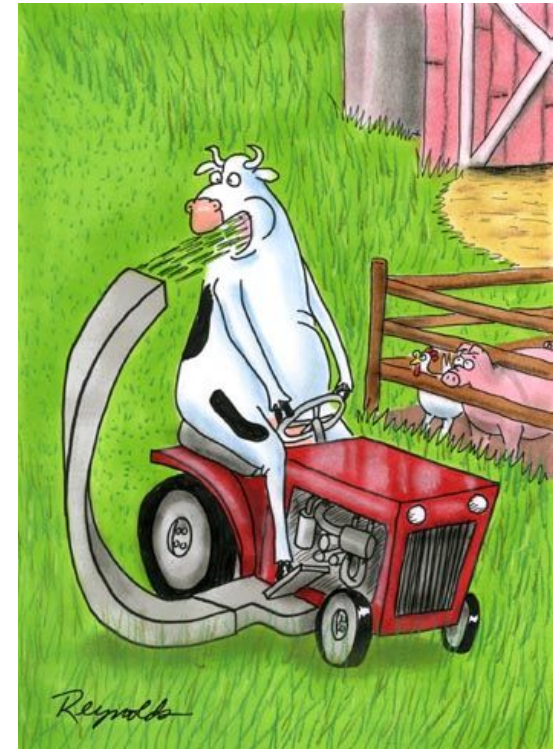
External Sorting

The due date for Problem Set #3 has been extended to 10/13. Problem Set #4 shoud be issued then as well.

Hope to get Midterm graded by next Tuesday

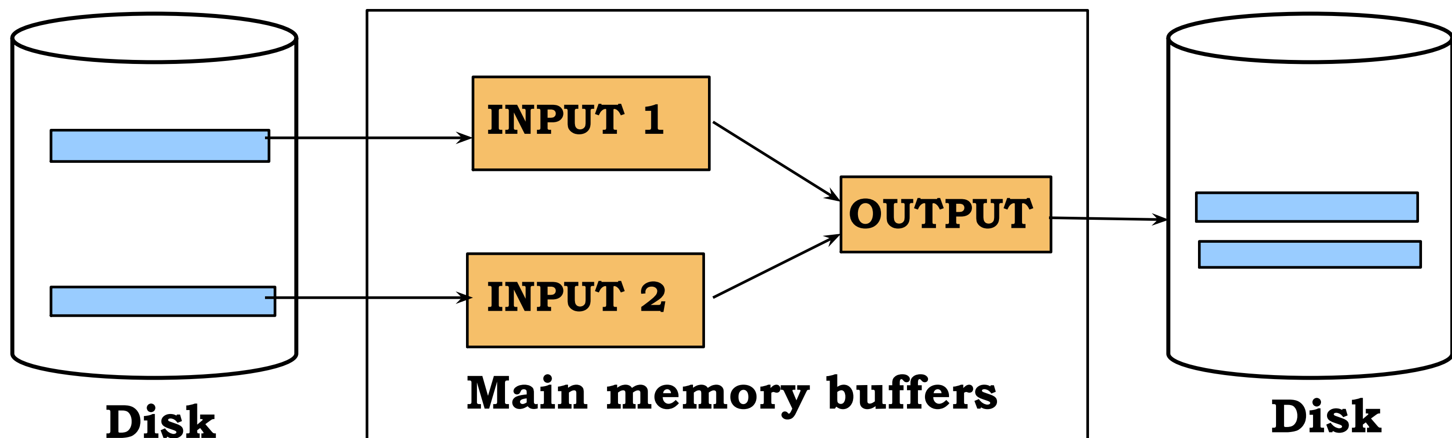


Why Sort?

- A classic problem in computer science!
- In databases there are advantages to requesting data in sorted order
  - gathers duplicates.
  - allows for efficient searches.
  - Sorting is first step in *bulk loading* B+ tree index.
  - *Sort-merge* join algorithm involves sorting.
- External Sort Problem:
  How to sort 20Gb of data with 1Gb of RAM.
- Why not let the OS handle it with virtual memory?

2-Way Sort: Requires 3 Buffers

- Pass 1: Read each page, sort it, write it back.
  - only one buffer page is used
- Pass 2, 3, …, N etc.:
  - Read 2 pages, merge them, and write 2 merged pages
  - Requires three buffer pages.



Two-Way External Merge Sort

- Each pass we read + write each page in file.
- N pages in the file => the number of passes

$$= \lceil \log_2 N \rceil + 1$$

- So toal cost is (2N = N reads + N writes):

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

- <u>*Idea:*</u> ***Divide and conquer:*** sort pages and merge

Input file: 3,4 | 6,2 | 9,4 | 8,7 | 5,6 | 3,1 | 2 | (empty)

PASS 0

1-page runs: 3,4 | 2,6 | 4,9 | 7,8 | 5,6 | 1,3 | 2 | (empty)

PASS 1

2-page runs: 2,3 4,6 | 4,7 8,9 | 1,3 5,6 | (empty) 2

PASS 2

4-page runs: 2,3 4,4 6,7 8,9 | (empty) 1,2 3,5 6

PASS 3

8-page runs: (empty) 1,2 2,3 3,4 4,5 6,6 7,8 9

General External Merge Sort

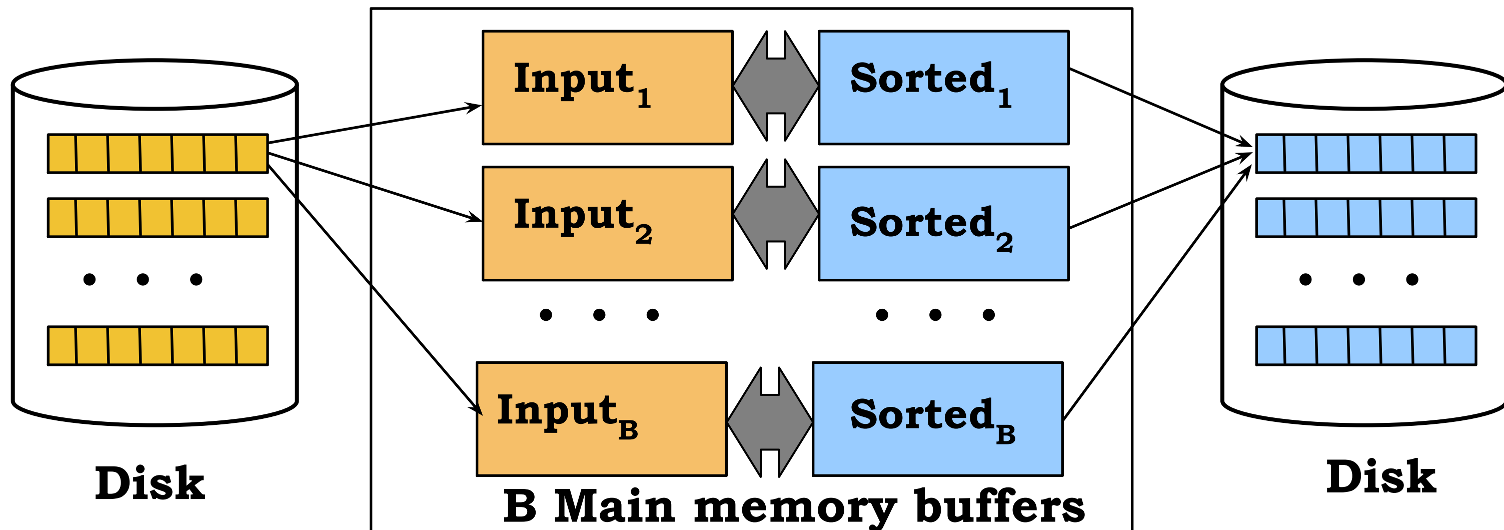
☞ ***More than 3 buffer pages. How can we utilize them?***

- Key Insight #1: We can merge more than 2 input buffers at a time… affects fanout base of log!
- Key Insight #2: The output buffer is generated incrementally, so only one buffer page is every needed for any size of *run*!
- 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 2, …, etc.: merge *B-1* runs, leaving one page for output.

General External Merge Sort

But we have more than 3 buffer-pool pages. How do we utilize them?

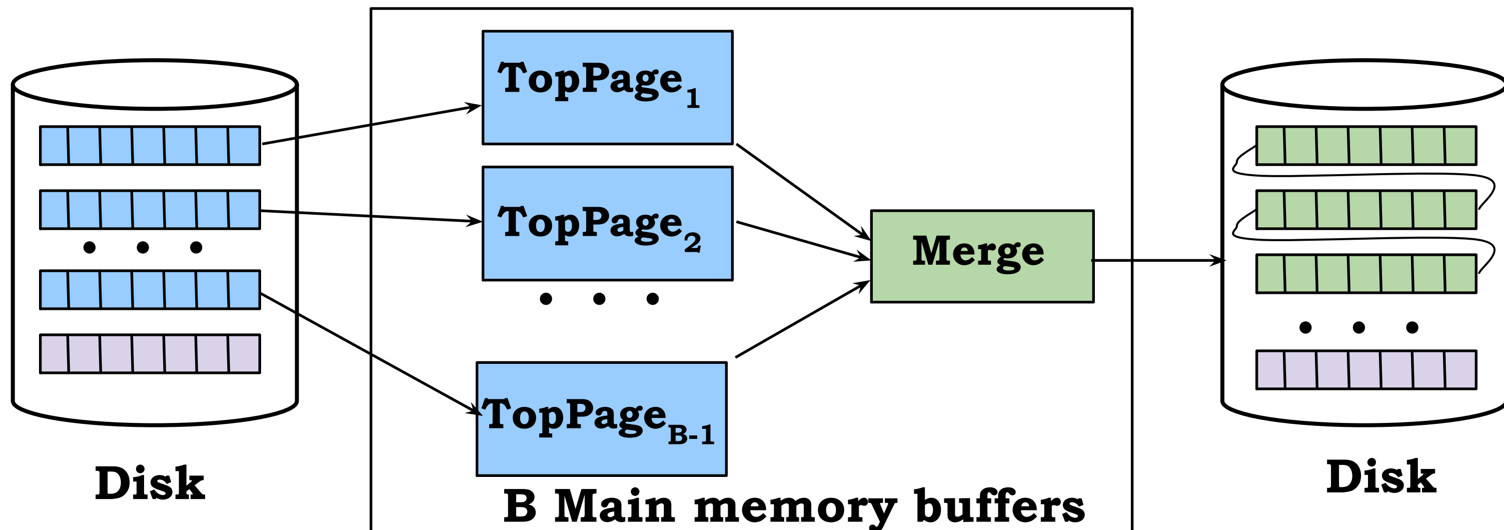
- To sort a file with *N* pages using *B* buffer pages:
  - Pass 0: read *B* pages. Produce $\lceil N / B \rceil$ sorted *runs* of *B* pages.



General External Merge Sort

But we have more than 3 buffer-pool pages. How do we utilize them?

- To sort a file with *N* pages using *B* buffer pages:
  - Pass 0: read *B* pages. Produce $\lceil N / B \rceil$ sorted *runs* of *B* pages.
  - Pass 1, …, etc.: merge *B-1* runs. Repeat.



Cost of External Merge Sort

- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Cost = 2N * (# of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: $\lceil 108 / 5 \rceil$ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: $\lceil 22 / 4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: $\lceil 6/4 \rceil$ = 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

$1 + \lceil \log_4 \lceil 108/5 \rceil \rceil = 1 + \lceil \log_4 (22) \rceil = 1 + 3 = 0,1,2,3$

108 * (1 read + 1 write) * 4 passes = 864 disk I/Os

Number External Sort Passes

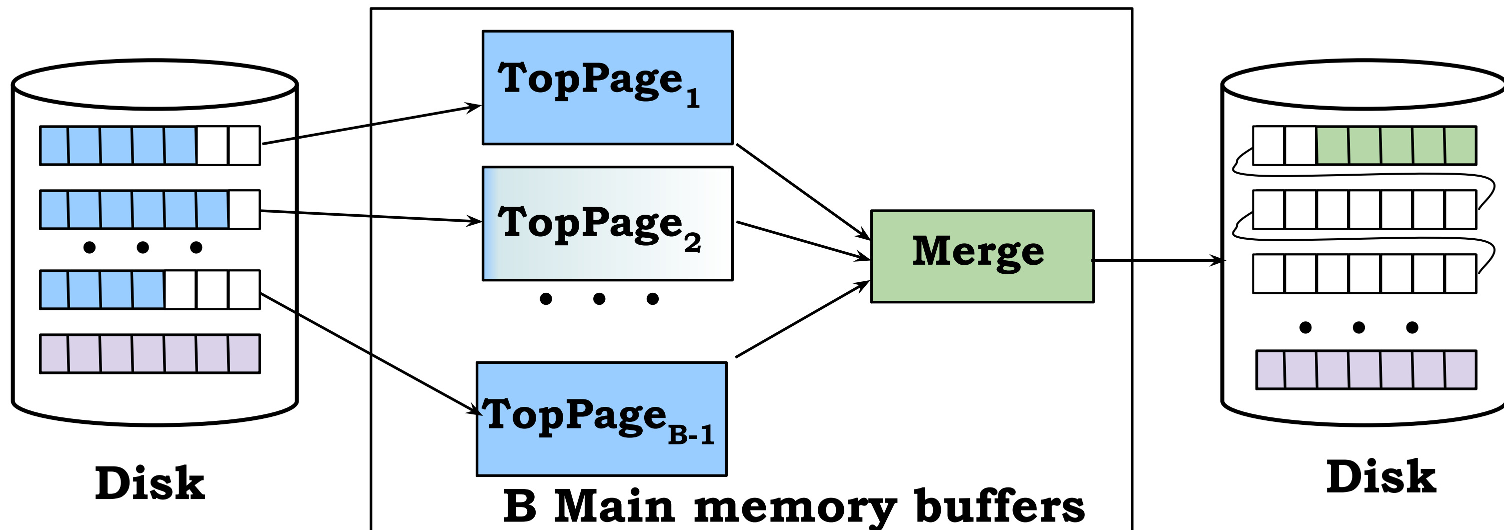
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
1,000,000,000	30	15	10	8	5	4

Internal Sort Algorithm

- Quicksort is a fast way to sort in memory.
  - Very fast on average
  - Worse case N^2 (i.e. bad pivots)
- Alternatives
  - Heap Sort, stable and always O(NlogN)
  - Merge Sort, same approach used in "out-of-core" sort but applied within a block recursively (low overhead)
  - Divides block into two halves, sorts each by dividing them recursively into two halves until there is only one item in the list. Then merges all of the "half-sized" lists while returning up the recursion.
- Another Problem… waiting to fill the buffer pool

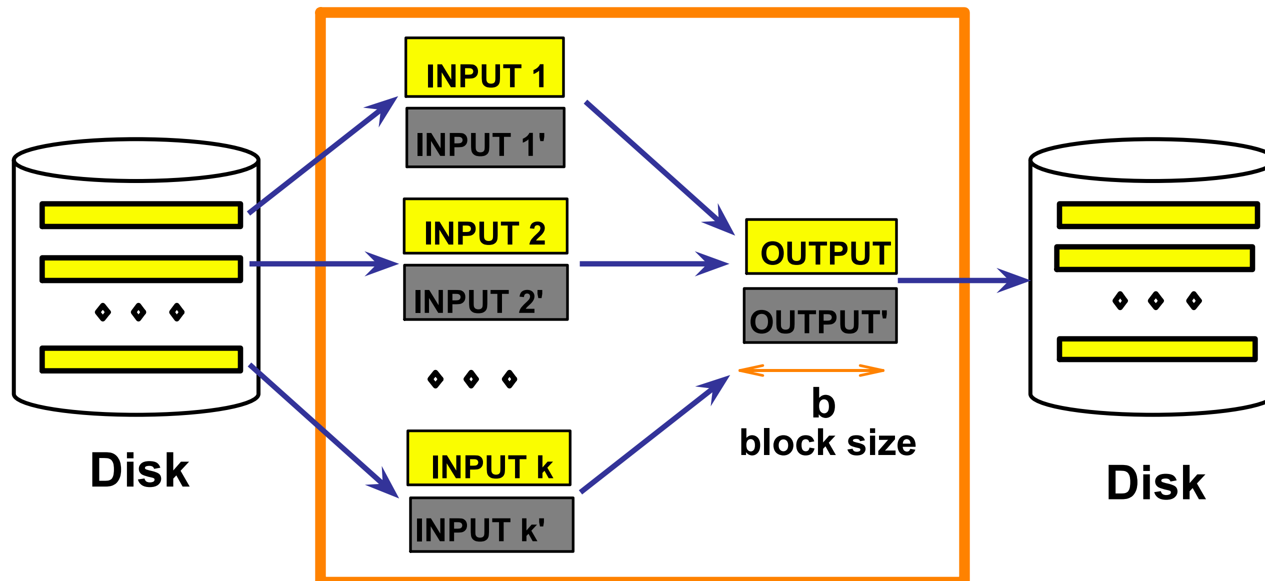
Sorting Stalls

- When a "top page" (the current top of the active run) empties, we have to ***wait*** for it to be refilled
- While waiting, we can't fill the merge ouput buffer using the other top pages, because the next value to be merged might come from the next block of the exhausted run



Double Buffering

- To reduce the waiting time for I/O request to complete, can *prefetch* the next page of the run into a "shadow block".
- Potentially, requires more passes; but, in practice, most files are *still* sorted in 2-3 passes, and waits are significant.
- We can also have a shadow block for the output



Sorting Records!

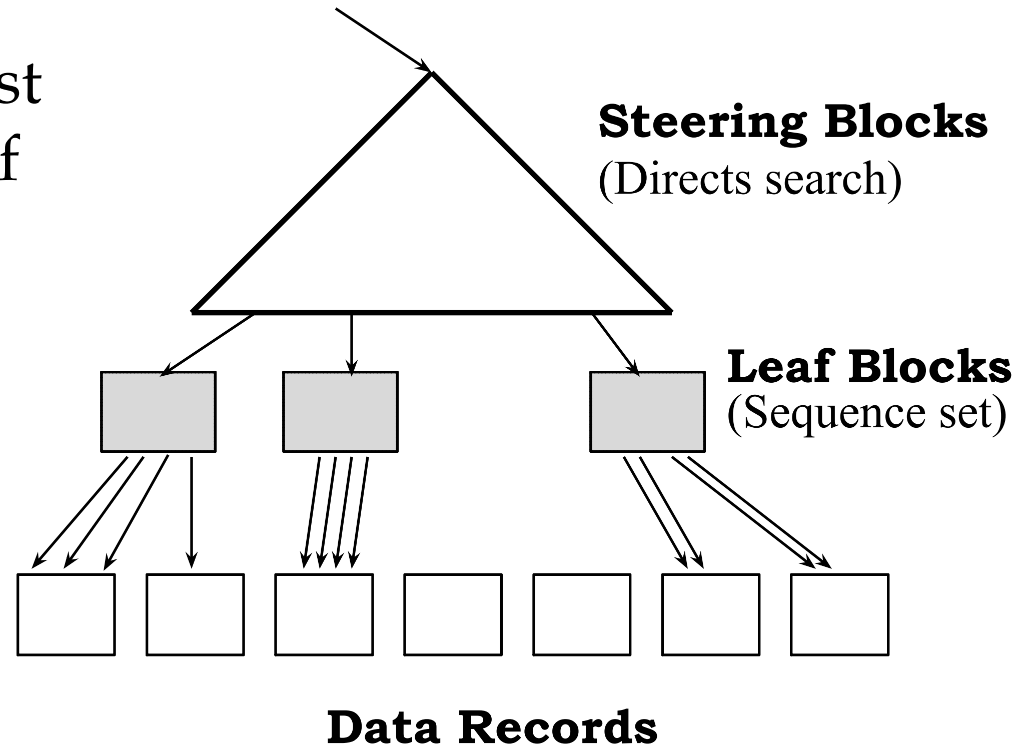
- Sorting is a blood sport!
  - Parallel external sorting is the name of the game ...
- 2019 Tencent Sort (Tencent Corp.)
  - Sort 100Tbyte of 100 byte records
  - Typical DBMS: > 10 days
  - World record: ***134 seconds***
    - 4x 10-core Power8 (PowerPC) CPUs
      Huawei ES3600P @ 2.9GHz with 512 nodes
    - Each node: 512 GB of RAM, and a 1.2TB NVMe SSD disk
- New benchmarks proposed:
  - Minute Sort: How much can you sort in 1 minute?
  - Cloud Sort: How many $ per TB sorted?
- Other ways to sort?

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 — ***Retriving records in order might revisit pages, could be slow***

Clustered B+ Tree Used for Sorting

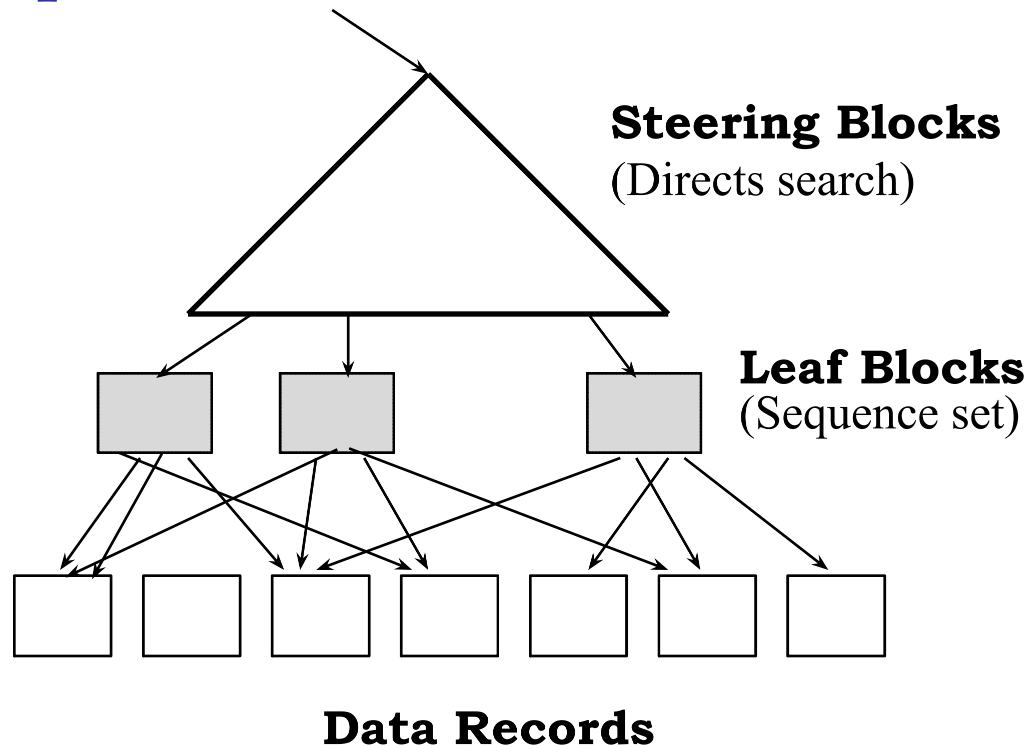
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.
- Introduces a some overhead extra for the "index" pages fetched



- *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

N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

- ***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 important; A DBMS often dedicates part of its buffer pool just for sorting!
- 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 runs rarely more than 2 or 3.

Summary, cont.

- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Alternative sorts
    - guaranteed N logN on worst case data
    - stable (ties retain their original order)
- 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.

Next Time

More on Query Evaluation