External sorting

R & G – Chapter 13

Brian Cooper
Yahoo! Research
A little bit about Y!

- Yahoo! is the most visited website in the world
  - Sorry Google
  - 500 million unique visitors per month
  - 74 percent of U.S. users use Y! (per month)
  - 13 percent of U.S. users’ online time is on Y!
## Why sort?

<table>
<thead>
<tr>
<th>Business Name</th>
<th>Address</th>
<th>Miles**</th>
</tr>
</thead>
<tbody>
<tr>
<td>King Pin Doughnuts</td>
<td>2521 Durant Ave # A</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Noah's Bagels</td>
<td>2344 Telegraph Ave</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Dream Fluff Donuts</td>
<td>2637 Ashby Ave</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Noah's Bagels</td>
<td>3170 College Ave</td>
<td>1.4</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>All Star Donut</td>
<td>1255 University Ave</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Noah's Bagels</td>
<td>1883 Solano Ave</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Boogie Woogie Bagel Boy</td>
<td>1281 Gilman St</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>Albany, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Boogie Woogie Bagel Boy</td>
<td>1218 Santa Fe Ave</td>
<td>1.8</td>
</tr>
<tr>
<td></td>
<td>Albany, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Berkeley Donut Shop</td>
<td>3043 San Pablo Ave</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
<tr>
<td>Happy Donuts</td>
<td>1041 Gilman St</td>
<td>2.1</td>
</tr>
<tr>
<td></td>
<td>Berkeley, CA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Map</td>
<td></td>
</tr>
</tbody>
</table>
"toy" > Toys & Games

Showing 1 - 24 of 260,516 Results  |  Page: 1 2 3 4 | Next » Sort by: Price: High to Low

1. **Steiff Germany: Giant Studio Elephant: Overall Size ~ 210cm high (82.68")**
   - **Buy new:** $22,000.00 $16,000.00
   - Usually ships in 3 to 5 weeks

2. **Miss Megan Modular Playground 3.5 Inch Posts**
   - **Buy new:** $12,922.00
   - Usually ships in 2 to 3 weeks
   > Show only SportsPlay items

3. **Meade LX200 GPS 16 in. UHTC SCT with Super Field Tripod**
   - **Buy new:** $10,988.71
   - In Stock

4. **Apollo 17 Astronaut Space Suit Replica**
   - Currently unavailable
   > 4 Stars

5. **Meade 14" f/8 RCX Advanced Ritchey-Chretien Telescope, with UHTC; Tri pod - 1408-40-01**
   - **Buy new:** $11,240.00 $9,599.99
   - 2 Used & new from $9,593.71
   - In Stock
   > Show only MEA items

6. **Lizard Thumb Piece Entry Way Lock Set - ETS241B - Thumbgrip Handlesets**
   - Currently unavailable
Why sort?

- Users usually want data sorted
- Sorting is first step in bulk-loading a B+ tree
- Sorting useful for eliminating duplicates
- Sort-merge join algorithm involves sorting

Blueberry
Strawberry
Kiwi
Mango
Orange
Apple
Grapefruit
Banana

Apple
Banana
Blueberry
Grapefruit
Kiwi
Mango
Orange
Strawberry
So?

- Don’t we know how to sort?
  - Quicksort
  - Mergesort
  - Heapsort
  - Selection sort
  - Insertion sort
  - Radix sort
  - Bubble sort
  - Etc.

- Why don’t these work for databases?
Key problem in database sorting

- How to sort data that does not fit in memory?

4 GB: $300

480 GB: $300
Example: merge sort
Example: merge sort

Banana
Grapefruit

Apple
Orange

Kiwi
Mango

Blueberry
Strawberry

Blueberry
Kiwi
Mango
Strawberry

Apple
Banana
Grapefruit

Kiwi
Mango
Orange
Strawberry
Isn’t that good enough?

- Consider a file with $N$ records
- Merge sort is $O(N \lg N)$ comparisons
- We want to minimize disk I/Os
  - Don’t want to go to disk $O(N \lg N)$ times!
- Key insight: sort based on pages, not records
  - Read whole pages into RAM, not individual records
  - Do some in-memory processing
  - Write processed blocks out to disk
  - Repeat
2-way sort

- Pass 0: sort each page
- Pass 1: merge two pages into one run
- Pass 2: merge two runs into one run
- ...
- Sorted!
What did that cost us?

- P pages in the file
- Each pass: read and wrote P pages
- How many passes?
  - Pass 0
  - Pass 1: went from P pages to P/2 runs
  - Pass 2: went from P/2 runs to P/4 runs
  - ...
  - Total number of passes: \([\log_2 P] + 1\)

- Total cost: \(2P \times ([\log_2 P] + 1)\)
What did that cost us?

- Why is this better than plain old merge sort?
  - \( N \gg P \)
  - So \( O(N \log N) \gg O(P \log P) \)

- Example:
  - 1,000,000 record file
    - 8 KB pages
    - 100 byte records
    - = 80 records per page
    - = 12,500 pages

  - Plain merge sort: 41,863,137 disk I/O’s
  - 2-way external merge sort: 365,241 disk I/O’s
  - 4.8 days versus 1 hour
Can we do better?

- 2-way merge sort only uses 3 memory buffers
  - Two buffers to hold input records
  - One buffer to hold output records
    - When that buffer fills up, flush to disk

- Usually we have a lot more memory than that
  - Set aside 100 MB for sort scratch space = 12,800 buffer pages

- Idea: read as much data into memory as possible each pass
  - Thus reducing the number of passes
  - Recall total cost:

  \[ 2P \times \text{Passes} \]
External merge sort

- Assign B input buffers and 1 output buffer
- Pass 0: Read in runs of B pages, sort, write to disk
- Pass 1: Merge B runs into one
  - For each run, read one block
  - When a block is used up, read next block of run
- Pass 2: Merge B runs into one
- ...
- Sorted!
Example

Input

Output
Example

____________________ Input _________________________ Output
Example

Input

Output
Example

Input

Output
Example
Example
Example

Input ———— Output
Example

_________________________ Input ____________________________ Output
Example

__________________________ Input __________________________ Output
Example

______________________________ Input ________________________________ Output
Example

Input   Output
Example

_________________________ Input ________________________ Output
Example

_________________________ Input ___________________________ Output
Example

__________________________ Input ____________________________ Output
Example

____________________ Input ____________________  Output
What did that cost us?

- P pages in file, B buffer pages in RAM
- P/B runs of size B
- Each pass: read and write P pages

How many passes?
- \([\log_{B-1} \left \lfloor \frac{P}{B} \right \rfloor] + 1\)

Total cost: \(2P \times [\log_{B-1} \left \lfloor \frac{P}{B} \right \rfloor] + 1\)
Example

- 1,000,000 records in 12,500 pages
- Use 10 buffer pages in memory
- 4 passes
- 100,000 disk I/Os
  - 17 minutes versus 1 hour for 2-way sort
Can I do two passes?

- Pass 0: sort runs
- Pass 1: merge runs

- Given B buffers
- Need:
  - No more than B-1 runs
  - Each run no longer than B pages

- Can do two passes if \( P \leq B \times (B-1) \)

- Question: what’s the largest file we can sort in three passes? N passes?
Make I/Os faster

- Cost = I/Os is a simplification
  - Sequential I/Os are cheaper than random I/Os

- Read blocks of pages at a time
  - $X = $Blocking factor
  - $B = $buffer pages
  - $(B/X - X)$ input “buffer blocks”, one output “buffer block”

- Result
  - Fewer runs merged per pass = more passes
  - Less time per I/O = quicker passes
  - Tradeoff!
    - Maximize total sort time by choosing $X$ given $B$, $P$ and I/O latencies
Overlap computation and I/O

Problem: CPU must wait for I/O

- Suppose I need to read a new block
  - Stop merging
  - Initiate I/O
  - Wait
  - Complete I/O
  - Resume merging
Solution: double buffering

- Keep a second set of buffers
  - Process one set while waiting for disk I/O to fill the other set
Solution: double buffering

- Keep a second set of buffers
  - Process one set while waiting for disk I/O to fill the other set
Solution: double buffering

- Keep a second set of buffers
  - Process one set while waiting for disk I/O to fill the other set
Solution: double buffering

- Keep a second set of buffers
  - Process one set while waiting for disk I/O to fill the other set
Solution: double buffering

- Keep a second set of buffers
  - Process one set while waiting for disk I/O to fill the other set
Solution: double buffering

- Keep a second set of buffers
  - Process one set while waiting for disk I/O to fill the other set
What if the data is already sorted?

- **Yay!**

- Often this happens because of a B+ tree index
  - Leaf level of a B+ tree has all records in sorted order
  - Two possibilities: B+ tree is clustered or unclustered
Clustered B+ tree

Sweep through leaf layer, reading data blocks in order
Clustered B+ tree

Sweep through leaf layer, reading leaf blocks in order
What did that cost us?

- Traverse B+ tree to left-most leaf page
- Read all leaf pages
  - For each leaf page, read data pages

- Data not in B+ tree:
  - Height + Width + Data pages

- Data in B+ tree:
  - Height + Width
Example

- 1,000,000 records, 12,500 data pages
- Assume keys are 10 bytes, disk pointers are 8 bytes
  - So $\approx 300$ entries per 8 KB B+ tree page (if two-thirds full)

**Data not in B+ tree**
- 12,500 entries needed = 42 leaf pages
- Two level B+tree
- Total cost: $1 + 42 + 12,500 = 12,543$ I/Os
- 2 minutes versus 17 minutes for external merge sort

**Data in B+ tree**
- Three level B+ tree, 12,500 leaf pages
- Total cost: $2 + 12,500 = 12,502$ I/Os
- Also about 2 minutes
What if the B+ tree is unclustered?

- We know the proper sort order of the data
- But retrieving the data is hard!
What if the B+ tree is unclustered?

- Result is that in the worst case, may need one disk I/O per record
  - Even though we know the sort order!

- Usually external merge sort is better in these cases
  - Unless all you need is the set of keys
Summary

- Sorting is very important

- Basic algorithms not sufficient
  - Assume memory access free, CPU is costly
  - In databases, memory (e.g. disk) access is costly, CPU is (almost free)

- Try to minimize disk accesses
  - 2-way sort: read and write records in blocks
  - External merge sort: fill up as much memory as possible
  - Blocked I/O: try to do sequential I/O
  - Double buffering: read and compute at the same time
  - Clustering B+ tree: the data is already sorted. Hooray!
  - Unclustered B+ tree: no help at all
Do YOU
YAHOO!
?