Comp115: Databases

External Sorting

Instructor: Manos Athanassoulis
External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using B⁺-Trees for sorting
Why Sort?

a classic problem in computer science!
but also a database specific problem:
(i) data requested in sorted order
e.g., find students in increasing *gpa* order
(ii) *bulk loading* B+ tree index
(iii) eliminating *duplicate* (Why?)
(iv) summarizing groups of tuples
(v) *Sort-merge* join [more about that later]
Sorting Challenges

(easy) problem:
how to sort 1GB data with 1GB memory?

(hard) problem:
how to sort 1GB data with 1MB memory?

why not virtual memory (i.e., swapping on disk)?
Goal

minimize disk accesses when working under memory constraints

Idea

stream over data, calculate something useful, and write back on disk
Streaming Data Through RAM

An important method for sorting & other DB operations

Simple case:

- Compute $f(x)$ for each record, write out the result
- Read a page from INPUT to Input Buffer
- Write $f(x)$ for each item to Output Buffer
- When Input Buffer is consumed, read another page
- When Output Buffer fills, write it to OUTPUT

Reads and Writes are *not* coordinated

- E.g., if $f()$ is Compress(), you read many pages per write.
- E.g., if $f()$ is DeCompress(), you write many pages per read.
2-Way Sort: Requires 3 Buffers

Pass 0: Read a page, sort it, write it.
- only one buffer page is used (as in previous slide)

Pass 1, 2, 3, ..., etc.:
- requires 3 buffer pages
- merge pairs of runs into runs twice as long
- three buffer pages used.
Two-Way External Merge Sort

Each pass we read + write each page in file.

N pages in the file => the number of passes

\[= \left\lceil \log_2 N \right\rceil + 1\]

So total cost is:

\[2N\left(\left\lceil \log_2 N \right\rceil + 1\right)\]

\textit{Idea: Divide and conquer:}\n
sort subfiles and merge
External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using B+*-Trees for sorting
To sort a file with $N$ pages using $B$ buffer pages:

- Pass 0: use $B$ buffer pages. Produce $\left\lceil \frac{N}{B} \right\rceil$ sorted runs of $B$ pages each.
- Pass 1, 2, ..., etc.: merge $B-1$ runs.

More than 3 buffer pages. How can we utilize them?
General External Merge Sort

N = 108 pages

0: 5 5  ... [108/5] = 22 sorted runs of 5 pages each (last run 3 pages)

1: 20 20  ... [22/4] = 6 sorted runs of 5 \cdot 4 = 20 pages each (last run 8)

2: 80  ... [6/4] = 2 sorted runs of 20 \cdot 4 = 20 pages (last run 28)

3: Sorted File!

B=5 buffer pages

\[N = 108\]
Cost of External Merge Sort

Number of passes: \( 1 + \left\lfloor \log_{B-1} \left[ \frac{N}{B} \right] \right\rfloor \)

Cost = \( 2N \times (\# \text{ of passes}) \)

to sort 108 page file with 5 buffers:

- Pass 0: \( \left\lfloor \frac{108}{5} \right\rfloor = 22 \) sorted runs of 5 pages each (last run is only 3 pages)
- Pass 1: \( \left\lfloor \frac{22}{4} \right\rfloor = 6 \) sorted runs of 20 pages each (last run is only 8 pages)
- Pass 2: 2 sorted runs, 80 pages and 28 pages
- Pass 3: Sorted file of 108 pages

Formula check: \( 1 + \left\lfloor \log_{B-1} \left[ \frac{N}{B} \right] \right\rfloor = 1 + \left\lfloor \log_4 22 \right\rfloor = 1 + 3 \)
Number of Passes of External Sort

I/O cost is $2N$ times number of passes: $2 \cdot N \cdot (1 + \lceil \log_{B-1}[N/B] \rceil)$

<table>
<thead>
<tr>
<th>N</th>
<th>B=3</th>
<th>B=5</th>
<th>B=9</th>
<th>B=17</th>
<th>B=129</th>
<th>B=257</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>10,000</td>
<td>13</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>100,000</td>
<td>17</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1,000,000</td>
<td>20</td>
<td>10</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10,000,000</td>
<td>23</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>100,000,000</td>
<td>26</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>30</td>
<td>15</td>
<td>10</td>
<td>8</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
In-Memory Sort Algorithm

Quicksort is fast (very fast)!!
we generate in Pass 0 N/B #runs of B pages each

can we generate longer runs?
why do we want that?

yes! Idea: maintain a current set as a heap
In-memory Heapsort

(aka “replacement sort”)

0: read in B-2 blocks
1: find the smallest record greater than the largest value to output buffer
   – add it to the end of the output buffer
   – fill moved record’s slot with next value from the input buffer, if empty refill input buffer
2: else: end run
3: goto (1)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input  current  output

file (on disk)
In-memory Heapsort

$N = 7$ pages (file), $B = 3$ pages (buffers)

\[ 30, 20 \quad 10, 40 \quad 22, 17 \quad 25, 73 \quad 16, 26 \quad 21, 13 \quad 22, 24 \]

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

\[ 10, 17, 20, 22, 30, 40 \quad 13, 16, 21, 25, 26, 73 \quad 22, 24 \]

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20 10, 40 22, 17 25, 73 16, 26 21, 13 22, 24

Normally we use 3-pages runs in Pass 0

10, 17, 20, 22, 30, 40 13, 16, 21, 25, 26, 73 22, 24

Heapsort 3-2=1 page

input     current     output

20, 30

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

<table>
<thead>
<tr>
<th>N = 7 pages (file)</th>
<th>B = 3 pages (buffers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30, 20</td>
<td>10, 40</td>
</tr>
<tr>
<td>10, 40</td>
<td>22, 17</td>
</tr>
<tr>
<td>25, 73</td>
<td>16, 26</td>
</tr>
<tr>
<td>21, 13</td>
<td>22, 24</td>
</tr>
<tr>
<td>10, 17, 20, 22, 30, 40</td>
<td>13, 16, 21, 25, 26, 73, 22, 24</td>
</tr>
</tbody>
</table>

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

```
input: 10, 40

output: 20, 30

file (on disk)
```
# In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

<table>
<thead>
<tr>
<th>Input</th>
<th>Current</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>30, 20</td>
<td>10, 40</td>
<td>22, 17</td>
</tr>
<tr>
<td>25, 73</td>
<td>16, 26</td>
<td>21, 13</td>
</tr>
<tr>
<td>22, 24</td>
<td>10, 17, 20, 22, 30, 40</td>
<td>13, 16, 21, 25, 26, 73</td>
</tr>
</tbody>
</table>

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input  current  output

40  30  10, 20

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20 10, 40 22, 17 25, 73 16, 26 21, 13 22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input

current 30, 40

output 10, 20

file (on disk)
**In-memory Heapsort**

N = 7 pages (file), B = 3 pages (buffers)

\[
\begin{align*}
30, 20 & & 10, 40 & & 22, 17 & & 25, 73 & & 16, 26 & & 21, 13 & & 22, 24 \\
\end{align*}
\]

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

- **input**
  - 22, 17

- **current**
  - 30, 40

- **output**
  - 10, 20

*file (on disk)*
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input | current | output
--- | --- | ---
22, 17 | 30, 40 | 20

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input  current  output

22       30, 40   17, 20

10

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

10, 17, 20, 22, 30, 40  13, 16, 21, 25, 26, 73  22, 24

Heapsort 3-2=1 page

\begin{align*}
\text{input} & : & 22 \\
\text{current} & : & 30, 40 \\
\text{output} & : & 20 \\
\end{align*}

\begin{align*}
\text{file (on disk)} & : & 10, 17 
\end{align*}
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use
3-pages runs in
Pass 0

Heapsort
3-2=1 page

input                  current                  output

file (on disk)

10, 17

30, 40

20, 22
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input

25, 73

current

30, 40

output

20, 22

file (on disk)

10, 17
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input

25, 73
current

30, 40
output

10, 17, 20, 22

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

10, 17, 20, 22, 30, 40  13, 16, 21, 25, 26, 73  22, 24

Heapsort 3-2=1 page

input  current  output

40, 73  25, 30

10, 17, 20, 22

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input

16, 26

current

40, 73

output

25, 30

10, 17, 20, 22

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

<table>
<thead>
<tr>
<th>input</th>
<th>current</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>16, 73</td>
<td>30, 40</td>
<td>25, 26</td>
</tr>
</tbody>
</table>

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

16, 73

10, 17, 20, 22, 25, 26

file (on disk)
## In-memory Heapsort

In-memory Heapsort

\[ \text{N} = 7 \text{ pages (file), } \text{B} = 3 \text{ pages (buffers)} \]

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
<td>10</td>
<td>40</td>
<td>22</td>
<td>17</td>
<td>25</td>
</tr>
<tr>
<td>73</td>
<td>16</td>
<td>26</td>
<td>21</td>
<td>13</td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

\begin{tabular}{|c|c|c|}
\hline
\text{input} & \text{current} & \text{output} \\
\hline
21 & 13 & 73 & 16 & 30 & 40 \\
\hline
\end{tabular}

\textit{file (on disk)}
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input  current  output

21, 13  73, 16

10, 17, 20, 22, 25, 26, 30, 40

file (on disk)
# In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

<table>
<thead>
<tr>
<th>input</th>
<th>current</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>30, 20</td>
<td>10, 40</td>
<td>22, 17</td>
</tr>
<tr>
<td>25, 73</td>
<td>16, 26</td>
<td>21, 13</td>
</tr>
<tr>
<td>22, 24</td>
<td>10, 17, 20, 22, 30, 40</td>
<td>22, 24</td>
</tr>
</tbody>
</table>

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20 10, 40 22, 17 25, 73 16, 26 21, 13 22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

file (on disk)

new file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

<table>
<thead>
<tr>
<th>input</th>
<th>current</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>30, 20</td>
<td>24</td>
<td>10, 17, 20, 22, 30, 40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13, 16, 21, 25, 26, 73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22, 24</td>
</tr>
</tbody>
</table>

file (on disk)

new file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20 10, 40 22, 17 25, 73 16, 26 21, 13 22, 24

Normally we use 3-pages runs in Pass 0

Heapsort

3-2=1 page

file (on disk)

new file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort

3-2=1 page

input current output

file (on disk)

new file (on disk)
In-memory Heapsort

N = 7 pages (file), B = 3 pages (buffers)

30, 20  10, 40  22, 17  25, 73  16, 26  21, 13  22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

input

10, 17, 20, 22, 30, 40

13, 16, 21, 25, 26, 73

output

file (on disk)

only 2 (longer) sorted runs!

10, 17, 20, 22, 25, 26, 30, 40, 73

new file (on disk)

13, 16, 21, 22, 24
More on Heapsort

Fact:

average length of a run in heapsort is $2(B-2)$

Worst-Case:

- What is min length of a run?
- How does this arise?

Best-Case:

- What is max length of a run?
- How does this arise?

Quicksort is faster, but ... longer runs often means fewer passes!
External Merge Sort Summary

Unsorted file of N pages

0:

\[ B \quad B \]

\([N/B]\) sorted runs of B pages each
(or, fewer of 2(B – 2) each)

1:

\[ B(B - 1) \quad B(B - 1) \]

\(\frac{[N/B]}{B-1} \) sorted runs of
B(B – 1) pages each

2:

\[ B(B - 1)^2 \quad B(B - 1)^2 \]

\(\frac{[N/B]}{(B-1)^2} \) sorted runs of
B(B – 1)^2 pages each

\[ \ldots \]

\( \log_{B-1} \left( \left\lfloor \frac{N}{B} \right\rfloor \right) : \)

\[
\frac{[N/B]}{(B-1)^{\log_{B-1}(\lfloor N/B \rfloor)}} = 1 \text{ sorted run! of } B \cdot (B - 1)^{\log_{B-1}(\lfloor N/B \rfloor)} = B \cdot \left\lfloor \frac{N}{B} \right\rfloor = N \text{ pages}
\]

B buffer pages:

\[ \ldots \]

45
I/O for External Merge Sort

Do I/O a page at a time
  – Not one I/O per record

In fact, read a block (chunk) of pages sequentially!

Suggests we should make each buffer (input/output) be a block of pages.
  – But this will reduce fan-in during merge passes!
  – In practice, most files still sorted in 2-3 passes.
Double Buffering

To reduce wait time for I/O request to complete, can \textit{prefetch} into \textit{“shadow block”}.

– Potentially, more passes; in practice, most files \textit{still} sorted in 2-3 passes.
Sorting Records!

Sorting has become a blood sport!

– Parallel sorting is the name of the game ... 

Minute Sort: how many 100-byte records can you sort in a minute?

Penny Sort: how many can you sort for a penny?

See http://sortbenchmark.org/
External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using $B^+$-Trees for sorting
Using B+ Trees for Sorting

Scenario: Table to be sorted has B+ tree index on sorting column(s).

Idea: Can retrieve records in order by traversing leaf pages.

Is this a good idea?

Cases to consider:

- B+ tree is clustered
- B+ tree is not clustered
Using B+ Trees for Sorting

Scenario: Table to be sorted has B+ tree index on sorting column(s).

Idea: Can retrieve records in order by traversing leaf pages.

Is this a good idea?

Cases to consider:

- B+ tree is clustered  
  Good idea!
- B+ tree is not clustered
Using B+ Trees for Sorting

Scenario: Table to be sorted has B+ tree index on sorting column(s).

Idea: Can retrieve records in order by traversing leaf pages.

Is this a good idea?

Cases to consider:

- B+ tree is clustered: Good idea!
- B+ tree is not clustered: Could be a very bad idea!
Clustered B+ Tree Used for Sorting

Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)

If Alternative 2 is used?
Additional cost of retrieving data records: each page fetched just once.

Always better than external sorting!
Unclustered B+ Tree Used for Sorting

Alternative (2) for data entries; each data entry contains *rid* of a data record. In general, one I/O per data record!

![Diagram of Unclustered B+ Tree]

Index
(Directs search)

Data Entries
("Sequence set")

Data Records
# External Sorting vs. Unclustered Index

<table>
<thead>
<tr>
<th>N</th>
<th>Sorting</th>
<th>p=1</th>
<th>p=10</th>
<th>p=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>100</td>
<td>1,000</td>
<td>10,000</td>
</tr>
<tr>
<td>1,000</td>
<td>2,000</td>
<td>1,000</td>
<td>10,000</td>
<td>100,000</td>
</tr>
<tr>
<td>10,000</td>
<td>40,000</td>
<td>10,000</td>
<td>100,000</td>
<td>1,000,000</td>
</tr>
<tr>
<td>100,000</td>
<td>600,000</td>
<td>100,000</td>
<td>1,000,000</td>
<td>10,000,000</td>
</tr>
<tr>
<td>1,000,000</td>
<td>8,000,000</td>
<td>1,000,000</td>
<td>10,000,000</td>
<td>100,000,000</td>
</tr>
<tr>
<td>10,000,000</td>
<td>80,000,000</td>
<td>10,000,000</td>
<td>100,000,000</td>
<td>1,000,000,000</td>
</tr>
</tbody>
</table>

- $p$: # of records per page
- $B=1,000$ and block size=32 for sorting
- $p=100$ is the more realistic value.
External sorting is important

External merge sort minimizes disk I/O cost:

- Pass 0: Produces sorted runs of size $B$ (# buffer pages). Later passes: merge runs.
- # of runs merged at a time depends on $B$, and block size.
- Larger block size means less I/O cost per page.
- Larger block size means smaller # runs merged.
- In practice, # of passes rarely more than 2 or 3.

Summary
Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament sort: slower (2x), longer runs

The best sorts are wildly fast:
  - Despite 40+ years of research, still improving!

Clustered B+ tree is good for sorting
Unclustered tree is usually very bad