External Sorting

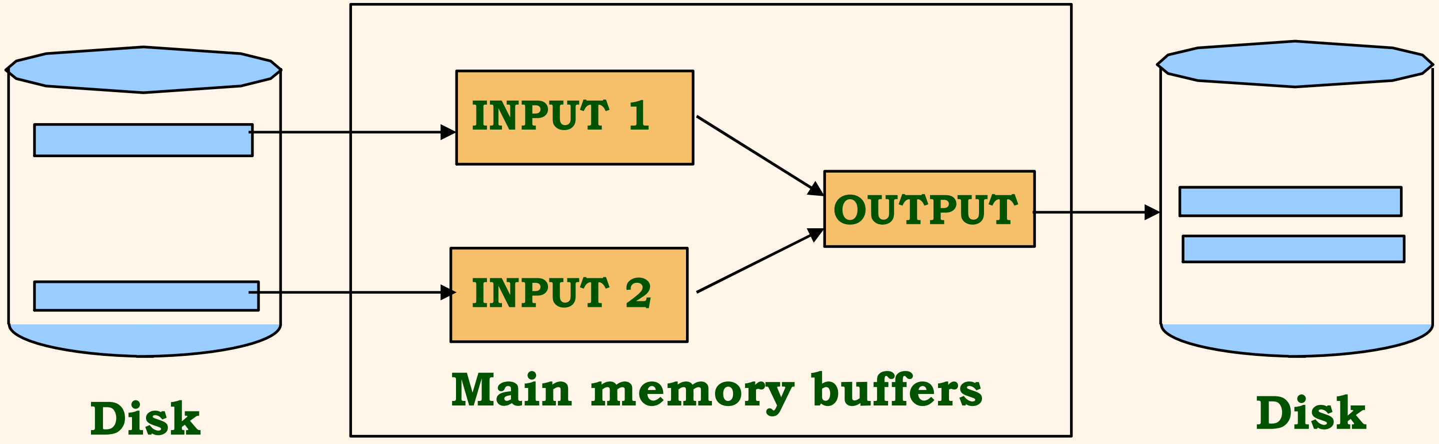
Chapter 11

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?)
- *Sort-merge* join algorithm involves sorting.
- Problem: sort 1Gb of data with 1Mb of RAM.
  - why not virtual memory?

2-Way Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it.
  - only one buffer page is used
- Pass 2, 3, ? etc.:
  - three buffer pages used.



Two-Way External Merge Sort

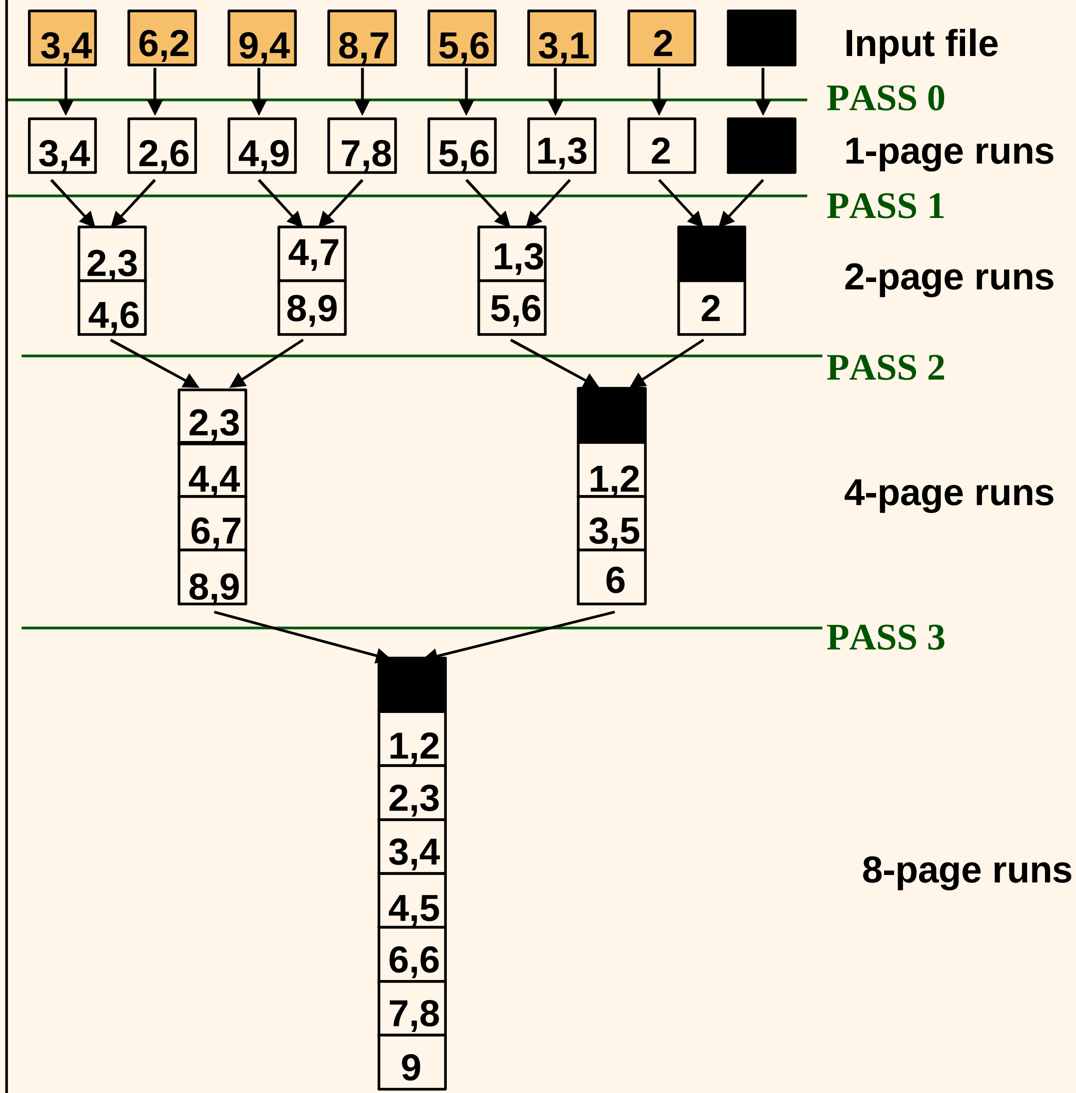
- Each pass we read + write each page in file.
- N pages in the file => the number of passes

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

- So toal cost is:

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

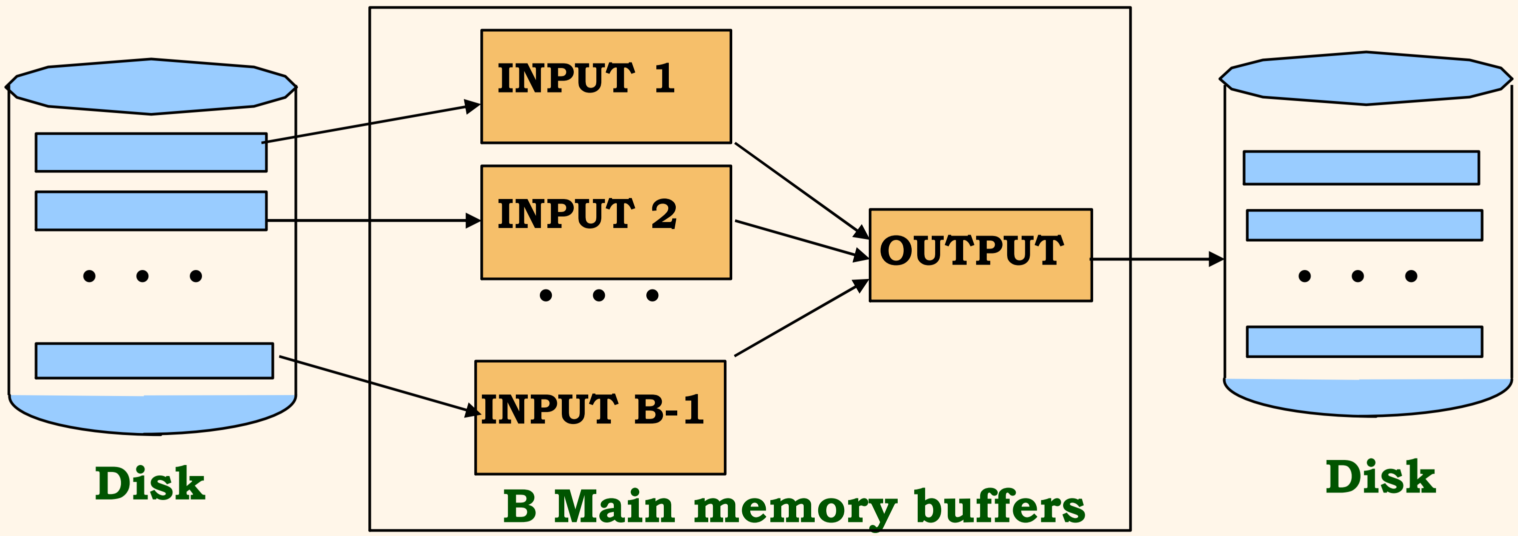
- <u>*Idea:*</u> ***Divide and conquer:*** sort subfiles and merge



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 $\lceil N / B \rceil$ 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

Number of Passes of External Sort

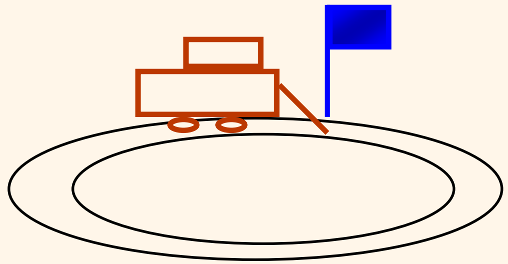
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Internal Sort Algorithm

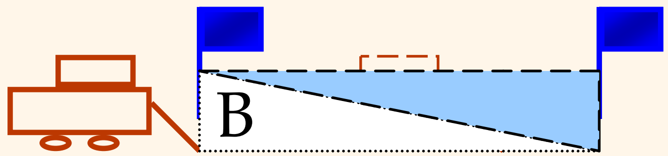
- Quicksort is a fast way to sort in memory.
- An alternative is tournament sort?(a.k.a. heapsort?
  - **`Top:`** Read in ***B*** blocks
  - **`Output:`** move smallest record to output buffer
  - Read in a new record *r*
  - insert *r* into heap
  - if *r* not smallest, then **`GOTO Output`**
  - else remove *r* from heap
  - output heap?in order; **`GOTO Top`**

More on Heapsort

- Fact: average length of a run in heapsort is *2B*



- 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 ...

I/O for External Merge Sort

- ?longer runs often means fewer passes!
- Actually, do I/O a page at a time
- In fact, read a ***block*** of pages sequentially!
- Suggests we should make each buffer (input/output) be a *block* 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
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.*

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 runs 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 are still improving!