CompSci 516

Data Intensive Computing Systems

Lecture 7

External Sorting

Instructor: Sudeepa Roy

Announcements

- Project proposal informal email due today
  - September 21 (Wednesday)
  - A few lines on the project + group members
- Homework 2:
  - due on Oct 12 (the same day as the midterm)
  - finish early -- should be ongoing!
  - remember to turn off instances
- Midterm
  - practice problems will be posted periodically

Reading Material

- (Complete Index from Lecture 5-6)

- [RG]
  - External sorting: Chapter 13
- [GUW]
  - Chapter 14

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

External Sorting

Why Sort?

- A classic problem in computer science
- Data requested in sorted order
  - e.g., find students in increasing gpa order
- Sorting is first step in bulk loading B+ tree index
- Sorting useful for eliminating duplicate copies in a collection of records
- Sort-merge join algorithm involves sorting
- Problem: sort 1Gb of data with 1Mb of RAM
  - need to minimize the cost of disk access

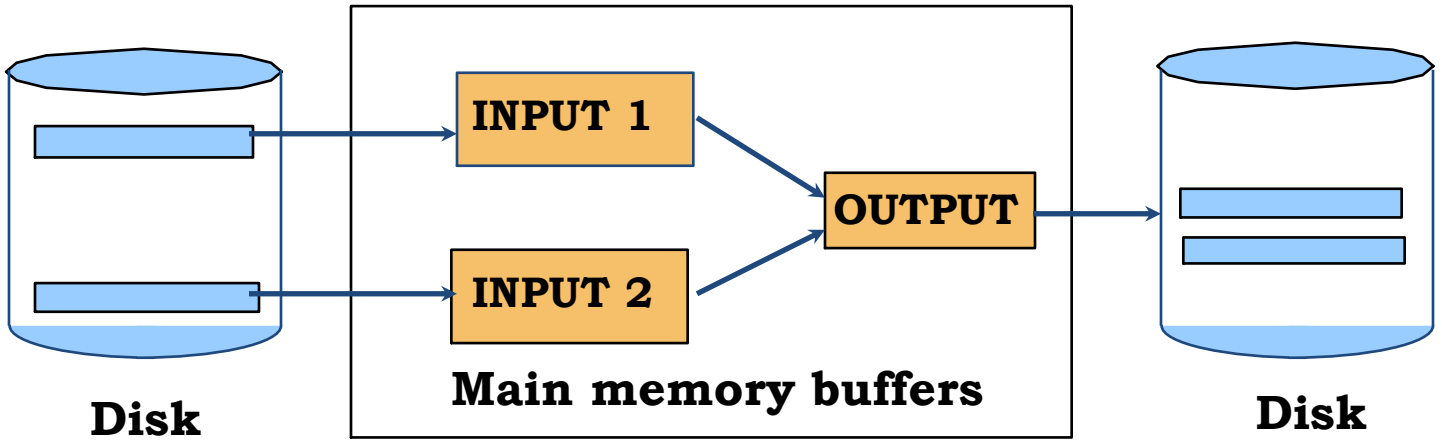
quick review of mergesort on whiteboard

A simple 2-way sort

- Not too practical, but useful to learn basic concepts for external sorting
- Utilizes only 3 pages of main memory
- Several sorted sub-files are generated at intermediate steps
- Each sorted sub-file is called a run
  - each run can contain multiple pages

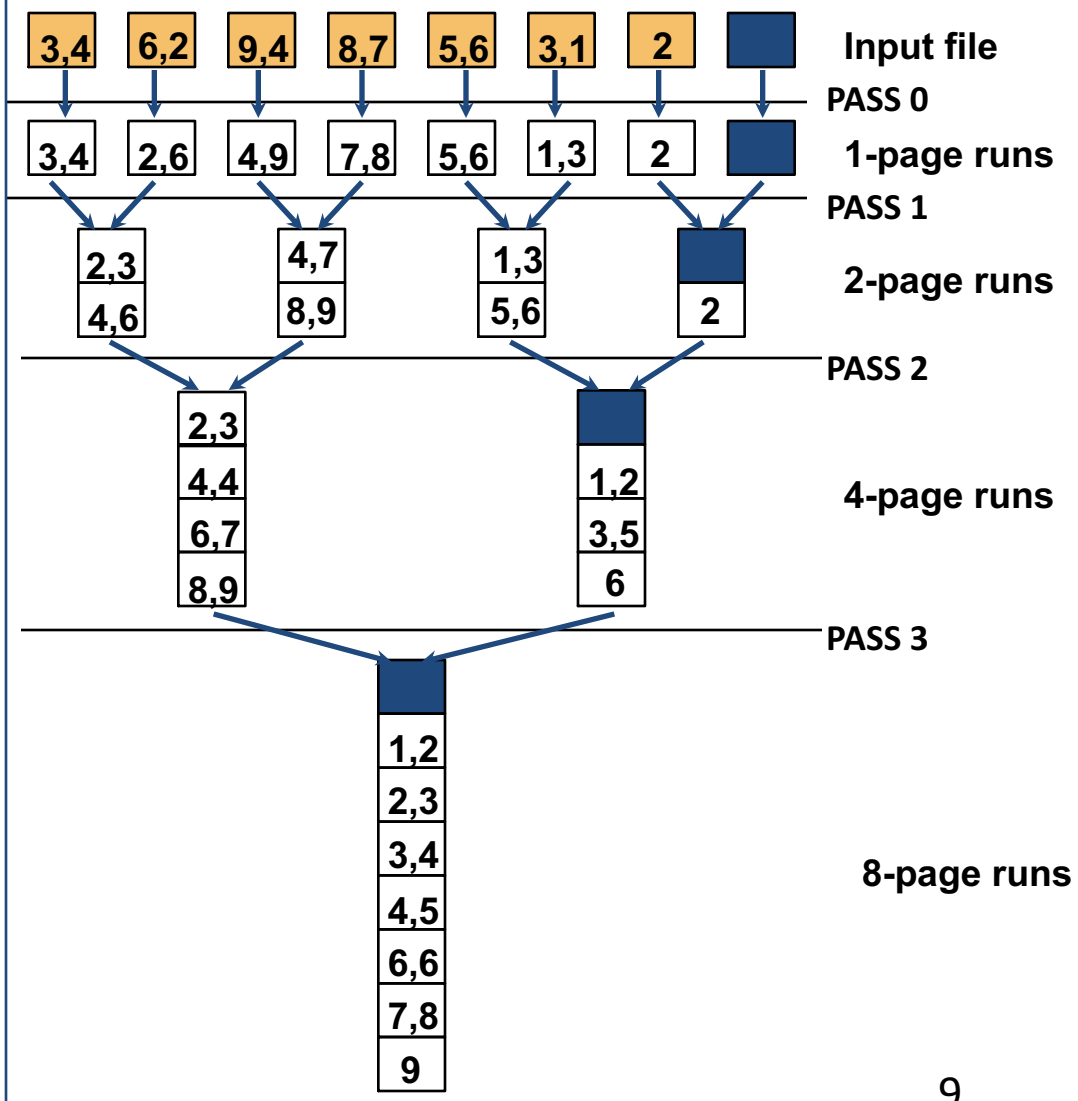
2-Way Sort: Requires 3 Buffers

- Suppose N = 2^k pages in the file
- Pass 0: Read a page, sort it, write it.
  - repeat for all 2^k pages
  - only one buffer page is used
- Pass 1:
  - Read two pages, sort (merge) them using one output page, write them to disk
  - repeat 2^{k-1} times
  - three buffer pages used
- Pass 2, 3, 4, ….. continue



Two-Way External Merge Sort

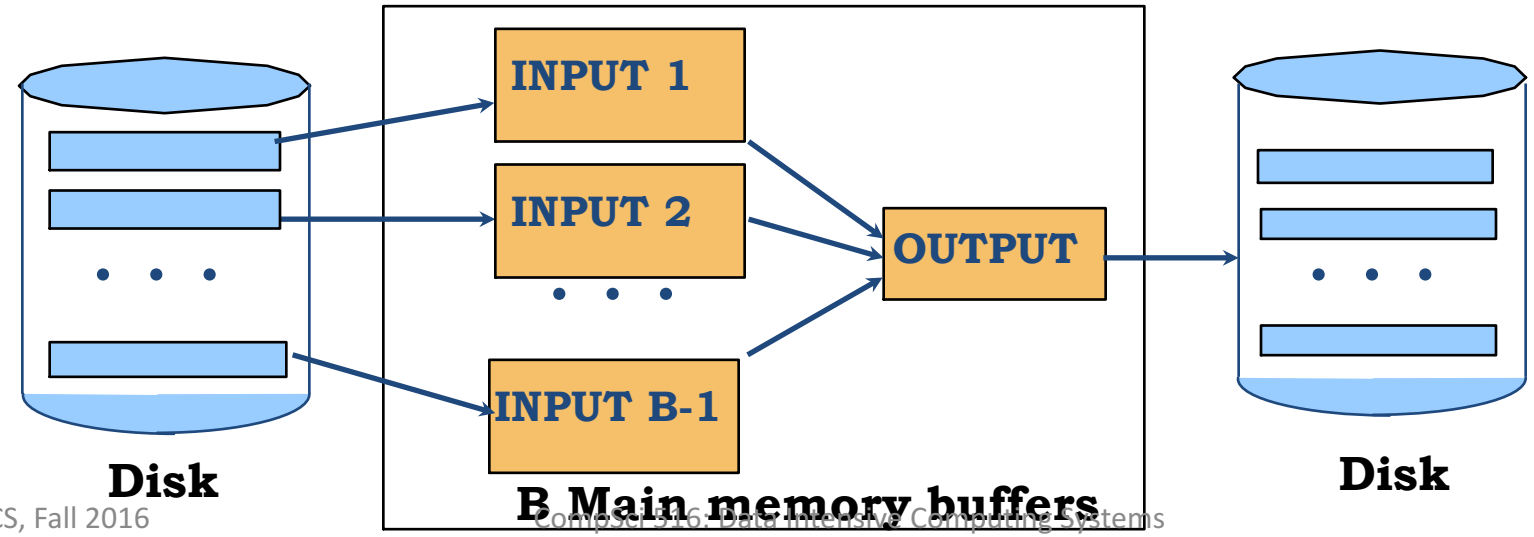
- Each pass we read + write each page in file.
- N pages in the file
- => the number of passes
$$= \lceil \log_2 N \rceil + 1$$
- So toal cost is:
$$2N\left(\lceil \log_2 N \rceil + 1\right)$$
- Idea: Divide and conquer: sort subfiles and merge



General External Merge Sort

- Suppose we have 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 [N/B] sorted runs of B pages each.
  - Pass 1, 2, …, etc.: merge B-1 runs to one output page
    - keep writing to disk once the output page is full



Cost of External Merge Sort

- Number of passes:1 + $\lceil \log_{B-1}[N/B] \rceil$
- Cost = 2N * (# of passes) – why 2 times?
- E.g., with 5 buffer pages, to sort 108 page file:
- Pass 0: sorting 5 pages at a time
  - [108/5] = 22 sorted runs of 5 pages each (last run is only 3 pages)
- Pass 1: 4-way merge
  - [22/4] = 6 sorted runs of 20 pages each (last run is only 8 pages)
- Pass 2: 4-way merge
  - (but 2-way for the last two runs)
  - [6/4] = 2 sorted runs, 80 pages and 28 pages
- Pass 3: 2-way merge (only 2 runs remaining)
  - Sorted file of 108 pages

Number of Passes of External Sort

High B is good, although CPU cost increases

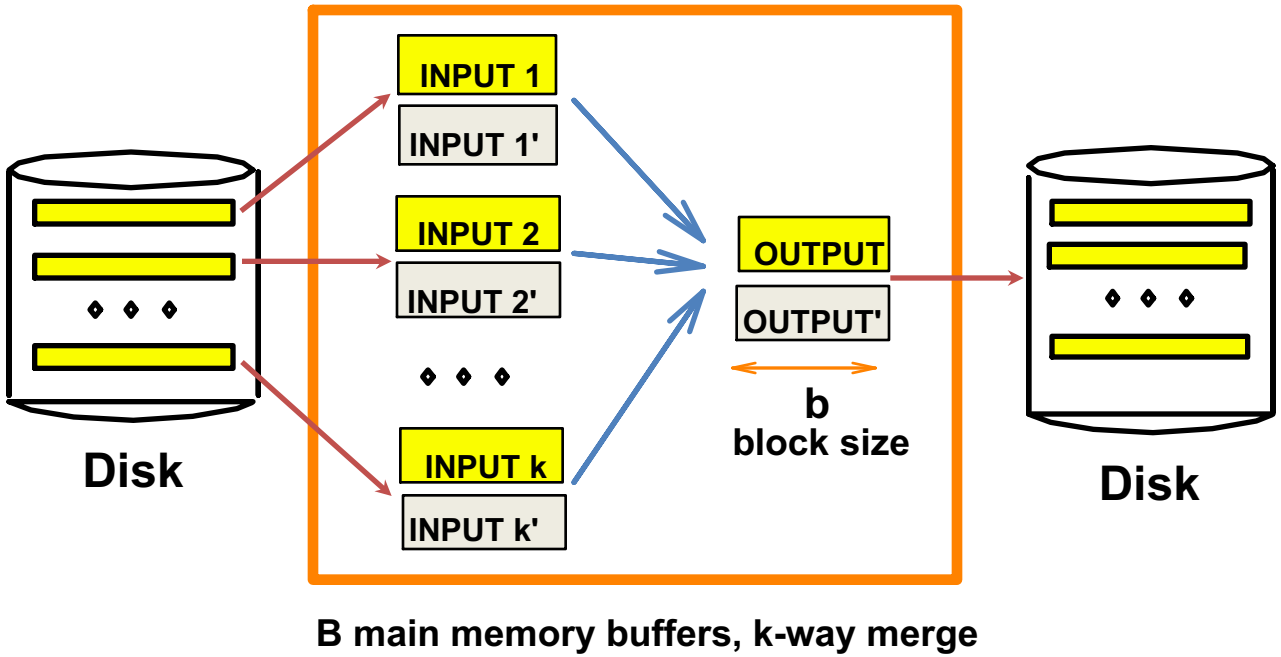
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

I/O for External Merge Sort

- If 10 buffer pages
  - either merge 9 runs at a time with one output buffer
  - or 8 runs with two output buffers
- If #page I/O is the metric
  - goal is minimize the #passes
  - each page is read and written in each pass
- If we decide to read a block of b pages sequentially
  - Suggests we should make each buffer (input/output) be a block of pages
  - But this will reduce fan-out during merge passes
    - i.e. not as many runs can be merged again any more
  - In practice, most files still sorted in 2-3 passes

Double Buffering

- To reduce CPU wait time for I/O request to complete, can prefetch into `shadow block'.

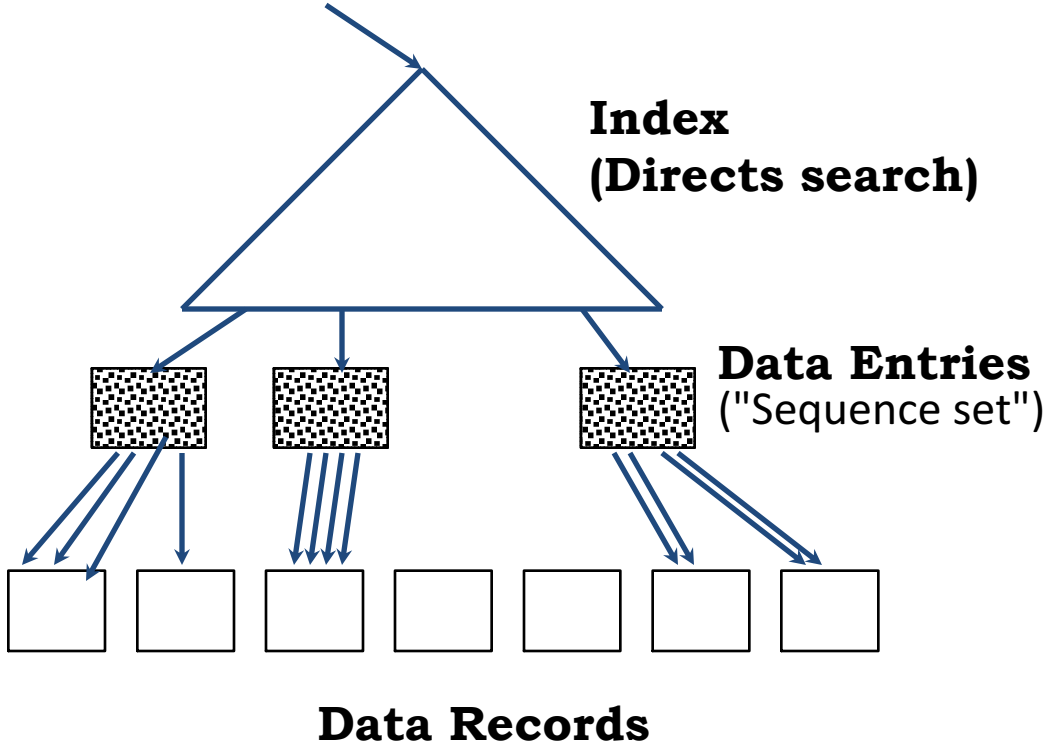


Using B+ Trees for Sorting

- Scenario: Table to be sorted has B+ tree index on sorting column(s)
- Idea: Can retrieve data entries (then 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

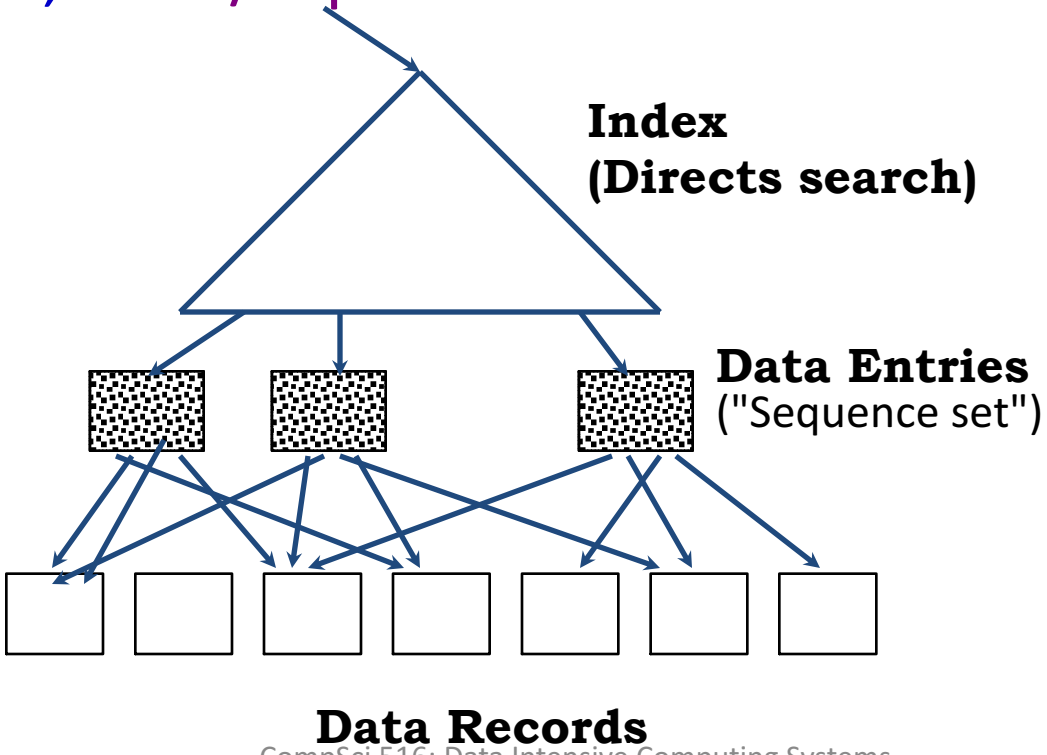
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once



☛ *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!



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 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