Why Sort in Databases

- Data can be requested in sorted order
  - `SELECT * FROM Foo ORDER BY bar`
- Loading data to an index. Bulk Loading in B+ Tree
- De-duplication (for example DISTINCT keyword in SQL). Remove duplicates by first ordering the records.
- Other query related operations such as joining multiple tables
- So, we will need to sort the keys, records for different needs
In-memory sorting vs. External Sorting

► You have seen sorting algorithms in your previous courses
► Why is it different for databases?
► The data can be much larger than the memory. In this case we have to think about the number of disk accesses.
► Running a sorting algorithm on disk will result in many record swaps that must be performed as disk operations. Will not work!
► We must design an algorithm which uses the limited primary memory wisely, and minimizes the disk accesses.
► Problem: Sort 1GB data with 1MB memory
Merge of Merge Sort

- Merge Sort algorithm is characterized by the merge operation
- Given two sorted lists, merge them to produce a single sorted list
- You can visualize the algorithm as merging two sorted deck of cards.
- Pick the smallest card (on top) from both decks.
- Since we know that the smaller card is not in the other deck, we can add it to the output deck.
Merge Sort Illustration
Merge Sort Illustration

4 6 7 9 10

2 3 5 8

1
Merge Sort Illustration
Merge Sort Illustration

4
6
7
9
10

5
8

1
2
3
Merge Sort Illustration
Merge Sort Illustration
Merge Sort Illustration
Merge Sort Illustration
General Idea

- Use the memory for sorting a part of the file
  - We will first consider sorting a page in memory
- Use the merge operation to merge runs until the whole file is sorted!
- The merge can be implemented for any two sublists of arbitrarily large sizes, as we only need the top of the decks
2-Way Sort

- Read a page, sort it, write it
  - Only one buffer page is used
  - Written sorted page is called as a run

- Pass 2, 3 ... etc:
  - 3 page memory buffer: 2 for reading runs, Merge & Write in one page
2-Way Merge Example

- Assume that we have 7 pages in disk.
- Each page can store 2 keys

Input File

Pass 0

Pass 1

Pass 2

In Pass 3, these two runs are merged, and the file is sorted!
2-Way Merge Analysis

- The number of passes needed for sorting a file of $2^k$ is $k$
- At each step we are merging two runs. So $\text{ceil}(\log_2 N)$, where $N$ is the number of pages in the file. If we add the initial Pass, $\text{ceil}(\log_2 N)$
- In each pass we have to read and write each page. So, the cost for a pass is $2N$
- The total cost of this procedure is $2N \times \text{ceil}(\log_2 N + 1)$
- So for our example, with 7 pages. We have 4 passes. At each pass we have $2 \times 7$ disk access. The total cost is 56 disk accesses.
How to do better?

- The complexity increases as the number of passes increases.
- Instead of merging two runs at a time, we will merge as much runs as it is possible
  - Number of runs that can fit in the memory
- Instead of 2 we use B pages to read to memory (e.g. B=4 pages)
  - Pass 0: Sort ceil(N/B) pages, where each page is of size B
  - Pass 1…k: Merge B-1 runs (1 page is used for output)
First Sort (Pass 0) B=4

After Pass 0
B-1 Way Merge

» After this step, we can merge B-1=3 different runs at the same time
» Algorithm for B-1 Way Merge is similar to two-way case:
  » At each step choose the smallest key
  » Write to output
Cost of External Merge Sort

- Number of passes:
- Cost = \(2N \times \text{(\# of passes)}\)
- \(1 + \left\lfloor \log_{B-1} \left( \frac{N}{B} \right) \right\rfloor\)

E.g., with 5 buffer pages, to sort 108 page file:
- 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
A note on complexity

- The asymptotic complexity of both algorithms is $O(N \log N)$
- The base of logarithm can be changed by the rule
  \[ \log_a b = \frac{\log_c b}{\log_c a} \]
  So a different base changes only the constant of the cost!
- However, when the base is $(B-1)$ the number of passes will drastically decrease
<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>
We have seen that performing a disk write or read for a continuous sequence of files is more efficient.

So, for example writing the whole file a page at a time will take more time than writing $F$ pages at a time.

Use $B-F$ pages for sorting, reserve $F$ blocks for writing.
Double Buffering

- While waiting to read or write a block, the CPU will be idle as we have no data to process.
- 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.
Sorting Records!

- Sorting has become a blood sport!
  - Parallel sorting is the name of the game ...

- Datamation: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - World record: 3.5 seconds
    - 12-CPU SGI machine, 96 disks, 2GB of RAM

- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?
Example

Assuming that our most general external sorting algorithm is used. For a file with 2,000,000 pages and 17 available buffer pages, answer the following

1. How many runs will you produce in the first pass?
2. How many passes will it take to sort the file completely?
3. What is the total I/O cost of sorting the file?
4. How many buffer pages do you need to sort the file completely in just two passes?
Answer

► How many runs are produced in Pass 0
  ► $\text{Ceil}(2000000/17) = 117648$ sorted runs.

► Number of passes required
  ► $\text{Ceil}(\log_{16} 117648) + 1 = 6$ passes.

► Total number of disk accesses
  ► $2 \times 2000000 \times 6 = 24000000$.

► How many Buffer pages do we need, to complete sort in two passes
  ► We have to produce less than equal to $B-1$ runs after first pass. So $\text{ceil}(N/B)$ must be less than equal to $B-1$. For $2 \times 10^6$ pages, if $B=10^3$ we have 2000 runs, $B=1415$ produces 1414 runs which can be merged in single pass.