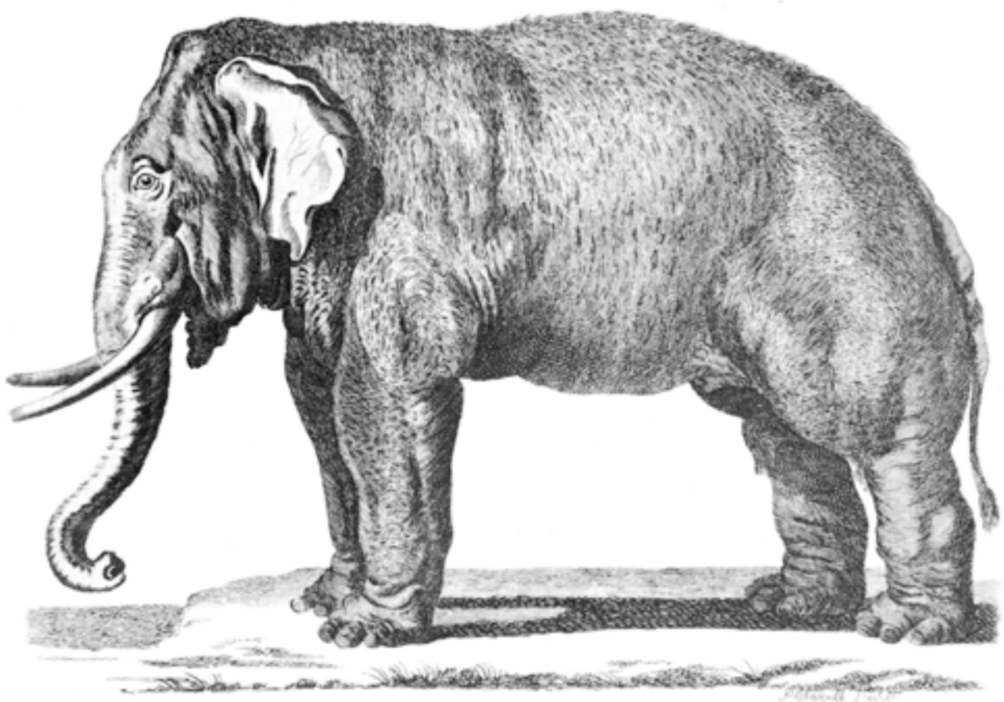
Checking & Spot-Checking the Correctness of Priority Queues

Matthew Chu & Sampath Kannan (UPenn) Andrew McGregor (UCSD)



Memory Checking

Memory Checking

- *Your resources:* A lot of cheap unreliable memory and a little expensive reliable memory.

Memory Checking

- *Your resources:* A lot of cheap unreliable memory and a little expensive reliable memory.
- *Your challenge:* Can you make use of the cheap memory? Want to identify (but not correct) any errors introduced by a malicious adversary.

Memory Checking

- *Your resources:* A lot of cheap unreliable memory and a little expensive reliable memory.
- *Your challenge:* Can you make use of the cheap memory? Want to identify (but not correct) any errors introduced by a malicious adversary.
- *Related Work:*

Program Checking	[Blum, Kannan '95]
Memory Checking	[Blum et al. '94]
Checking linked Data Structures	[Amato, Loui '94]

Priority Queues

Priority Queues

- *Priority Queue:*

  Supports a sequence of *inserts* and *extract-min's.*

  Is "correct" if each extract-min returns the smallest value inserted and not extracted.

Priority Queues

- *Priority Queue:*

  Supports a sequence of *inserts* and *extract-min's*.

  Is "correct" if each extract-min returns the smallest value inserted and not extracted.

- *Interaction Sequence:* $c_1, c_2, \ldots, c_{2n}$ where c_t is either

  (u,t) if the user inserts u at step t

  (u,t') if the user extract-min's at step t and *PQ* claims u, inserted at time t', is the min.

Priority Queues

- *Priority Queue:*

  Supports a sequence of *inserts* and *extract-min's.*

  Is "correct" if each extract-min returns the smallest value inserted and not extracted.

- *Interaction Sequence:* $c_1, c_2, \ldots, c_{2n}$ where c_t is either

  (u,t) if the user inserts u at step t

  (u,t') if the user extract-min's at step t and *PQ* claims u, inserted at time t', is the min.

- *Example:* Insert 5, Insert 4, Extract-min, Insert 7,... would correspond to the sequence (5,1), (4,2), (4,2), (7,4), ... if the *PQ* was correct.

The Checking Problem

The Checking Problem

- *Input:* A sequence $c_1, c_2, \ldots, c_{2n}$ with n inserts and n extract-mins.

The Checking Problem

- *Input:* A sequence $c_1, c_2, \ldots, c_{2n}$ with n inserts and n extract-mins.
- *Goal:* Fail the stream with high probability if it is not correct and pass otherwise.

The Checking Problem

- *Input:* A sequence $c_1, c_2, \ldots, c_{2n}$ with n inserts and n extract-mins.
- *Goal:* Fail the stream with high probability if it is not correct and pass otherwise.
- *Constraints:* The interaction sequence is observed as a stream and has limited space.

The Checking Problem

- *Input:* A sequence $c_1, c_2, \dots, c_{2n}$ with n inserts and n extract-mins.
- *Goal:* Fail the stream with high probability if it is not correct and pass otherwise.
- *Constraints:* The interaction sequence is observed as a stream and has limited space.
- We are interested in *offline* checkers that identify errors by the end of the interaction sequence.

Results

Results

- <u>*Checkers:*</u>

  A randomized, offline, $O(\sqrt{n} \log n)$-space checker that identifies errors with prob. $1-1/n$.

  Any randomized, offline checker of a "certain type" requires $\Omega(\sqrt{n})$ space.

  Online or deterministic requires $\Omega(n)$ space.

Results

- ***Checkers:***

  A randomized, offline, $O(\sqrt{n} \log n)$-space checker that identifies errors with prob. $1-1/n$.

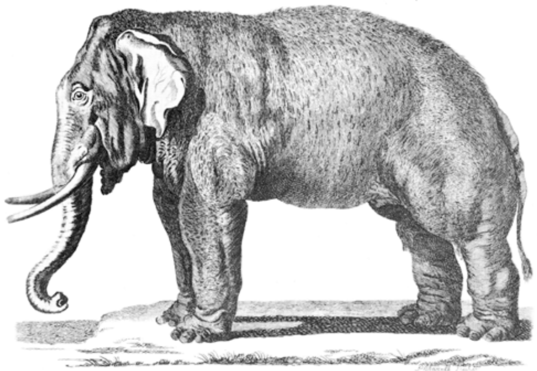
  Any randomized, offline checker of a "certain type" requires $\Omega(\sqrt{n})$ space.

  Online or deterministic requires $\Omega(n)$ space.

- ***Spot-Checker:***

  A randomized, offline, $O(\varepsilon^{-1} \log^2 n)$-space spot-checker that identifies a priority queue that is "ε-far" from correct with prob. $1-1/n$.

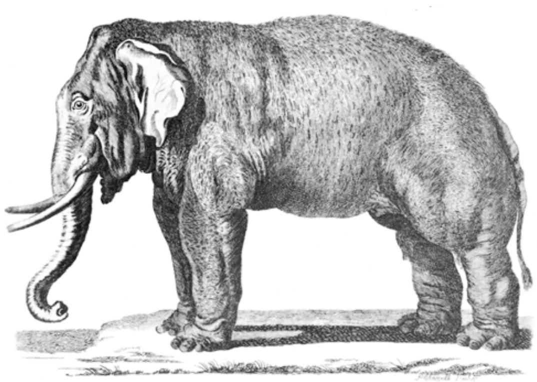
1: Preliminaries
2: Checking
3: Spot-Checking



1: Preliminaries

2: Checking

3: Spot-Checking



Correctness

Correctness

- *Thm:* An interaction sequence is correct iff it satisfies:

  $C1$: $\{(u,t)\}=\{(u,t)\}$

  $C2$: For all $c_s=(u,t)$: t<s

  $C3$: For all $c_{tb}=(u,ta)$ and $c_{sb}=(v,sa)$:

  $((u,ta) < (v,sa))$ then $(sb<ta$ or $tb<sa)$

- *Proof Idea:* If correct then clearly $C1$, $C2$, & $C3$. For other direction consider first incorrect extract-min...

Correctness

- *Thm:* An interaction sequence is correct iff it satisfies:

  $C1$: $\{(u,t)\}=\{(u,t)\}$

  $C2$: For all $c_s=(u,t)$: $t<s$

  $C3$: For all $c_{tb}=(u,ta)$ and $c_{sb}=(v,sa)$:

  $((u,ta) < (v,sa))$ then $(sb<ta$ or $tb<sa)$

- *Proof Idea:* If correct then clearly $C1$, $C2$, & $C3$. For other direction consider first incorrect extract-min...

Hashing

Hashing

- *Thm (Naor & Naor):* Can construct a hash function *h* on length *n* strings such that

  $$\Pr[h(x) = h(y)] \leq \delta \qquad \text{if } x \neq y.$$

  It uses O(lg *n*) random bits and can be constructed in O(lg *n*) space even if the characters of each string are revealed in an arbitrary order.

Hashing

- *Thm (Naor & Naor):* Can construct a hash function h on length n strings such that

$$\Pr[h(x) = h(y)] \leq \delta \quad \text{if } x \neq y.$$

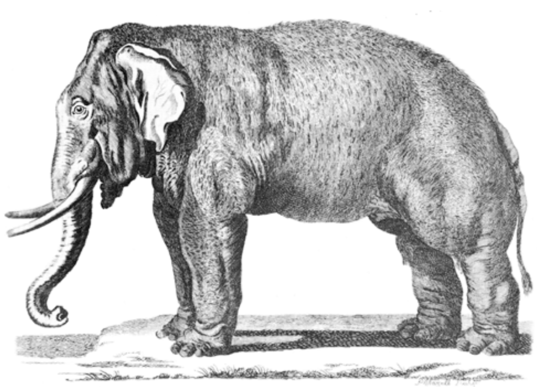
  It uses O(lg n) random bits and can be constructed in O(lg n) space even if the characters of each string are revealed in an arbitrary order.

- *What it means for us:*
  Let x_t be (u,t) if u was inserted at time t
  Let y_t be (u,t) if an extract returns (u,t)
  Hence can easily check CI: {(u,t)}={(u,t)}

1: Preliminaries

2: Checking

3: Spot-Checking



Checking Results

- *Thm:* A randomized, offline, $O(\sqrt{n} \lg n)$-space checker that identifies errors with prob. 1-1/n.
- *Thm:* Any randomized online checker that is correct with prob. 3/4 requires $\Omega(n/\lg n)$ space.
- *Thm:* Any deterministic offline checker requires $\Omega(n)$ space.
- Outline why $\Omega(\sqrt{n})$ space looks necessary for randomized, offline checkers...

Algorithm Intuition

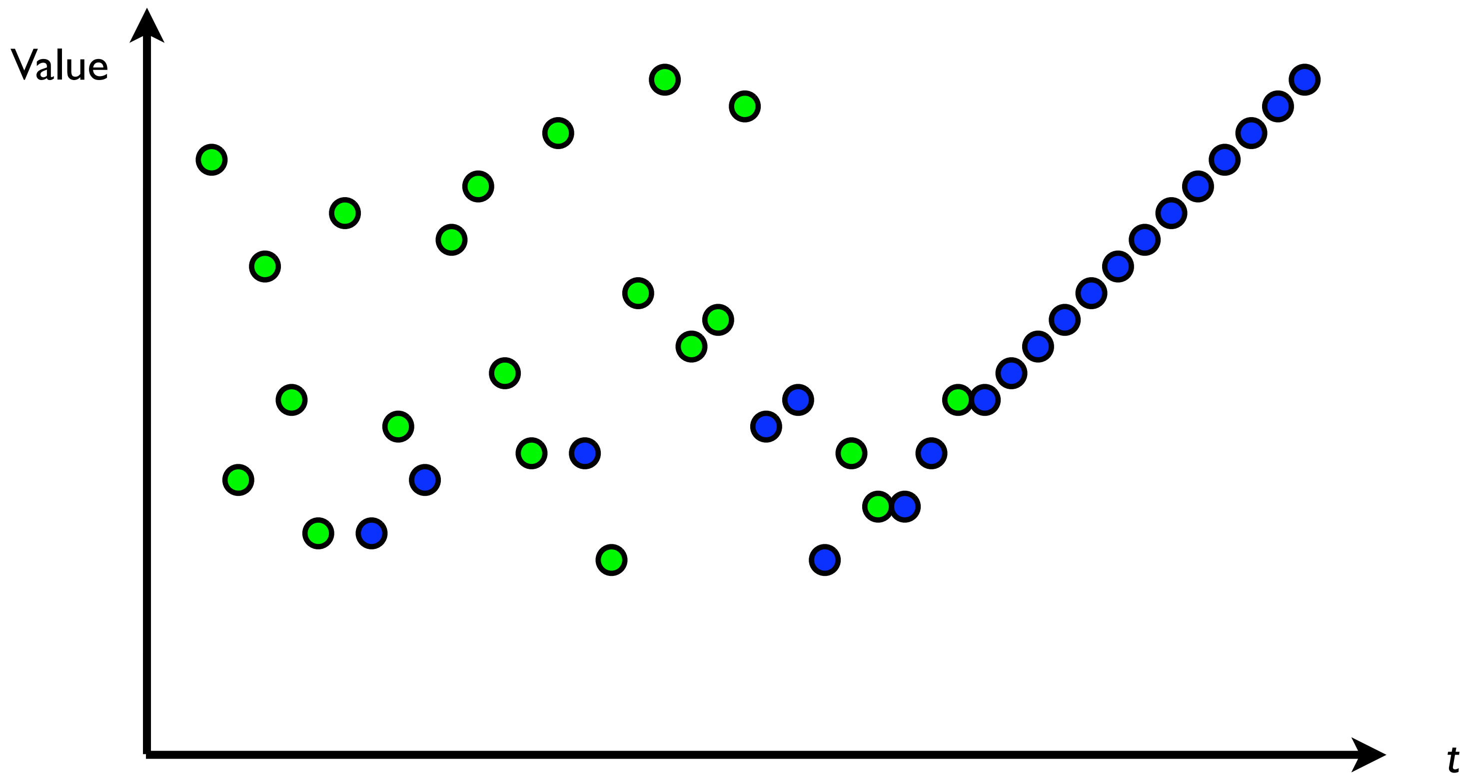
- *Key Idea:* $c_{ta}=(u,t)$ should imply that all elements inserted before *ta* and not extracted are greater than c_{ta}

Algorithm Intuition

- *Key Idea:* c_{ta}=(u,t) should imply that all elements inserted before *ta* and not extracted are greater than c_{ta}

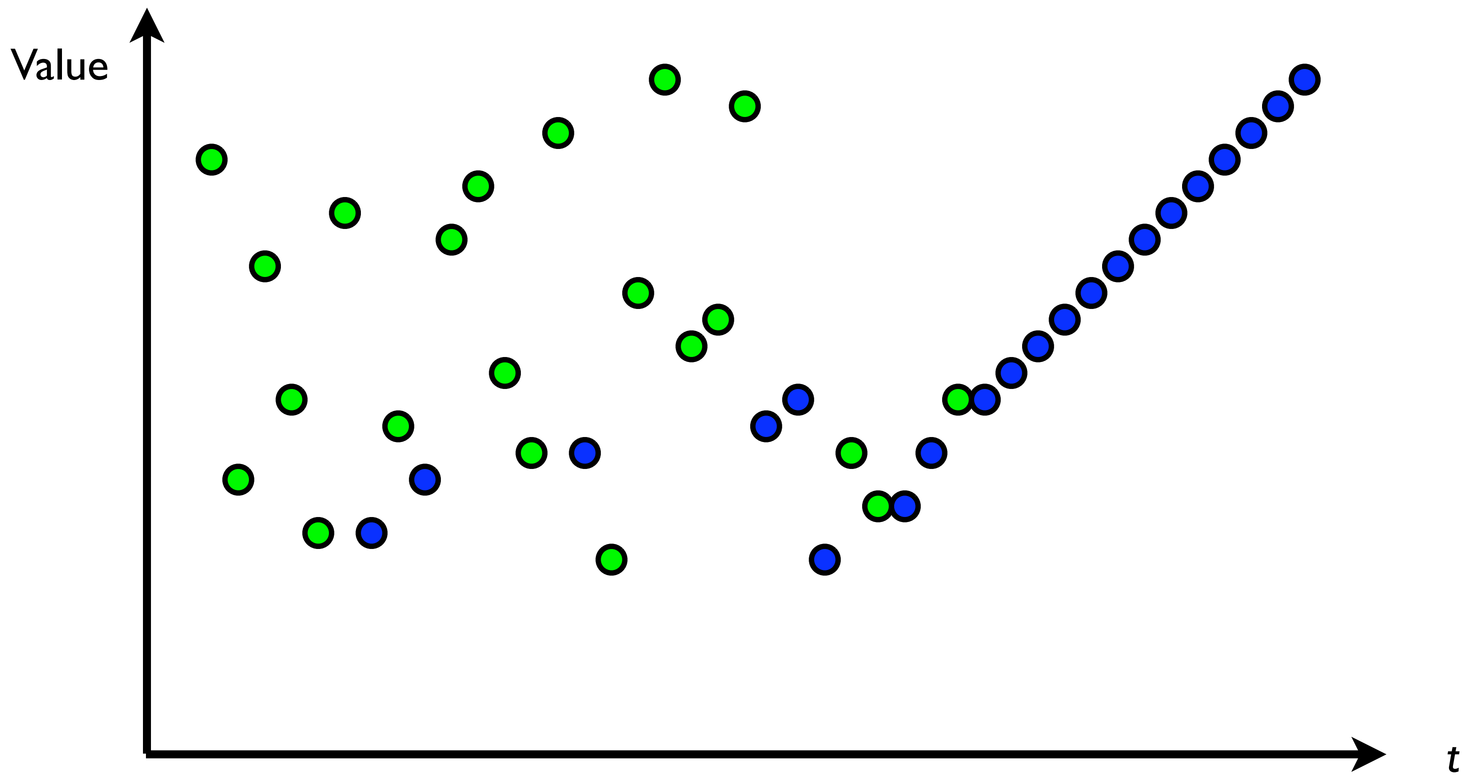
Algorithm Intuition

- *Key Idea:* c_{ta}=(u,t) should imply that all elements inserted before *ta* and not extracted are greater than c_{ta}



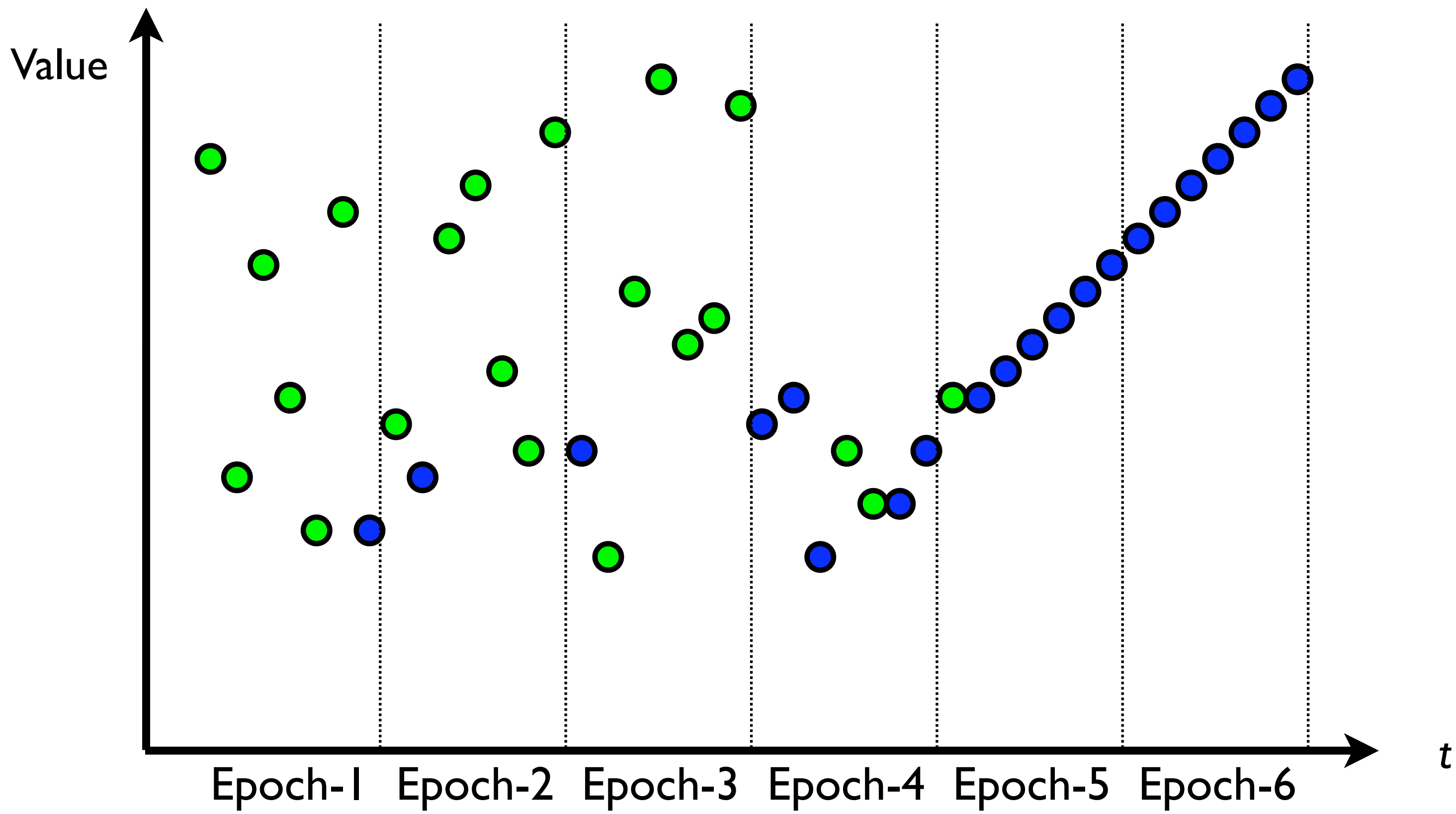
Algorithm Outline

- *Split* sequence into $\sqrt{n}$-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



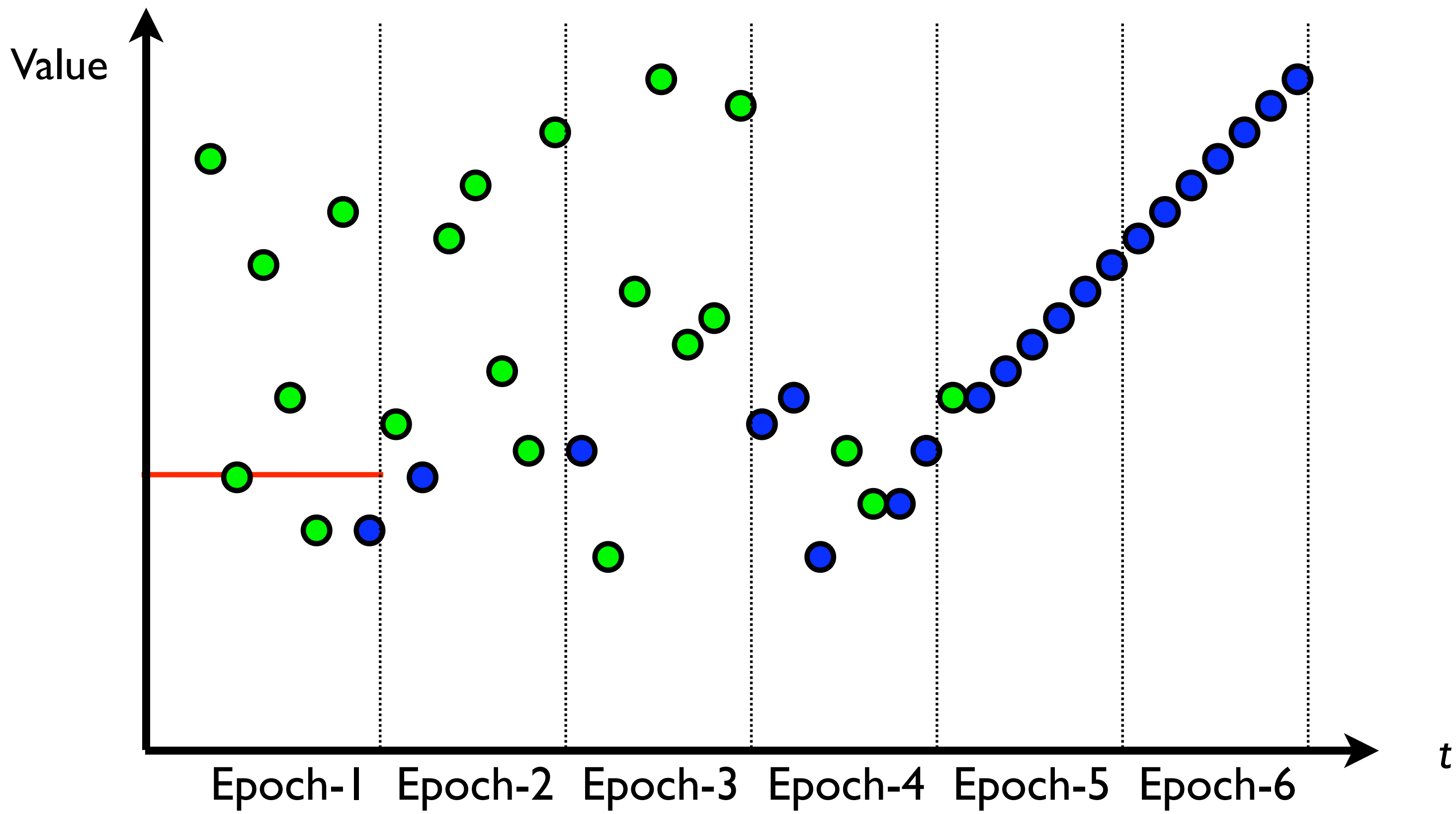
Algorithm Outline

- *Split* sequence into √n-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



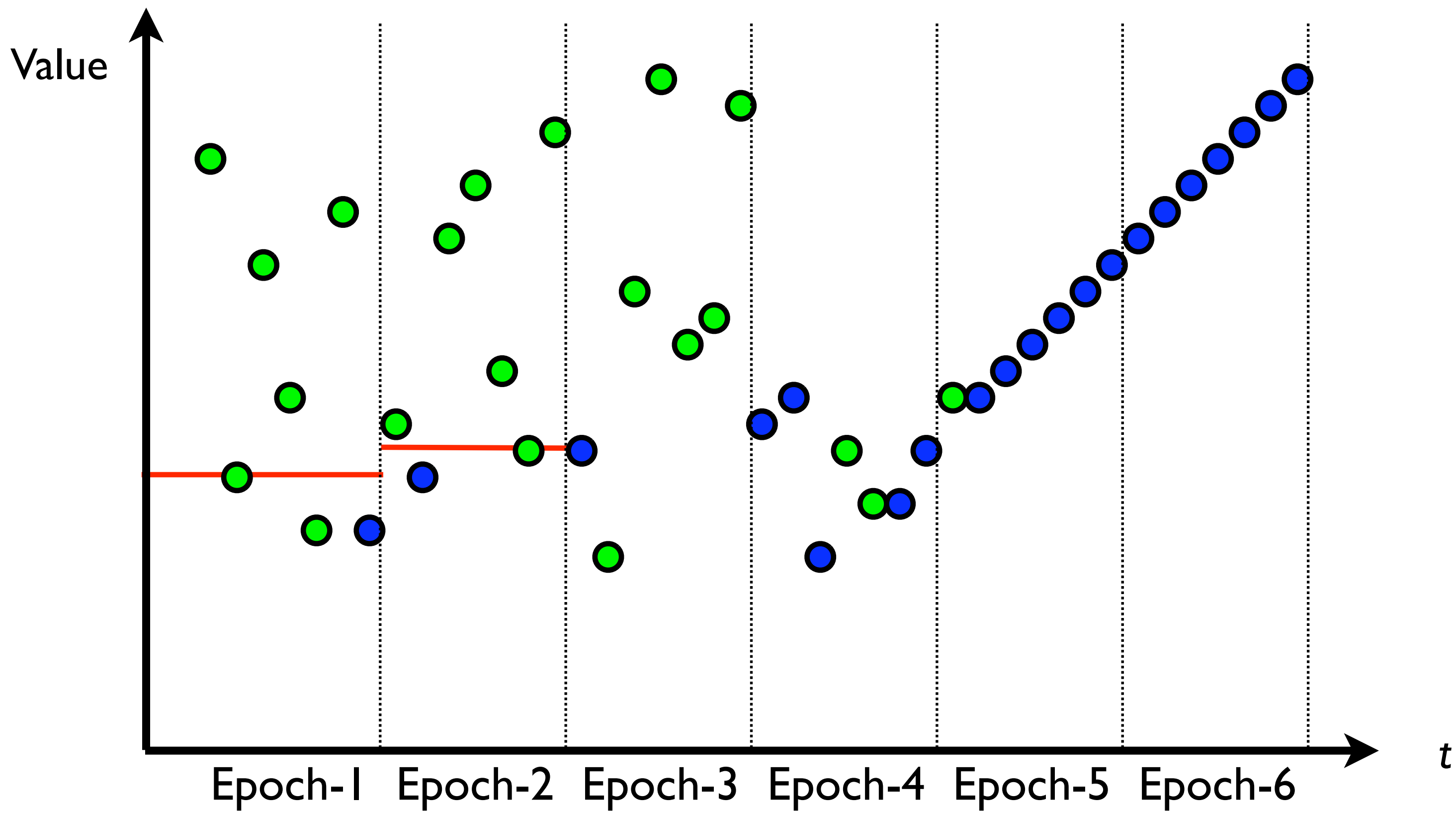
Algorithm Outline

- *Split* sequence into √n-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



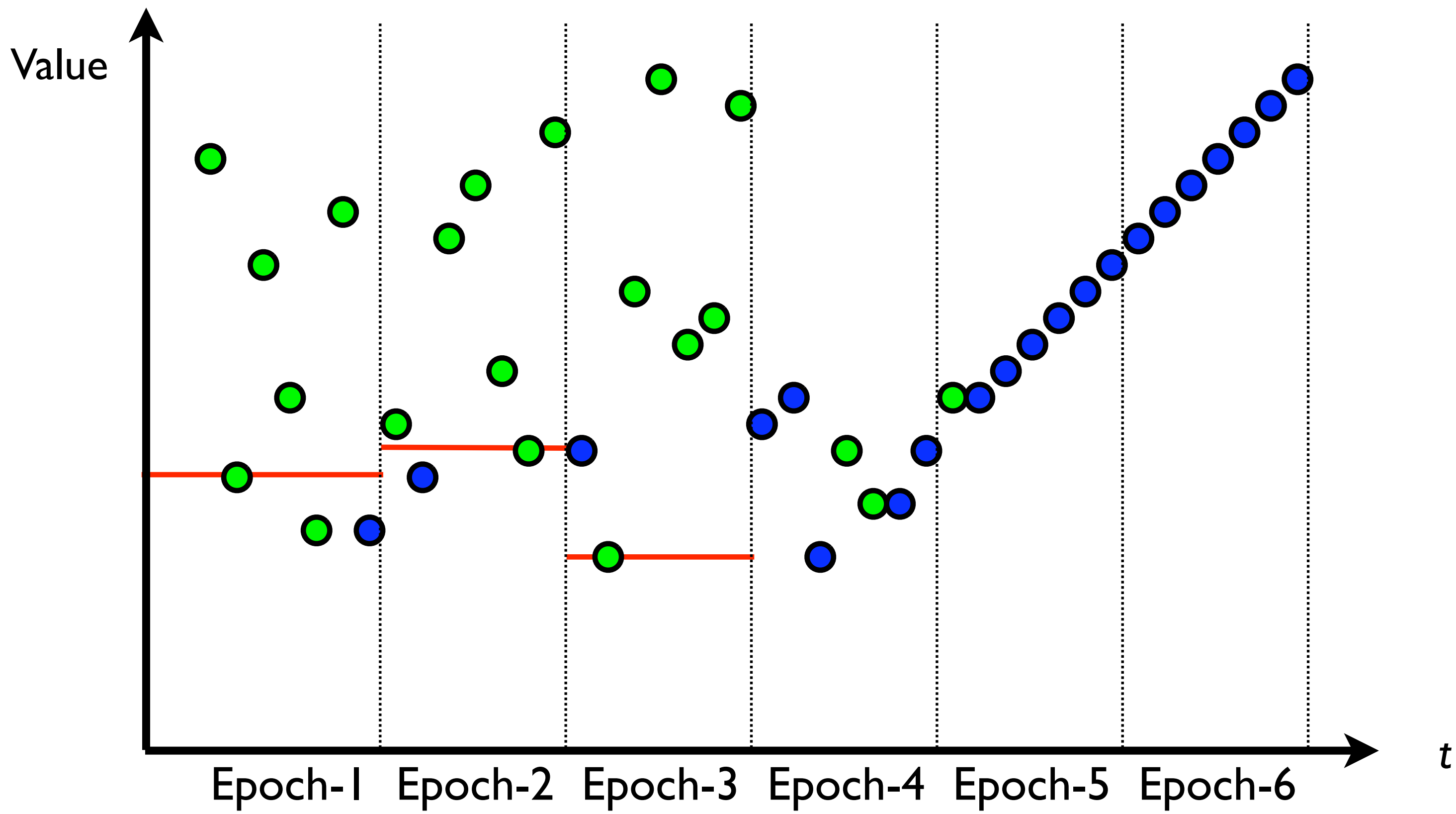
Algorithm Outline

- *Split* sequence into √n-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



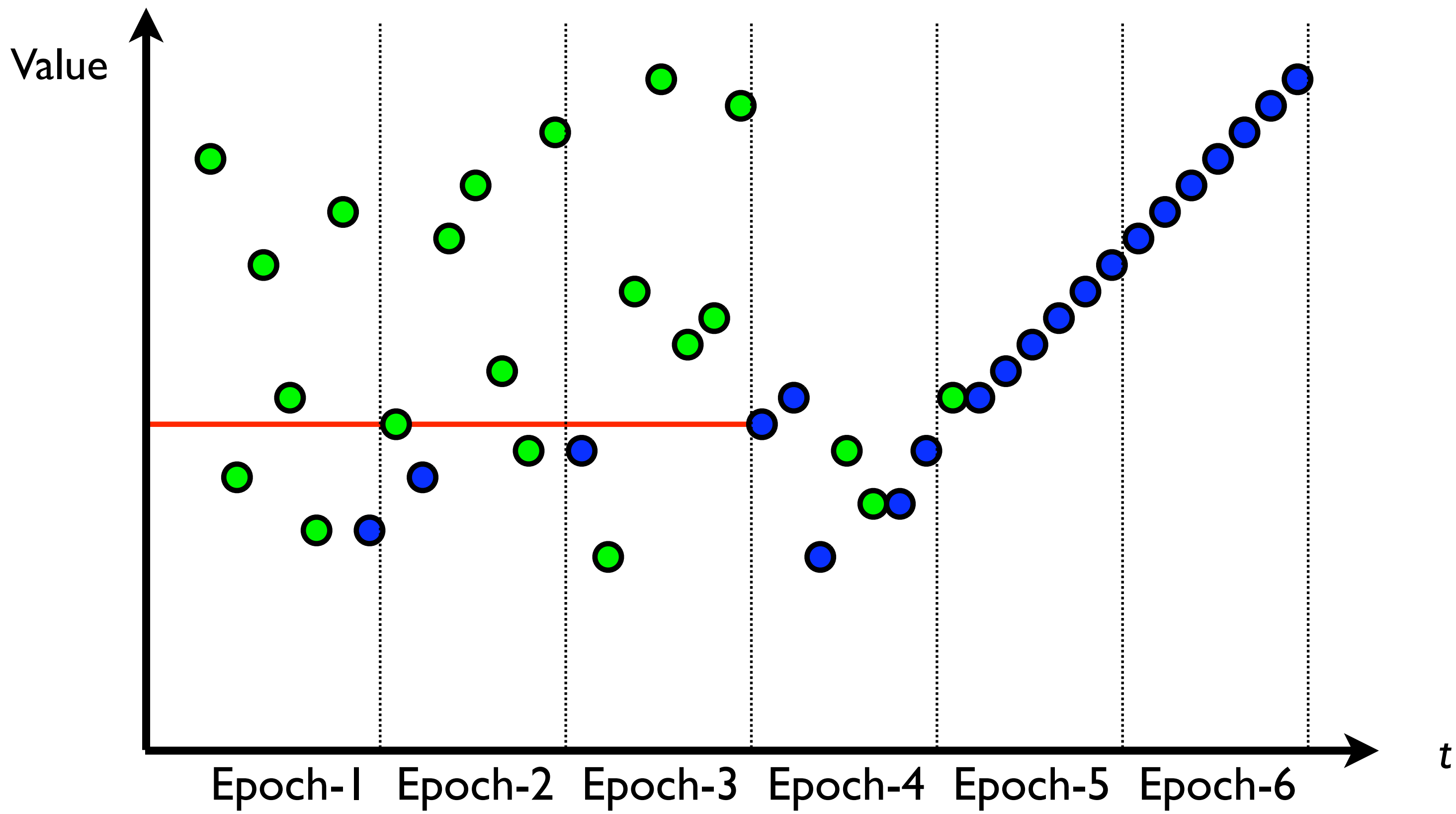
Algorithm Outline

- *Split* sequence into √n-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



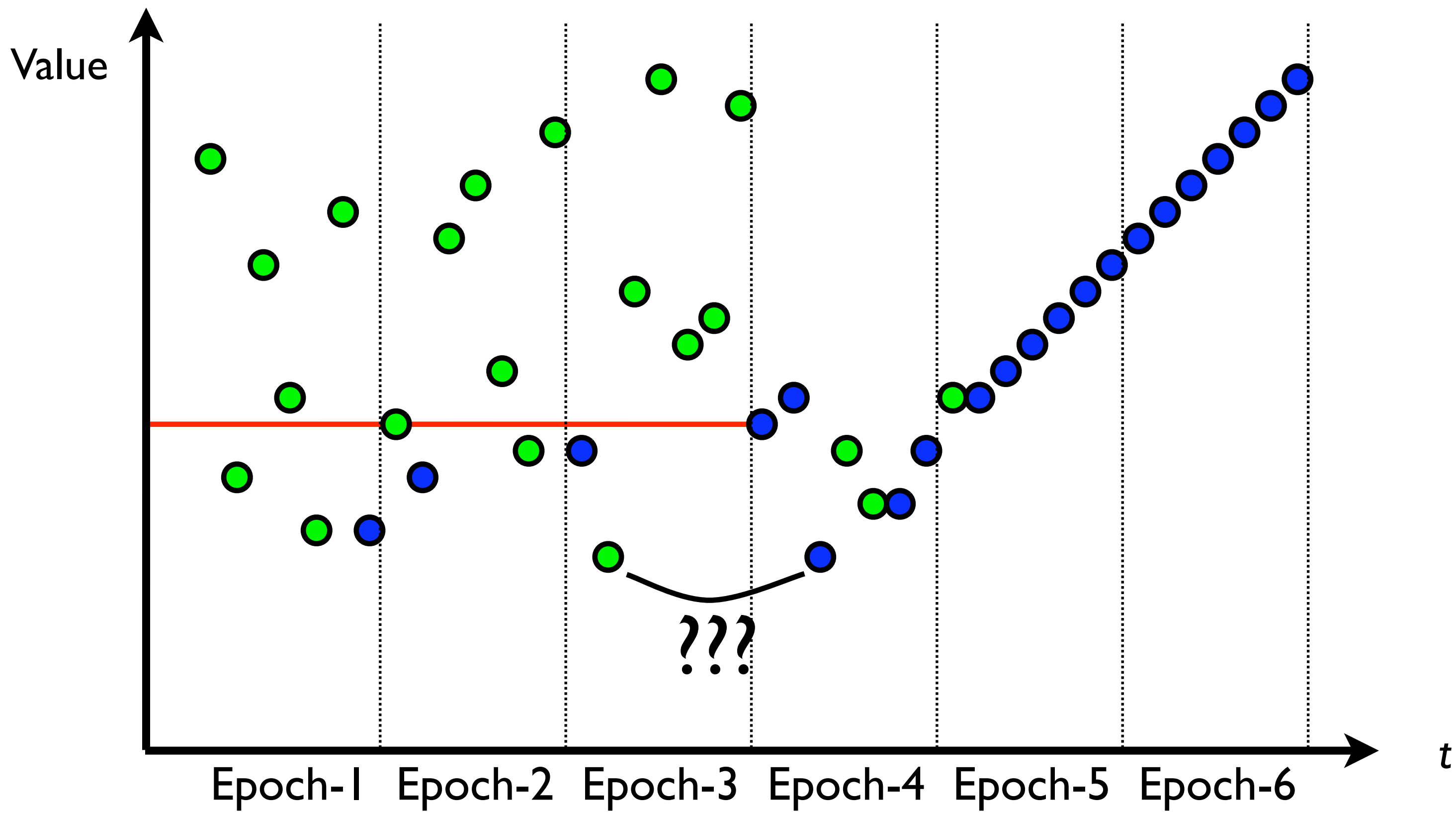
Algorithm Outline

- *Split* sequence into √n-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



Algorithm Outline

- *Split* sequence into √n-length *Epochs*
- *Identify* errors within present epoch immediately
- *Maintain* lower-bound on contents of past epochs.



Algorithm Detail

```
For k in [2√n], let f(k)=0
For i=1 to 2√n:
  Let Buffer be empty
  For j in Epoch-i={(i-1)√n+1,...,i√n}:
    If c_i=(u,t), add c_i to B
    If c_i=(u,t):
      If t in Epoch-k (k<i) and f(k)>c_i then FAIL!
      If t in Epoch-i and c_i > min Buffer then FAIL!
      Remove c_i from Buffer (if present)
      For k<i, let f(k)=max(f(k),c_i)
    Let f(i)=min Buffer
```

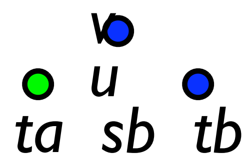
Proof of Correctness

Proof of Correctness

- We may assume *C1* and *C2* are satisfied.

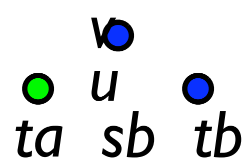
Proof of Correctness

- We may assume *C1* and *C2* are satisfied.
- Consider error: c_{tb}=(u,ta) and c_{sb}=(v,sa) such that $(u,ta)<(v,sa)$ and $ta<sb<tb$:



Proof of Correctness

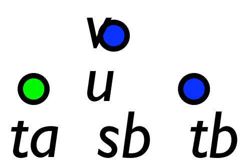
- We may assume *C1* and *C2* are satisfied.
- Consider error: c_{tb}=(u,ta) and c_{sb}=(v,sa) such that $(u,ta)<(v,sa)$ and $ta<sb<tb$:



- Let *ta* and *sb* be in Epoch-*i* and Epoch-*j* resp.

Proof of Correctness

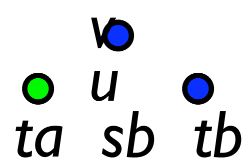
- We may assume $C1$ and $C2$ are satisfied.
- Consider error: $c_{tb}=(u,ta)$ and $c_{sb}=(v,sa)$ such that $(u,ta)<(v,sa)$ and $ta<sb<tb$:



- Let *ta* and *sb* be in Epoch-*i* and Epoch-*j* resp.
- *Case 1:* If $i=j$ then $v>$min Buffer and hence we fail at time *sb* (or before.)

Proof of Correctness

- We may assume *C1* and *C2* are satisfied.
- Consider error: $c_{tb}=(u,ta)$ and $c_{sb}=(v,sa)$ such that $(u,ta)<(v,sa)$ and $ta<sb<tb$:



- Let *ta* and *sb* be in Epoch-*i* and Epoch-*j* resp.
- *Case 1:* If $i=j$ then $v>$min Buffer and hence we fail at time *sb* (or before.)
- *Case 2:* If $i<j$ then $f(i)\geq(v,sb)$ and hence we fail at time *tb* (or before.)

Online or Deterministic?

- *Thm:* Any online checker that is correct with prob. 3/4 requires $\Omega(n/\lg n)$ space.
- *Thm:* Any offline deterministic checker requires $\Omega(n)$ space.

Alice

length n

binary string x

Bob

length n

binary string y

& index i in $[n]$

Alice
length n
binary string x

"Is the length i prefix of x and y equal?"

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor '07]

Bob
length n
binary string y
& index i in [n]

Alice

length n

binary string x

"Is the length i prefix of x and y equal?"

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor '07]

Bob

length n

binary string y

& index i in $[n]$

- Assume there exists a S-space online checker that works with prob. 3/4.

$(2+x_1,1), (4+x_2,2), \ldots ,(2n+x_n,n)$

“Is the length *i* prefix of *x* and *y* equal?”

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor ’07]

Alice
length *n*
binary string *x*

Bob
length *n*
binary string *y*
& index *i* in [*n*]

- Assume there exists a *S*-space online checker that works with prob. 3/4.

$(2+x_1,1), (4+x_2,2), \dots ,(2n+x_n,n)$ $(2+y_1,1), (4+y_2,2), \dots ,(2n+y_n,n)$

Alice

length *n*

binary string *x*

"Is the length *i* prefix of *x* and *y* equal?"

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor '07]

Bob

length *n*

binary string *y*

& index *i* in [*n*]

- Assume there exists a *S*-space online checker that works with prob. 3/4.

$(2+x_1,1), (4+x_2,2), \ldots, (2n+x_n,n)$ $(2+y_1,1), (4+y_2,2), \ldots, (2n+y_n,n)$

Alice

length n binary string x

"Is the length i prefix of x and y equal?"

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor '07]

Bob

length n binary string y & index i in $[n]$

- Assume there exists a S-space online checker that works with prob. 3/4.
- Checker fails after $(4+y_j,j)$ iff prefixes equal.

$(2+x_1,1), (4+x_2,2), \ldots ,(2n+x_n,n)$ $(2+y_1,1), (4+y_2,2), \ldots ,(2n+y_n,n)$

Alice

length n
binary string x

"Is the length i prefix of x and y equal?"

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor '07]

MEMORY STATE OF ALGORITHM

Bob

length n
binary string y
& index i in $[n]$

- Assume there exists a S-space online checker that works with prob. 3/4.
- Checker fails after $(4+y_j,j)$ iff prefixes equal.

$(2+x_1,1), (4+x_2,2), \ldots ,(2n+x_n,n)$ $(2+y_1,1), (4+y_2,2), \ldots ,(2n+y_n,n)$

Alice
length *n*
binary string *x*

"Is the length *i* prefix of *x* and *y* equal?"

Lemma: Needs $\Omega(n/\lg n)$ bits transmitted.

[Chakrabarti, Cormode, McGregor '07]

MEMORY STATE OF ALGORITHM

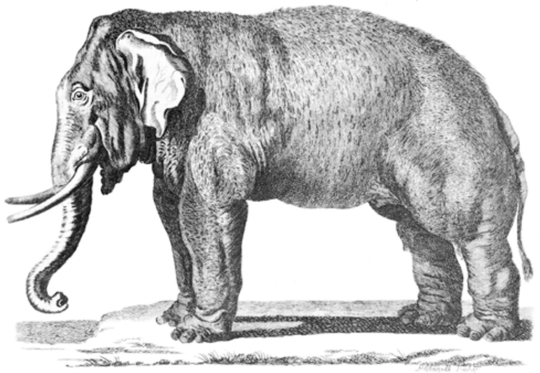
Bob
length *n*
binary string *y*
& index *i* in [*n*]

- Assume there exists a *S*-space online checker that works with prob. 3/4.
- Checker fails after $(4+y_j,j)$ iff prefixes equal.
- *Thm:* $S=\Omega(n/\lg n)$

1: Preliminaries

2: Checking

3: Spot-Checking



Spot-Checking

Spot-Checking

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.

Spot-Checking

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.

- Consider interaction sequence $c_1, \ldots, c_{2n}$ and perm. π of [2n]. Define new interaction sequence $d_1, \ldots, d_{2n}$ where

  $d_{\pi(i)} = (u, \pi(i))$ if $c_i = (\mathrm{u}, \mathrm{i})$

  $d_{\pi(i)} = (u, \pi(j))$ if $c_i = (\mathrm{u}, \mathrm{j})$

Spot-Checking

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.

- Consider interaction sequence $c_1, \ldots, c_{2n}$ and perm. π of [2n]. Define new interaction sequence $d_1, \ldots, d_{2n}$ where

  $d_{\pi(i)} = (u,\pi(i))$ if $c_i = (u,i)$

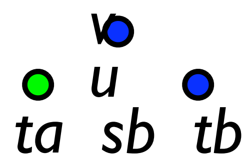
  $d_{\pi(i)} = (u,\pi(j))$ if $c_i = (u,j)$

- Say interaction sequence $c_1, \ldots, c_{2n}$ is ε-far if no permutation with less than εn rearrangements results in a correct interaction sequence.

Revealing Tuples

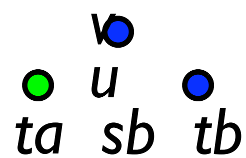
Revealing Tuples

- Say (u,ta) is a ***revealing*** if there exists $c_{sb}=(v,sa)>(u,ta)$ and $c_{tb}=(u,ta)$ such that $ta<sb<tb$:



Revealing Tuples

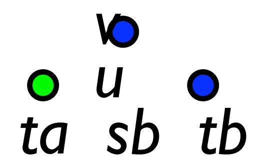
- Say (u,ta) is a ***revealing*** if there exists $c_{sb}=(v,sa)>(u,ta)$ and $c_{tb}=(u,ta)$ such that $ta<sb<tb$:



- ***Thm:*** An interaction sequence that is ε-far from being correct has at least εn revealing tuples.

Revealing Tuples

- Say (u,ta) is a *revealing* if there exists $c_{sb}=(v,sa)>(u,ta)$ and $c_{tb}=(u,ta)$ such that $ta<sb<tb$:



- *Thm:* An interaction sequence that is ε-far from being correct has at least εn revealing tuples.

- *Proof:*

  Find first incorrect extract-min, say $c_{sb}=(v,sa)$.

  Since this isn't minimum element, there exists (u,ta) and $c_{tb}=(u,ta)$ such that $ta<sb<tb$.

  Moving tb to sb reduces # of revealing tuples.

  Continue until sequence is correct.

Correctness

Correctness

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.

Correctness

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.
- *Proof:*

Correctness

- ***Thm:*** A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.
- ***Proof:***

  Samples $O(\varepsilon^{-1} \lg^2 n)$ insertions. Call these S.

Correctness

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.
- *Proof:*

  Samples $O(\varepsilon^{-1} \lg^2 n)$ insertions. Call these S.

  W.h.p. there exists a revealing tuple (u,ta) in S.

Correctness

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.
- *Proof:*

  Samples $O(\varepsilon^{-1} \lg^2 n)$ insertions. Call these S.

  W.h.p. there exists a revealing tuple (u,ta) in S.

  Monitor elements between the insertion and extraction of each element in S.

Correctness

- *Thm:* A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that fails a PQ queue that is "ε-far" from correct w.h.p.
- *Proof:*

  Samples $O(\varepsilon^{-1} \lg^2 n)$ insertions. Call these S.

  W.h.p. there exists a revealing tuple (u,ta) in S.

  Monitor elements between the insertion and extraction of each element in S.

  Will identify $c_{sb}=(v,sa)>(u,ta)$ and $c_{tb}=(u,ta)$ such that $ta<sb<tb$.

Summary

- ***Checkers:***

  A randomized, offline, $O(\sqrt{n} \log n)$-space checker that identifies errors with prob. $1-1/n$.

  Any randomized, offline checker of a "certain type" requires $\Omega(\sqrt{n})$ space.

  Online or deterministic requires $\Omega(n)$ space.

- ***Spot-Checker:***

  A randomized, offline, $O(\varepsilon^{-1} \lg^2 n)$-space spot-checker that identifies a priority queue that is "ε-far" from correct with prob. $1-1/n$.

- *... and that's how you mind you P.Q.'s!*