Today: Cache-oblivious algorithms II
- search: binary
  B-ary
  cache-oblivious
- sorting: mergesort
  cache-oblivious
- follow-on classes

Recall:
- external-memory model:
  - total size $M$
  - $B$ words/block
  - count # (block) memory transfers $MT(N)$

- cache-oblivious model:
  - algorithm doesn't know $B$ or $M$
  - automatic block loads & eviction of Least Recently Used (LRU) block
Why LRU block replacement strategy?

\[ \text{LRU}_m \leq 2 \cdot \text{OPT}_{m/2} \]

[Slater & Tarjan 1985] 

**RESOURCE AUGMENTATION** (changing \( M \))

Proof:
- partition block access sequence into maximal phases of \( M/B \) distinct blocks
- LRU spends \( \leq M/B \) memory transfers/phase
- OPT must spend \( \geq \frac{M}{a/B} \) memory transfers per phase: at best, starts phase with entire \( M/2 \) cache with needed items, but there are \( M/B \) blocks during phase, so \( \leq \) half free

**ONLINE ALGORITHMS** — comparing regular "online" algorithm (can't see the future) against offline/prescient optimal algorithm

- changing \( M \) by factor of 2 doesn't affect bounds like \( O(\frac{N^2}{B\sqrt{M}}) \)
Search: preprocess n elements in comparison model to support predecessor search for x.

1. B-trees support predecessor (insert & delete) in $O(\log_{B+1} N)$ memory transfers. 
   - each node occupies $\Theta(1)$ blocks
   - height $= \Theta(\log_B N)$
   - need to know B

Cache oblivious?
2. Binary search: divide & conquer is good, right?
   - different block until in \( x \)'s block
   \[ MT(N) = \Theta(lg N - lg B) = \Theta(lg N/B) \text{ slow} \]

3. Van Emde Boas layout: [Prokop 1999]
   - store \( N \) elements in complete BST
   - carve BST at middle level of edges:
     \[ \begin{array}{c}
     \uparrow \\
     \downarrow \\
     \frac{1}{2} lg N \\
     \frac{1}{2} lg N
     \end{array} \]
     middle level
   - recursively layout the pieces & concatenate:
     \[ \begin{array}{c}
     \uparrow \\
     \downarrow \\
     \frac{1}{2} lg N \\
     \frac{1}{2} lg N
     \end{array} \]
     top & bottom subtrees
   - like block matrix multiplication, order of pieces doesn't matter; just need each piece to be stored consecutively

Example:

Order in memory
Analysis of BST search in vEB layout:
- consider recursive level of refinement at which $\Delta$ has $\leq B$ nodes:

- $\Delta$ height is between $\frac{1}{3} \lg B$ & $\lg B$  
  (binary searching on height)
  $\Rightarrow$ size is between $\sqrt{B}$ & $B$
  $\Rightarrow$ any root-to-node path (search path) visits $\leq \frac{\lg N}{\frac{1}{3} \lg B} = 2 \log_B N \Delta$'s

- each $\Delta$ occupies $\leq 2$ memory blocks
  $\Rightarrow$ $\leq 4 \log_B N = O(\log_B N)$ memory transfers

- generalizes to height not a power of 2,
  B-trees of constant branching factor, &
  dynamic B-trees: $O(\log_B N)$ insert/del.

[Benep, Demaine, Farach-Colton 2008]
(see 6.851: Advanced DSs)
Sorting:

1. \( N \) inserts into (cache-oblivious) B-tree
   \[ \Rightarrow MT(N) = \Theta(N \log_B N) \quad \text{-- NOT OPTIMAL} \]
   - by contrast, BST sort is optimal \( O(N \log N) \)

2. (binary) mergesort is cache-oblivious
   - merge is 3 parallel scans
   \[ \Rightarrow MT(N) = 2 \cdot MT\left(\frac{N}{2}\right) + O\left(\frac{N}{B} + 1\right) \]
   \[ MT(M) = O\left(\frac{N}{B}\right) \]
   - recursion tree:
     \[ \frac{N}{B} \]
     \[ \frac{1}{2} \frac{N}{B} \]
     \[ \frac{1}{4} \frac{N}{B} \]
     \[ \vdots \]
     \[ O\left(\frac{N}{M}\right) \text{ leaves} \]
   \[ \Rightarrow MT(N) = \frac{N}{B} \log \frac{N}{M} \leq \frac{B}{\log B} \text{ faster than (1)!} \]

3. \( \frac{N}{B} \)-way mergesort: (vs. binary mergesort)
   - split array into \( \frac{N}{B} \) equal subarrays
   - recursively sort each (contiguous)
   - merge via \( \frac{N}{B} \) parallel scans
     (keeping one “current” block per list)
\[ MT(N) = \frac{M}{B} \cdot MT\left(\frac{N}{\frac{M}{B}}\right) + O\left(\frac{N}{B} + 1\right) \]

\[ MT(M) = O\left(\frac{N}{B}\right) \]

\[ \Rightarrow \text{height becomes } \log_{\frac{M}{B}} \frac{N}{M} + 1 \]
\[ = \log_{\frac{M}{B}} \frac{N}{B} \cdot \frac{B}{M} + 1 \]
\[ = \log_{\frac{M}{B}} \frac{N}{B} - \log_{\frac{M}{B}} \frac{M}{B} + 1 \]
\[ = \log_{\frac{M}{B}} \frac{N}{B} \]

\[ \Rightarrow MT(N) = O\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right) \leftarrow \text{asymptotically optimal} \]

(in comparison model)

4) cache-oblivious sorting requires

tall-cache assumption:
\[ M = \Omega(B^{1+\varepsilon}) \] for some fixed \( \varepsilon > 0 \)

e.g. \( M = \Omega(B^2) \) i.e. \( \frac{M}{B} = \Omega(B) \)

- then \( \sim N^\varepsilon \)-way mergesort with recursive ("funnel") merge works

5) priority queues:
\[ O\left(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B}\right) \]
per insert or delete-min

\[ \Rightarrow \text{generalizes sorting} \]
- external memory & cache oblivious!
- see 6.851
Algorithms classes at MIT: (post-6.046)
- 6.047: Computational Biology (genomes, phylogeny, etc.)
- 6.854: Advanced Algorithms (intense survey of whole field)
- 6.850: Geometric Computing (working with points, lines, polygons, meshes, ...)
- 6.849: Geometric Folding Algorithms (origami, robot arms, protein folding, ...)
- 6.851: Advanced Data Structures (sublogarithmic performance)
- 6.852: Distributed Algorithms (reaching consensus in a network with faults)
- 6.853: Algorithmic Game Theory (Nash equilibria, auction mechanism design, ...)
- 6.855: Network Optimization (optimization in graph: beyond shortest paths)
- 6.856: Randomized Algorithms (how randomness makes algns. simpler & faster)
- 6.857: Network and Computer Security (applied cryptography)
- 6.875: Cryptography and Cryptanalysis (theoretical cryptography)
- 6.816: Multicore Programming
Other theory classes:

- 6.045: Automata, Computability, & Complexity
- 6.840: Theory of Computing
- 6.841: Advanced Complexity Theory
- 6.842: Randomness & Computation
- 6.845: Quantum Complexity Theory
- 6.440: Essential Coding Theory
- 6.441: Information Theory

— Frisbee Competition —