Finding monotone patterns

Shoham Letzter

ETH–ITS

joint with Omri Ben-Eliezer, Clément Canonne, Erik Waingarten

Mittagsseminar

December 2019

Property testing

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

E.g. determine, with high probability, if a graph G is bipartite, or cannot be made bipartite by removing at most $\varepsilon|G|^2$ edges.

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

E.g. determine, with high probability, if a graph G is bipartite, or cannot be made bipartite by removing at most $\varepsilon|G|^2$ edges.

(This can be done in time $O(1)$.)

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

E.g. determine, with high probability, if a graph G is bipartite, or cannot be made bipartite by removing at most $\varepsilon|G|^2$ edges.

(This can be done in time $O(1)$.)

We consider testing with **one-sided error**:

Property testing

Aim: design fast (randomised) algorithms that determine, with probability at least 0.99, if a given (large) object

- has property $\mathcal{P}$,
- or is far from having property $\mathcal{P}$.

E.g. determine, with high probability, if a graph G is bipartite, or cannot be made bipartite by removing at most $\varepsilon|G|^2$ edges.

(This can be done in time $O(1)$.)

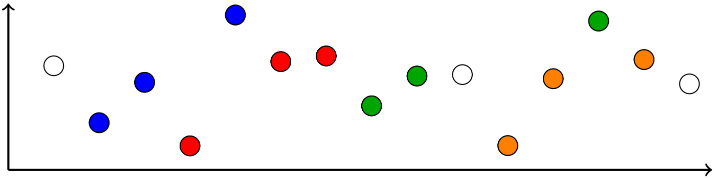
We consider testing with **one-sided error**: if an object is far from having $\mathcal{P}$, provide evidence.

Testing for $(1\ldots k)$-freeness

Testing for (1...k)-freeness

Fix $k \geq 2$.

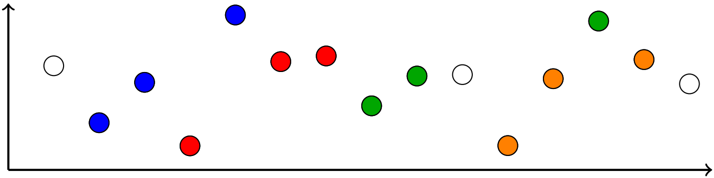
- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.



Testing for (1...k)-freeness

Fix $k \geq 2$.

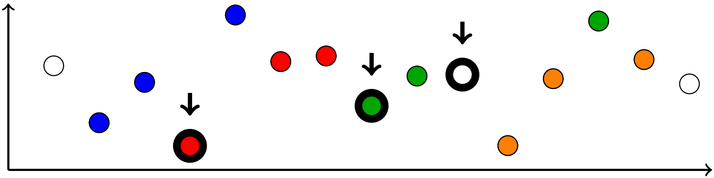
- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.
- **Aim.** Find, with high probability, an increasing k-tuple.



Testing for (1...k)-freeness

Fix $k \geq 2$.

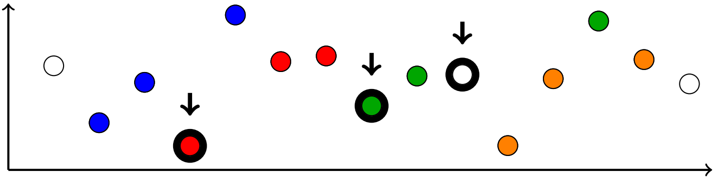
- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.
- **Aim.** Find, with high probability, an increasing k-tuple.



Testing for (1...k)-freeness

Fix $k \geq 2$.

- **Input.** $f : [n] \to \mathbb{R}$ with $\Omega(n)$ disjoint increasing k-tuples.
- **Aim.** Find, with high probability, an increasing k-tuple.



We sometimes refer to an increasing k-tuple as a (**1**...**k**)-**copy**.

History

History

$\boldsymbol{k = 2}$**:** monotonicity testing (with one-sided error).

History

$k = 2$: monotonicity testing (with one-sided error).

Ergün, Kannan, Kumar, Rubinfeld, Viswanathan '98. Optimal non-adaptive monotonicity testers make $\Theta(\log n)$ queries.

(non-adaptivity: queries do not depend on previous outcomes.)

History

$k = 2$: monotonicity testing (with one-sided error).

Ergün, Kannan, Kumar, Rubinfeld, Viswanathan '98. Optimal non-adaptive monotonicity testers make $\Theta(\log n)$ queries.

(non-adaptivity: queries do not depend on previous outcomes.)

Fischer '09. Adaptivity does not help monotonicity testing!

History

$k = 2$: monotonicity testing (with one-sided error).

Ergün, Kannan, Kumar, Rubinfeld, Viswanathan '98. Optimal non-adaptive monotonicity testers make $\Theta(\log n)$ queries.

(non-adaptivity: queries do not depend on previous outcomes.)

Fischer '09. Adaptivity does not help monotonicity testing!

Newman, Rabinovich, Rajendraprasad, Sohler '17. For $k \geq 2$, there is a (non-adaptive) tester which makes $(\log n)^{O(k^2)}$ queries.

Our results

Our results

Theorem (Ben-Eliezer, Canonne, L., Waingarten)

An optimal non-adaptive algorithm for testing $(1\ldots k)$-freeness makes $\Theta_k\left((\log n)^{\lfloor \log_2 k \rfloor}\right)$ queries.

Our results

Theorem (Ben-Eliezer, Canonne, L., Waingarten)

An optimal non-adaptive algorithm for testing $(1\ldots k)$-freeness makes $\Theta_k\left((\log n)^{\lfloor \log_2 k \rfloor}\right)$ queries.

Theorem (Ben-Eliezer, L., Waingarten)

An optimal adaptive algorithm for testing $(1\ldots k)$-freeness makes $\Theta_k(\log n)$ queries.

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x, y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x, y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$,

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x, y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$, then $b(x, y) = 3$.

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1\textcolor{red}{0}11_{\text{bin}} \\ y = 13 = 1\textcolor{red}{1}01_{\text{bin}} \end{array}$, then $b(x, y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x, y) : x, y \in A\}$.

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max\left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$, then $b(x,y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x,y) : x, y \in A\}$.

Claim

$|b(A)| \leq |A| - 1$ *for every finite* $A \subseteq \mathbb{N}$.

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$, then $b(x,y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x,y) : x, y \in A\}$.

Claim

$|b(A)| \le |A| - 1$ *for every finite* $A \subseteq \mathbb{N}$.

Proof.

Let $i = \max b(A)$.

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$, then $b(x,y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x,y) : x, y \in A\}$.

Claim

$|b(A)| \le |A| - 1$ *for every finite* $A \subseteq \mathbb{N}$.

Proof.

Let $i = \max b(A)$. For $\sigma \in \{0,1\}$, define

$$A_\sigma = \{x \in A : \text{the } i^{\text{th}} \text{ digit of } x \text{ in binary is } \sigma\}.$$

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1{\color{red}0}11_{\text{ bin}} \\ y = 13 = 1{\color{red}1}01_{\text{ bin}} \end{array}$, then $b(x,y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x,y) : x, y \in A\}$.

Claim

$|b(A)| \le |A| - 1$ *for every finite* $A \subseteq \mathbb{N}$.

Proof.

Let $i = \max b(A)$. For $\sigma \in \{0,1\}$, define

$$A_\sigma = \{x \in A : \text{the } i^{\text{th}} \text{ digit of } x \text{ in binary is } \sigma\}.$$

$|b(A)| \le |b(A_0)| + |b(A_1)| + 1$

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$, then $b(x,y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x,y) : x, y \in A\}$.

Claim

$|b(A)| \leq |A| - 1$ *for every finite* $A \subseteq \mathbb{N}$.

Proof.

Let $i = \max b(A)$. For $\sigma \in \{0,1\}$, define

$$A_\sigma = \{x \in A : \text{the } i^{\text{th}} \text{ digit of } x \text{ in binary is } \sigma\}.$$

$|b(A)| \leq |b(A_0)| + |b(A_1)| + 1 \leq |A_0| - 1 + |A_1| - 1 + 1$

Towards a lower bound for $k = 2$: binary profiles

For $x, y \in \mathbb{N}$: $\boldsymbol{b(x,y)} = \max \left\{ i : \begin{array}{l} \text{the binary representations} \\ \text{of } x \text{ and } y \text{ differ in } i^{\text{th}} \text{ digit} \end{array} \right\}$.

E.g. If $\begin{array}{l} x = 11 = 1011_{\text{bin}} \\ y = 13 = 1101_{\text{bin}} \end{array}$, then $b(x,y) = 3$.

For $A \subseteq \mathbb{N}$: $b(A) := \{b(x,y) : x, y \in A\}$.

Claim

$|b(A)| \leq |A| - 1$ *for every finite* $A \subseteq \mathbb{N}$.

Proof.

Let $i = \max b(A)$. For $\sigma \in \{0,1\}$, define

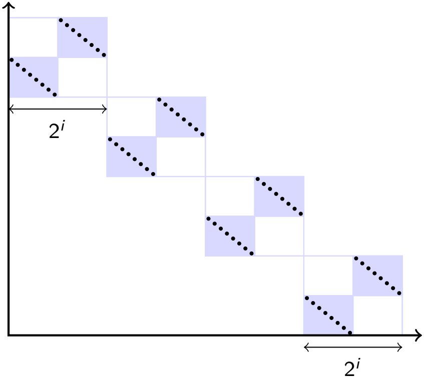
$$A_\sigma = \{x \in A : \text{the } i^{\text{th}} \text{ digit of } x \text{ in binary is } \sigma\}.$$

$|b(A)| \leq |b(A_0)| + |b(A_1)| + 1 \leq |A_0| - 1 + |A_1| - 1 + 1 = |A| - 1.$ □

Lower bound for $k = 2$

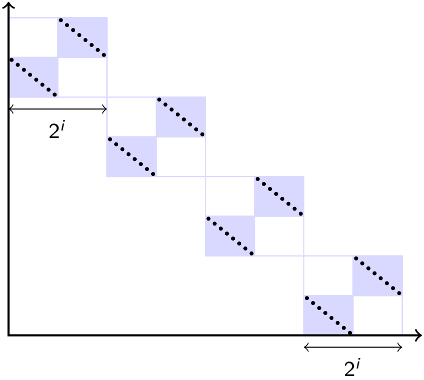
Lower bound for $k = 2$

For $i \in [\log n]$, define $f_i : [0, n-1] \to [0, n-1]$ by



Lower bound for $k = 2$

For $i \in [\log n]$, define $f_i : [0, n-1] \to [0, n-1]$ by



Observation

If (x, y) is an increasing pair in f_i, then $b(x, y) = i$.

Lower bound for $k = 2$

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

$\mathbb{P}(\mathbf{A}$ has an increasing pair w.r.t. $\mathbf{f}) =$

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

$\mathbb{P}(\mathbf{A}$ has an increasing pair w.r.t. $\mathbf{f}) =$

$$\sum_{A:\, |A| \le \alpha \log n} \mathbb{P}(\mathbf{A} = A) \cdot \frac{1}{\log n} \cdot \sum_i \mathbb{1}\left\{ \begin{array}{l} A \text{ has an increasing} \\ \text{pair w.r.t. } f_i \end{array} \right\} \le$$

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

$\mathbb{P}(\mathbf{A}$ has an increasing pair w.r.t. $\mathbf{f}) =$

$$\sum_{A\colon |A| \le \alpha \log n} \mathbb{P}(\mathbf{A} = A) \cdot \frac{1}{\log n} \cdot \sum_i \mathbb{1}\left\{ \begin{array}{l} A \text{ has an increasing} \\ \text{pair w.r.t. } f_i \end{array} \right\} \le$$

$$\sum_{A\colon |A| \le \alpha \log n} \mathbb{P}[\mathbf{A} = A] \cdot \frac{1}{\log n} \cdot \underbrace{\sum_i \mathbb{1}[i \in b(A)]}$$

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

$\mathbb{P}(\mathbf{A}$ has an increasing pair w.r.t. $\mathbf{f}) =$

$$\sum_{A\colon |A| \le \alpha \log n} \mathbb{P}(\mathbf{A} = A) \cdot \frac{1}{\log n} \cdot \sum_i \mathbb{1}\left\{ \begin{array}{l} A \text{ has an increasing} \\ \text{pair w.r.t. } f_i \end{array} \right\} \le$$

$$\sum_{A\colon |A| \le \alpha \log n} \mathbb{P}[\mathbf{A} = A] \cdot \frac{1}{\log n} \cdot \underbrace{\sum_i \mathbb{1}[i \in b(A)]}_{= |b(A)| \le |A| \le \alpha \log n}$$

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

$\mathbb{P}(\mathbf{A}$ has an increasing pair w.r.t. $\mathbf{f}) =$

$$\sum_{A \colon |A| \le \alpha \log n} \mathbb{P}(\mathbf{A} = A) \cdot \frac{1}{\log n} \cdot \sum_i \mathbb{1}\left\{ \begin{array}{l} A \text{ has an increasing} \\ \text{pair w.r.t. } f_i \end{array} \right\} \le$$

$$\sum_{A \colon |A| \le \alpha \log n} \mathbb{P}[\mathbf{A} = A] \cdot \frac{1}{\log n} \cdot \underbrace{\sum_i \mathbb{1}[i \in b(A)]}_{= |b(A)| \le |A| \le \alpha \log n} \le \alpha.$$

Lower bound for $k = 2$

Choose $\mathbf{f}$ uniformly at random from $\{f_1, \ldots, f_{\log n}\}$.

Let $\mathbf{A}$ be a random subset of $[0, n-1]$ of size at most $\alpha \log n$.

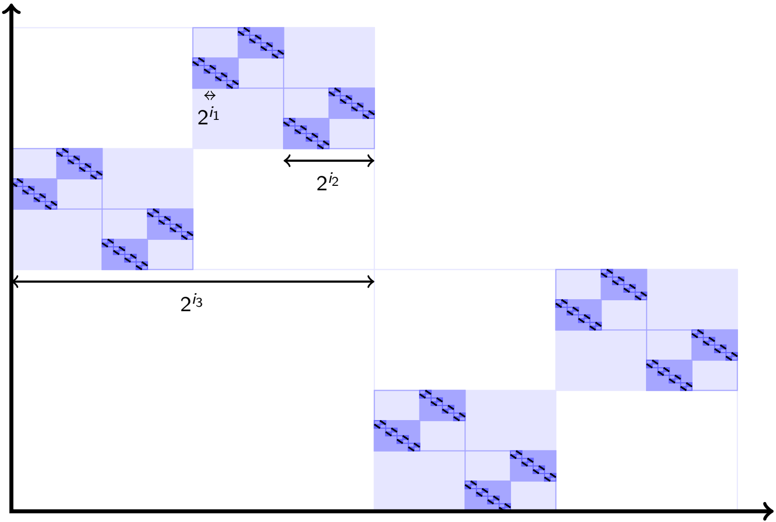
$\mathbb{P}(\mathbf{A}$ has an increasing pair w.r.t. $\mathbf{f}) =$

$$\sum_{A\colon |A| \le \alpha \log n} \mathbb{P}(\mathbf{A} = A) \cdot \frac{1}{\log n} \cdot \sum_i \mathbb{1}\left\{ \begin{array}{l} A \text{ has an increasing} \\ \text{pair w.r.t. } f_i \end{array} \right\} \le$$

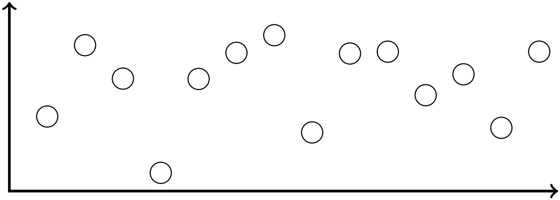
$$\sum_{A\colon |A| \le \alpha \log n} \mathbb{P}[\mathbf{A} = A] \cdot \frac{1}{\log n} \cdot \underbrace{\sum_i \mathbb{1}[i \in b(A)]}_{= |b(A)| \le |A| \le \alpha \log n} \le \alpha.$$

Thus, to find an increasing pair with probability at least 0.99 need at least $0.99 \log n$ queries.

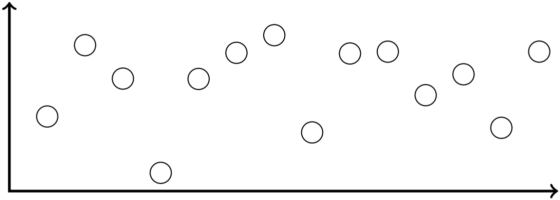
Lower bound for $k = 2^\kappa$: iterated construction



Choosing copies greedily

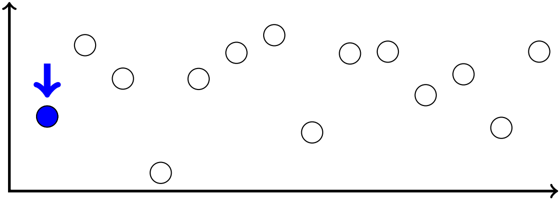


Choosing copies greedily



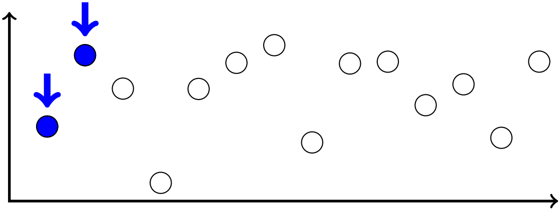
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



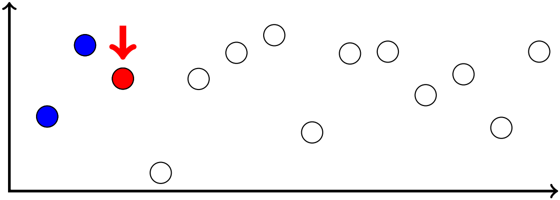
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



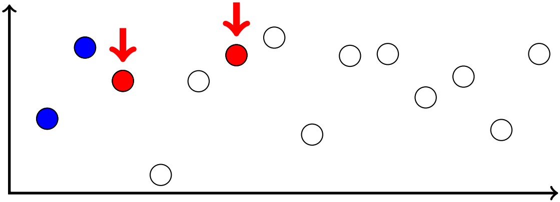
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



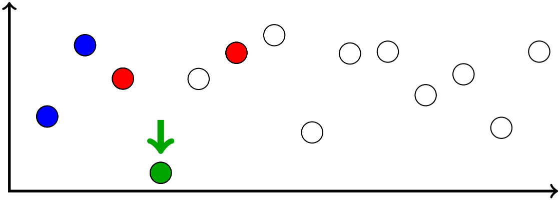
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



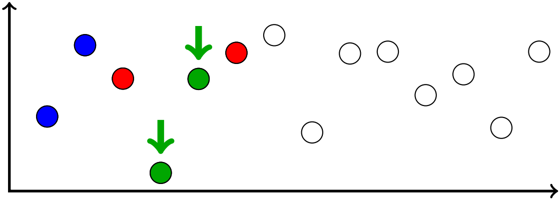
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



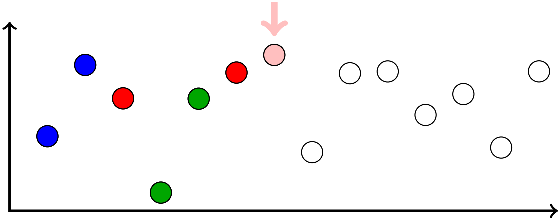
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



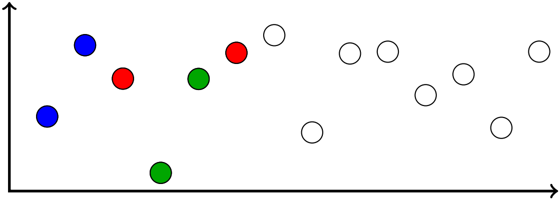
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



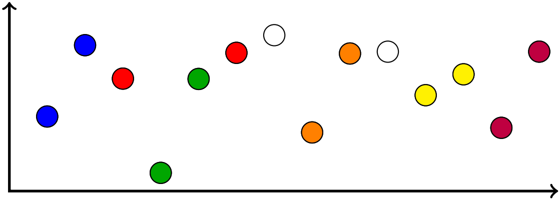
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



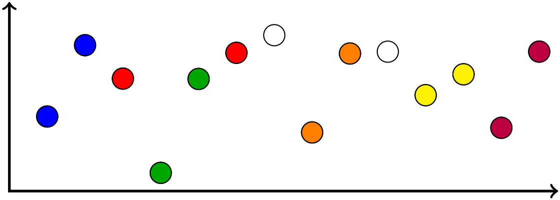
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily



- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Choosing copies greedily

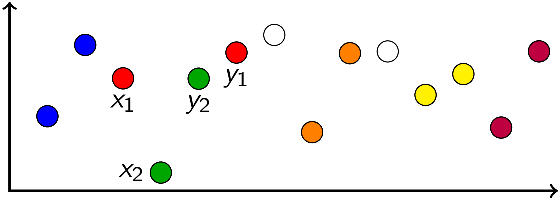


- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Observation

If $(x_1, y_1), (x_2, y_2) \in \mathcal{F}$, and $x_1 < x_2$, $y_1 > y_2$, then $f(y_1) > f(y_2)$.

Choosing copies greedily

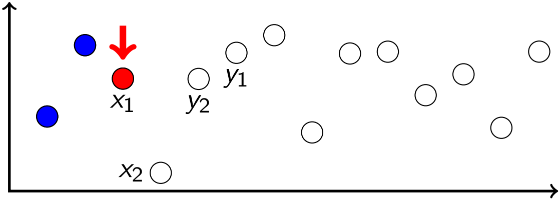


- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Observation

If $(x_1, y_1), (x_2, y_2) \in \mathcal{F}$, and $x_1 < x_2$, $y_1 > y_2$, then $f(y_1) > f(y_2)$.

Choosing copies greedily



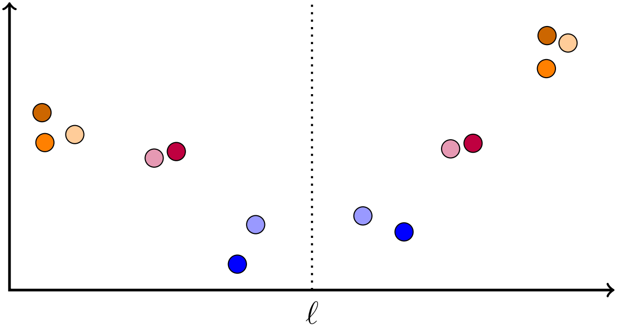
- Start with $\mathcal{F} = \emptyset$.
- For every x, from left to right, if x still unused:
  Let $y > x$ be leftmost with $f(y) > f(x)$; add (x, y) to $\mathcal{F}$.

Observation

If $(x_1, y_1), (x_2, y_2) \in \mathcal{F}$, and $x_1 < x_2$, $y_1 > y_2$, then $f(y_1) > f(y_2)$.

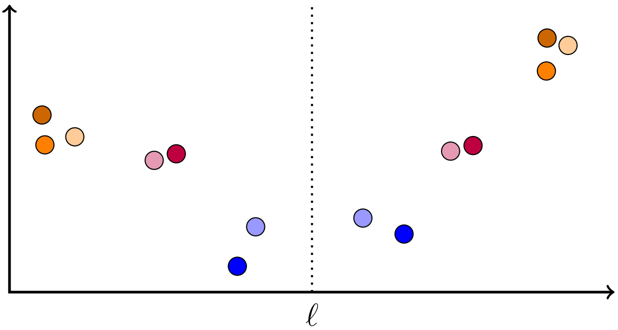
Using the greedy pairing

Using the greedy pairing



Suppose ℓ lies 'roughly in middle' of many pairs of different widths.

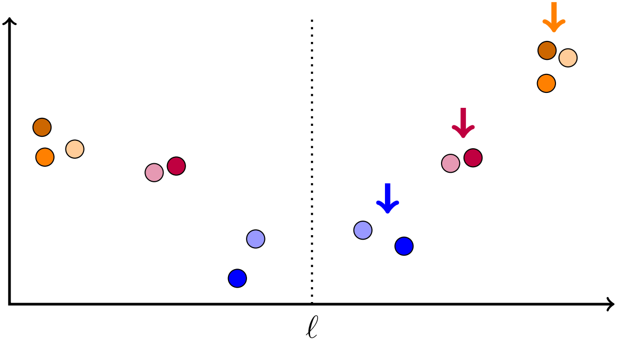
Using the greedy pairing



Suppose ℓ lies 'roughly in middle' of many pairs of different widths.

- If hit entries of k different-widths copies to the right of ℓ, find $(1 \ldots k)$.

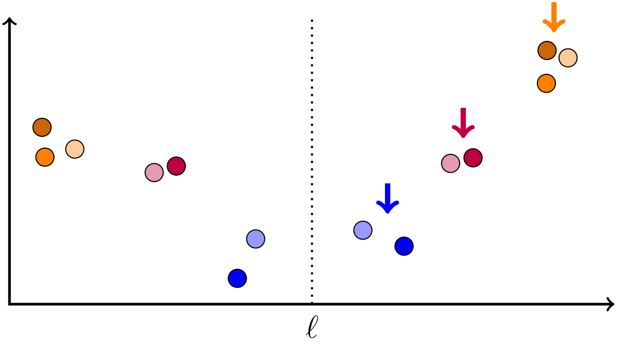
Using the greedy pairing



Suppose ℓ lies 'roughly in middle' of many pairs of different widths.

- If hit entries of k different-widths copies to the right of ℓ, find $(1 \dots k)$.

Using the greedy pairing



Suppose ℓ lies 'roughly in middle' of many pairs of different widths.

- If hit entries of k different-widths copies to the right of ℓ, find $(1 \dots k)$.
- Can be done with $O(\log n)$ queries: sample $\Theta(1)$ elements from $[\ell, \ell + 2^i]$ for every $i \in [\log n]$.

Structure thereom

Structure thereom

- $\Omega(n)$ many ℓ's are 'roughly in middle' of many copies.

Structure thereom

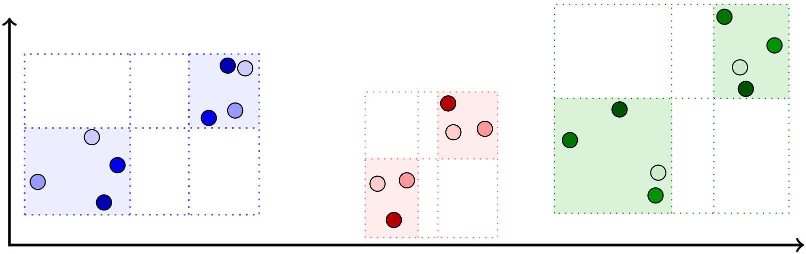
- $\Omega(n)$ many ℓ's are 'roughly in middle' of many copies.
- If many of them are in middle of many copies of different widths, can find $(1...k)$ with $O(\log n)$ queries.

Structure thereom

- $\Omega(n)$ many ℓ's are 'roughly in middle' of many copies.
- If many of them are in middle of many copies of different widths, can find $(1 \ldots k)$ with $O(\log n)$ queries.
- Otherwise, can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.

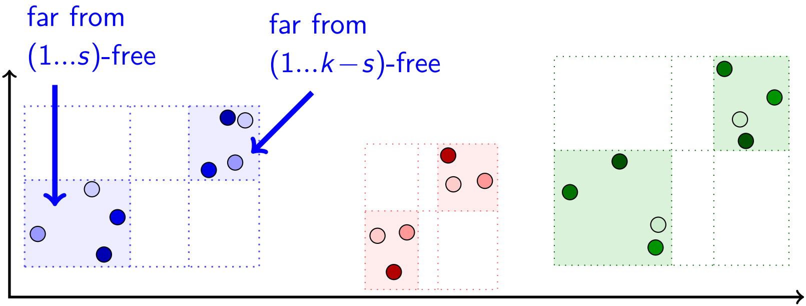
Structure thereom

- $\Omega(n)$ many ℓ's are 'roughly in middle' of many copies.
- If many of them are in middle of many copies of different widths, can find $(1 \dots k)$ with $O(\log n)$ queries.
- Otherwise, can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.



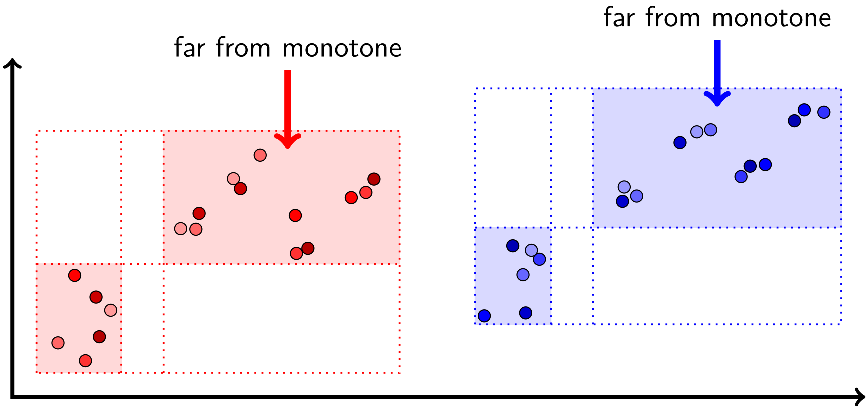
Structure thereom

- $\Omega(n)$ many ℓ's are 'roughly in middle' of many copies.
- If many of them are in middle of many copies of different widths, can find $(1...k)$ with $O(\log n)$ queries.
- Otherwise, can cover $\Omega(n)$ entries with disjoint 'splittable intervals'.

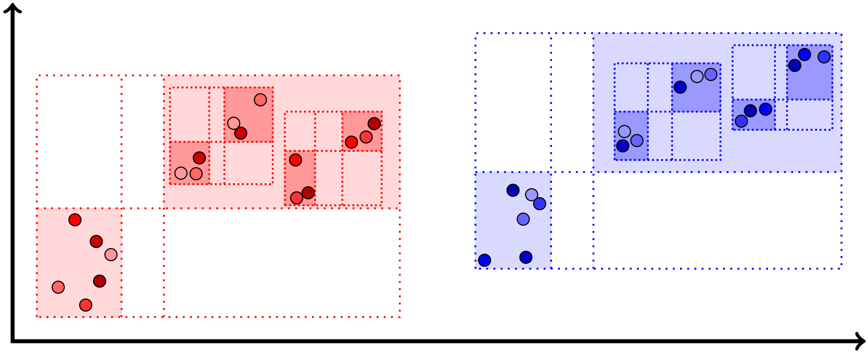


Finding (123) with $O(\log n)$ queries

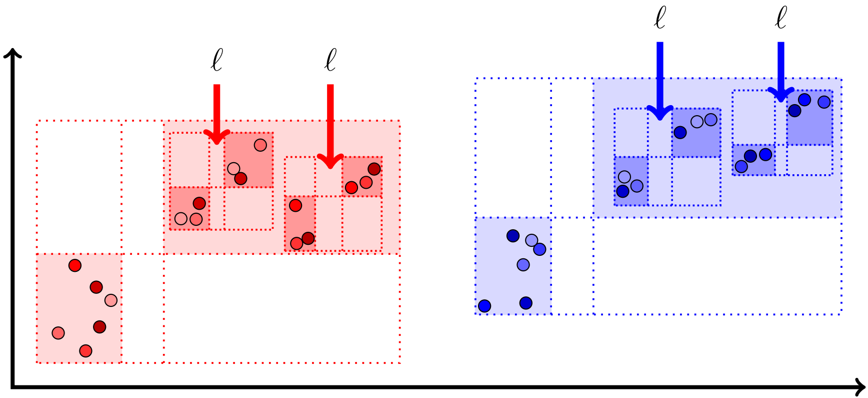
Finding (123) with $O(\log n)$ queries



Finding (123) with $O(\log n)$ queries

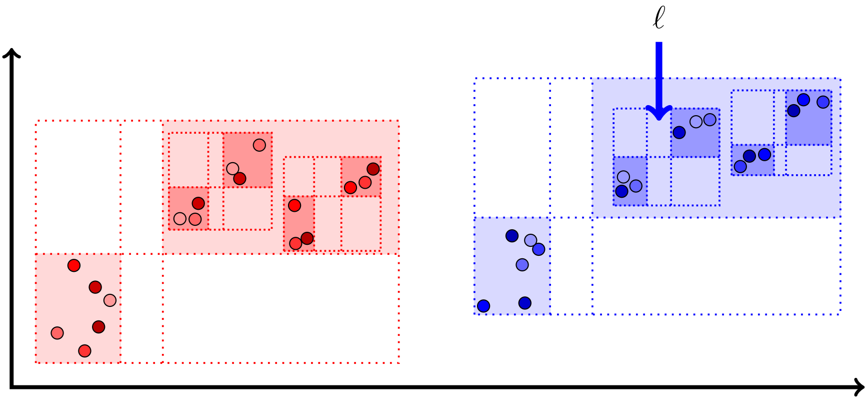


Finding (123) with $O(\log n)$ queries



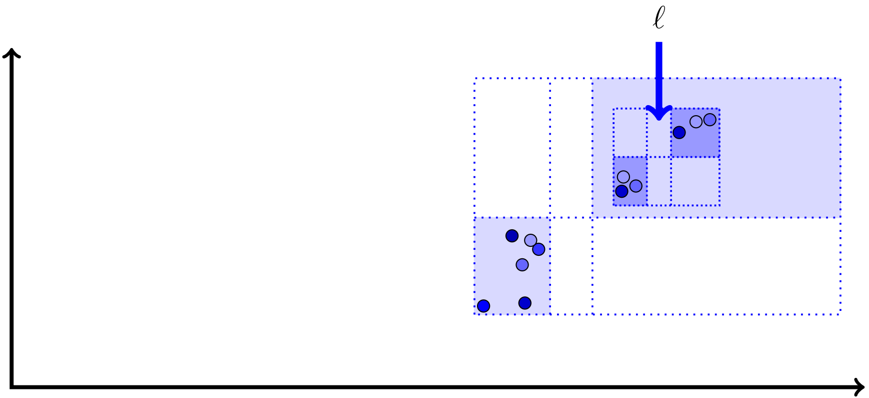
- $\Theta(1)$ queries to find ℓ as in figure.

Finding (123) with $O(\log n)$ queries



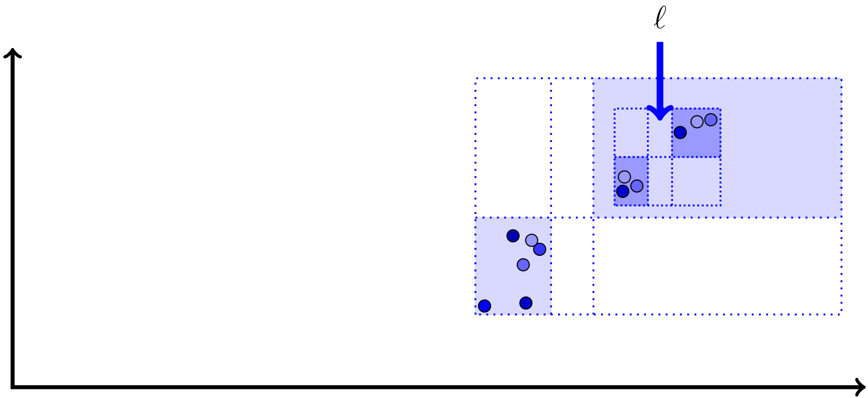
- $\Theta(1)$ queries to find ℓ as in figure.

Finding (123) with $O(\log n)$ queries



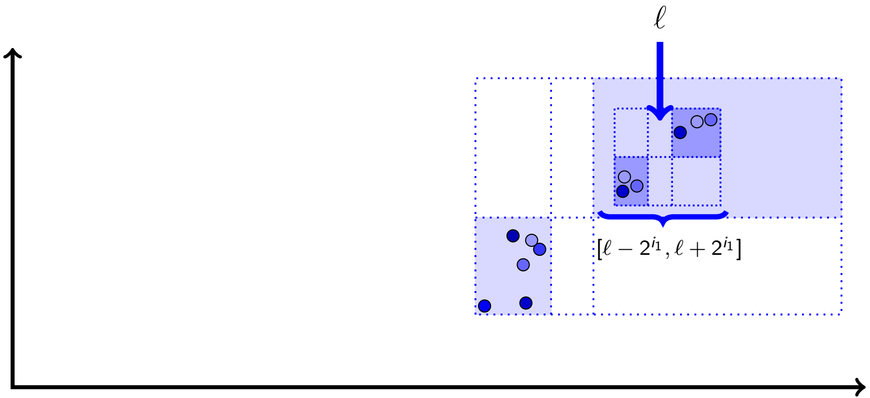
- $\Theta(1)$ queries to find ℓ as in figure.

Finding (123) with $O(\log n)$ queries



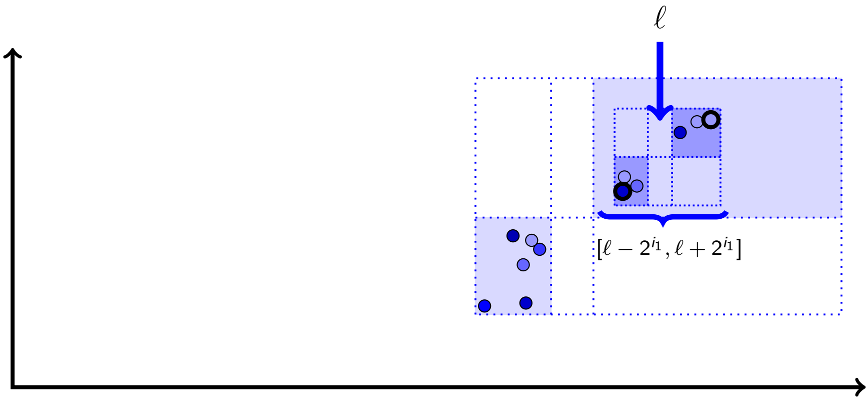
- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Finding (123) with $O(\log n)$ queries



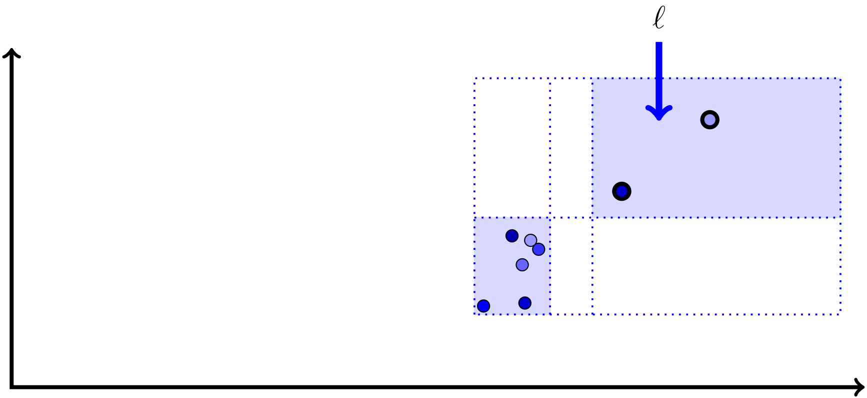
- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Finding (123) with $O(\log n)$ queries



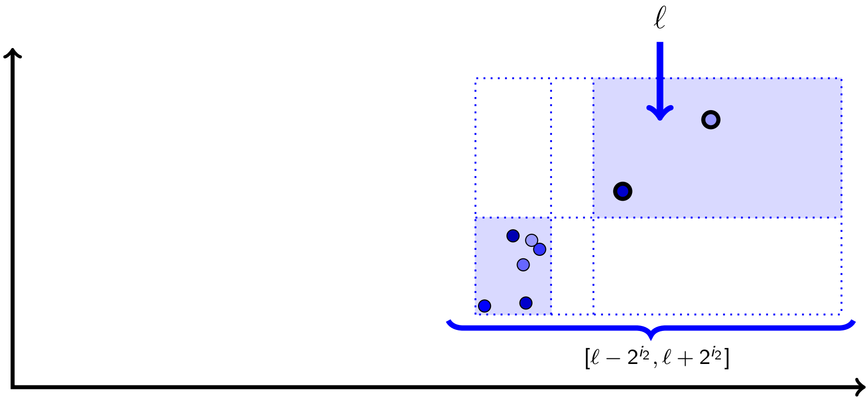
- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Finding (123) with $O(\log n)$ queries



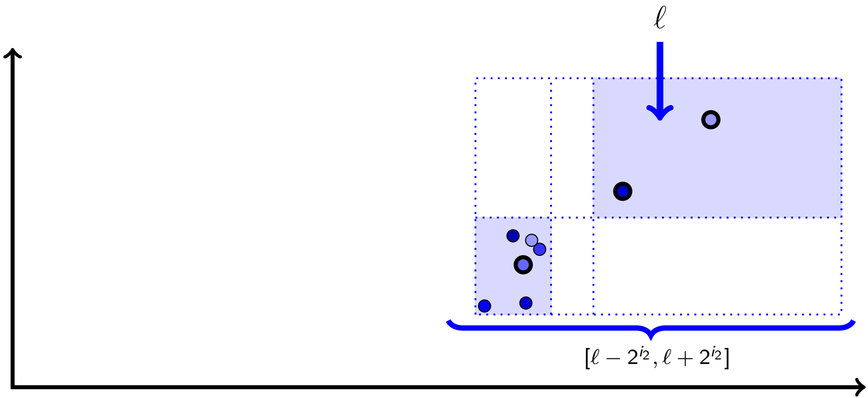
- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Finding (123) with $O(\log n)$ queries



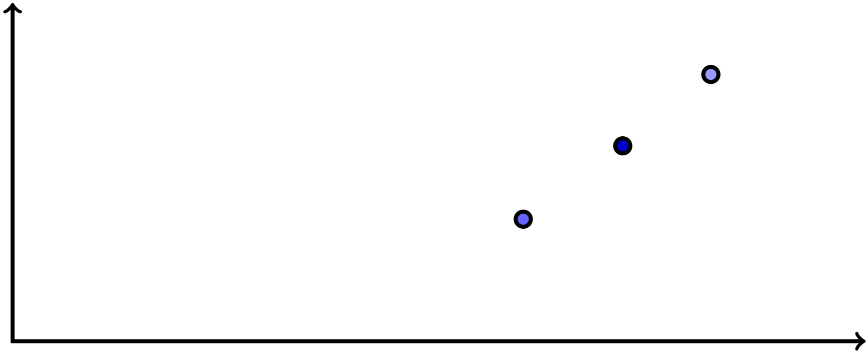
- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Finding (123) with $O(\log n)$ queries



- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Finding (123) with $O(\log n)$ queries



- $\Theta(1)$ queries to find ℓ as in figure.
- For each $i \in [\log n]$: make $\Theta(1)$ queries in $[\ell - 2^i, \ell + 2^i]$.

Open problems

Open problems

- k not fixed?

Open problems

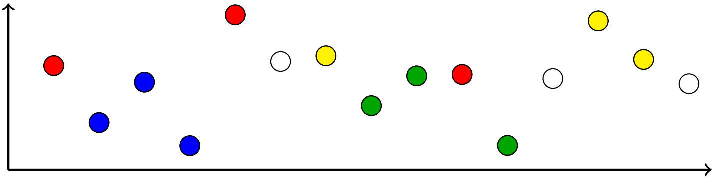
- k not fixed?
- **Testing for other permutations.**

Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.

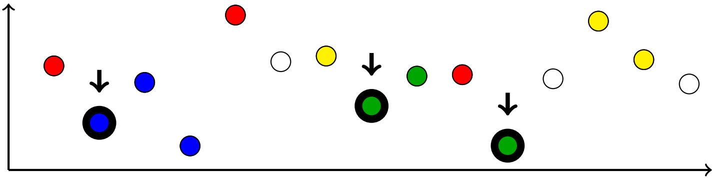
Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.



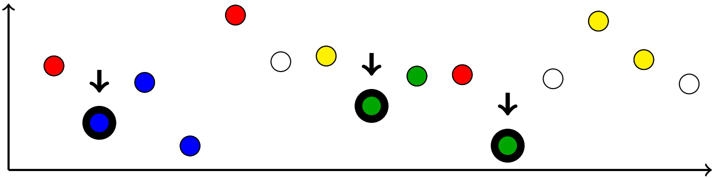
Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.



Open problems

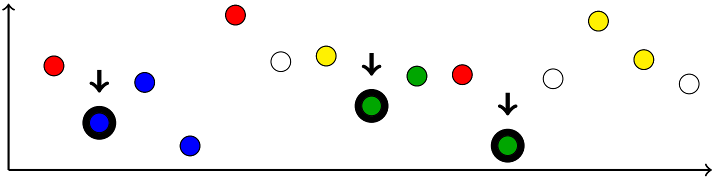
- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.



NRRS '17. If π not monotone, need $\Omega(\sqrt{n})$ non-adaptive queries.

Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.

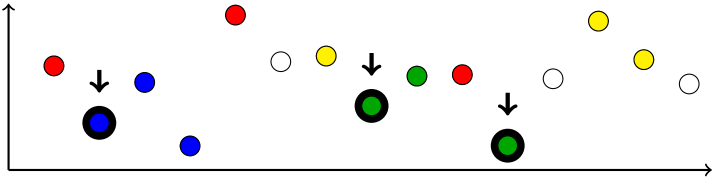


NRRS '17. If π not monotone, need $\Omega(\sqrt{n})$ non-adaptive queries.

Can π-freeness be tested adaptively in polylog n queries?

Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.



NRRS '17. If π not monotone, need $\Omega(\sqrt{n})$ non-adaptive queries.

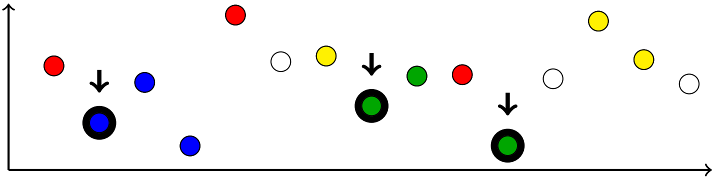
Can π-freeness be tested adaptively in polylog n queries?

- Finding a π-copy (length k) in a permutation of length n:

Fox, '13. $2^{O(k^2)} n$.

Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.



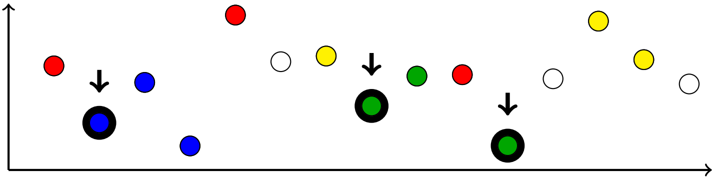
NRRS '17. If π not monotone, need $\Omega(\sqrt{n})$ non-adaptive queries.

Can π-freeness be tested adaptively in polylog n queries?

- Finding a π-copy (length k) in a permutation of length n: **Fox, '13.** $2^{O(k^2)}n$. Better algorithms?

Open problems

- k not fixed?
- **Testing for other permutations.** E.g. $\pi = (231)$.



NRRS '17. If π not monotone, need $\Omega(\sqrt{n})$ non-adaptive queries.

Can π-freeness be tested adaptively in polylog n queries?

- Finding a π-copy (length k) in a permutation of length n:

Fox, '13. $2^{O(k^2)}n$. Better algorithms?

Thank you!!!