Wilber's second lower bound \[ \text{wilber}_2(x_j) \]:
- look at where \( x_j \) fits among \( x_i, x_{i+1}, \ldots, x_{j-1} \) for \( i = j-1, j-2, \ldots \) until previous access to \( x_j \)
- say \( a_i < x_j < b_i \) is the tightest pair for \( i \)
- as \( i \) decreases, consider move \( x_i \)’s
  \( \Rightarrow \) bounds tighten: \( a_i \) increases & \( b_i \) decreases
- \( \text{wilber}_2(x_j) \) = # alternations between \( a_i \) increasing & \( b_i \) decreasing

Geometric view:

- \( \text{wilber}_2(x) = \sum_{j=1}^{m} \text{wilber}_2(x_j) \)

is a lower bound on all BSTs for \( x \)
[proof somewhat similar to Wilber 1 proof below]
**Key-independent optimality:** [Iacono – ISAAC 2002]

- Suppose key values are “meaningless”
- Might as well permute them uniformly at random
- Then \( E_{\pi}[\text{dynamic OPT}(\pi(x_1), \pi(x_2), \ldots, \pi(x_n))] \)
  \[ = \text{working-set bound } \Theta(\sum_{i=1}^{m} \lg t_i(x_i)) \]

\( \Rightarrow \) splay trees, etc. are key-independently optimal

**Proof sketch:**

0 trivial; \( \Omega \) via wilber2

- \( \text{wilber2}(x_j) \) just considers working set of \( x_j \):
  \( W = \{ \text{elements accessed since last access to } x_j \} \)
- Only one access to \( x_i, i < j \), plays a role
- \( \pi \) permutes this working set
- \( x_j \) falls roughly in the middle of \( W \)
- \( E[\# \text{times } a_i \text{ increases}] = \Theta(\lg |W|) \)
  - Like \( E[\# \text{times max. changes in prefixes of random list}] = \frac{\sum_{i=1}^{k} 1}{i} \)

- \( E[\# \text{times } b_i \text{ increases}] = \Theta(\lg |W|) \)
- Claim: expected constant fraction of these changes interleave

\( \Rightarrow \) \( \text{wilber2}(x_j) = \Theta(\lg |W|) = \Theta(\lg t_j(x_j)) \)

\( \square \)

**OPEN:** \( \text{wilber2} = \Theta(\text{dynamic OPT})? \)
Wilber's first lower bound: [Wilber–SIComp 1989]

- fix a lower-bound tree on same keys
  e.g. perfect binary tree
- for each node y of P:
  - label each access $x_i$ as
    L if key $x_i$ is in y's left subtree
    R if key $x_i$ is in y's right subtree
    blank if $x_i = y$ or $x_i$ outside y's subtree
- $\text{interleave}(y) =$ number of alternations between L & R
- $\text{wilber}1(x) = \text{interleave}(x) = \sum_y \text{interleave}(y)$

is a lower bound on all BSTs for x

Proof sketch:

- define transition point of node y in P to be
  highest node z in BST T such that root-to-z
  path includes nodes from left & right subtrees of y
- well-defined
- doesn't change until z touched
- different for different y

- must touch transition point of y in next
  L or R access i.e. within next 2 interleaves
  $\Rightarrow$ must touch $\geq \text{interleave}(x)/2$ nodes. $\square$
O(lg \lg n)-competitive BST: Tango trees

[Demaine, Harmon, Iacono, Patrascu - FOCS 2004]

- define preferred child of node y in P to be left if accessed left subtree(y) more recently right if accessed right subtree(y) more recently none if no access to either subtree yet

- preferred path = chain of preferred child pointers
- partition of nodes in P
- idea: store each preferred path in auxiliary tree:

  - conceptually separate balanced BST
  - leaves link to roots of aux. trees of children paths
  - search starts at top aux. tree (containing root(P))
  - aux. tree has \leq lg n nodes [if height(P)=lg n]

  \Rightarrow search in O(lg \lg n) time

  - each jump to next aux. tree = nonpreferred edge
  - preferred edge change = interleave

  - visit k aux. trees \Rightarrow interleave \geq k-1

- O(k lg lg n) search cost, \Omega(k) lower bound

  \Rightarrow O(lg lg n)-competitive (modulo update cost)

- dynamic finger aux. trees \Rightarrow get O(k lg \frac{lg n}{k})

  \Rightarrow cost = O(OPT \cdot \lg \frac{lg n}{OPT/n}) ... actually wilber1, not OPT

- tightest possible bound in terms of n & wilber1 (by entropy)
Auxiliary trees:
- changing a preferred child corresponds to cutting one path & joining two paths:
  \[ P \]
  \[ \text{cut} \quad \text{join} \]
- cut path into nodes with depth \( \leq d \) & \( > d \)
- join two paths with consecutive depths
- like split & concatenate in balanced BSTs (immediate pointer-machine DS)
- except aux. trees ordered by key, not depth
\[ \Rightarrow \text{depth} > d \text{ translates to interval of keys} \]
- extraction/insertion of interval of keys can be implemented by \( O(1) \) splits & concatenates
\[ \Rightarrow O(lg \text{ aux. tree}) = O(lg lg n) \text{ time/interleave} \]
Combining aux. trees into one BST:
- each aux. tree is a contiguous subtree
- mark root of each aux. tree
- define $\text{split}$ & $\text{concat}$

modifications of standard logarithmic $\text{split}$ & $\text{concat}$ to use temporary roots, ignoring children trees

- cut is then:

- join is the reverse
- need depth augmentation to find $l$ & $r$