External Sorting

External Sorting

- Problem: Sort 1GB of data with 1MB of RAM.
- When a file doesn't fit in memory, there are two stages in sorting:
  1. File is divided into several segments, each of which sorted separately
  2. Sorted segments are *merged.*

  (<u>Each stage involves reading and writing the file at least once</u>)

How to sort segments

- Any efficient sorting algorithm (such as Quicksort) can be used to sort segments
- Each sorted segment is called a **run**.
- Each run will be the size of the available memory.

External sort for Large Files

Basic idea

1. Form runs (i.e. sorted segments):
   - bring as many records as possible to main memory, sort them, save it into a small file on disk.
   - Repeat this until we have read all records from the original file and written them as sorted segments (i.e. runs) to disk.
2. Do a multiway merge of the runs.

External sort for Large Files (cont.)

- How big is each run?
  - As big as the available memory.
- What is the time it takes to create all sorted segments?
  - Ignoring the seek time and assuming *b* blocks in the file, where heap processing overlaps (approximately) with I/O.
  - The time for creating the initial sorted segments is **2*b***btt** (read in the file as segments and write out the runs)
    - Note that the entire file has not been sorted yet. These are just sorted segments, and the size of each segment is limited to the size of the available memory used for this purpose.

Multiway Merging

- K-way merge: we want to merge K sorted input lists to create a single sorted output list. (K is the order of a K-way merge)
- We will adapt the 2-way merge algorithm:
  - Instead of two lists, keep an array of lists: list[0], list[1], … list[k-1]

Multiway Merging to sort Large Files

Let us consider the following example:

- File to be sorted:
  - 8,000,000 records
  - R = 100 bytes

⇒Total file size = 800MB

- Memory available as a work area: 10MB (not counting memory used to hold program, other variables, O.S., I/O buffers etc.)

Step 1: Create Runs

Disk I/Os are performed in the following procedures:

1. Reading records into main memory for sorting
2. Writing runs to disk.

These two steps are done as follows:

- Read 10MB, write 10MB (repeat this 80 times, because file is 800MB). So there will be 80 runs
  - In terms of basic disk operations, this operation costs:
    - For reading: 80 seeks + a total transfer time of 800 MB
    - Same for writing: 80 seeks + a total transfer time of 800 MB

Step 2: Multiway Merging

- Read runs into memory for merging.
  - Available memory is 10 MB and each run size is 10MB.
  - There are 80 runs.
  - We need to do an 80-way merge.
  - 10M / 80 = 125,000 bytes are available for each list to be merged.
- Read one piece of each run. Each piece can have 1250 records (because record size is 100)
  - How many pieces to be read for each run?

    Size of run/size of chunk = 10,000,000/125,000= 80 pieces

    Reading each piece involves average seeking.

Step 2 (continues)

- Total # of seeks = Total # of pieces (counting all runs):

  There are 80 runs and 80 pieces in each run:

  ✓ 80^2 pieces = 6400 seeks for reading

- Writing the merged records to the big sorted file:
- The number of separate writes closely approximates reads:

  ✓ 80^2 pieces = 6400 seeks for writing

- So we estimate two seeks (one for reading and one for writing) for each piece: 80* 80 pieces therefore **a total of 2*6400 = 12800 seeks are necessary.**

Sorting a File that is 10 times larger

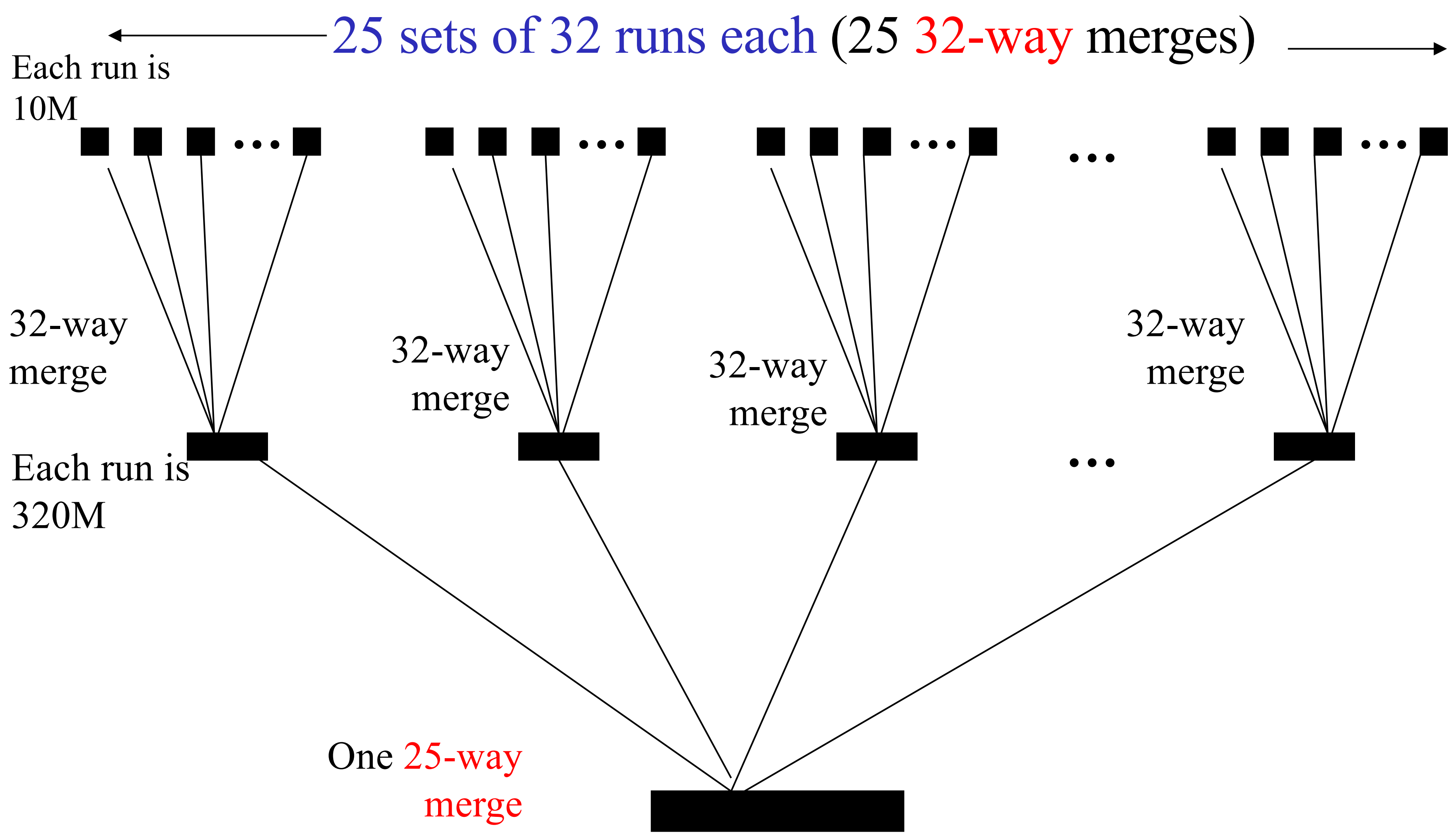
- How is the time for merge phase affected if the file is 80 million records?
  - More runs: 800 runs
  - 800-way merge in 10MB memory
  - i.e. divide the memory into 800 lists.
  - Each list holds 1/800th of a run
  - So, 800 runs * 800 seeks/run = 640,000 seeks
  - Total = 2 * 640000 seeks

The cost of increasing the file size

- In general, for a K-way merge of K runs, the buffer size for each run is
  - (1/K) * size of memory space = (1/K) * size of each run
- So K seeks are required to read all of the records in each run and K seeks to write records.
- Since there are K runs, merge requires a total of $2K^2$ seeks.
- Because K is directly proportional to N it also follows that the total cost is an $O(N^2)$ operation.

Multiple-step multiway merges

- Instead of merging all runs at once, we break the original set of runs into small groups and merge the runs in these groups separately.
  - more memory space is available for each run; hence fewer seeks are required per run.
- When all of the smaller merges are completed, a second pass merges the new set of merged runs.



Two-step merge of 800 runs

Cost of multi-step merging

- 25 sets of 32 runs, followed by 25-way merge:
  - Disadvantage: we read every record twice.
  - Advantage: we can use larger lists and avoid a large number of disk seeks.
- Calculations:

  First Merge Step:
  - List size = 1/32 run => 32*32 = 1024 seeks
  - For 25 32-way merges=> 25 * 1024 = 25,600 seeks

Second Merge Step:

- For each of the 25 final runs, 1/25 memory space is allocated.
- So each input list is 0.4M and it can hold 4000 records (or 1/800 run)
- Hence, 800 seeks per run, so we end up making 25 * 800 = 20,000 seeks.

Total number of seeks for reading in two steps:

25600 + 20000 = 45,600

- What about the total time for merge?
  - We now have to transmit all of the records 4 times instead of two.
  - We also write the records twice, requiring an extra 45,600 seeks.
  - **Total number of seeks : 91200 seeks**
- Still the trade is profitable.

Total Cost

Total cost of block transfers $= 2b*btt+(\log_k m)*2b*btt$

$= 2b*btt*(\log_k m +1)$

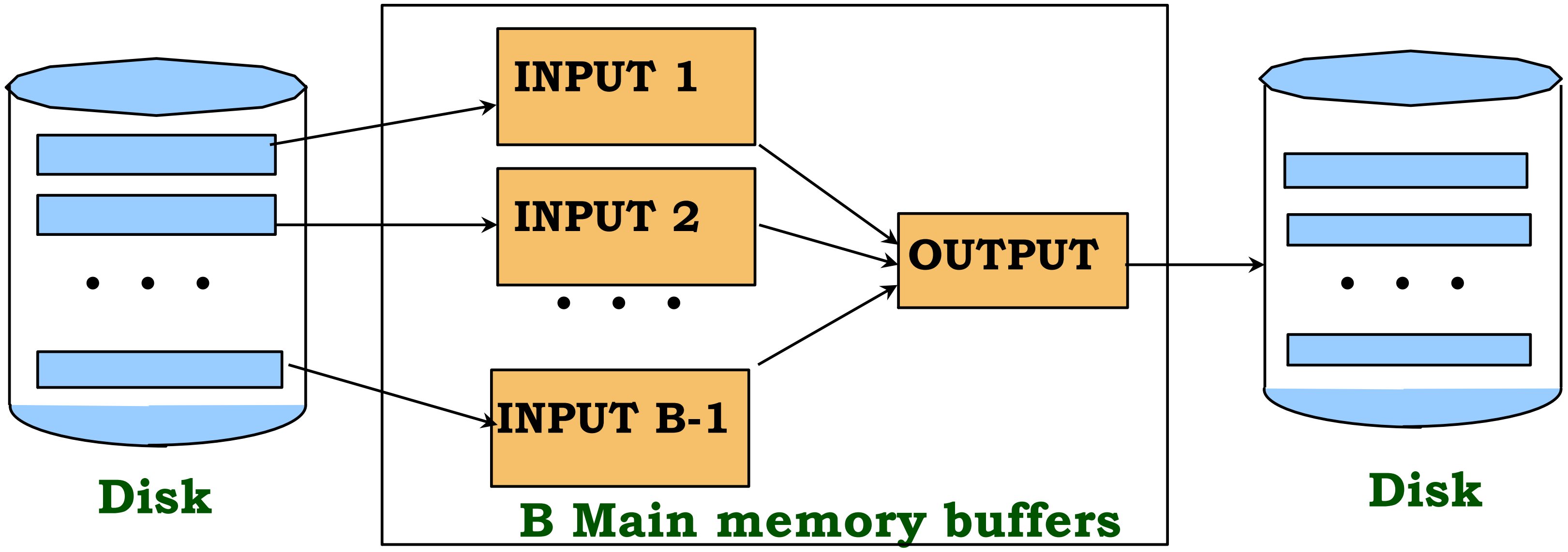
(when k-way merge is used for m runs.)

Total cost including # of seeks:

$$= \underbrace{2b*btt}_{\text{creating sorted runs}} + \underbrace{(\log_k m)* [2k*m*(s+r) + 2b*btt]}_{\text{k-way multi-step merge (handled in } \log_k m \text{ steps)}}$$

DBMS view (from Raghu's book)

- Sort a file with *N* blocks (pages)
- Available memory *B* blocks (buffer pages):
  - Pass 0: use *B* buffer pages. Produce $\lceil N / B \rceil$ 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)
  - 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

Problem

Assume you are given a file of 800,000 records, 400 bytes each, a block size of 4000 bytes with btt of 1 msec, and the available memory size is 8MB. You are asked to sort this file using 4-way merge with one disk drive. Assume s = 5 ms, r = 4 ms, btt= 1ms.

1. How many runs (sorted segments) are created ?
2. What is the size of each run?
3. Estimate the cost of creating these runs.

Solution

1. How many runs (sorted segments) are created?

   Number of runs = *file size/ size of avail.memory* = $800000*400/\ 8000000$ = 40 sorted segments

2. What is the size of each run?

   Size of each run = size of available memory = 8MB

3. Estimate the cost of creating these runs

- Number of blocks = b = $800000\ *400/4000=80000\ blocks$
- Cost of creating the runs = $2*b*btt$=$2*80000*(1ms)$=$160000\ ms$

Problem (cont.)

4. What is the number of passes necessary to merge the sorted segments using 4-way merge?

Number of passes = ⌈ $\log_4 40$ ⌉ = 3

5. Estimate the cost of one pass of merging.

Cost of one pass of merging =

= 2 * *p* * (# of runs) * (s+r) + 2 * b * btt

where *p* is the number of pieces in each run

= 2 * 4 * 40 * (9 ms) + 2 * 80000 * (1ms) = 2880 ms + 160000 ms = 162880 ms

Problem (cont.)

6. Estimate the total cost of sorting the file, including the creation of sorted runs and the multi- step merging.

Total cost of sorting the file =
 # of passes * cost for one pass of merging + cost of heap sort

= 3 * (162880 ms) + 160000 ms = 648640 ms