TODAY: Memory Hierarchies I (of 3)

- external-memory model
- cache-oblivious model
- cache-oblivious B-trees

External memory/I/O/Disk Access Model:

[Aggarwal & Vitter - CACM 1988]

two-level memory hierarchy

- focus on # memory transfers:
  blocks read/written between cache & disk

  \[ \leq \text{RAM running time} \]

  \[ \geq \frac{\text{cell-probe LB}}{B} \]

- when can we save this factor of \( \geq B \)?
Basic results in external memory:

1. **Scanning**: $O\left(\frac{N}{B}\right)$ to read/write $N$ words in order
   - B-trees with branching factor $O(B)$
     - Support insert, delete, predecessor search in $O(\log_{B+1} N)$ memory transfers
     - $\Omega(\log_{B+1} N)$ for search in comparison model:
       - Where query fits among $N$ items requires $\log (N+1)$ bits of information
       - Each block read reveals where query fits among $B$ items $\Rightarrow \leq \log (B+1)$ bits of info.
       - $\Rightarrow$ need $\geq \frac{\log (N+1)}{\log (B+1)}$ memory transfers

   - Also optimal in “block-probe model” if $B \geq w$ [Patrascu & Thorup – see L11]

2. **Sorting**: $O\left(\frac{N}{B} \log_{wB} \frac{N}{B}\right)$ memory transfers
   $\Rightarrow B \times$ faster than B-tree sort!
   $\Omega$(ditto) in comparison model

3. **Permuting**: $O\left(\min\left\{ N, \frac{N}{B} \log_{wB} \frac{N}{B}\right\}\right)$
   $\Omega$(ditto) in indivisible model
   *Physical* execution ≤ Can’t pack pieces of input words in words

4. **Buffer tree**: $O\left(\frac{1}{B} \log_{wB} \frac{N}{B}\right)$ amortized mem. transf
   for delayed queries & batched updates & $O(\phi)$ find-min ($\Rightarrow$ priority queues)
Cache-oblivious model: [Frigo, Leiserson, \(6.046\)]

- like external-memory model
- but algorithm doesn't know \(B\) or \(M\) (!)
- \(\Rightarrow\) must work for all \(B\) & \(M\)
- automatic block transfers triggered by word access with offline optimal block replacement
  - FIFO, LRU, or any conservative replacement is 2-competitive given cache of \(2x\) size (resource augmentation)
- dropping \(M \geq M/2\) doesn't affect typical bounds e.g. sorting bound

Cool:
- clean model: algorithm just like RAM
- adapts to changing \(B\) (disk tracks & cache) & \(M\) (competing processes)
- OPEN: formalize this for \(B\)
- changing \(M\) formalized in [Bender, Ebrahimi, Fineman, Ghasemieh, Johnson, McCauley - SODA 2014; Bender, Demaine, Ebrahimi, Fineman, Johnson, Lincoln, Lynch, McCauley - SPAA 2016]
- adapts to all levels of multilevel memory hierarchy:
- often possible!
Basic cache-oblivious results:

1. **Scanning**: same algorithm & bound
   in $O(\log_{b+1} N)$ memory transfers
   
   [Bender, Demaine, Farach-Colton – FOCS 2000/SICOMP 2005]
   [Bender, Duon, Iacono, Wu – SODA 2002/JAAlg 2004]
   [Brodal, Fagerberg, Jacob – SODA 2002]
   - best constant is $\log_e$, not 1
   
   [Bender, Brodal, Fagerberg, Ge, He, Hu, Iacono, López-Ortiz – FOCS 2003]

2. **Sorting**: $O\left(\frac{N}{B} \log_{mb} \frac{N}{B}\right)$ memory transfers
   
   [Frigo et al. 1999; Brodal & Fagerberg – ICALP 2002]
   - uses tall-cache assumption: $M = \Omega(B^{1+\epsilon})$
   - impossible otherwise [Brodal & Fagerberg – STOC 2003]

3. **Permuting**: min impossible [Brodal & Fagerberg – same]

4. **Priority queue**: $O\left(\frac{1}{B} \log_{mb} \frac{N}{B}\right)$ amortized mem. transf.
   - uses tall-cache assumption

   [Arge, Bender, Demaine, Holland-Minkley, Munro – STOC 2002/SICOMP 2007; Brodal & Fagerberg – ISAAC 2002]
Cache-oblivious static search trees:

(binary search) [Prokop-MEng 1999]
- store \( N \) elements in \( N \)-node complete BST
- carve tree at middle level of edges
  \( \Rightarrow \) one top piece, \( \approx \sqrt{N} \) bottom pieces, each size \( \approx \sqrt{N} \)

\[ \lg N \uparrow \frac{1}{2} \lg N \downarrow \frac{1}{2} \lg N \]

- recursively lay out pieces & concatenate:
  \( \Rightarrow \) order to store nodes
  \( \Rightarrow \) generalizes to \[ \text{"van Emde Boas layout"} \] [Bender, Demaine, Farach-Colton 2000]
  - height not a power of 2
  - node degrees \( \geq 2 \) & \( O(1) \)
Analysis:
- level of detail (refinement) straddling B:
  \[
  \begin{array}{c}
  \text{cutting height in half until piece size } \leq B \\
  \Rightarrow \text{height of piece between } \frac{1}{2} \log B \text{ and } \log B \text{ (slppy)} \\
  \Rightarrow \text{size between } \sqrt{B} \text{ and } B \\
  \Rightarrow \text{pieces along root-to-leaf path } \leq \frac{\log N}{\frac{1}{2} \log B} = 2 \log_B N \\
  \Rightarrow \text{each piece stores } \leq B \text{ elements consecutively} \\
  \Rightarrow \text{occupies } \leq 2 \text{ blocks (depending on alignment)} \\
  \Rightarrow \text{memory transfers } \leq 4 \log_B N \text{ (assuming } M \geq 2B) \\
  \text{(really should be } B+1) \\
  \end{array}
  \]

Improvements: [BBFGHHIL 2003]
1. randomize starting location (w.r.t. block)
   \Rightarrow \text{expected cost } \leq (2 + \frac{3}{\sqrt{B}}) \log_B N
2. split height into \( \frac{1}{2} - \epsilon : \frac{1}{2} + \epsilon \) ratio
   \Rightarrow \text{expected cost } \leq (\log e + o(1)) \log_B N
   \approx O(\log \log B / \log B)
Cache-oblivious B-trees as in [Bender, Duan, Iacono, Wu]

1. ordered file maintenance: (to do in L8)
   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 $O(\log^2 N)$ am.

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

\[
\begin{array}{c}
\text{vEB layout}
\end{array}
\]

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
   - delete similar
4) update analysis:

- If \( K \) cells change in ordered file, then update tree in \( O(\frac{K}{B} + \log_B N) \) memory.

- Look at level of detail straddling \( B \).

- Look at bottom two levels:

  - Within chunk of \( \geq B \), jumping between \( \leq 2 \) pieces of \( \leq B \) (assume \( M \geq 4B \)).

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

  \[ O(\frac{K}{B}) \] 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 \( O(\frac{K}{8}) \)

    - Path from LCA to root of tree: costs \( O(\log_B N) \) as above

  \[ O(\frac{K}{8} + \log_B N) \] total memory transfers

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

- Update cluster by complete rewrite $\Rightarrow O(\frac{\log N}{\log_b N})$ memory transfers.
- Split/merge clusters as necessary to keep between 25% & 100% full $\Rightarrow \Omega(\log N)$ updates to charge to $\Rightarrow O(\frac{\log^2 N}{\log_b N})$ update cost in top structure only “every” $\Omega(\log N)$ actual updates.
- Amortized update cost $O(\frac{\log N}{\log_b N})$ (plus search cost).

**Finally:** $O(\log_b N)$ insert, delete, predecessor, successor, just like B-trees in external mem. (known B)