**Massive Data**

- Pervasive use of computers and sensors
- Increased ability to acquire, store and process data

→ Massive data collected everywhere

**Examples (2002):**

- **Phone:** AT&T 20TB phone call database, wireless tracking
- **Consumer:** WalMart 70TB database, buying patterns
- **WEB/Network:** Google index $8 \times 10^9$ pages, internet routers
- **Geography:** NASA satellites generate TB each day
Example: Grid Terrain Data

- Appalachian Mountains (800km x 800km)
  - 100m resolution ⇒ ~ 64M cells
    ⇒ ~128MB raw data (~500MB when processing)
  - ~ 1.2GB at 30m resolution
    NASA SRTM mission acquired 30m data for 80% of the earth land mass
  - ~ 12GB at 10m resolution (much of US available from USGS)
  - ~ 1.2TB at 1m resolution (quickly becoming available)
Example: LIDAR Terrain Data

• Massive (irregular) point sets (~1m resolution)
  – Becoming relatively cheap and easy to collect
Example: LIDAR Terrain Data

- COWI A/S (and others) have scanned Denmark
Example: LIDAR Terrain Data

- ~2 million points at 30 meter (<1GB)
- ~18 billion points at 1 meter (>1TB)
Application Example: Flooding Prediction

+1 meter
+2 meter
Random Access Machine Model

- Standard theoretical model of computation:
  - Infinite memory
  - Uniform access cost
- Simple model crucial for success of computer industry
Hierarchical Memory

Modern machines have complicated memory hierarchy
- Levels get larger and slower further away from CPU
- Data moved between levels using large blocks
Slow I/O

- Disk access is $10^6$ times slower than main memory access

Disk systems try to amortize large access time transferring large contiguous blocks of data (8-16Kbytes)

- Important to store/access data to take advantage of blocks (locality)

"The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on one’s desk or by taking an airplane to the other side of the world and using a sharpener on someone else’s desk." (D. Comer)
Scalability Problems

- Most programs developed in RAM-model
  - Run on large datasets because OS moves blocks as needed

- Moderns OS utilizes sophisticated paging and prefetching strategies
  - But if program makes scattered accesses even good OS cannot take advantage of block access

↓

Scalability problems!
We assume (for convenience) that $M > B^2$
### Fundamental Bounds

<table>
<thead>
<tr>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanning: $N$</td>
<td>$\frac{N}{B}$</td>
</tr>
<tr>
<td>Sorting: $N \log N$</td>
<td>$\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}$</td>
</tr>
<tr>
<td>Permuting: $N$</td>
<td>$\min{N, \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}}$</td>
</tr>
<tr>
<td>Searching: $\log_2 N$</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: $\frac{N}{B} < \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B} << N$
- Cannot sort optimally with search tree
Scalability Problems: Block Access Matters

- Example: Traversing linked list (List ranking)
  - Array size $N = 10$ elements
  - Disk block size $B = 2$ elements
  - Main memory size $M = 4$ elements (2 blocks)

- Large difference between $N$ and $N/B$ large since block size is large
  - Example: $N = 256 \times 10^6$, $B = 8000$, 1ms disk access time
    $$N \text{ I/Os take } 256 \times 10^3 \text{ sec} = 4266 \text{ min} = 71 \text{ hr}$$
    $$N/B \text{ I/Os take } 256/8 \text{ sec} = 32 \text{ sec}$$
Queues and Stacks

- **Queue**: Maintain push and pop blocks in main memory

  Push → [Diagram showing push and pop] → Pop

  \[ O(1/B) \] Push/Pop operations

- **Stack**: Maintain push/pop blocks in main memory

  [Diagram showing push and pop] ↔

  \[ O(1/B) \] Push/Pop operations
Sorting

- \(<M/B\) sorted lists (queues) can be merged in \(O(N/B)\) I/Os

- Unsorted list (queue) can be distributed using \(<M/B\) split elements in \(O(N/B)\) I/Os
I/O-Algorithms

Sorting

- Merge sort:
  - Create $N/M$ memory sized sorted lists
  - Repeatedly merge lists together $\Theta(M/B)$ at a time

\[ \Rightarrow O\left(\log_{M/B} \frac{N}{M}\right) \text{ phases using } O\left(\frac{N}{B}\right) \text{ I/Os each } \Rightarrow O\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right) \text{ I/Os} \]
Sorting

- **Distribution sort** (multiway quicksort):
  - Compute $\Theta(M/B)$ splitting elements
  - Distribute unsorted list into $\Theta(M/B)$ unsorted lists of equal size
  - Recursively split lists until fit in memory

$$\Rightarrow O(\log_{M/B} \frac{N}{M}) \text{ phases}$$

$$\Rightarrow O(\frac{N}{B} \log_{M/B} \frac{N}{B}) \text{ I/Os if splitting elements computed in } O(N/B) \text{ I/Os}$$
Computing Splitting Elements

- In internal memory (deterministic) quicksort split element (median) found using linear time selection

- **Selection algorithm**: Finding $i$’th element in sorted order
  1) Select median of every group of 5 elements
  2) Recursively select median of $\sim N/5$ selected elements
  3) Distribute elements into two lists using computed median
  4) Recursively select in one of two lists

- **Analysis**:
  - Step 1 and 3 performed in $O(N/B)$ I/Os.
  - Step 4 recursion on at most $\sim \frac{7}{10} N$ elements

\[ T(N) = O\left(\frac{N}{B}\right) + T\left(\frac{N}{5}\right) + T\left(\frac{7N}{10}\right) = O\left(\frac{N}{B}\right) \text{ I/Os} \]
Sorting

• Distribution sort (multiway quicksort):

  – $\Theta(M/B)$ times linear I/O selection $\Rightarrow O(NM/B^2)$ I/O algorithm
  – But can use selection algorithm to compute $\sqrt{M/B}$ splitting elements in $O(N/B)$ I/Os, partitioning into lists of size $\leq \frac{3N}{2\sqrt{M/B}}$
  $\Rightarrow O(\log\left(\frac{N}{\sqrt{M/B}}\right)) = O(\log_{M/B}\frac{N}{M})$ phases $\Rightarrow O\left(\frac{N}{B} \log_{M/B}\frac{N}{B}\right)$ algorithm
Computing Splitting Elements

1) Sample \( \frac{4N}{\sqrt{M/B}} \) elements:
   - Create \( N/M \) memory sized sorted lists
   - Pick every \( \frac{1}{4} \sqrt{M/B} \)’th element from each sorted list
2) Choose \( \sqrt{M/B} \) split elements from sample:
   - Use selection algorithm \( \sqrt{M/B} \) times to find every \( \frac{4N}{\sqrt{M/B}} \)’th element

• Analysis:
  - Step 1 performed in \( O(N/B) \) I/Os
  - Step 2 performed in \( \sqrt{M/B} \cdot O\left(\frac{N}{\sqrt{M/B}}\right) = O\left(\frac{N}{B}\right) \) I/Os
  \Rightarrow O(N/B) \) I/Os
Computing Splitting Elements

1) Sample $\frac{4N}{\sqrt{M/B}}$ elements:
   - Create $N/M$ memory sized sorted lists
   - Pick every $\frac{1}{4} \sqrt{M/B}$'th element from each sorted list

2) Choose $\sqrt{M/B}$ split elements from sample:
   - Use selection algorithm $\sqrt{M/B}$ times to find every $\frac{4N}{M/B}$'th element
Computing Splitting Elements

- Elements in range $R$ defined by consecutive split elements
  - Sampled elements in $R$: $\frac{4N}{M_B} - 1$
  - Between sampled elements in $R$: $(\frac{4N}{M_B} - 1) \cdot (\frac{1}{4} \sqrt{\frac{M}{B}} - 1)$
  - Between sampled element in $R$ and outside $R$: $2 \frac{N}{M} \cdot (\frac{1}{4} \sqrt{\frac{M}{B}} - 1)$

\[ \Rightarrow \quad \left< \frac{4N}{M_B} + \left( \frac{N}{\sqrt{\frac{M}{B}}} - \frac{4N}{M_B} \right) \right> + \frac{N}{2B \sqrt{\frac{M}{B}}} \quad \left< \frac{3}{2} \frac{N}{\sqrt{\frac{M}{B}}} \right> \]
Project 1: Implementation of Merge Sort
Summary/Conclusion: Sorting

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

- Optimal
  - As we will prove next time
I/O-Algorithms

References

• Input/Output Complexity of Sorting and Related Problems
  A. Aggarwal and J.S. Vitter. *CACM* 31(9), 1998

• External partition element finding
  Lecture notes by L. Arge and M. G. Lagoudakis.