Thesis Defense: Adaptive Binary Search Trees

Jonathan C. Derryberry

Department of Computer Science
Carnegie Mellon University

December 16, 2009

Thesis Committee:
Daniel Sleator (chair), Guy Blelloch, Gary Miller, Seth Pettie (Michigan)

The Search Problem

- Membership-testing, dictionary, successor/predecessor, etc.
- *Sequence* of queries $\sigma_1 \cdots \sigma_m$
- Assume each $\sigma_j \in \{1, \ldots, n\}$

The Search Problem

- Membership-testing, dictionary, successor/predecessor, etc.
- *Sequence* of queries $\sigma_1 \cdots \sigma_m$
- Assume each $\sigma_j \in \{1, \ldots, n\}$

The Search Problem

- Membership-testing, dictionary, successor/predecessor, etc.
- *Sequence* of queries $\sigma_1 \cdots \sigma_m$
- Assume each $\sigma_j \in \{1, \ldots, n\}$

Outline

Outline

Which Computational Model?

- Hashing (RAM): $O(1)$
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
- Comparison model: $O(\lg n)$
- BST model: $O(\lg n)$

Which Computational Model?

- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
- Comparison model: $O(\lg n)$
- BST model: $O(\lg n)$

Which Computational Model?

- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
- Comparison model: $O(\lg n)$
- BST model: $O(\lg n)$

Which Computational Model?

- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
  (requires integers, no augmenting)
- Comparison model: $O(\lg n)$
- BST model: $O(\lg n)$

Which Computational Model?

- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
  (requires integers, no augmenting)
- Comparison model: $O(\lg n)$
- BST model: $O(\lg n)$

Which Computational Model?

- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
  (requires integers, no augmenting)
- Comparison model: $O(\lg n)$
  (no augmenting)
- BST model: $O(\lg n)$

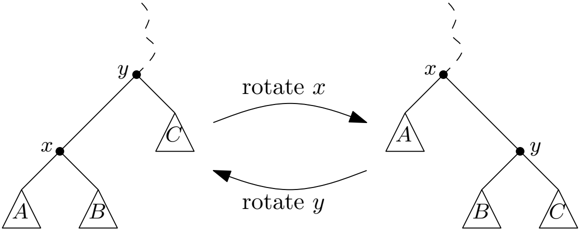
Which Computational Model?

- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
  (requires integers, no augmenting)
- Comparison model: $O(\lg n)$
  (no augmenting)
- BST model: $O(\lg n)$

Which Computational Model?

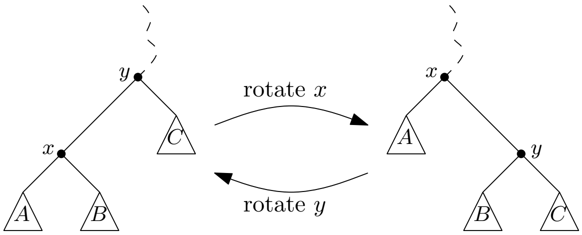
- Hashing (RAM): $O(1)$
  (but no successor)
- vEB/Fusion Trees (RAM): $O(\sqrt{\lg n})$
  (requires integers, no augmenting)
- Comparison model: $O(\lg n)$
  (no augmenting)
- BST model: $O(\lg n)$
  (supports augmenting)

BST Model [Wil89]



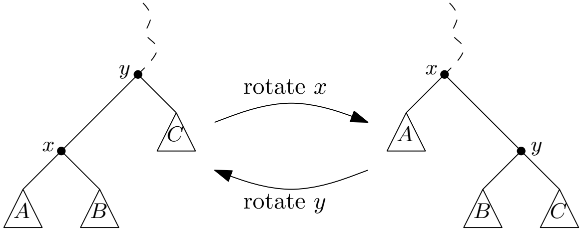
- Binary tree in symmetric order
- Modifiable with BST rotations
- Must rotate queried node to root

BST Model [Wil89]



- Binary tree in symmetric order
- Modifiable with BST rotations
- Must rotate queried node to root

BST Model [Wil89]



- Binary tree in symmetric order
- Modifiable with BST rotations
- Must rotate queried node to root

Flexibility of the BST Model

- Can easily do $O(\lg n)$ worst-case
- Can adapt to sequences with patterns
- To adapt, rotate nodes likely to be accessed to near the root

Flexibility of the BST Model

- Can easily do $O(\lg n)$ worst-case
- Can adapt to sequences with patterns
- To adapt, rotate nodes likely to be accessed to near the root

Flexibility of the BST Model

- Can easily do $O(\lg n)$ worst-case
- Can adapt to sequences with patterns
- To adapt, rotate nodes likely to be accessed to near the root

Outline

Why Prove a Lower Bound?

- Help prove optimality
- Invalidate the computational model
- Understand the model/problem

Why Prove a Lower Bound?

- Help prove optimality
- Invalidate the computational model
- Understand the model/problem

Why Prove a Lower Bound?

- Help prove optimality
- Invalidate the computational model
- Understand the model/problem

- Online BST costs $\Omega(\lg n)$?
- Offline BST costs $\Omega(\lg n)$?
- Instance-specific bounds?

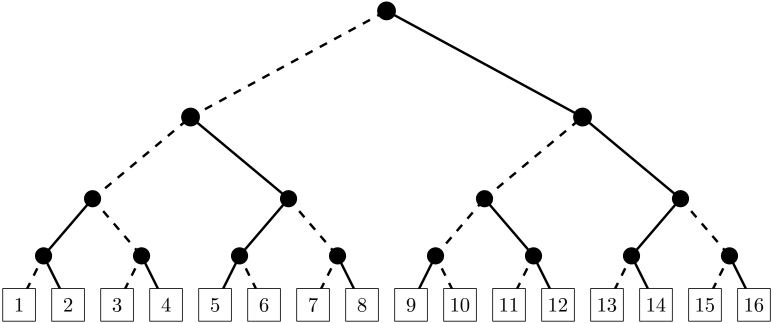
Lower Bound Basics

- Online BST costs $\Omega(\lg n)$?
- Offline BST costs $\Omega(\lg n)$?
- Instance-specific bounds?

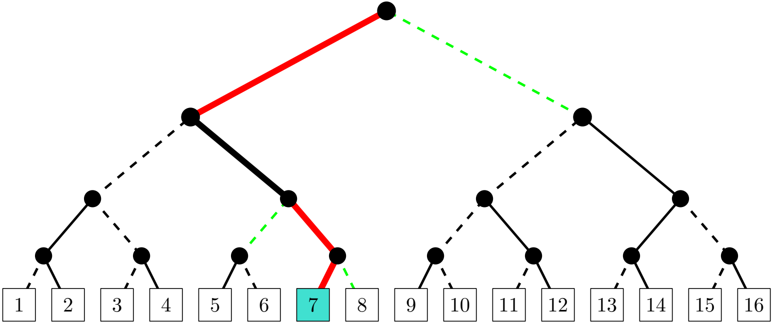
Lower Bound Basics

- Online BST costs $\Omega(\lg n)$?
- Offline BST costs $\Omega(\lg n)$?
- Instance-specific bounds?

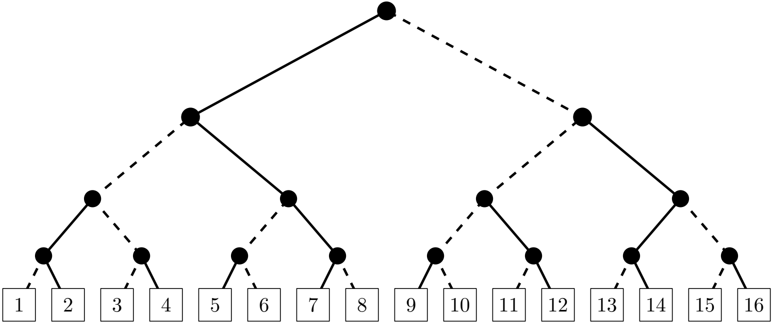
Wilber's First Bound [Wil89]



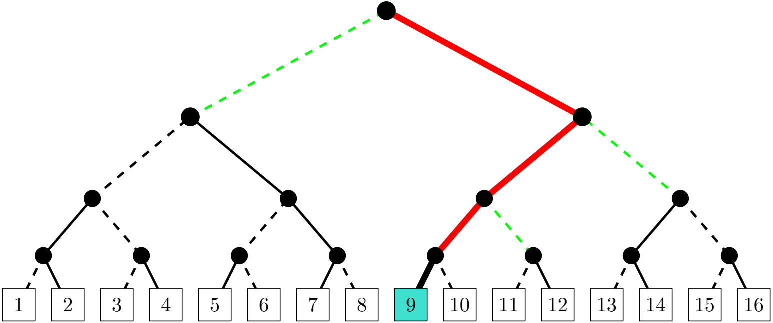
Wilber's First Bound [Wil89]



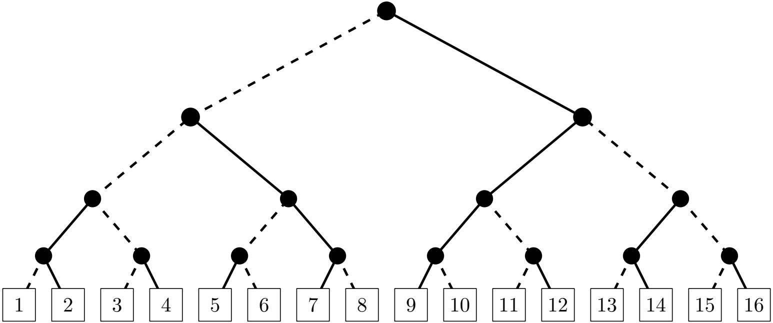
Wilber's First Bound [Wil89]



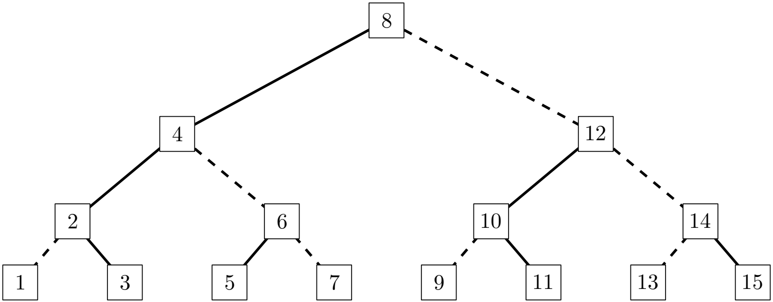
Wilber's First Bound [Wil89]



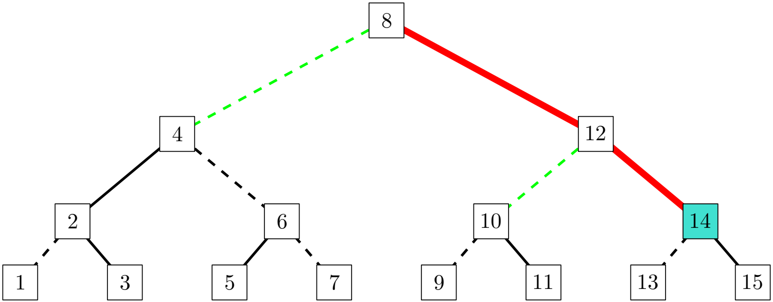
Wilber's First Bound [Wil89]



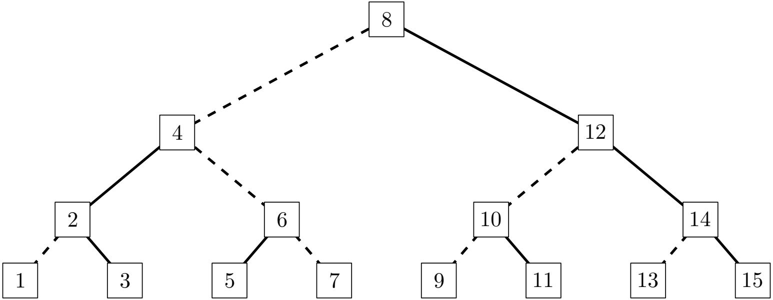
Interleave Bound [DHIP04]



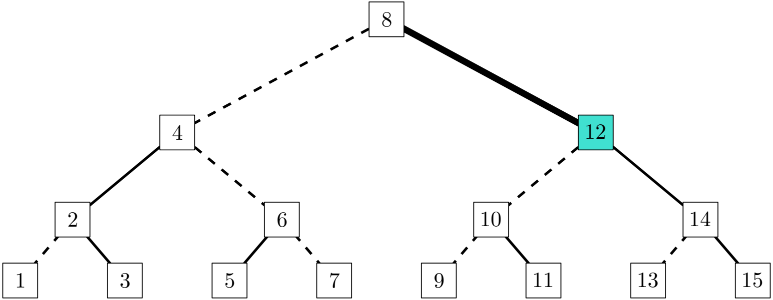
Interleave Bound [DHIP04]

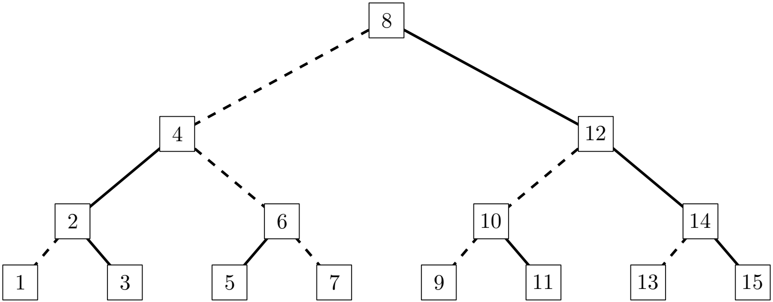


Interleave Bound [DHIP04]

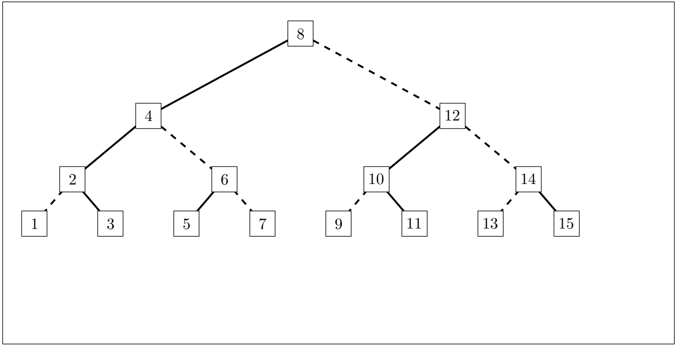


Interleave Bound [DHIP04]




8
4
12
2
6
10
14
1
3
5
7
9
11
13
15

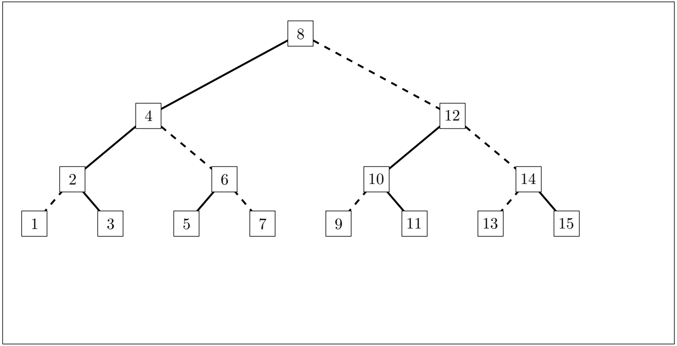
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

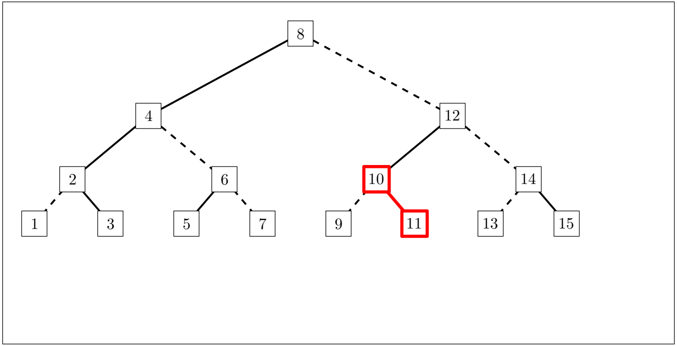
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

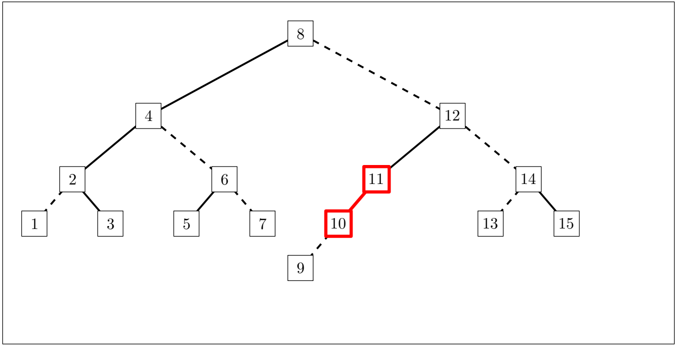
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

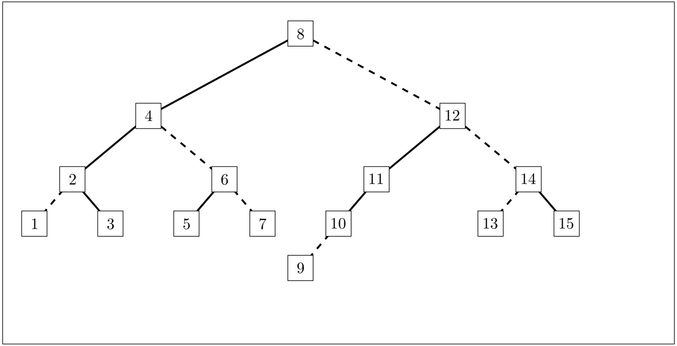
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

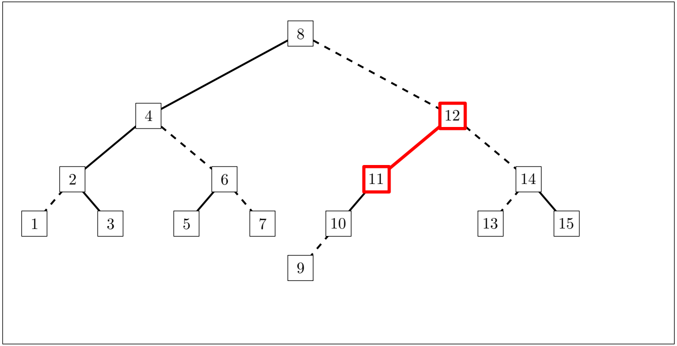
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

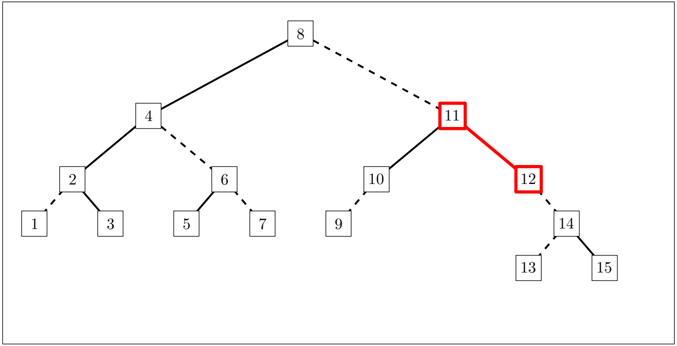
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

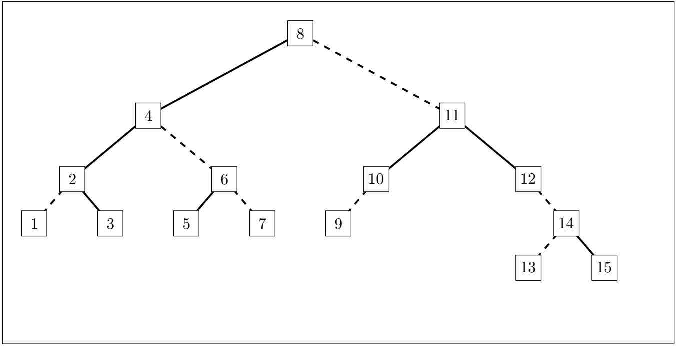
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

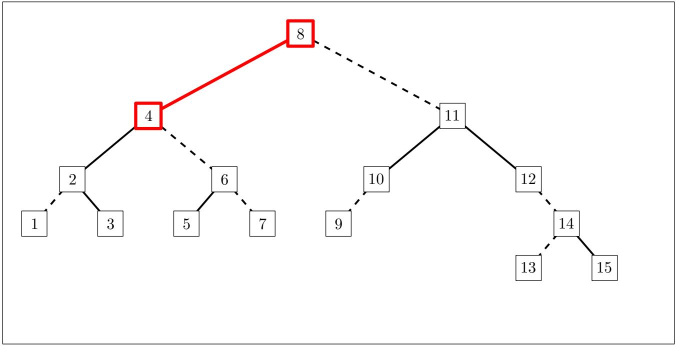
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

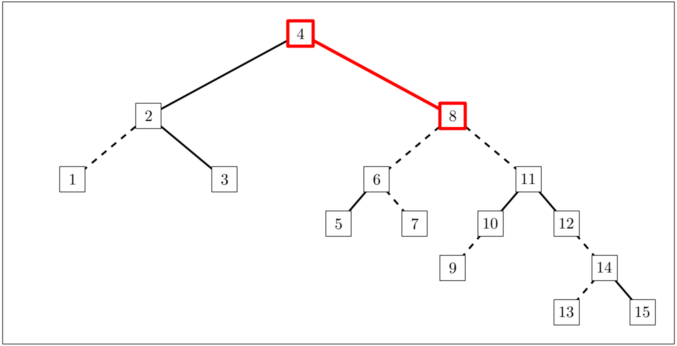
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

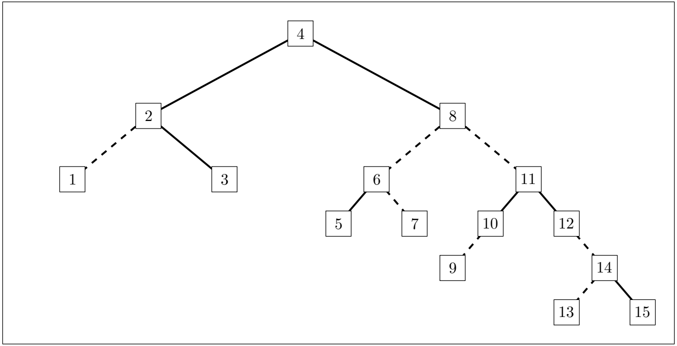
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

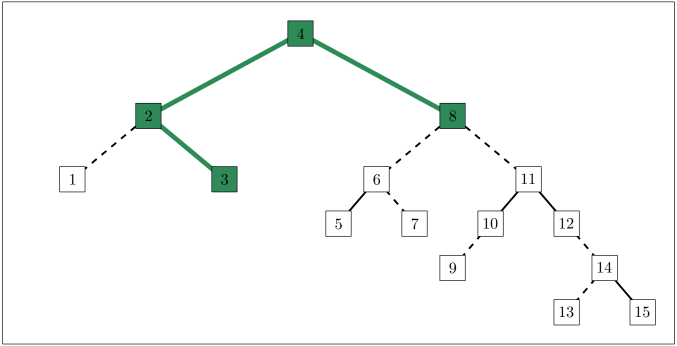
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

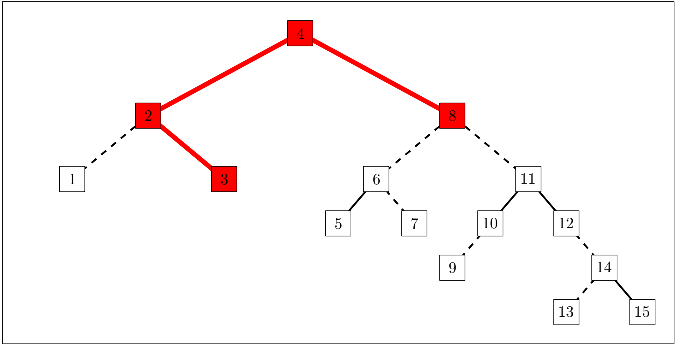
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

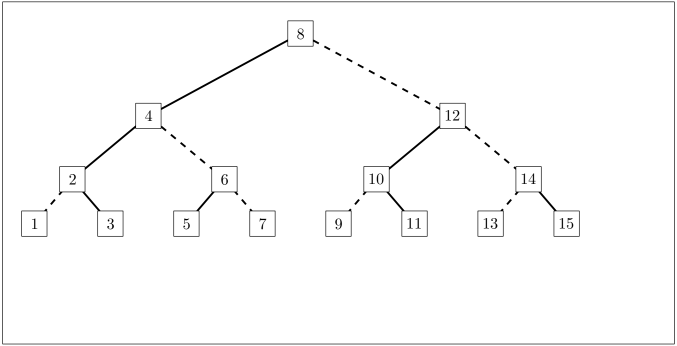
Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

Uses of the Interleave Bound



Implications

- Balanced BSTs dynamically optimal for random sequences
- Rotate paths (only in BST) into a red-black tree [DHIP04]
- Swap red-black trees for splay trees and gain [WDS06]
- Can also rotate *lower bound tree* to allow updates [WDS06]

How good is Wilber-1/Interleave?

- Lets us compete ($O(\lg \lg n)$-competitiveness)
- Every static lower bound tree is loose by $\Omega(\lg \lg n)$
- Can the dynamic interleave bound help?
- Can Wilber's second bound help?

How good is Wilber-1/Interleave?

- Lets us compete ($O(\lg\lg n)$-competitiveness)
- Every static lower bound tree is loose by $\Omega(\lg\lg n)$
- Can the dynamic interleave bound help?
- Can Wilber's second bound help?

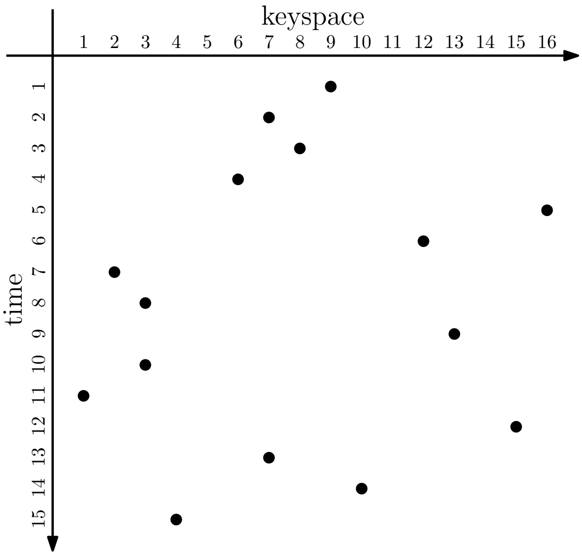
How good is Wilber-1/Interleave?

- Lets us compete ($O(\lg\lg n)$-competitiveness)
- Every static lower bound tree is loose by $\Omega(\lg\lg n)$
- Can the dynamic interleave bound help?
- Can Wilber's second bound help?

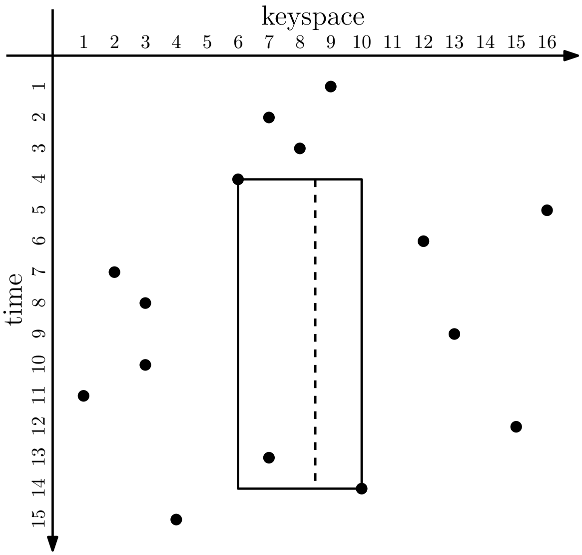
How good is Wilber-1/Interleave?

- Lets us compete ($O(\lg\lg n)$-competitiveness)
- Every static lower bound tree is loose by $\Omega(\lg\lg n)$
- Can the dynamic interleave bound help?
- Can Wilber's second bound help?

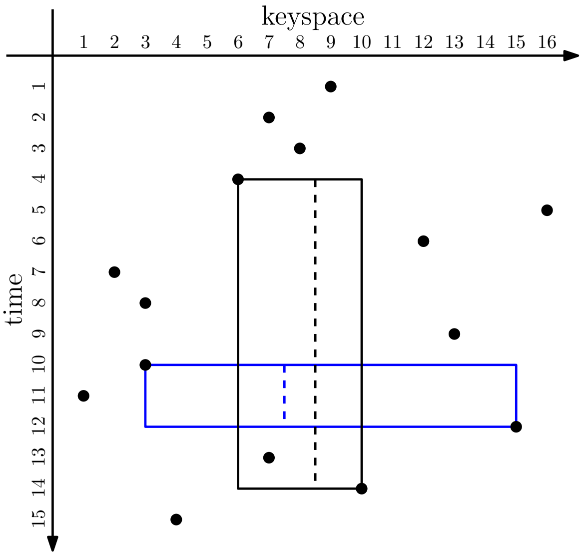
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



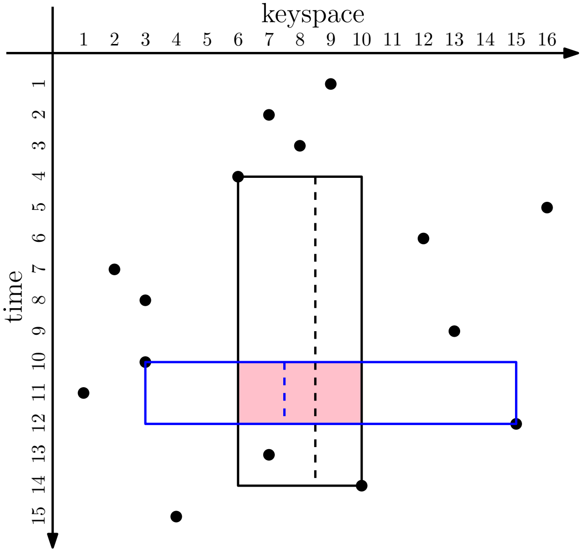
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



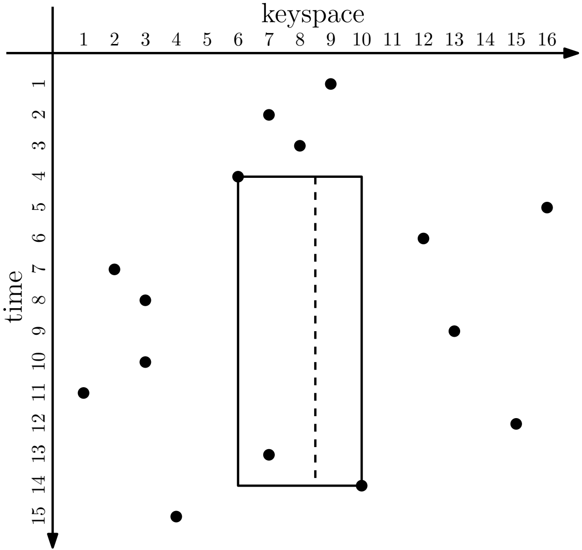
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



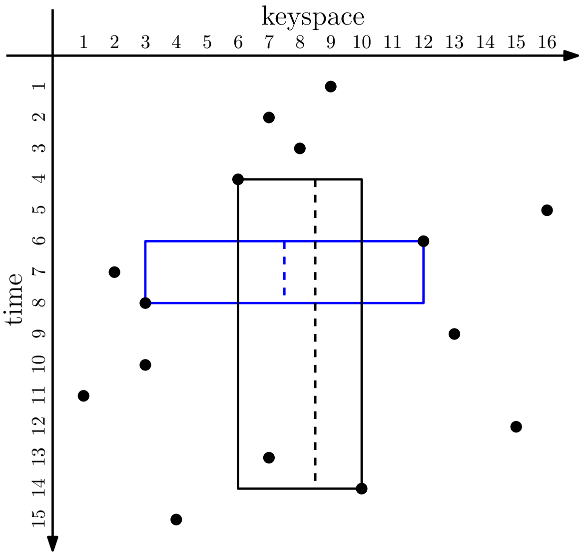
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



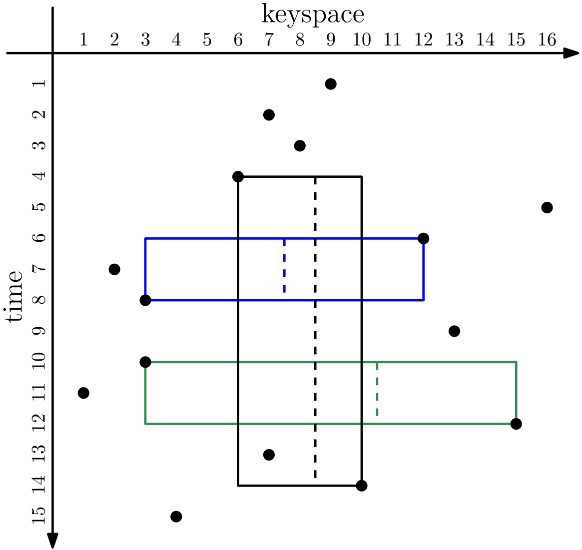
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



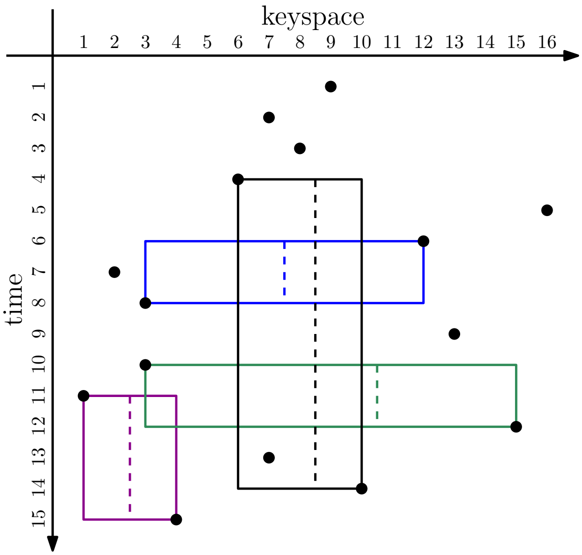
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



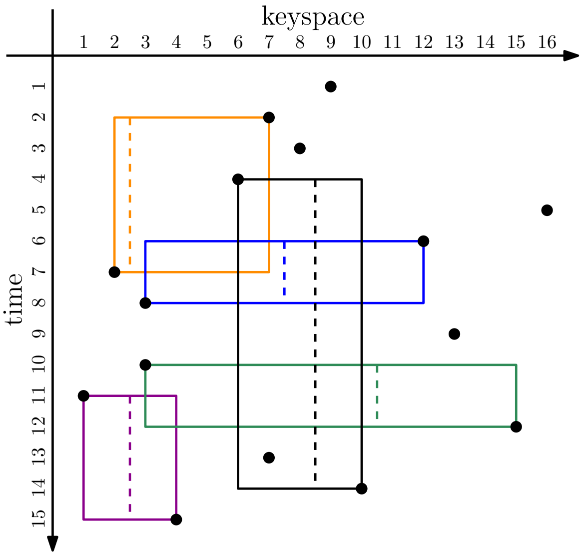
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



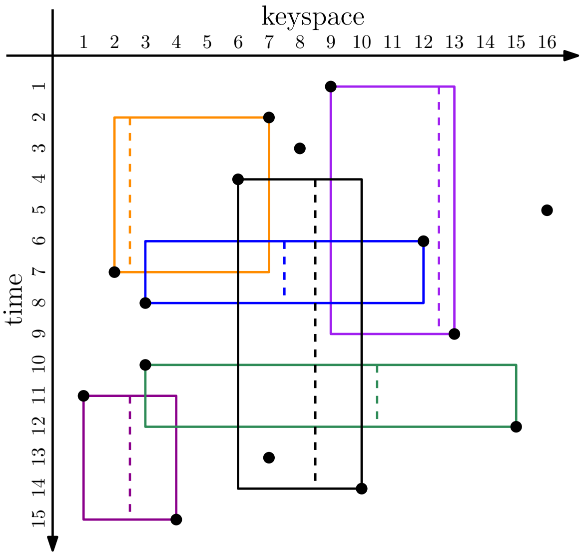
MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]



MIBS: Generalizing Wilber-1 and Wilber-2 [DSW05]

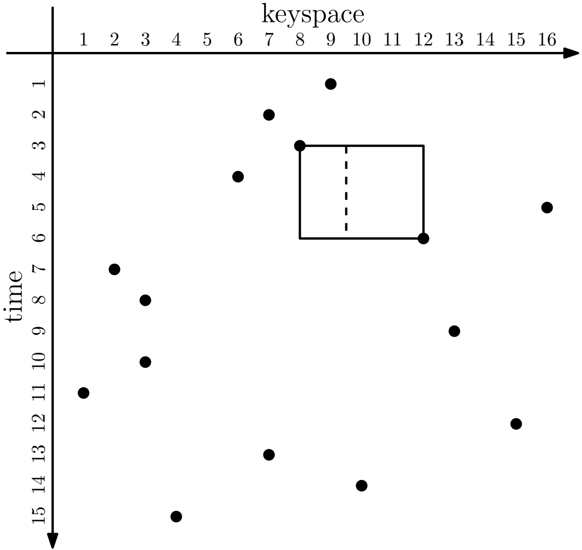


The MIBS Bound

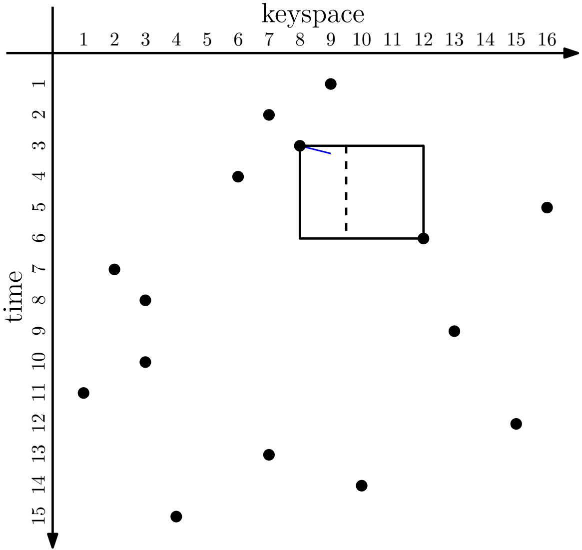
Theorem

The number of boxes is a lower bound on $\mathrm{OPT}(\sigma)$.

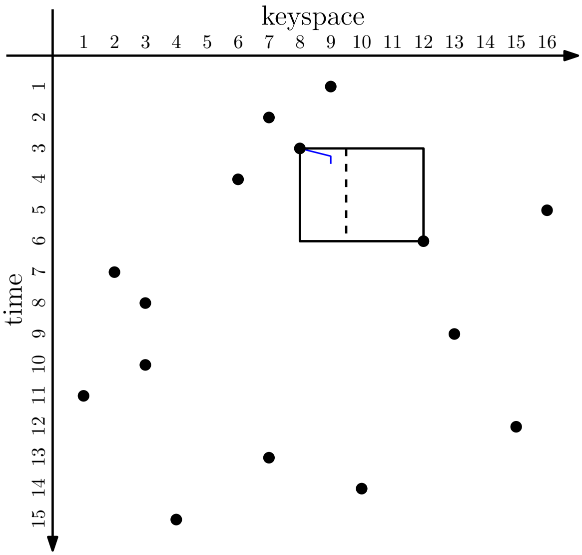
Proof of MIBS Bound



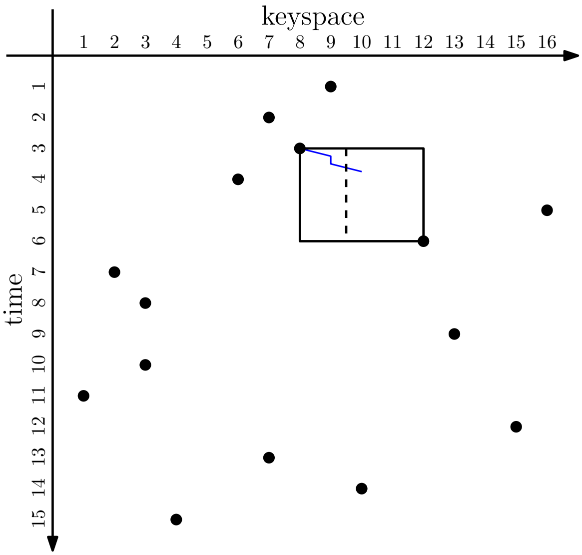
Proof of MIBS Bound



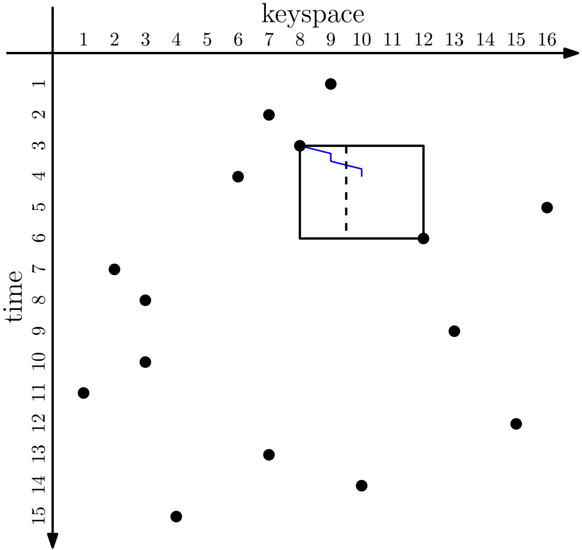
Proof of MIBS Bound



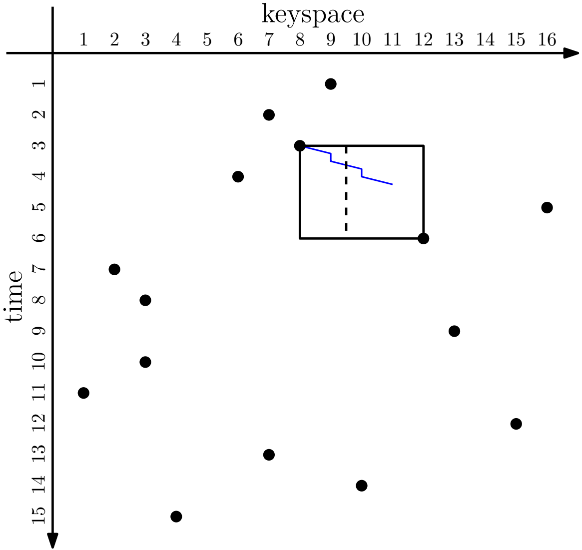
Proof of MIBS Bound



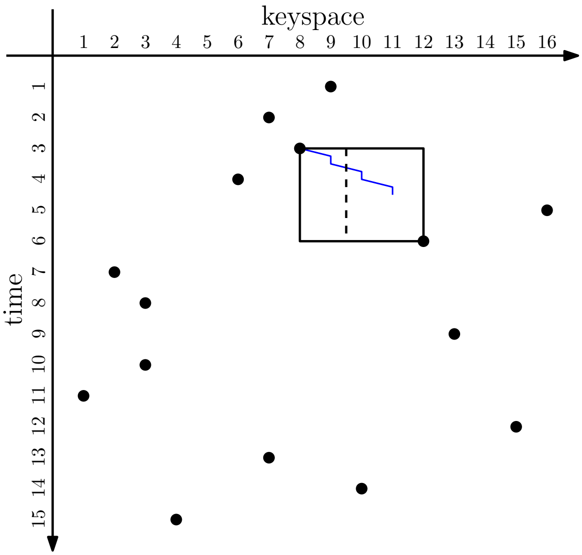
Proof of MIBS Bound



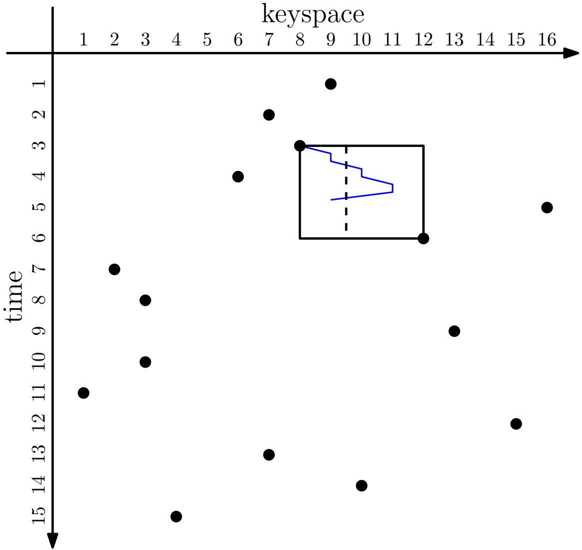
Proof of MIBS Bound



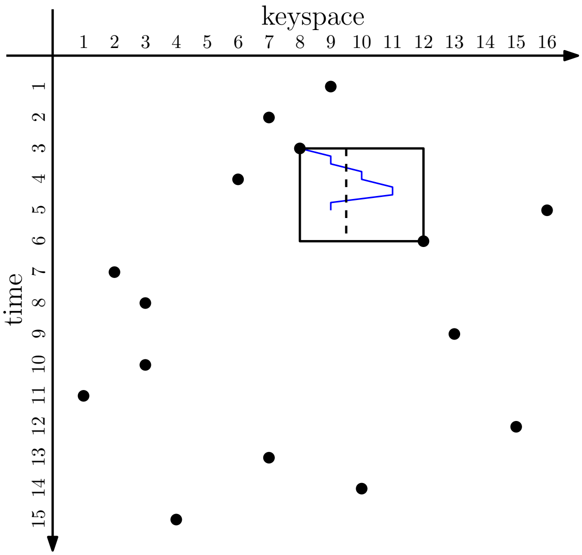
Proof of MIBS Bound



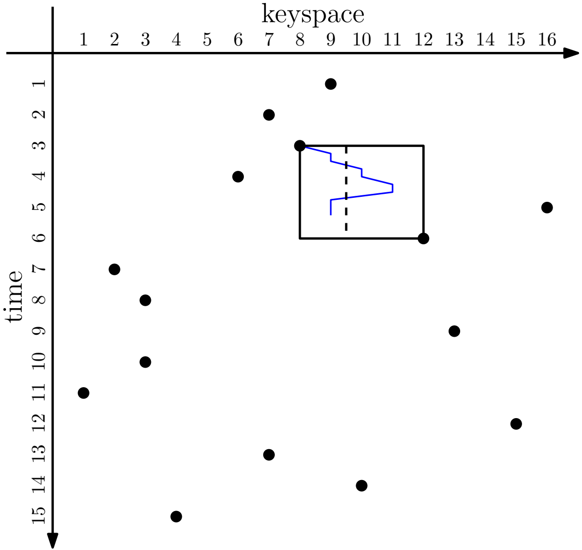
Proof of MIBS Bound



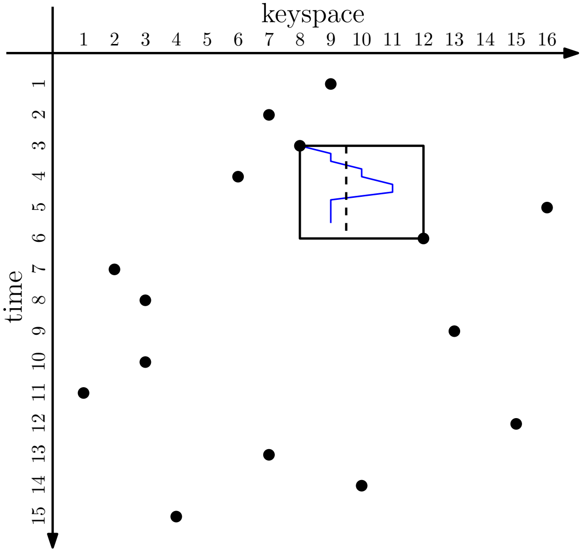
Proof of MIBS Bound



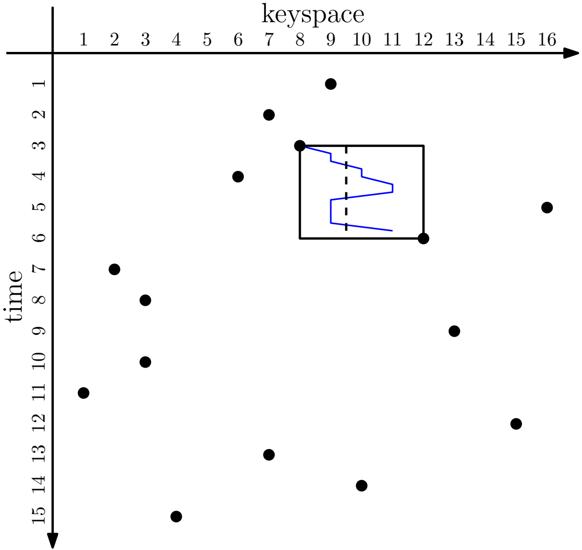
Proof of MIBS Bound



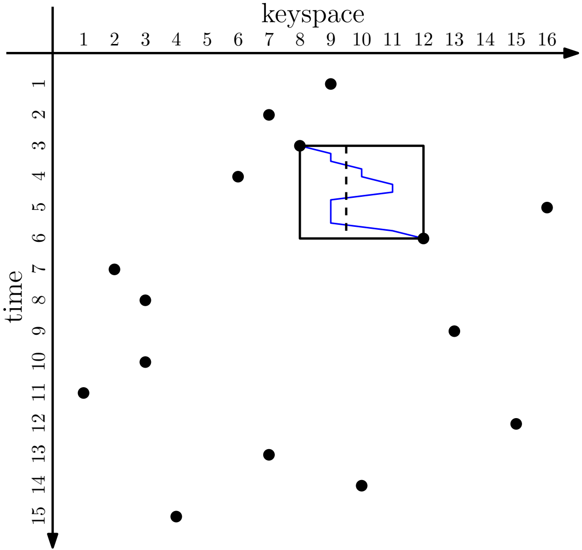
Proof of MIBS Bound



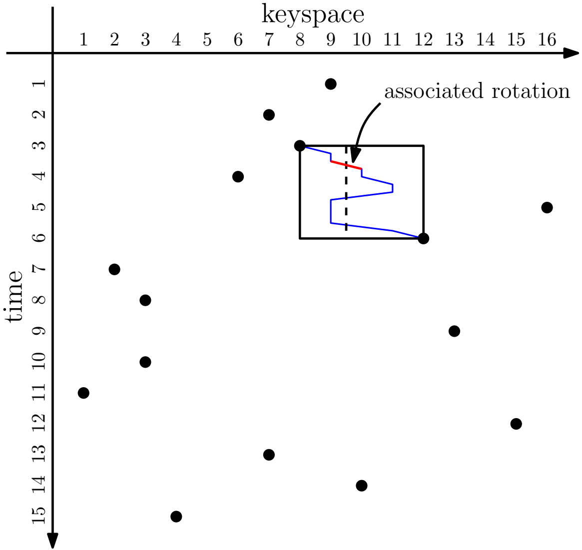
Proof of MIBS Bound



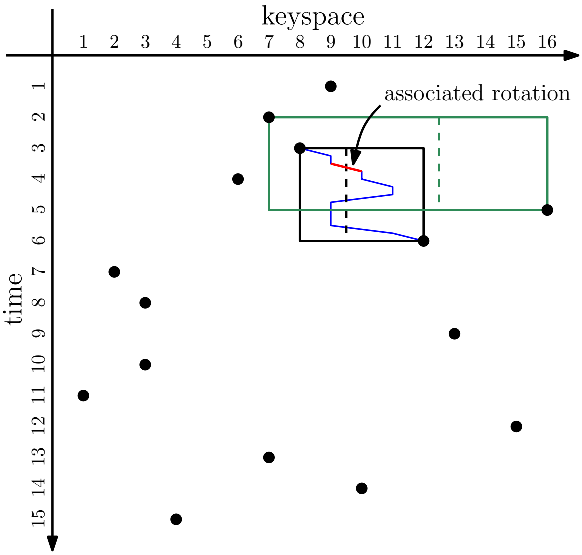
Proof of MIBS Bound



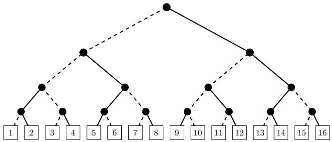
Proof of MIBS Bound

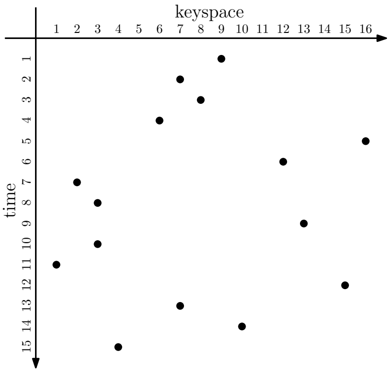


Proof of MIBS Bound

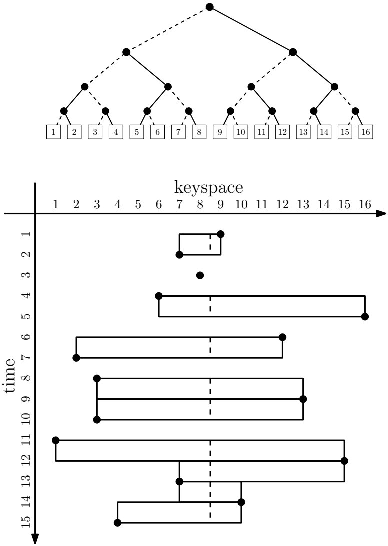


MIBS ≥ Wilber's First Bound

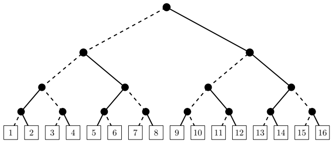


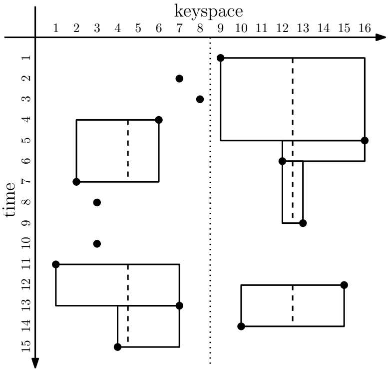


MIBS ≥ Wilber's First Bound

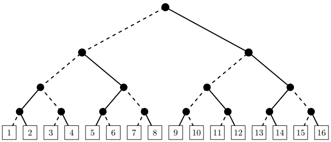


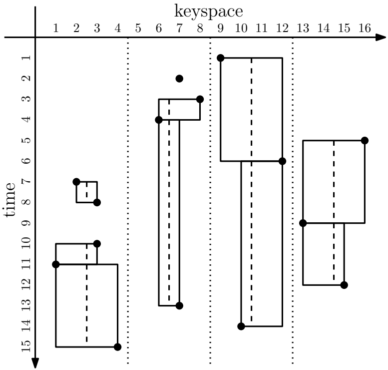
MIBS ≥ Wilber's First Bound



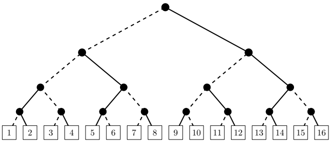


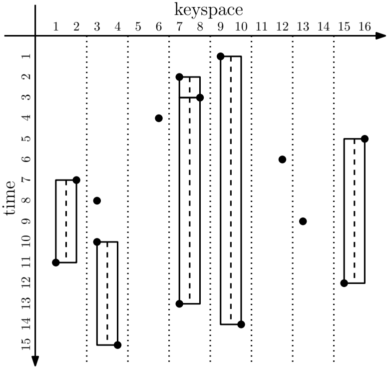
MIBS ≥ Wilber's First Bound





MIBS ≥ Wilber's First Bound





Musing on MIBS

- Generalizes previous bounds
- Does not help improve competitiveness so far
- Strengthens connection between partial-sums *problem* and BST *model*

Musing on MIBS

- Generalizes previous bounds
- Does not help improve competitiveness so far
- Strengthens connection between partial-sums *problem* and BST *model*

Musing on MIBS

- Generalizes previous bounds
- Does not help improve competitiveness so far
- Strengthens connection between partial-sums *problem* and BST *model*

Partial-Sums Problem

- Maintain an array under update operations
- Return sum$(1, \ldots, i)$ when requested for current array values
- Input: sequence of update and sum operations
- Output: sequence of sums
- Lower bound: $\Omega(\lg n)$ in cell-probe model [PD04]
- Lower bound: MIBS, if we require explicitly computed sums

Partial-Sums Problem

- Maintain an array under update operations
- Return sum$(1, \ldots, i)$ when requested for current array values
- Input: sequence of update and sum operations
- Output: sequence of sums
- Lower bound: $\Omega(\lg n)$ in cell-probe model [PD04]
- Lower bound: MIBS, if we require explicitly computed sums

Partial-Sums Problem

- Maintain an array under update operations
- Return sum$(1, \ldots, i)$ when requested for current array values
- Input: sequence of update and sum operations
- Output: sequence of sums
- Lower bound: $\Omega(\lg n)$ in cell-probe model [PD04]
- Lower bound: MIBS, if we require explicitly computed sums

Partial-Sums Problem

- Maintain an array under update operations
- Return $\text{sum}(1, \ldots, i)$ when requested for current array values
- Input: sequence of update and sum operations
- Output: sequence of sums
- Lower bound: $\Omega(\lg n)$ in cell-probe model [PD04]
- Lower bound: MIBS, if we require explicitly computed sums

Partial-Sums Problem

- Maintain an array under update operations
- Return sum$(1, \ldots, i)$ when requested for current array values
- Input: sequence of update and sum operations
- Output: sequence of sums
- Lower bound: $\Omega(\lg n)$ in cell-probe model [PD04]
- Lower bound: MIBS, if we require explicitly computed sums

Partial-Sums Problem

- Maintain an array under update operations
- Return sum$(1, \ldots, i)$ when requested for current array values
- Input: sequence of update and sum operations
- Output: sequence of sums
- Lower bound: $\Omega(\lg n)$ in cell-probe model [PD04]
- Lower bound: MIBS, if we require explicitly computed sums

History of BST Lower Bounds

- [Wil89]: first offline, instance-specific bounds
- [BCK02]: offline information-theoretic bound
- [DHIP04]: BST-like bound
- [WDS06]: rotatable BST-like bound
- [DSW05],[DHI$^+$09]: generalization of Wilber-1 and Wilber-2

History of BST Lower Bounds

- [Wil89]: first offline, instance-specific bounds
- [BCK02]: offline information-theoretic bound
- [DHIP04]: BST-like bound
- [WDS06]: rotatable BST-like bound
- [DSW05],[DHI$^+$09]: generalization of Wilber-1 and Wilber-2

History of BST Lower Bounds

- [Wil89]: first offline, instance-specific bounds
- [BCK02]: offline information-theoretic bound
- [DHIP04]: BST-like bound
- [WDS06]: rotatable BST-like bound
- [DSW05],[DHI$^+$09]: generalization of Wilber-1 and Wilber-2

History of BST Lower Bounds

- [Wil89]: first offline, instance-specific bounds
- [BCK02]: offline information-theoretic bound
- [DHIP04]: BST-like bound
- [WDS06]: rotatable BST-like bound
- [DSW05],[DHI+09]: generalization of Wilber-1 and Wilber-2

History of BST Lower Bounds

- [Wil89]: first offline, instance-specific bounds
- [BCK02]: offline information-theoretic bound
- [DHIP04]: BST-like bound
- [WDS06]: rotatable BST-like bound
- [DSW05],[DHI^{+}09]: generalization of Wilber-1 and Wilber-2

History of BST Competitiveness

- [BCK02]: dynamic search optimality
- [DHIP04]: $O(\lg \lg n)$-competitive
- [WDS06, DSW09]: support insert/delete, deque, working set
- [Geo08]: like multi-splay, only $O(\lg \lg n)$-competitive
- [BDDF09]: support worst-case $O(\lg n)$ running time

History of BST Competitiveness

- [BCK02]: dynamic search optimality
- [DHIP04]: $O(\lg \lg n)$-competitive
- [WDS06, DSW09]: support insert/delete, deque, working set
- [Geo08]: like multi-splay, only $O(\lg \lg n)$-competitive
- [BDDF09]: support worst-case $O(\lg n)$ running time

History of BST Competitiveness

- [BCK02]: dynamic search optimality
- [DHIP04]: $O(\lg \lg n)$-competitive
- [WDS06, DSW09]: support insert/delete, deque, working set
- [Geo08]: like multi-splay, only $O(\lg \lg n)$-competitive
- [BDDF09]: support worst-case $O(\lg n)$ running time

History of BST Competitiveness

- [BCK02]: dynamic search optimality
- [DHIP04]: $O(\lg \lg n)$-competitive
- [WDS06, DSW09]: support insert/delete, deque, working set
- [Geo08]: like multi-splay, only $O(\lg \lg n)$-competitive
- [BDDF09]: support worst-case $O(\lg n)$ running time

History of BST Competitiveness

- [BCK02]: dynamic search optimality
- [DHIP04]: $O(\lg \lg n)$-competitive
- [WDS06, DSW09]: support insert/delete, deque, working set
- [Geo08]: like multi-splay, only $O(\lg \lg n)$-competitive
- [BDDF09]: support worst-case $O(\lg n)$ running time

Outline

Formulaic Adaptivity

- Strength of competitiveness depends on the model
- Alternative: input-sensitive bounds with intuitive meaning

Formulaic Adaptivity

- Strength of competitiveness depends on the model
- Alternative: input-sensitive bounds with intuitive meaning

Bounds with Intuitive Meaning

Working Set Bound (exploiting temporal locality)

Access x, then t distinct keys. Pay $O(\lg t)$ to access x again.

Dynamic Finger Bound (exploiting spatial locality)

Access x. Then, access y at cost $O(\lg |x - y|)$.

Bounds with Intuitive Meaning

Working Set Bound (exploiting temporal locality)

Access x, then t distinct keys. Pay $O(\lg t)$ to access x again.

Dynamic Finger Bound (exploiting spatial locality)

Access x. Then, access y at cost $O(\lg |x - y|)$.

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg|x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2} + 1, 2, \frac{n}{2} + 2, 3, \frac{n}{2} + 3, \ldots$

Need three fingers

$1, \frac{n}{3} + 1, \frac{2n}{3} + 1, 2, \frac{n}{3} + 2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x (\lg t_x + \lg |x - y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg|x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x - y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \dots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \dots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg|x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg |x-y|)$$

What If There Is a Mix of Spatial and Temporal Locality?

Need two fingers

$1, \frac{n}{2}+1, 2, \frac{n}{2}+2, 3, \frac{n}{2}+3, \ldots$

Need three fingers

$1, \frac{n}{3}+1, \frac{2n}{3}+1, 2, \frac{n}{3}+2, \ldots$

Support many fingers: Unified Bound [Iac01]

Cost of a query to y:

$$\min_x(\lg t_x + \lg|x-y|)$$

Relationship of Unified Bound to Dynamic Optimality

- Generalizes working set and dynamic finger bounds
- Iacono achived the bound in the pointer model
- Not sufficient for optimality:

$$1, n^{1/2}+1, 2n^{1/2}+1, 3n^{1/2}+1, \ldots, (n^{1/2}-1)n^{1/2}+1,$$
$$2, n^{1/2}+2, 2n^{1/2}+2, 3n^{1/2}+2, \ldots, (n^{1/2}-1)n^{1/2}+2,$$
$$3, n^{1/2}+3, 2n^{1/2}+3, 3n^{1/2}+3, \ldots, (n^{1/2}-1)n^{1/2}+3,$$
$$\vdots$$
$$n^{1/2}, n^{1/2}+n^{1/2}, 2n^{1/2}+n^{1/2}, 3n^{1/2}+n^{1/2}, \ldots, n.$$

- Necessary for optimality? Open until this work.

Relationship of Unified Bound to Dynamic Optimality

- Generalizes working set and dynamic finger bounds
- Iacono achived the bound in the pointer model
- Not sufficient for optimality:

$$1, n^{1/2}+1, 2n^{1/2}+1, 3n^{1/2}+1, \ldots, (n^{1/2}-1)n^{1/2}+1,$$
$$2, n^{1/2}+2, 2n^{1/2}+2, 3n^{1/2}+2, \ldots, (n^{1/2}-1)n^{1/2}+2,$$
$$3, n^{1/2}+3, 2n^{1/2}+3, 3n^{1/2}+3, \ldots, (n^{1/2}-1)n^{1/2}+3,$$
$$\vdots$$
$$n^{1/2}, n^{1/2}+n^{1/2}, 2n^{1/2}+n^{1/2}, 3n^{1/2}+n^{1/2}, \ldots, n.$$

- Necessary for optimality? Open until this work.

Relationship of Unified Bound to Dynamic Optimality

- Generalizes working set and dynamic finger bounds
- Iacono achived the bound in the pointer model
- Not sufficient for optimality:

$$1, n^{1/2}+1, 2n^{1/2}+1, 3n^{1/2}+1, \ldots, (n^{1/2}-1)n^{1/2}+1,$$
$$2, n^{1/2}+2, 2n^{1/2}+2, 3n^{1/2}+2, \ldots, (n^{1/2}-1)n^{1/2}+2,$$
$$3, n^{1/2}+3, 2n^{1/2}+3, 3n^{1/2}+3, \ldots, (n^{1/2}-1)n^{1/2}+3,$$
$$\vdots$$
$$n^{1/2}, n^{1/2}+n^{1/2}, 2n^{1/2}+n^{1/2}, 3n^{1/2}+n^{1/2}, \ldots, n.$$

- Necessary for optimality? Open until this work.

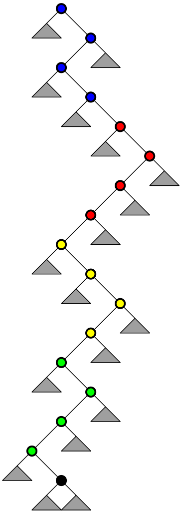
Relationship of Unified Bound to Dynamic Optimality

- Generalizes working set and dynamic finger bounds
- Iacono achived the bound in the pointer model
- Not sufficient for optimality:

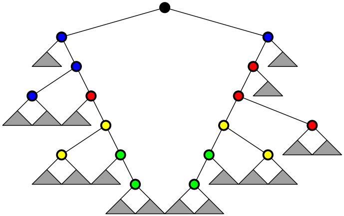
$$\begin{gathered} 1, n^{1/2}+1, 2n^{1/2}+1, 3n^{1/2}+1, \ldots, (n^{1/2}-1)n^{1/2}+1, \\ 2, n^{1/2}+2, 2n^{1/2}+2, 3n^{1/2}+2, \ldots, (n^{1/2}-1)n^{1/2}+2, \\ 3, n^{1/2}+3, 2n^{1/2}+3, 3n^{1/2}+3, \ldots, (n^{1/2}-1)n^{1/2}+3, \\ \vdots \\ n^{1/2}, n^{1/2}+n^{1/2}, 2n^{1/2}+n^{1/2}, 3n^{1/2}+n^{1/2}, \ldots, n. \end{gathered}$$

- Necessary for optimality? Open until this work.

Why Should Splay Trees Satisfy the Unified Bound?



Why Should Splay Trees Satisfy the Unified Bound?



Why Is This Hard to Prove?

- Unified Bound subsumes dynamic finger bound
- Proof of dynamic finger bound is 80 pages! [CMSS00, Col00]
- Also, Unified Bound subsumes the deque bound
- Not proven for splay trees (best is $\alpha^*(n)$ [Pet08])

Why Is This Hard to Prove?

- Unified Bound subsumes dynamic finger bound
- Proof of dynamic finger bound is 80 pages! [CMSS00, Col00]
- Also, Unified Bound subsumes the deque bound
- Not proven for splay trees (best is $\alpha^*(n)$ [Pet08])

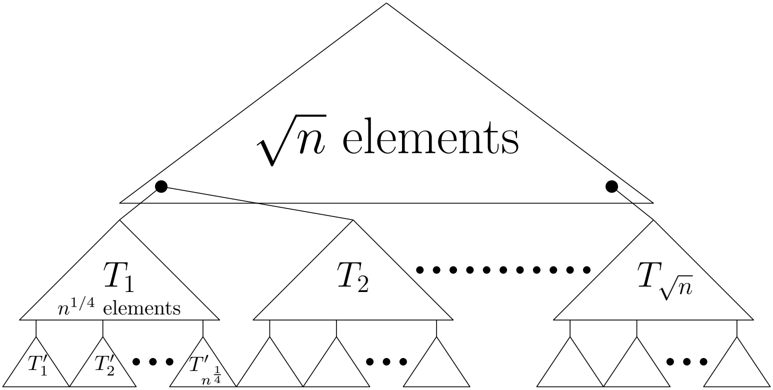
Why Is This Hard to Prove?

- Unified Bound subsumes dynamic finger bound
- Proof of dynamic finger bound is 80 pages! [CMSS00, Col00]
- Also, Unified Bound subsumes the deque bound
- Not proven for splay trees (best is $\alpha^*(n)$ [Pet08])

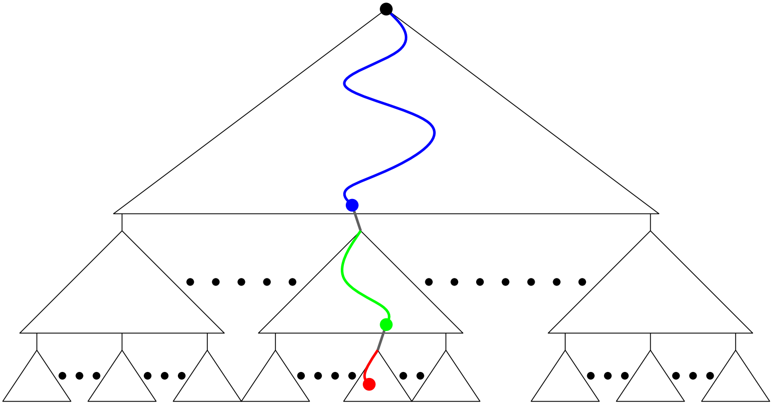
Why Is This Hard to Prove?

- Unified Bound subsumes dynamic finger bound
- Proof of dynamic finger bound is 80 pages! [CMSS00, Col00]
- Also, Unified Bound subsumes the deque bound
- Not proven for splay trees (best is $\alpha^*(n)$ [Pet08])

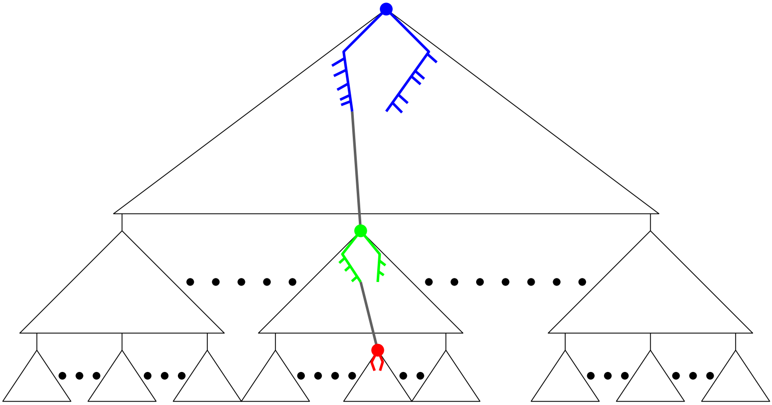
Skip-Splay: Adding a Small Amount of Structure



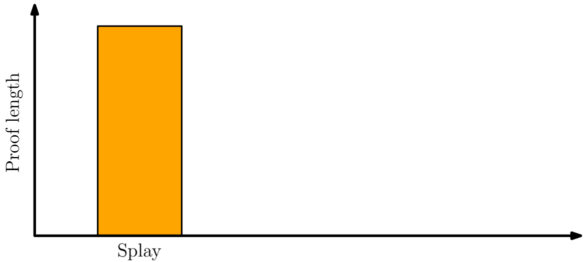
Skip-Splay: Adding a Small Amount of Structure



Skip-Splay: Adding a Small Amount of Structure

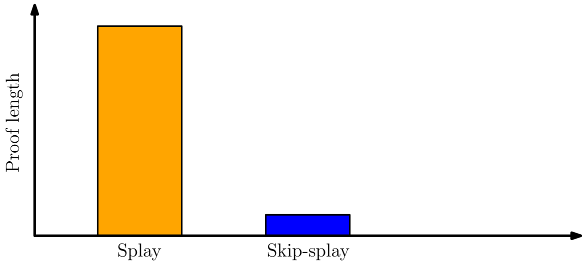


Simple BSTs Versus Simple Proofs



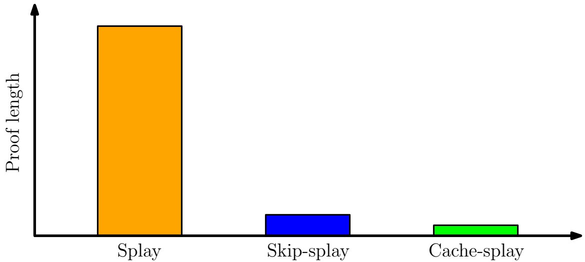
- Splay requires 80-page proof for dynamic finger
- Skip-splay requires 8-page proof for Unified Bound plus $\lg \lg n$
- Cache-splay requires 4-page proof for Unified Bound

Simple BSTs Versus Simple Proofs



- Splay requires 80-page proof for dynamic finger
- Skip-splay requires 8-page proof for Unified Bound plus $\lg \lg n$
- Cache-splay requires 4-page proof for Unified Bound

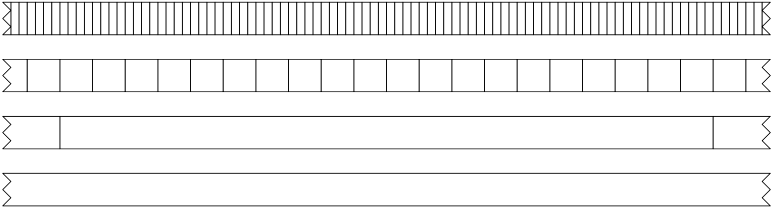
Simple BSTs Versus Simple Proofs



- Splay requires 80-page proof for dynamic finger
- Skip-splay requires 8-page proof for Unified Bound plus $\lg \lg n$
- Cache-splay requires 4-page proof for Unified Bound

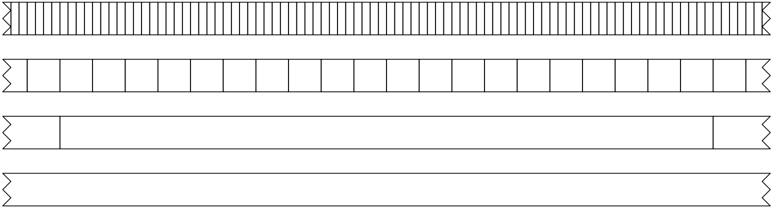
Outline

The Cache-Splay Hierarchy of Keys



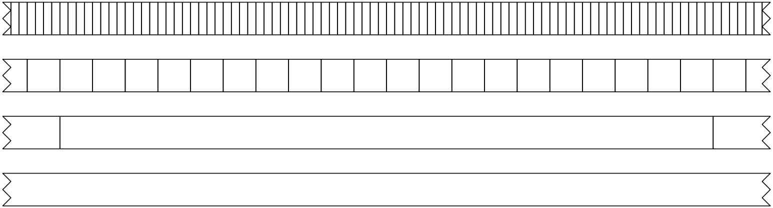
- Level-1 block contains $b_1 = 4$ keys
- Level-2 block contains b_1 level-1 blocks (16 keys)
- Level-i block contains b_{i-1} level-$(i-1)$ blocks (b_{i-1}^2 keys)
- Level-$\lg \lg n$ block contains all keys

The Cache-Splay Hierarchy of Keys



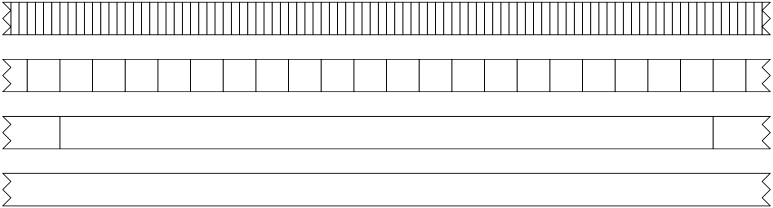
- Level-1 block contains $b_1 = 4$ keys
- Level-2 block contains b_1 level-1 blocks (16 keys)
- Level-i block contains b_{i-1} level-$(i-1)$ blocks (b_{i-1}^2 keys)
- Level-$\lg\lg n$ block contains all keys

The Cache-Splay Hierarchy of Keys



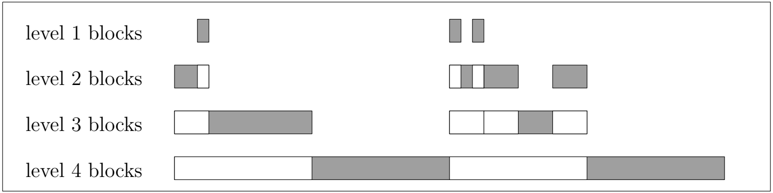
- Level-1 block contains $b_1 = 4$ keys
- Level-2 block contains b_1 level-1 blocks (16 keys)
- Level-i block contains b_{i-1} level-$(i-1)$ blocks (b_{i-1}^2 keys)
- Level-$\lg \lg n$ block contains all keys

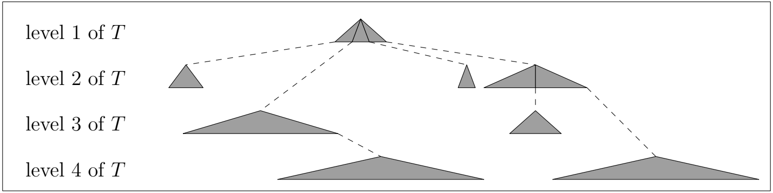
The Cache-Splay Hierarchy of Keys



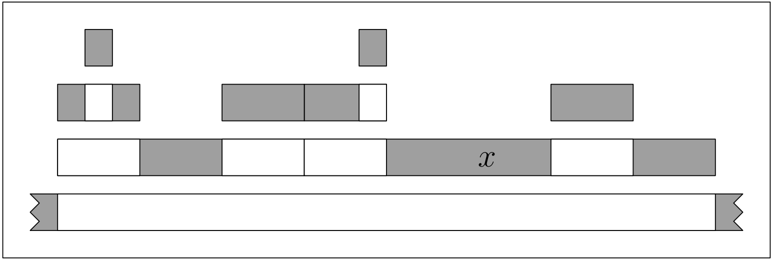
- Level-1 block contains $b_1 = 4$ keys
- Level-2 block contains b_1 level-1 blocks (16 keys)
- Level-i block contains b_{i-1} level-$(i-1)$ blocks (b_{i-1}^2 keys)
- Level-$\lg \lg n$ block contains all keys

The Cache View and the Tree View





The Cache View of a Query



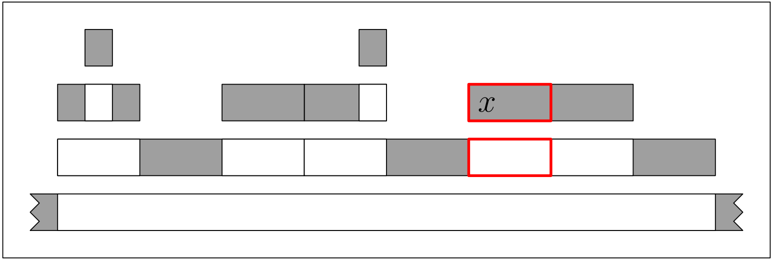
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

The Cache View of a Query



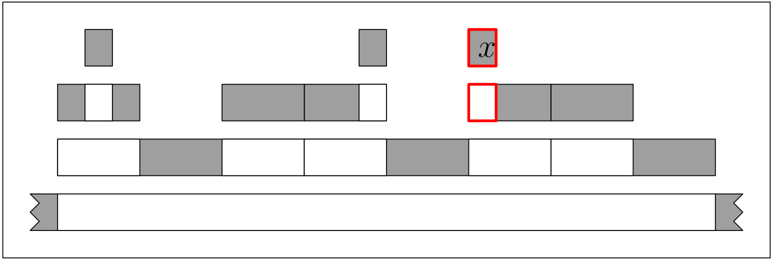
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

The Cache View of a Query



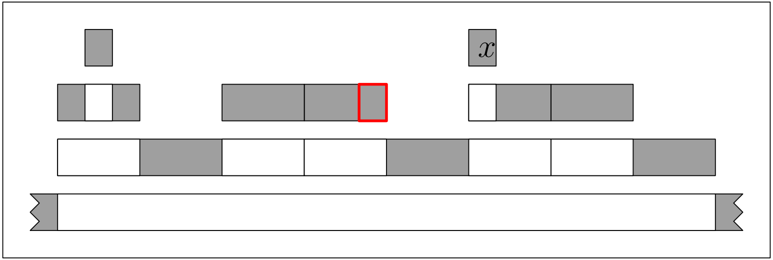
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

The Cache View of a Query



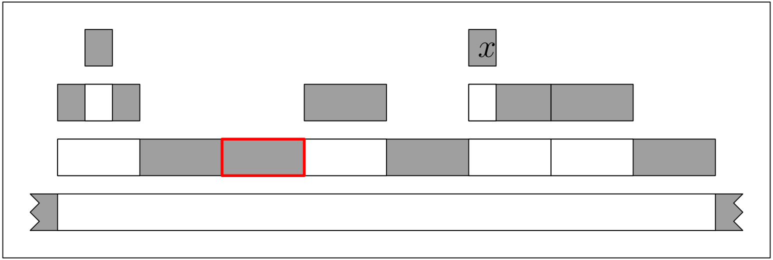
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

The Cache View of a Query



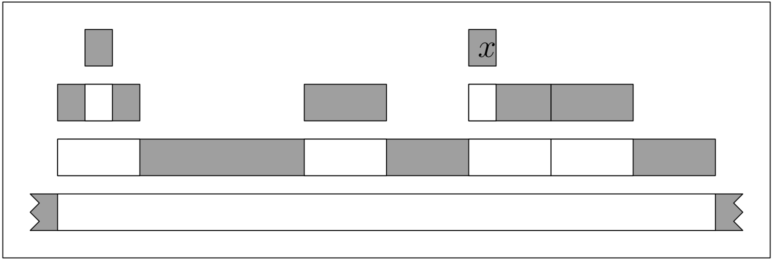
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

The Cache View of a Query



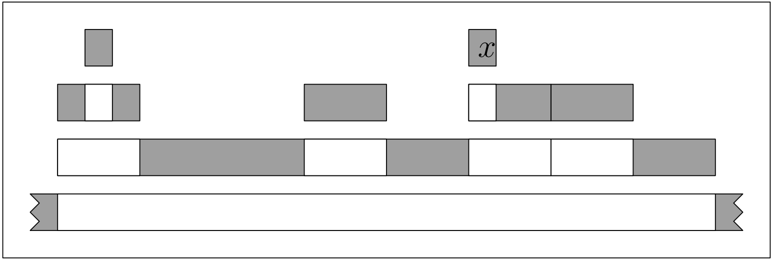
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

The Cache View of a Query



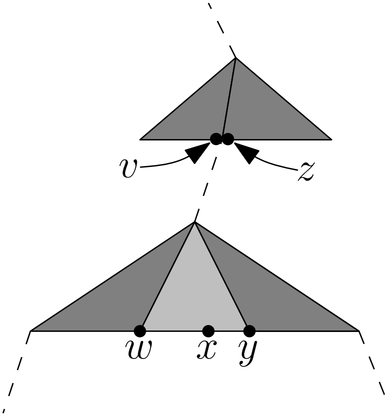
Cache view of a query to x

- Cache loop, iteration 1
- Cache loop, iteration 2
- Eject loop, iteration 1
- Eject loop, iteration 2

Important Fact

If t keys have been queried since x, then the size of x's current block size is $t^{O(1)}$.

BST Implementation of the Cache Loop

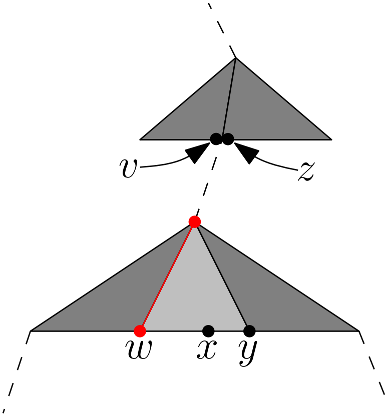


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs O(lg(block size for lower level))

BST Implementation of the Cache Loop

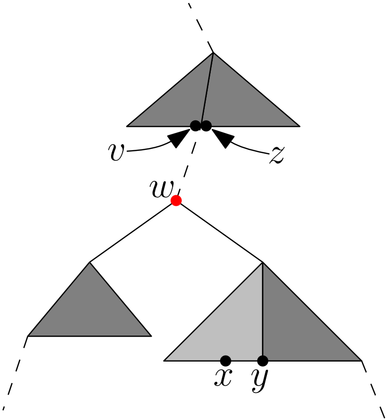


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

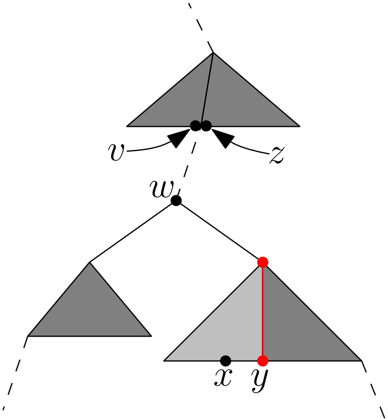


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

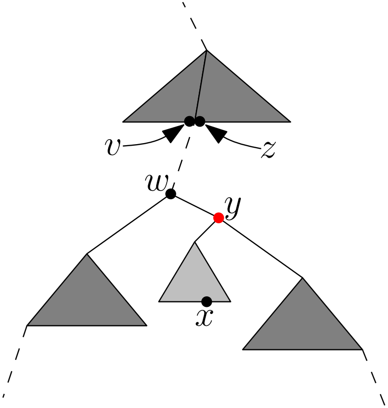


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

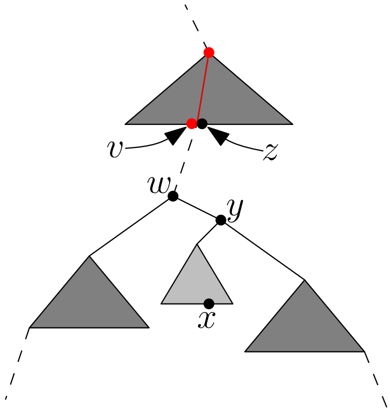


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

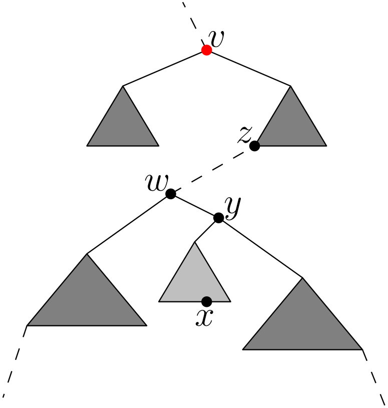


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

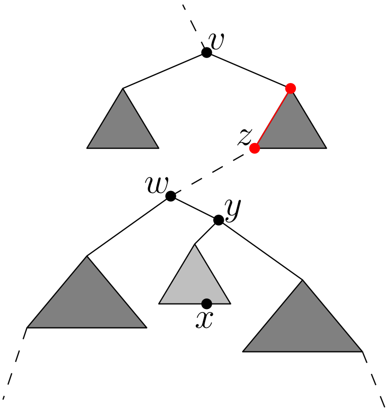


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs O(lg(block size for lower level))

BST Implementation of the Cache Loop

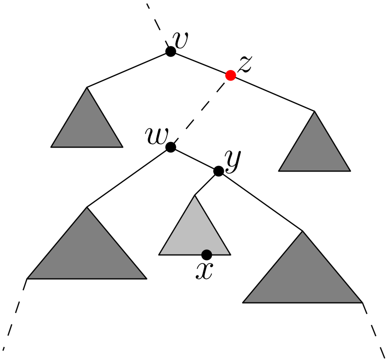


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

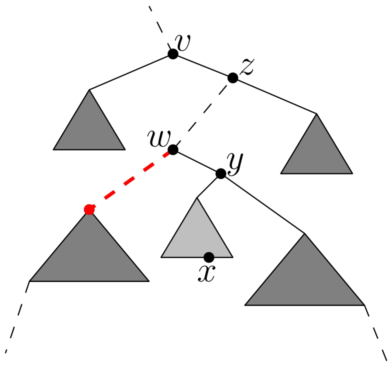


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

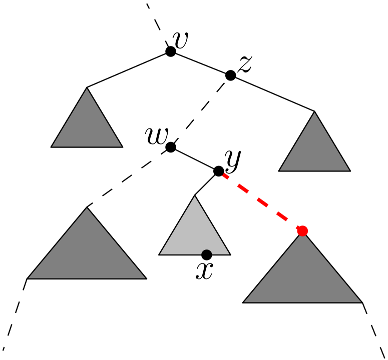


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs O(lg(block size for lower level))

BST Implementation of the Cache Loop

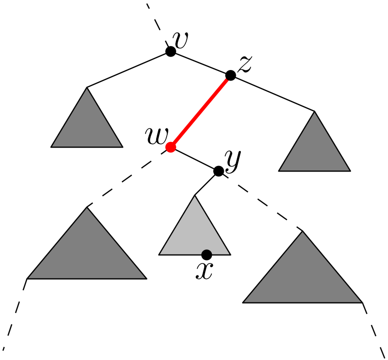


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

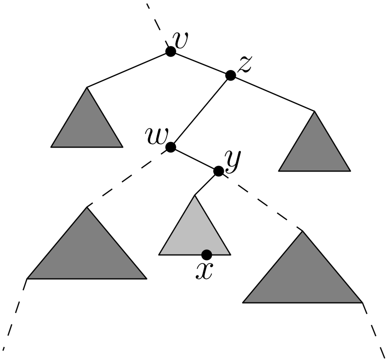


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Cache Loop

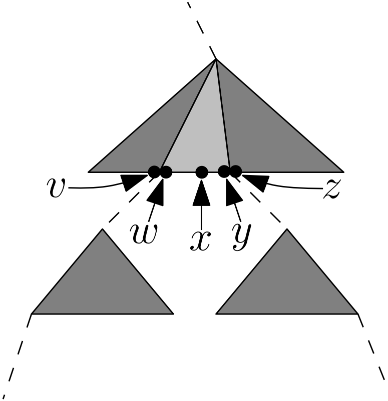


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs O(lg(block size for lower level))

BST Implementation of the Cache Loop

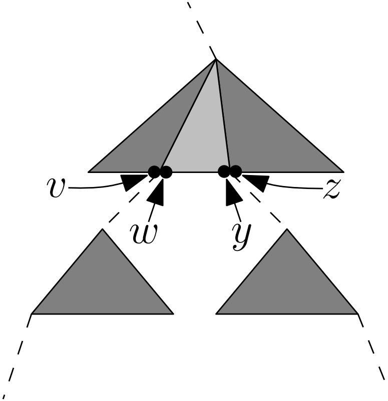


One cache loop iteration

- splay(w)
- splay(y)
- splay(v)
- splay(z)
- incRoot(leftChild(w))
- incRoot(rightChild(y))
- decRoot(w)

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

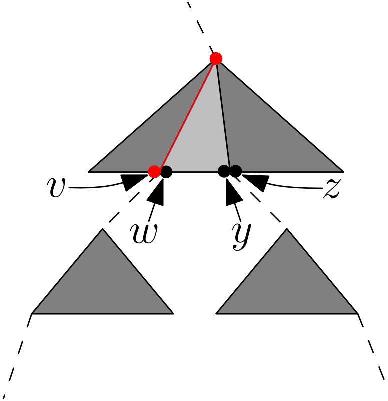


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

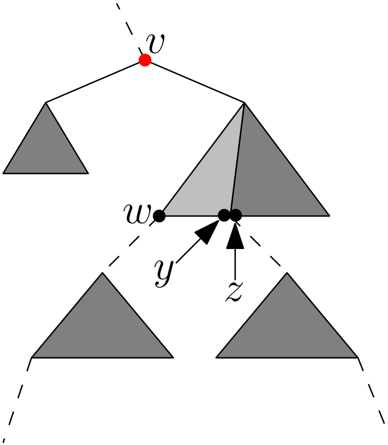


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

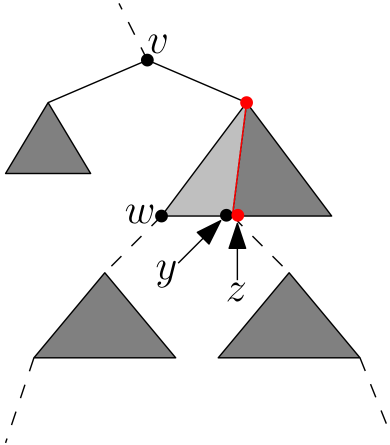


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

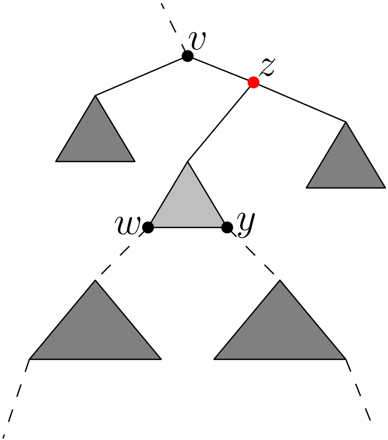


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

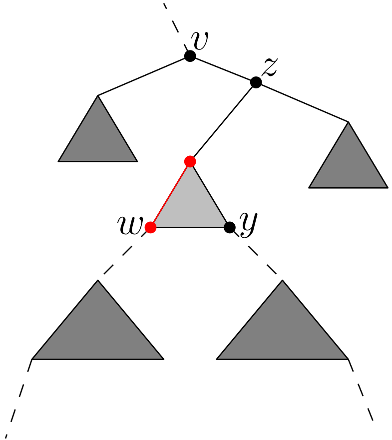


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

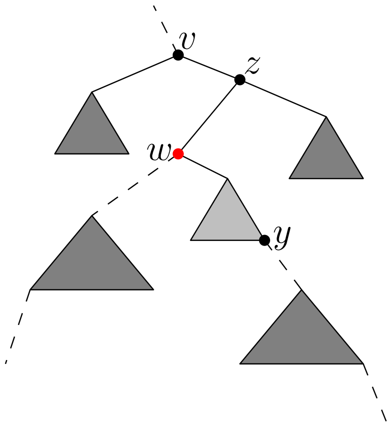


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

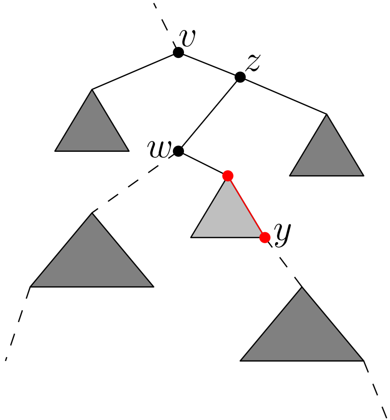


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

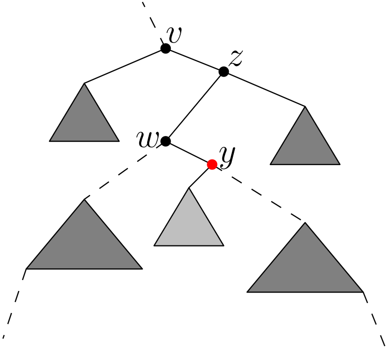


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

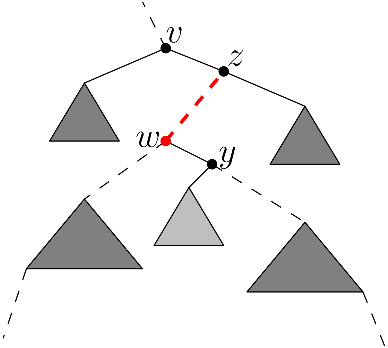


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

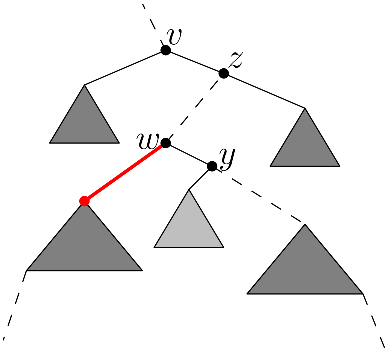


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

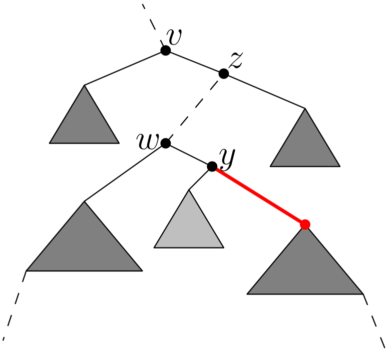


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

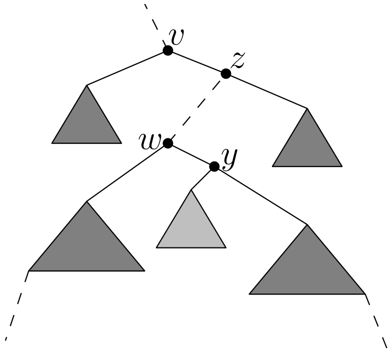


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

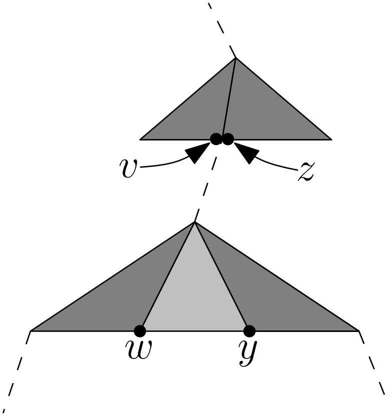


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

BST Implementation of the Eject Loop

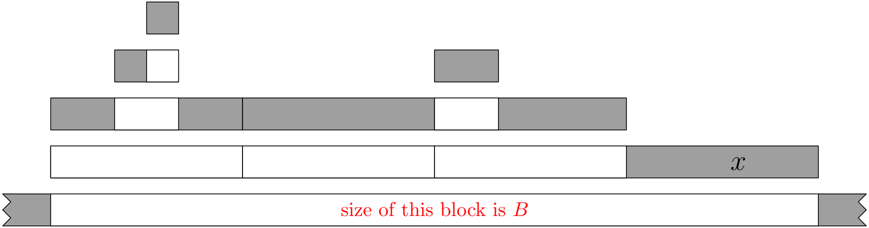


One eject loop iteration

- splay(v)
- splay(z)
- splay(w)
- splay(y)
- incRoot(w)
- decRoot(leftChild(w))
- decRoot(rightChild(y))

Each operation costs $O(\lg(\text{block size for lower level}))$

The Cost of a Query

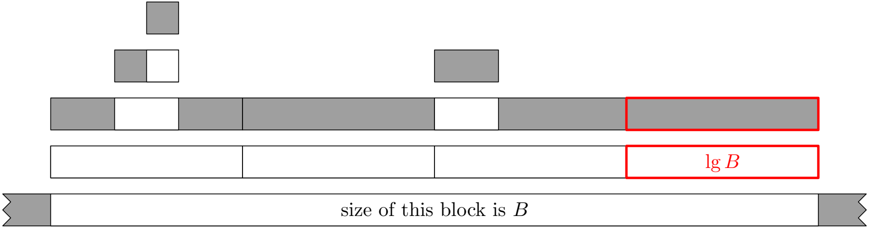


$$\lg B + \tfrac{1}{2}\lg B + \tfrac{1}{4}\lg B + \tfrac{1}{4}\lg B + \tfrac{1}{2}\lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\text{time since queried}))$.

The Cost of a Query

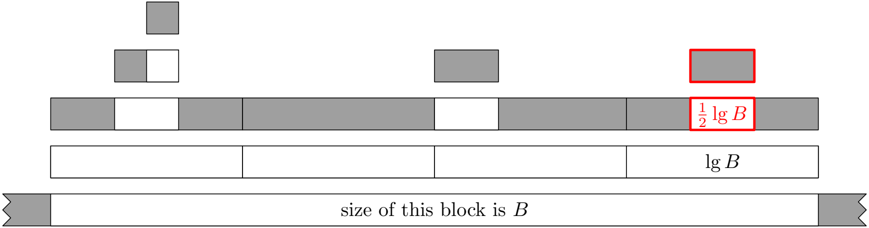


$$\lg B + \tfrac{1}{2}\lg B + \tfrac{1}{4}\lg B + \tfrac{1}{4}\lg B + \tfrac{1}{2}\lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\textit{time since queried}))$.

The Cost of a Query

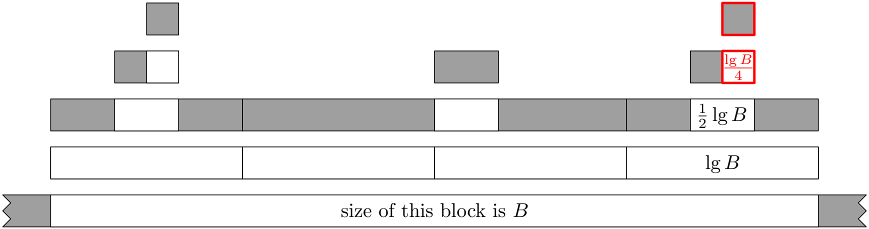


$$\lg B + \frac{1}{2}\lg B + \frac{1}{4}\lg B + \frac{1}{4}\lg B + \frac{1}{2}\lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\textit{time since queried}))$.

The Cost of a Query

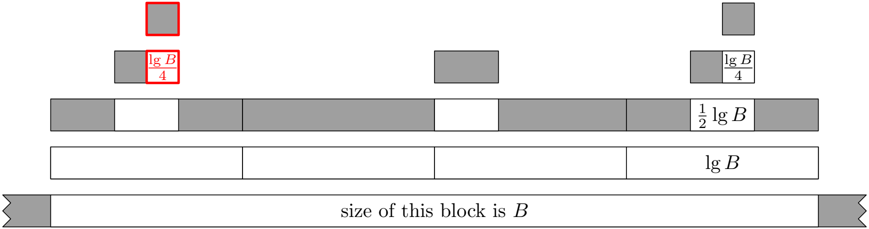


$$\lg B + \frac{1}{2}\lg B + \frac{1}{4}\lg B + \frac{1}{4}\lg B + \frac{1}{2}\lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\text{time since queried}))$.

The Cost of a Query

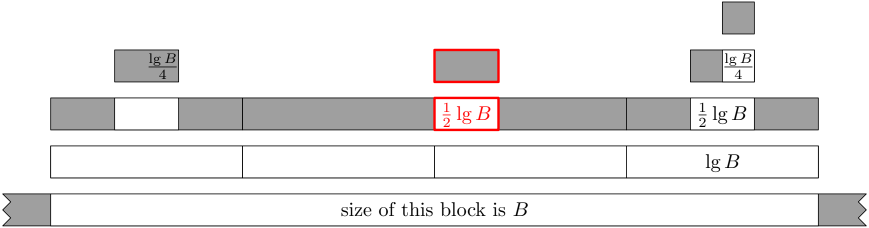


$$\lg B + \frac{1}{2} \lg B + \frac{1}{4} \lg B + \frac{1}{4} \lg B + \frac{1}{2} \lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\textit{time since queried}))$.

The Cost of a Query

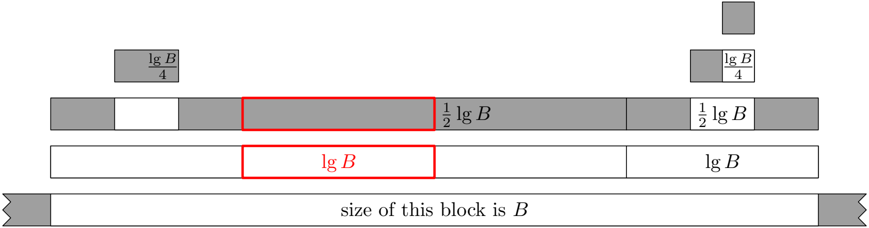


$$\lg B + \frac{1}{2} \lg B + \frac{1}{4} \lg B + \frac{1}{4} \lg B + \frac{1}{2} \lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\text{time since queried}))$.

The Cost of a Query

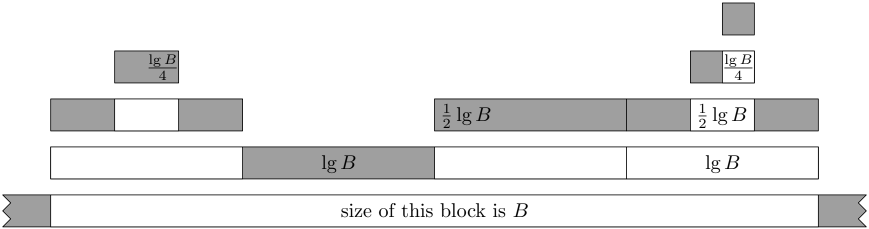


$$\lg B + \frac{1}{2} \lg B + \frac{1}{4} \lg B + \frac{1}{4} \lg B + \frac{1}{2} \lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\text{time since queried}))$.

The Cost of a Query

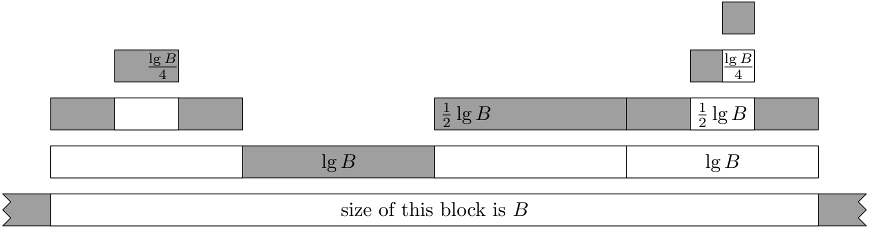


$$\lg B + \frac{1}{2}\lg B + \frac{1}{4}\lg B + \frac{1}{4}\lg B + \frac{1}{2}\lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\textit{time since queried}))$.

The Cost of a Query

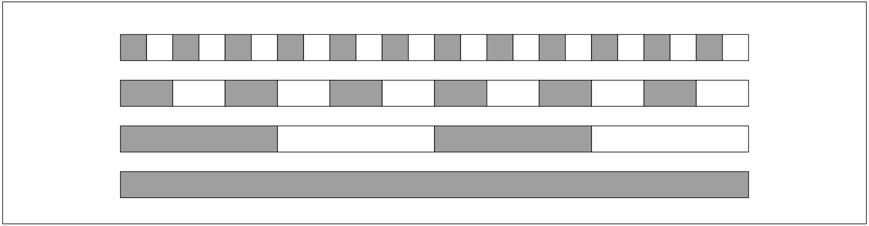


$$\lg B + \frac{1}{2}\lg B + \frac{1}{4}\lg B + \frac{1}{4}\lg B + \frac{1}{2}\lg B + \lg B = O(\lg B)$$

Lemma (Query Cost)

A query to level i costs $O(\lg b_i) = O(\lg(\textit{time since queried}))$.

Adding an Offset to the Blocks of the Cache



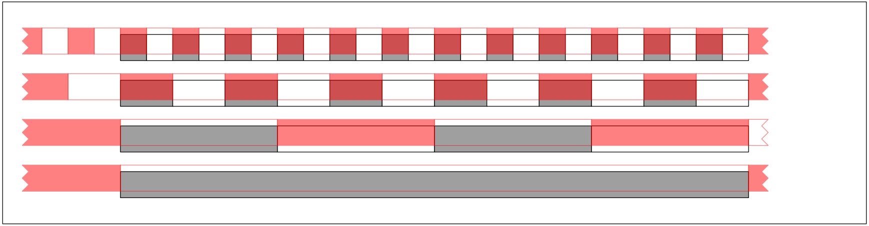
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



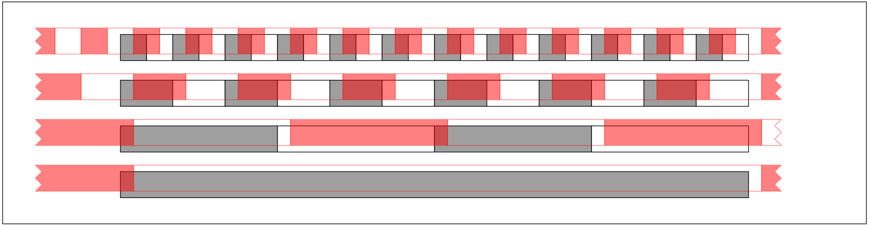
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



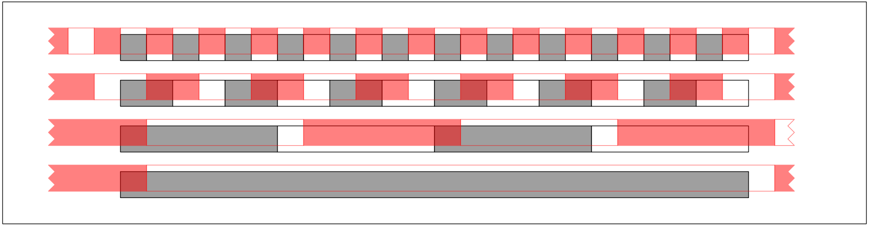
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



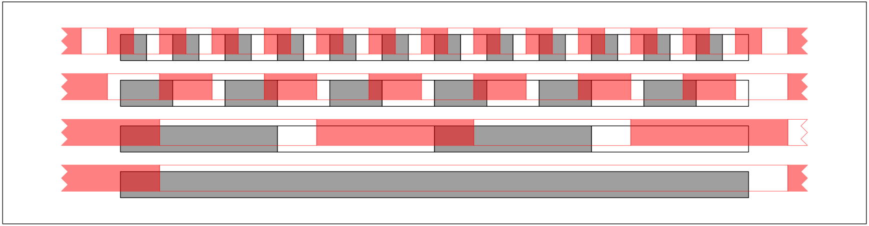
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



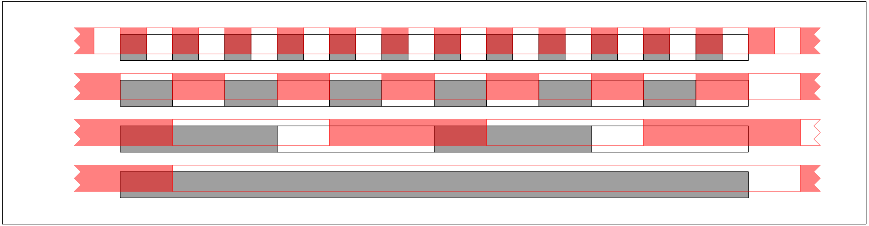
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



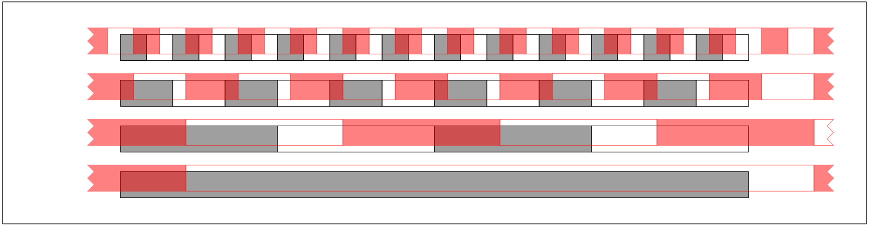
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



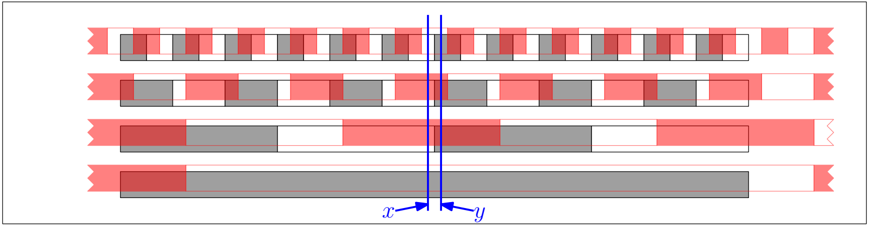
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



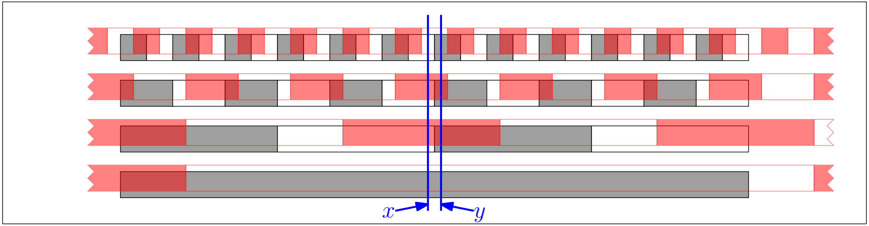
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



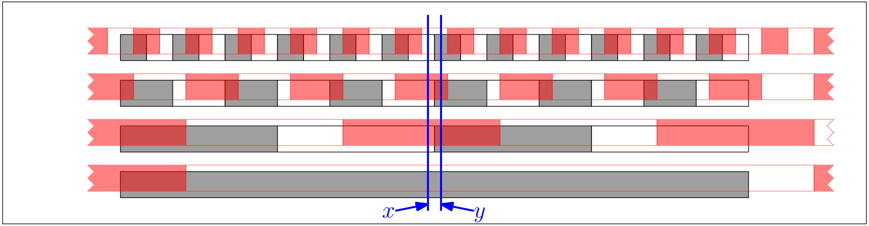
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg(\textit{time since virtual block queried}))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Adding an Offset to the Blocks of the Cache



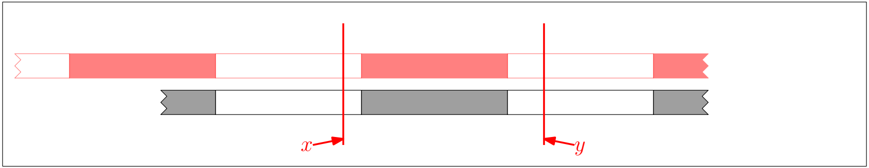
Lemma (Offset Query Cost)

A query to level i of the "virtual cache" costs amortized $O(\lg b_i)$, which is $O(\lg($time since virtual block queried$))$.

Proof.

Potential of $\lg b_j$ for each level-j block that is overlapped by a level-j virtual block that is at a higher level in the virtual cache. □

Randomizing the Offset

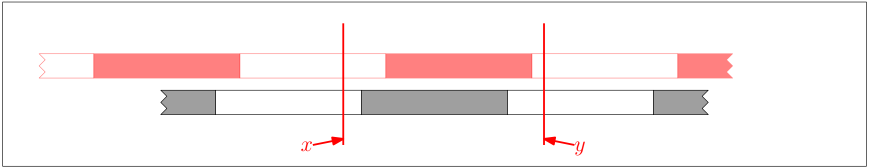


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$,*
- *If $|x - y| < b_i$,*

Randomizing the Offset

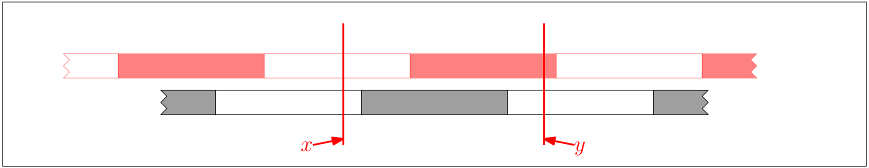


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- If $|x - y| \geq b_i$,
- If $|x - y| < b_i$,

Randomizing the Offset

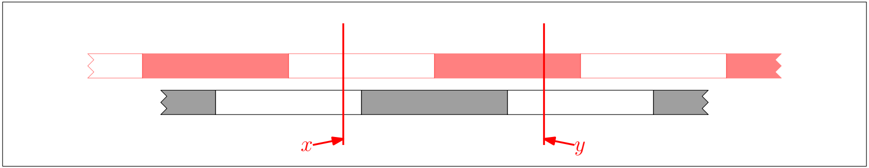


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- If $|x - y| \geq b_i$,
- If $|x - y| < b_i$,

Randomizing the Offset

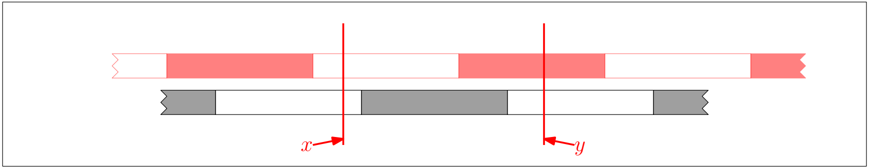


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$,*
- *If $|x - y| < b_i$,*

Randomizing the Offset

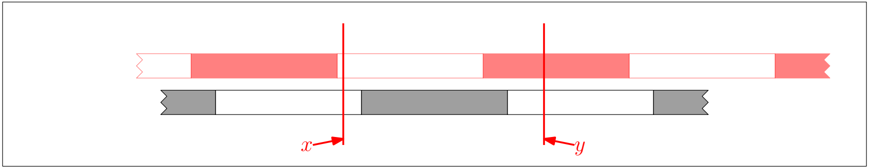


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- If $|x - y| \geq b_i$,
- If $|x - y| < b_i$,

Randomizing the Offset

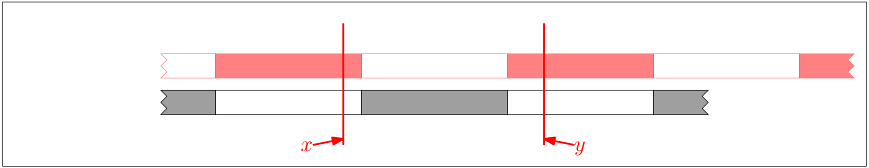


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$,*
- *If $|x - y| < b_i$,*

Randomizing the Offset

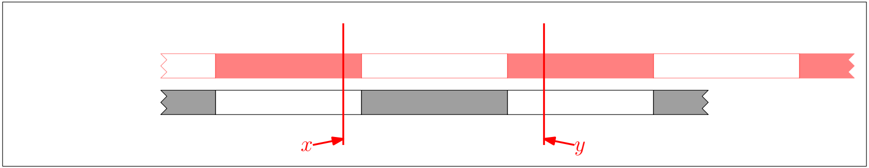


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- If $|x - y| \geq b_i$,
- If $|x - y| < b_i$,

Randomizing the Offset

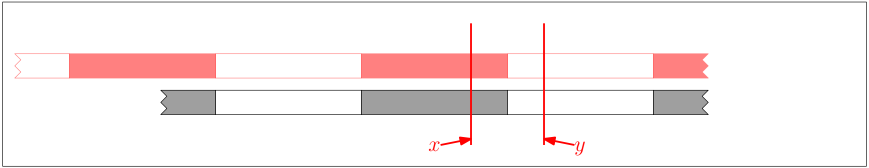


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If* $|x - y| \geq b_i$, *then probability* $= 1$.
- *If* $|x - y| < b_i$,

Randomizing the Offset

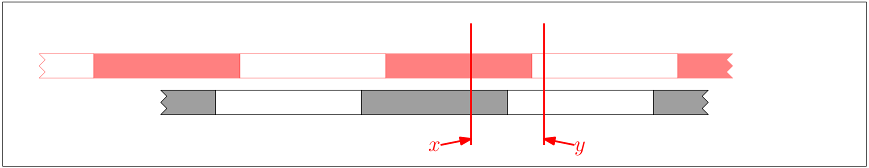


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$, then probability $= 1$.*
- *If $|x - y| < b_i$,*

Randomizing the Offset

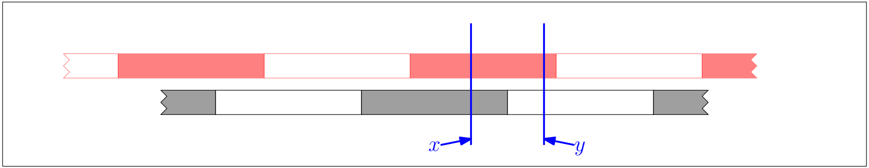


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$, then probability $= 1$.*
- *If $|x - y| < b_i$,*

Randomizing the Offset

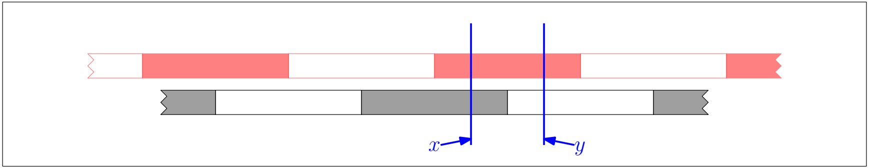


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- If $|x - y| \geq b_i$, then probability $= 1$.
- If $|x - y| < b_i$,

Randomizing the Offset

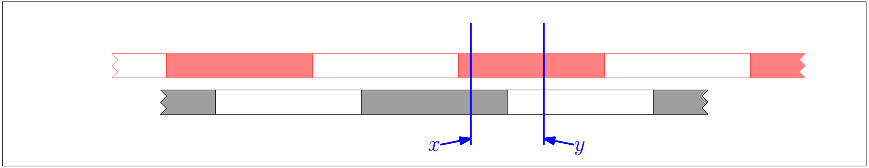


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If* $|x - y| \geq b_i$*, then probability* $= 1$.
- *If* $|x - y| < b_i$,

Randomizing the Offset

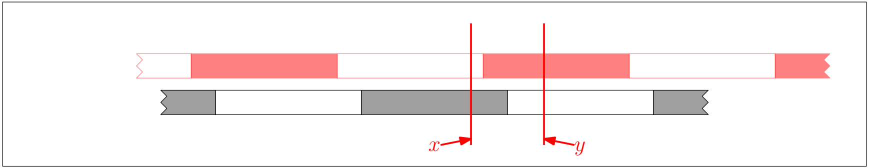


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$, then probability $= 1$.*
- *If $|x - y| < b_i$,*

Randomizing the Offset

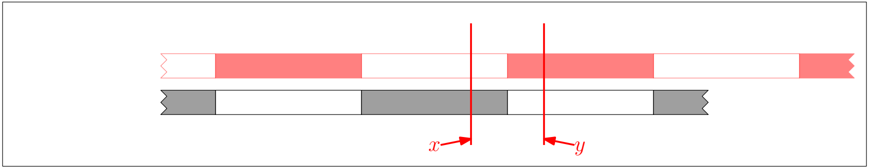


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$, then probability $= 1$.*
- *If $|x - y| < b_i$,*

Randomizing the Offset

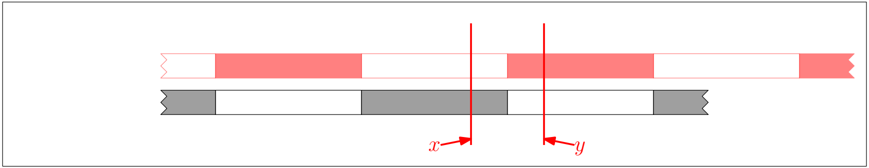


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$, then probability $= 1$.*
- *If $|x - y| < b_i$,*

Randomizing the Offset

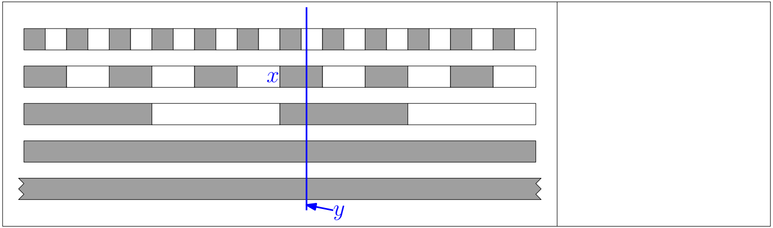


Lemma (Random Offset)

For random offset, are x and y in different level-i virtual blocks?

- *If $|x - y| \geq b_i$, then probability $= 1$.*
- *If $|x - y| < b_i$, then probability $= \frac{|x-y|}{b_i}$.*

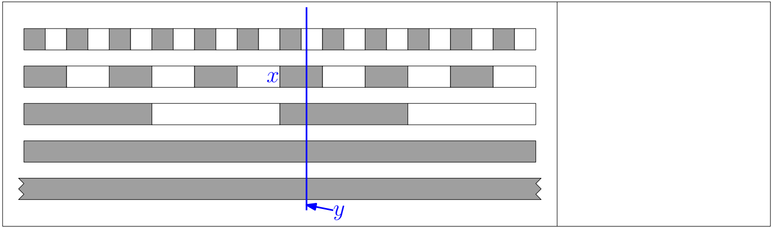
Cache-Splay Satisfies the Unified Bound



Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x - y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x - y| \geq b_i$, then cost is expected $O(\lg |x - y|)$.
- Cost is expected $O(\quad \lg t_x + \lg |x - y| \quad)$.

□

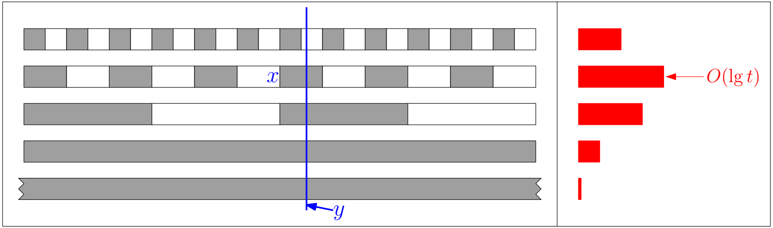


Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x - y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x - y| \geq b_i$, then cost is expected $O(\lg |x - y|)$.
- Cost is expected $O(\quad \lg t_x + \lg |x - y| \quad)$.

□

Cache-Splay Satisfies the Unified Bound

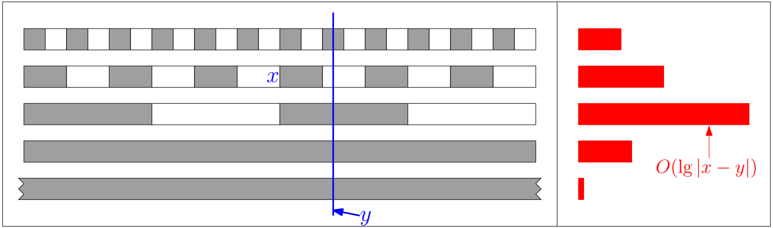


Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x - y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x - y| \geq b_i$, then cost is expected $O(\lg |x - y|)$.
- Cost is expected $O(\quad \lg t_x + \lg |x - y| \quad)$.

□

Cache-Splay Satisfies the Unified Bound

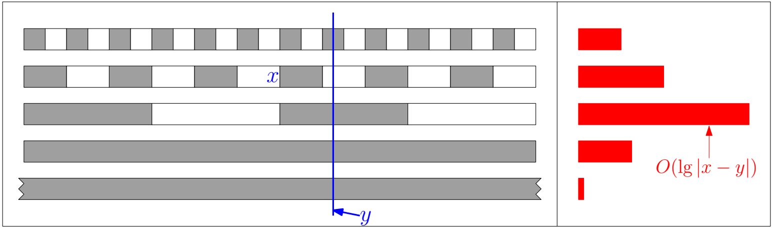


Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x - y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x - y| \geq b_i$, then cost is expected $O(\lg |x - y|)$.
- Cost is expected $O(\quad \lg t_x + \lg |x - y| \quad)$.

□

Cache-Splay Satisfies the Unified Bound

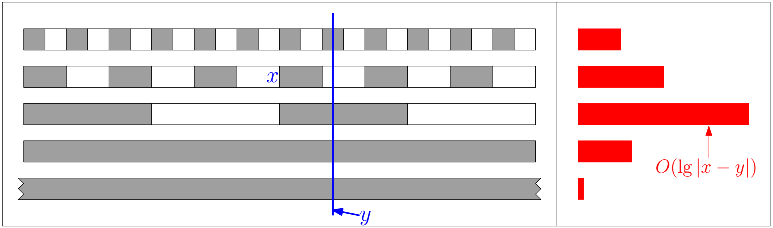


Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x-y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x-y| \geq b_i$, then cost is expected $O(\lg|x-y|)$.
- Cost is expected $O(\quad \lg t_x + \lg|x-y| \quad)$.

□

Cache-Splay Satisfies the Unified Bound

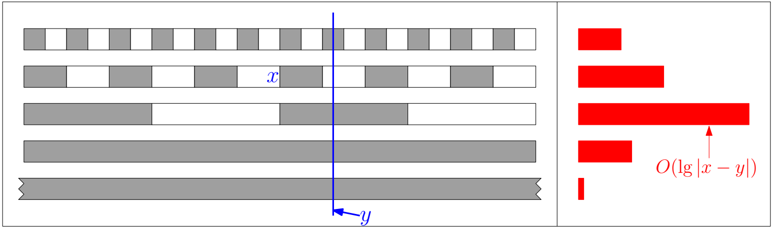


Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x-y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x-y| \geq b_i$, then cost is expected $O(\lg|x-y|)$.
- Cost is expected $O(\min_x(\lg t_x + \lg|x-y|))$.

□

Cache-Splay Satisfies the Unified Bound



Proof.

- Suppose we query x, then t_x other keys, then y.
- Then x's level in virtual cache has block size $b_i = t_x^{O(1)}$.
- If $|x - y| < b_i$, then cost is expected $O(\lg t_x)$.
- If $|x - y| \geq b_i$, then cost is expected $O(\lg |x - y|)$.
- Cost is ~~expected~~ $O(\min_x(\lg t_x + \lg |x - y|))$.

□

Implications

- Splay trees must satisfy the Unified Bound to be $O(1)$-competitive
- Search for even more general formulaic bounds?

Implications

- Splay trees must satisfy the Unified Bound to be $O(1)$-competitive
- Search for even more general formulaic bounds?

Outline

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG$^+$07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG+07]
  - Easy instances of combinatorial auctions [CDS04]

Contributions

- Lower bounds
  - Dynamic interleave bound [WDS06]
  - MIBS bound [DSW05]: generalization of Wilber's bounds that is computable in polynomial time
  - ...also shows the strength of the BST model because it is a lower bound for partial-sums (in thesis)
- BST upper bounds
  - Multi-splay [WDS06, DSW09]: first $O(\lg \lg n)$-competitive BST to achieve other properties of an optimal BST
  - Skip-splay [DS09]: simple BST similar to splaying within additive $O(\lg \lg n)$ of the Unified Bound
  - Cache-splay (in thesis): more complicated splay-based algorithm that achieves the Unified Bound (open since [Iac01])
- Other results
  - High-dimensional finger search [DSSW08]: first finger search data structure for k-d approximate nearest-neighbor
  - Experiments with bipartite parametric max-flow [BDG$^+$07]
  - Easy instances of combinatorial auctions [CDS04]

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg\lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Future Work

- Better bounds for *splaying*: Unified Bound, $o(\lg n)$-competitiveness, digit-reversal permutation, generalization of the Unified Bound, working set for splaying without rotate-to-root, new toolbox for analyzing splay trees with splaying over induced subtrees.
- Better bounds for *any* BST: use better lower bounds to show $o(\lg \lg n)$-competitiveness for some BST, show that some formulaic bound that implies BST competitiveness to within a $o(\lg n)$ factor
- Further justification for the BST *model* itself: show MIBS is a lower bound for partial-sums in a more general model, reduce BST model to partial-sums problem.

Thanks!

[BCK02] Avrim Blum, Shuchi Chawla, and Adam Kalai. Static optimality and dynamic search-optimality in lists and trees. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA 2002)*, pages 1–8, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[BDDF09] Prosenjit Bose, Karim Douïeb, Vida Dujmović, and Rolf Fagerberg. An o(log log n)-competitive binary search tree with optimal worst-case access times. Obtained on December 7, 2009 from: http://cgm.cs.mcgill.ca/ vida/pubs/papers/ZipperTrees.pdf, 2009.

[BDG+07] Maxim Babenko, Jonathan Derryberry, Andrew Goldberg, Robert Tarjan, and Yunhong Zhou. Experimental evaluation of parametric max-flow algorithms. In *Proceedings of the 6th Workshop on Experimental Algorithms (WEA 2007)*, pages 256–269, 2007.

[CDS04] Vincent Conitzer, Jonathan Derryberry, and Tuomas Sandholm. Combinatorial auctions with structured item graphs. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI 2004)*, pages 212–218. AAAI Press / The MIT Press, 2004.

[CMSS00] Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. part I: Splay sorting log n-block sequences. *SIAM Journal on Computing*, 30(1):1–43, 2000.

[Col00] Richard Cole. On the dynamic finger conjecture for splay trees. part II: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

[DHI+09] Erik D. Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms (SODA 2009)*, pages 496–505, 2009.

[DHIP04] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality – almost. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 484–490, 2004.

[DS09] Jonathan C. Derryberry and Daniel D. Sleator. Skip-splay: Toward achieving the unified bound in the bst model. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures (WADS 2009)*, pages 194–205, Berlin, Heidelberg, 2009. Springer-Verlag.

[DSSW08] Jonathan Derryberry, Don Sheehy, Daniel D. Sleator, and Maverick Woo. Achieving spatial adaptivity while finding approximate nearest neighbors. In *Proceedings of the 20th Canadian Conference on Computational Geometry (CCCG 2008)*, pages 163–166, 2008.

[DSW05] Jonathan Derryberry, Daniel Dominic Sleator, and Chengwen Chris Wang. A lower bound framework for binary search trees with rotations. Technical Report CMU-CS-05-187, Carnegie Mellon University, 2005.

[DSW09] Jonathan Derryberry, Daniel Sleator, and Chengwen Chris Wang. Properties of multi-splay trees. Technical Report CMU-CS-09-171, Carnegie Mellon University, 2009.

[Geo08] George F. Georgakopoulos. Chain-splay trees, or, how to achieve and prove log log n-competitiveness by splaying. *Information Processing Letters*, 106(1):37–43, 2008.

[Iac01] John Iacono. Alternatives to splay trees with o(log n) worst-case access times. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA 2001)*, pages 516–522, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[PD04] Mihai Pătrașcu and Erik D. Demaine. Tight bounds for the partial-sums problem. In *In Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 20–29, Philadelphia, PA, USA, 2004. Society for Industrial and Applied Mathematics.

[Pet08] Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA 2008)*, pages 1115–1124, 2008.

[WDS06] Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O(log log n)-competitive dynamic binary search trees. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete*

Algorithms (SODA 2006), pages 374–383, New York, NY, USA, 2006. ACM.

[Wil89] Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989.