Property testing against adversarial erasures

Nithin Varma
Boston University

Joint work with Kashyap Dixit, Sofya Raskhodnikova and Abhradeep Thakurta.
Accepted to SIAM Journal of Computing.

January 8, 2018

Talk outline

- Property testing

Talk outline

- Property testing
- Erasure-resilient property testing

Talk outline

- Property testing
- Erasure-resilient property testing
- Overview of results

Talk outline

- Property testing
- Erasure-resilient property testing
- Overview of results
- Erasure-resilient monotonicity tester

Talk outline

- Property testing
- Erasure-resilient property testing
- Overview of results
- Erasure-resilient monotonicity tester
- Current and future directions

Property testing

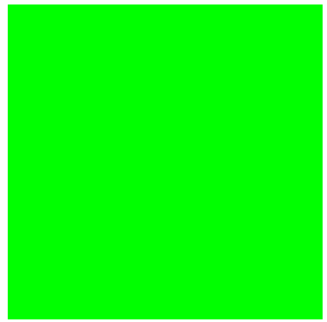
[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.

Property testing

[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.

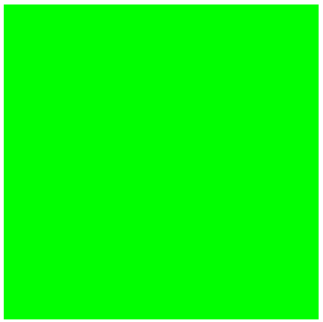


Green square

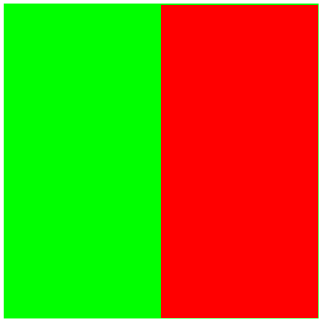
Property testing

[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.



Green square



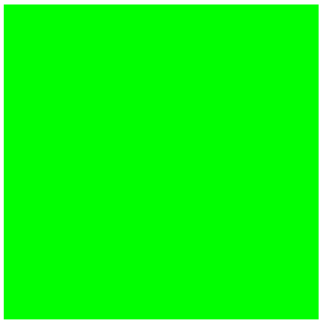
$\frac{1}{2}$-far from green square

Property testing

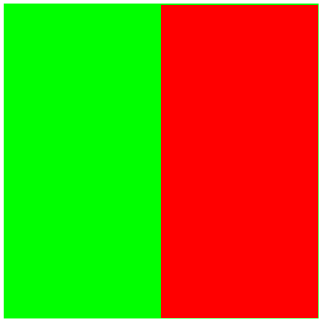
[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.

	-3	-2	-1	0	2	7	12	19	27	



Green square



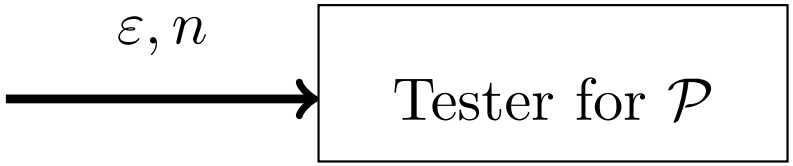
$\frac{1}{2}$-far from green square

Property testing

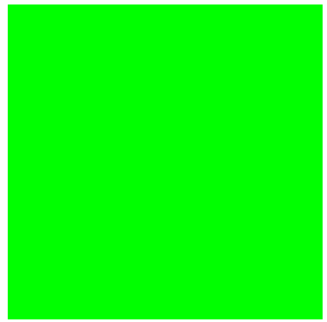
[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.

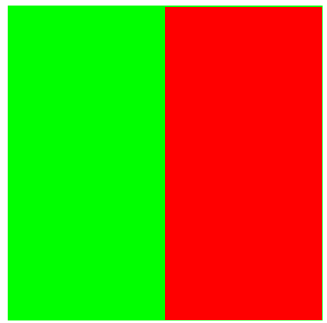
	-3	-2	-1	0	2	7	12	19	27	



(n: Size of the domain)



Green square

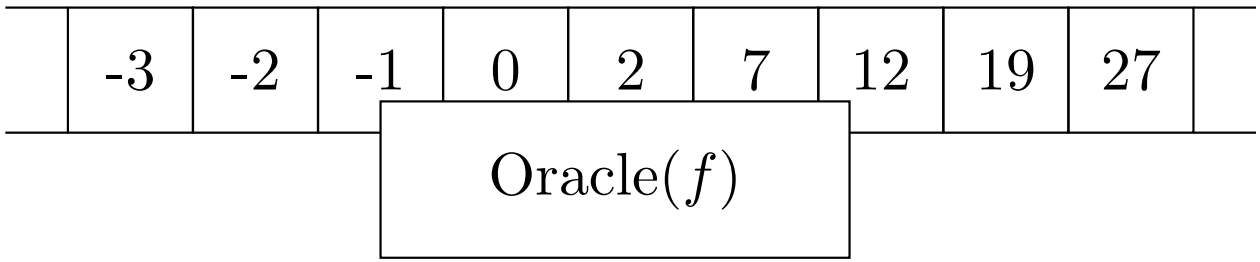


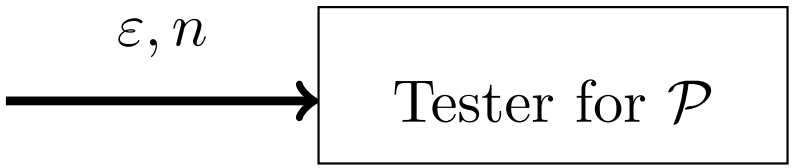
$\frac{1}{2}$-far from green square

Property testing

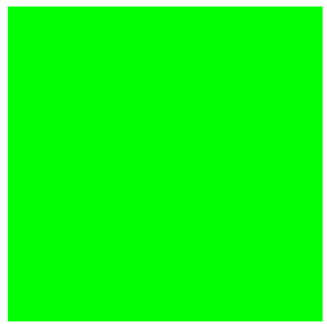
[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.

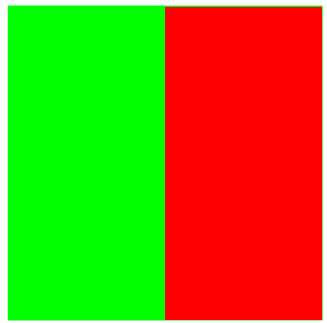




(n: Size of the domain)



Green square

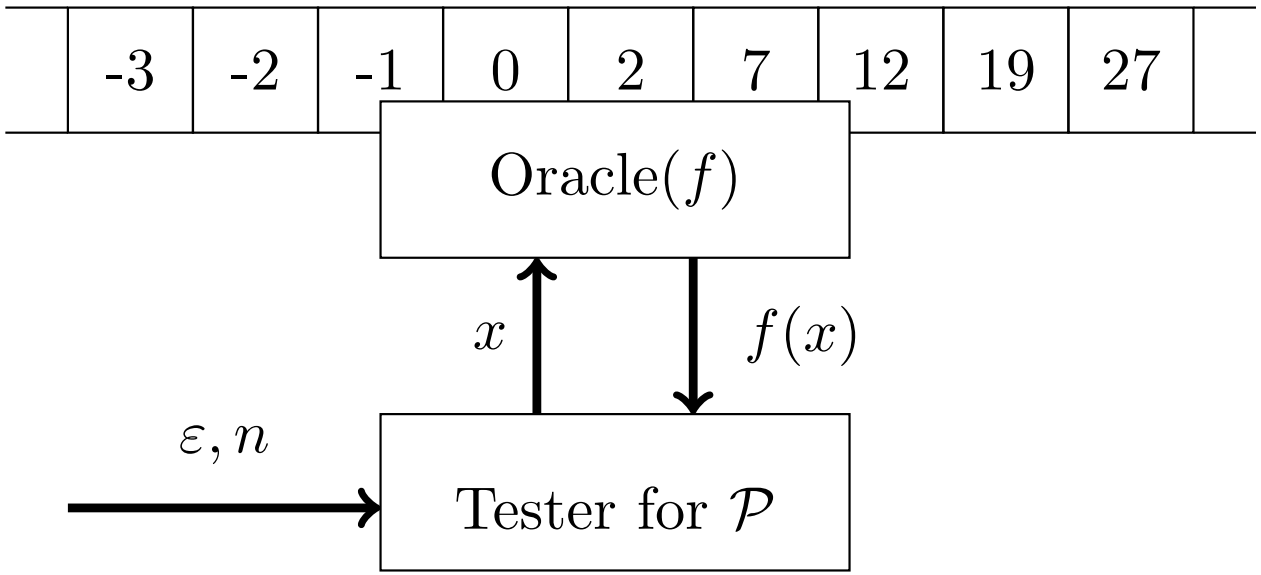


$\frac{1}{2}$-far from green square

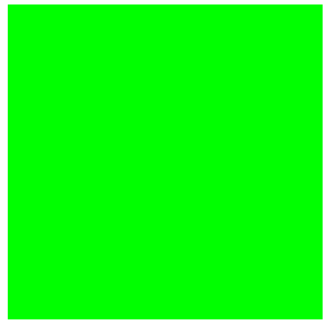
Property testing

[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

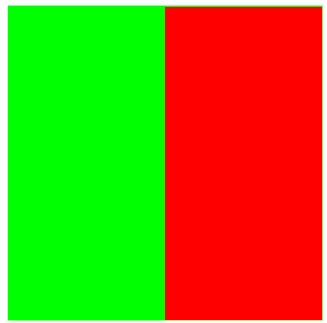
Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.



(n: Size of the domain)



Green square

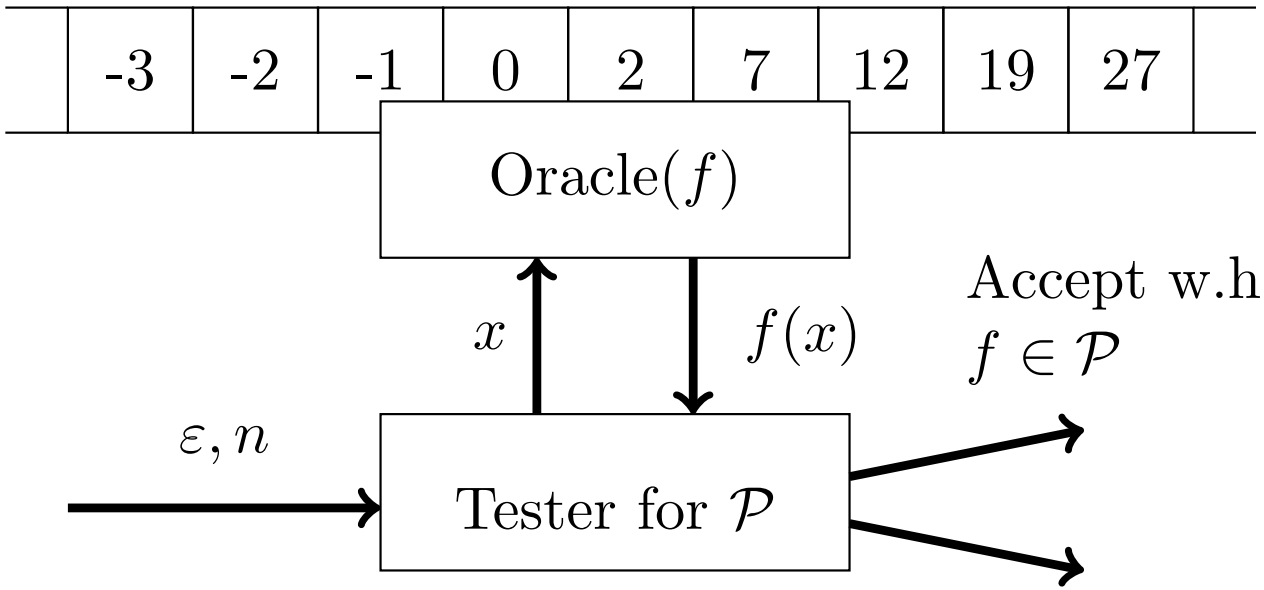


$\frac{1}{2}$-far from green square

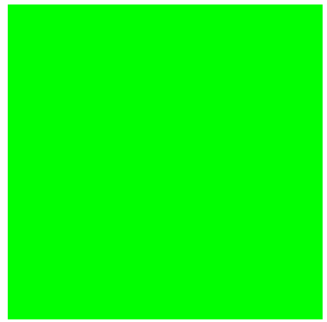
Property testing

[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

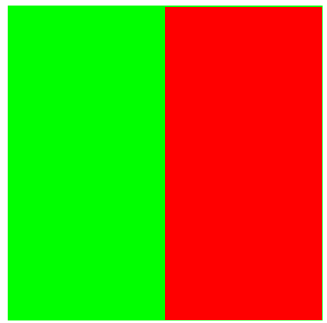
Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.



(n: Size of the domain)



Green square

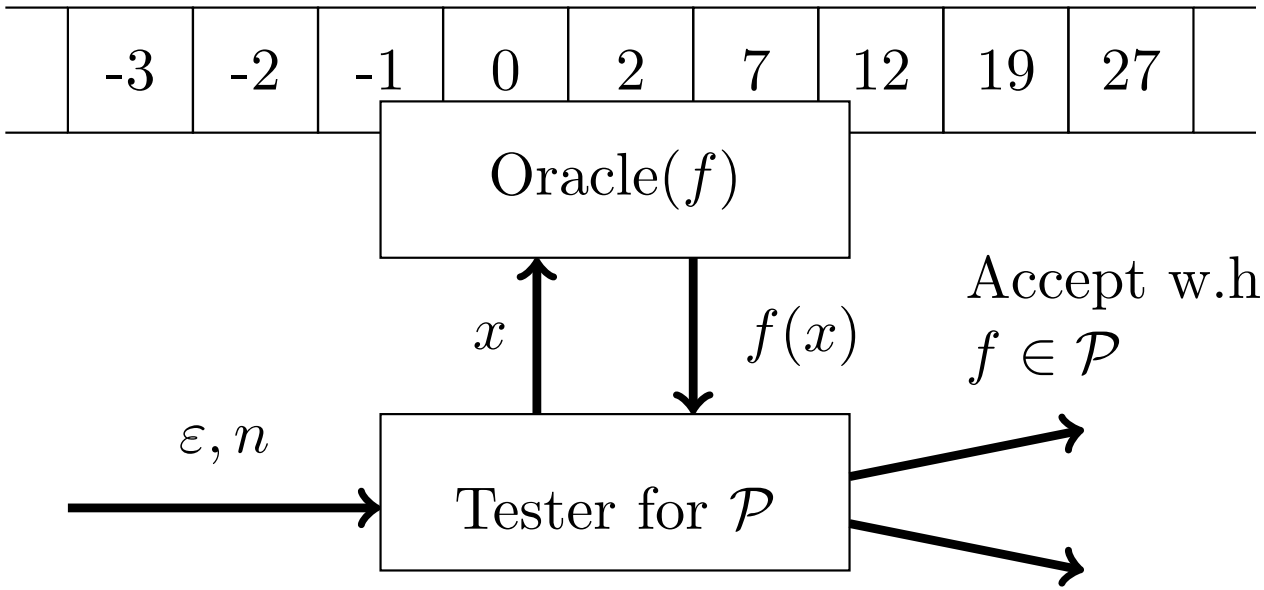


$\frac{1}{2}$-far from green square

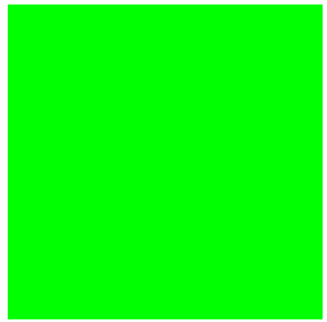
Property testing

[Rubinfeld and Sudan '96, Goldreich, Goldwasser and Ron '98]

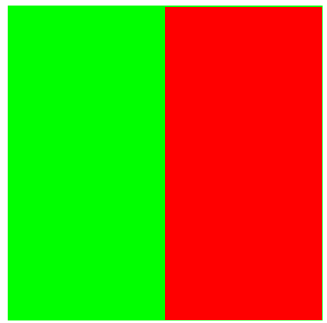
Decide w.p. $\geq 2/3$ if a function f satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$.



(n: Size of the domain)



Green square



$\frac{1}{2}$-far from green square

Assumption: Oracle returns function values at all queried points.

Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).

Erasure-resilient property testing

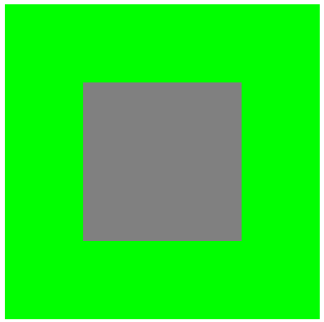
[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).

Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).

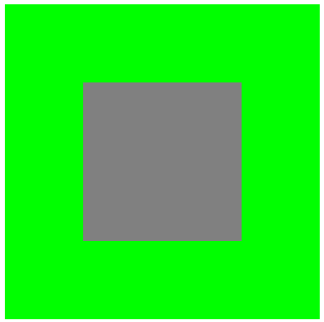


Green square

Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).



Green square

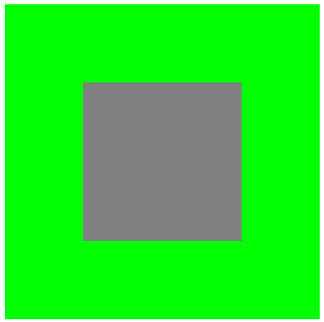
$\frac{1}{2}$-far from green square

Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).

	-3	$\perp$	-1	0	2	$\perp$	12	19	$\perp$	



Green square

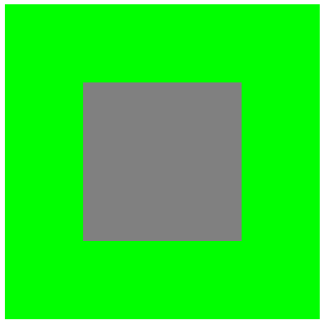
$\frac{1}{2}$-far from green square

Erasure-resilient property testing

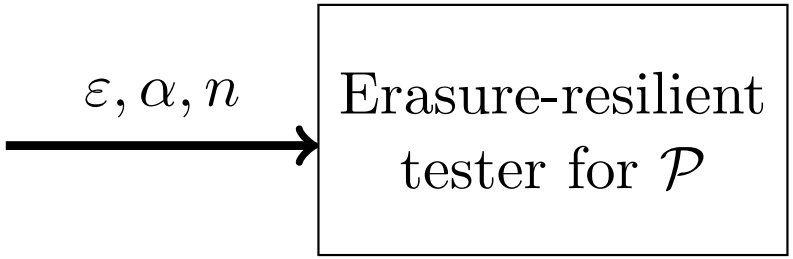
[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).

	-3	$\perp$	-1	0	2	$\perp$	12	19	$\perp$	



Green square

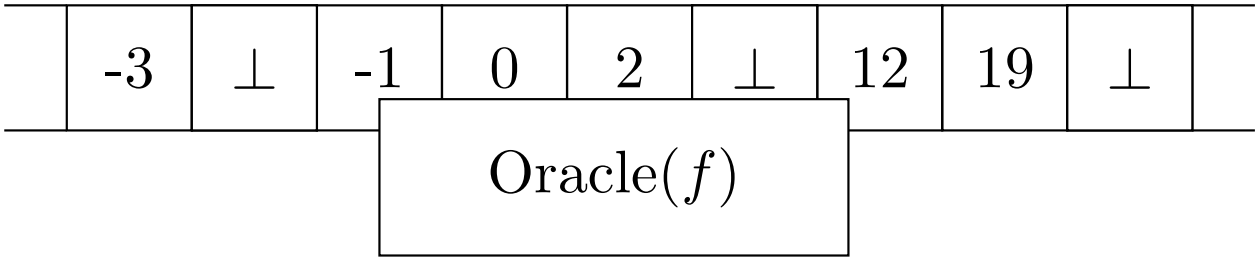


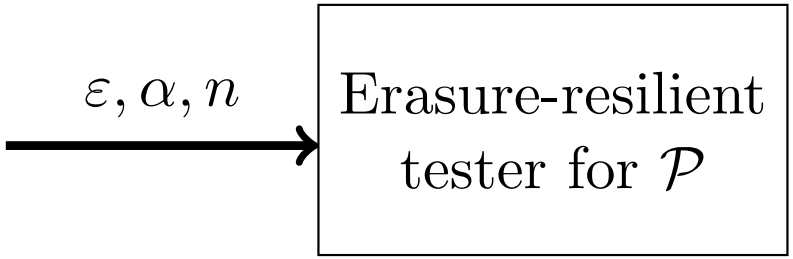
$\frac{1}{2}$-far from green square

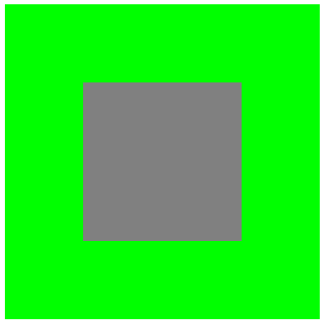
Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. $2/3$ if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).







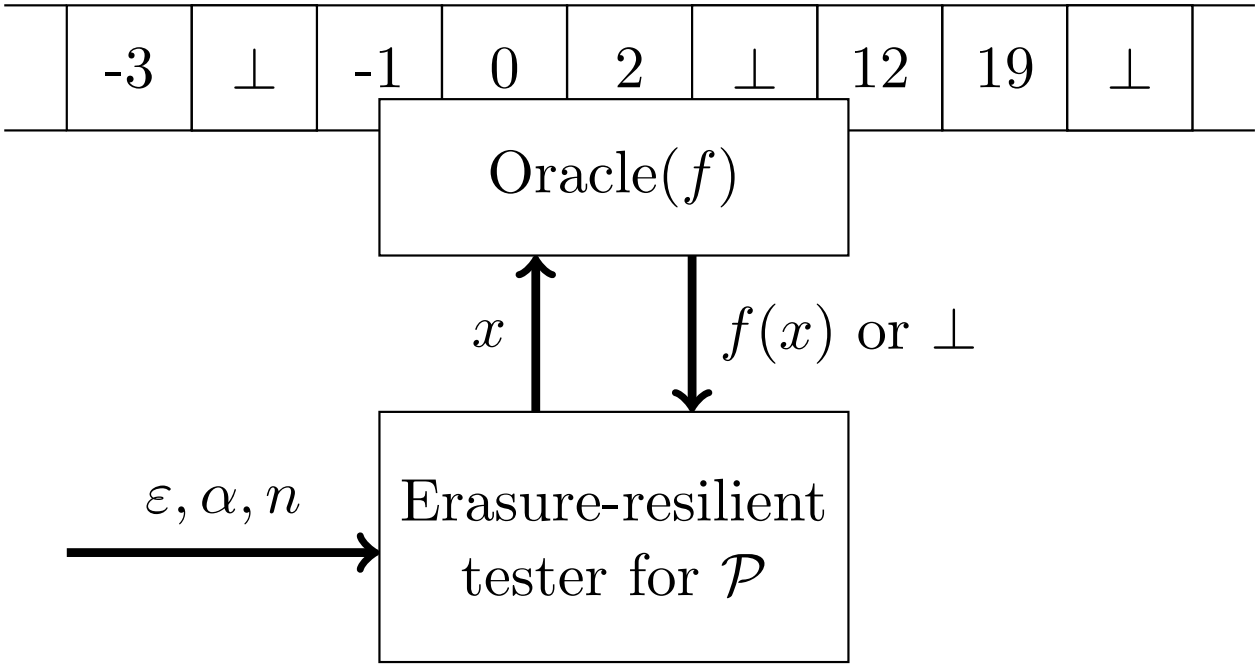
Green square

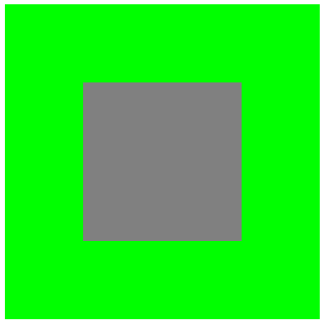
$\frac{1}{2}$-far from green square

Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. 2/3 if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).





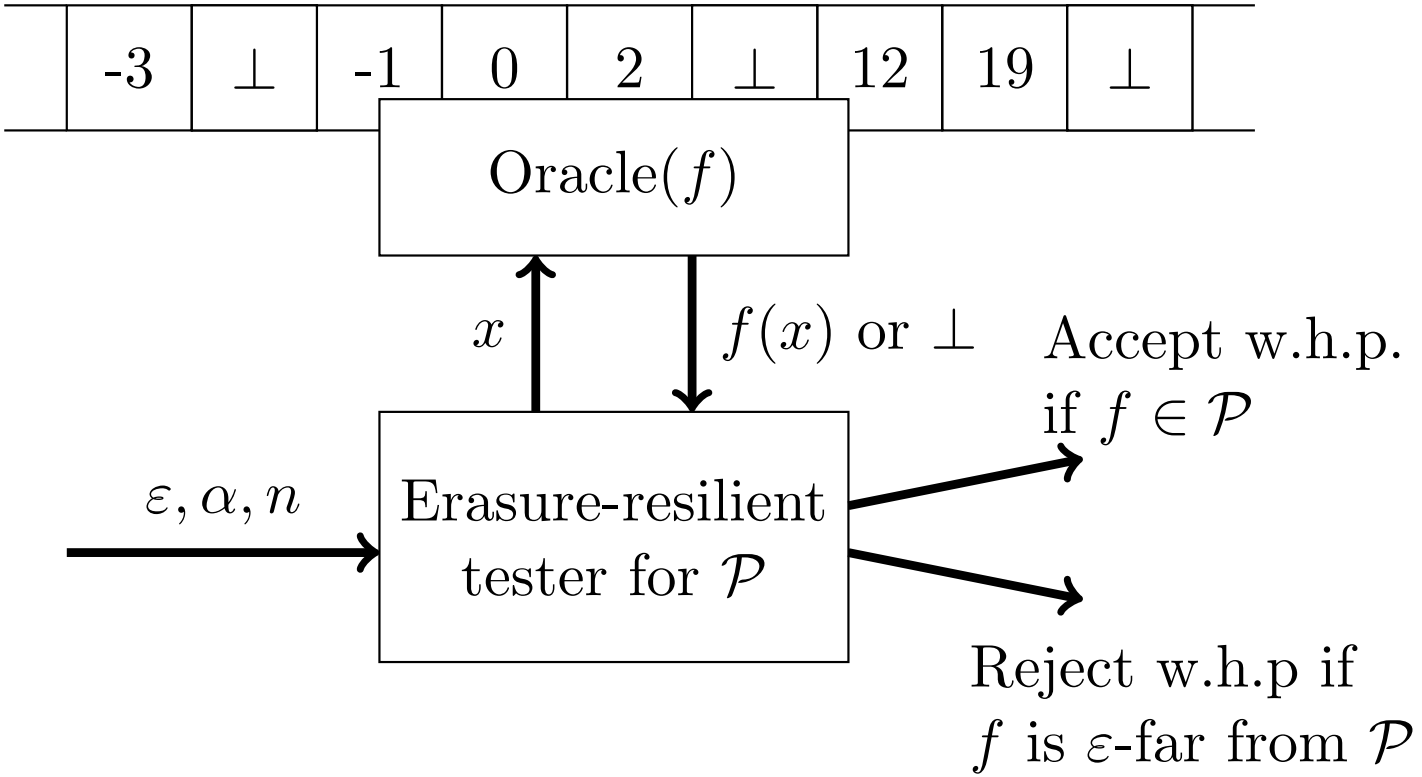
Green square

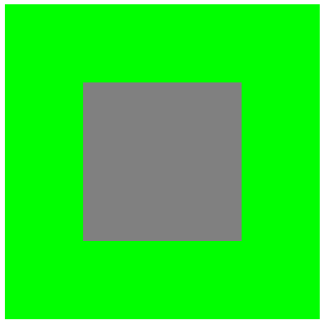
$\frac{1}{2}$-far from green square

Erasure-resilient property testing

[Dixit Raskhodnikova Thakurta Varma]

Decide w.p. 2/3 if a function f, with $\leq \alpha$ fraction of values erased, satisfies a property $\mathcal{P}$ or is ε-far from $\mathcal{P}$ (w.r.t nonerased points).





Green square

$\frac{1}{2}$-far from green square

Motivation: Some function values could be protected for privacy, or erased by an adversary.

Features of our model

- Function values could be erased adversarially (worst case).

Features of our model

- Function values could be erased adversarially (worst case).
- Tester does not know the locations of erasures in advance.

Features of our model

- Function values could be erased adversarially (worst case).
- Tester does not know the locations of erasures in advance.
- Falls in between standard property testing and tolerant property testing [Parnas, Ron and Rubinfeld 06].

Features of our model

- Function values could be erased adversarially (worst case).
- Tester does not know the locations of erasures in advance.
- Falls in between standard property testing and tolerant property testing [Parnas, Ron and Rubinfeld 06].
  - Standard testing : All function values are present and correct.
  - Tolerant testing : Some function values are adversarially corrupted.
  - Erasure-resilient testing : Some function values are adversarially erased.

Properties that we study

1. Monotonicity, Lipschitz property and convexity of real-valued functions.
2. Widely studied in property testing

   [GGLRS00, DDGLRRS99, EKKRV00, FLNRRS02, PRR03, F04, PRR06, BGJRW09, BCGM12, BBM12, CS13a, CS13b, JR13, CST14, BerRY14, BlaRY14, CDST15, KMS15, CDJS15, BB16, ..].

Properties that we study

1. Monotonicity, Lipschitz property and convexity of real-valued functions.
2. Widely studied in property testing

   [GGLRS00, DDGLRRS99, EKKRV00, FLNRRS02, PRR03, F04, PRR06, BGJRW09, BCGM12, BBM12, CS13a, CS13b, JR13, CST14, BerRY14, BlaRY14, CDST15, KMS15, CDJS15, BB16, ..].

A $f : [n] \to \mathbb{R}$ is

- **monotone** if $x < y \implies f(x) \leq f(y)$ for all $x, y \in [n]$.

Properties that we study

1. Monotonicity, Lipschitz property and convexity of real-valued functions.
2. Widely studied in property testing

   [GGLRS00, DDGLRRS99, EKKRV00, FLNRRS02, PRR03, F04, PRR06, BGJRW09, BCGM12, BBM12, CS13a, CS13b, JR13, CST14, BerRY14, BlaRY14, CDST15, KMS15, CDJS15, BB16, ..].

A $f : [n] \to \mathbb{R}$ is

- **monotone** if $x < y \implies f(x) \leq f(y)$ for all $x, y \in [n]$.
- c-**Lipschitz** if $|f(x) - f(y)| \leq c \cdot |x - y|$ for all $x, y \in [n]$.

Properties that we study

1. Monotonicity, Lipschitz property and convexity of real-valued functions.
2. Widely studied in property testing

   [GGLRS00, DDGLRRS99, EKKRV00, FLNRRS02, PRR03, F04, PRR06, BGJRW09, BCGM12, BBM12, CS13a, CS13b, JR13, CST14, BerRY14, BlaRY14, CDST15, KMS15, CDJS15, BB16, ..].

A $f : [n] \to \mathbb{R}$ is

- **monotone** if $x < y \implies f(x) \leq f(y)$ for all $x, y \in [n]$.
- c-**Lipschitz** if $|f(x) - f(y)| \leq c \cdot |x - y|$ for all $x, y \in [n]$.
- **convex** if $x < y < z \implies \frac{f(y)-f(x)}{y-x} \leq \frac{f(z)-f(y)}{z-y}$ for all $x, y, z \in [n]$.

Overview of our results

- Black box transformation of some ‘simple’ standard testers to efficient erasure-resilient testers.
- Efficient erasure-resilient testers in cases where our transformation does not apply.
- Existence of a property that has an ‘efficient’ standard tester and no ‘efficient’ erasure-resilient tester.

Our results: A black box transformation

- Makes uniform testers for extendable properties α-erasure-resilient.
- Uses the original testers as black box.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for all $\alpha \in [0, 1)$.

Our results: A black box transformation

- Makes uniform testers for extendable properties α-erasure-resilient.
- Uses the original testers as black box.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for all $\alpha \in [0, 1)$.

- Applies to:
  - Monotonicity over general partial orders

    [Fischer Lehman Newman Raskhodnikova Rubinfeld Samorodnitsky 02].
  - Convexity of black and white images

    [Berman Murzabulatov Raskhodnikova 15].
  - Boolean functions having at most k alternations in values.

A limitation of our black box transformation

Example: Testing sortedness of n-length arrays

A limitation of our black box transformation

Example: Testing sortedness of n-length arrays

- Every uniform tester requires $\Omega(\sqrt{n})$ queries.

A limitation of our black box transformation

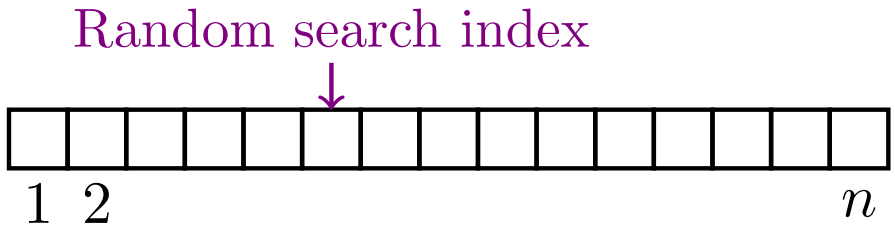
Example: Testing sortedness of n-length arrays

- Every uniform tester requires $\Omega(\sqrt{n})$ queries.
- Better (optimal) tester that makes $O(\log n)$ queries [Ergün Kannan Kumar Rubinfeld Vishwanathan 00].

A limitation of our black box transformation

Example: Testing sortedness of n-length arrays

- Every uniform tester requires $\Omega(\sqrt{n})$ queries.
- Better (optimal) tester that makes $O(\log n)$ queries [Ergün Kannan Kumar Rubinfeld Vishwanathan 00].

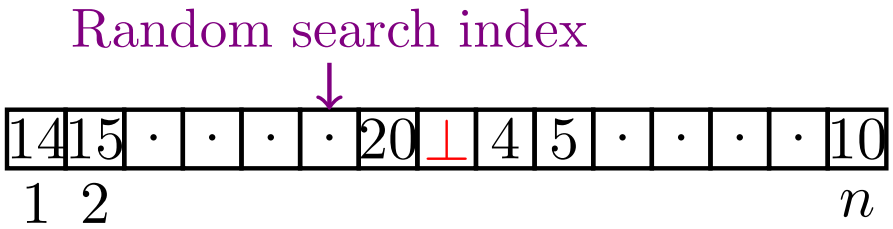


- Perform a binary search on $[1..n]$ for a random search index.
- Query the points along the search path.
- Reject if there are array values that violate sortedness.

A limitation of our black box transformation

Example: Testing sortedness of n-length arrays

- Every uniform tester requires $\Omega(\sqrt{n})$ queries.
- Better (optimal) tester that makes $O(\log n)$ queries [Ergün Kannan Kumar Rubinfeld Vishwanathan 00].



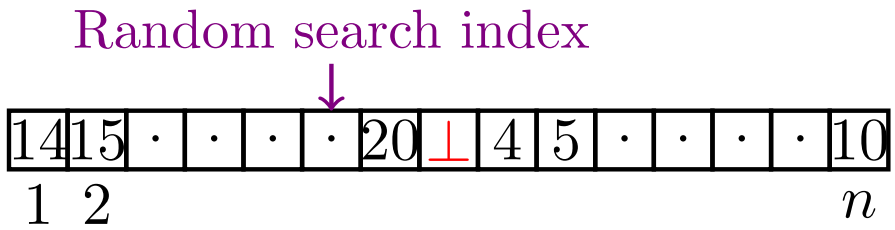
  - Perform a binary search on $[1..n]$ for a random search index.
  - Query the points along the search path.
  - Reject if there are array values that violate sortedness.

- Just one erasure breaks this tester.

A limitation of our black box transformation

Example: Testing sortedness of n-length arrays

- Every uniform tester requires $\Omega(\sqrt{n})$ queries.
- Better (optimal) tester that makes $O(\log n)$ queries [Ergün Kannan Kumar Rubinfeld Vishwanathan 00].



  - Perform a binary search on $[1..n]$ for a random search index.
  - Query the points along the search path.
  - Reject if there are array values that violate sortedness.

- Just one erasure breaks this tester.

Some of the known optimal testers for monotonicity, Lipschitz property and convexity also break on a constant number of erasures.

Our results

Erasure-resilient testers for monotonicity, Lipschitz property and convexity

For functions $f : [n] \to \mathbb{R}$

- α-erasure-resilient testers for monotonicity, Lipschitz property and convexity.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for all $\alpha \in [0, 1)$.

Our results

Erasure-resilient testers for monotonicity, Lipschitz property and convexity

For functions $f : [n] \to \mathbb{R}$

- α-erasure-resilient testers for monotonicity, Lipschitz property and convexity.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for all $\alpha \in [0, 1)$.

For functions $f : [n]^d \to \mathbb{R}$

- α-erasure-resilient testers for monotonicity and Lipschitz property.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for $\alpha = O\left(\frac{\varepsilon}{d}\right)$.

Our results

Erasure-resilient testers for monotonicity, Lipschitz property and convexity

For functions $f : [n] \to \mathbb{R}$

- α-erasure-resilient testers for monotonicity, Lipschitz property and convexity.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for all $\alpha \in [0, 1)$.

For functions $f : [n]^d \to \mathbb{R}$

- α-erasure-resilient testers for monotonicity and Lipschitz property.

$O(\frac{1}{1-\alpha})$ factor query complexity overhead for $\alpha = O\left(\frac{\varepsilon}{d}\right)$.

- Hard example for our algorithms.
  - Our algorithms will not detect violations if $\alpha = \Omega(\frac{\varepsilon}{\sqrt{d}})$.

Our results

Separation of erasure-resilient testing from standard testing

There exists a property $\mathcal{P}$ and a constant $c > 0$ such that

- standard ε-testing of $\mathcal{P}$ with $O\left(\frac{1}{\varepsilon}\right)$ queries
- α-erasure-resilient testing of $\mathcal{P}$ needs $\Omega(n^c)$ queries $\forall$ constant α.

Erasure-resilient testing is much harder than standard testing in general.

Summary of our contributions

- Definition of a model of property testing that accounts for erasures in the input data.
- A black box transformation from some simple property testers to erasure-resilient testers.
- Efficient erasure-resilient testers for monotonicity, Lipschitz property and convexity.
- A strong separation between testing in the presence of erasures and testing in the absence of erasures.

Erasure-resilient monotonicity tester

Theorem

There exists an α-erasure-resilient ε-tester for monotonicity of functions $f : [n] \to \mathbb{R}$ that makes $O\left(\frac{\log n}{\varepsilon(1-\alpha)}\right)$ queries for all $\alpha \in [0, 1), \varepsilon \in (0, 1)$.

Erasure-resilient monotonicity tester

Theorem

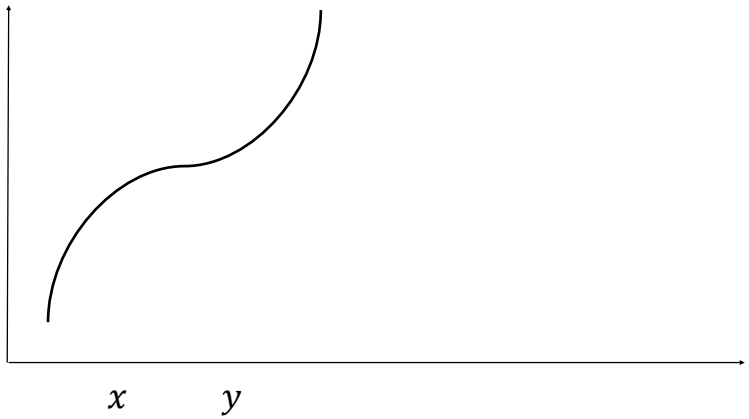
There exists an α-erasure-resilient ε-tester for monotonicity of functions $f : [n] \to \mathbb{R}$ that makes $O\left(\frac{\log n}{\varepsilon(1-\alpha)}\right)$ queries for all $\alpha \in [0, 1), \varepsilon \in (0, 1)$.

- ε-testing of monotonicity using $\Theta(\frac{\log n}{\varepsilon})$ queries [Ergün Kannan Kumar Rubinfeld Vishwanathan 00, Fischer 04].

Monotonicity

A partially erased function $f : [n] \to \mathbb{R}$ is monotone, if for all $x, y \in [n]$ that are nonerased

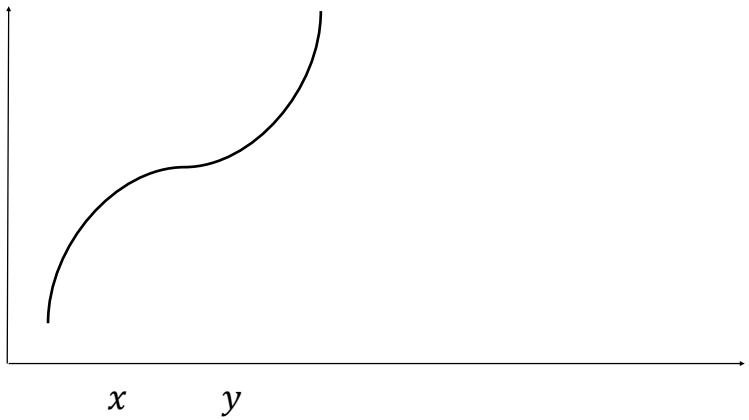
$$x < y \implies f(x) \leq f(y).$$



Monotonicity

A partially erased function $f : [n] \to \mathbb{R}$ is monotone, if for all $x, y \in [n]$ that are nonerased

$$x < y \implies f(x) \leq f(y).$$

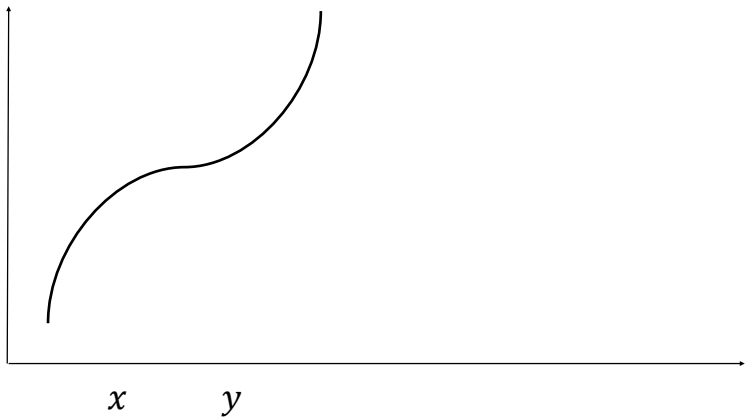


- **Violation:** Two nonerased points $x, y \in [n]$ such that $x < y$ and $f(x) > f(y)$.

Monotonicity

A partially erased function $f : [n] \to \mathbb{R}$ is monotone, if for all $x, y \in [n]$ that are nonerased

$$x < y \implies f(x) \leq f(y).$$

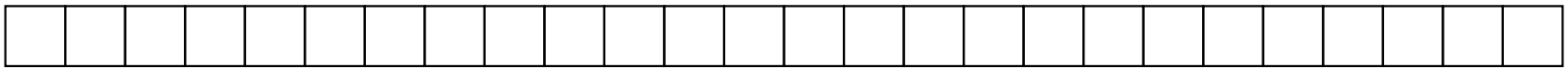


- **Violation:** Two nonerased points $x, y \in [n]$ such that $x < y$ and $f(x) > f(y)$.
- Monotonicity of functions over $[n] \equiv$ Sortedness of n-length arrays.

Our erasure-resilient sortedness tester

Our erasure-resilient sortedness tester

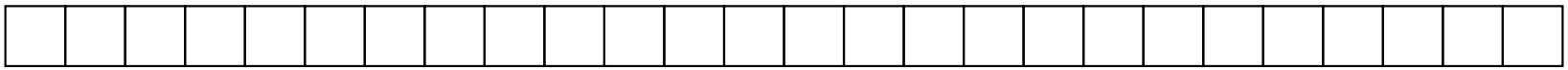
Input: $\alpha \in [0, 1), \varepsilon \in (0, 1)$; query access to array



Our erasure-resilient sortedness tester

Input: $\alpha \in [0, 1), \varepsilon \in (0, 1)$; query access to array

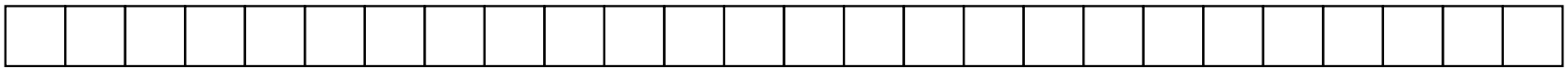
- Repeat $\Theta(1/\varepsilon)$ times:



Our erasure-resilient sortedness tester

Input: $\alpha \in [0, 1), \varepsilon \in (0, 1)$; query access to array

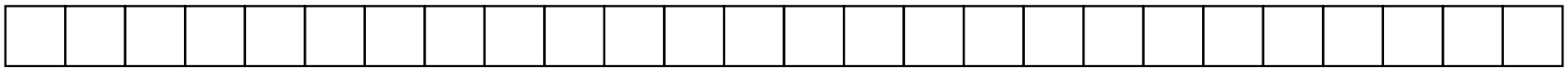
- Repeat $\Theta(1/\varepsilon)$ times:



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

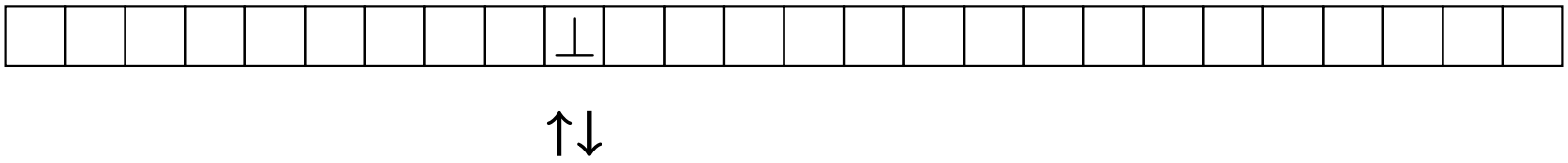
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

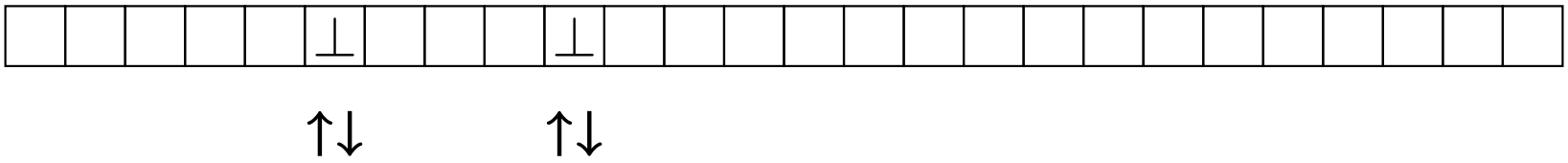
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0, 1), \varepsilon \in (0, 1)$; query access to array

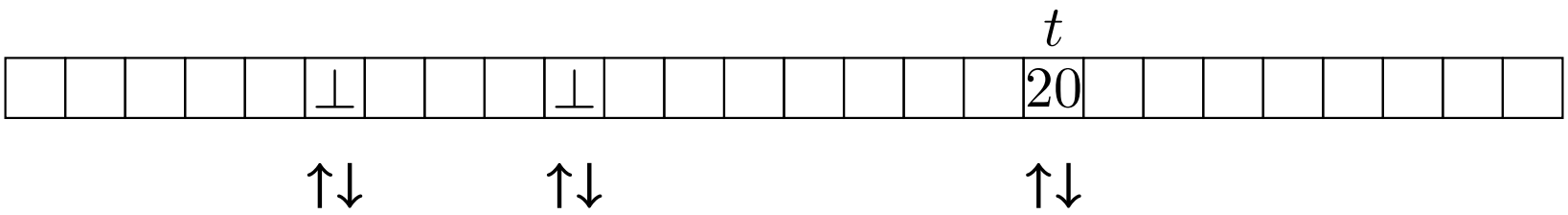
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

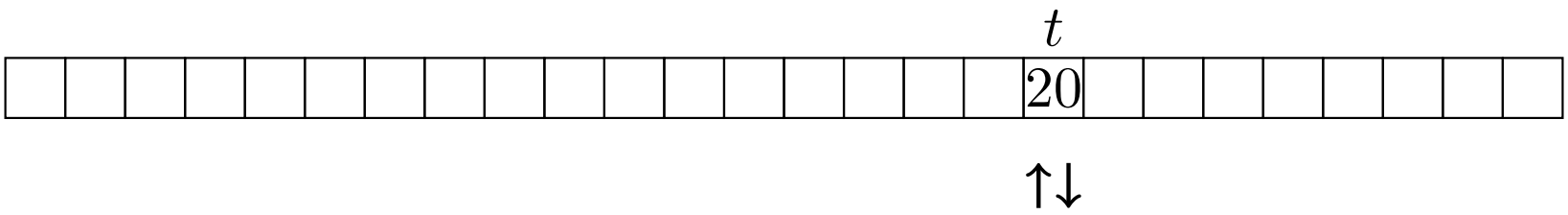
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

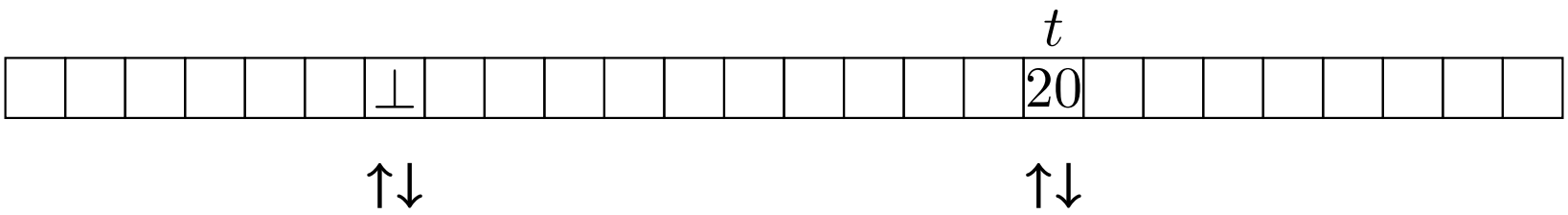
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

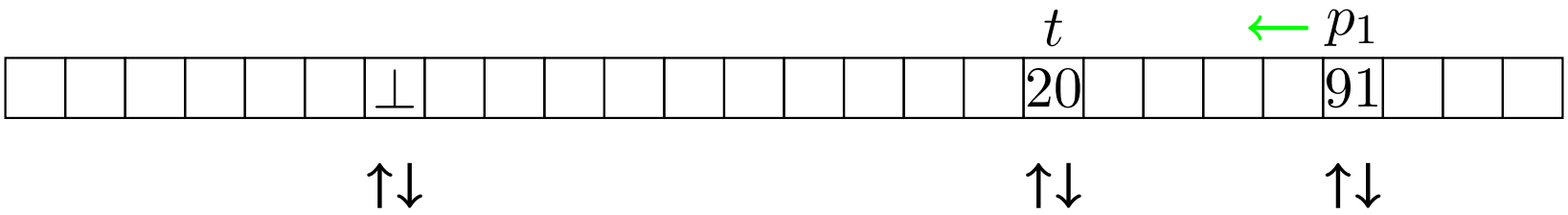
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

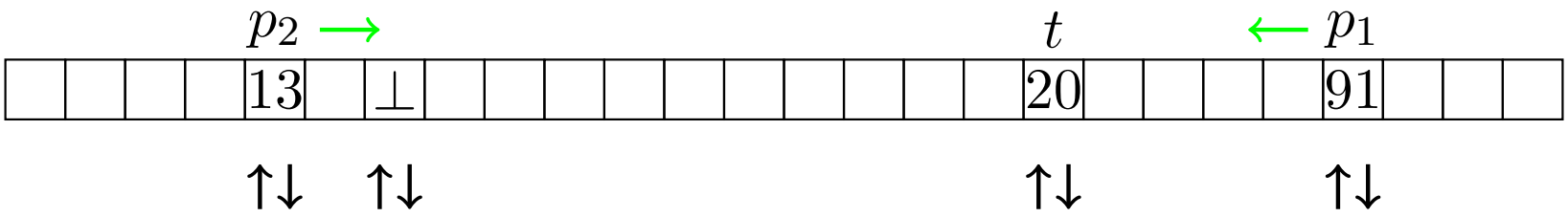
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

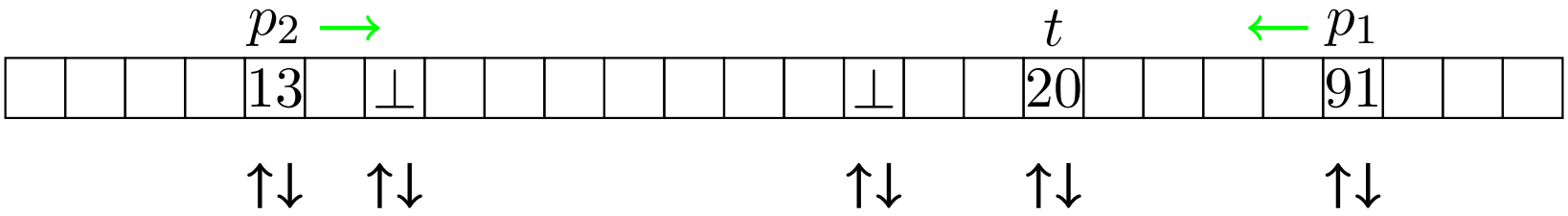
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

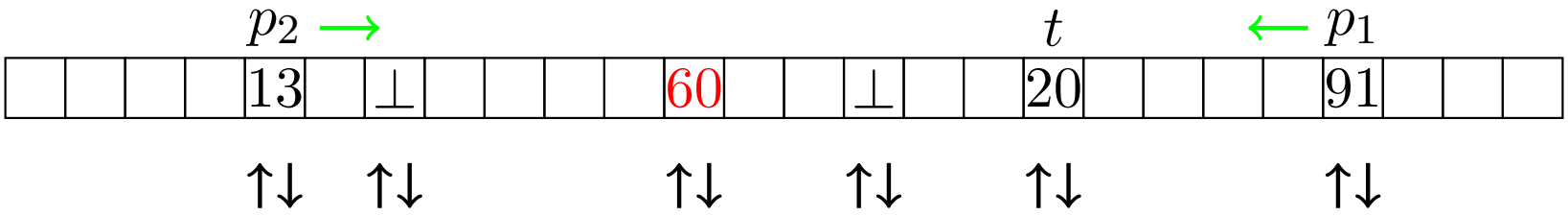
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

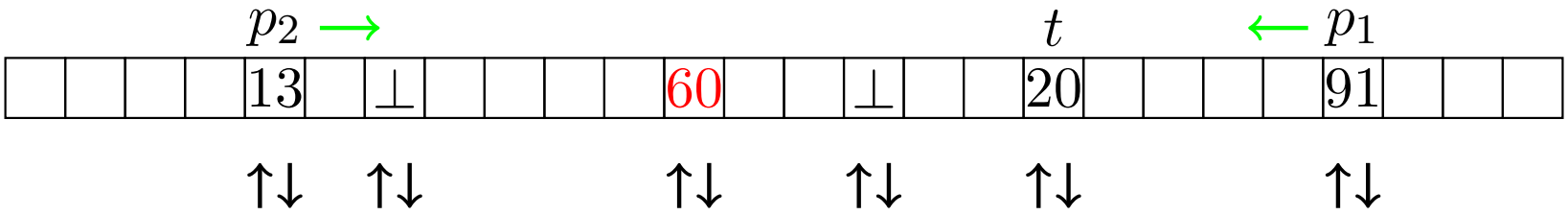
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

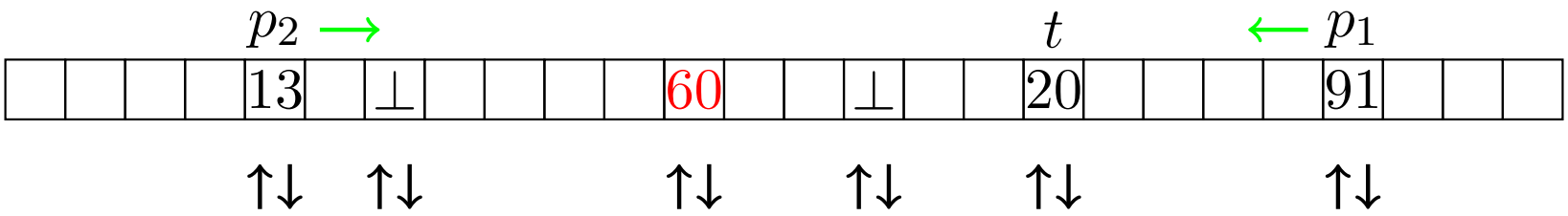
- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.
  3. Reject if there are violations along the search path.



Our erasure-resilient sortedness tester

Input: $\alpha \in [0,1), \varepsilon \in (0,1)$; query access to array

- Repeat $\Theta(1/\varepsilon)$ times:
  1. Sample until you get a nonerased search point t.
  2. Binary search for t with "uniform" nonerased 'split points'.
  3. Reject if there are violations along the search path.
- Accept if no violations were found.



Main steps of analysis

1. Array is sorted $\implies$ Tester accepts.

Main steps of analysis

1. Array is sorted $\implies$ Tester accepts.
2. Array is ε-far from sorted $\implies$ Detects violation w.p. $\geq \varepsilon$ in an iteration.

Main steps of analysis

1. Array is sorted $\implies$ Tester accepts.
2. Array is ε-far from sorted $\implies$ Detects violation w.p. $\geq \varepsilon$ in an iteration.
   - In $\Theta(1/\varepsilon)$ independent iterations:

     Tester detects a violation with high constant probability.

Main steps of analysis

1. Array is sorted $\implies$ Tester accepts.
2. Array is ε-far from sorted $\implies$ Detects violation w.p. $\geq \varepsilon$ in an iteration.
   - In $\Theta(1/\varepsilon)$ independent iterations:

     Tester detects a violation with high constant probability.
3. The expected number of queries made is $O\left(\frac{\log n}{\varepsilon(1-\alpha)}\right)$.

Main steps of analysis

1. Array is sorted $\implies$ Tester accepts.
2. Array is ε-far from sorted $\implies$ Detects violation w.p. $\geq \varepsilon$ in an iteration. (Witness lemma)
   - In $\Theta(1/\varepsilon)$ independent iterations:

     Tester detects a violation with high constant probability.
3. The expected number of queries made is $O\left(\frac{\log n}{\varepsilon(1-\alpha)}\right)$. (Query lemma)

Query lemma

Claim

The expected number of queries in one iteration is $O\left(\frac{\log n}{1-\alpha}\right)$.

Query lemma

Claim

The expected number of queries in one iteration is $O\left(\frac{\log n}{1-\alpha}\right)$.

1. Tester traverses a uniformly random search path in a random binary search tree.

Query lemma

Claim

The expected number of queries in one iteration is $O\left(\frac{\log n}{1-\alpha}\right)$.

1. Tester traverses a uniformly random search path in a random binary search tree.
2. The number of levels in a random binary search is $O(\log n)$ w.h.p.

Query lemma

Claim

The expected number of queries in one iteration is $O\left(\frac{\log n}{1-\alpha}\right)$.

1. Tester traverses a uniformly random search path in a random binary search tree.
2. The number of levels in a random binary search is $O(\log n)$ w.h.p.
3. Expected # of queries to one level of binary search is $O\left(\frac{1}{1-\alpha}\right)$.

Expected number of queries to a single level

Claim: $\mathbb{E}\left[\#\text{ of queries to one level of binary search}\right] = O\left(\frac{1}{1-\alpha}\right)$.

Expected number of queries to a single level

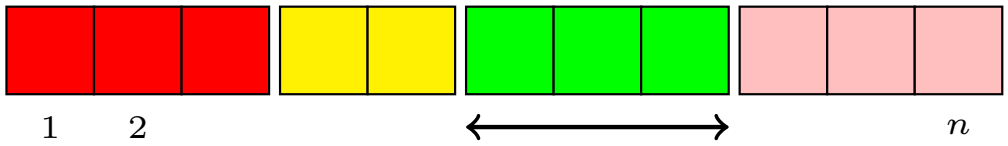
Claim: $\mathbb{E}\left[\#\text{ of queries to one level of binary search}\right] = O\left(\frac{1}{1-\alpha}\right)$.

Level k:

Expected number of queries to a single level

Claim: $\mathbb{E}[\#$ of queries to one level of binary search$] = O\left(\frac{1}{1-\alpha}\right)$.

Level k:



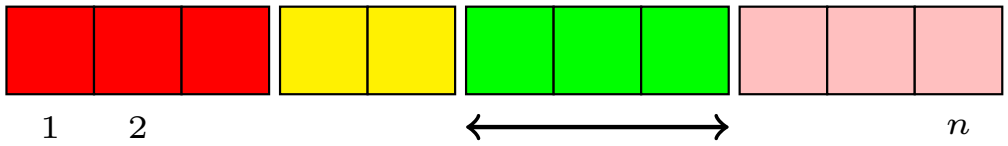
Interval I

α_I = fraction of erasures in I

Expected number of queries to a single level

Claim: $\mathbb{E}[\#$ of queries to one level of binary search$] = O\left(\frac{1}{1-\alpha}\right)$.

Level k:



Interval I

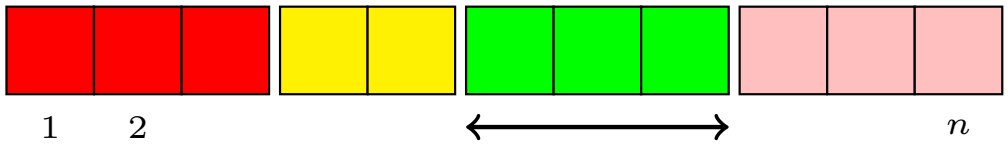
α_I = fraction of erasures in I

$$\Pr[\text{ search point is in } I] = \frac{\#\text{ nonerased points in } I}{\text{Total } \#\text{ nonerased points}} \leq \frac{|I|(1-\alpha_I)}{n(1-\alpha)}.$$

Expected number of queries to a single level

Claim: $\mathbb{E}[\#$ of queries to one level of binary search$] = O\left(\frac{1}{1-\alpha}\right)$.

Level k:



Interval I

α_I = fraction of erasures in I

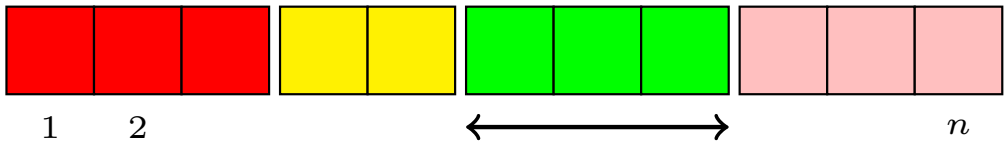
$$\Pr[\text{ search point is in } I] = \frac{\#\text{ nonerased points in } I}{\text{Total } \#\text{ nonerased points}} \leq \frac{|I|(1-\alpha_I)}{n(1-\alpha)}.$$

$$\mathbb{E}[\#\text{ queries to } I] = \frac{1}{1-\alpha_I}.$$

Expected number of queries to a single level

Claim: $\mathbb{E}[\#$ of queries to one level of binary search$] = O\left(\frac{1}{1-\alpha}\right)$.

Level k:



Interval I

α_I = fraction of erasures in I

$$\Pr[\text{ search point is in } I] = \frac{\#\text{ nonerased points in } I}{\text{Total } \#\text{ nonerased points}} \leq \frac{|I|(1-\alpha_I)}{n(1-\alpha)}.$$

$$\mathbb{E}[\#\text{ queries to } I] = \frac{1}{1-\alpha_I}.$$

$$\therefore \mathbb{E}[\#\text{ of queries to level } k] \leq \sum_{\text{intervals } I \text{ in level } k} \frac{|I|}{n(1-\alpha)} \leq \frac{1}{1-\alpha}.$$

Analysis overview

1. Array is sorted $\implies$ Tester accepts.
2. Array is ε-far from sorted $\implies$ Detects violation w.p. $\geq \varepsilon$ in an iteration.
   - In $2/\varepsilon$ independent iterations:

     $$\Pr[\text{tester does not detect a violation}] \leq (1-\varepsilon)^{2/\varepsilon} \leq \frac{1}{3}.$$

     Tester rejects with high constant probability.
3. The expected number of queries made is $O\left(\frac{\log n}{\varepsilon(1-\alpha)}\right)$.

What we showed

Theorem

There exists an α-erasure-resilient ε-tester for sortedness of n-length arrays that makes $O\left(\frac{\log n}{\varepsilon(1-\alpha)}\right)$ queries for all $\alpha \in [0,1), \varepsilon \in (0,1)$.

Ongoing work

- Separating erasures from corruptions.
  - Is tolerant testing harder than erasure-resilient testing in general?
- Adversarial erasures vs. random erasures

Future directions

- Better erasure-resilient testers for monotonicity and Lipschitz property of functions $f : [n]^d \mapsto \mathbb{R}$.
- Other models of erasures
  - Random erasures.
  - Semi-adaptive erasures.

Thank you!