MADALGO Cache-Obl. Data Struct.	Aug. 19, 2008

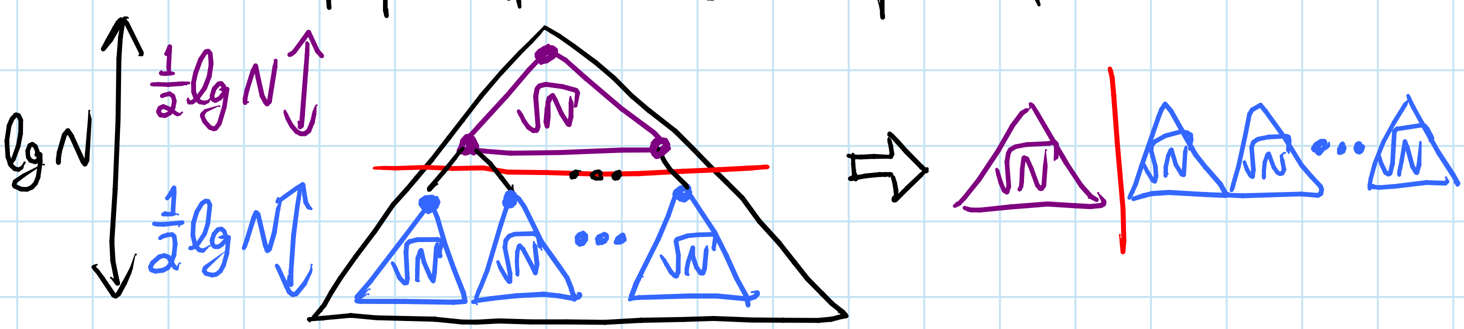
Static search trees: [P99; BDF05]

- predecessor/successor among N elements in $O(\log_{B+1} N)$ memory transfers
- binary search, but elts. in special order

van Emde Boas layout:

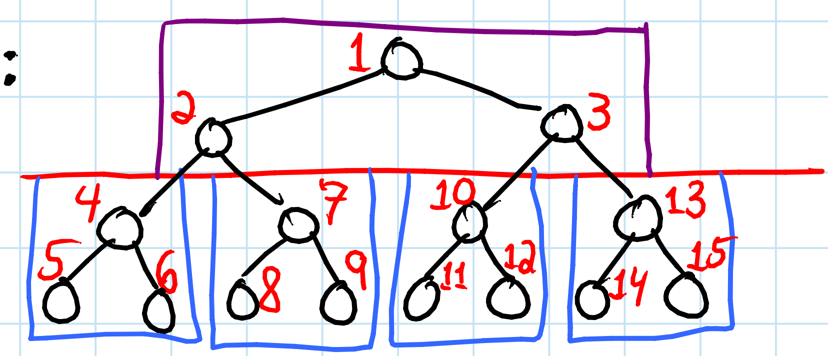
- build complete BST on N nodes storing N elements in order
- carve tree at middle level of edges

$\Rightarrow$ one top piece, $\approx\sqrt{N}$ bottom pieces, each size $\approx\sqrt{N}$



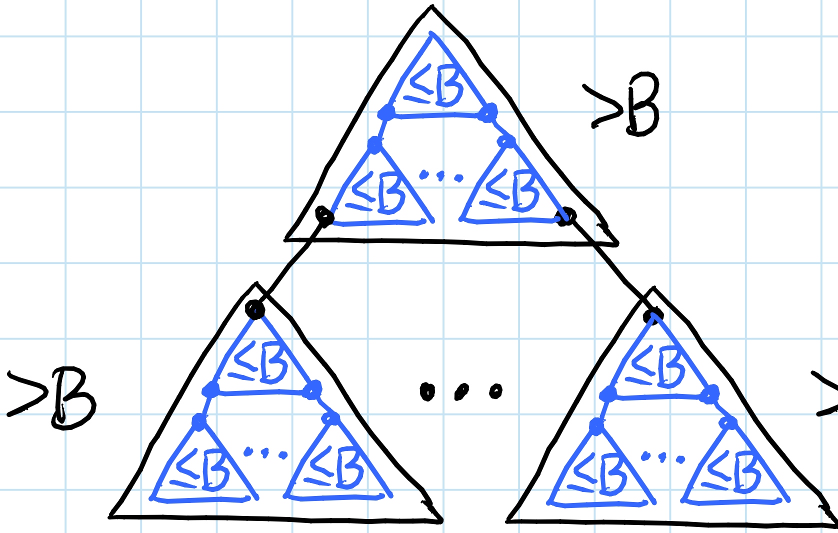
- recursively lay out pieces & concatenate

Example:



Analysis:

- level of detail (refinement) straddling B:



- cutting height in half until piece size $\le B$

$\Rightarrow$ height of piece between $\frac{1}{2}\lg B$ & $\lg B$ (sloppy)
($\Rightarrow$ size between $\sqrt{B}$ & B)

$\Rightarrow$ # pieces along root-to-leaf path $\le \dfrac{\lg N}{\frac{1}{2}\lg B} = 2\log_B N$

- each piece stores $\le B$ elements consecutively

$\Rightarrow$ occupies ≤ 2 blocks (depending on alignment)

$\Rightarrow$ # memory transfers $\le 4\log_B N$ (assuming $M \ge 2B$)
(really should be $B+1$)

Improvements: [BBFGHHIL$\emptyset$3]

① randomize starting location (w.r.t. block)
$\Rightarrow$ expected cost $\le (2+\frac{3}{\sqrt{B}})\log_B N$

② split height into $\frac{1}{2}-\varepsilon : \frac{1}{2}+\varepsilon$ ratio
$\Rightarrow$ expected cost $\le (\lg e + \underbrace{o(1)}_{\approx O(\lg\lg B/\lg B)})\log_B N$

Dynamic search trees: [BDF05; BDIW04; BFJ02]

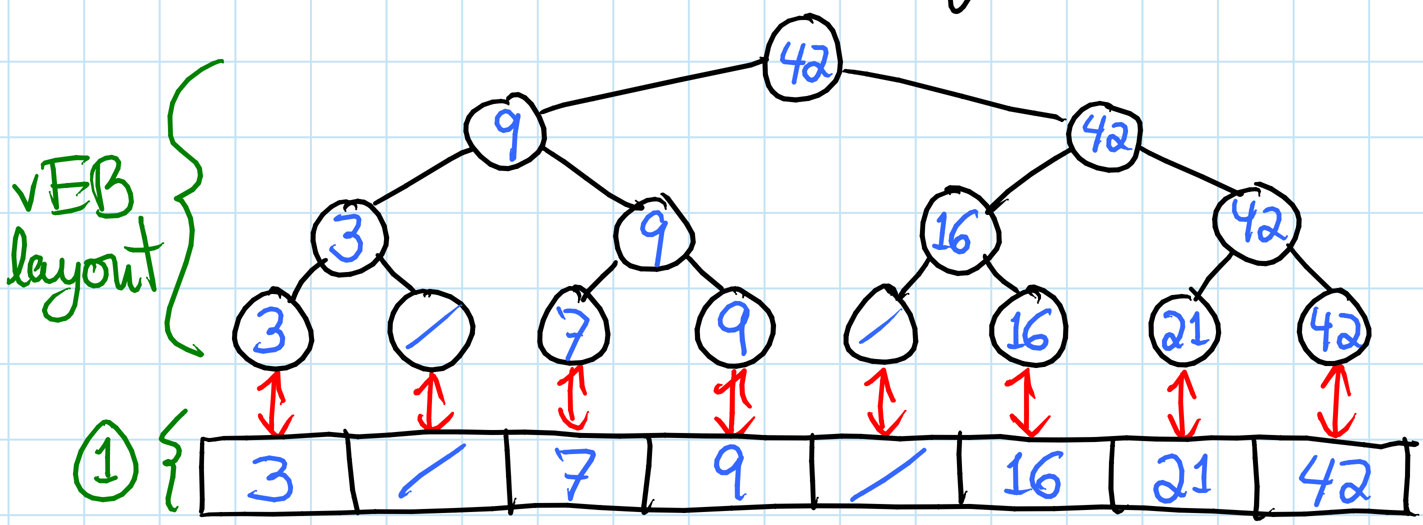
① ordered file maintenance: [later]

store N elements in specified order
in an array of size $O(N)$ with $O(1)$ gaps

— updates: insert element between two given
delete element

by re-arranging array interval of $O(\lg^2 N)$ am.

② build static search tree on top:
each node stores max key in subtree (if any)



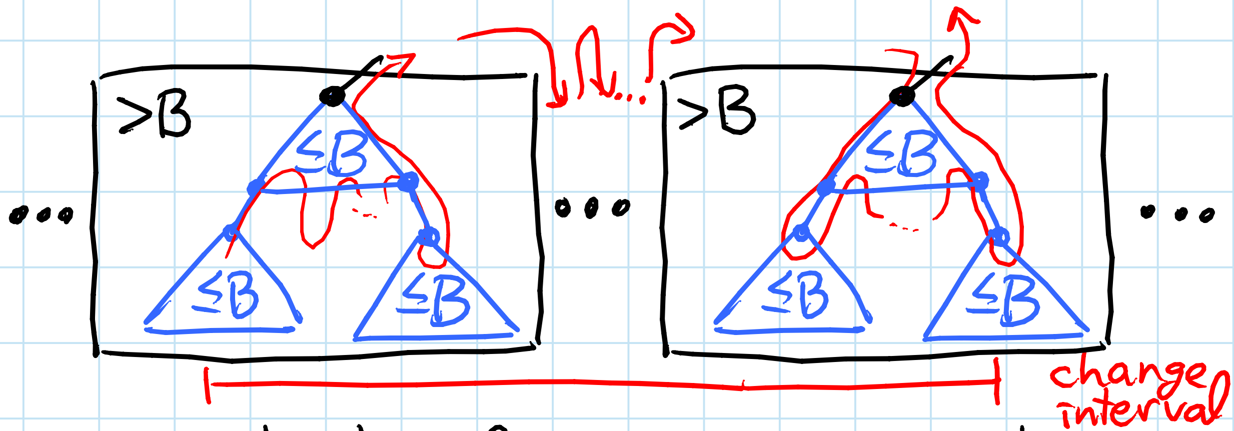
③ operations:

— binary search via left child's key
— insert(x) finds predecessor & successor,
inserts there in ordered file,
& updates leaves & max's up tree
via postorder traversal
— delete similar

(4) update analysis:

if K cells change in ordered file then update tree in $O(\frac{K}{B} + \log_B N)$ mem.tr.

- look at level of detail straddling B
- look at bottom two levels:



- within chunk of $>B$, jumping between ≤ 2 pieces of $\leq B$ (assume $M \geq 2B$)

$\Rightarrow O(\text{chunk}/B)$ memory transfers in chunk
portion in update interval +3 maybe (first, last, & root)

$\Rightarrow O(\frac{K}{B})$ memory transfers in bottom 2 levels

- updated nodes above these two levels:
  - subtree of $\leq \frac{K}{B}$ chunk roots up to their LCA: costs $O(\frac{K}{B})$
  - path from LCA to root of tree: costs $O(\log_B N)$ as above

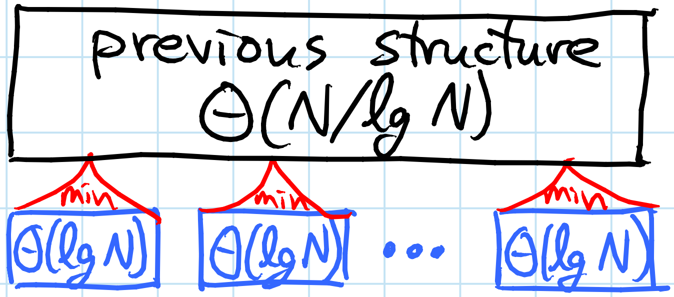
$\Rightarrow O(\frac{K}{B} + \log_B N)$ total memory transfers

So far: search in $O(\log_B N)$
update in $O(\log_B N + \frac{\lg^2 N}{B})$ amortized

bad if $B = o(\lg N \lg\lg N)$

⑤ <u>indirection</u>:

- cluster elements into $\Theta(\frac{N}{\lg N})$ groups, each of size $\Theta(\lg N)$
- use previous structure on min's of clusters



- update cluster by complete rewrite
  $\Rightarrow O(\frac{\lg N}{B})$ memory transfers
- split/merge clusters as necessary to keep between 25% & 100% full
  $\Rightarrow \Omega(\lg N)$ updates to charge to

$\Rightarrow O(\frac{\lg^2 N}{B})$ update cost in top structure only "every" $\Omega(\lg N)$ actual updates

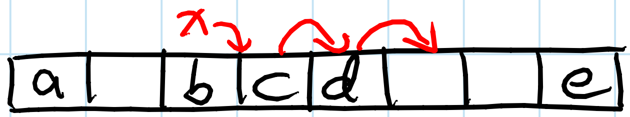
$\Rightarrow$ amortized update cost $O(\frac{\lg N}{B})$ (plus search cost)

<u>Finally</u>: $O(\log_B N)$ insert, delete, predecessor, successor just like B-trees in external mem. (known B)

Variations:

- partially persistent [BCR02]
- scan support [BCDFC02]
- (suboptimal) update-query trade-off [BFFFKN07]
- concurrent & lock-free [BFGK05]
- implicit [FG03a]
- worst-case [FG03b]

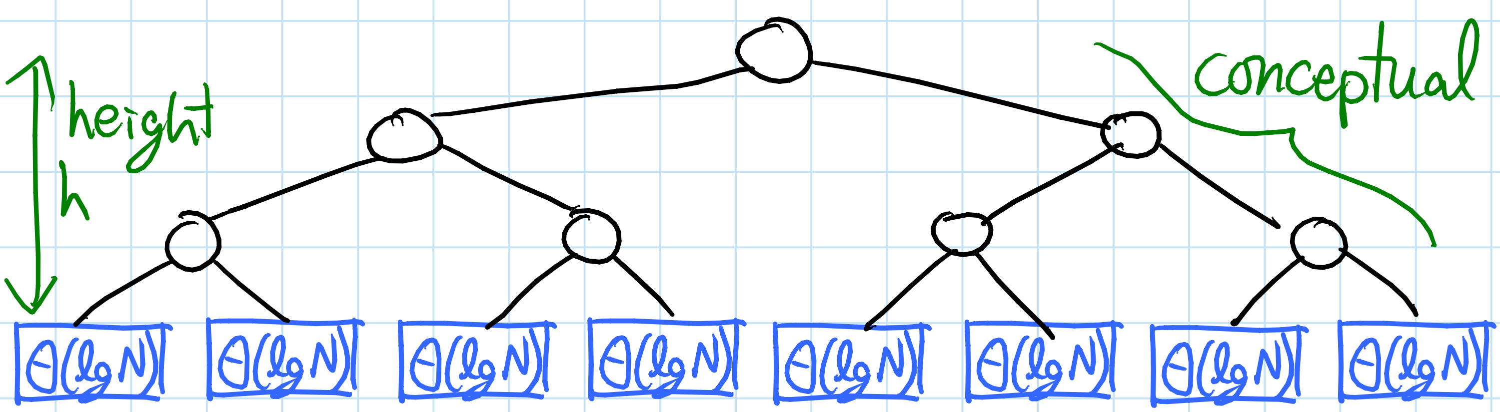
Ordered-file maintenance: [Itai, Konheim, Rodeh 1981]
[BDFC05]



<u>Idea</u>: allow arbitrary density anywhere, but when updating an element, ensure locally not too dense or sparse

- grow an interval around the element until interval not too dense or sparse (requirement depends on interval size)
- evenly redistribute elements in interval

<u>In fact</u>: grow intervals by walking up complete binary tree built atop $\Theta(\lg n)$-size chunks of array (indirection)



Update:

① update leaf by rewriting $\Theta(\lg n)$-size chunk

② walk up tree until reach ancestor whose

$$\underline{\text{density}}(\text{node}) = \frac{\#\text{ elts. stored below node}}{\#\text{ array slots in interval}}$$

is <u>within threshold</u> at its depth d:

– density $\geq \frac{1}{2} - \frac{1}{4}\frac{d}{h} \in [\frac{1}{4}, \frac{1}{2}]$ (not too sparse)

– density $\leq \frac{3}{4} + \frac{1}{4}\frac{d}{h} \in [\frac{3}{4}, 1]$ (not too dense)

③ evenly rebalance elements below nodes within array interval of node

Analysis:

– thresholds get tighter as we go up

⇒ rebalancing a node puts children <u>far</u> within their threshold:

$$|\text{density} - \text{threshold}| \gtrsim \frac{1}{4}\frac{1}{h} = \Theta\left(\frac{1}{\lg N}\right)$$

– this node won't be rebalanced again until ≥ 1 child out of threshold

⇒ $\underbrace{\Omega\left(\frac{\text{capacity}}{\lg N}\right)}_{\Omega(1)\text{ because leaf = chunk has size }\Theta(\lg N)}$ updates to charge to

⇒ $O(\lg N)$ amortized rebuild cost to update element below a node

– each leaf is below $h = \Theta(\lg N)$ ancestors

⇒ $O(\lg^2 N)$ amortized cost per update

<u>Worst-case</u> possible [Willard 1992; BCDFCZ02]

Linked list: [BCDFC02]

delete/insert element between two given

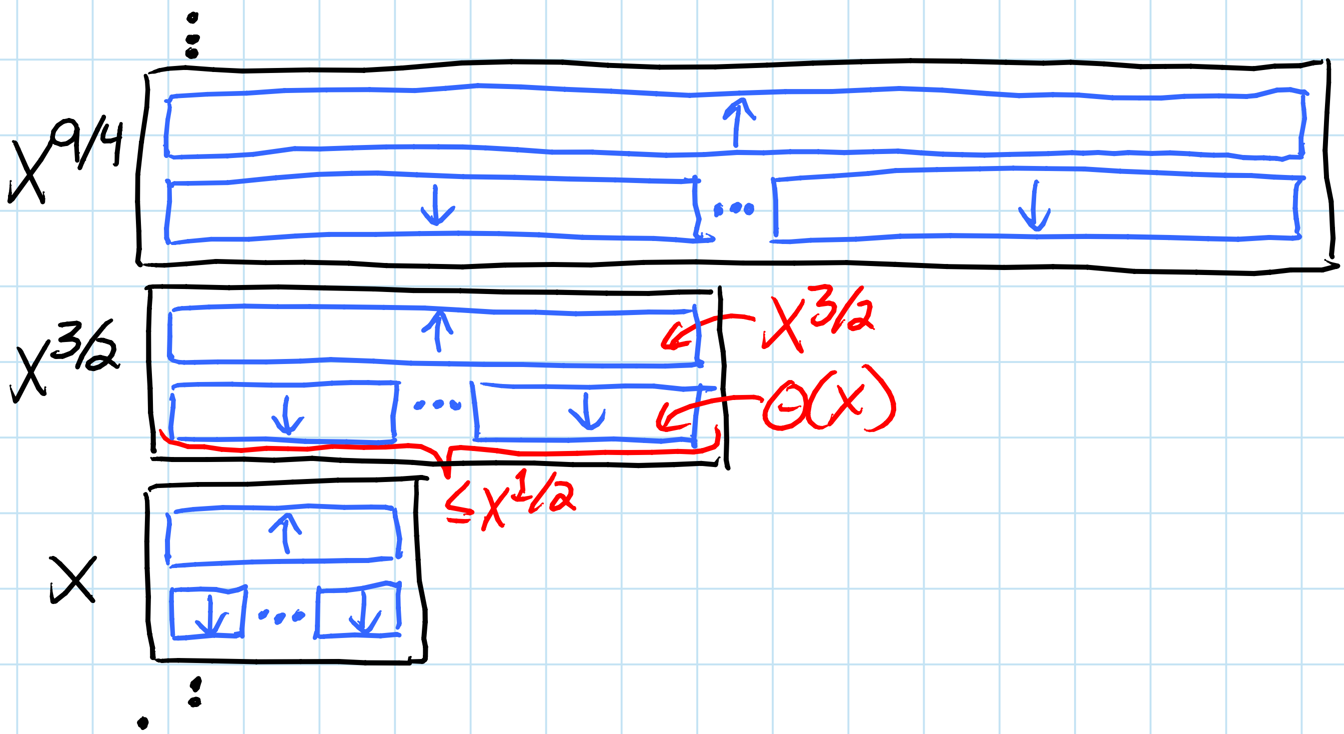
scan K consecutive elements

- for $O(\lceil \frac{K}{B} \rceil)$ worst-case scans,
  best known is $O(\frac{\lg^2 N}{B})$ update
  from ordered-file maintenance
- for $O(\lceil \frac{K}{B} \rceil + [B^\varepsilon \text{ if } B^{1-\varepsilon} \le K \le B^{1+\varepsilon}])$ w.-c. scans,
  best known is $O\left(\frac{(\lg \lg N)^{2+\varepsilon}}{B}\right)$ am. update

- but with $O(\lceil \frac{K}{B} \rceil)$ amortized scans,
  can achieve $O(1)$ amortized update:
  - insert & delete like linked list
    (insert allocates memory somewhere)
    $\Rightarrow$ add 1 or 2 discontinuities
  - scan costs $O(D + \lceil \frac{K}{B} \rceil)$ for D discontinuities
    & rewrites the K elements contiguously
    $\Rightarrow$ add ≤ 2 discontinuities & remove D
    $\Rightarrow$ can charge $O(D)$ cost to D-2 decrease

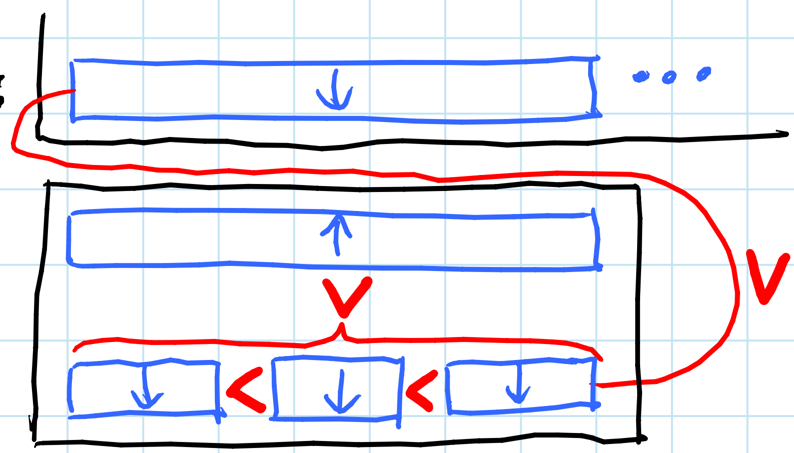
Priority queue: [ABDHM07; BF02a]

- $\lg \lg n$ levels of size $N, N^{2/3}, N^{4/9}, \ldots, c = O(1)$
- level $X^{3/2}$ has 1 up buffer of size $X^{3/2}$ & $\leq X^{1/2}$ down buffers each of size $\Theta(X)$ where all but first is const. frac. full



Layout: store levels in order, small to large

Invariants:



- down buffers ordered in a level (but unsorted)
- down buffers @ $X^{3/2}$ < down buffers @ $X^{9/4}$
- down buffers < up buffer in same level

Find-min: smallest element in smallest down buffer

Delete-min: delete from down buffer; if empty, pull

Insert: put into level c (up or down buffer)
- if up buffer overflows: push

Push X elements into level $X^{3/2}$
all > down buffers at level X below
1. sort elements
2. distribute among down & up buffers:
   - scan elements, visiting down bufs. in order
   - when down buf. overflows, split in half & link
   - when # down bufs. overflows, move last to up buf.
   - when up buf. overflows, push it up to $X^{9/4}$

Pull X smallest elts. from level $X^{3/2}$ (& above)
1. sort first two down bufs. & extract leading elts.
2. if < X: pull $X^{3/2}$ smallest elts. from $X^{9/4}$ (& above)
   sort these elements & up buffer
   refill up buffer to previous size
   with largest elements
   extract needed smallest elts. till X total
   split rest up into down buffers

<u>Analysis</u>: push/pull at level $X^{3/2}$ sans recursion costs $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers

- assume all levels of size $\le M$ stay in cache
- tall cache assumption: $M \ge B^2$ (say)
- push at level $X^{3/2} \ge B^2 \Rightarrow X > B^{4/3} \Rightarrow \frac{X}{B} > 1$
  - sort costs $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers
  - distribute costs $O(X^{1/2} + \frac{X}{B})$ mem. transf.

    startup per down buf. scan
  - if $X \ge B^2$ then cost $= O(\frac{X}{B})$
  - else: only <u>one</u> such level: $B^{4/3} \le X \le B^2$

    can keep 1 block per down buf. in cache:

    $X \le B^2 \Rightarrow X^{1/2} \le B \le \frac{M}{B}$ by tall cache

    so just pay $O(\frac{X}{B})$ at this level too
- pull at level $X^{3/2} \ge B^2$:
  - sort costs $O(\frac{X}{B} \log_{M/B} \frac{X}{B})$ memory transfers
  - another sort of $X^{3/2}$ elts. <u>only</u> when recursing $\Rightarrow$ charge to recursive pull

<u>Total</u>: each element goes up & then down

(roughly — real proof harder)

& costs $O(\frac{1}{B} \log_{M/B} \frac{X}{B})$ per push & pull @ X

$\Rightarrow O(\frac{1}{B} \sum_{X} \log_{M/B} \frac{X}{B})$ amortized cost per element

geometric

$= O(\frac{1}{B} \log_{M/B} \frac{N}{B})$.

Buffered repository tree: [ABDHM07]

supports insert(x) in $O(\frac{1}{B} \lg \frac{N}{B})$ memory trans.

& extract-all-copies(x) in $O(\lg N)$, amortized

(using just $M = O(B)$ cache)

Structure: balanced binary search tree

+ linked list of down buffers per node

+ linked list of up buffers per node

Invariants: up buffers store keys matching node's

& down buffers store keys

belonging in this subtree

Insert: add to down buffer of root

- keep just one down buffer at root via stack or doubling array $\Rightarrow O(\frac{1}{B})$

Extract: binary search for desired key

& return up buffers there

- on each node along path:

- split down bufs. into $<$key, $=$key, $>$key

up buffer

left down buf. right down buf.

$\Rightarrow O(\#\text{down bufs.} + \#\text{elts.}/B)$ mem. trans.

charge to past splits

- pay for $O(\lg N)$ splits

(I think can also rebalance new keys in $O(\lg n)$)

OPEN: cache-oblivious "buffer trees"
[Zeh] supporting
- insert(x)
[- delete(x)]
- delayed-predecessor/successor(x)
- "give me all answers" once @end

in $O(\frac{1}{B} \log_{M/B} \frac{N}{B})$ amortized mem. trans./op.