External Sorting

Chapter 13

Why Sort?

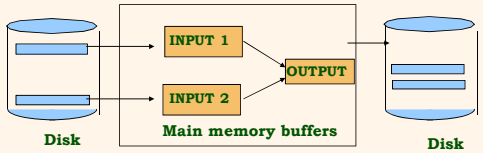
- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- Sorting is the first step in *bulk loading of* B+ tree index.
- Sorting is useful for eliminating *duplicate copies* in a collection of records (Why?)
- *Sort-merge* join algorithm involves sorting.
- Problem: sort 100Gb of data with 1Gb of RAM.
  - why not virtual memory?
- Take a look at sortbenchmark.com
- Take a look at main memory sort algos at www.sorting-algorithms.com

Sorting a file in RAM

- Three steps:
  - Read the entire file from disk into RAM
  - Sort the records using a standard sorting procedure, such as Shell sort, heap sort, bubble sort, … (100's of algos)
  - Write the file back to disk
- How can we do the above when the data size is 100 or 1000 times that of available RAM size?
- And keep I/O to a minimum!
  - Effective use of buffers
  - Merge as a way of sorting
  - Overlap processing and I/O (e.g., heapsort)

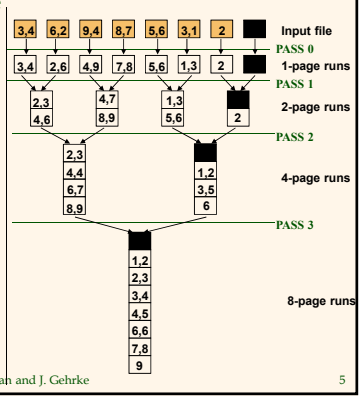
2-Way Sort of N pages

- Requires Minimum of 3 Buffers
- Pass 0: Read a page, sort it, write it.
  - only one buffer page is used
  - How many I/O's does it take?
- Pass 1, 2, 3, …, etc.:
  - Minimum three buffer pages are needed! (Why?)
  - How many i/o's are needed in each pass? (Why?)
  - How many passes are needed? (Why?)



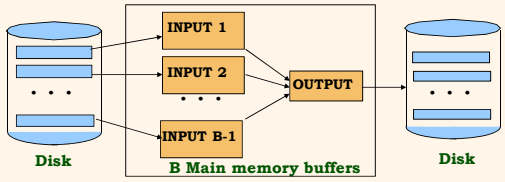
Two-Way External Merge Sort

- 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 total cost is:
  $2N(\lceil \log_2 N \rceil + 1)$
- *Idea:* ***Divide and conquer:*** sort subfiles and merge
- Can we improve upon this? How?



General External Merge Sort

- Suppose we can have more than 3 buffers! Can we use them effectively and do better?
- 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.



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: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages
- Note that with 3 buffers, initial can be of 3-page runs (not 1)
  - Cost is: 1 + $2N(\lceil \log_2 N/3 \rceil + 1)$

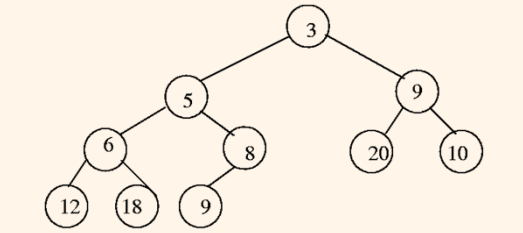
Number of Passes of External Sort

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

Internal (main memory) Sort Algorithm

- Quicksort is a fast way to sort in memory.
  - A divide-and-conquer algorithm
  - Partition initial array into 2 (preferably equal size) with some property for each partition
  - Sort each partition recursively
  - In-place sort algorithm
  - Sorts a fixed size input to generate a fixed-size output!
- An alternative is "tournament sort" (aka "heapsort")
  - You build a max- or min-heap (binary tree with some property)
    - A node's key >= its children's keys
    - Can be implemented using an array
  - Can HEAPIFY a HEAP to insert a new value!

Min-Heap



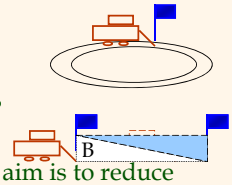
http://www.sorting-algorithms.com/random-initial-order

Internal (main memory) Sort Algorithm

- Given B buffers, Use 1 input, B-2 current set, and 1 output buffer
- Use heapsort on the current set and output the smallest to output buffer
- Insert new record into current set and output the smallest from current set which is greater than the largest in the output (for ascending sort)
- Terminating condition
  - when all values in the current set is smaller than the output, start a new run
- Instead of discreet sort, we are doing a continuous sort (snow plow example)

More on Heapsort

- Fact: average length of a run in heapsort is *2B*
  - The "snowplow" analogy
- Worst-Case:
  - What is min length of a run?
  - How does this arise?
- Best-Case:
  - What is max length of a run?
  - How does this arise?



- Quicksort is faster, but our aim is to reduce the number of initial runs and hence reduce the number of passes!!
  - Hence heapsort is better for disk-based sorting!

I/O for External Merge Sort

- … longer runs often means fewer passes!
- We are assuming that I/O is done one page at a time
- In fact, can read a *block* of pages sequentially!
  - Much faster/cheaper than reading pages of the block individually
- Suggests we should make each buffer (input /output) be a *block* of pages.
  - But this will reduce fan-out during merge passes! Why?
  - In practice, most files still sorted in 2-3 passes.
- Minimizes I/O cost, not the # of I/O's
- Also, double buffering. What does this reduce?

I/O for External Merge Sort (2)

- Blocked I/O
  - Suppose a block is b pages
  - We need b buffer pages for output (1 block)
  - We can only merge ceiling ((B-b)/b) runs (instead of B-1 runs when we read 1 page at a time)
  - If we have 10 buffer pages, we can
    - Either merge nine runs without using blocks, or
    - Four runs if we assume 2 page blocks
  - This tradeoff between using blocks vs. the number of runs needs to be taken into account for external merge sort!
  - The good news is that with greater memory, both block sizes and #runs can be kept to a decent value

Number of Passes of Optimized Sort

N	B=1,000	B=5,000	B=10,000
100	1	1	1
1,000	1	1	1
10,000	2	2	1
100,000	3	2	2
1,000,000	3	2	2
10,000,000	4	3	3
100,000,000	5	3	3
1,000,000,000	5	4	3

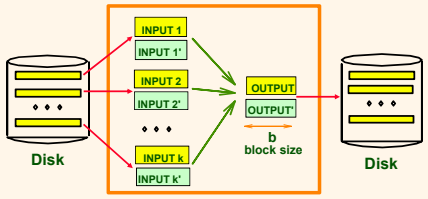
- *Block size = 32, initial pass produces runs of size 2B.*

Blocked I/O

- Let b be the units of read and write
- Given B buffers, # of runs that can be merged is floor ((B-b)/b)
- If we have 10 buffers, we can
  - Merge 9 runs at a time with 1 page buffer, or
  - Merge 4 runs at a time with 2 page input (for each block) and output buffer blocks
- How does it reduce I/O cost?

Double Buffering

- To reduce wait time for I/O request to complete, can *prefetch* into `shadow block'.
  - Tradeoff between buffers and passes; in practice, most files *still* sorted in 2-3 passes.



B main memory buffers, k-way merge

Does double buffering reduce the # i/o's?

Sorting Records! (http://sortbenchmark.org/)

Since 2007, it is done by a the sort benchmark committee

Daytona (stock car)	Indy (formula 1)
Sort code must be general purpose.	Need only sort 100-byte records with 10-byte keys.

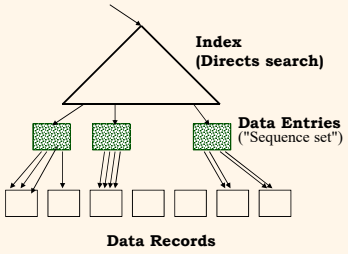
Gray	2016, 44.8 TB/min **Tencent Sort** 100 TB in 134 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub	2016, 60.7 TB/min **Tencent Sort** 100 TB in 98.8 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub
Cloud	2016, $1.44 / TB **NADSort** 100 TB for $144 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk) Qian Wang, Rong Gu, Yihua Huang Nanjing University Reynold Xin Databricks Inc. Wei Wu, Jun Song, Junluan Xia Alibaba Group Inc.	2016, $1.44 / TB **NADSort** 100 TB for $144 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk) Qian Wang, Rong Gu, Yihua Huang Nanjing University Reynold Xin Databricks Inc. Wei Wu, Jun Song, Junluan Xia Alibaba Group Inc.
Minute	2016, 37 TB **Tencent Sort** 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub	2016, 55 TB **Tencent Sort** 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub

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 a very bad idea!***

Clustered B+ Tree Used for Sorting

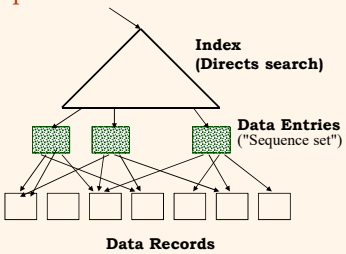
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.



- *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!



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; 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*** *or more* (# 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 passes rarely more than 2 or 3.

Summary, cont.

- 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 B+ tree is usually very bad.