Database Systems

External Sort

Based on slides by Feifei Li, University of Utah

What's external sorting?

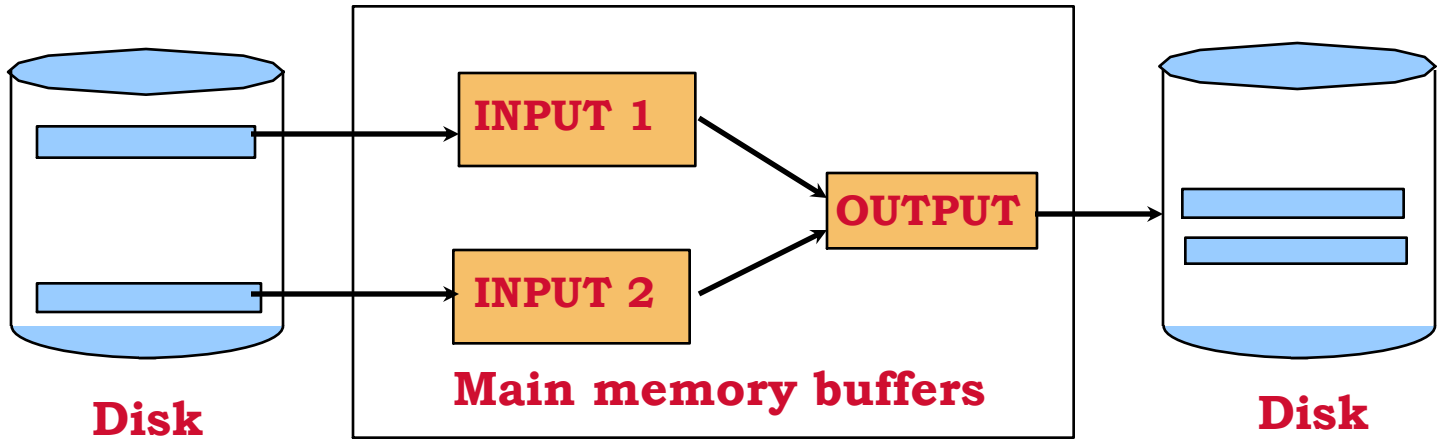
- Problem: sort 1TB of data with 1GB of RAM.
  - why not virtual memory?
    - Swap involves expensive IOs

Using secondary storage effectively

- General Wisdom :
  - I/O costs dominate
  - Design algorithms to reduce I/O

2-Way Sort: Requires 3 Buffers

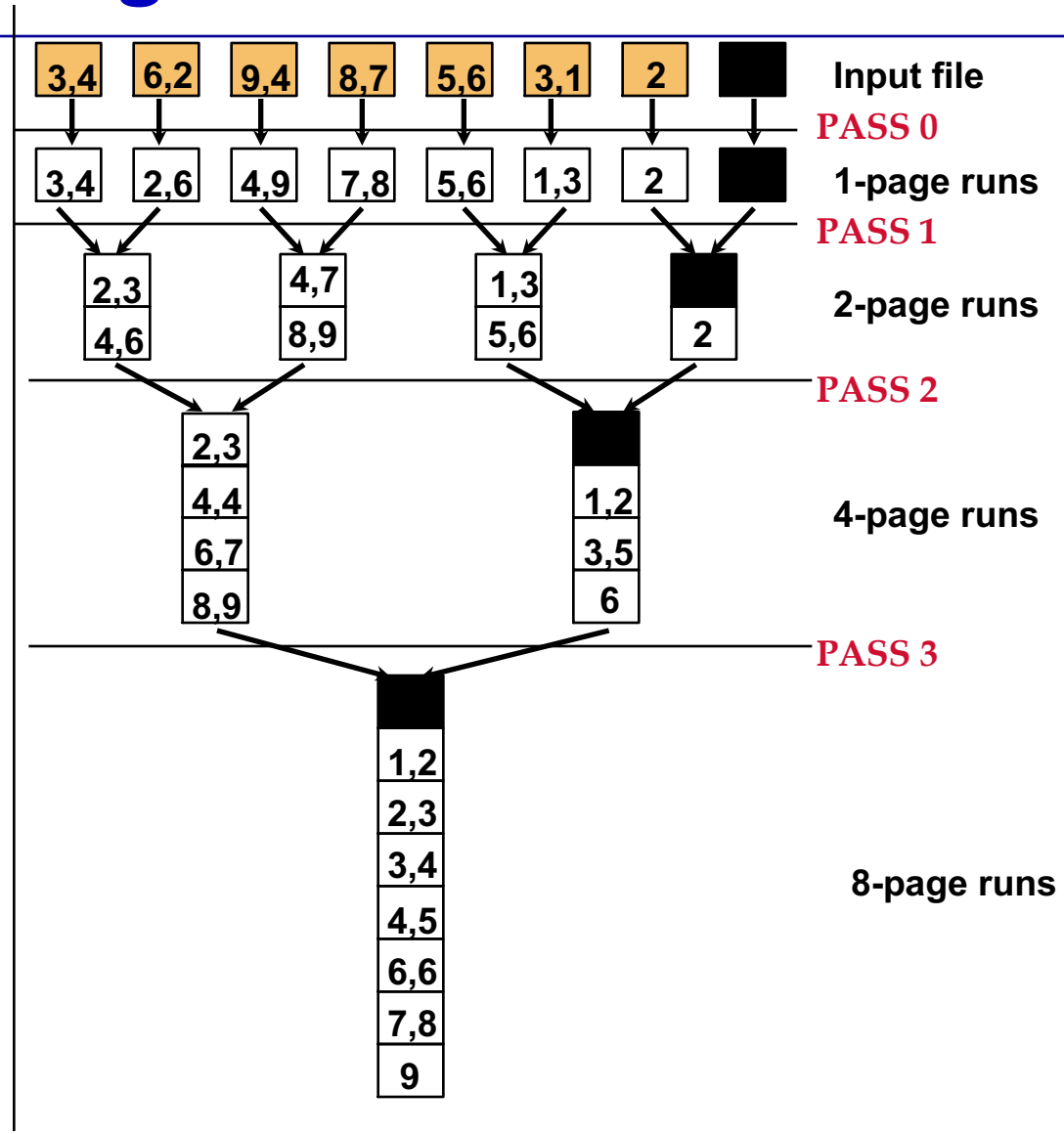
- Phase 1: PREPARE.
  - Read a page, sort it, write it.
  - only one buffer page is used
- Phase 2, 3, …, etc.: MERGE:
  - three buffer pages used.



Two-Way External Merge Sort

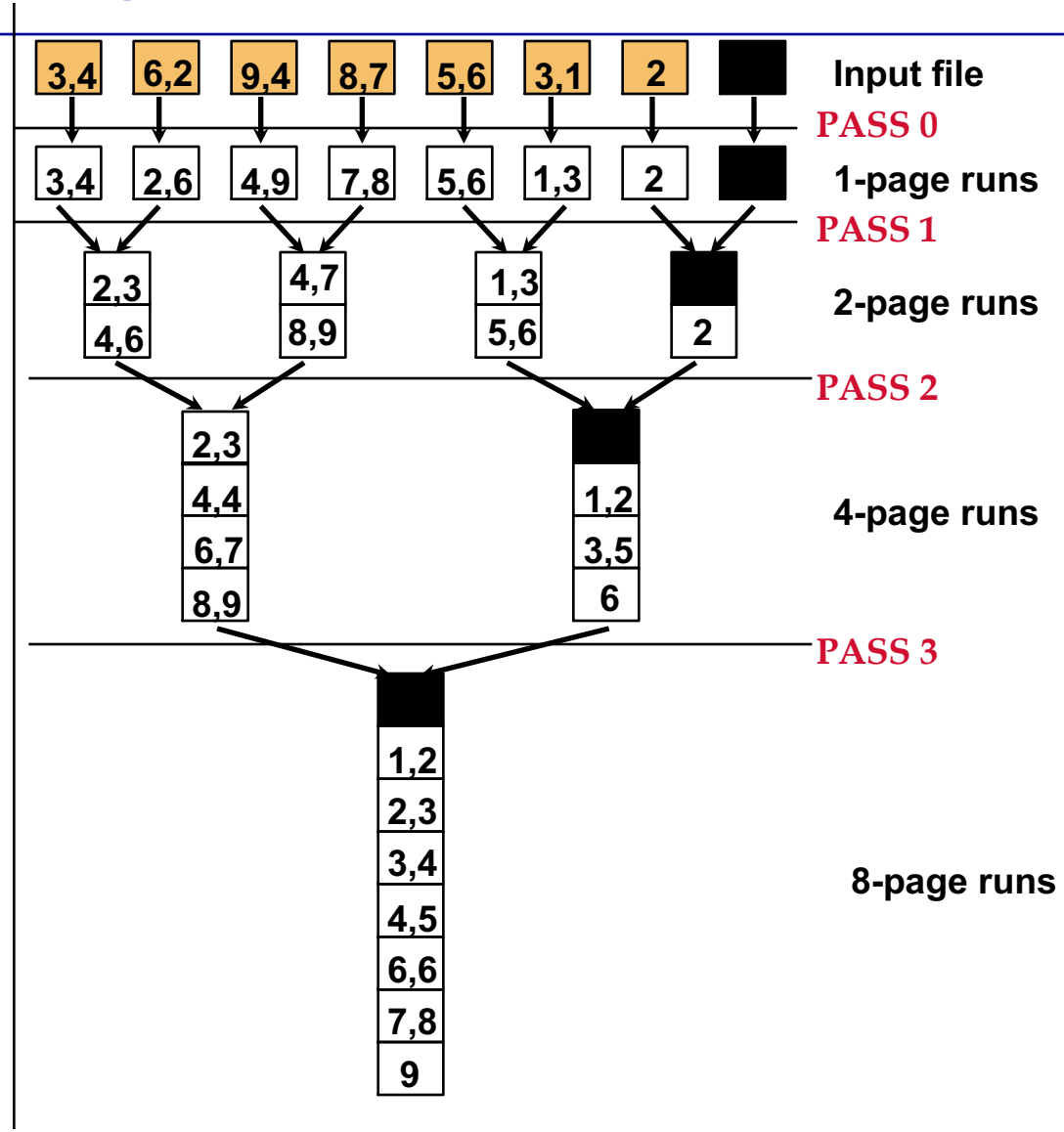
- *Idea:* *Divide and conquer:* sort sub-files and merge into larger sorts

- N is the number of records
- B is the number of records per page
- M is the size of main memory in number of records



Two-Way External Merge Sort

- Costs for pass :
  all pages
- # of passes :
  height of tree
- Total cost :
  product of above



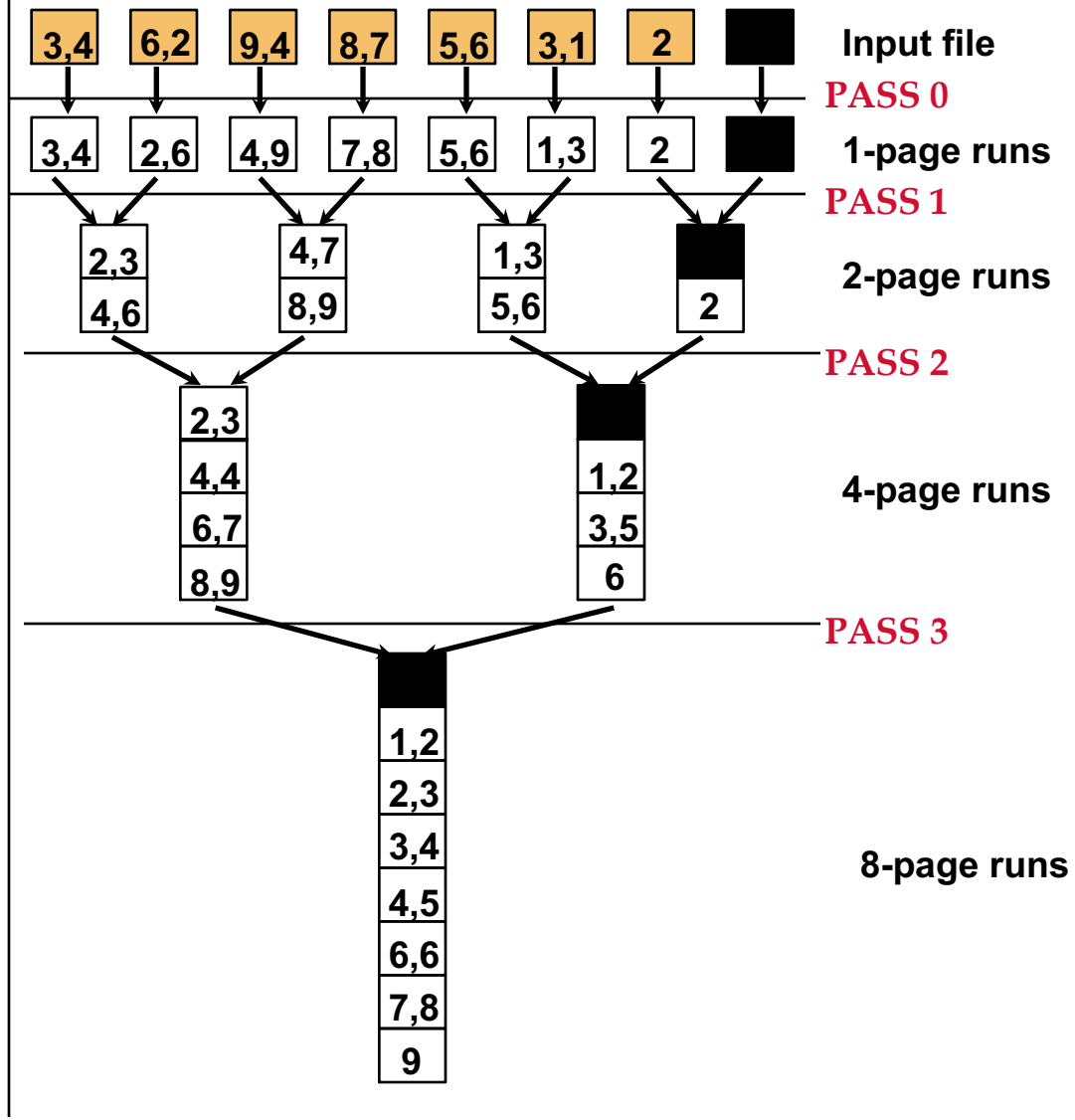
Two-Way External Merge Sort

- Each pass we read + write each page in file.
- N/B pages in file => 2N/B
- Number of passes

$$= \left\lceil \log_2 \frac{N}{B} \right\rceil + 1$$

- So total cost is:

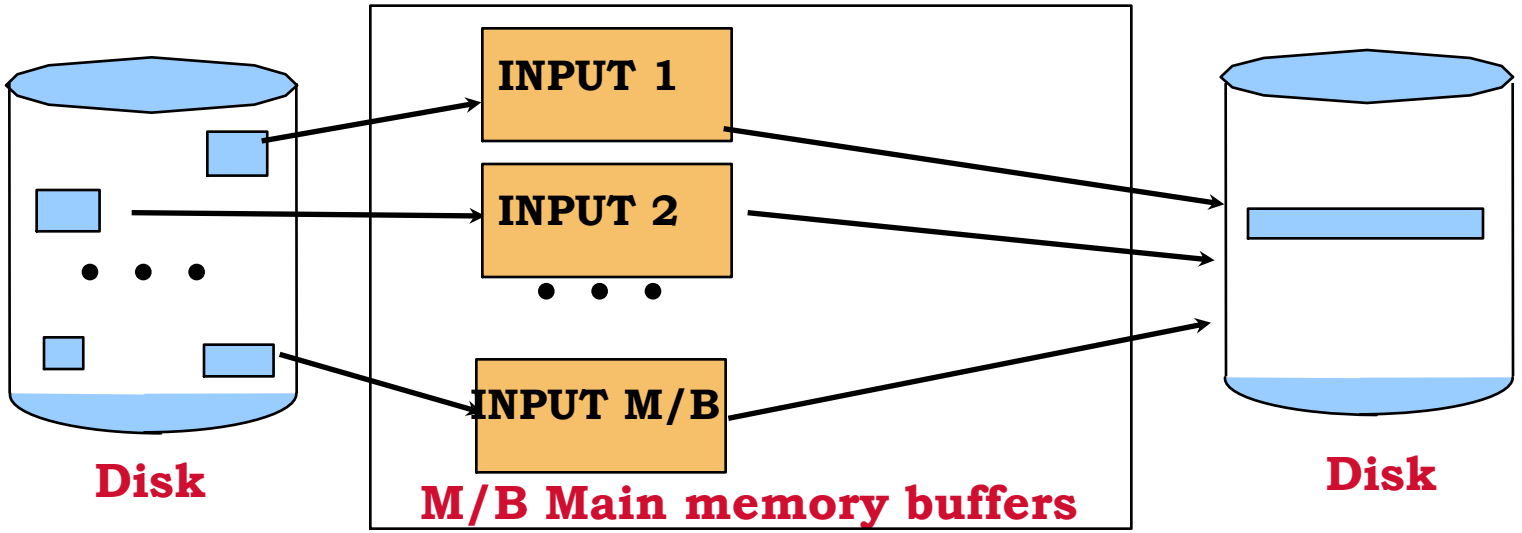
$$2\frac{N}{B}\left(\left\lceil \log_2 \frac{N}{B} \right\rceil + 1\right)$$



External Merge Sort

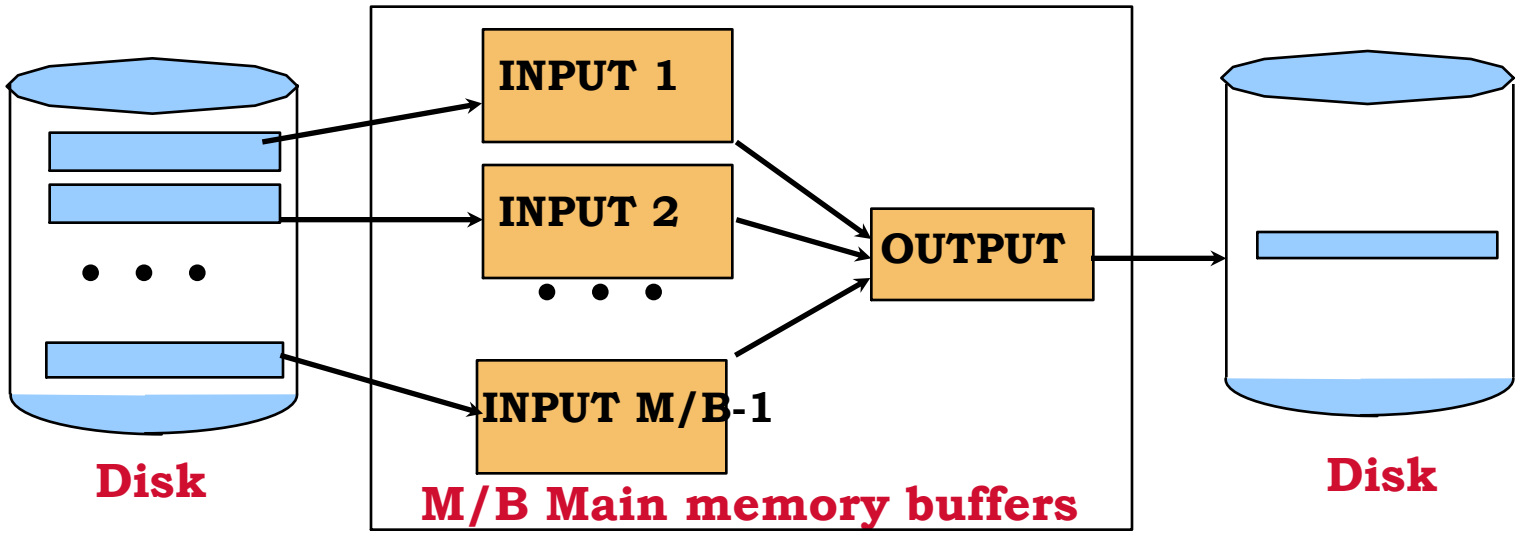
- What if we had more buffer pages?
- How do we utilize them wisely ?

Phase 1 : Prepare



- **Construct as large as possible starter lists.**

Phase 2 : Merge

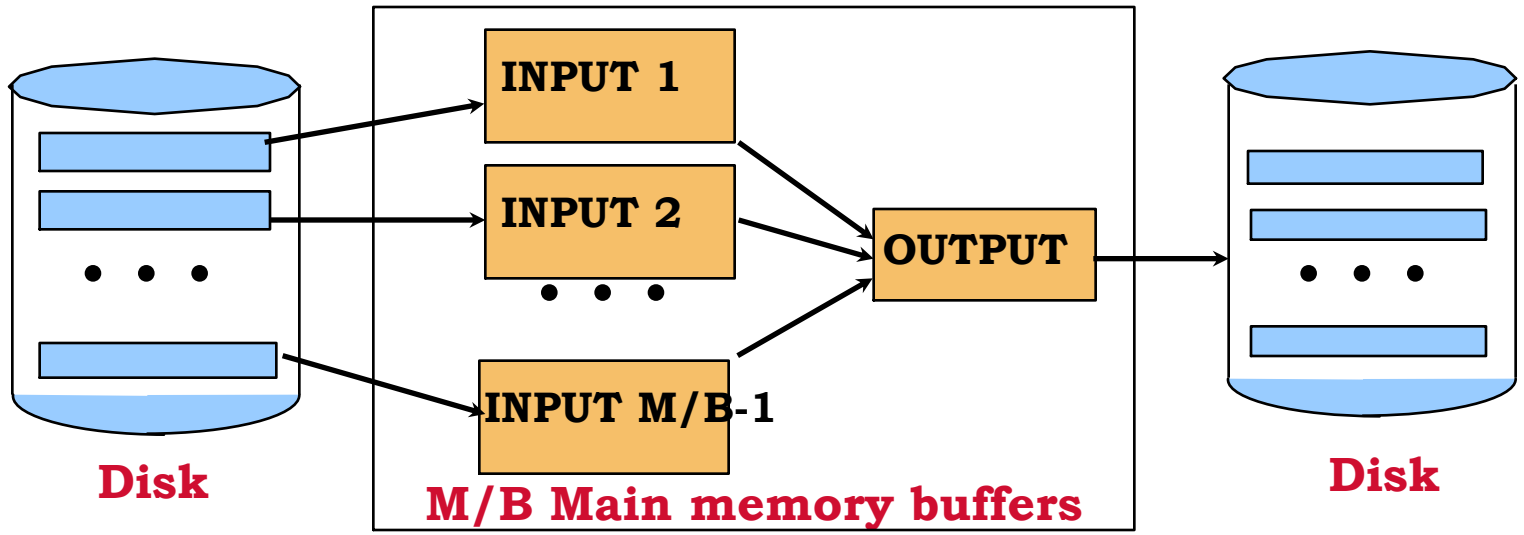


Compose as many sorted sublists into one long sorted list.

General External Merge Sort

☛ *How can we utilize more than 3 buffer pages?*

- To sort a file with *N/B* pages using *M*/B buffer pages:
  - Pass 0: use *M/B* buffer pages. Produce $\left\lceil \frac{N}{B} / \frac{M}{B} \right\rceil$ sorted runs of *M/B* pages each.
  - Pass 1, 2, …, etc.: merge *M/B-1* runs.



Cost of External Merge Sort

- Number of passes:
- Cost = 2N/B * (# of passes) $1+\lceil \log_{M/B-1} \lceil N/M \rceil \rceil$

Example

- Buffer : with 5 buffer pages
- File to sort : 108 pages

  - Pass 0:
    - Size of each run?
    - Number of runs?
  - Pass 1:
    - Size of each run?
    - Number of runs?
  - Pass 2: ???

Example

- Buffer : with 5 buffer pages
- File to sort : 108 pages
  - 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

- **Total I/O costs: ?**

Example

- Buffer : with 5 buffer pages
- File to sort : 108 pages

  - 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

- **Total I/O costs: 2*108 * (4 passes)**

Number of Passes of External Sort

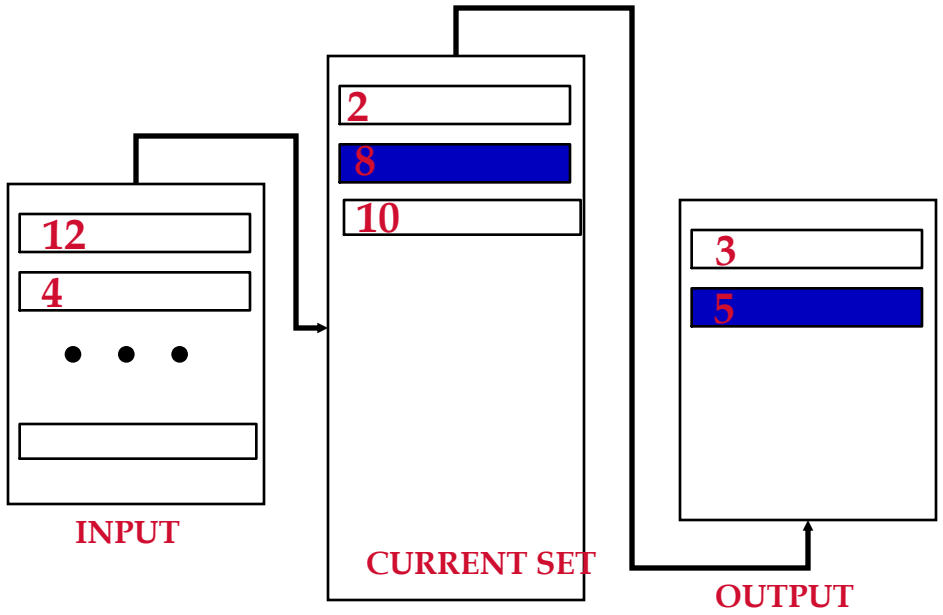
- gain of utilizing all available buffers
- importance of a high fan-in during merging

N/B	M/B=3	=5	=9	=17	=129	=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

Optimizing External Sorting

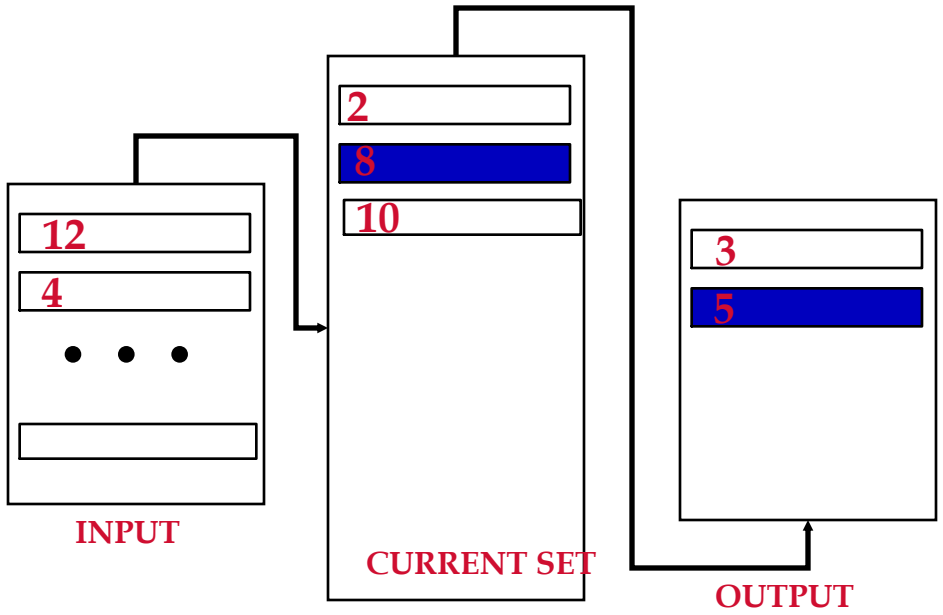
- Cost metric ?
  - I/O only (till now)
  - CPU is nontrivial, worth reducing

Internal Sort Algorithm



- 1 input, 1 output, M/B-2 current set
- Main idea: repeatedly pick tuple in current set with smallest k value that is still greater than largest k value in output buffer and append it to output buffer

Internal Sort Algorithm



- **Input & Output?**

  new input page is read in if it is consumed, output is written out when it is full

- **When terminate current run?**

  When all tuples in current set are smaller than largest tuple in output buffer.

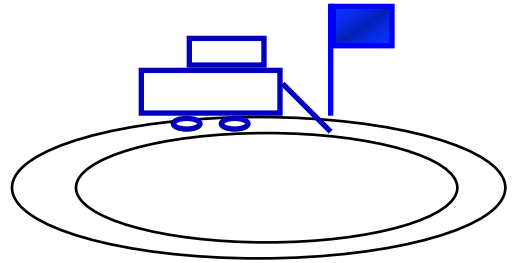
Internal Sort Algorithm

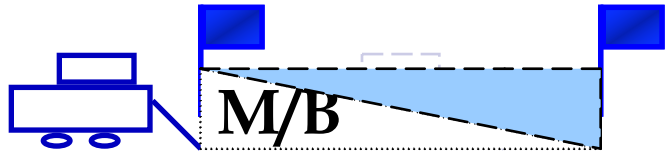
- Quicksort is a fast way to sort in memory.
- Alternative: “tournament sort” (a.k.a. “heapsort”, “replacement selection”)
- Keep two heaps in memory, H1 and H2

```
read M/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(M/B-2)*
  - The “snowplow” analogy





- Quicksort is faster, but … longer runs often means fewer passes!

Optimizing External Sorting

- Further optimization for external sorting.
  - Blocked I/O
  - Double buffering

I/O for External Merge Sort

- Thus far : do 1 I/O a page at a time
- But cost also includes real page read/write time.

- Reading a *block* of pages sequentially is cheaper!

- 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.

I/O for External Merge sort

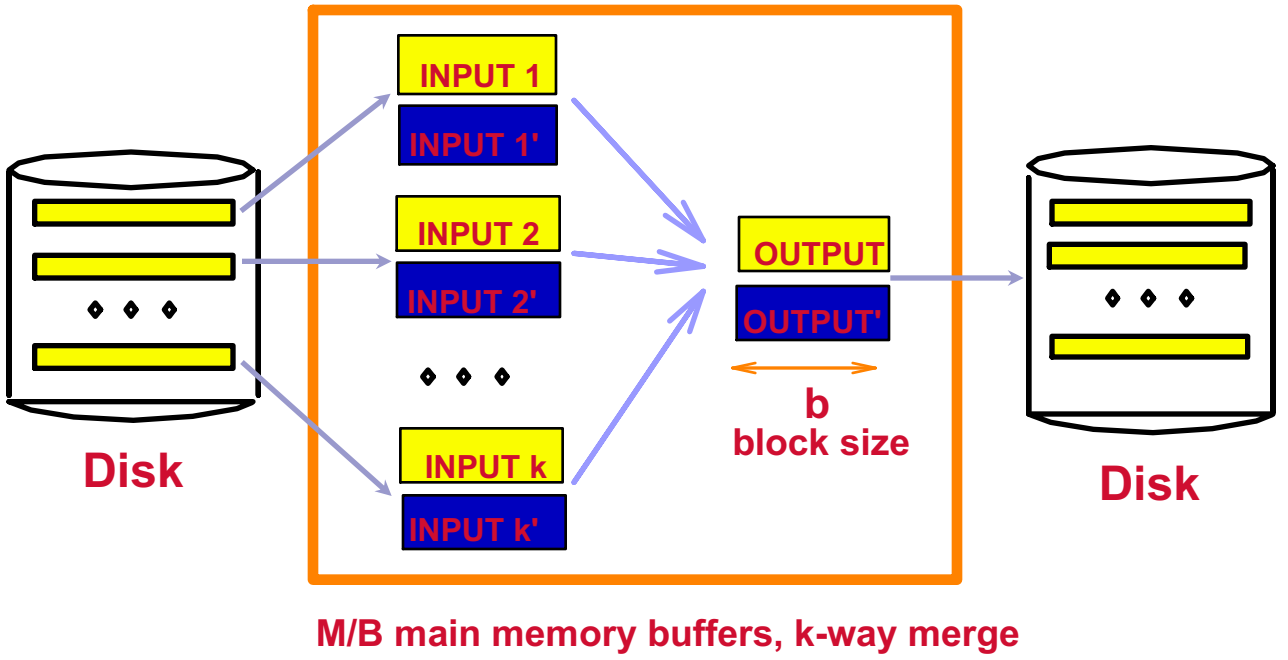
- Example

  buffer blocks = b pages
  set one buffer block for input, one buffer block for output
  merge |(M/B-b)/b| runs in each pass

  e.g., 10 buffer pages
  9 runs at a time with one-page input and output buffer blocks
  4 runs at a time with two-page input and output buffer block

Double Buffering – Overlap CPU and I/O

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



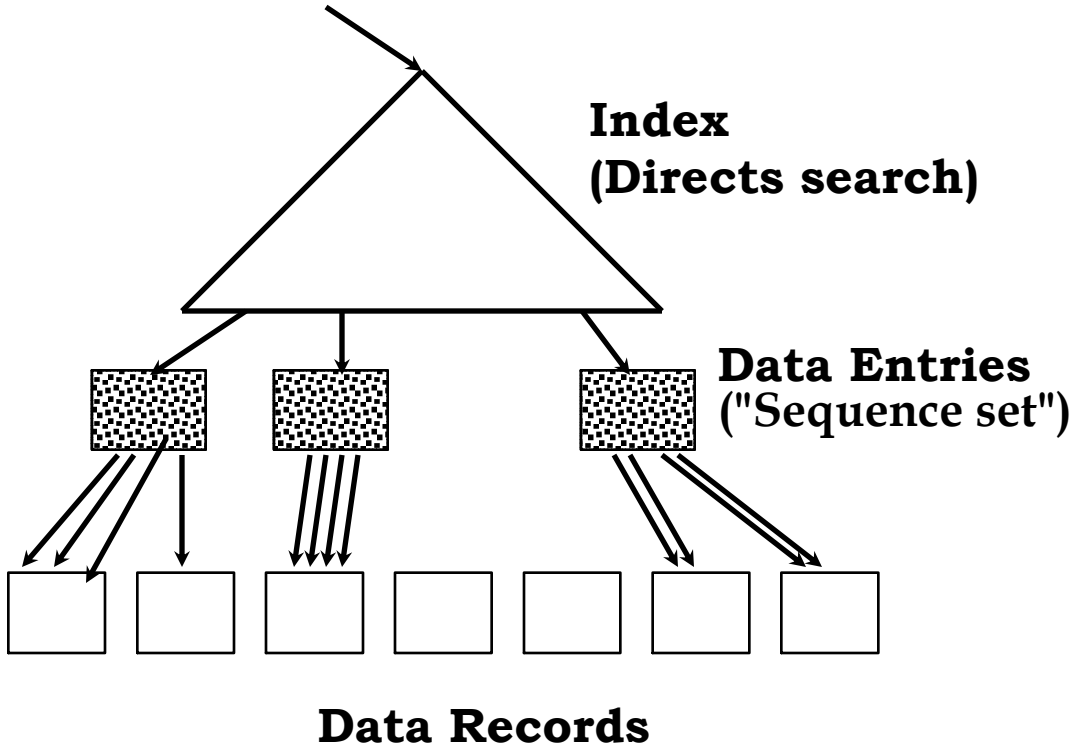
Using B+ Trees for Sorting

- 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:

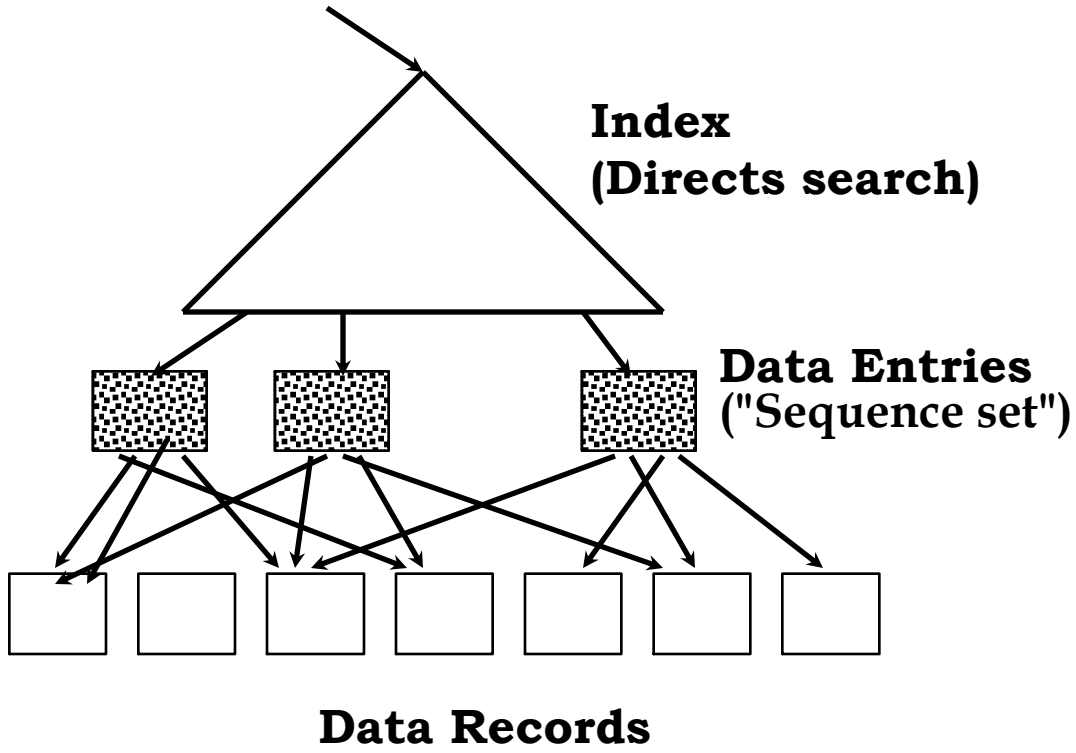
  root to left-most leaf, then retrieve all leaf pages (Alternative 1)
- For Alternative 2, additional cost of retrieving data records: each page fetched just once.



☛ *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!



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 M/***B*** (# buffer pages). Later passes: ***merge*** runs.
  - # of runs merged at a time depends on M/***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.