External Sorting

Chapter 13

One of the advantages of being disorderly is that one is constantly making exciting discoveries.

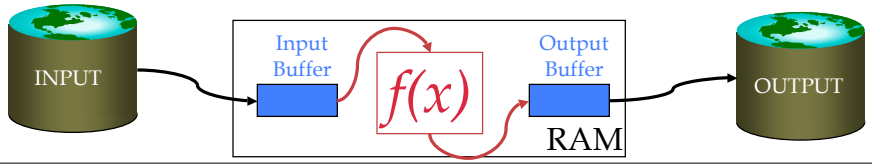
A. A. Milne

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 (Why?)**
- **Sorting is useful for summarizing related groups of tuples**
- ***Sort-merge* join algorithm involves sorting.**
- **Problem: sort 1Gb of data with 1Mb of RAM.**
  - why not virtual memory?

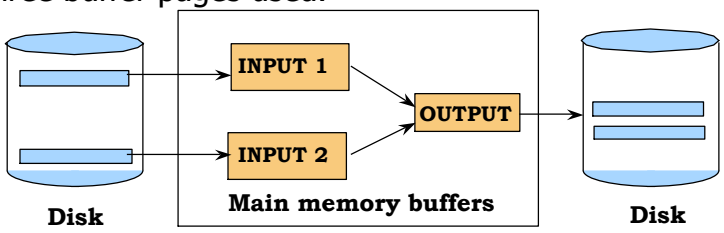
Streaming Data Through RAM

- **An important detail for sorting & other DB operations**
- **Simple case:**
  - Compute *f(x)* for each record, write out the result
  - Read a page from INPUT to Input Buffer
  - Write *f(x)* for each item to Output Buffer
  - When Input Buffer is consumed, read another page
  - When Output Buffer fills, write it to OUTPUT
- **Reads and Writes are *not* coordinated**
  - E.g., if f() is Compress(), you read many pages per write.
  - E.g., if f() is DeCompress(), you write many pages per read.



2-Way Sort

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



Two-Way External Merge Sort

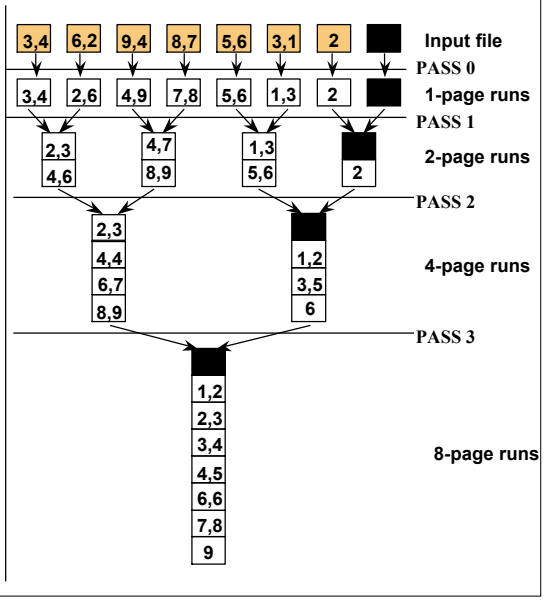
- **Each pass we read + write each page in file.**
- **N pages in the file => the number of passes**

$$= \lceil \log_2 N \rceil + 1$$

- **So total cost is:**

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

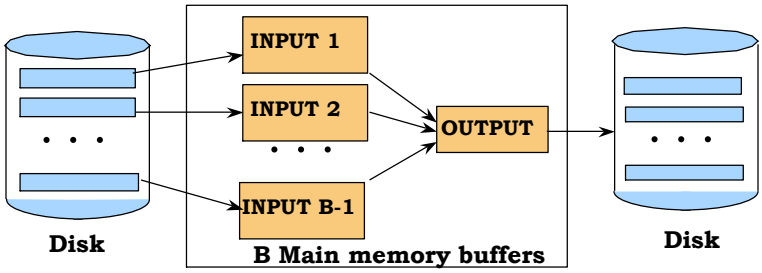
- ***Idea:*** *Divide and conquer:* **sort subfiles and merge**



General External Merge Sort

☛ *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.



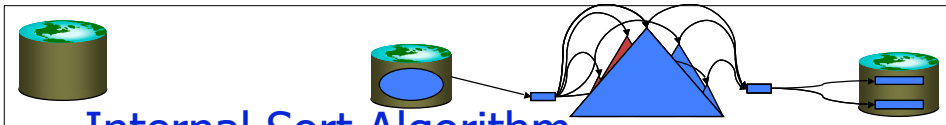
Cost of External Merge Sort

- **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)
- **Now, do four-way (B-1) merges**
  - 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

(I/O cost is 2N times number of passes)

N	B=3	B=5	B=9	B=17	B=129	B=257
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted],[redacted],[redacted],[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]



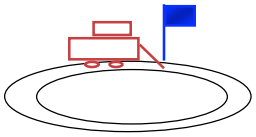
Internal Sort Algorithm

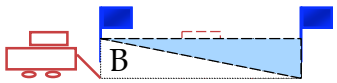
- **Quicksort is a fast way to sort in memory.**
- **Alternative: "tournament sort" (a.k.a. "heapsort", "replacement selection")**
- **Keep two heaps in memory, H1 and H2**

```
read B-2 pages of records, inserting into H1;
while (records left) {
    m = H1.removemin();  put m in output buffer;
    if (H1 is empty)
        H1 = H2;  H2.reset();  start new output run;
    else
        read in a new record r (use 1 buffer for
          input pages);
        if (r < m)  H2.insert(r);
        else        H1.insert(r);
}
H1.output();  start new run;  H2.output();
```

More on Heapsort

- **Fact: average length of a run is *2(B-2)***
  - 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 … longer runs often means fewer passes!**





I/O for External Merge Sort

- **Actually, doing I/O a page at a time**
  - Not an I/O per record
- **In fact, read a *block (chunk)* of pages sequentially!**
- **Suggests we should make each buffer (input/output) be a *chunk* of pages.**
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 2-3 passes.

Number of Passes of Optimized Sort

N	B=1,000	B=5,000	B=10,000
■■■	■	■	■
■,■■■	■	■	■
■■,■■■	■	■	■
■■■,■■■	■	■	■
■,■■■,■■■	■	■	■
■■,■■■,■■■	■	■	■
■■■,■■■,■■■	■	■	■
■,■■■,■■■,■■■	■	■	■

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

Sorting Records!

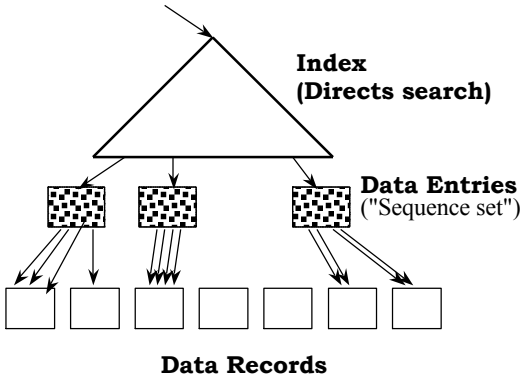
- **Sorting has become a blood sport!**
  - Parallel sorting is the name of the game ...
- **Minute Sort: how many 100-byte records can you sort in a minute?**
  - Typical DBMS: 10MB (~100,000 records)
  - Current World record: 40 ***GB***
  - ***On Fujitsu hardware (128 disks)***
- **Penny Sort: how many can you sort for a penny?**
  - Current world record: 32 GB
  - On a $760 system
- **See http://research.microsoft.com/barc/SortBenchmark/**

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

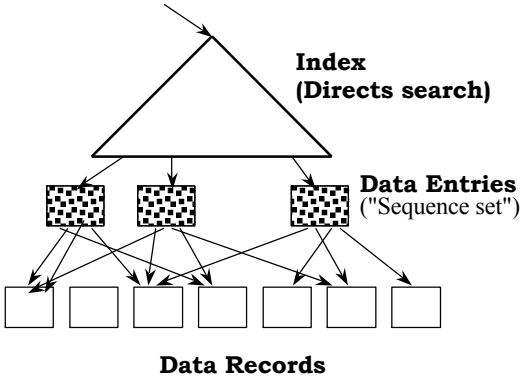
- **Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)**
- **If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.**



☛ *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
■■■	■■■	■■■	■,■■■	■■,■■■
■,■■■	■,■■■	■,■■■	■■,■■■	■■■,■■■
■■,■■■	■■,■■■	■■,■■■	■■■,■■■	■,■■■,■■■
■■■,■■■	■■■,■■■	■■■,■■■	■,■■■,■■■	■■,■■■,■■■
■,■■■,■■■	■,■■■,■■■	■,■■■,■■■	■■,■■■,■■■	■■■,■■■,■■■
■■,■■■,■■■	■■,■■■,■■■	■■,■■■,■■■	■■■,■■■,■■■	■,■■■,■■■,■■■

- ☛ *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*** (# 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 tree is usually very bad.**