Random Access Machine Model

- Standard theoretical model of computation:
  - Infinite memory
  - Uniform access cost
Hierarchical Memory

- Modern machines have complicated memory hierarchy
  - Levels get larger and slower further away from CPU
  - Large access time amortized using block transfer between levels

- Bottleneck often transfers between largest memory levels in use
I/O-Bottleneck

- I/O is often bottleneck when handling massive datasets
  - Disk access is $10^6$ times slower than main memory access
  - Large transfer block size (typically 8-16 Kbytes)

- Important to obtain “locality of reference”
  - Need to store and access data to take advantage of blocks
I/O-algorithms

I/O-Model

- Parameters
  \[ N = \text{# elements in problem instance} \]
  \[ B = \text{# elements that fits in disk block} \]
  \[ M = \text{# elements that fits in main memory} \]
  \[ T = \text{# output size in searching problem} \]

- We often assume that \( M > B^2 \)

- I/O: Movement of block between memory and disk
### Fundamental Bounds

<table>
<thead>
<tr>
<th></th>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning</td>
<td>$N$</td>
<td></td>
</tr>
<tr>
<td>Sorting</td>
<td>$N \log N$</td>
<td>$\frac{N}{B} \log_{M/B} \frac{N}{B}$</td>
</tr>
<tr>
<td>Permuting</td>
<td>$N$</td>
<td>$\min{N, \frac{N}{B} \log_{M/B} \frac{N}{B}}$</td>
</tr>
<tr>
<td>Searching</td>
<td>$\log_2 N$</td>
<td>$\log_B N$</td>
</tr>
</tbody>
</table>

**Note:**
- Linear I/O: $O(N/B)$
- Permuting not linear
- Permuting and sorting bounds are equal in all practical cases
- $B$ factor VERY important: $\frac{N}{B} < \frac{N}{B} \log_{M/B} \frac{N}{B} \ll N$
- Cannot sort optimally with search tree
Merge Sort

- **Merge sort:**
  - Create $N/M$ memory sized sorted runs
  - Merge runs together $M/B$ at a time

$\Rightarrow O(\log_{M/B} \frac{N}{M})$ phases using $O(\frac{N}{B})$ I/Os each

- **Distribution sort similar (but harder – partition elements)**
Permuting Lower Bound

Permuting $N$ elements according to a given permutation takes $\Omega\left(\min\{N, \frac{N}{B} \log \frac{M}{B}, \frac{N}{B}\}\right)$ I/Os in “indivisibility” model.

- Indivisibility model: Move of elements only allowed operation
- Note:
  - We can allow copies (and destruction of elements)
  - Bound also a lower bound on sorting

- Proof:
  - View memory and disk as array of $N$ tracks of $B$ elements
  - Assume all I/Os track aligned (assumption can be removed)
Permuting Lower Bound

- Array contains permutation of $N$ elements at all times
- We will count how many permutations can be reached (produced) with $t$ I/Os

- Input:
  - Choose track: $N$ possibilities
  - Rearrange $\leq B$ element in track and place among $\leq M-B$ elements in memory:
    - $\leq B! \cdot \binom{M}{B}$ possibilities if “fresh” track
    - $\leq \binom{M}{B}$ otherwise
    $\Rightarrow$ at most $(N \cdot \binom{M}{B})^t \cdot (B!)^{\frac{N}{B}}$ permutations after $t$ inputs

- Output:
  - Choose track: $N$ possibilities
Permuting Lower Bound

- Permutation algorithm needs to be able to produce $N!$ permutations

\[
(N \cdot \binom{M}{B})^t \cdot (B!)^{N/B} \geq N!
\]

\[
\frac{N}{B} \log(B!) + t(\log N + \log(M/B)) \geq \log(N!)
\]

\[
N \log B + t(\log N + B \log \frac{M}{B}) \geq N \log N
\]

\[
t \geq \frac{N \log \frac{N}{B}}{\log N + B \log \frac{M}{B}}
\]

(Using Stirlings formula \(\log x! \approx x \log x\) and \(\log \left(\binom{M}{B}\right) \approx B \log \frac{M}{B}\))

- If \(\log N \leq B \log \frac{M}{B}\) we have \(t \geq \frac{N \log \frac{N}{B}}{2B \log \frac{M}{B}} = \Omega\left(\frac{N}{B} \log \frac{M}{B} \frac{N}{B}\right)\)

- If \(\log N > B \log \frac{M}{B}\) we have \(B << \sqrt{N}\) and thus \(t \geq \frac{N \log \frac{N}{B}}{2 \log N} = \frac{1}{2} \left(N - N \frac{\log B}{\log N}\right) \geq \frac{1}{2} \left(N - \frac{1}{2} N\right) = \Omega(N)\)

\[
\Rightarrow t = \Omega\left(\min\{N, \frac{N}{B} \log \frac{M}{B} \frac{N}{B}\}\right)
\]
Sorting lower bound

Sorting $N$ elements takes $\Omega\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}\right)$ I/Os in comparison model

**Proof:**
- Initially $N$ elements stored in $N/B$ first blocks on disk
- Initially all $N!$ possible orderings consistent with our knowledge
- After $t$ I/Os?
Sorting lower bound

- Consider one input assuming:
  - \( S \) consistent orderings before input
  - Compute total order of elements in memory
  - Adversary choose ”worst” outcome of comparisons done
- \( \leq \binom{M}{B} \cdot B! \) possible orderings of \( M-B \) ”old” and \( B \) new elements in memory
- Adversary can choose outcome such that still \( \geq S / (\binom{M}{B} \cdot B!) \) consistent orderings
- Only get \( B! \) term \( N/B \) times

\[
\geq \frac{N!}{\left(\binom{M}{B}^t \cdot (B!)^{N/B}\right)} \text{ consistent orderings after } t \text{ I/Os}
\Rightarrow \frac{N!}{\left(\binom{M}{B}^t \cdot (B!)^{N/B}\right)} = 1 \Rightarrow t = \Omega\left(\frac{N}{B} \log_{\binom{M}{B}} \frac{N}{B}\right)
\]
Summary/Conclusion: Sorting

- External merge or distribution sort takes $O\left(\frac{N}{B} \log^{\frac{M/B}{N/B}} \frac{N}{B}\right)$ I/Os
  - Merge-sort based on $M/B$-way merging
  - Distribution sort based on $\sqrt{M/B}$-way distribution and partition elements finding

- Optimal in comparison model

- Can prove $\Omega\left(\min\{N, \frac{N}{B} \log^{M/B} \frac{N}{B}\}\right)$ lower bound in stronger model
  - Holds even for permuting
External Search Trees

- Binary search tree:
  - Standard method for search among $N$ elements
  - We assume elements in leaves

- Search traces at least one root-leaf path
- If nodes stored arbitrarily on disk
  $\Rightarrow$ Search in $O(\log_2 N)$ I/Os
  $\Rightarrow$ Rangesearch in $O(\log_2 N + T)$ I/Os
External Search Trees

- **BFS blocking:**
  - Block height $O(\log_2 N) / O(\log_2 B) = O(\log_B N)$
  - Output elements blocked
  - \[ \downarrow \]
    - Rangesearch in $O(\log_B N + \frac{T}{B})$ I/Os

- **Optimal:** $O(N/B)$ space and $O(\log_B N + \frac{T}{B})$ query
External Search Trees

- Maintaining BFS blocking during updates?
  - Balance normally maintained in search trees using rotations

- Seems very difficult to maintain BFS blocking during rotation
  - Also need to make sure output (leaves) is blocked!
B-trees

- BFS-blocking naturally corresponds to tree with fan-out $\Theta(B)$

- B-trees balanced by allowing node degree to vary
  - Rebalancing performed by splitting and merging nodes
(a,b)-tree

• $T$ is an $(a,b)$-tree ($a \geq 2$ and $b \geq 2a-1$)
  – All leaves on the same level and contain between $a$ and $b$ elements
  – Except for the root, all nodes have degree between $a$ and $b$
  – Root has degree between 2 and $b$

• $(a,b)$-tree uses linear space and has height $O(\log_a N)$
  \[ \downarrow \]
  Choosing $a, b = \Theta(B)$ each node/leaf stored in one disk block
  \[ \downarrow \]
  $O(N/B)$ space and $O(\log_B N + T/B)$ query
(a,b)-Tree Insert

• **Insert:**

Search and insert element in leaf $v$

DO $v$ has $b+1$ elements/children

**Split** $v$:

- make nodes $v'$ and $v''$ with
  - $\left\lceil \frac{b+1}{2} \right\rceil \leq b$ and $\left\lfloor \frac{b+1}{2} \right\rfloor \geq a$ elements
  - insert element (ref) in $parent(v)$
  - (make new root if necessary)

$v = parent(v)$

• **Insert touch** $O(\log_a N)$ nodes
(2,4)-Tree Insert
(a,b)-Tree Delete

- Delete:

  Search and delete element from leaf \( v \)
  DO \( v \) has \( a-1 \) elements/children
  
  **Fuse** \( v \) with sibling \( v' \):
  move children of \( v' \) to \( v \)
  delete element (ref) from \( \text{parent}(v) \)
  (delete root if necessary)
  
  If \( v \) has \( >b \) (and \( \leq a+b-1 < 2b \)) children split \( v \)
  \( v = \text{parent}(v) \)

- Delete touch \( O(\log_a N) \) nodes
(2,4)-Tree Delete
\((a,b)\)-tree properties:

- If \(b=2a-1\) every update can cause many rebalancing operations.
- If \(b \geq 2a\) update only cause \(O(1)\) rebalancing operations amortized.
- If \(b > 2a\) only \(O\left(\frac{1}{b-2a}\right) = O\left(\frac{1}{a}\right)\) rebalancing operations amortized.
  
  * Both somewhat hard to show.
- If \(b = 4a\) easy to show that update causes \(O\left(\frac{1}{a} \log_a N\right)\) rebalance operations amortized.
  
  * After split during insert a leaf contains \(\cong 4a/2 = 2a\) elements.
  * After fuse during delete a leaf contains between \(\cong 2a\) and \(\cong 5a\) elements (split if more than \(3a\) \(\Rightarrow\) between \(3/2a\) and \(5/2a\)).
Summary/Conclusion: B-tree

- **B-trees:** \((a,b)\)-trees with \(a,b = \Theta(B)\)
  - \(O(N/B)\) space
  - \(O(\log_B N + T/B)\) query
  - \(O(\log_B N)\) update

- B-trees with **elements in the leaves** sometimes called \(B^+\)-tree

- Construction in \(O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)\) I/Os
  - Sort elements and construct leaves
  - Build tree level-by-level bottom-up
Summary/Conclusion: B-tree

• B-tree with branching parameter $b$ and leaf parameter $k$ ($b,k \geq 8$)
  – All leaves on same level and contain between $1/4k$ and $k$ elements
  – Except for the root, all nodes have degree between $1/4b$ and $b$
  – Root has degree between 2 and $b$

• B-tree with leaf parameter $k = \Omega(B)$
  – $O(N/B)$ space
  – Height $O\left(\log_b \frac{N}{B}\right)$
  – $O\left(\frac{1}{k}\right)$ amortized leaf rebalance operations
  – $O\left(\frac{1}{b \cdot k} \log_b \frac{N}{B}\right)$ amortized internal node rebalance operations

• B-tree with branching parameter $B^c$, $0 < c \leq 1$, and leaf parameter $B$
  – Space $O(N/B)$, updates $O\left(\log_B N\right)$, queries $O\left(\log_B N + \frac{T}{B}\right)$
Secondary Structures

- When secondary structures used, a rebalance on $v$ often requires $O(w(v))$ I/Os ($w(v)$ is weight of $v$)
  - If $\Omega(w(v))$ inserts have to be made below $v$ between operations
    $\Rightarrow O(1)$ amortized split bound
    $\Rightarrow O(\log_B N)$ amortized insert bound

- Nodes in standard B-tree do not have this property
I/O-algorithms

**BB[α]-tree**

- In internal memory BB[α]-trees have the desired property
- Defined using weight-constraint
  - Ratio between weight of left child and weight of right child of a node \( v \) is between \( \alpha \) and \( 1-\alpha \) \((\alpha<1)\)
  \[\downarrow\]
  Height \( O(\log N) \)
- If \( 2^{1/11} < \alpha < 1 - \frac{1}{2} \sqrt{2} \) rebalancing can be performed using rotations

- Seems hard to implement BB[α]-trees I/O-efficiently
Weight-balanced B-tree

**Idea:** Combination of B-tree and BB$[\alpha]$-tree
- Weight constraint on nodes instead of degree constraint
- Rebalancing performed using split/fuse as in B-tree

**Weight-balanced B-tree** with parameters $b$ and $k$ ($b>8$, $k\geq8$)
- All leaves on same level and contain between $k/4$ and $k$ elements
- Internal node $v$ at level $l$ has $w(v) < b^l k$
- Except for the root, internal node $v$ at level $l$ has $w(v) > \frac{1}{4} b^l k$
- The root has more than one child
Weight-balanced B-tree

- Every internal node has degree between
  \[ \frac{1}{4} b^l k / b^{l-1} k = \frac{1}{4} b \quad \text{and} \quad b^l k / \frac{1}{4} b^{l-1} k = 4b \]

\[ \downarrow \]

Height \( O(\log_b \frac{N}{k}) \)

- External memory:
  - Choose \( 4b = B \) (or even \( B^c \) for \( 0 < c \leq 1 \))
  - \( k = B \)

\[ \downarrow \]

\( O(N/B) \) space, \( O(\log_B N + T/B) \) query
Weight-balanced B-tree Insert

- Search for relevant leaf $u$ and insert new element
- Traverse path from $u$ to root:
  - If level $l$ node $v$ now has $w(v) = b^l k + 1$
    then split into nodes $v'$ and $v''$ with
    
    $w(v') \geq \left\lceil \frac{1}{2} (b^l k + 1) \right\rceil - b^{l-1} k$ and
    $w(v'') \leq \left\lfloor \frac{1}{2} (b^l k + 1) \right\rfloor + b^{l-1} k$

- Algorithm correct since $b^{l-1} k \leq \frac{1}{8} b^l k$
  such that $w(v') \geq \frac{3}{8} b^l k$ and $w(v'') \leq \frac{5}{8} b^l k$
  - touch $O(\log_b \frac{N}{k})$ nodes
- Weight-balance property:
  - $\Omega(b^l k)$ updates below $v'$ and $v''$ before next rebalance operation
Weight-balanced B-tree Delete

- Search for relevant leaf \( u \) and delete element
- Traverse path from \( u \) to root:
  - If level \( l \) node \( v \) now has \( w(v) = \frac{1}{4} b^l k - 1 \)
    then fuse with sibling into node \( v' \)
    with \( \frac{2}{4} b^l k - 1 \leq w(v') \leq \frac{5}{4} b^l k - 1 \)
  - If now \( w(v') \geq \frac{7}{8} b^l k \) then split into nodes
    with weight \( \geq \frac{7}{16} b^l k - 1 - b^{l-1} k \geq \frac{5}{16} b^l k - 1 \)
    and \( \leq \frac{5}{8} b^l k + b^{l-1} k \leq \frac{6}{8} b^l k \)

- Algorithm correct and touch \( O(\log_b \frac{N}{k}) \) nodes
- Weight-balance property:
  - \( \Omega(b^l k) \) updates below \( v' \) and \( v'' \) before next rebalance operation
Summary/Conclusion: Weight-balanced B-tree

- Weight-balanced B-tree with branching parameter $b$ and leaf parameter $k = \Omega(B)$
  - $O(N/B)$ space
  - Height $O(\log_b \frac{N}{k})$
  - $O(\log_b N)$ rebalancing operations after update
  - $\Omega(w(v))$ updates below $v$ between consecutive operations on $v$

- Weight-balanced B-tree with branching parameter $B^c$ and leaf parameter $B$
  - Updates in $O(\log_B N)$ and queries in $O(\log_B N + T/B)$ I/Os

- Construction bottom-up in $O(\frac{N}{B} \log \frac{M_B}{B} \frac{N}{B})$ I/O
References

• **Lower bound on External Permuting/Sorting**
  Lecture notes by L. Arge.

• **External Memory Geometric Data Structures**
  Lecture notes by Lars Arge.
  – Section 1-3