CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos – A. Pavlo

Lecture#12: External Sorting

CMU SCS

Today's Class

- Sorting Overview
- Two-way Merge Sort
- External Merge Sort
- Optimizations
- B+trees for sorting

CMU SCS

Why do we need to sort?

CMU SCS

Why do we need to sort?

- **SELECT...ORDER BY**
  - e.g., find students in increasing *gpa* order
- *Bulk loading* B+ tree index.
- *Duplicate elimination* (**DISTINCT**)
- **SELECT...GROUP BY**
- *Sort-merge* join algorithm involves sorting.

CMU SCS

Why do we need to sort?

- What do we do if the data that we want to sort is larger than the amount of memory that is available to the DBMS?
- What if multiple queries are running at the same time and they all want to sort data?
- *Why not just use virtual memory?*

CMU SCS

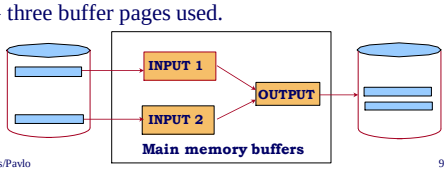
Overview

- Files are broken up into ***N*** pages.
- The DBMS has a finite number of ***B*** fixed-size buffers.
- *Let's start with a simple example…*

CMU SCS

Two-way Merge Sort

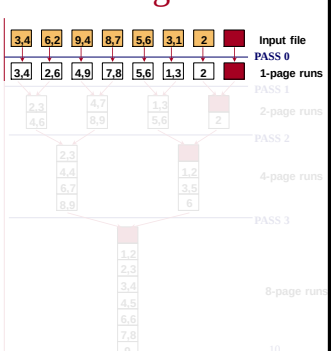
- **Pass 0**: Read a page, sort it, write it.
  - only one buffer page is used
- **Pass 1,2,3,…**: requires 3 buffer pages
  - merge pairs of **runs** into runs twice as long
  - three buffer pages used.



CMU SCS

Two-way External Merge Sort

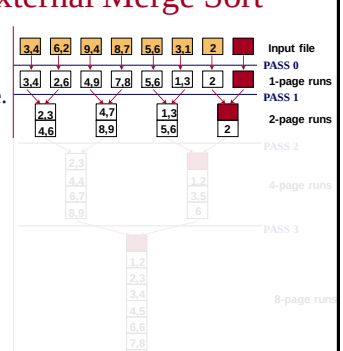
- Each pass we read + write each page in file.



CMU SCS

Two-way External Merge Sort

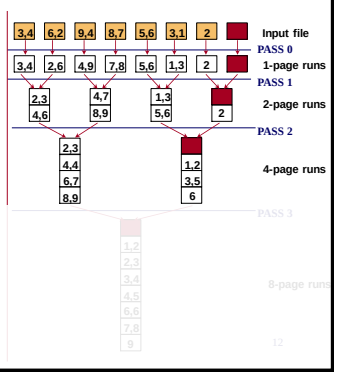
- Each pass we read + write each page in file.



CMU SCS

Two-way External Merge Sort

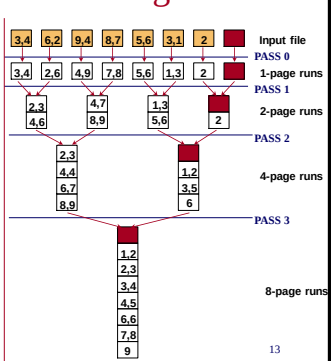
- Each pass we read + write each page in file.



CMU SCS

Two-way External Merge Sort

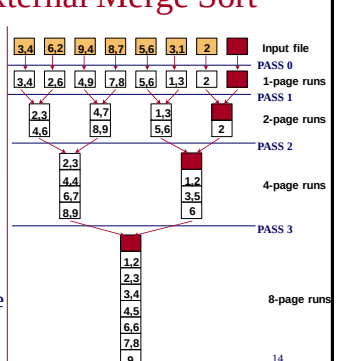
- Each pass we read + write each page in file.



CMU SCS

Two-way External Merge Sort

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



CMU SCS

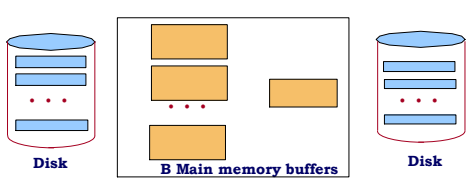
Two-way External Merge Sort

- This algorithm only requires three buffer pages.
- Even if we have more buffer space available, this algorithm does not utilize it effectively.
- *Let's look at the general algorithm…*

CMU SCS

General External Merge Sort

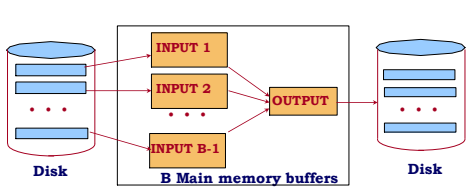
- ***B>3*** buffer pages.
- How to sort a file with ***N*** pages?



CMU SCS

General External Merge Sort

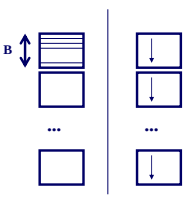
- **Pass 0**: Use ***B*** buffer pages. Produce $\lceil N / B \rceil$ sorted runs of ***B*** pages each.
- **Pass 1,2,3,…**: Merge ***B-1*** runs.



CMU SCS

Sorting

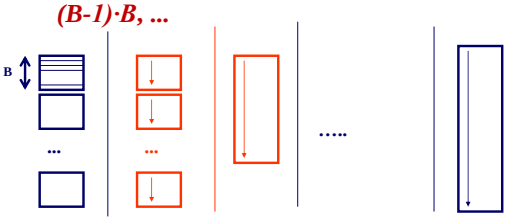
- Create sorted runs of size ***B*** (how many?)
- Merge them (how?)



CMU SCS

Sorting

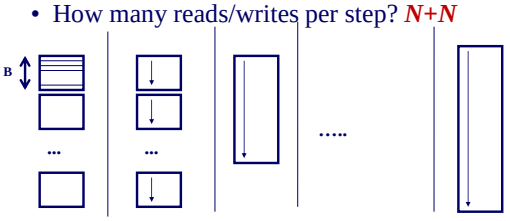
- Create sorted runs of size ***B***
- Merge first ***B-1*** runs into a sorted run of ***(B-1)·B, ...***



CMU SCS

Sorting

- How many steps we need to do? 'i', where ***B·(B-1)^i > N***
- How many reads/writes per step? ***N+N***



CMU SCS

Cost of External Merge Sort

- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- ***Cost = 2N·(# of passes)***

CMU SCS

Example

- Sort 108 page file with 5 buffer pages:
  - **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:** 2 sorted runs, 80 pages and 28 pages
  - **Pass 3:** Sorted file of 108 pages

Formula check: $\lceil \log_4 22 \rceil = 3 \ldots + 1 \rightarrow 4$ passes ✓

CMU SCS

of Passes of External Sort

Cost = 2N·(# of 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
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

CMU SCS

Today's Class

- Sorting Overview
- Two-way Merge Sort
- External Merge Sort
- Optimizations
- B+trees for sorting

CMU SCS

Optimizations

- Which internal sort algorithm should we uses for **Phase 0**?
- How do we prevent the DBMS from blocking when it needs input?

CMU SCS

Internal Sort Algorithm

- **Quicksort** is a fast way to sort in memory.
- But we get ***B*** buffers, and produce one run of length ***B*** each time.
- Can we produce longer runs than that?

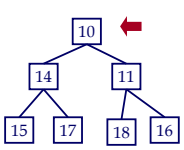
CMU SCS

Heapsort

- Alternative sorting algorithm (a.k.a. "replacement selection")
- Produces runs of length ~ ***2·B***
- Clever, but **not** implemented, for subtle reasons: tricky memory management on variable length records

CMU SCS

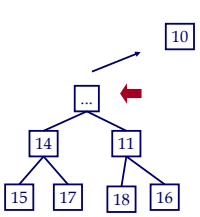
Reminder: Heapsort



pick smallest, write to output buffer:

CMU SCS

Reminder: Heapsort

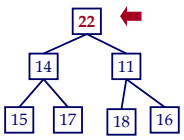


pick smallest, write to output buffer:

CMU SCS

Reminder: Heapsort

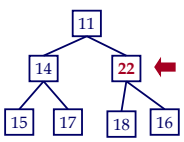
get next key; put at top and 'sink' it



CMU SCS

Reminder: Heapsort

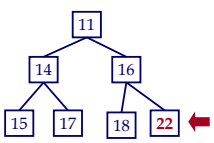
get next key; put at top and 'sink' it



CMU SCS

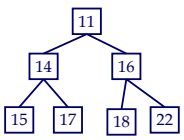
Reminder: Heapsort

get next key; put at top and 'sink' it



CMU SCS

Reminder: Heapsort



When done, pick top (= smallest) and output it, if 'legal' (ie., >=10 in our example

This way, we can keep on reading new key values (beyond the B ones of quicksort)

CMU SCS

Blocked I/O & Double-buffering

- So far, we assumed random disk access.
- The cost changes if we consider that runs are written (and read) sequentially.
- What could we do to exploit it?

CMU SCS

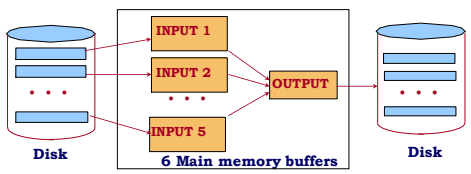
Blocked I/O & Double-buffering

- So far, we assumed random disk access.
- The cost changes if we consider that runs are written (and read) sequentially.
- What could we do to exploit it?
  - **Blocked I/O:** exchange a few r.d.a for several sequential ones using bigger pages.
  - **Double-buffering:** mask I/O delays with prefetching.

CMU SCS

Blocked I/O

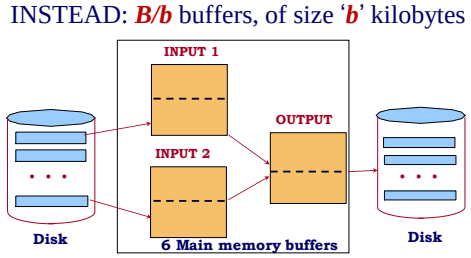
- Normally, ***B*** buffers of size (say) 4K



CMU SCS

Blocked I/O

- Normally, ***B*** buffers of size (say) 4K
- INSTEAD: ***B/b*** buffers, of size '***b***' kilobytes



CMU SCS

Blocked I/O

- Normally, ***B*** buffers of size (say) 4K
- INSTEAD: ***B/b*** buffers, of size '***b***' kilobytes
- Advantages?
- Disadvantages?

CMU SCS

Blocked I/O

- Normally, ***B*** buffers of size (say) 4K
- INSTEAD: ***B/b*** buffers, of size '***b***' kilobytes
- Advantages?
  Fewer random disk accesses because some of them are sequential.
- Disadvantages?

CMU SCS

Blocked I/O

- Normally, ***B*** buffers of size (say) 4K
- INSTEAD: ***B/b*** buffers, of size '***b***' kilobytes
- Advantages?
  Fewer random disk accesses because some of them are sequential.
- Disadvantages?
  Smaller fanout may cause more passes.

CMU SCS

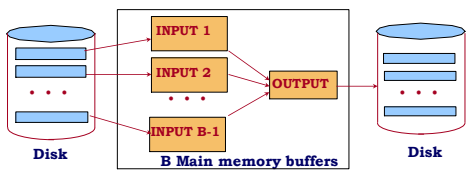
Blocked I/O & Double-buffering

- So far, we assumed random disk access
- Cost changes, if we consider that runs are written (and read) sequentially
- What could we do to exploit it?
  - **Blocked I/O:** exchange a few r.d.a for several sequential ones using bigger pages.
  - **Double-buffering:** mask I/O delays with prefetching.

CMU SCS

Double-buffering

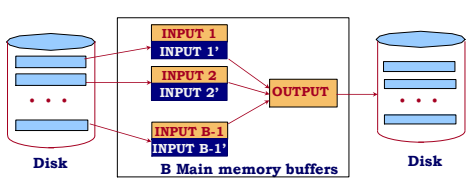
- Normally, when, say 'INPUT1' is exhausted
  - We issue a "read" request and then we wait …



CMU SCS

Double-buffering

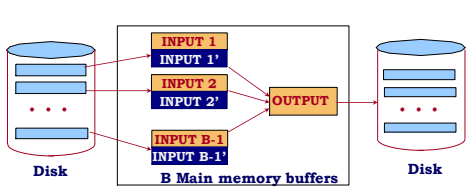
- We *prefetch* INPUT1' into "*shadow block*"
  - When INPUT1 is exhausted, we issue a "read",
  - BUT we proceed with INPUT1'



CMU SCS

Double-buffering

- This potentially requires more passes.
- But in practice, most files still sorted in 2-3 passes.



CMU SCS

Today's Class

- Sorting Overview
- Two-way Merge Sort
- External Merge Sort
- Optimizations
- B+trees for sorting

CMU SCS

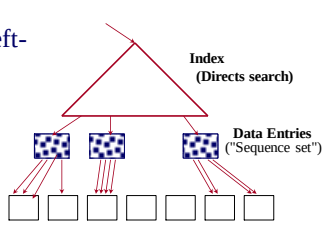
Using B+ Trees for Sorting

- 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 ***Could be Bad!***

CMU SCS

Clustered B+ Tree for Sorting

- Traverse to the left-most leaf, then retrieve all leaf pages.

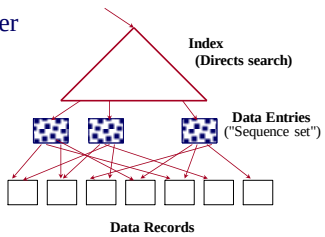


Always better than external sorting!

CMU SCS

Unclustered B+ Tree for Sorting

- Chase each pointer to the page that contains the data.



In general, one I/O per data record!

CMU SCS

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

***N*: # pages**
***p*: # of records per page**
***B*=1,000 and *block size*=32 for sorting**
***p*=100 is the more realistic value.**

CMU SCS

Summary

- External sorting is important
- External merge sort minimizes disk I/O:
  - **Pass 0:** Produces sorted ***runs*** of size ***B*** (# buffer pages).
  - **Later Passes**: ***merge*** runs.
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.