CS222/CS122C: Principles of Data Management

UCI, Fall 2019

Notes #09

External Sorting

Disk based

Instructor: Chen Li

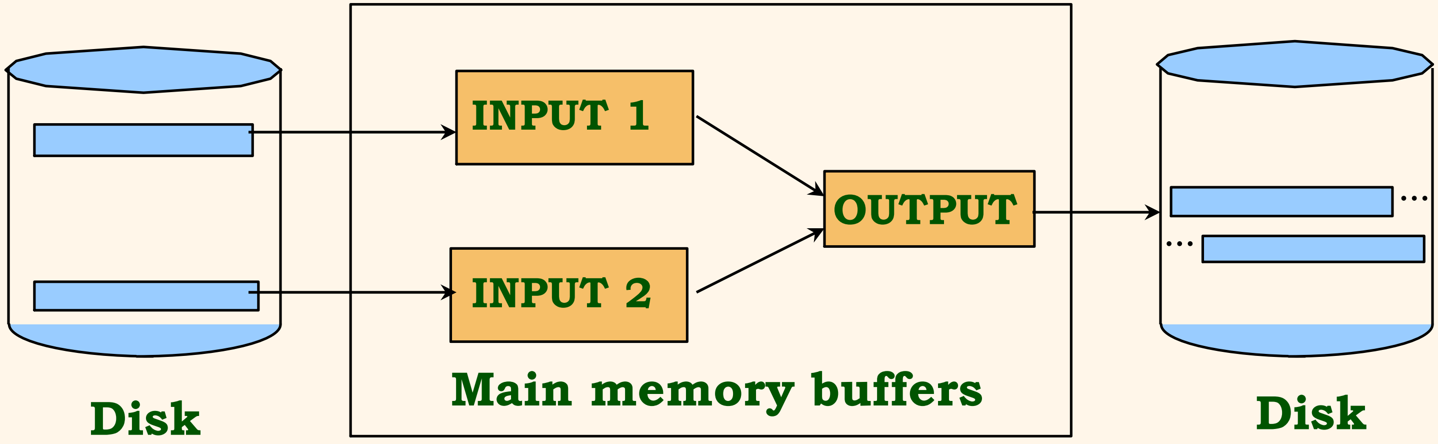
Why Sort?

- A classic problem in computer science! (☺)
- Data sometimes requested in sorted order.
  - E.g., find students in decreasing *gpa* order.
- Sorting is first step in *bulk loading* B+ tree index.
- Sorting useful for eliminating *duplicate copies* (i.e., SELECT DISTINCT) in a collection of records. (Why?)
- *Sort-merge* join algorithm involves sorting.
- Problem: sort 1 TB of data with 1 GB of RAM.
  - *Q*: Why not just use virtual memory?

* Access heap file using sorted RIDs from B+ tree.

2-Way Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it out
  - only one buffer page is used
- Pass 2, 3, …, etc.: Read and merge pairs of runs
  - three buffer pages are used:



Sorting N=2^k Pages of Data

- Pass 0:
  - Read, sort, write → 2^k 1-page runs (subfiles)
- Pass 1:
  - Read+merge 1-page pairs, write → 2^{k-1} 2-page runs
- Pass 2:
  - Read+merge 2-page pairs, write → 2^{k-2} 4-page runs

…

- Pass *k*-1:
  - Read+merge 2^{k-2} -page pairs, write → 2 2^{k-1} -page runs
- Pass *k*:
  - Read+merge 2^{k-1}-page pairs, write → 1 2^k –page result

Two-Way External Merge Sort

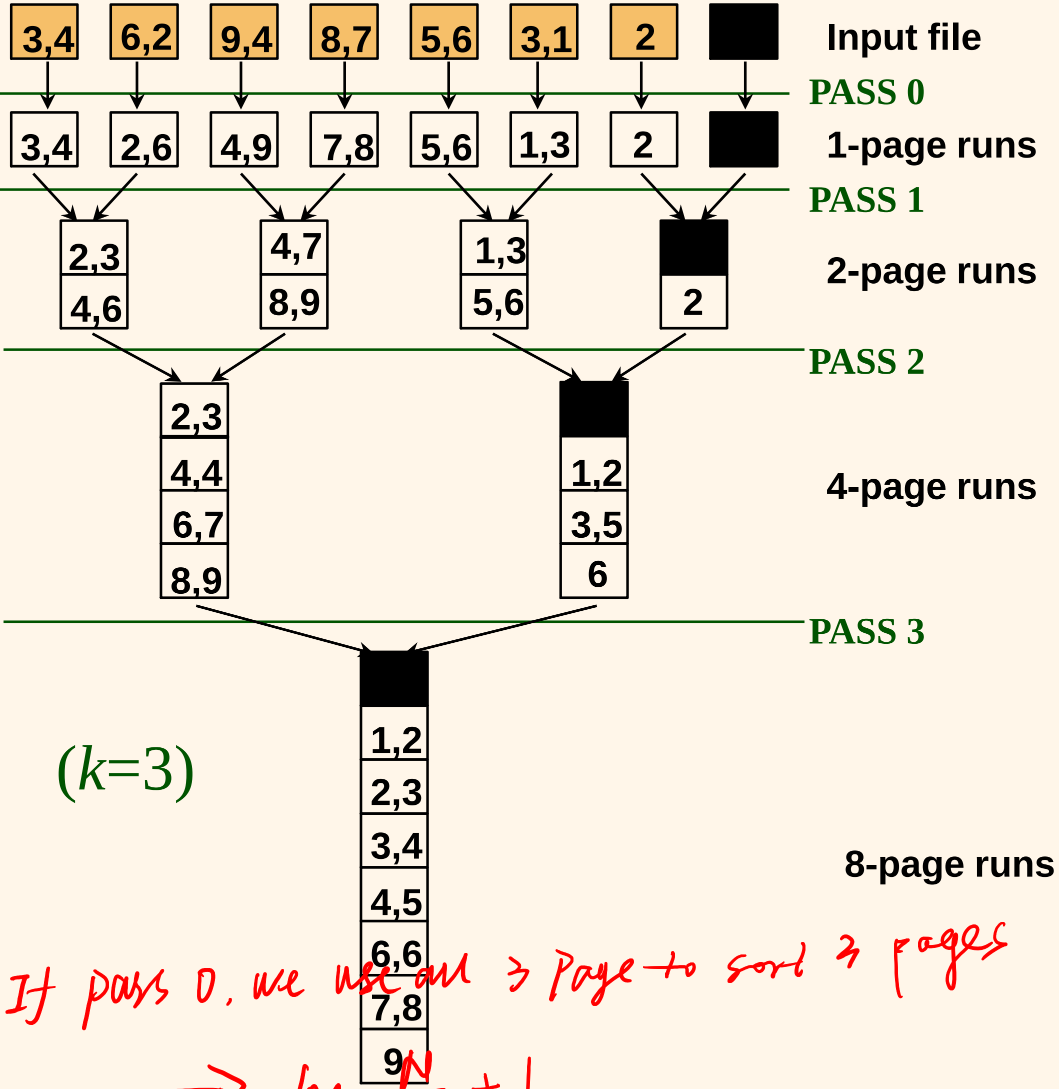
- Each pass we read + write each page in file.
- N pages in the file => the number of passes

$$=\lceil \log_2 N \rceil + \boxed{1} \rightarrow \text{sort the page}$$

- So total I/O cost is:

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

- *Idea*: ***Divide and conquer:*** sort subfiles and merge
- *Q*: See any room to do better, w/just 3 pages?



(*k*=3)

If pass 0, we use all 3 page to sort 3 pages

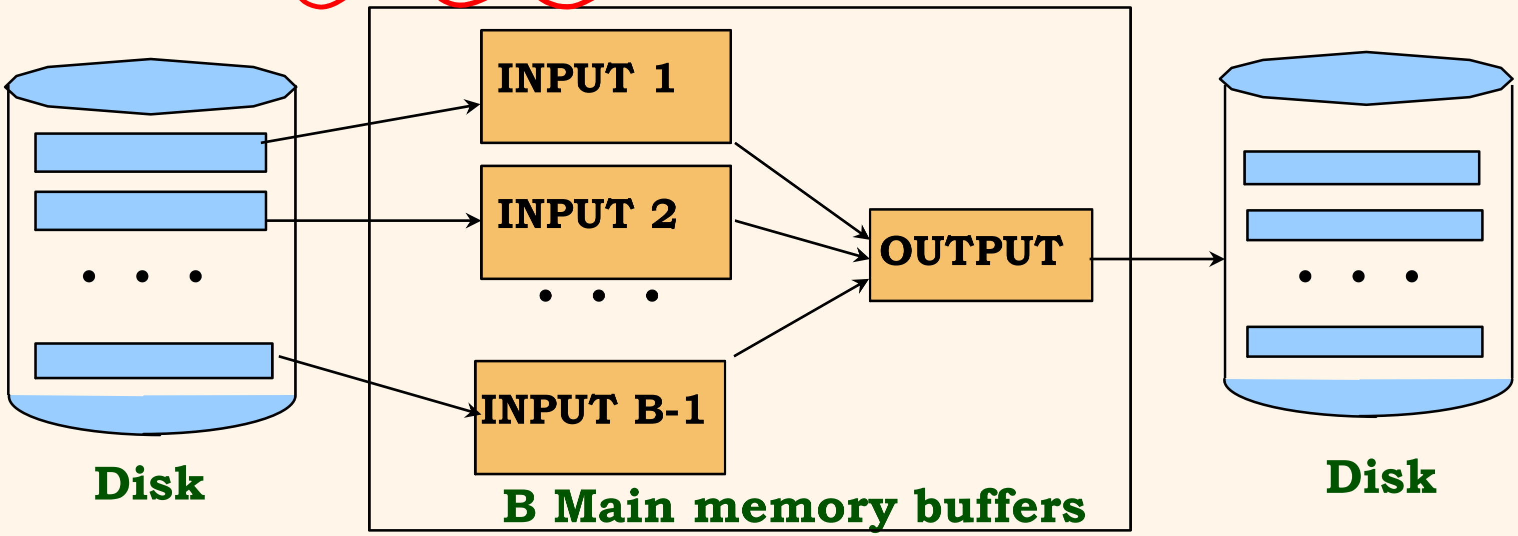
$$\rightarrow \log_2 \frac{N}{3} + 1$$

General External Merge Sort

Read B pages
use quick sort,
and flush back
to disk
No output page is
Pass 0

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. *(Actually B-1 instead of B w/variable-length records.)*
  - Pass 2, …, etc.: merge *B-1* runs.



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)

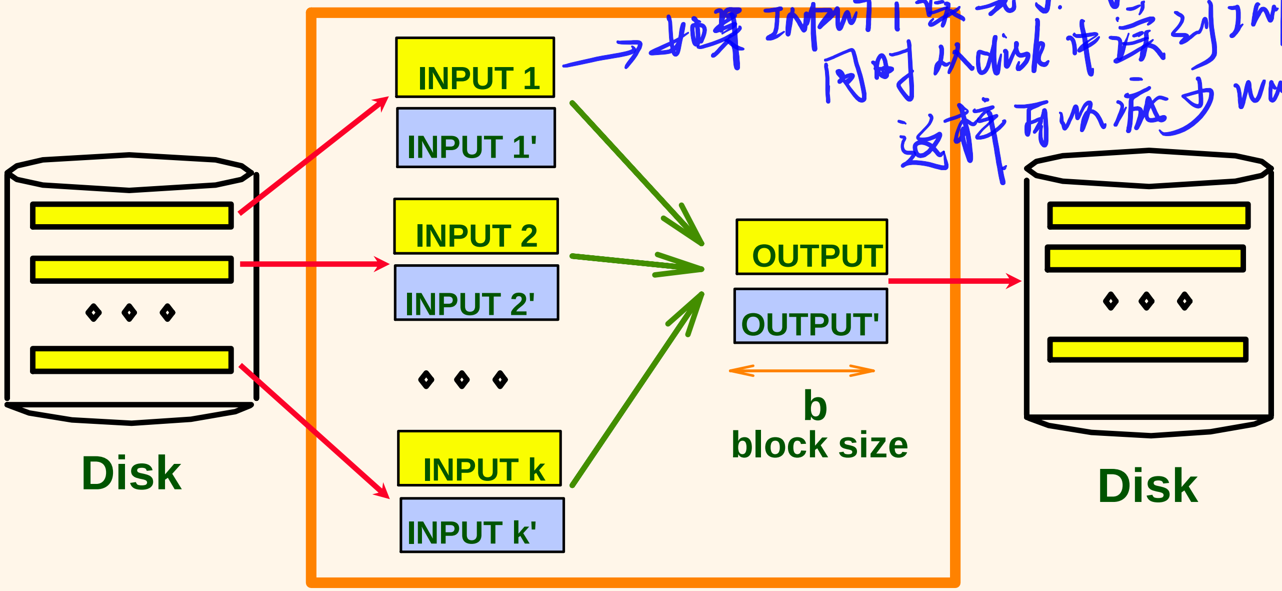
    In memory sort B pages
  - Pass 1: $\lceil 22/4 \rceil$ = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Number of Passes of External Sort

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
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

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.



→ 如果 INPUT 1 读完了. 可以直接读 INPUT 1' 同时从 disk 中读到 INPUT 1. 这样可以减少 wait time.

B main memory buffers, k-way merge

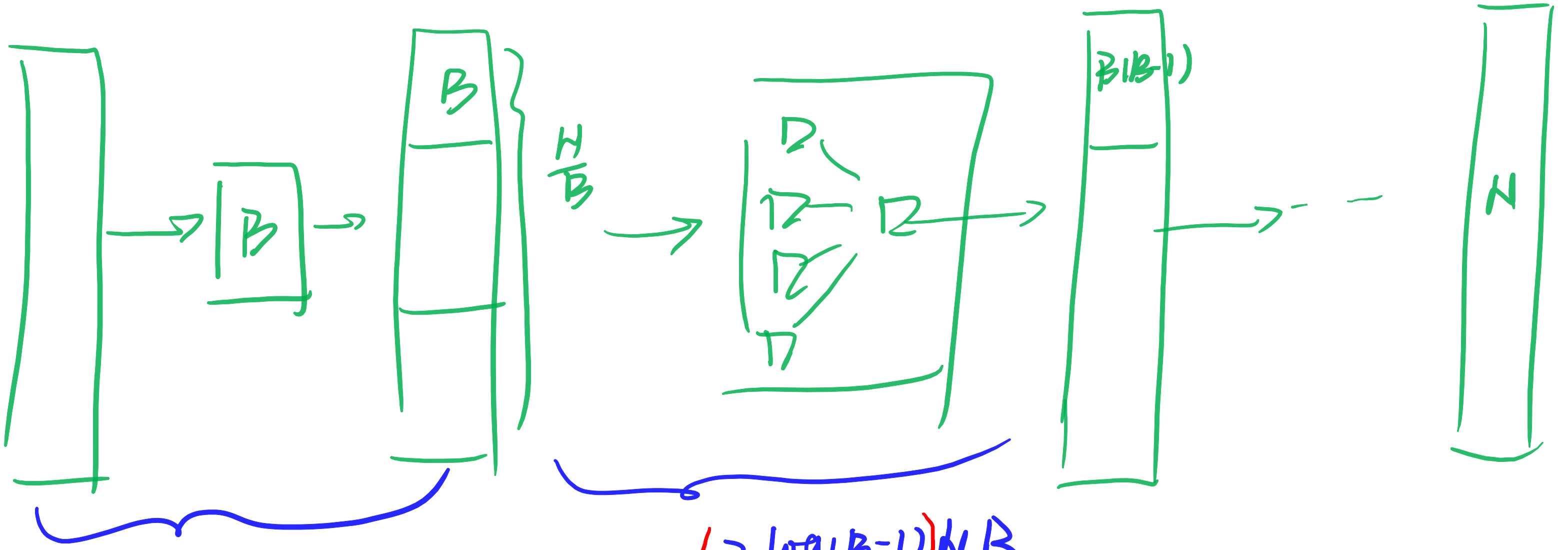
Sorting Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produce sorted ***runs*** of size ***B*** (# buffer pages), or of size ***B-1*** if we are handling variable-length records. Passes > 0: ***Merge*** runs (until just one run is produced).
  - # of runs merged at a time depends on ***B*** and ***block size***.
  - Larger block size means less I/O cost per page of data.
  - Larger block size means fewer runs merged per step.
  - In practice, # of passes needed rarely more than 2 or 3.

SQL: called SPJ: select

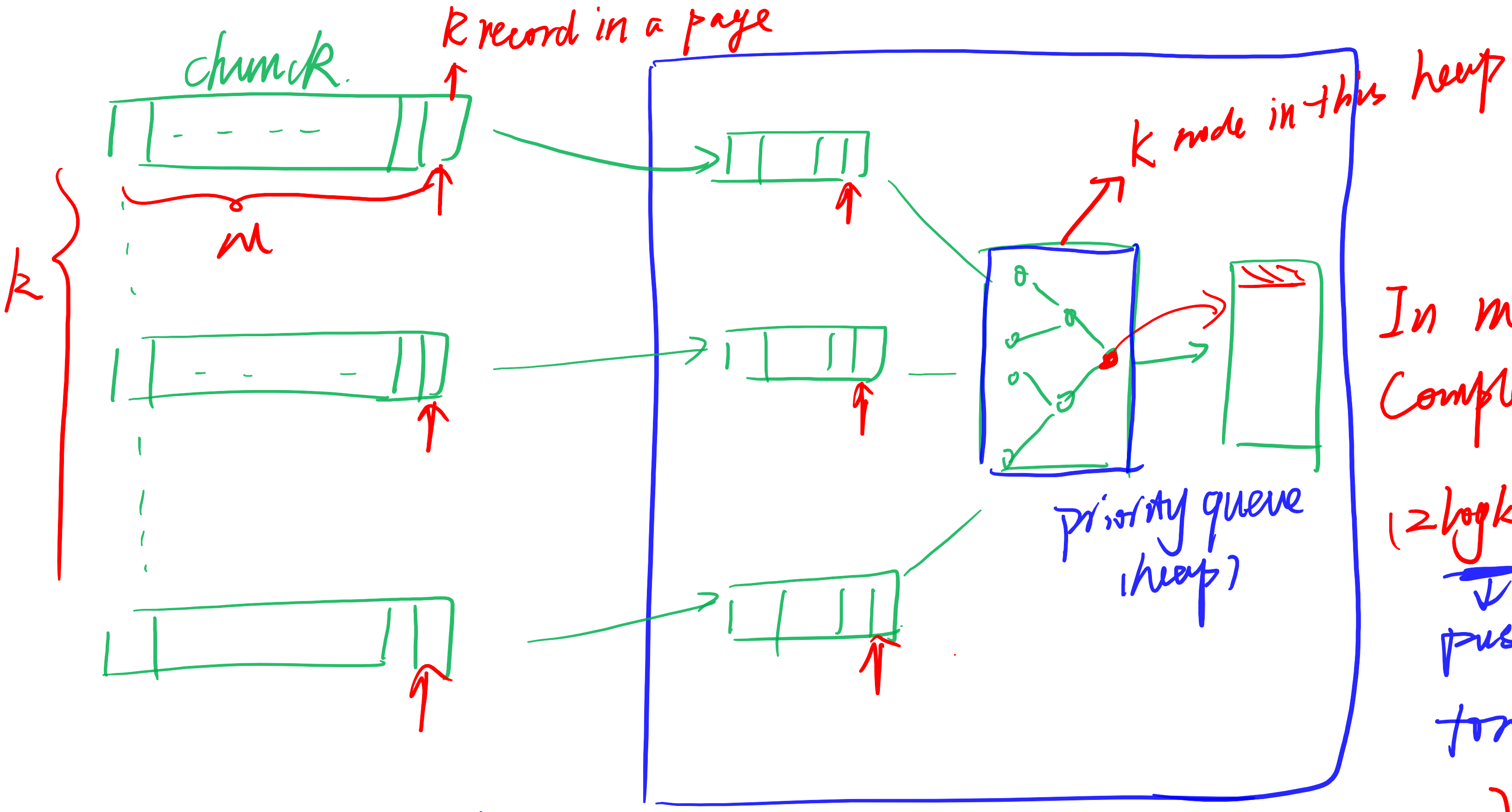
selection
project
join.

From
where.
Group by
having
order by



$$BR\,\underbrace{\log(B\cdot R)}_{\Downarrow\ \text{in-memory sort for B pages}}\cdot\frac{N}{B}$$

$$\underbrace{(2\log(B-1))}_{\rightarrow\ \text{push/pop time for heap}}NR$$



In memory
Complexity

$(2\log k)\ KMR$

↓

push and pop
for one record

↓

$2\log(B-1)\ NR$

- Build the initial heap
- keep a cursor with one input page
- use pop / push to go through each page.
- whenever the output page is full, flush it to disk
- if one input page is exhausted, read next page

Q: for N page, what's the minimum # of pages that can sort N pages using 2 iteration

(pass 0, pass 1)

pass 0 → in-memory sort

pass 1 → merge sort

$$\left\{ \begin{array}{l} \frac{N}{B} \le B-1 \\ \text{or} \\ N \le B(B-1) \end{array} \right.$$