CS5443 Database Management Systems
External Sorting

Dr. Weining Zhang

Department of Computer Science
University of Texas at San Antonio

March 4, 2011

Outline

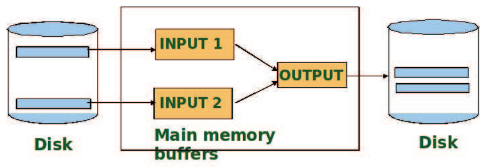
1 External Sorting
- K-Way Merge Sort
- Other Sorting Methods

Introduction

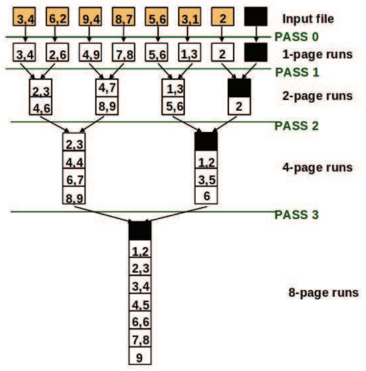
- Sorting is needed in many situations.
  - Data may be requested in sorted order
    - e.g., find students in ascending order of their GPA
  - Sorting is the first step in bulk loading B+ tree index.
  - Sorting is useful for eliminating duplicate copies in a collection of records (Why?)
  - Sort-merge join algorithm involves sorting.
- Problem: How to sort 1TB of data using 1GB of RAM.
  - why not virtual memory?

2-Way Merger Sort

- Idea: Divide and conquer.
  - sort and merge sub-files
- Pass 0: For each page, read it into buffer, sort records in it, write it to disk.
  - only one buffer page is needed
- Pass 1, 2, 3, . . . , etc.: merge sorted runs, two at a time
  - three buffer pages are needed



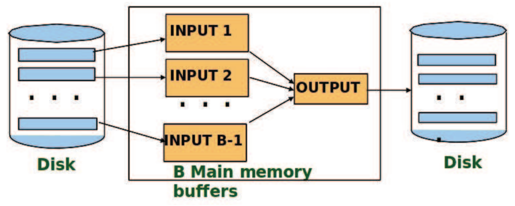
2-Way Merger Sort: An Example



- In each pass, each page is read once and written once.
- If the file has N pages, the sorting will take $\lceil \log_2 N \rceil + 1$ passes
- So, $Cost = 2N \cdot (\lceil \log_2 N \rceil + 1)$

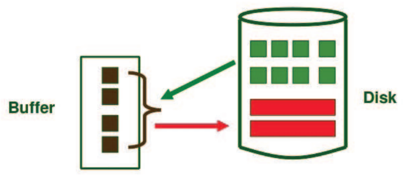
K-Way Merge Sort

- Sort a file of N pages using B (> 3) buffer frames:
  - Pass 0: Create initial runs of using B buffer frames. Result in $\lceil N/B \rceil$ sorted runs of B pages long.
  - Pass 1, 2, ..., etc.: merge $K \leq B - 1$ runs.



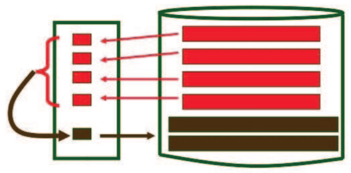
Create Initial Runs

1. Read B pages of file R into the buffer
2. Sort data records in the buffer
   - Quicksort is a fast way to sort in memory. But it can only create initial runs of B pages long
   - An alternative is ***replacement sort*** which can create initial runs of $2B$ pages long
3. Write the B sorted pages to a separate file as an initial run



K-Way Merge

```
While there is more than one input run do
    read first pages of next $K$ input runs into buffer
    initialize a new output run
    While there are more pages in the $K$ input runs do
        repeatedly move the "smallest" record to the output buffer
        if an input frame is empty, load the next input page
        if the output frame is full, write it to the output run
```



Cost of K-Way Merge Sort

- Suppose the file has N pages and the buffer has B frames.
- Each initial run can have B pages, and there are $\lceil N/B \rceil$ initial runs
- Each iteration of merge can handle $K \leq B-1$ runs. It will need $\lceil \log_{B-1}(N/B) \rceil$ passes to finish the merging
  - because if $K^{I-1}B \leq N \leq K^{I}B$, then $I = \lceil \log_K(N/B) \rceil$
- So, $Cost = 2N \cdot (\lceil \log_{B-1}(N/B) \rceil + 1)$

K-Way Merge Sort: An Example

- To sort a file of 108 pages using 5 buffer pages :
  - Pass 0: $\lceil 108/5 \rceil = 22$ sorted runs of 5 pages each (last run has only 3 pages)
  - Pass 1: $\lceil 22/4 \rceil = 6$ sorted runs of 20 pages each (last run has only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Number of Passes of KWMS

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

Replacement Sort

- If there are $B > 3$ buffer frames, use one frame for ***input***, one frame for ***output***, and $B-2$ frames for ***staging area***
- Start a new output run and fill the staging area with input records
- Repeat until all records in staging area are smaller than the “largest” record in the output frame
  - find the “smallest” record in staging area that is no smaller than the “largest” record in the output frame
  - append it to the output frame
  - move a record from the input frame to the staging area
- Flush output frame, close current output run, start a new output run
- During the process, If output frame is full, flush it to disk; If input frame is empty, read the next page

More on Replacement Sort

- Idea: keep looking for the next record in staging area to move to output until no more record can be moved.
- Advantage: the average length of an initial run is $2B$
  - longer runs often means fewer passes
- What is the worst-case scenario?
  - What is min length of an initial run?
- What is the best-case scenario?
  - What is max length of an initial run?

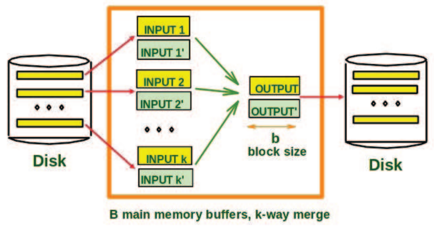
Number of Passes (Optimized)

N	B=1000	5000	10000
10^2	1	1	1
10^3	1	1	1
10^4	2	2	1
10^5	3	2	2
10^6	3	2	2
10^7	4	3	3
10^8	5	3	3
10^9	5	4	3

- Block size: 32; Initial run size: 2B

Double Buffering

- To reduce wait time for I/O request to complete, can pre-fetch into ***shadow block***.
  - Potentially, more passes; in practice, most files still be sorted in 2-3 passes.



Efficiency of Sorting

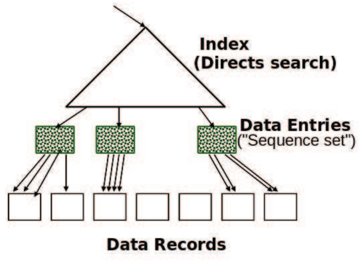
- Parallel sorting is the name of the game
- Datamation: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - World record: 3.5 seconds
    - 12-CPU SGI machine, 96 disks, 2GB of RAM
- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?

Sorting Using B+ Tree

- Scenario: Table to be sorted has a (secondary) B+ tree index on sorting column(s).
- Idea: Can retrieve records in order by traversing leaf nodes of the index.
- Is this a good idea?
  - If the B+ tree is clustered, it is a good idea.
  - If the B+ tree is not clustered, it could be a very bad idea!

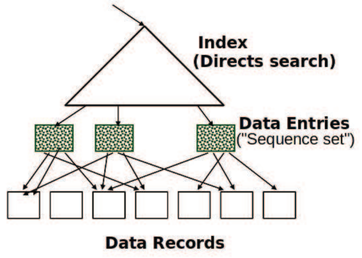
Using Clustered B+ Tree

- Cost:
  - If Alternative 1 is used, go from root to the left-most leaf, then retrieve all leaf pages
  - If Alternative 2 is used, additional cost of retrieving data records: each page fetched just once.



Using Unclustered B+ Tree

- Alternative (2) for data entries; each data entry contains rid of a data record. In general, one I/O per data record!



Number of Passes

N	Sorting	P=1	10	100
10^2	2×10^2	10^2	10^3	10^4
10^3	2×10^3	10^3	10^4	10^5
10^4	4×10^4	10^4	10^5	10^6
10^5	6×10^5	10^5	10^6	10^7
10^6	8×10^6	10^6	10^7	10^8
10^7	8×10^7	10^7	10^8	10^9

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

- 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.
- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament sort: slower (2x), longer runs
- 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.