CSE 350: Advanced Data Structures and Indexes (Spring 2026)

Lecture 15: External Sorting

3/24/2026

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

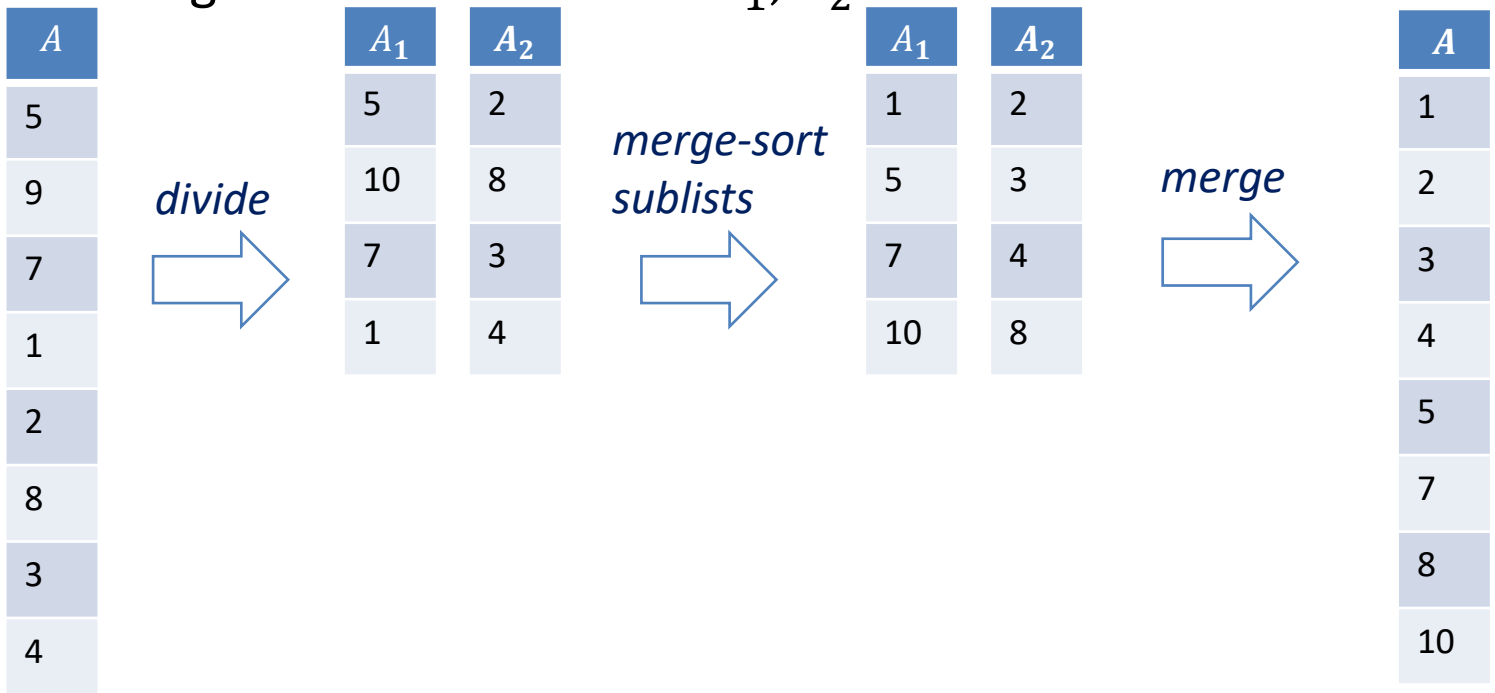
External sorting

- Problem: sort or hashing 1TB of data over 1GB of RAM
  - Why not virtual memory?
    - Swaps involve expensive random I/Os
  - Why not using B-Tree/extendible hashing/linear hashing?
    - Dynamic structures carry additional overhead for maintenance (not needed in QP)
    - Missing optimization opportunities with hybrid approach (see later)

- General wisdom:
  - I/O cost dominates the total cost
  - Design algorithms to reduce the number of I/Os

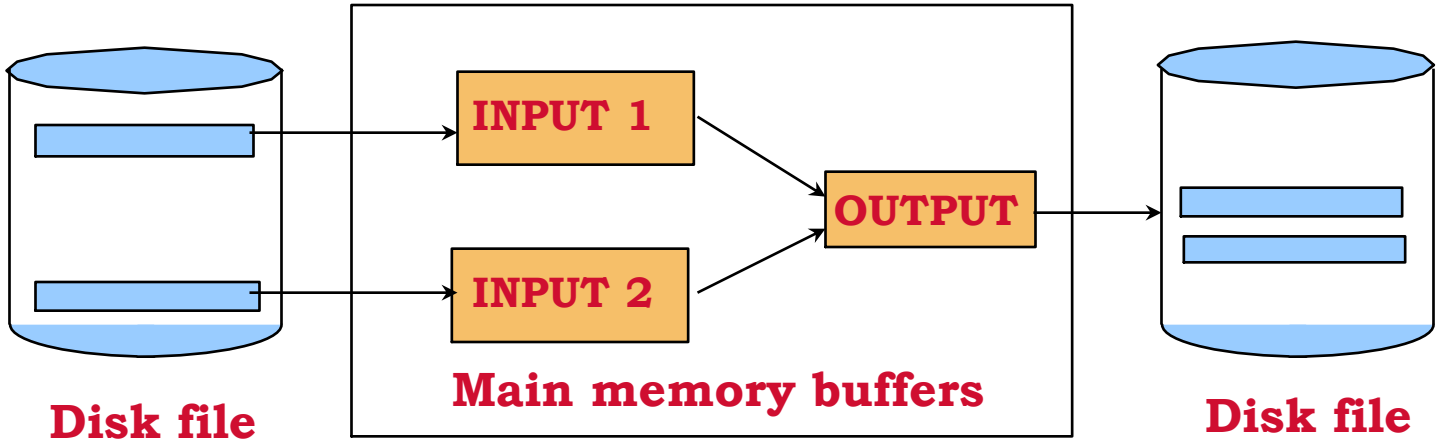
In-memory two-way merge-sort: a starting point

- Recall the two-way merge-sort
  - given a list of items in $A[0..n-1]$
  - recursively divide and conquer the problem
    - divide the list into two halves $A_1\left[0..\left\lfloor\frac{n}{2}\right\rfloor\right], A_2\left[\left\lfloor\frac{n}{2}\right\rfloor+1, n-1\right]$
    - merge-sort A_1 and A_2 individually
    - merge the two sorted list A_1, A_2



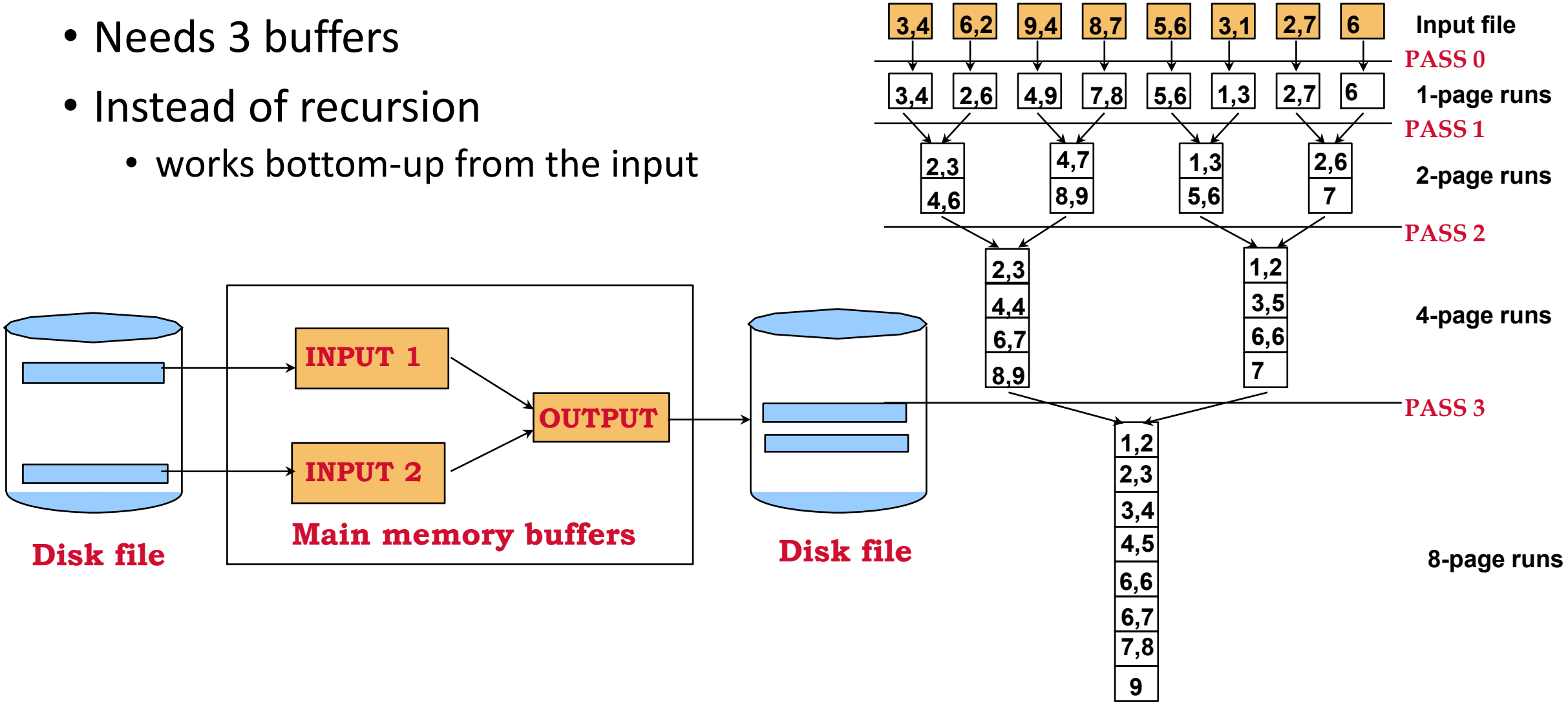
External two-way merge sort

- Needs 3 buffers
- Instead of recursion
  - works bottom up from the input



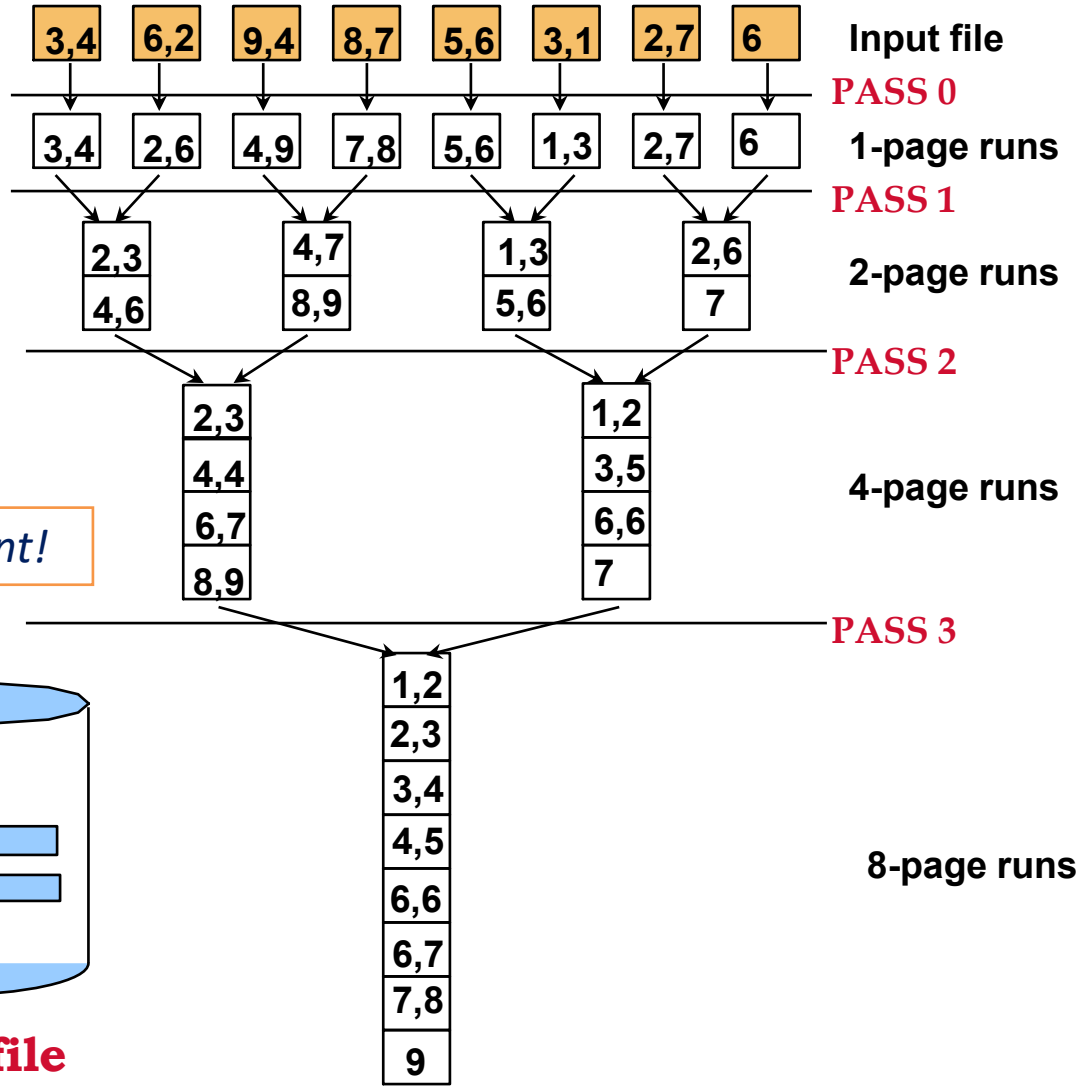
External two-way merge sort

- Needs 3 buffers
- Instead of recursion
  - works bottom-up from the input

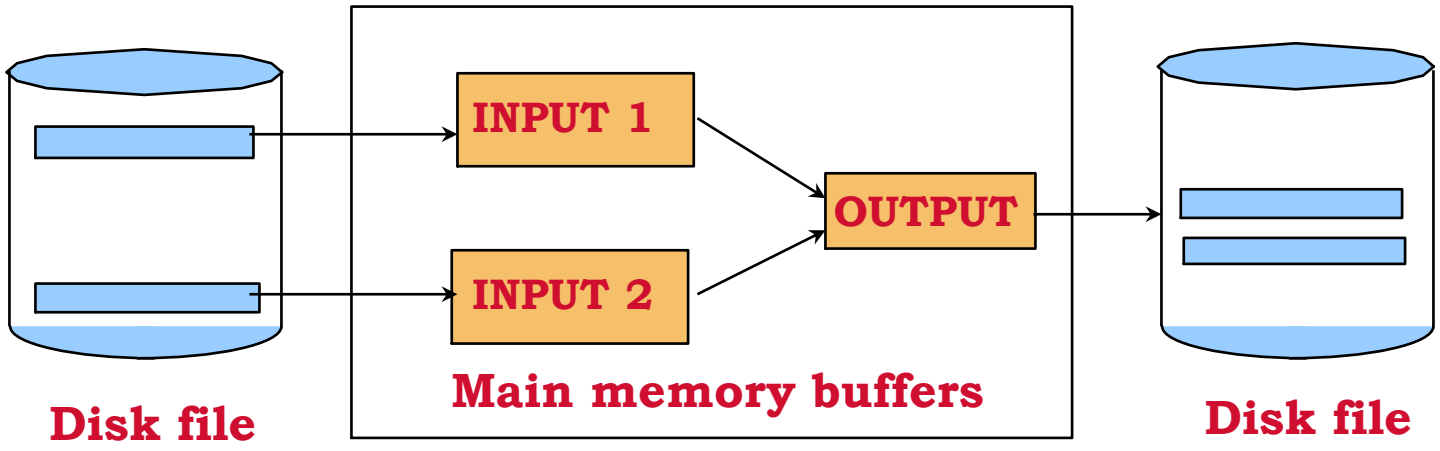


External two-way merge sort

- Input: N pages
- Cost for a pass: reading & writing N pages once
- # of passes: height of the tree = $\lceil \log_2 N \rceil + 1$
- Total cost: $2N(\lceil \log_2 N \rceil + 1)$ I/Os



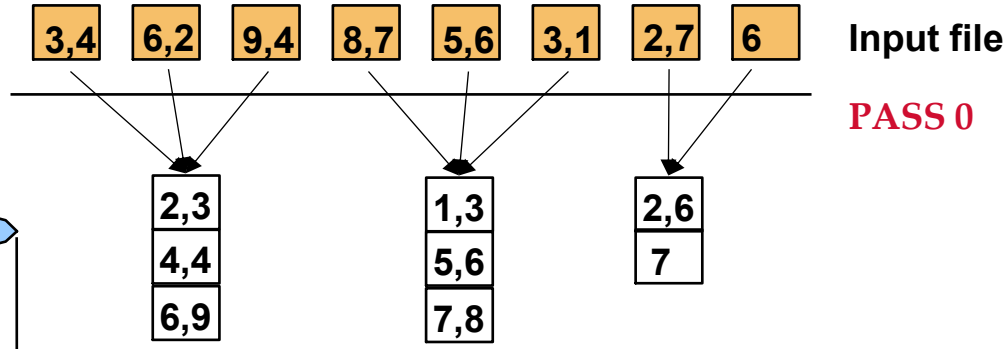
Not so efficient!

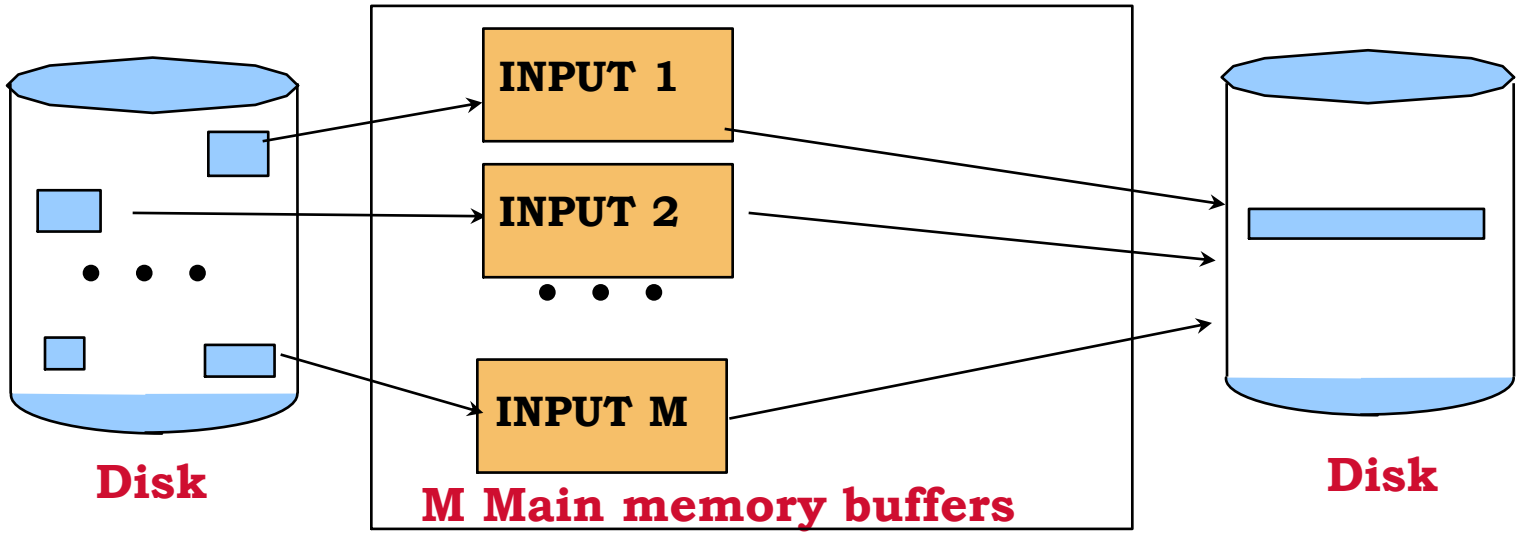


External multi-way merge sort

- How do we fully utilize all the M buffers?
  - Solution: (M-1)-way merge-sort
- Pass 0: internal sort to produce initial runs
  - read every M pages into memory
  - use some internal sorting algorithm (e.g., quick sort)
    - *can produce even larger runs (later)*
  - write all the M pages as a run

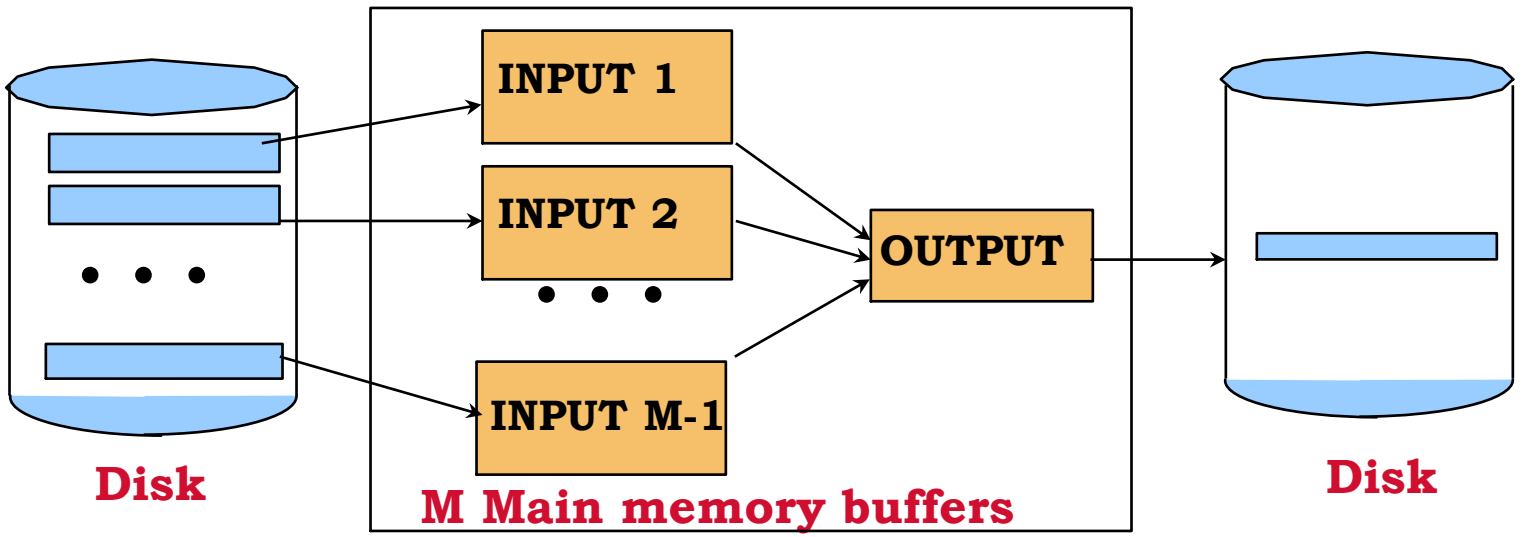
N pages in input
$\lceil \frac{N}{M} \rceil$ runs after pass 0
Cost:
$2N$ pages I/Os

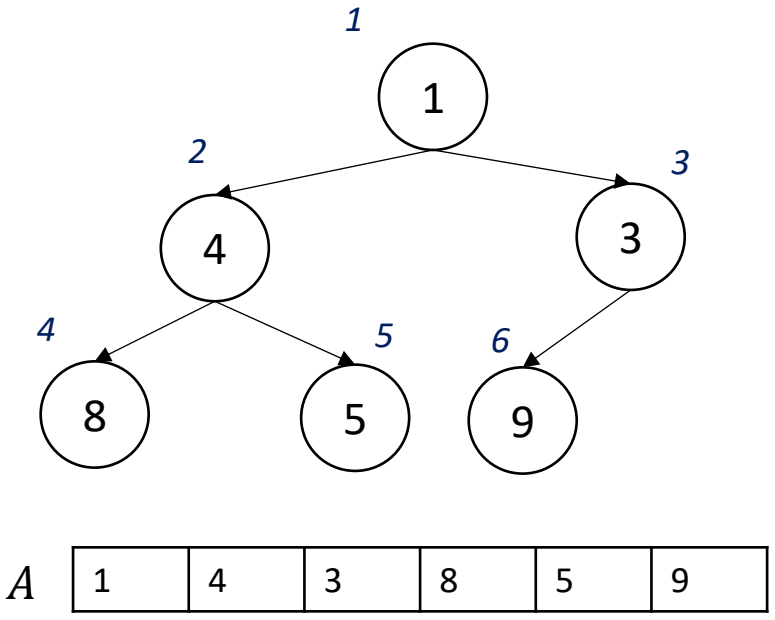




General multi-way merge sort

- Pass 1, 2, …: merge as many runs as possible from previous pass into a sorted run
  - maintain *a min-heap/max-heap (aka priority queue)*
    - supports $O(logM)$ time insertion of any item and deletion of the smallest/largest item
    - a complete binary tree where parent is smaller/larger than both children
    - how to implement
      - numbering nodes level by level sequentially from 1, store in an array $A[1..n]$
        - (how to translate 1-based index to 0-based in C/C++?)
      - parent of $A[i]$ is $A[i/2]$, left child of $A[i]$ is $A[i*2]$, right child of $A[i]$ is $A[i*2+1]$
      - push-down or push-up to maintain the variant

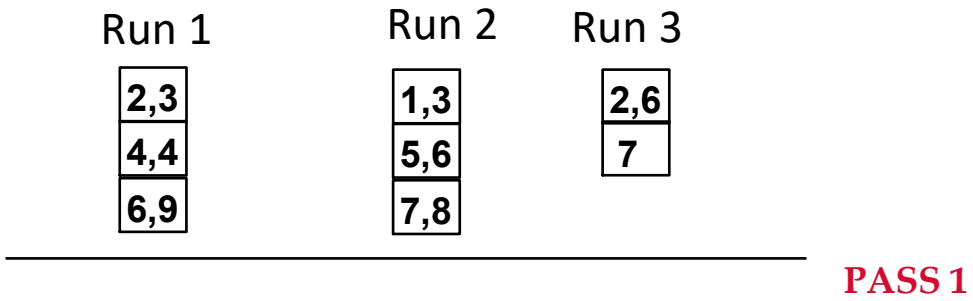




General multi-way merge sort

- Pass 1, 2, …: merge as many runs as possible from previous pass into a sorted run
  - maintain *a min-heap*
  - load one page from each of the $M - 1$ runs
    - and maintain pointers of next page to read
  - for each loaded page
    - insert the first key into the min-heap
    - maintain next slot ids for each page
  - Repeatedly remove the smallest item from the min heap
    - and replace it with the next key in its run
    - write out the output page once it's full

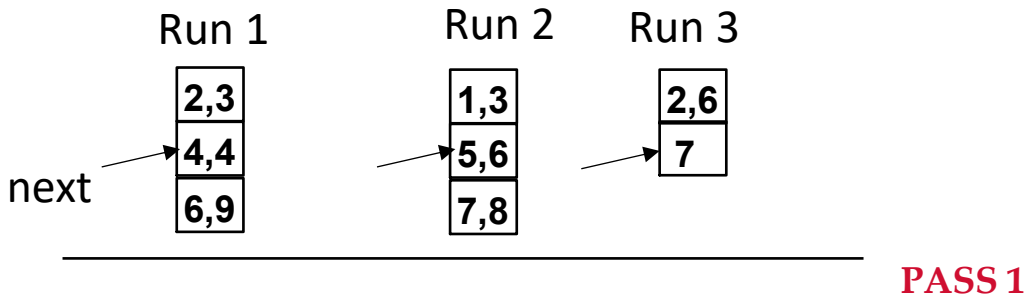
For illustration, let's now assume $M = 4$ instead of 3 from now on.

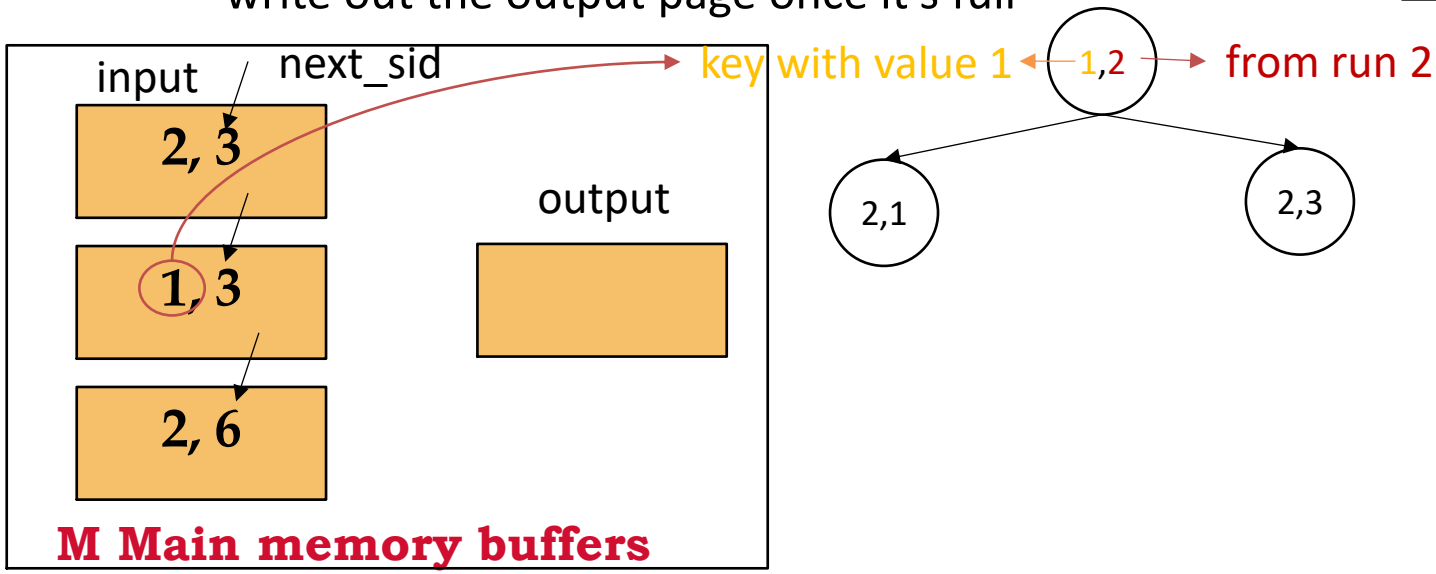


General multi-way merge sort

- Pass 1, 2, …: merge as many runs as possible from previous pass into a sorted run
  - maintain *a min-heap*
  - load one page from each of the $M - 1$ runs
    - and maintain pointers of next page to read
  - for each loaded page
    - insert the first key into the min-heap
    - maintain next slot ids for each page
  - Repeatedly remove the smallest item from the min heap
    - and replace it with the next key in its run
    - write out the output page once it's full

For illustration, let's now assume $M = 4$ instead of 3 from now on.

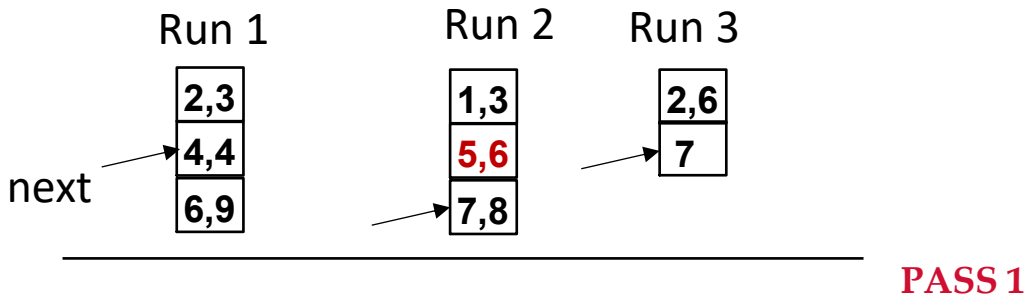


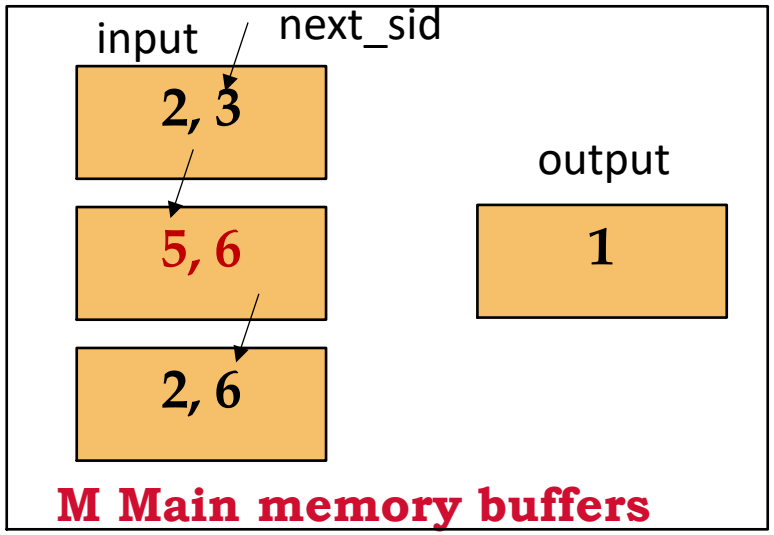


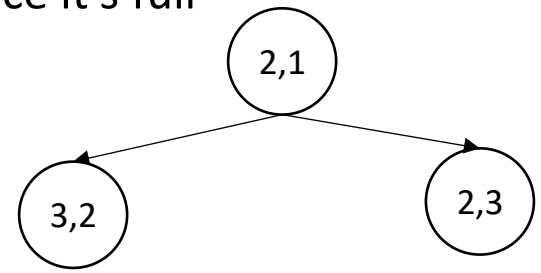
General multi-way merge sort

- Pass 1, 2, …: merge as many runs as possible from previous pass into a sorted run
  - maintain *a min-heap*
  - load one page from each of the $M-1$ runs
    - and maintain pointers of next page to read
  - for each loaded page
    - insert the first key into the min-heap
    - maintain next slot ids for each page
  - Repeatedly remove the smallest item from the min heap
    - and replace it with the next key in its run
    - write out the output page once it's full

For illustration, let's now assume $M = 4$ instead of 3 from now on.



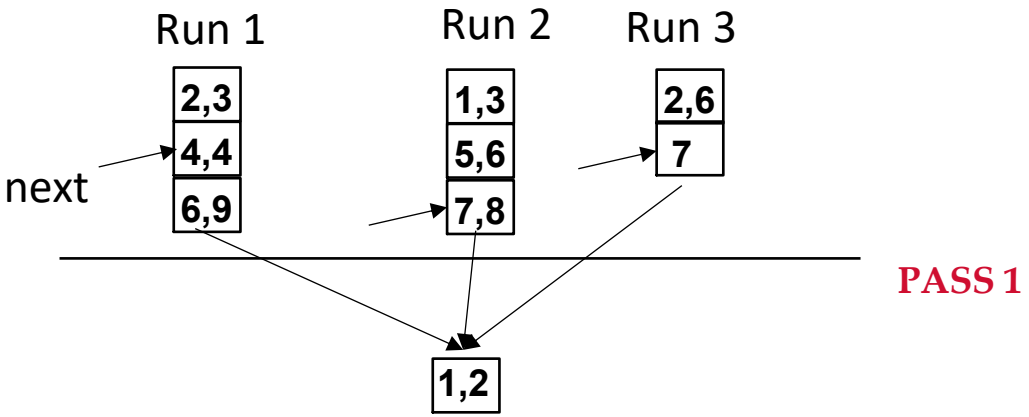


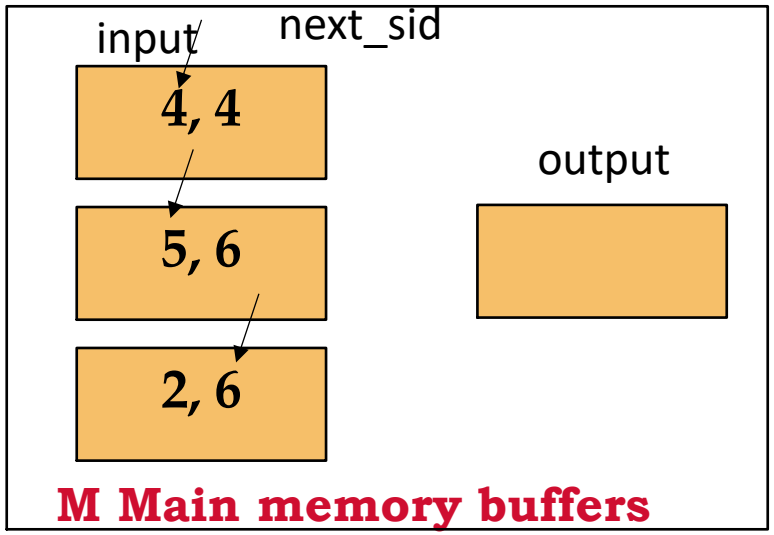


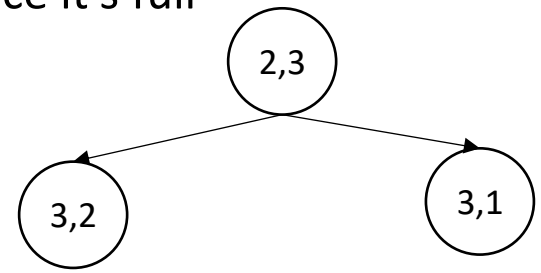
General multi-way merge sort

- Pass 1, 2, …: merge as many runs as possible from previous pass into a sorted run
  - maintain *a min-heap*
  - load one page from each of the $M - 1$ runs
    - and maintain pointers of next page to read
  - for each loaded page
    - insert the first key into the min-heap
    - maintain next slot ids for each page
  - Repeatedly remove the smallest item from the min heap
    - and replace it with the next key in its run
    - write out the output page once it's full

For illustration, let's now assume $M = 4$ instead of 3 from now on.

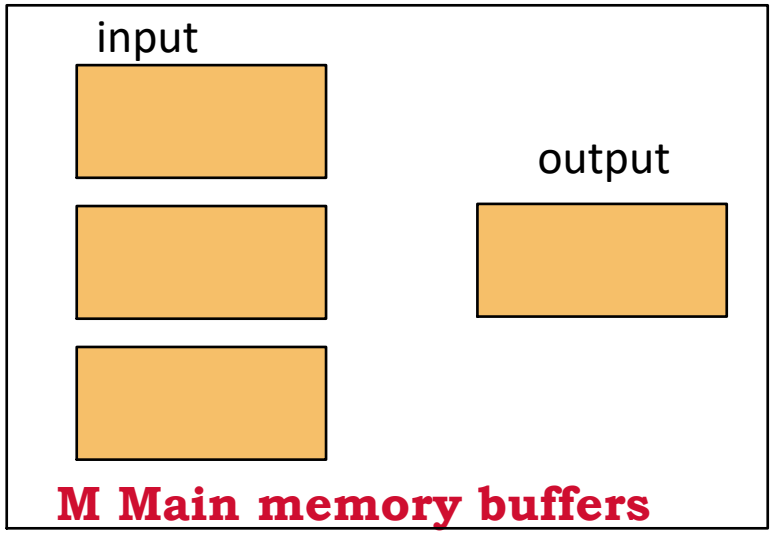




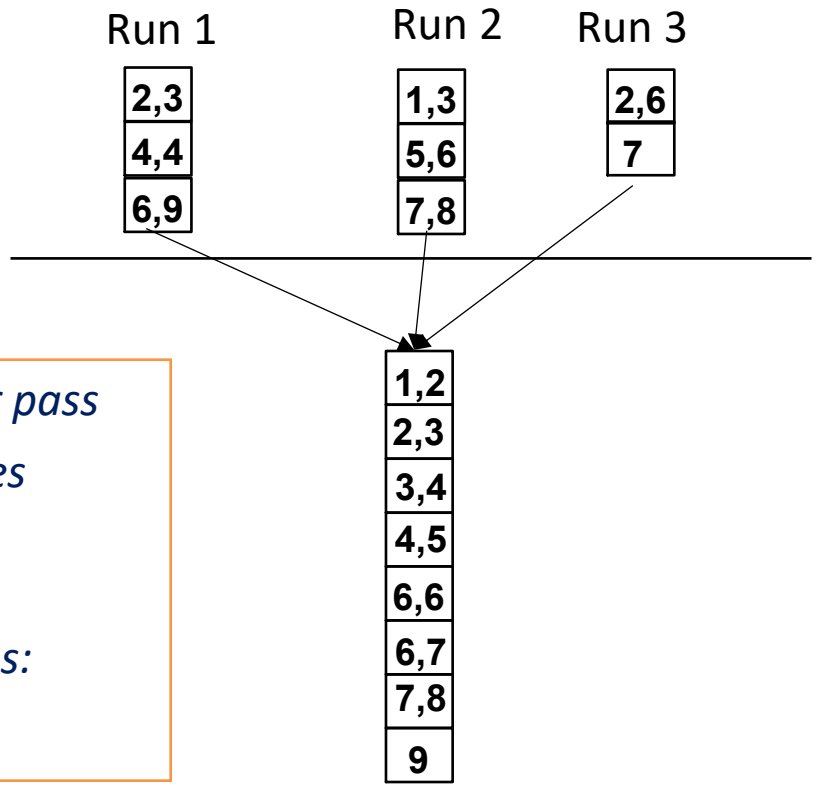


General multi-way merge sort

- Pass 1, 2, …: merge as many runs as possible from previous pass into a sorted run
  - maintain *a min-heap*
  - load one page from each of the $M - 1$ runs
    - and maintain pointers of next page to read
  - for each loaded page
    - insert the first key into the min-heap
    - maintain next slot ids for each page
  - Repeatedly remove the smallest item from the min heap
    - and replace it with the next key in its run
    - write out the output page once it's full



For illustration, let's now assume $M = 4$ instead of 3 from now on.



N pages to read/write per pass

$\lceil log_{M-1} \lceil \frac{N}{M} \rceil \rceil$ merge passes

Cost per merge pass:

$2N$ pages I/Os

Total cost for merge passes:

$2N\lceil \log_{M-1} \lceil \frac{N}{M} \rceil \rceil$

Cost analysis

- Cost analysis:
  - Pass 0: $2N$
  - Pass 1, 2, … combined: $2N\lceil \log_{M-1}\lceil \frac{N}{M} \rceil \rceil$
  - Total = $2N\left(\left\lceil log_{M-1}\left\lceil \frac{N}{M} \right\rceil \right\rceil + 1\right)$

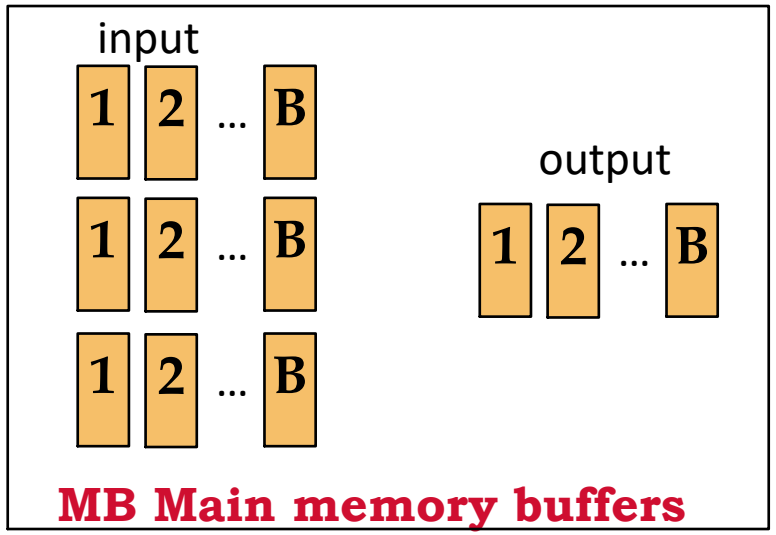
- *gain of utilizing all available buffers*
- *importance of a high fan-in during merging*

N	M=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

- Can we do it better?

Batching I/Os for merge sort

- Refinement 1
  - reducing random I/Os by reading/writing B pages per run during merge
  - using $(M-1)$-way merge sort
    - memory usage increases to MB pages
    - number of pages transferred do not change
    - but the number of random seeks per merge pass reduced to approximately $2\lceil\frac{N}{B}\rceil$
  - Total I/O cost becomes $2N\left(\left\lceil log_{M-1}\left\lceil\frac{N}{MB}\right\rceil\right\rceil+1\right)$



Pipelining output

- Refinement 2
  - in most cases, do not need to write the final file
    - pipelining to the next operator
    - or output to user
  - Hence, no need to count the write of the final pass
  - total cost reduced to $N\left(2\left\lceil log_{\lfloor \frac{M}{B} \rfloor - 1}\left\lceil \frac{N}{M} \right\rceil \right\rceil + 1\right)$