CS3223 Tutorial 3

Gary Lim

Agenda

Chapter Review

Tutorial Questions

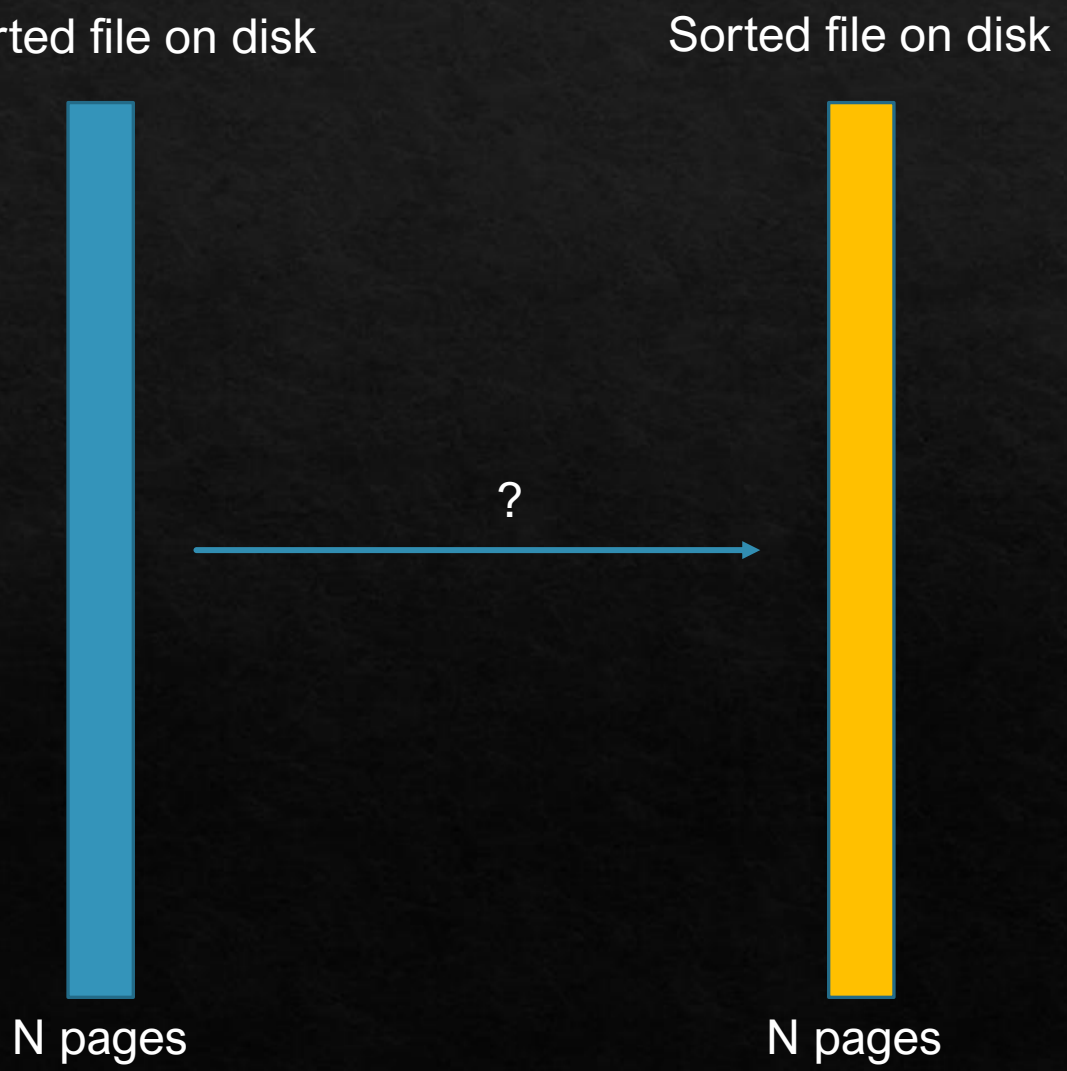
Q & A

Chapter Review

1. External Sort
2. Internal Sort – Replacement Selection
3. Sequential I/O vs Random I/O

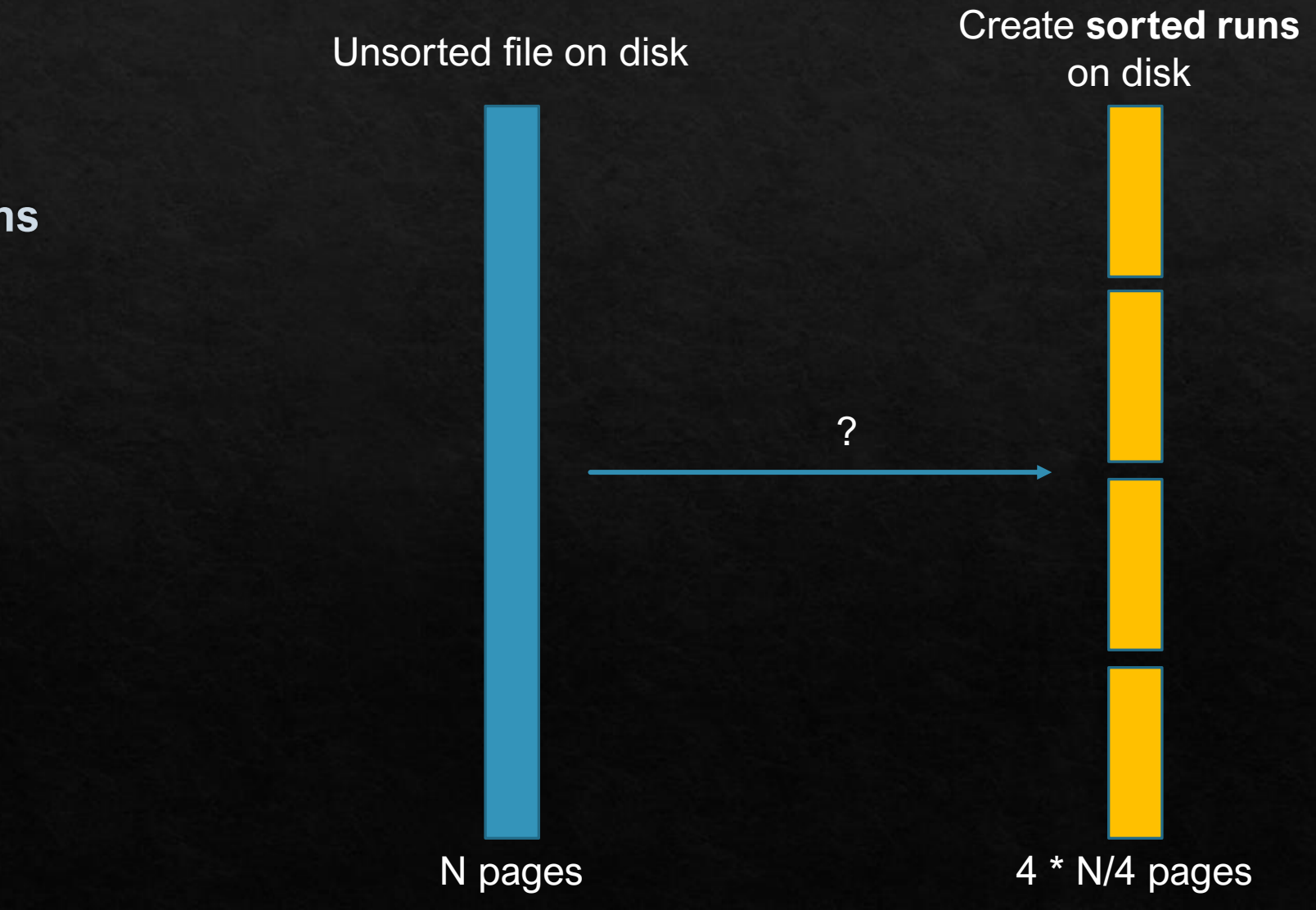
External Sort

- Why do we need external sort?
  - Data requested in **sorted order**
  - Can’t perform in-memory sort on entire file
- 2 Phases:
  - Generate sorted runs
  - Merge sorted runs



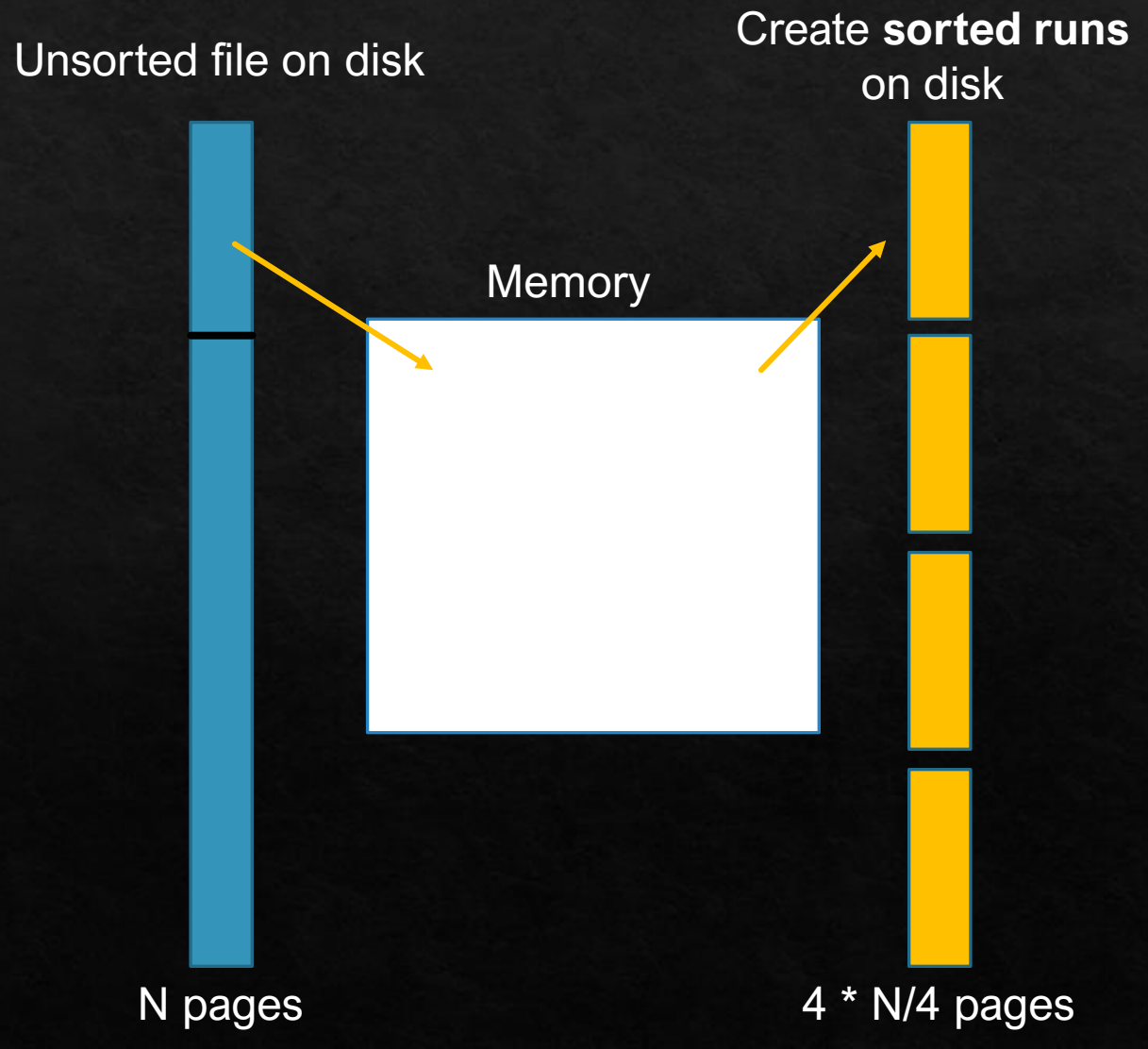
External Sort

- 2 Phases:
  - **Generate sorted runs**
  - Merge sorted runs



External Sort

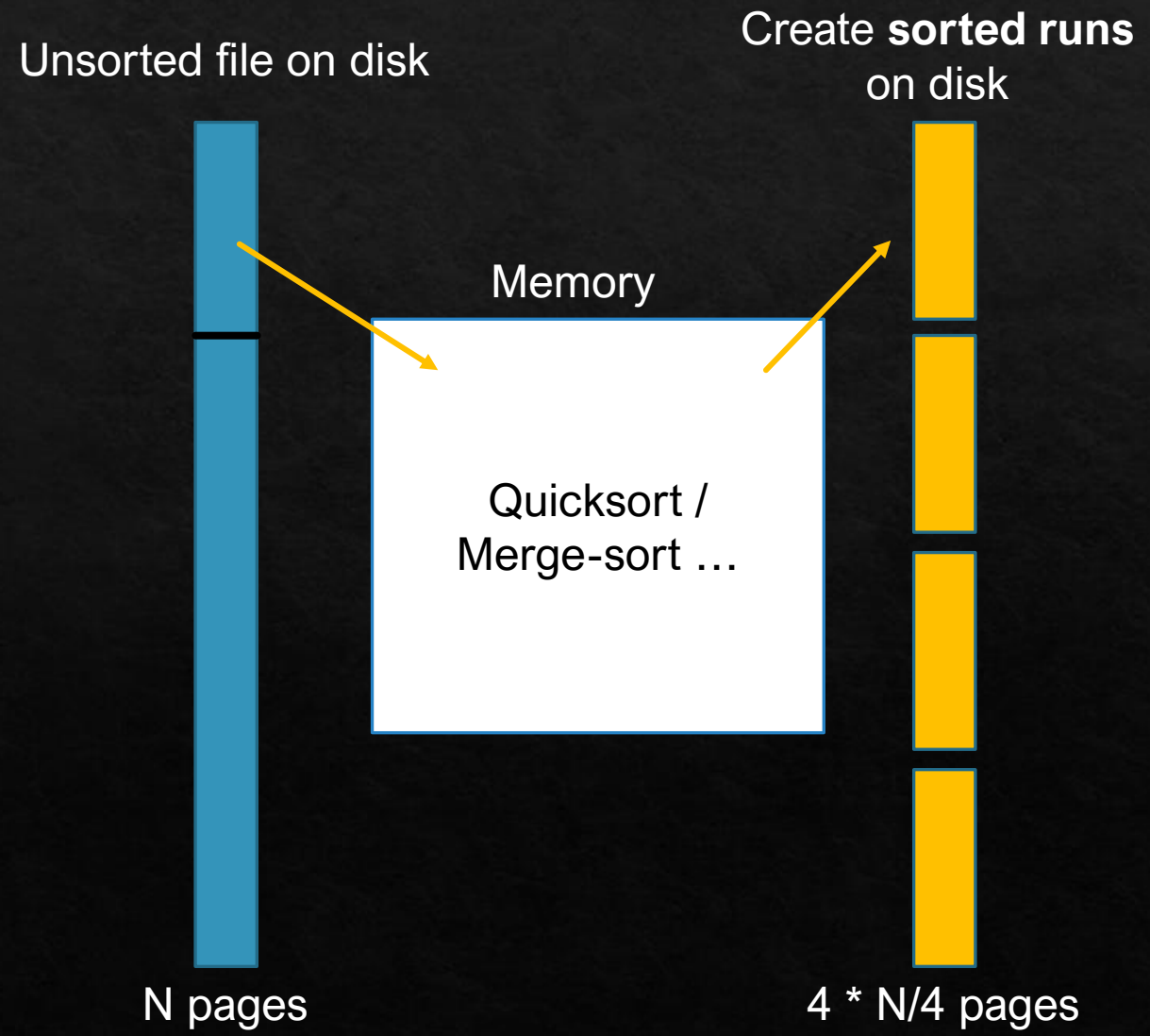
- 2 Phases:
  - **Generate sorted runs**
  - Merge sorted runs



External Sort

- 2 Phases:
  - **Generate sorted runs**
    - **Cost: 2N**
  - Merge sorted runs

What is the size (in # pages) of one sorted run?

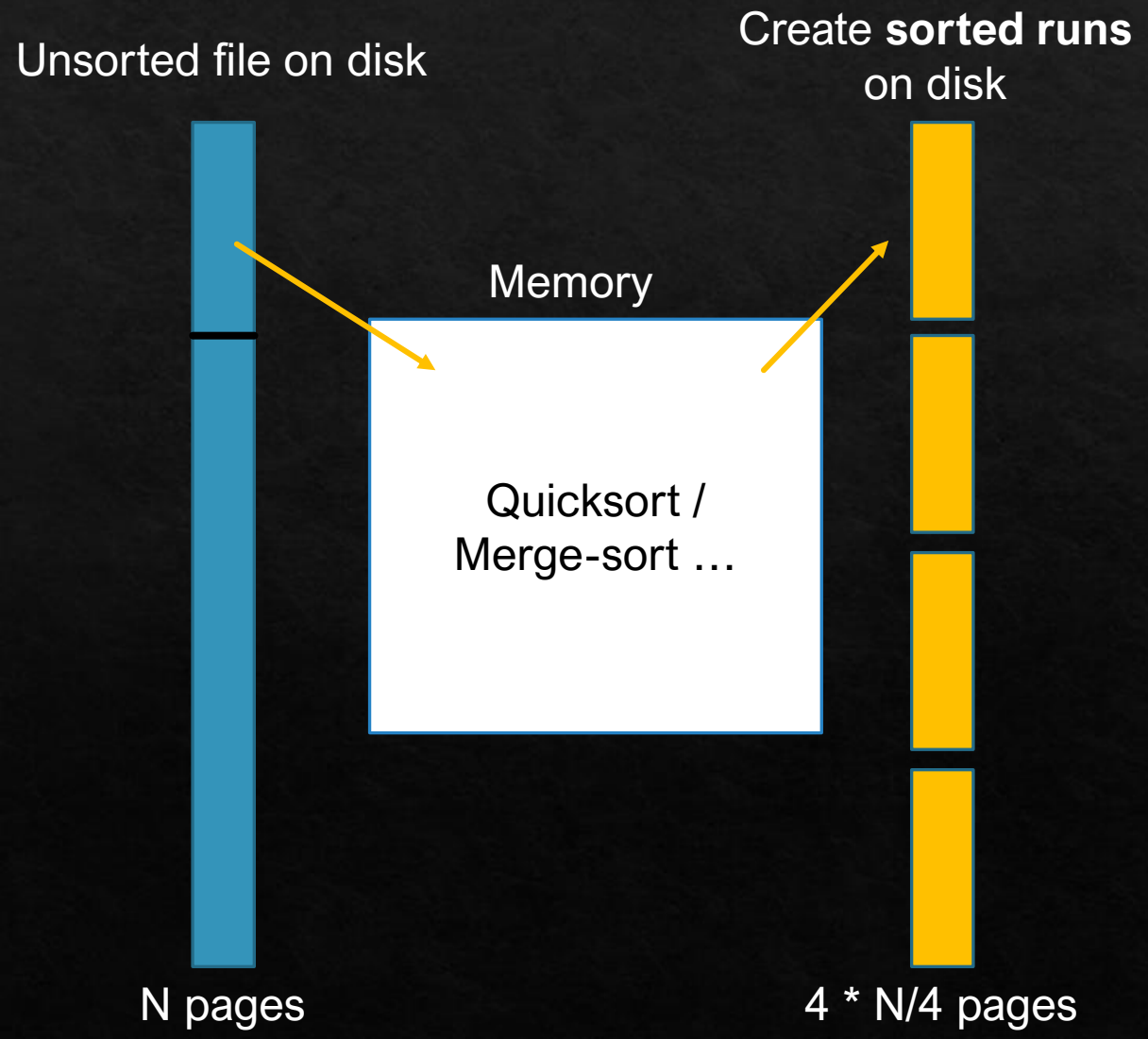


External Sort

- 2 Phases:
  - **Generate sorted runs**
    - **Cost: 2N**
  - Merge sorted runs

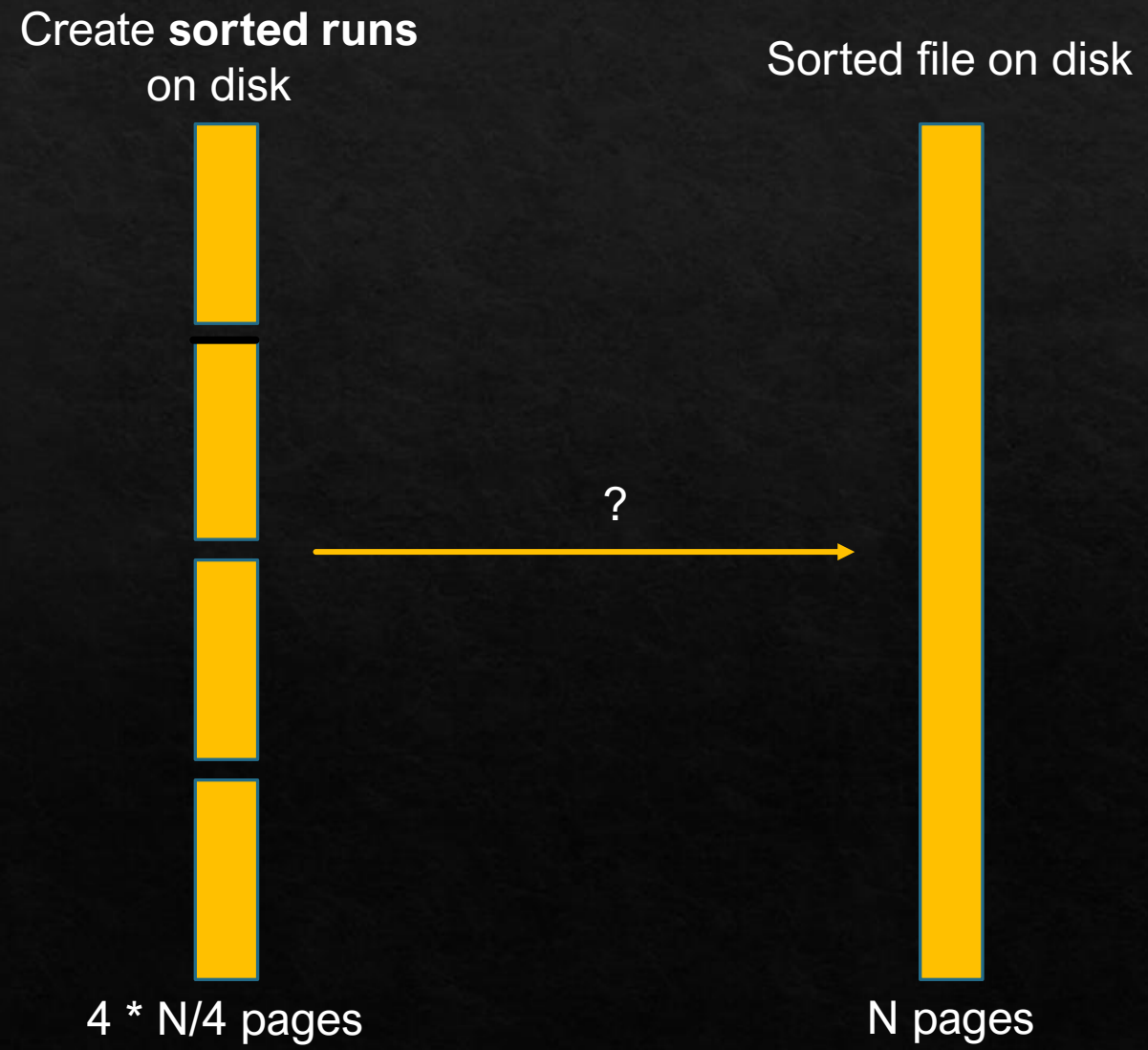
What is the size (in # pages) of one sorted run?

Size of memory buffer.



External Sort

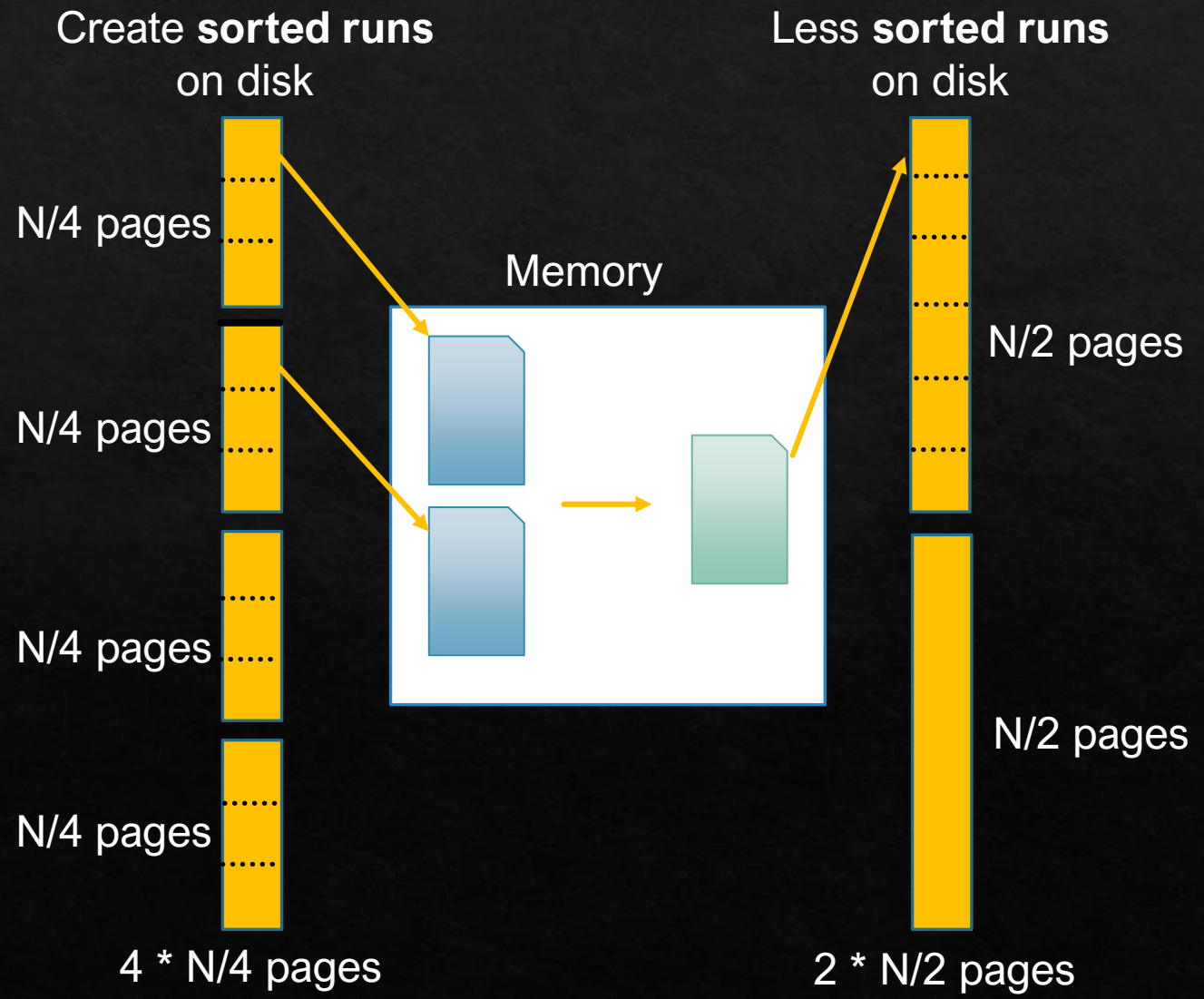
- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - **Merge sorted runs**



External Sort

- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - **Merge sorted runs**

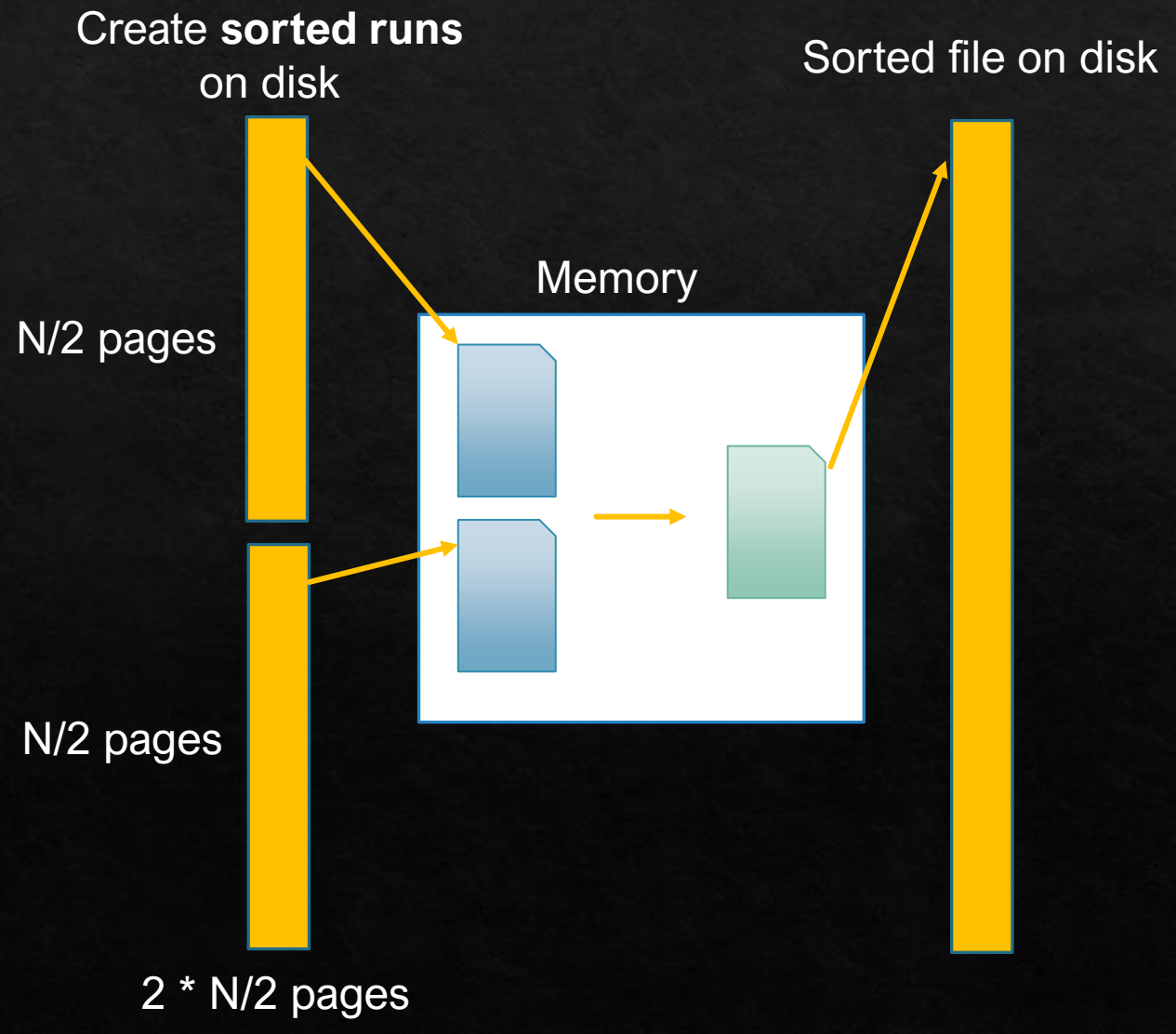
Assume memory can hold 3 pages (i.e. N/4 = 3 pages)



External Sort

- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - **Merge sorted runs**

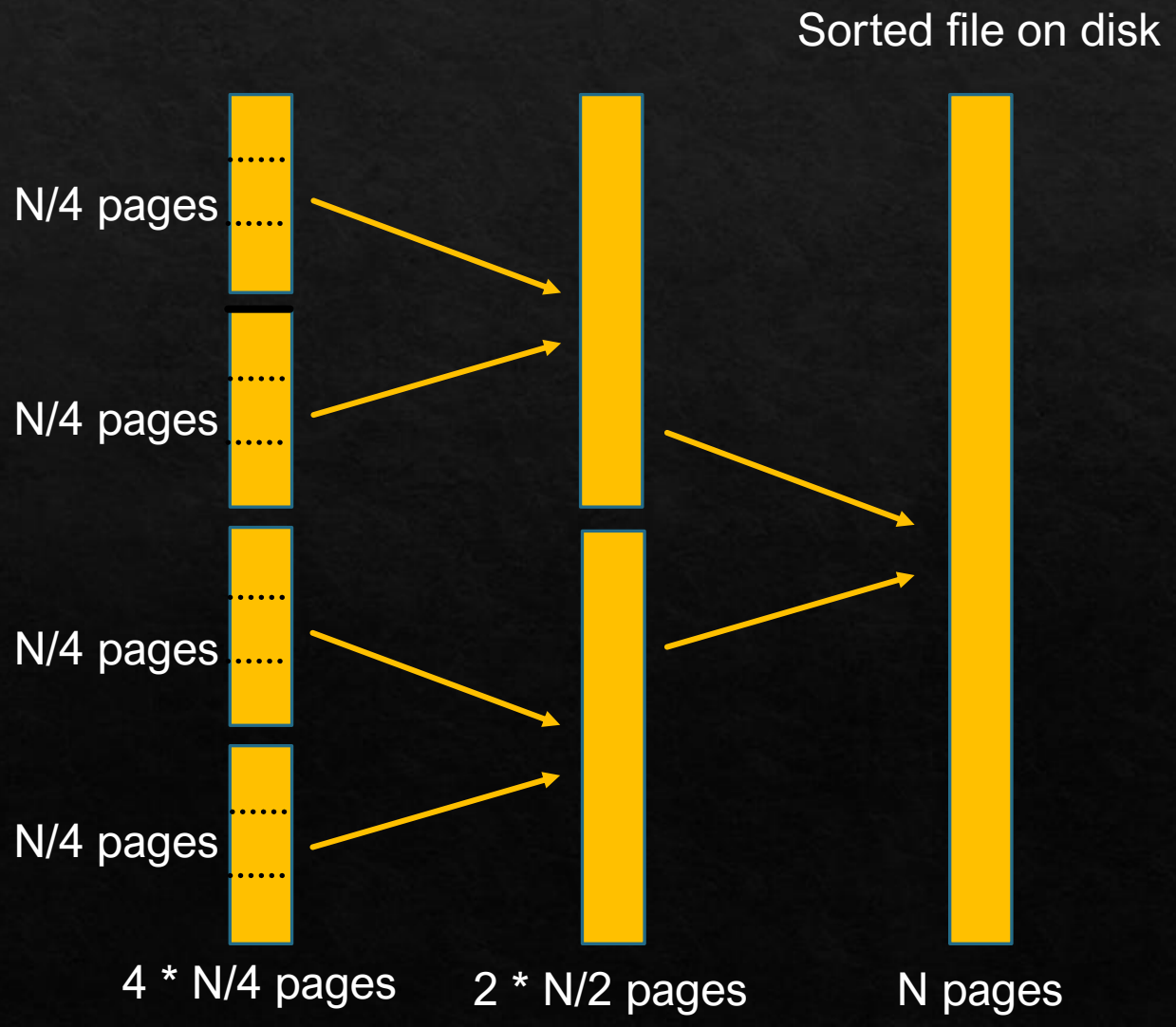
Assume memory can hold 3 pages (i.e. N/4 = 3 pages)



External Sort

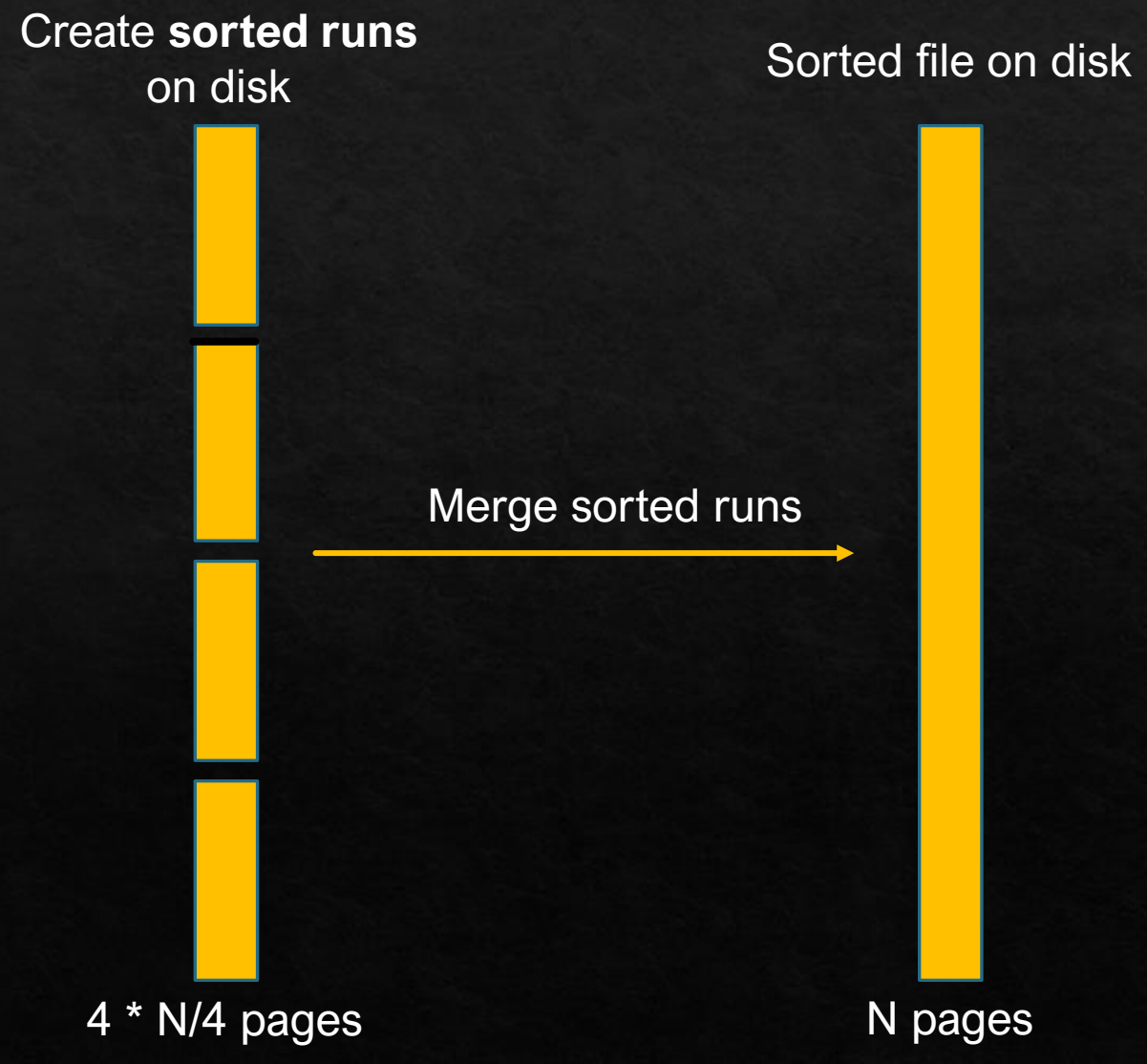
- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - **Merge sorted runs**
    - **# of passes: [$\log_{B-1}$([N / B])]**

B = size of memory
[N / B] = # sorted runs initially



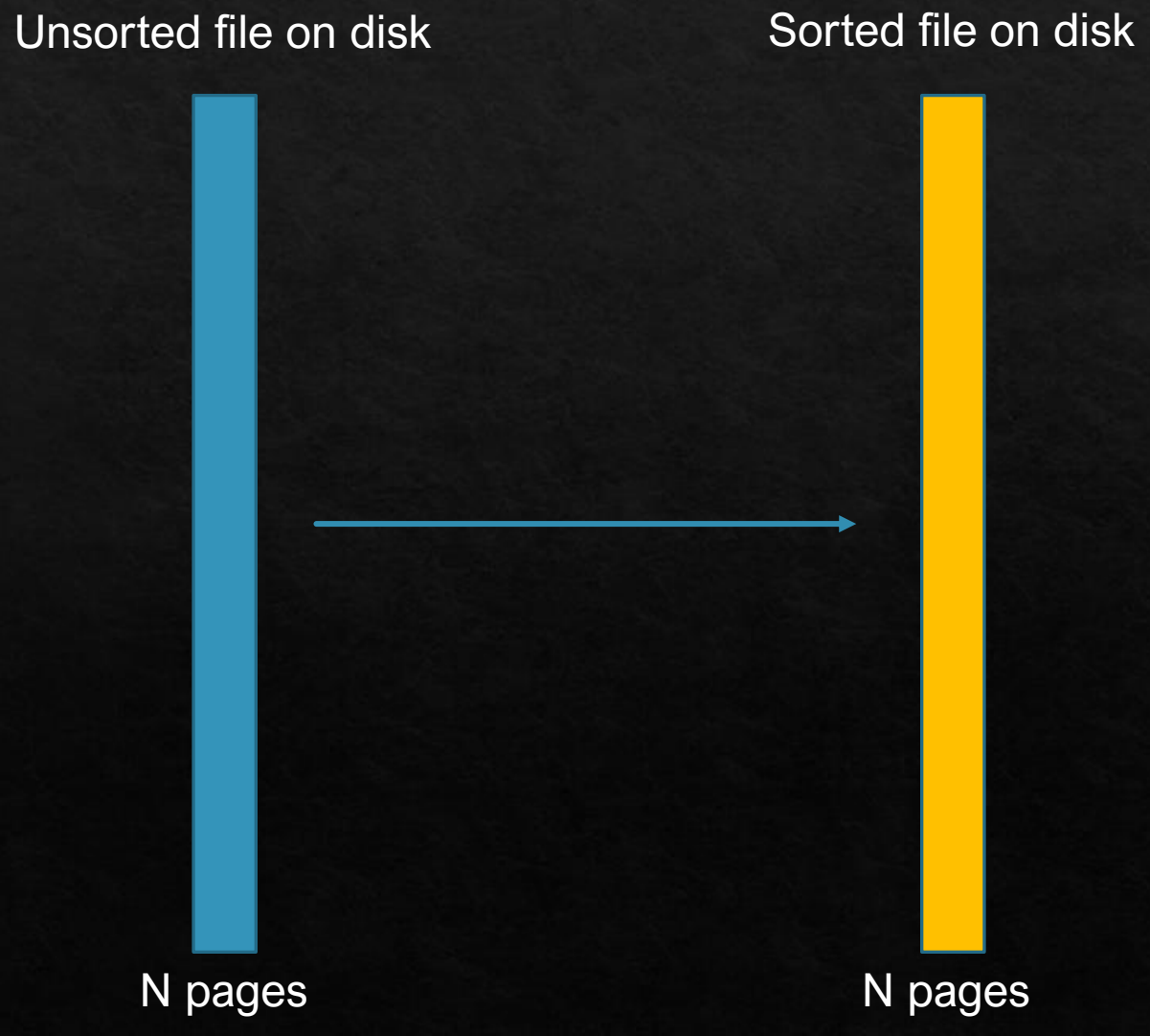
External Sort

- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - **Merge sorted runs**
    - **Cost:** $\lceil \log_{B-1}(\lceil N / B \rceil) \rceil * 2N$



External Sort

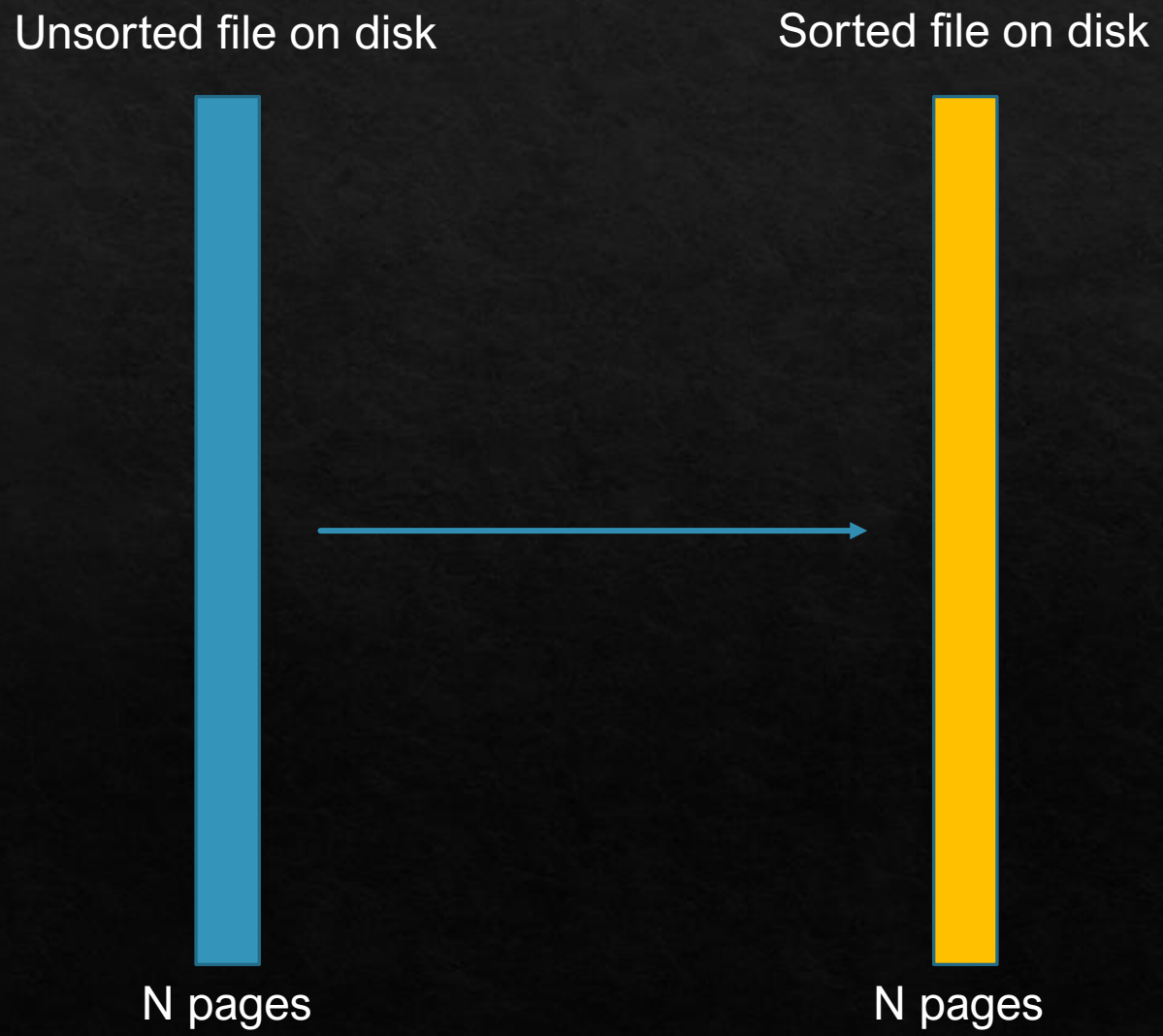
- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - Merge sorted runs
    - Cost: $\lceil \log_{B-1}(\lceil N / B \rceil) \rceil * 2N$
  - **Total Cost:**
    - $(\lceil \log_{B-1}(\lceil N / B \rceil) \rceil + 1) * 2N$



External Sort

- 2 Phases:
  - Generate sorted runs
    - Cost: 2N
  - Merge sorted runs
    - Cost: $\lceil \log_{B-1}(\lceil N / B \rceil) \rceil * 2N$
  - **Total Cost:**
    - $(\lceil \log_{B-1}(\lceil N / B \rceil) \rceil + 1) * 2N$

This is the total number of passes on the file



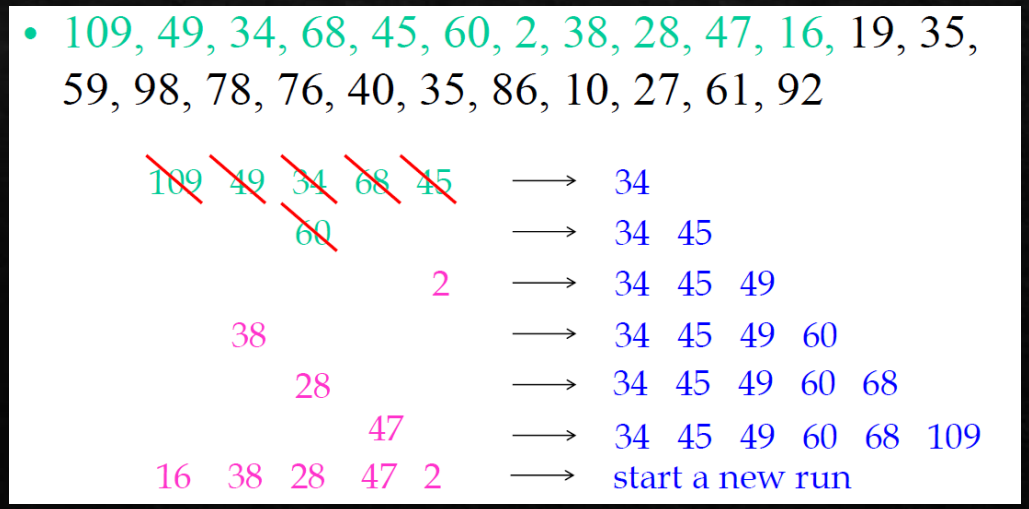
Replacement Selection

- A way to reduce **number of sorted runs**, and hence number of passes
- Average length of a run: **2B**
- Min length of a run: **B**
  - Worst case
  - When file is sorted in descending order
- Max length of a run: **N**
  - Best case
  - When file is sorted in ascending order

Entry is freeze if it is less than / equal to the last flushed value

Recall:

Total Cost:
$(\lceil \log_{B-1}(\lceil N / B \rceil) \rceil + 1) * 2N$



Sequential vs Random I/O

- Sequential I/O:
  - When retrieving one entire file, we assume all its pages are stored sequentially on the same track of the disk, therefore no seek time / rot delay between pages
- Random I/O:
  - When retrieving a different file (e.g. another sorted run), we incur additional seek time as the sorted run is likely to be stored at a different location of the disk

Tutorial Questions

Q1

Suppose that you just finished inserting several records into a heap file and now want to sort those records. Assume that the DBMS uses **external sort** and makes efficient use of the available buffer space when it sorts a file. Here is some potentially useful information about the newly loaded file and the DBMS software available to operate on it: The **number of records in the file is 4500. The sort key for the file is 4 bytes** long. You can assume **that pointers are 8 bytes long. Each record is a total of 48 bytes long**. The **page size is 512 bytes**. Each page has **12 bytes of control information (e.g. like metadata)** on it. **Four buffer pages** are available.

Q1a

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be?

- Available space per page: ? bytes

Q1a

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be?

- Available space per page: 512 – 12 = **500 bytes**
- # of records per page: ?

Q1a

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be?

- Available space per page: 512 – 12 = **500 bytes**
- # of records per page: floor(500 / 48) = **10 records**
- Total # of pages in the file: ?

Q1a

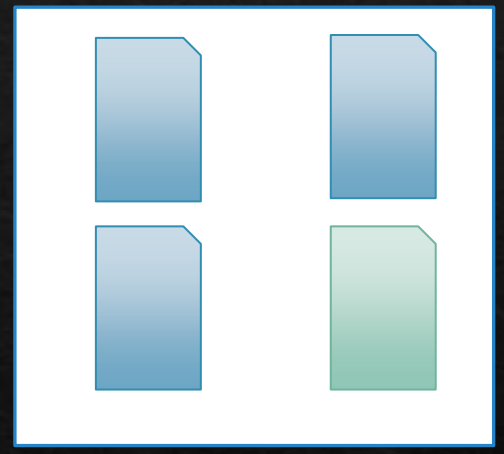
- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be?

- Available space per page: 512 – 12 = **500 bytes**
- # of records per page: floor(500 / 48) = **10 records**
- Total # of pages in the file: 4500 / 10 = **450 pages**
- Number of sorted runs: ? Size of sorted run: ?

Q1a

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

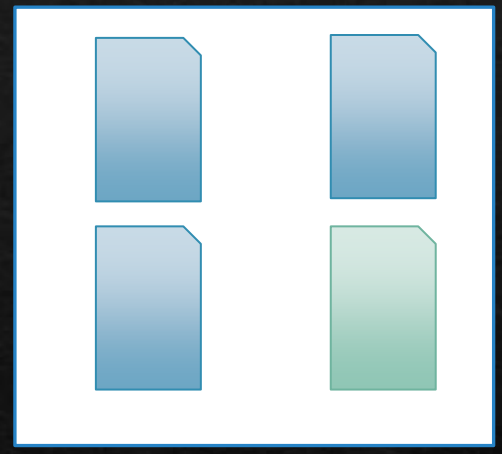


How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be?

- Available space per page: 512 – 12 = **500 bytes**
- # of records per page: floor(500 / 48) = **10 records**
- Total # of pages in the file: 4500 / 10 = **450 pages**
- Number of sorted runs: ceil(450 / 4) = **113** Size of each sorted run: **4 pages**

Q1b

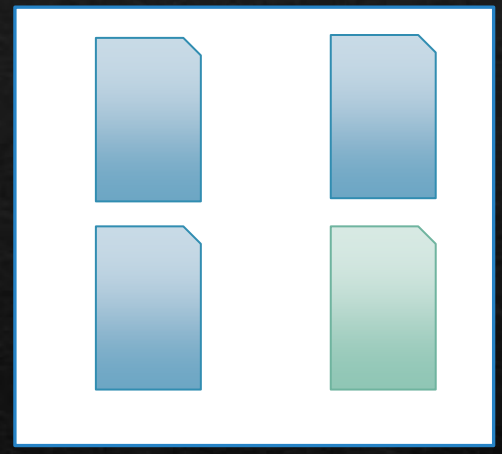
- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**



How many passes (including the initial pass just considered) are required to sort this file?

Q1b

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**



How many passes (including the initial pass just considered) are required to sort this file?

- Number of passes in 2nd phase:
  - $\lceil \log_{B-1}(\lceil N / B \rceil) \rceil =$
  - $\lceil \log_3(113) \rceil =$ **5**
- **Total number of passes: 5 + 1 = 6**

Q1c

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

What is the total I/O cost for sorting this file?

Q1c

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

What is the total I/O cost for sorting this file?

- Number of passes * 2N =
- 6 * 2 * 450 = **5400 I/O**

Q1d

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

i. The index uses Alternative (1) for data entries.

ii. The index uses Alternative (2) and is unclustered. (You can compute the worst-case cost in this case.)

Q1d

Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

i. The index uses Alternative (1) for data entries.

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

Q1d

Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

i. The index uses Alternative (1) for data entries.

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

Assume 67% occupancy (if you assume 100% occupancy, answer will be different)

Number of leaf pages = 450 / 0.67 = 672

2d * 4 + (2d + 1) * 8 <= 500

d = 20, max 40 entries per node, but 67% occupancy so about **27 entries** **(and 28 pointers)** **per internal node,**

About 672/28 = 24 internal nodes above leaf level, 1 root node above this internal node level.

Total I/O = 1+1+672 = 674

Q1d

Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

ii. The index uses Alternative (2) and is unclustered. (You can compute the worst-case cost in this case.)

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

Q1d

Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

ii. The index uses Alternative (2) and is unclustered. (You can compute the worst-case cost in this case.)

- **number of records in the file: 4500**
- **sort key: 4 bytes**
- **pointers: 8 bytes**
- **record: 48 bytes**
- **page size: 512 bytes, 12 bytes used for control information**
- **Buffer pages: 4**

Each node can hold up to 40 entries, assuming 67% occupancy, **27 keys (28 ptrs)** per node.

There will be 4500 / 27 = 167 leaf nodes

Number of levels in B+ tree (including leaf level) = 1 + ceil($\log_{28}(167)$) = 1 + 2 = 3

Since unclustered, when scanning B+ tree leaf pages, each entry will require fetching a data page (means one I/O per entry)

Total I/O = 4500 + 167 + 3 = 4670 I/O

Q1d

- Unclustered index is really bad if you wish to scan the entire index (to retrieve a sorted file)

Q2a

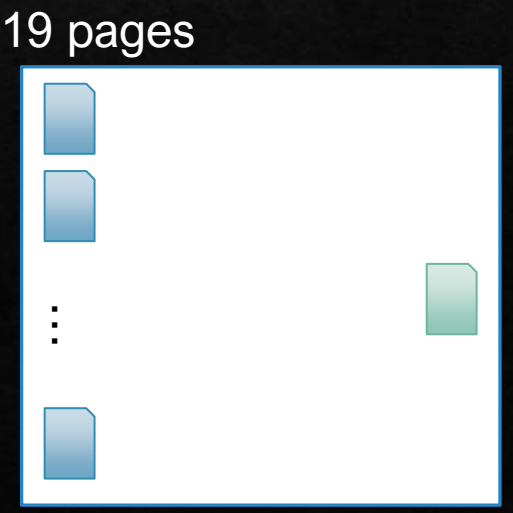
Consider a disk with an **average seek time of 10ms**, **average rotational delay of 5ms** and a transfer time of **1ms for a 4K page**. Assume that the cost of reading/writing a page is the sum of these values (i.e., 16 ms) unless a sequence of pages is read/written. In this case, the cost is the average seek plus the average rotational delay (to find the first page in the sequence) plus 1 ms per page (to transfer data). You are given 320 buffer pages and asked to sort a file with 10,000,000 pages. Assume that you begin by creating **sorted runs of 320 pages** each in the first pass. Evaluate the cost of the following approaches for the subsequent merging passes:

- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each

Q2a

- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each (10,000,000 pages in total)

Do 319-way merges. If there are fewer than 319 runs to merge, then allocate one input buffer per run, and one output buffer, i.e., ignore the other “spare” buffer pages.



Q2a

- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each (10,000,000 pages in total)

passes = ceil($\log_{319}(31250)$) = 2

Do 319-way merges. If there are fewer than 319 runs to merge, then allocate one input buffer per run, and one output buffer, i.e., ignore the other "spare" buffer pages.

- 1st pass: 31250 * 320 pages * (10+5+1) * 2 = **320 * 10^6 ms**
  - 31250 / 319 = 98 sorted runs
- 2nd pass: 10,000,000 pages * (10+5+1) * 2 = **320 * 10^6 ms**
- Total: **640 * 10^6 ms**

All pages read and written individually

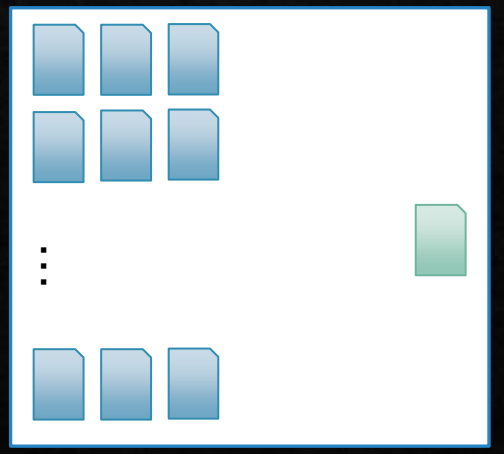
Q2a

- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each (10,000,000 pages in total)

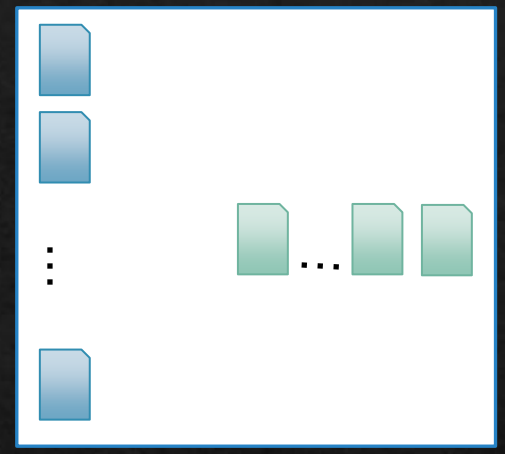
Do 319-way merges. If there are fewer than 319 runs to merge, then allocate one input buffer per run, and one output buffer, i.e., ignore the other "spare" buffer pages.

- 1st pass: 31250 * 320 pages * (10+5+1) * 2 = **$320 * 10^6$ ms**
  - 31250 / 319 = 98 sorted runs
- 2nd pass: 10,000,000 pages * (10+5+1) * 2 = **$320 * 10^6$ ms**
- Total: **$640 * 10^6$ ms**

Technically we have spare buffers to optimize in the second pass (319 buffers, 98 runs) but can ignore since question says allocate one buffer per run



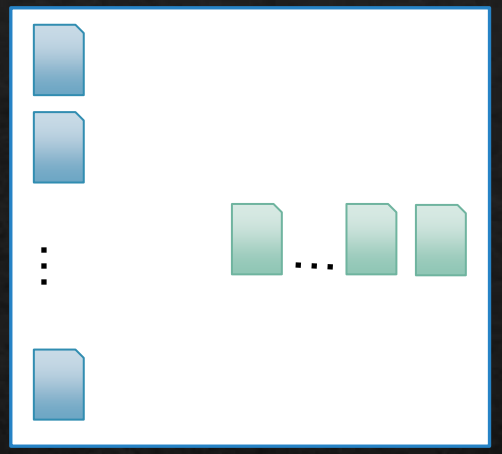
Q2b



- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each (10,000,000 pages in total)

Create 256 ‘input’ buffers of 1 page each, create an ‘output’ buffer of 64 pages, and do 256-way merges.

Q2b



- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page

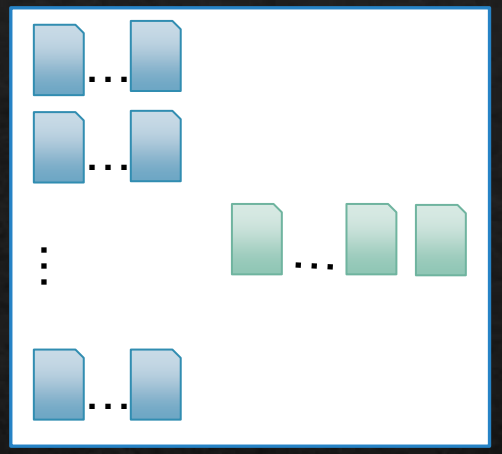
passes = ceil($\log_{256}(31250)$) = 2

- Initial sorted runs: 31250 sorted runs of 320 pages each (10,000,000 pages in total)

Create 256 ‘input’ buffers of 1 page each, create an ‘output’ buffer of 64 pages, and do 256-way merges.

- 1st pass: 10,000,000 pages * (10+5+1) + 10,000,000 / 64 * (10+5 + 64 * 1)
  - 31250 / 256 = 123 sorted runs of 81920 pages each
- 2nd pass: 10,000,000 pages * (10+5+1) + 10,000,000 / 64 * (10+5 + 64 * 1)
- Total: **344,687,500 ms**

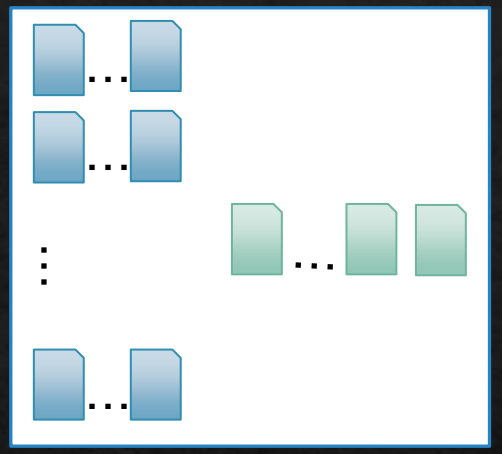
Q2c



- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each

Create 16 ‘input’ buffer of 16 pages each, create an ‘output’ buffer of 64 pages, and do 16-way merges.

Q2c



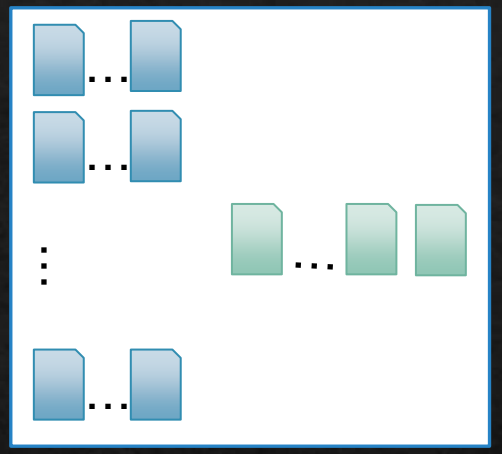
- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each

passes = ceil($\log_{16}(31250)$) = 4

Create 16 ‘input’ buffer of 16 pages each, create an ‘output’ buffer of 64 pages, and do 16-way merges.

- 1st pass: 10,000,000 / 16 * (10 + 5 + 16*1) + 10,000,000 / 64 * (10 + 5 + 64*1)
- 2nd pass: 10,000,000 / 16 * (10 + 5 + 16*1) + 10,000,000 / 64 * (10 + 5 + 64*1)
- 3rd pass: 10,000,000 / 16 * (10 + 5 + 16*1) + 10,000,000 / 64 * (10 + 5 + 64*1)
- 4th pass: 10,000,000 / 16 * (10 + 5 + 16*1) + 10,000,000 / 64 * (10 + 5 + 64*1)
- Total: **126,875,000 ms**

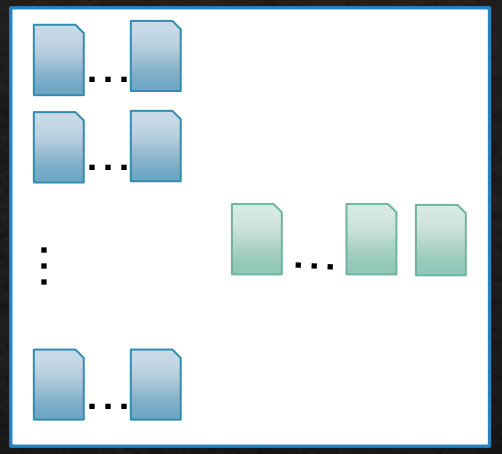
Q2d



- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page
- Initial sorted runs: 31250 sorted runs of 320 pages each

Create 4 ‘input’ buffer of 64 pages each, create an ‘output’ buffer of 64 pages, and do 4-way merges.

Q2d



- Avg seek time: 10ms
- Avg rot delay: 5ms
- Transfer time: 1ms per 4k page

passes = ceil($\log_4(31250)$) = 8

- Initial sorted runs: 31250 sorted runs of 320 pages each

Create 4 ‘input’ buffer of 64 pages each, create an ‘output’ buffer of 64 pages, and do 4-way merges.

- Each pass: 10,000,000 / 64 * (10 + 5 + 64*1) + 10,000,000 / 64 * (10 + 5 + 64*1)
- Total: **197,500,000 ms**

Q2

What conclusion(s) can you draw from these results?

319 buffer x 1 page, 1 output pages: 640,000,000 ms

256 buffer x 1 page, 64 output pages: 344,687,500 ms

16 buffer x 16 pages, 64 output pages: 126,875,000 ms

4 buffer x 64 pages, 64 output pages: 197,500,000 ms

Q2

What conclusion(s) can you draw from these results?

(a) The cost of merging phase varies from a low of 126,875,000 ms to a high of 640,000,000 ms.

(b) The highest cost is associated with the option of maximizing fanout, **choosing a buffer size of 1 page**! Thus the effect of blocked I/O is significant.

However, as the block size is increased, the number of passes increases slowly, and there is a tradeoff to be considered:

- Increase block size, less random I/O and more sequential I/O
- Increase block size, less blocks per pass, more passes

Optimization problem

Note that in practice, normally the read block and write block has the same size

Q3

Suppose we have a relation whose n tuples each require R bytes, and we have a machine whose main memory M bytes and disk-block size B are just sufficient to sort the n tuples in 2 passes (i.e., first pass to generate runs, and second pass to merge the runs). How would the maximum n change if we change one of the parameters as follows?

a) Double B

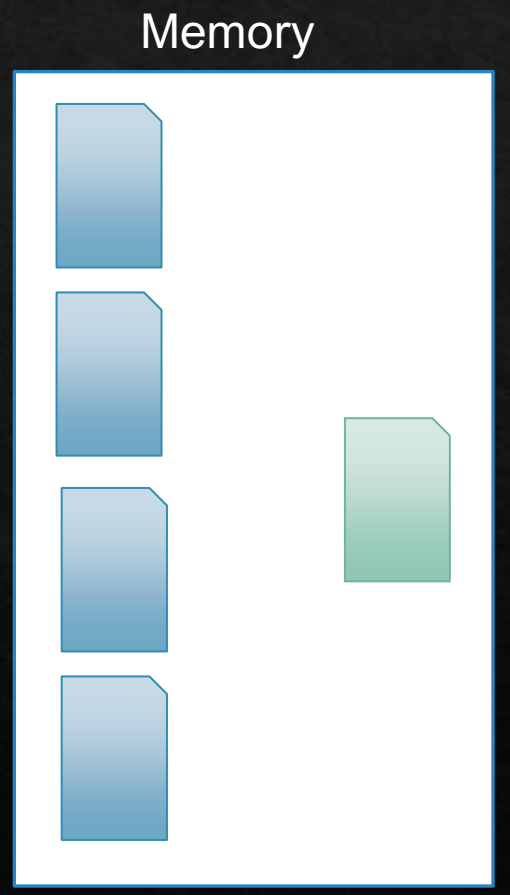
b) Double R

c) Double M

Q3

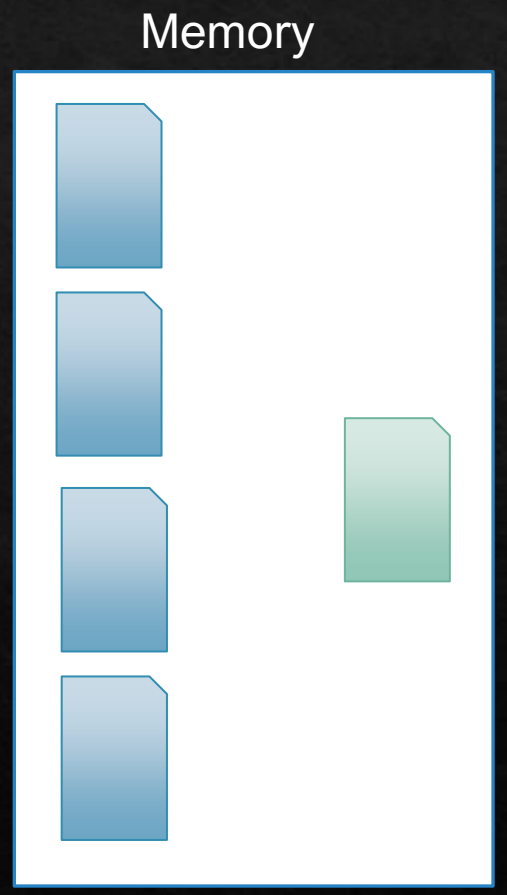
Suppose we have a relation whose **n tuples each require R bytes**, and we have a machine whose main memory **M bytes** and disk-block **size B** are just sufficient to sort the n tuples in 2 passes (i.e., first pass to generate runs, and second pass to merge the runs).

- The number of sorted sublists s we need is $s = nR/M$.
- On the 2^{nd} pass, we need one block for each sublist, plus one for the merged list. Thus,
  - $s < M/B$
  - $Bs < M$.
- Substituting for s, we get $nRB/M < M$, or $n < M^2/RB$.



Q3 Alternative

Suppose we have a relation whose **n tuples each require R bytes**, and we have a machine whose main memory **M bytes** and disk-block **size B** are just sufficient to sort the n tuples in 2 passes (i.e., first pass to generate runs, and second pass to merge the runs).



From Yap Dian Hao (T11)

- # of available buffer pages = M / B
- # of pages in relation = nR / B
- # sorted runs = (nR / B) / (M/B) = nR / M
- To merge in one pass, nR / M <= M/B-1
- Substituting for s, we get nRB/M < M, or n < M^2/RB.

Q3

n – number of tuples

R – size of tuple

M – size of main memory

B – size of each block

$n < M^2/RB$

a) Double B

b) Double R

c) Double M

Q3

n – number of tuples

R – size of tuple

M – size of main memory

B – size of each block

$n < M^2/RB$

a) Double B – halves n (the larger the block size, the smaller the relation we can sort in a 2-phase merge sort – primarily due to less blocks available)

b) Double R

c) Double M

Q3

n – number of tuples

R – size of tuple

M – size of main memory

B – size of each block

$n < M^2/RB$

a) Double B – halves n (the larger the block size, the smaller the relation we can sort in a 2-phase merge sort – primarily due to less blocks available)

b) Double R – same as above, halves n. (with larger tuple size, we are only able to sort a smaller relation within 2-phase merge sort)

c) Double M

Q3

n – number of tuples

R – size of tuple

M – size of main memory

B – size of each block

$n < M^2/RB$

a) Double B – halves n (the larger the block size, the smaller the relation we can sort in a 2-phase merge sort – primarily due to less blocks available)

b) Double R – same as above, halves n. (with larger tuple size, we are only able to sort a smaller relation within 2-phase merge sort)

c) Double M – n increases by 4 times! (more memory means we can sort more records)

Additional Question

Additional Question

Consider the refinement to the external sort algorithm that produces runs of length 2B on average, where B is the number of buffer pages (this is the replacement selection algorithm). This refinement was described in Section 11.2.1 **under the assumption that all records are the same size**. Explain why this assumption is required and extend the idea to cover the case of variable length records.

Additional Question

Consider the refinement to the external sort algorithm that produces runs of length 2B on average, where B is the number of buffer pages (this is the replacement selection algorithm). This refinement was described in Section 11.2.1 **under the assumption that all records are the same size**. Explain why this assumption is required and extend the idea to cover the case of variable length records.

The assumption that all records are of the same size is used when the algorithm moves the smallest entry with a key value larger than k to the output buffer and replaces it with a value from the input buffer.

This ”replacement” will only work if the records are of the same size. If the entries are of variable size, then we must also keep track of the size of each entry, and replace the moved entry with a new entry that fits in the available memory location