

Monday March 23rd, 2009

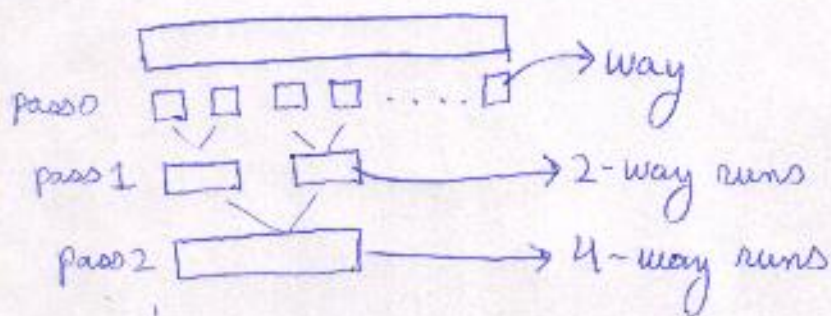
I. Sorting in Databases:

When sorting in databases, we worry more about I/Os than computation.

1. range queries
2. ORDER BY (from user's query)
3. Sort-Merge join
4. Eliminating Duplicates
5. Bulk loading of tree based indexes

II Basic Idea:

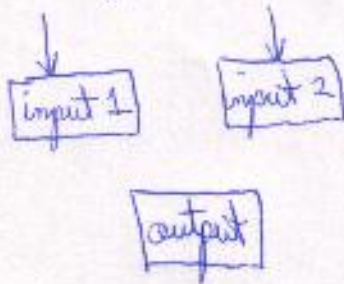
Merge-Sort (Divide & Conquer)



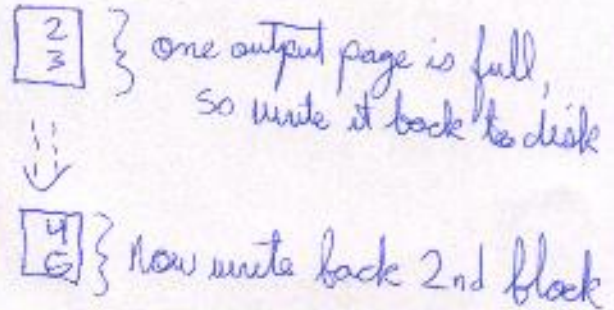
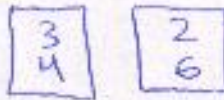
$\log_2 N$ passes

on every pass we have $2N$ I/Os.

Using only 3 pages:



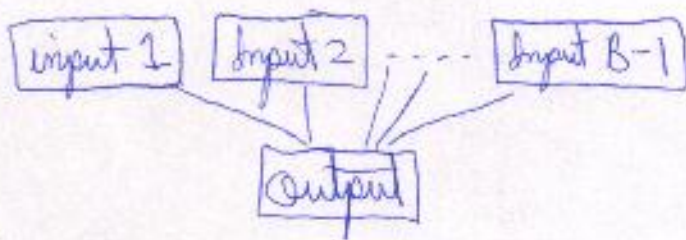
example:



still $2N \lceil \log_2 N \rceil$ I/Os

III. B-way runs

B buffer pages available for sorting.



Merge B-1 pages using 1 output buffer. When the output buffer fills up, write it back to disk.

Note: Cpu time may have a more significant impact in this case.

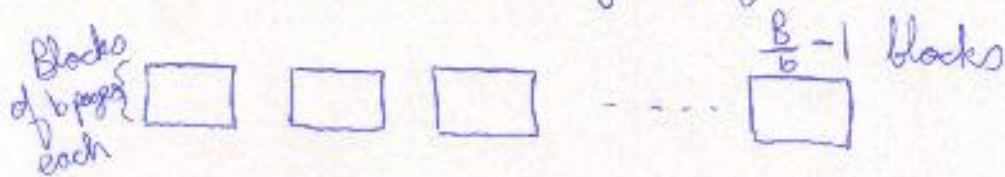
$$\# \text{ passes} = 1 + \lceil \log_{B-1} \lceil N/B \rceil \rceil$$

still $2N$ I/Os for each pass, but there are less passes.

$$\text{total cost} = 2N * (\# \text{ of passes})$$

IV. Block-Based Sorting

Do I/Os in chunks of b pages each



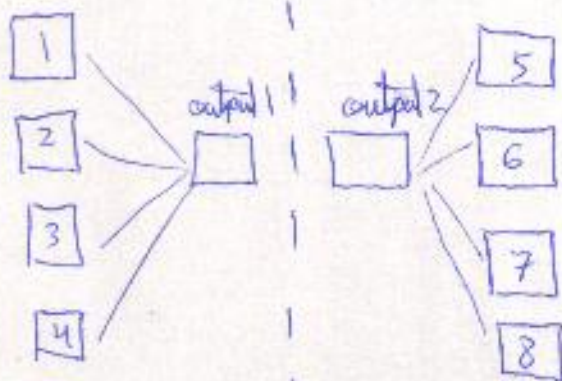
$$\# \text{ passes} = \log_{\left(\frac{B}{b} - 1\right)} \left(\frac{N}{B}\right)$$

$$\text{total cost} = c(2N) \left(\log_{\left(\frac{B}{b} - 1\right)} \left(\frac{N}{B}\right)\right)$$

$c < 1$

V. Double-buffer

CPU sits idle while writing back output page, so use double-buffering.



While output 1 is being written back to disk, continue sorting using output 2 and so on.

Sorting Using Indexes:



choice 1) Clustered Index

I/O cost: $\# \text{ data pages} + \# \text{ leaf pages}$

2) Unclustered Index

I/O cost: $(\# \text{ of pointers}) + \# \text{ leaf pages}$