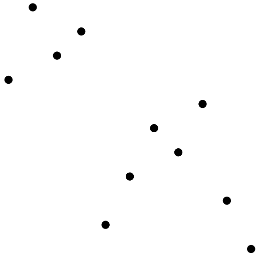
Spanning tree construction: Whenever we merge two rectangles, connect arbitrary points within them.

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 π-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.

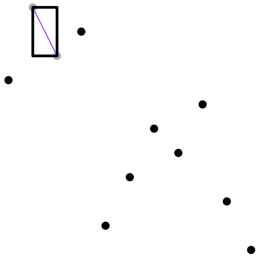


Distance-balanced merge sequences for MST

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 π-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.

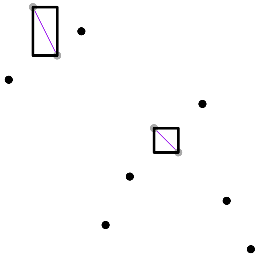


Distance-balanced merge sequences for MST

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 π-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.

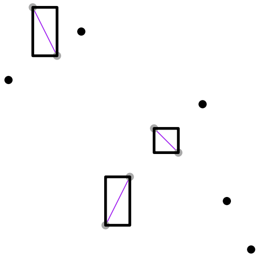


Distance-balanced merge sequences for MST

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 π-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.

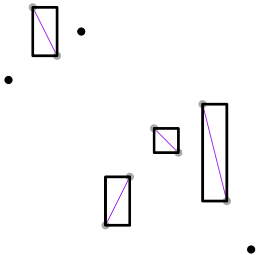


Distance-balanced merge sequences for MST

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 π-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.

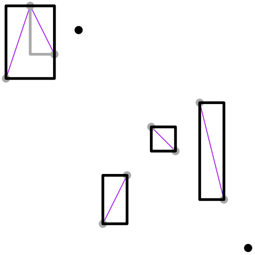


Distance-balanced merge sequences for MST

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 π-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.

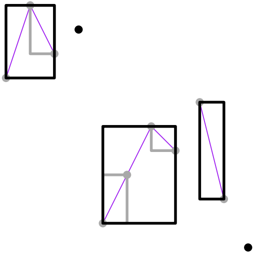


Distance-balanced merge sequences for MST

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 π-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.

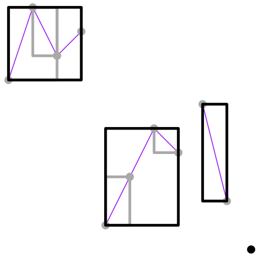


Distance-balanced merge sequences for MST

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 π-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.

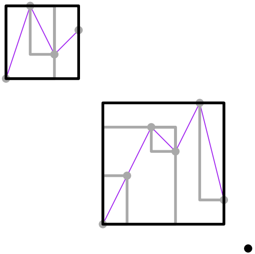


Distance-balanced merge sequences for MST

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 π-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.

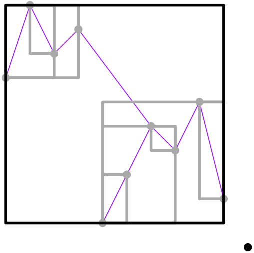


Distance-balanced merge sequences for MST

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 π-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.

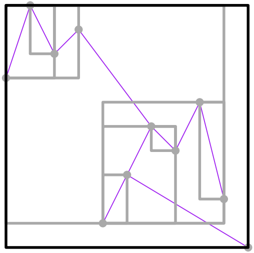


Distance-balanced merge sequences for MST

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 π-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.

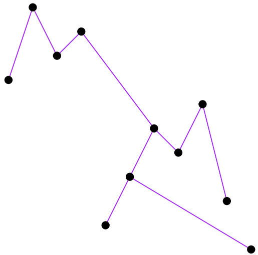


Distance-balanced merge sequences for MST

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 π-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.

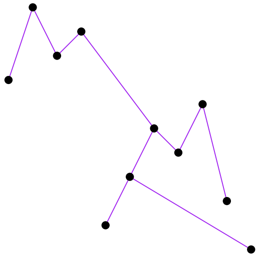


Distance-balanced merge sequences for MST

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 π-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.



Length: $$\sum_{i=1}^{n-1} \frac{1}{n-i} \approx \log n$$