6.851 | Lecture 8 | Mar. 13, 2012

<u>TODAY</u>: Memory Hierarchies II (of 3)
- ordered file maintenance (for B-tree in L7)
- list labeling (for persistence in L1)
- cache-oblivious priority queue

<u>Ordered file maintenance</u>: [Itai, Konheim, Rodeh – ICALP 1981; Bender, Demaine, Farach-Colton – FOCS 2000]

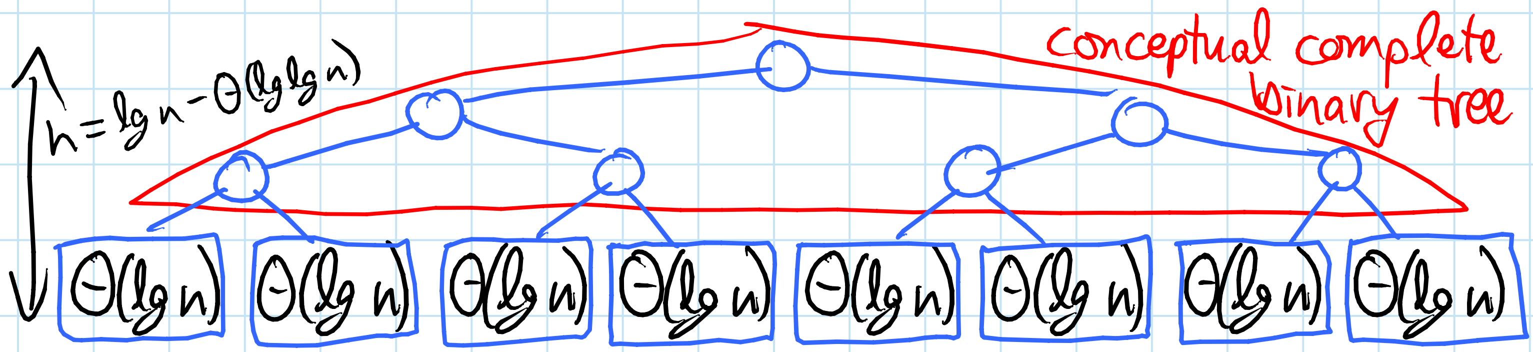
<u>Goal</u>: store N elements in specified order in an array of size $O(N)$ with gaps of size $O(1)$

$\Rightarrow$ scanning K consecutive elts. costs $O(\lceil \frac{K}{B} \rceil)$ mem. trans.

subject to elt. deletion & insertion between 2 elts. by re-arranging elts. in array interval of $O(\lg^2 N)$ amortized elts., via $O(1)$ interleaved scans

$\Rightarrow$ costs $O(\frac{\lg^2 N}{B})$ amortized memory transfers

<u>Idea</u>: upon updating element, ensure locally not too dense/sparse by redistributing elements in surrounding interval
- intervals defined by nodes in complete binary tree on $\Theta(\lg n)$-size chunks of array:



Update:

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

② walk up tree until reach ancestor whose

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

is within threshold 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 node

Analysis:

- thresholds get tighter as we go up

$\Rightarrow$ rebalancing node puts children FAR within threshold:

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

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

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

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

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

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

Worst-case bounds possible [Willard — I&C 1992; Bender, Cole, Demaine, Farach-Colton, Zito — ESA 2002]

$\boxed{\text{~~Conjecture~~ theorem}}$: $\Omega(\lg^2 N)$ necessary [Bulánek, Koucký, Saks — SICOMP 2013]

List labeling: closely related problem

maintain explicit integer label in each node in a linked list, subject to insert/delete node here, such that labels are monotone at all times

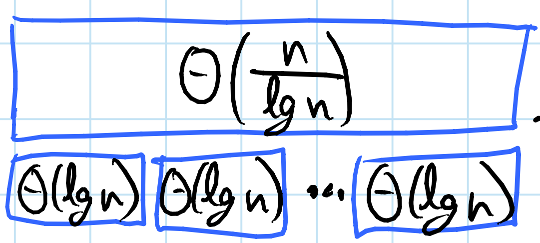
(label = index in array)

label space	best known time/update
$(1+\varepsilon)n$ .. $n \lg n$	$\Theta(\lg^2 n)$ ↗ O via ordered-file maintenance ↘ Ω [Bulánek, Koucký, Saks – SICOMP 2013]
$n^{1+\varepsilon}$ .. $n^{O(1)}$	$\Theta(\lg n)$ ↗ O via modified threshold: density $\le \frac{1}{\alpha^d}$, $1<\alpha\le 2$ ↘ Ω [Bulánek, Koucký, Saks – ICALP 2013]
2^n	$\Theta(1)$ – trivial

List order maintenance: easier problem, from L1

maintain linked list subject to insert/delete node here & order query: is node x before node y?

– $O(1)$ solution via indirection: [Dietz & Sleator – STOC 1987; Bender, Cole, Demaine, Farach-Colton, Zito – ESA 2002]



} $\Theta(\lg n)$ solution using label space $n^{\Theta(1)}$

} trivial $O(1)$ solution using exponential label space

– implicit node label = $\underbrace{\text{(top label, bottom label)}}_{O(\lg n) \text{ bits}}$

⇒ can compare two labels in $O(1)$ time

– top updates change many implicit labels at once (impossible in list labeling)

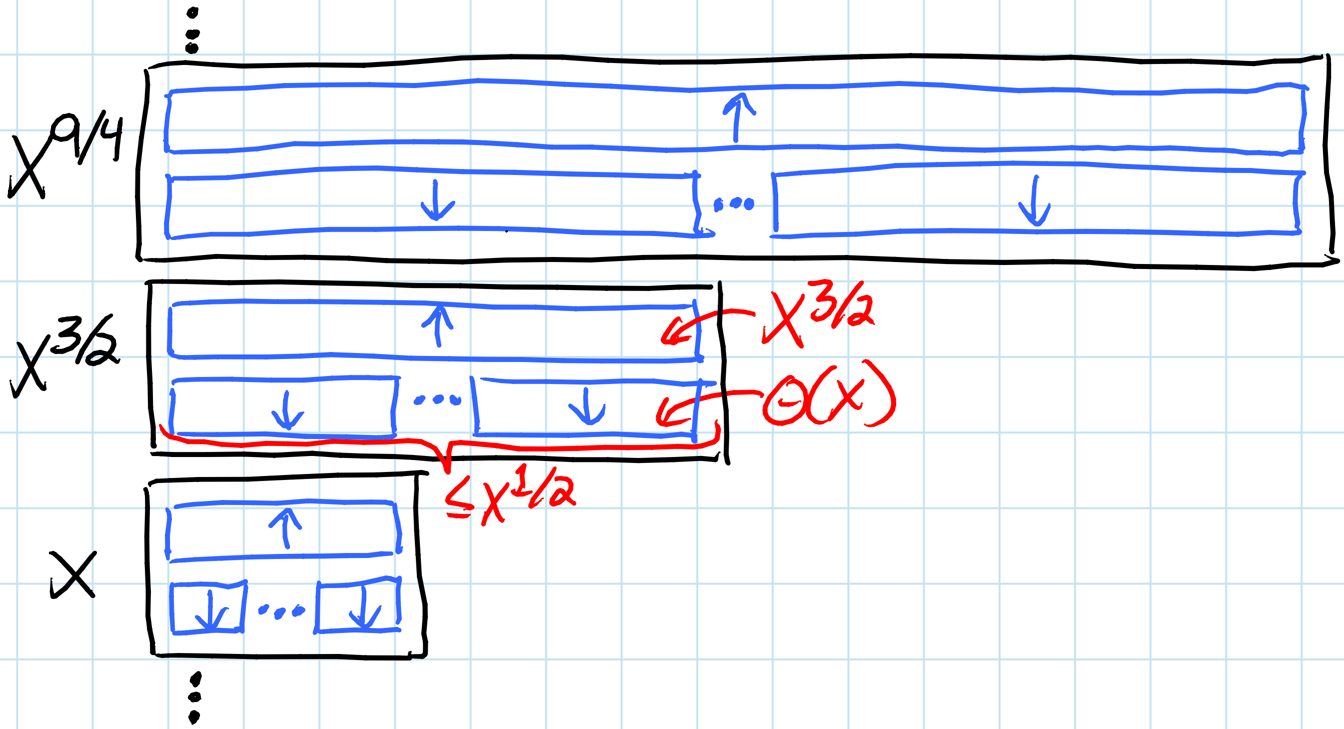
– bottom chunks slow top updates by $\Theta(\lg n)$ factor

⇒ $O(1)$ amortized cost

– worst-case bounds possible [same refs.]

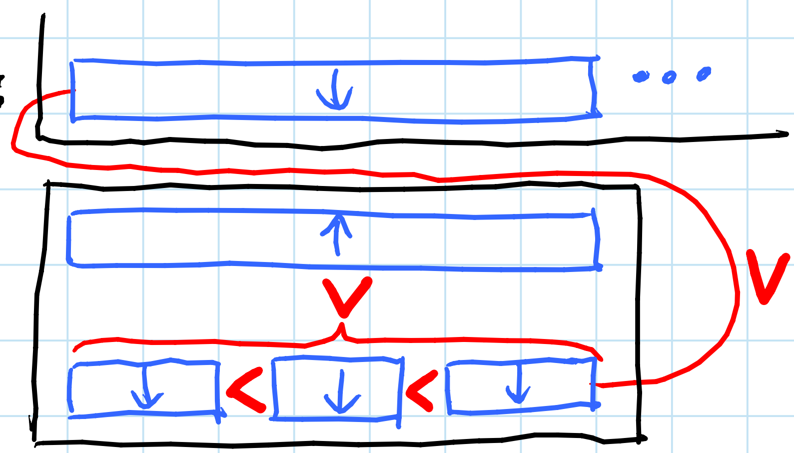
Cache-oblivious priority queue: (as in Arge et al. 2007)

- $\lg \lg n$ levels of size $N, N^{2/3}, N^{4/9}, \ldots, c = O(1)$
- level $X^{3/2}$ has 1 up buffer of size $X^{3/2}$ & $\le X^{1/2}$ 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:
1. append to bottom up buffer
2. swap into bottom down buffers if necessary
3. if up buffer overflows: push

Push X elements into level $X^{3/2}$
all > down buffers at level X & below
1. sort elements (see L9 for cache-obl. alg.)
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

exp. geometric geometric

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