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
  \[ \Rightarrow \text{one top piece, } \sqrt[2]{N} \text{ bottom pieces, each size } \approx \sqrt[2]{N} \]

\[ \begin{array}{c}
\frac{1}{2} \log N \\
\frac{1}{2} \log N
\end{array} \]

- recursively lay out pieces & concatenate

Example:
Analysis:
- level of detail (refinement) straddling $B$:

- cutting height in half until piece size $\leq B$
  $\Rightarrow$ height of piece between $\frac{1}{2} \log B$ & $\log B$ (slppy)
  $\Rightarrow$ size between $\sqrt{B}$ & $B$
  $\Rightarrow$ # pieces along root-to-leaf path $\leq \frac{\log N}{\frac{1}{2} \log B} = 2 \log_B N$
- each piece stores $\leq B$ elements consecutively
  $\Rightarrow$ occupies $\leq 2$ blocks (depending on alignment)
  $\Rightarrow$ # memory transfers $\leq 4 \log_B N$ (assuming $M \geq 2B$)
  (really should be $B+1$)

Improvements:
- BBFGHHIL03
  1. randomize starting location (w.r.t. block)
     $\Rightarrow$ expected cost $\leq (2 + \frac{3}{\sqrt{B}}) \log_B N$
  2. split height into $\frac{1}{2} - \varepsilon : \frac{1}{2} + \varepsilon$ ratio
     $\Rightarrow$ expected cost $\leq (\log e + o(1)) \log_B N$
     $= O(\log \log B / \log B)$
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 $O(l^2 N)$ am.

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

![VEB layout]

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 $O\left(\frac{K}{B} + \log_B N\right)$ mem.tr.
   - Look at level of detail straddling B.
   - Look at bottom two levels:

   ![Diagram of tree structure]

   - Within chunk of $\geq B$, jumping between $\geq 2$ pieces of $\leq B$ (assume $M \geq 2B$)
   - $O(\text{chunk }/B)$ memory transfers in chunk portion in update interval +3 maybe (first, last, & root)

   - $O\left(\frac{K}{B}\right)$ memory transfers in bottom 2 levels
   - Updated nodes above these two levels:
     - Subtree of $\leq \frac{K}{8}$ chunk roots up to their LCA: costs $O\left(\frac{K}{8}\right)$
     - Path from LCA to root of tree: costs $O\left(\log_B N\right)$ as above

   $\Rightarrow O\left(\frac{K}{8} + \log_B N\right)$ total memory transfers

   **So far:** search in $O\left(\log_B N\right)$
   update in $O\left(\log_B N + \frac{\log^2 N}{B}\right)$ amortized
5. Indirection:
   - Cluster elements into \( \Theta\left(\frac{N}{\log N}\right) \) groups, each of size \( \Theta(\log N) \).
   - Use previous structure on min's of clusters:
     \[ \Theta\left(\frac{N}{\log N}\right) \]
     \[ \Theta(\log N) \quad \Theta(\log N) \quad \cdots \quad \Theta(\log N) \]

   - Update cluster by complete rewrite:
     \[ \Rightarrow \Theta\left(\frac{N}{\log N}\right) \] memory transfers.
   - Split/merge clusters as necessary to keep between 25% & 100% full:
     \[ \Rightarrow \Omega(\log N) \] updates to charge to.
   - \[ \Rightarrow \Theta\left(\frac{N}{\log \log N}\right) \] update cost in top structure.
   - Only "every" \( \Omega(\log N) \) actual updates.
   - Amortized update cost \( \Theta\left(\frac{N}{\log B}\right) \) (plus search cost).

Finally: \( \Theta(\log_B N) \) insert, delete, predecessor, successor, just like B-trees in external mem. (known B)
Variations:
- partially persistent \([\text{BCRO2}]\)
- scan support \([\text{BCDFCO2}]\)
- (suboptimal) update-query trade-off \([\text{BFFFKN07}]\)
- concurrent & lock-free \([\text{BFGK05}]\)
- implicit \([\text{FG03a}]\)
- worst-case \([\text{FG03b}]\)
Ordered-file maintenance: [Itai, Konheim, Rodeh 1981] [BDF05]

Idea: 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

In fact: 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
density(n node) = # elts. stored below node
# array slots in interval
is within threshold at its depth \( d \):
\[ \text{density} \geq \frac{1}{3} - \frac{1}{4} \frac{d}{h} \in \left[ \frac{1}{4}, \frac{1}{2} \right] \] (not too sparse)
\[ \text{density} \leq \frac{3}{4} + \frac{1}{4} \frac{d}{h} \in \left[ \frac{3}{4}, 1 \right] \] (not too dense)
3. evenly rebalance elements below nodes
within array interval of node

Analysis:
- thresholds get tighter as we go up
\[ \Rightarrow \text{rebalancing a node puts children} \]
far within their threshold:
\[ \left| \text{density} - \text{threshold} \right| \approx \frac{1}{4} \frac{1}{h} = \Theta \left( \frac{1}{\lg N} \right) \]
- this node won't be rebalanced again until
\[ \geq 1 \text{ child out of threshold} \]
\[ \Rightarrow \Omega \left( \frac{\text{capacity}}{\lg N} \right) \text{ updates to charge to} \]
\[ \Omega(1) \text{ because leaf = chunk has size } \Theta(\lg N) \]
\[ \Rightarrow O(\lg N) \text{ amortized rebuid cost} \]
to update element below a node
- each leaf is below \( h = \Theta(\lg N) \) ancestors
\[ \Rightarrow O(\lg^2 N) \text{ amortized cost per update} \]

Worst-case possible [Willard 1992; BCDFCZ02]
Linked list: [BCDFCO2]

delete/insert element between two given scan K consecutive elements
- for $O\left(\frac{K}{B}\right)$ worst-case scans, best known is $O\left(\frac{\log^2 N}{B}\right)$ update from ordered-file maintenance
- for $O\left(\frac{K}{B}\right) + [B^\varepsilon \text{ if } B^{1-\varepsilon} \leq K \leq B^{1+\varepsilon}]$ w.-c. scans, best known is $O\left(\frac{\log \log N}{B}\right)^{2+\varepsilon}$ am. update

- but with $O\left(\frac{K}{B}\right)$ 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 $O(D + \frac{K}{B})$ for D discontinuities & rewrites the K elements contiguously  
    $\Rightarrow$ add $\leq 2$ discontinuities & remove D  
    $\Rightarrow$ can charge $O(D)$ cost to $D-2$ decrease
Priority queue: \[\text{[ABD1M07; BFO20a]}\]
- \(\lg \lg n\) levels of size \(N, N^{3/4}, N^{4/9}, \ldots\), \(c = O(1)\)
- 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 \(\leq x^{3/2}\) < down buffers \(\leq x^{9/4}\)
- 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 buts. in order
   - when down buf. overflows, split in half & link
   - when #down buts. 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 buts. & 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
Analysis: push/pull at level $X^{3/2}$ sans recursion costs $O\left(\frac{X}{B} \log m_B \frac{X}{B}\right)$ memory transfers

- assume all levels of size $\leq M$ stay in cache
- tall cache assumption: $M \geq B^2$ (say)
- push at level $X^{3/2} = B^2 \Rightarrow X > B^{2/3} \Rightarrow \frac{X}{B} > 1$
  - sort costs $O\left(\frac{X}{B} \log m_B \frac{X}{B}\right)$ memory transfers
  - distribute costs $O\left(X^{1/2} + \frac{X}{B}\right)$ mem. transf.
    - startup per down buf. $\Rightarrow$ scan

- if $X > B^2$ then cost $= O\left(\frac{X}{B}\right)$
- else: only one such level: $B^{2/3} \leq X \leq B^2$
  - can keep 1 block per down buf. in cache:
    - $X \leq B^2 \Rightarrow X^{1/2} \leq B \leq \frac{M}{B}$ by tall cache
    - so just pay $O\left(\frac{X}{B}\right)$ at this level too

- pull at level $X^{3/2} \geq B^2$:
  - sort costs $O\left(\frac{X}{B} \log m_B \frac{X}{B}\right)$ memory transfers
  - another sort of $X^{3/2}$ elts. only when recursing $\Rightarrow$ charge to recursive pull

Total: each element goes up & then down
  (roughly—real proof harder)
  & costs $O\left(\frac{1}{B} \log m_B \frac{X}{B}\right)$ per push & pull @ $X$
  $\Rightarrow O\left(\frac{1}{B} \log m_B \frac{X}{B}\right)$ amortized cost per element
  $\Rightarrow$ geometric

$= O\left(\frac{1}{B} \log m_B \frac{X}{B}\right)$. 
Buffered repository tree: \([ABDHM07]\)
supports insert \((x)\) in \(O(\frac{1}{B} \log \frac{N}{B})\) memory trans. 
& extract-all-copies \((x)\) in \(O(\log V)\), 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: up buffer
    - split down buf. into \(<key, =key, >key\)
      left down buf. right down buf.
    \(\Rightarrow O(\#\text{down buf.} + \#\text{elts}/B)\) mem. trans.
      charge to past splits
  - pay for \(O(\log N)\) splits
(I think can also rebalance new keys in \(O(\log n)\))
OPEN: cache-oblivious “buffer trees”

- insert(x)
- delete(x)
- “delayed predecessor/successor(x)"
- "give me all answers" once @ end

in \( O(\frac{1}{b} \log \frac{N}{b}) \) amortized mem. trans./op.