Query Evaluation
-- External Sorting

References:
- [RG-3ed] Chapter 13
- [SKS-6ed] Chapter 12.4
Outline

- Why sorting?
- External sorting
  - 2-way sort
  - M-way sort
- Other considerations
  - Blocked I/O
  - Double buffering
- Sorting using index
  - Clustered
  - Non-clustered
A classic problem in computer science!
Data requested in sorted order
  e.g., find students in increasing GPA order
Sorting is the first step in bulk loading B+ tree index.
Sorting is useful for eliminating duplicate copies in a collection of records (Why?)
Sort-merge join algorithm involves sorting.
Problem: sort 1Gb of data with 1Mb of RAM.
We may build an index on the relation, and then use the index to read the relation in sorted order.

- May lead to one disk block access for each tuple.

For relations that fit in memory, techniques like quicksort can be used.

For relations that do not fit in memory, external sort-merge is a good choice.
Sorting in Commercial RDBMs

- External merge sort
  - DB2, Informix, SQL Server, Oracle 8, Sybase ASE
  - None of these systems uses the optimization that produces runs larger than available memory
  - I/O is asynchronous and prefetching
- In-memory
  - Miscrost, Sybase ASE: merge sort
  - DB2 and Sybase IQ: radix sorting
  - Oracle: insertion sort
Concepts

- **Run**
  - When sorting a file, several sub-files are typically generated in intermediate steps. Each sorted sub-file is referred to as a run.

- **Pass**

- Available buffer main memory: $M$
- Number of pages in the file: $b_r$
2-Way Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it.
  - only one buffer page is used
- Pass 2, 3, …, etc.:
  - three buffer pages used.
- Each pass we read + write each page in file.
- Let $b_r$ pages in the file
  - the number of passes
    - $= \lceil \log_2 b_r \rceil + 1$
- So total cost is:
  - $2b_r(\lceil \log_2 b_r \rceil + 1)$

- Idea: Divide and conquer: sort sub-files and merge
More than 3 buffer pages. How can we utilize them?

To sort a file with $b_r$ pages using $M$ buffer pages:

- **Pass 0**: use $M$ buffer pages. Produce $\left\lfloor \frac{b_r}{M} \right\rfloor$ sorted runs of $M$ pages each.

- **Pass 2, ..., etc.**: merge $M-1$ runs.

Diagram:

```
Disk -> INPUT 1 -> M Main memory buffers -> OUTPUT -> Disk
          |                      |
          |                      |
          |                      |
          |                      |
```

```
Disk
```

```
```

```
```
External Sort-Merge

\(b_r:\) total number of pages in a file
\(M:\) memory buffer size (in pages).

1. **Create sorted runs.**

   Let \(i\) be 0 initially.

   Repeatedly do the following till to the end of the relation:
   
   (a) Read \(M\) blocks of relation into memory
   (b) Sort the in-memory blocks
   (c) Write sorted data to run file \(R_i\)
   (d) Increment \(i\)

   total number of runs \(N_1 = \lfloor b_r/M \rfloor\)

2. **Merge the runs** (*next slide*).....
External Sort-Merge (Cont.)

2. Merge the runs ([b_r/M]-way merge).

   We assume (for now) that \([b_r/M] < M\).

   1. Use \([b_r/M]\) blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each run into its buffer page.

2. repeat

   1. Select the first record (in sort order) among all buffer pages.
   2. Write the record to the output buffer. If the output buffer is full, write it to disk.
   3. Delete the record from its input buffer page. 
      If the buffer page becomes empty then read the next block (if any) of the run into the buffer.

3. until all input buffer pages are empty:
Merge the runs. If $\lceil b_r/M \rceil \geq M$,
several merge passes are required.
In each pass, contiguous groups of $M-1$ runs are merged.
Repeated passes are performed till all runs have been merged into one.

A pass reduces the number of runs by a factor of $M -1$, and creates runs longer by the same factor.
E.g. If $M=11$, and there are 90 runs after the 1st pass, one pass reduces the number of runs to 9 ($90/(M-1) = 90/10 = 9$), each 10 times the size of the initial runs.
Example: External Sorting Using Sort-Merge

- \( b_r = 12 \) (each record uses one page.
- \( M = 3 \)

\[\begin{array}{c}
\text{initial relation} \\
\text{create runs} \\
\text{merge pass–1} \\
\text{merge pass–2} \\
\text{sorted output}
\end{array}\]
External Merge Sort (Cont.)

- Cost analysis:
  - # of initial runs $N_1 = \lceil b_r / M \rceil$
  - # of merge passes $= \lceil \log_{M-1} N_1 \rceil$
  - # of passes (include initial pass) $= \lceil \log_{M-1} N_1 \rceil + 1$
  - Block transfers: for initial run creation as well as in each pass is $2b_r$
    - for final pass, we do not count write cost
      - If ignore final write cost for all operations
        » This may happen when the output of an operation is sent to the parent operation without being written to disk
        » Thus total number of block transfers for external sorting: $b_r (2 \times (\# \text{ of merge passes}) + 1)$
      - If include the final write cost: $b_r (2 \times \# \text{ of passes})$
Example: External Sorting Using Sort-Merge

<table>
<thead>
<tr>
<th></th>
<th>g</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

initial relation

create runs

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

runs

merge pass–1

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

merge pass–2

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

sorted output

runs

Example: External Sorting Using Sort-Merge

- \( b_r = 12 \) (each record uses one page.
- \( M = 3 \)
- \( b_b = 1 \)

- \# of initial runs \( N_1 = 12/3 = 4 \)
- \# of merge passes: \( \log_2 4 = 2 \)
- \# of passes: \( 2+1 = 3 \)

- Transfer cost (without final writing) \( 12 \times (2 \times 2 + 1) = 12 \times 5 = 60 \)

- Transfer cost (with final writing) \( 12 \times (2 \times 3) = 72 \)
Cost of External Merge Sort

- # of passes: 1 + \[\log_{M-1}\left[\frac{b_r}{M}\right]\]
- Transferring Cost = 2b_r \times (# of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: \([108/5] = 22\) sorted runs of 5 pages each (last run contains only 3 pages)
  - Pass 1: \([22/4] = 6\) sorted runs of 20 pages each (last run contains only 8 pages)
  - Pass 2: 2 (= \([6/4]\)) sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages
- # of passes = 1 + \log_4 22 = 1 + 3 = 4
- Cost = 108 \times (4 \times 2) = 864 (with final write)
## Number of Passes of External Sort

<table>
<thead>
<tr>
<th>$br$</th>
<th>M=3</th>
<th>M=5</th>
<th>M=9</th>
<th>M=17</th>
<th>M=129</th>
<th>M=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>

- $b_r = 100, M=3$  \[ N1 = \lceil \frac{100}{3} \rceil = 34, \quad 1 + \log_2 34 = 1 + 6 = 7 \]
- $b_r = 100,000, M=9$  \[ N1 = \lceil \frac{100,000}{9} \rceil = 11112, \quad 1 + \log_8 11112 = 1 + 5 = 6 \]
Suppose you have a file with 10,000 pages and you have three buffer pages.

Answer the following questions

Q1: How many runs will you produce in the first pass
Q2: How many passes will it take to sort the file completely
Q3: What is the total I/O cost of sorting the file (with final write)?
Q4: How many buffer pages do you need to sort the file completely in just two passes?
Exercise

- Suppose you have a file with 10,000 pages and you have three buffer pages.
- Answer the following questions
  - Q1: How many runs will you produce in the first pass
  - Q2: How many passes will it take to sort the file completely
  - Q3: What is the total I/O cost of sorting the file (with final write)?
  - Q4: How many buffer pages do you need to sort the file completely in just two passes?

- \( b_r = 10,000; M = 3 \)
  - Q1: \# of initial runs = \( \lceil b_r/M \rceil = 3334 \)
  - Q2: \( 1 + \lceil \log_2 3334 \rceil = 1+12 = 13 \) passes
  - Q3: \( 10000 \times (2\times13) = 260,000 \)
  - Q4: Pass 0 \( \lceil b_r/M \rceil \); pass 1 need to finish the merging, thus \( M-1 \geq \lceil b_r/M \rceil \), what is the minimum \( M \)? \( M = 101 \)
Exercise

Suppose you have a file with 100,000 pages and you have five buffer pages.

Answer the following questions

- Q1: How many runs will you produce in the first pass
- Q2: How many passes will it take to sort the file completely
- Q3: What is the total I/O cost of sorting the file?
- Q4: How many buffer pages do you need to sort the file completely in just two passes?
External Merge Sort – blocked I/O (S.S.)

- Cost (include writing final results)
  - \( b_r \times (2 \times \# \text{ of passes}) = 2b_r \times ([\log_{M-1} N1] + 1) \)
- Minimize cost \( \rightarrow \) minimize the number of passes \( \rightarrow \) maximize the fan-in merging

- Blocked access: read a block of pages sequentially!
  - Each time: read and write a block of \( b \) pages
  - Output block pages: \( b \)
  - Input block pages: \( M-b \)
    - Number of input blocks \( [(M-b)/b] \)
  - Merge at most \( [(M-b)/b] \) runs in each pass
    - E.g., \( M=10 \), one-page input/output block: fan-in = \( M-1 = 9 \)
    - 2-page input/output block: fan-in = \( (10-2)/2 = 4 \)
- \# of page I/Os trade off per-page I/O cost
In fact, 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 can be sorted in just two passes, even using blocked I/O.

First pass,

- generate # of runs $N_1 = \lceil b_r / M \rceil$

Fan-in factor: $F = \lfloor M / b \rfloor - 1$

# of passes: $1 + \lceil \log_F N_1 \rceil$
Number of Passes of Optimized Sort (S.S.)

- Block size $b = 32$, initial pass produces runs of size $M$
- $M=1000$, $b_r = 10,000$ pages
- $M=5000$, $b_r = 10,000,000$

<table>
<thead>
<tr>
<th>$M$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>$\left\lfloor \frac{1000}{32} \right\rfloor - 1 = 30$</td>
</tr>
<tr>
<td>5000</td>
<td>$\left\lfloor \frac{5000}{32} \right\rfloor - 1 = 155$</td>
</tr>
<tr>
<td>10000</td>
<td>$\left\lfloor \frac{5000}{32} \right\rfloor - 1 = 311$</td>
</tr>
</tbody>
</table>
Number of Passes of Optimized Sort (S.S.)

- Block size $b = 32$, initial pass produces runs of size $M$
- $M=1000$, $b_{r} = 10,000$ pages
- $M=5000$, $b_{r} = 10,000,000$

<table>
<thead>
<tr>
<th>$M$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>$\lceil 1000/32 \rceil - 1 = 30$</td>
</tr>
<tr>
<td>5000</td>
<td>$\lceil 5000/32 \rceil - 1 = 155$</td>
</tr>
<tr>
<td>10000</td>
<td>$\lceil 5000/32 \rceil - 1 = 311$</td>
</tr>
</tbody>
</table>

$M=1000$, $b_{r} = 10,000$ pages \hspace{1cm} N1 = $\lceil br/M \rceil = 10,$ \hspace{1cm} # of passes = $1 + \lceil \log_{30}10 \rceil = 2$

$M=5000$, $br = 10,000,000$ \hspace{1cm} N2 = $\lceil br/M \rceil = 2000$ \hspace{1cm} # of passes = $1 + \lceil \log_{155}2000 \rceil = 3$
Exercise (S.S.)

- $M = 5000$

<table>
<thead>
<tr>
<th>$b$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>???</td>
</tr>
<tr>
<td>32</td>
<td>???</td>
</tr>
<tr>
<td>64</td>
<td>???</td>
</tr>
</tbody>
</table>
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.
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 \textit{Good idea!}
  - B+ tree is not clustered \textit{Could be a very bad idea!}
Clustered B+ Tree Used for Sorting

- Cost:
  - root to the left-most leaf (<4)
  - retrieve all leaf pages
- If <key, rid> pair is used in the index?
  - Additional cost of retrieving data records: each page fetched just once.
Unclustered B+ Tree Used for Sorting

- <key, rid> pair is used for data entries; each data entry contains rid of a data record. In general, one I/O per data record!

Diagram:
- Index (Directs search)
- Data Entries ("Sequence set")
- Data Records
Unclustered B+ Tree Used for Sorting

- \( p \): average number of records per data page
  - Bigger than 10
- \( b_r \): data pages
- \( f \): (the size of a data entry)/ (size of a data record)
  - Usually 0.1

**Cost**

- # of data records: \( p \cdot b_r \)
- Approximate number of leaf pages: \( f \cdot b_r \)
- Total cost: \( (f+p) \cdot b_r \)
- Approximation: \( p \cdot b_r \)
## External Sorting vs. Unclustered Index (S.S.)

<table>
<thead>
<tr>
<th>br</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
- $M=1,000$ and block size $b=32$ for sorting
- $p=100$ is the more realistic value

- Cost of unclustered index appr.: $p \times br$
- Cost of sorting: calculate $F$, then $N1$ or $N2$, then cost
Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!

- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted runs of size $M$ (# buffer pages).
  - Later passes: merge runs.
  - # of runs merged at a time depends on $M$, and block size.
  - In practice, # of runs is rarely more than 2 or 3.

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

- Clustered B+ tree is good for sorting;

- Unclustered tree is usually very bad.