CS460: Intro to Database Systems

Class 12: External Sorting

Instructor: Manos Athanassoulis

https://bu-disc.github.io/CS460/
Midterm next week on Friday

Thursday 10/22 during class (no lecture): answer questions about topics covered up to now (including today).

Friday 10/23: we will have the midterm.

Available for 24 hours, you will have 120 minutes to complete it once you start.

We will announce all the details in a follow-up message in Piazza.
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, with many use cases:
Why Sort?

a *classic problem* in computer science!

but also a *database specific* problem, with many use cases:

(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 (*what is that?*) GROUP BY!

(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 \textbf{1MB} memory?

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

minimize disk accesses when working under memory constraints

Idea

stream 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 ??

So total cost is: ??

Idea

Divide and conquer
sort sub-files and merge
Two-Way External Merge Sort

Each pass we read + write each page in file.
N pages in the file =>
the number of passes = \([\log_2 N] + 1\)

So total cost is: \(2N([\log_2 N] + 1)\)

**Idea**

*Divide and conquer*
sort sub-files and merge
External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using B\(^+\)-Trees for sorting
General External Merge Sort

More than 3 buffer pages. How can we utilize them?

To sort a file with $N$ pages using $B$ buffer pages:

- Pass 0: use $B$ buffer pages. Produce $\lceil N/B \rceil$ sorted runs of $B$ pages each.
- Pass 1, 2, ..., etc.: merge $B-1$ runs.
General External Merge Sort

N = 108 pages

0:

5 5 ... \( \frac{108}{5} = 22 \) sorted runs of 5 pages each (last run 3 pages)

1:

20 20 ... \( \frac{22}{4} = 6 \) sorted runs of \( 5 \cdot 4 = 20 \) pages each (last run 8)

2:

80 ... \( \frac{6}{4} = 2 \) sorted runs of \( 20 \cdot 4 = 20 \) pages (last run 28)

3:

Sorted File!

\( B=5 \) buffer pages
Cost of External Merge Sort

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

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

to sort 108 page file with 5 buffers:

- Pass 0: \([108/5] = 22\) sorted runs of 5 pages each (last run is only 3 pages)
- Pass 1: \([22/4] = 6\) 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: \(1 + \left\lfloor \log_{B-1}[N/B] \right\rfloor = 1 + [\log_4 22] = 1 + 3\)
Number of Passes of External Sort

I/O cost is $2N$ times number of passes: $2 \cdot N \cdot \left(1 + \left\lceil \log_{|B|} \left(\frac{N}{B}\right) \right\rceil \right)$

<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)

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  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

30, 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

20, 30

output

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

Input: 10, 17, 20, 22, 30, 40

Output: 13, 16, 21, 25, 26, 73, 22, 24

Heapsort 3-2=1 page

```
input | current | output
10, 40 | 20, 30   |
```

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          20, 30          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

Heapsort 3-2=1 page

<table>
<thead>
<tr>
<th>input</th>
<th>current</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>30</td>
<td>10, 20</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

<table>
<thead>
<tr>
<th>input</th>
<th>current</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30, 40</td>
<td>10, 20</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

input

22, 17

current

30, 40

output

10, 20

update the heap

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, 20
30, 40
10, 17

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

20, 22

current

30, 40

output

10, 17

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></td>
<td>30, 40</td>
<td>20, 22</td>
</tr>
</tbody>
</table>

file (on disk)

10, 17

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

10, 17, 20, 22, 30, 40 13, 16, 21, 25, 26, 73 22, 24
**In-memory Heapsort**

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

10, 17, 20, 22, 30, 40

13, 16, 21, 25, 26, 73

22, 24

Normally we use 3-pages runs in Pass 0

Heapsort 3-2=1 page

Here we end up writing both values, one at a time (no change by resorting)
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

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

input  current  output

25, 73    30, 40

10, 17, 20, 22

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
--- | --- | ---
10, 17, 20, 22, 30, 40 | 40, 73 | 25, 30

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

16, 26  40, 73  25, 30

10, 17, 20, 22

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

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

10, 17, 20, 22, 30, 40

13, 16, 21, 25, 26, 73  22, 24

input  current  output

16, 73  30, 40  25, 26

10, 17, 20, 22

file (on disk)
# In-memory Heapsort

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

Normally we use

3-pages runs in Pass 0

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

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></td>
</tr>
</tbody>
</table>

10, 17, 20, 22, 25, 26

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

21, 13

current

73, 16

output

30, 40

10, 17, 20, 22, 25, 26

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>21, 13</td>
<td>73, 16</td>
<td></td>
</tr>
</tbody>
</table>

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

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 21

current 13, 16

output 73

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

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>21</td>
<td>13, 16</td>
<td></td>
</tr>
</tbody>
</table>

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

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

file (on disk)

new file (on disk)
# In-memory Heapsort

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

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

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>22, 24</td>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>

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

file (on disk)

11, 17, 20, 22, 25, 26, 30, 40, 73

new file (on disk)

13, 16
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

10, 17, 20, 22, 30, 40

Heapsort

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

output

21, 22

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

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

file (on disk)

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

new file (on disk)

13, 16, 21, 22
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

output

current

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

file (on disk)

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

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

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

only 2 (longer) sorted runs!

file (on disk)

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:

\[
\begin{align*}
B & \quad B & \quad B
\end{align*}
\]

\([N/B] \text{ sorted runs of } B \text{ pages each}
\]

(or, fewer of \(2(B - 2)\) each)

1:

\[
\begin{align*}
B(B - 1) & \quad B(B - 1) & \quad B(B - 1)
\end{align*}
\]

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

2:

\[
\begin{align*}
B(B - 1)^2 & \quad B(B - 1)^2 & \quad B(B - 1)^2
\end{align*}
\]

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

\[\ldots\]

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

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

Total #I/O: \(2 \cdot N \cdot (1 + \left\lfloor \log_{B-1}([N/B]) \right\rfloor)\)

B buffer pages:
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 *prefetch* into “shadow block”.

- Potentially, more passes; in practice, most files *still* sorted in 2-3 passes.

![Diagram of double buffering](https://bu-disc.github.io/CS460/)

**B main memory buffers, k-way merge**
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!
### 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>

- if $B \geq N$ then only quick sort!
- Special case, that the tree is always behaving like a clustered tree

$p$: # of records per page

$B=1,000$ and block size=32 for sorting

$p=100$ is the more realistic value.
Summary

External sorting is used for many different operations in DBs

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 fewer runs merged.
- In practice, # of passes rarely more than 2 or 3.
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