

CS660: Intro to Database Systems

# Class 9: External Sorting

Instructor: Manos Athanassoulis

<https://bu-disc.github.io/CS660/>

# External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using B<sup>+</sup>-Trees for sorting

# Why Sort?

a *classic problem* in computer science!

but also a *database specific* problem, with many use cases:



# Why Sort?

a *classic problem* in computer science!

but also a *database specific* problem, with many use cases:

(i) data requested in sorted order

e.g., find students in increasing *gpa* order (using ORDER BY)

(ii) *bulk loading* B+ tree index

(iii) eliminating *duplicates* (*why?*)



(iv) summarizing groups of tuples (*what is that?*)

**GROUP BY!**

(v) *Sort-merge* join [more about that later]

# Sorting Challenges

(easy) problem:

how to sort 1GB data with 1GB memory?



(hard) problem:

how to sort 1GB data with **1MB** memory?



why not virtual memory (i.e., swapping on disk)?



## Goal

minimize disk accesses when working under memory constraints

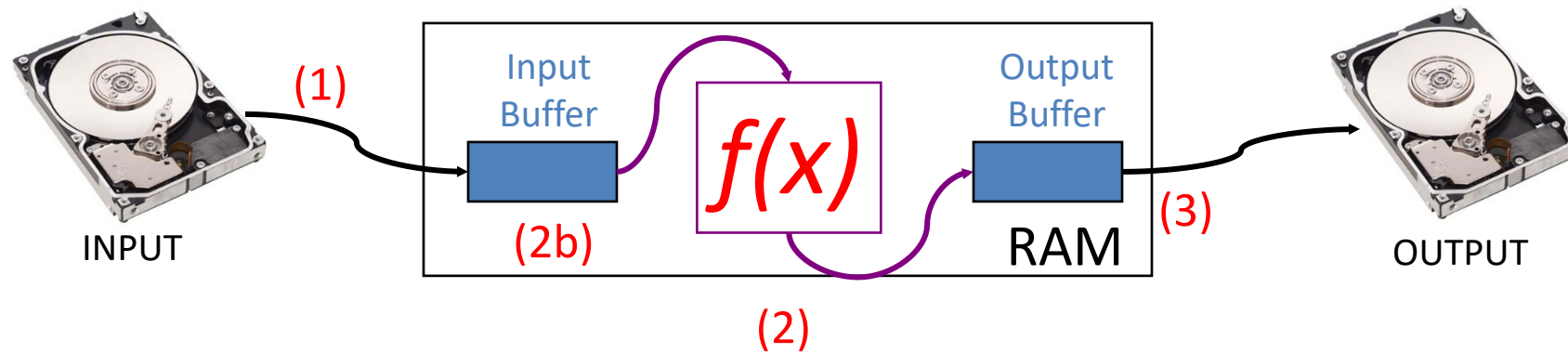
## Idea

stream data, calculate *something useful*, and write back on disk

# Streaming Data Through RAM

An important method for sorting & other DB operations

**Compute  $f(x)$  for each record, write out the result**



(1) Read a page (from INPUT to Input Buffer)

(3) When Output Buffer fills, write it to OUTPUT

(2) Calculate  $f(x)$  for each item (e.g., sort, (de-)compress, discard rows [selection], discard columns [projection])

(2b) When Input Buffer is consumed, read another page

What about  $f()$  being `sort()`?

**Note that** reads and writes are not (always) coordinated!

- For  $f()$  being `compress()`, `select()`, `project()` we may **read many pages per write**
- For  $f()$  being `decompress()` we may **write many pages per read**



Let's apply this to sorting!



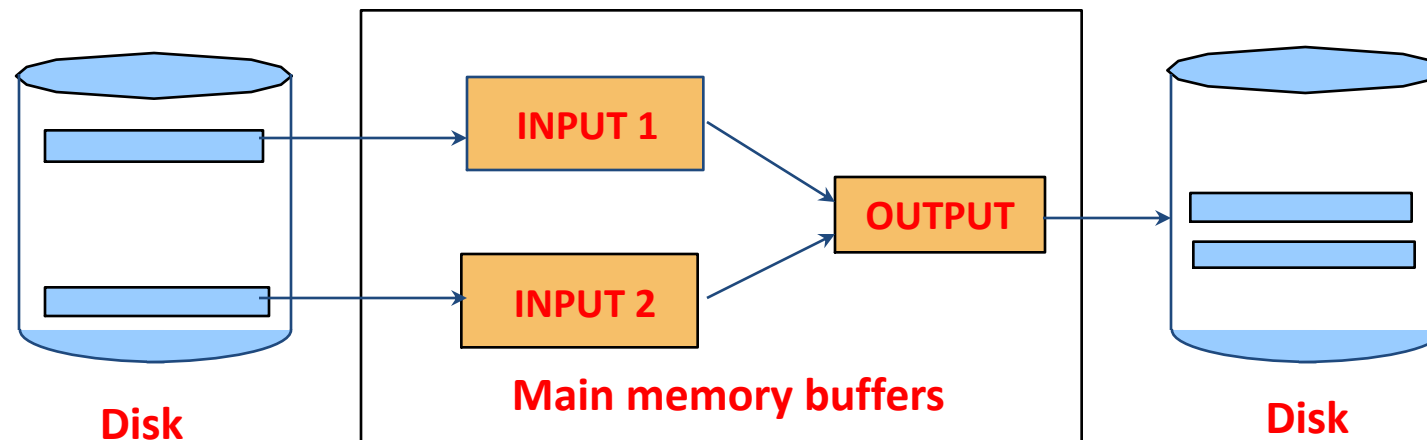
# 2-Way Sort: Requires 3 Buffers

**Pass 0: Read a page, sort it, write it.**

- only one buffer page is used (as in previous slide)

**Pass 1, 2, 3, ..., etc.:**

- requires 3 buffer pages
- merge pairs of runs into runs twice as long
- three buffer pages used.



# Two-Way External Merge Sort

Each pass we read + write each page in file.

$N$  pages in the file =>

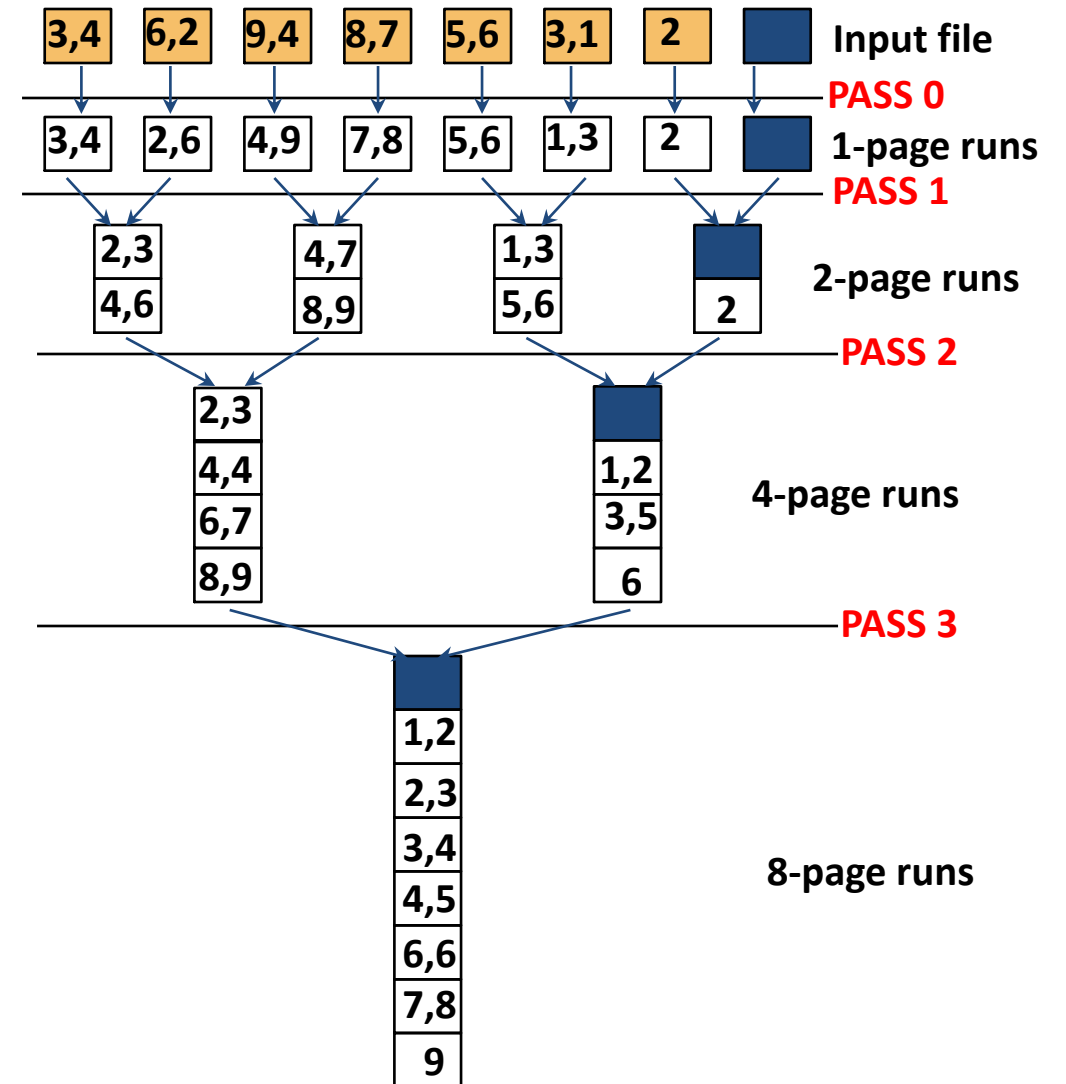
the number of passes ??

So total cost is: ??

*Idea*

Divide and conquer

sort sub-files and merge



# 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 total cost is:  $2N(\lceil \log_2 N \rceil + 1)$

Idea

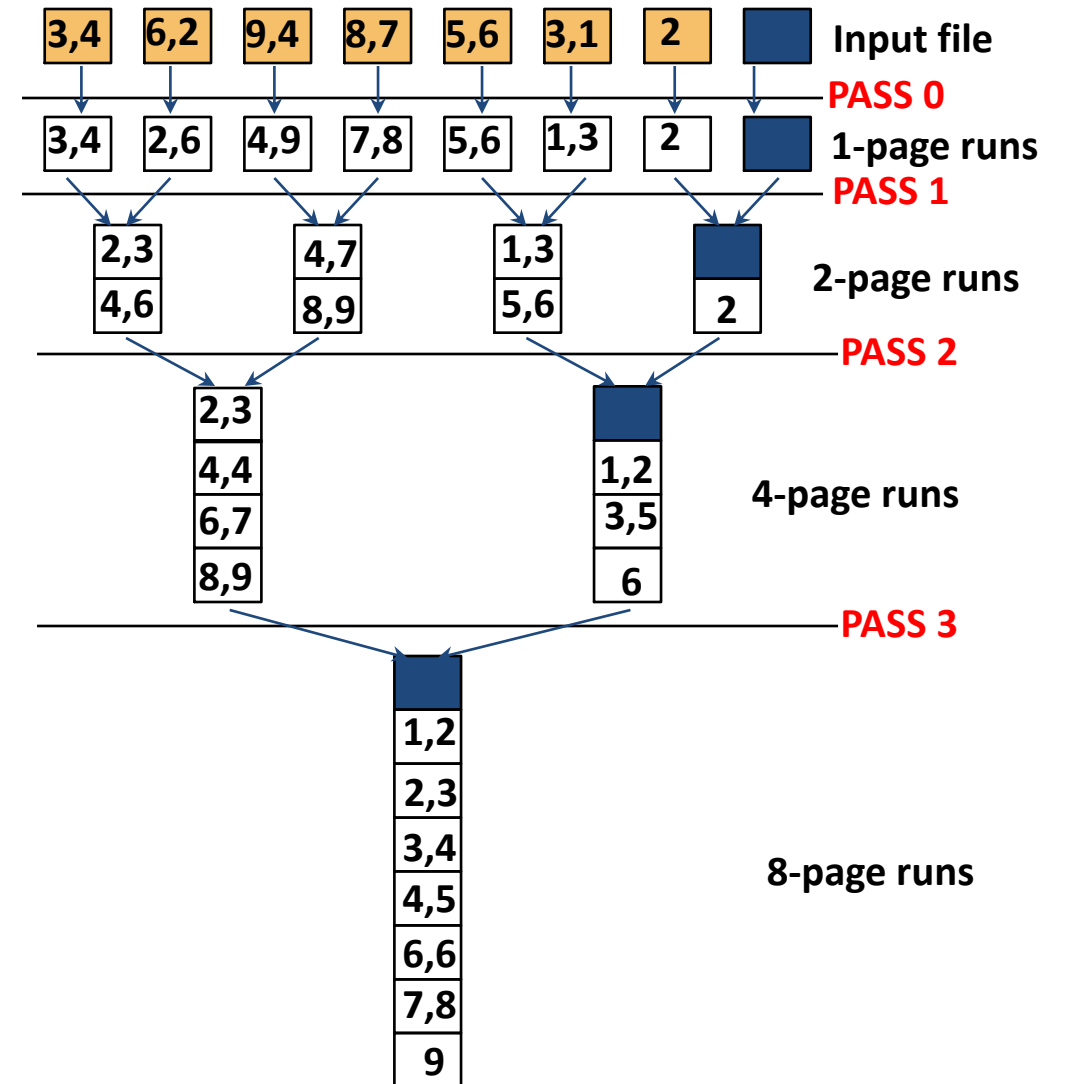
Divide and conquer

sort sub-files and merge

is this good enough?



No! why?



# External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

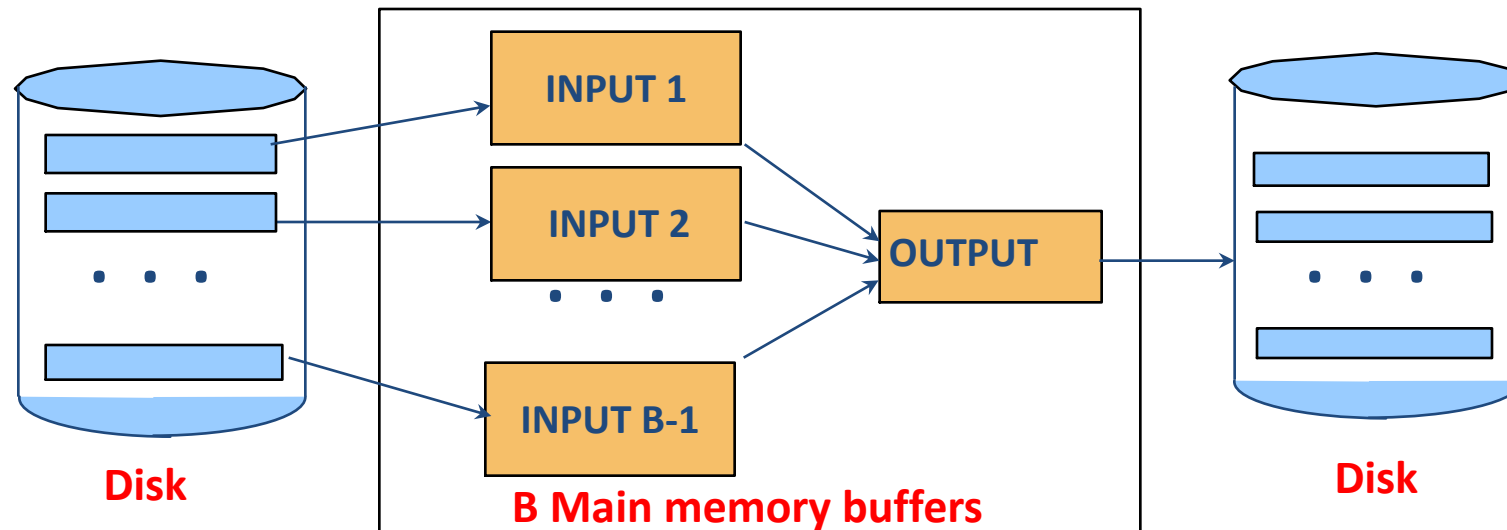
Using B<sup>+</sup>-Trees for sorting

# General External Merge Sort

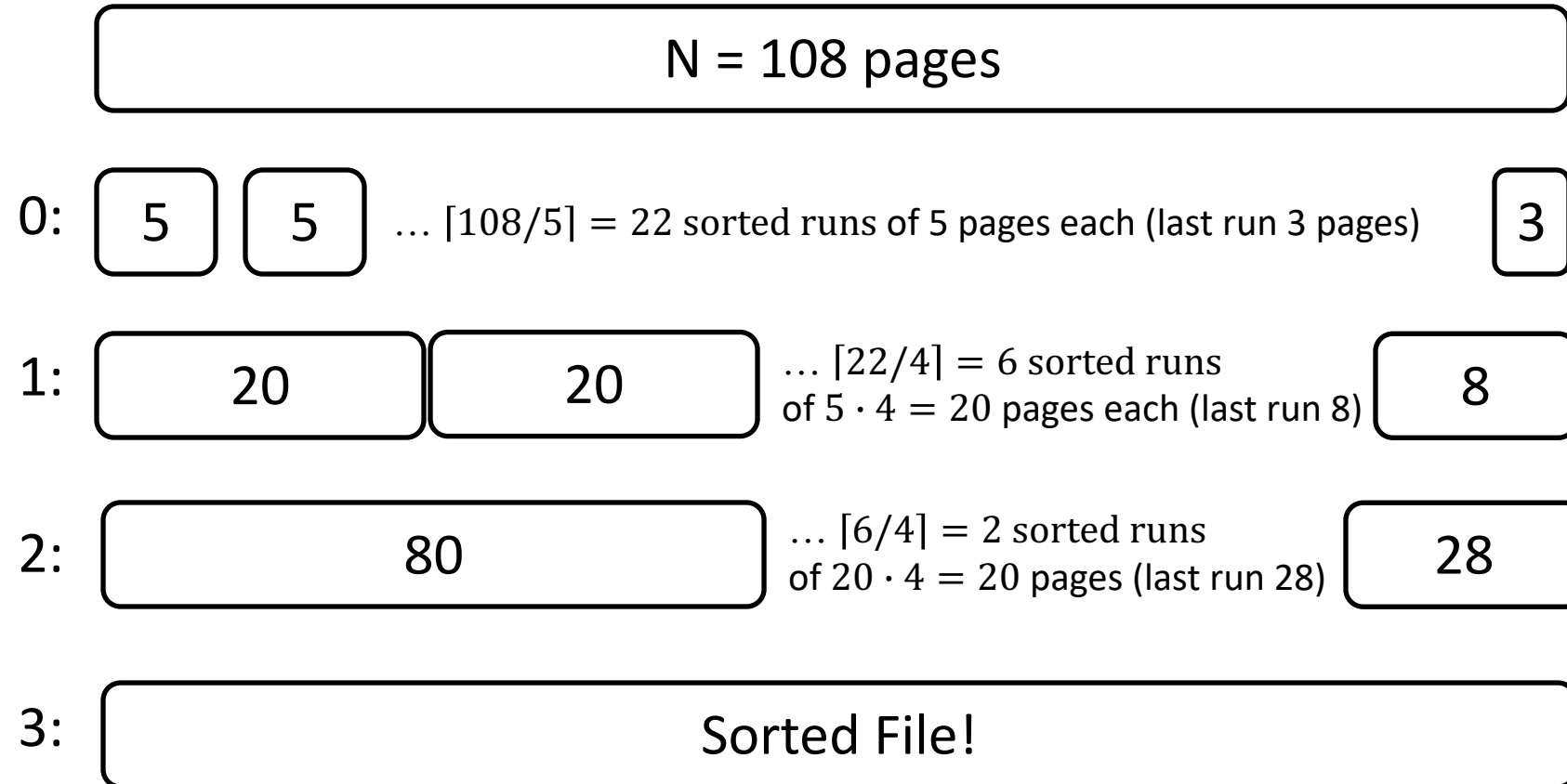
*How can we exploit more than 3 buffer pages?*

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 1, 2, ..., etc.:** merge  $B-1$  runs.



# General External Merge Sort



*B=5 buffer pages*



# Cost of External Merge Sort

Number of passes:  $1 + \lceil \log_{B-1} [N/B] \rceil$

Cost =  $2N \cdot (\# \text{ of passes})$

to sort 108-page file with 5 buffers:

- 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

Formula check:  $1 + \lceil \log_{B-1} [N/B] \rceil = 1 + \lceil \log_4 22 \rceil = 1 + 3$

# Number of Passes of External Sort

I/O cost is  $2N$  times number of passes:  $2 \cdot N \cdot (1 + \lceil \log_{B-1} [N/B] \rceil)$

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



# In-Memory Sort Algorithm

**Quicksort** is fast (very fast)!!

we generate in Pass 0  $N/B$  #runs of  $B$  pages each

can we generate longer runs?

why do we want that?



yes! Idea: maintain a current set as a heap

# In-memory Heapsort

(aka “replacement sort”)

0: read in B-2 blocks

1: find the smallest record greater than the largest value to output buffer

- add it to the end of the output buffer
- fill moved record’s slot with next value from the input buffer, if empty refill input buffer

2: **else:** end run

3: goto (1)

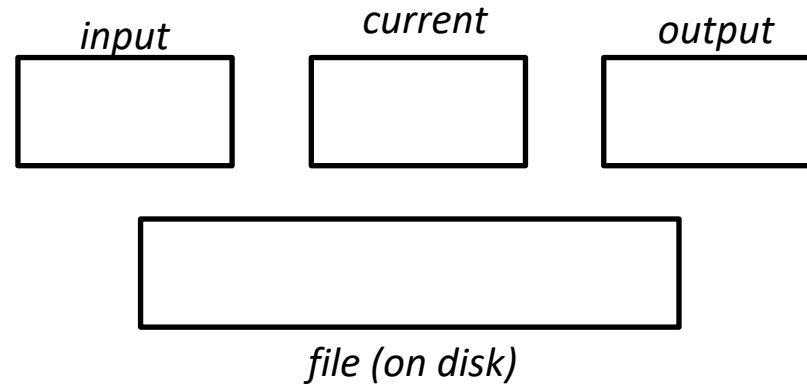
# In-memory Heapsort

$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20 10, 40 22, 17 25, 73 16, 26 21, 13 22, 24  
 10, 17, 20, 22, 30, 40 13, 16, 21, 25, 26, 73 22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page



# In-memory Heapsort

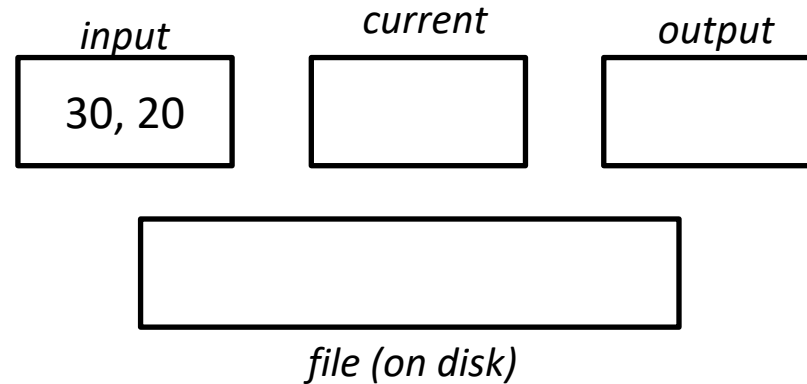
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

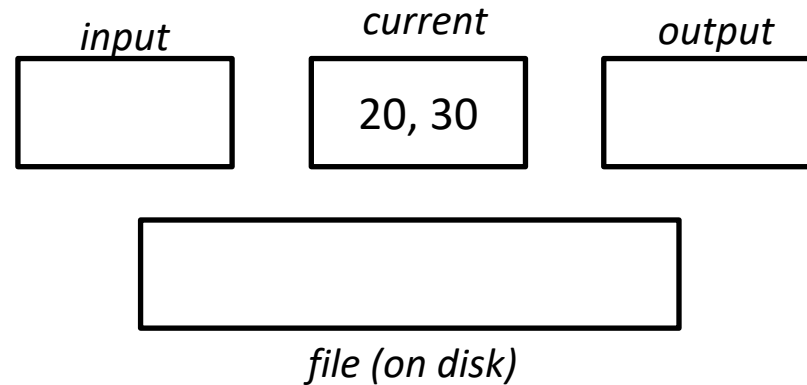
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20   **10, 40**   22, 17   25, 73   16, 26   21, 13   22, 24

10, 17, 20, 22, 30, 40   13, 16, 21, 25, 26, 73   22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page



# In-memory Heapsort

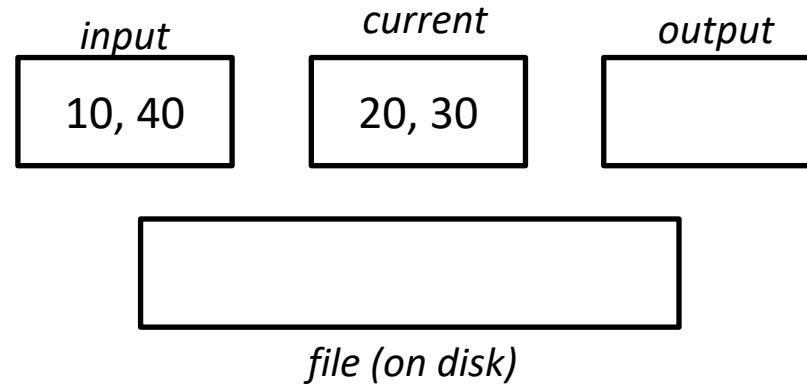
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

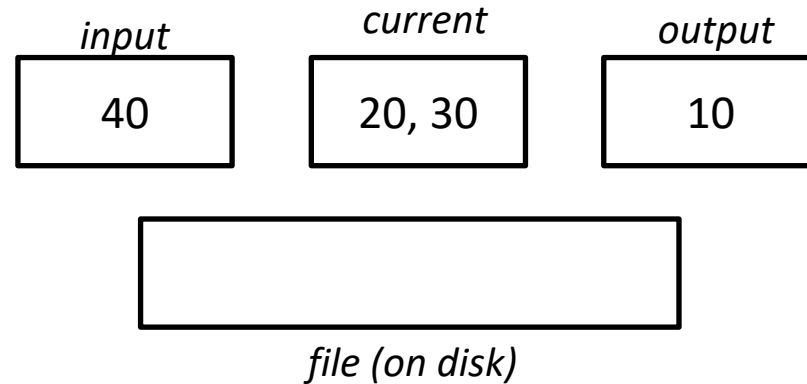
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

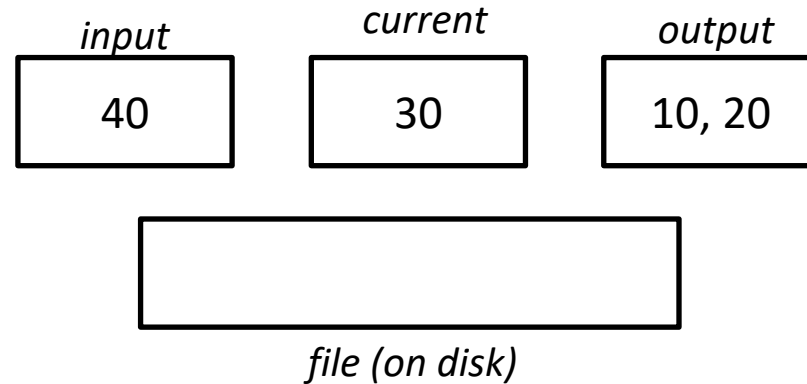
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page





# In-memory Heapsort

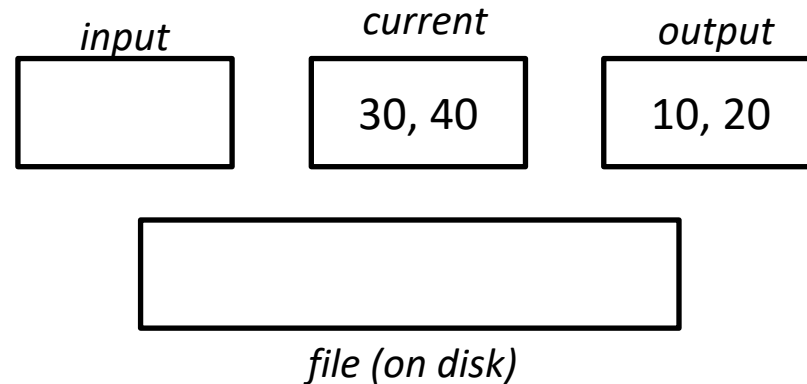
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    **22, 17**    25, 73    16, 26    21, 13    22, 24

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page



# In-memory Heapsort

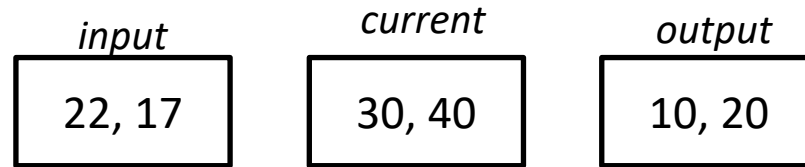
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



*update the heap*



*file (on disk)*

# In-memory Heapsort

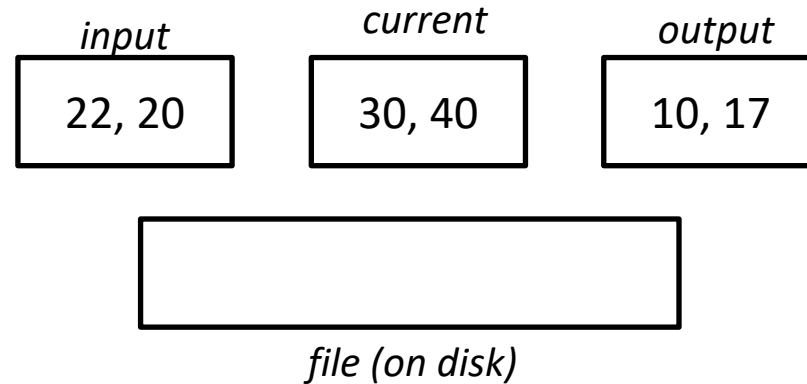
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

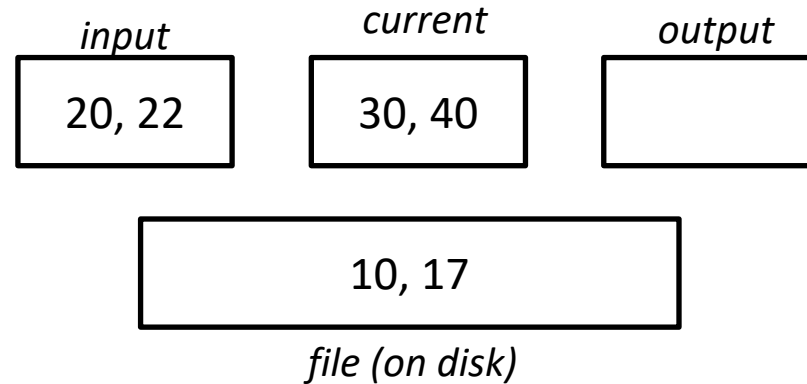
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    **25, 73**    16, 26    21, 13    22, 24

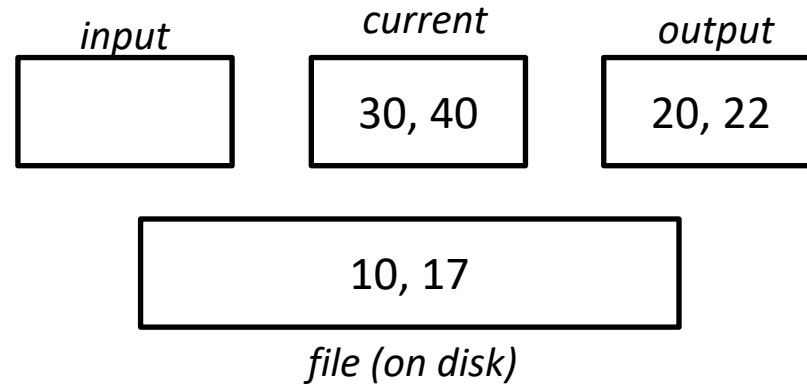
10, 17, 20, 22, 30, 40

13, 16, 21, 25, 26, 73

22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page



# In-memory Heapsort

$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

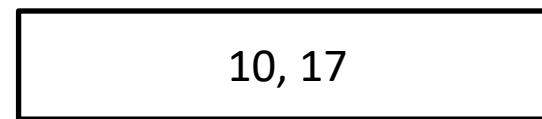
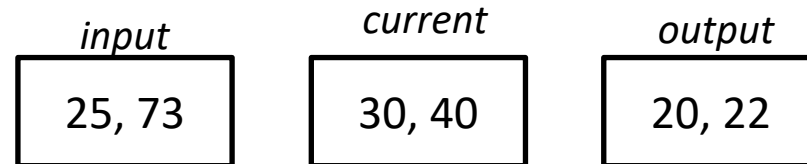
Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40

13, 16, 21, 25, 26, 73

22, 24

Heapsort  
 $3-2=1$  page



*file (on disk)*

*here we end up writing both values,  
one at a time (no change by resorting)*

# In-memory Heapsort

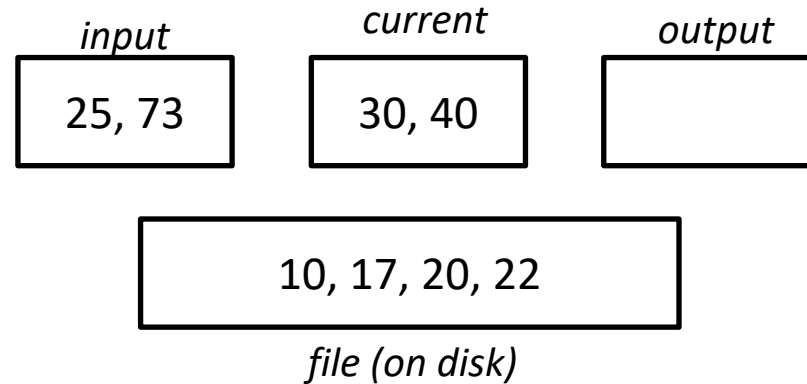
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

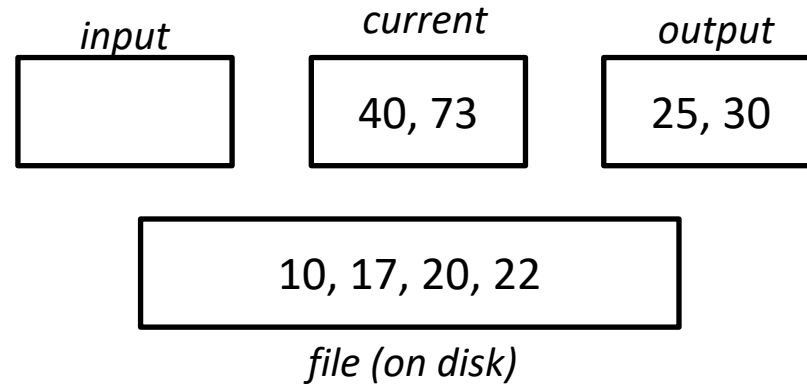
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    **16, 26**    21, 13    22, 24

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page





# In-memory Heapsort

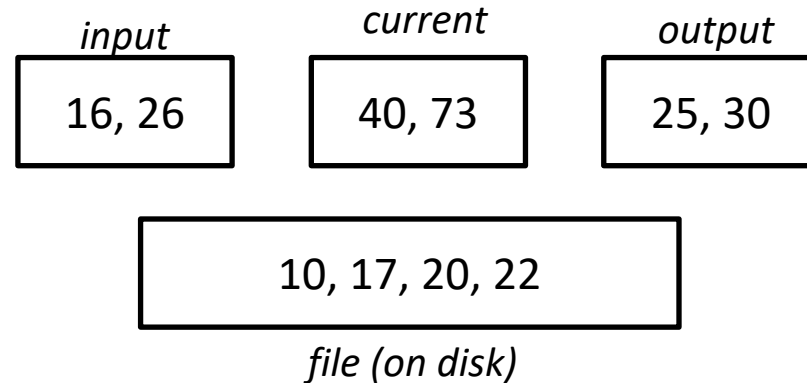
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

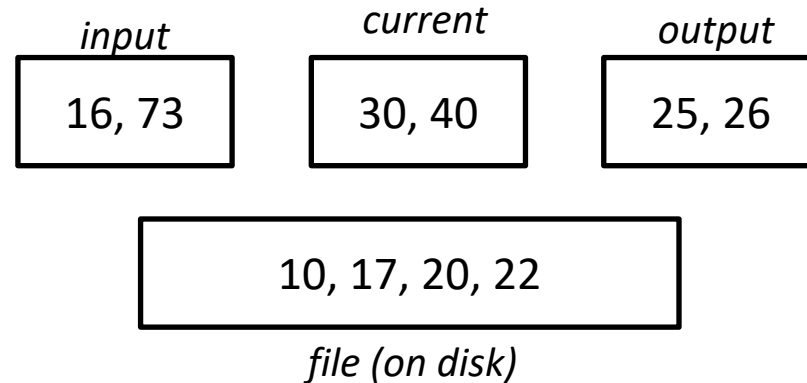
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

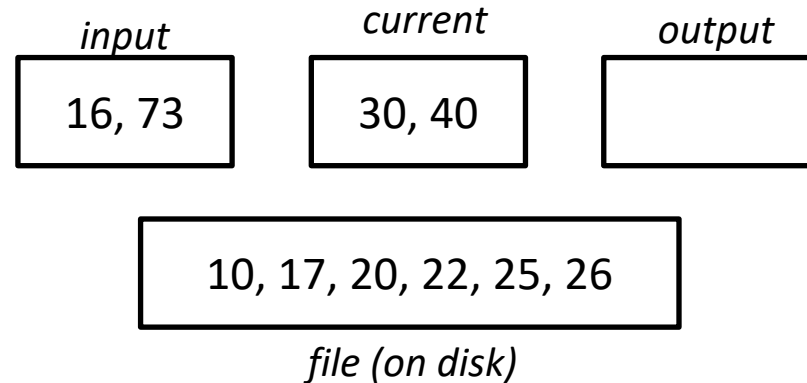
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    **21, 13**    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                      13, 16, 21, 25, 26, 73                      22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

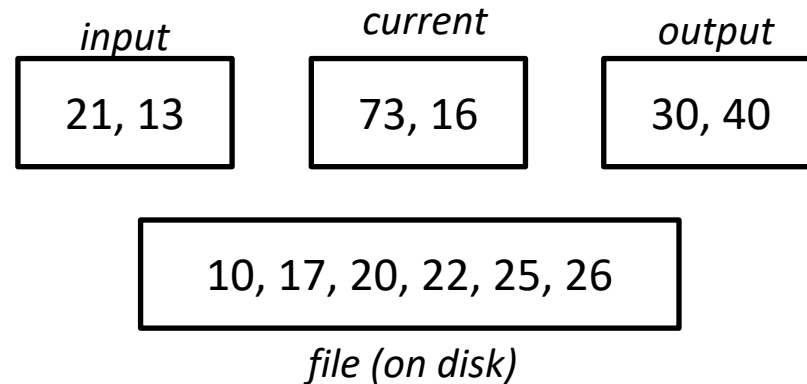
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page



# In-memory Heapsort

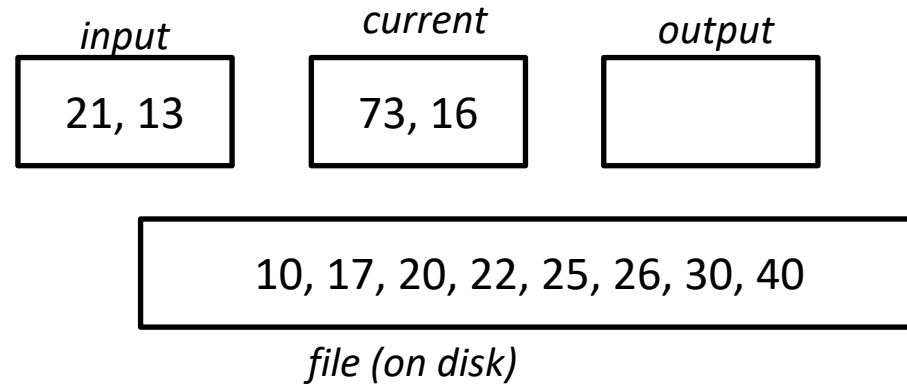
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

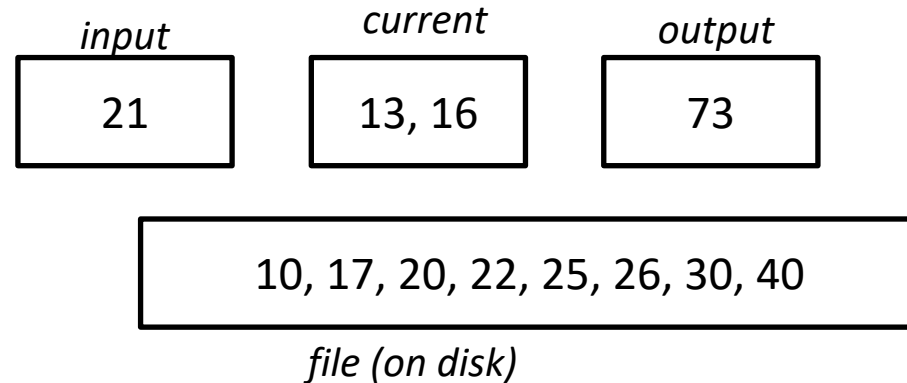
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

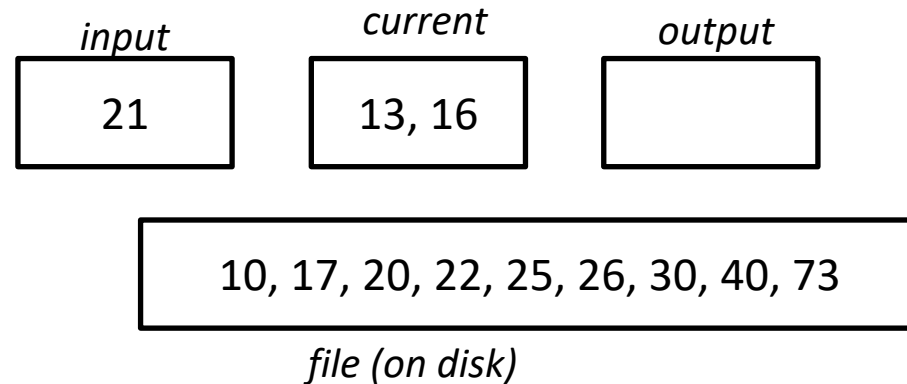
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    **22, 24**

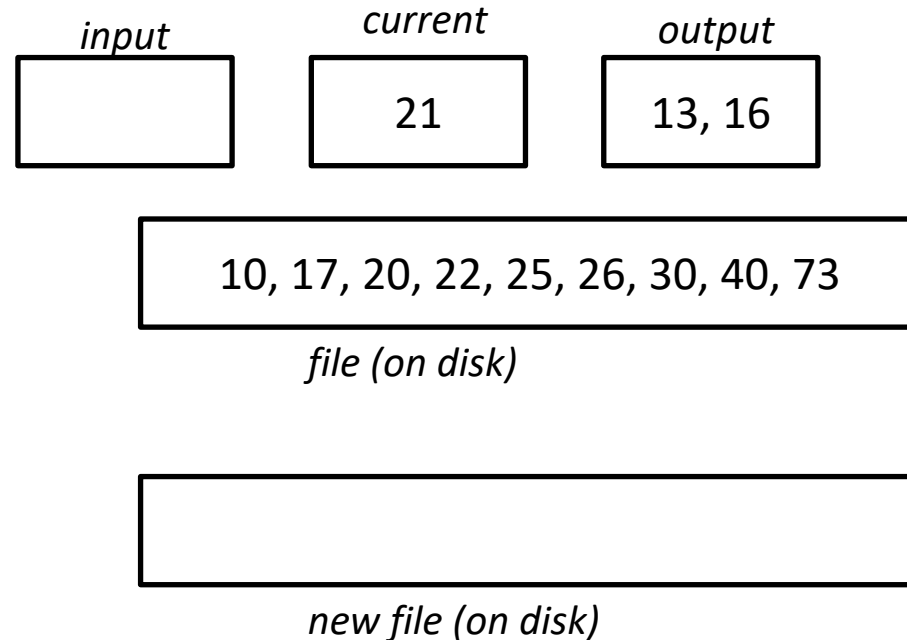
10, 17, 20, 22, 30, 40

13, 16, 21, 25, 26, 73

22, 24

Normally we use  
3-pages runs in  
Pass 0

Heapsort  
3-2=1 page





# In-memory Heapsort

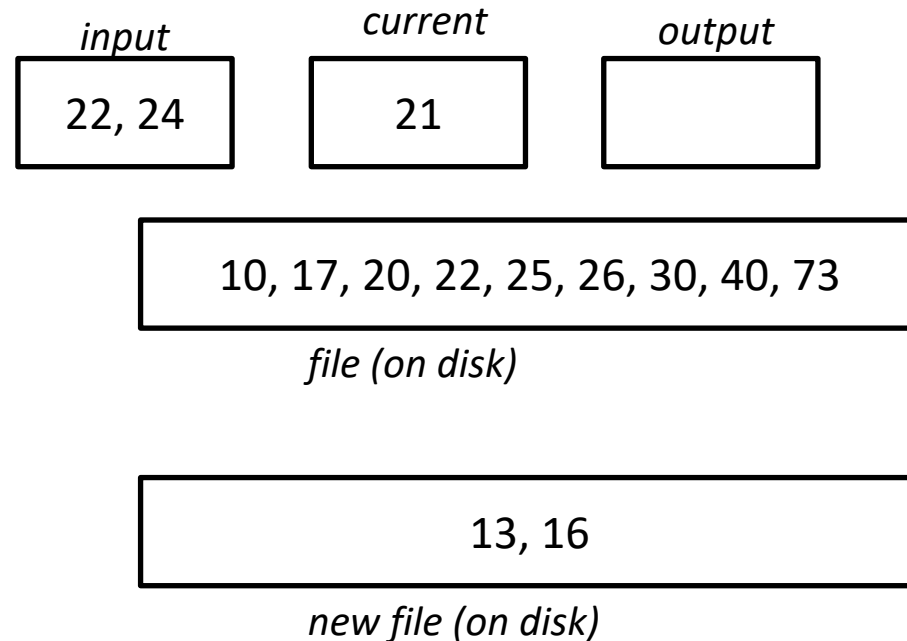
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



# In-memory Heapsort

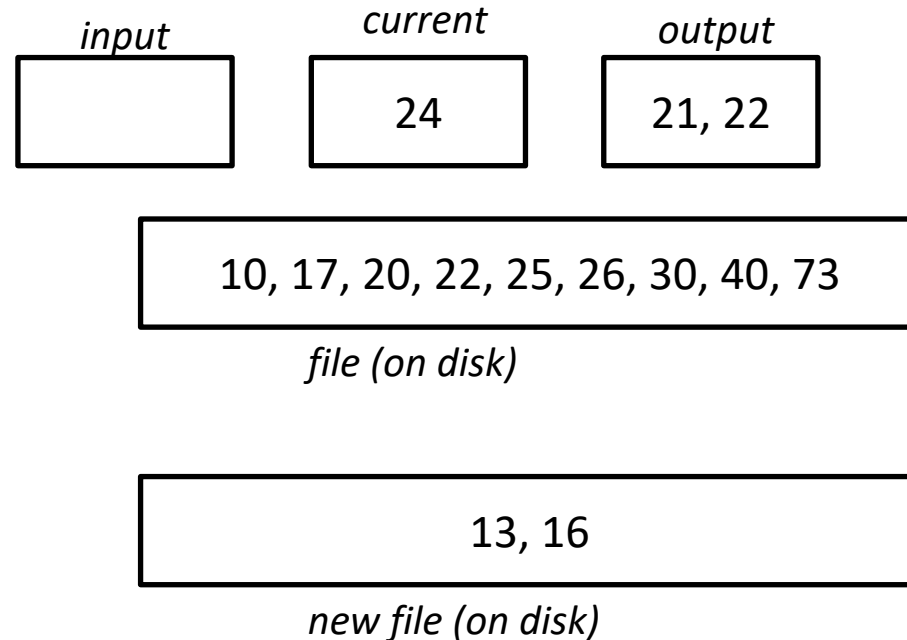
$N = 7$  pages (file),  $B = 3$  pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page







# In-memory Heapsort

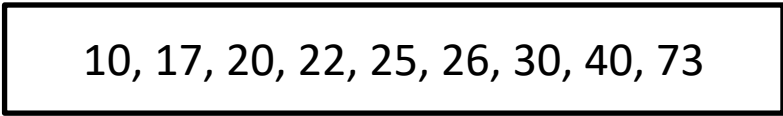
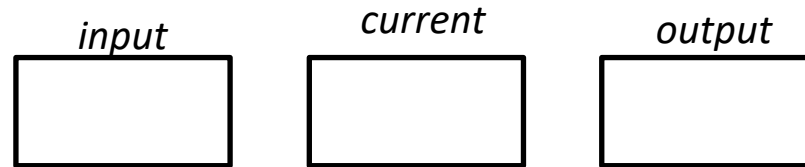
N = 7 pages (file), B = 3 pages (buffers)

30, 20    10, 40    22, 17    25, 73    16, 26    21, 13    22, 24

Normally we use  
3-pages runs in  
Pass 0

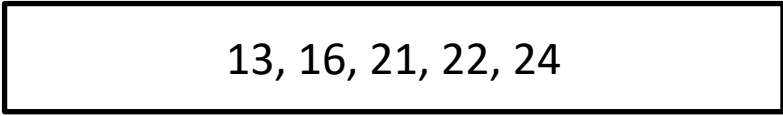
10, 17, 20, 22, 30, 40                    13, 16, 21, 25, 26, 73                    22, 24

Heapsort  
3-2=1 page



*file (on disk)*

*only 2 (longer) sorted runs!*



*new file (on disk)*

# More on Heapsort

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

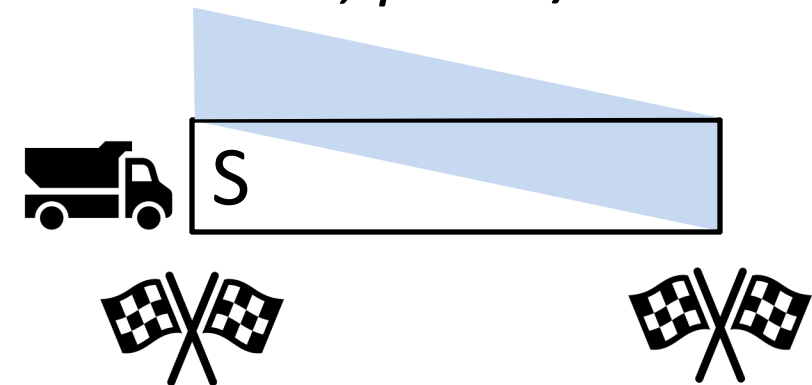
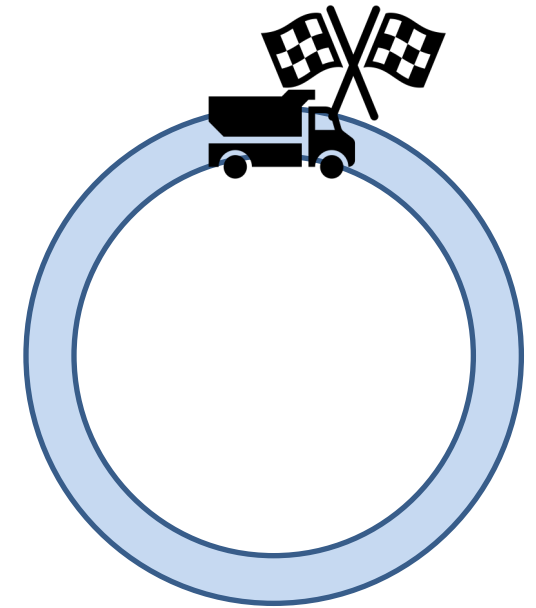
## The snowplow analogy

(1) Imagine a snowplow moving around a circular track with a steady rate of snow fall.

(2) At any instant, there is a certain amount of snow  $S$  on the track. Some falling snow comes in front of the plow, some behind.

(3) During the next revolution of the plow, all of this is removed, plus  $1/2$  of what falls during that revolution.

(4) Thus, the plow removes  $2S$  amount of snow.



# More on Heapsort

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

## Worst-Case:

- What is min length of a run?
- How does this arise?

$B-2$

when the file is reversely sorted

## Best-Case:

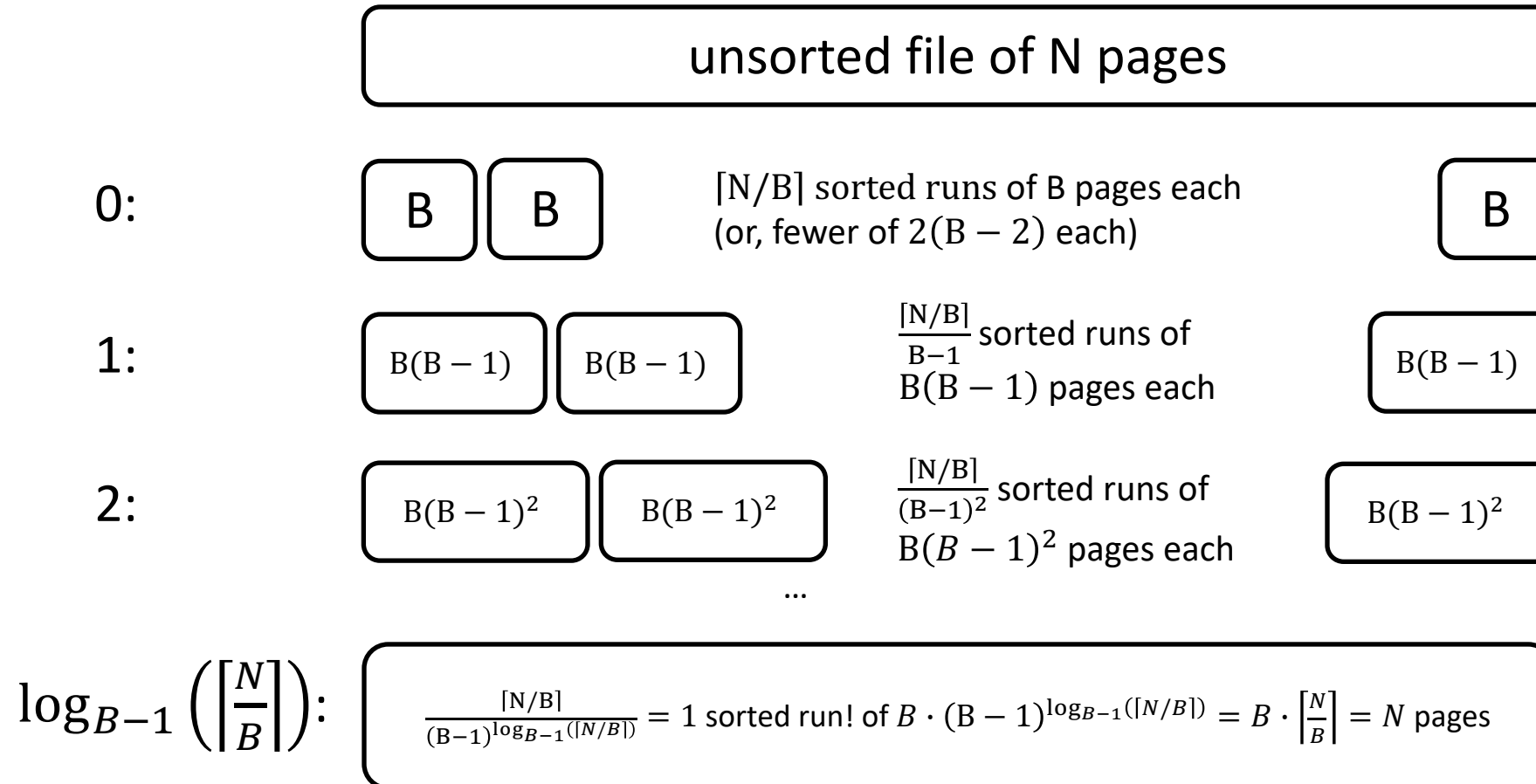
- What is max length of a run?
- How does this arise?

the entire file

when the file is sorted

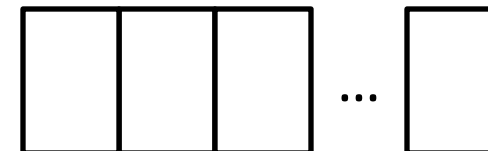
Quicksort is faster, but ... longer runs often means fewer passes!

# External Merge Sort Summary



total #I/O:  $2 \cdot N \cdot (1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil)$

$B$  buffer pages:





# I/O for External Merge Sort

Do I/O a page at a time

- Not one I/O per record

In fact, read a *block* (*chunk*) of pages sequentially!

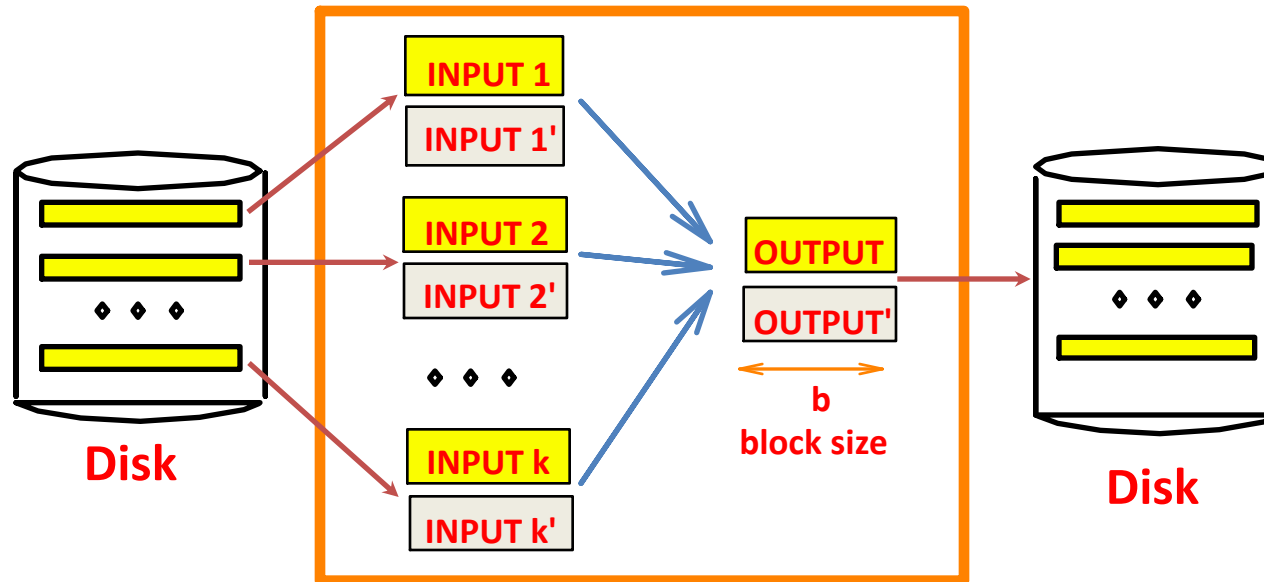
Suggests we should make each buffer (input/output) be a *block* of pages.

- But this will reduce fan-in during merge passes!
- In practice, most files still sorted in **2-3 passes**.

# Double Buffering

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*.



**B main memory buffers, k-way merge**

# Sorting Records!

Sorting has become a blood sport!

– **Parallel sorting** is the name of the game ...

**Minute Sort**: how many 100-byte records can you sort in a minute?

**Penny Sort**: how many can you sort for a penny?

*See <http://sortbenchmark.org/>*

# External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using B<sup>+</sup>-Trees for sorting

# 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**
- B+ tree is **not clustered**

# 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**

# 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:

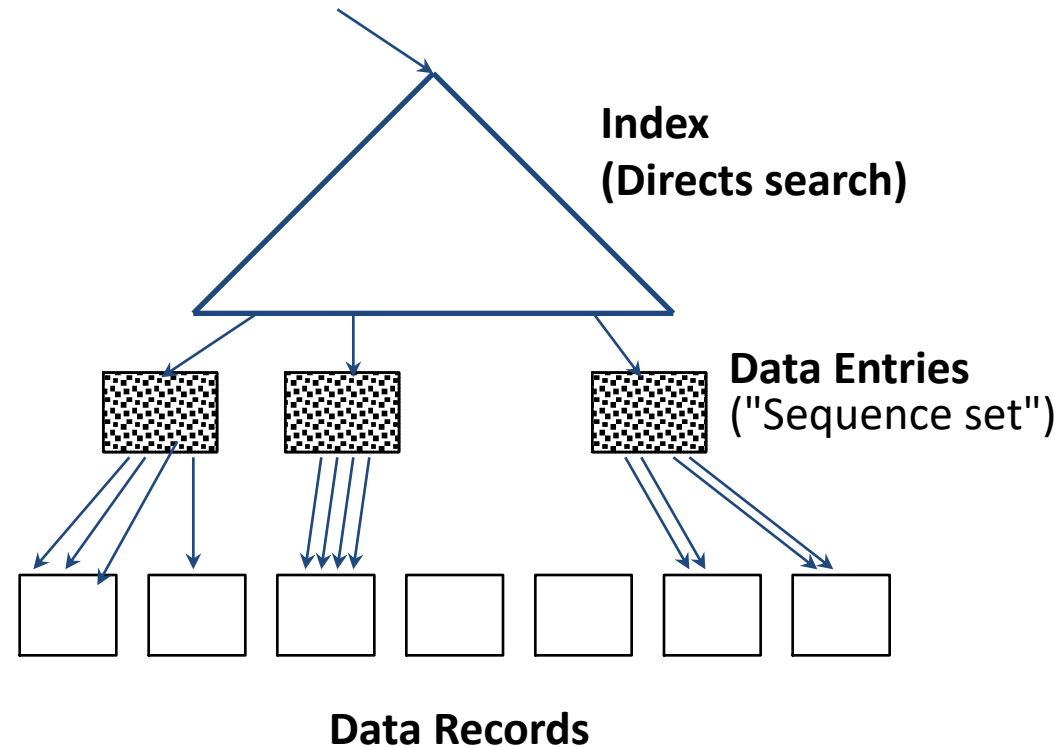
- B+ tree is **clustered**      ***Good idea!***
- B+ tree is **not clustered**      ***Could be a very bad idea!***

# Clustered B+ Tree Used for Sorting

Cost: root to the left-most leaf, then retrieve all leaf pages  
(Alternative 1)

If Alternative 2 is used?

Additional cost of retrieving  
data records: each page  
fetched just once.

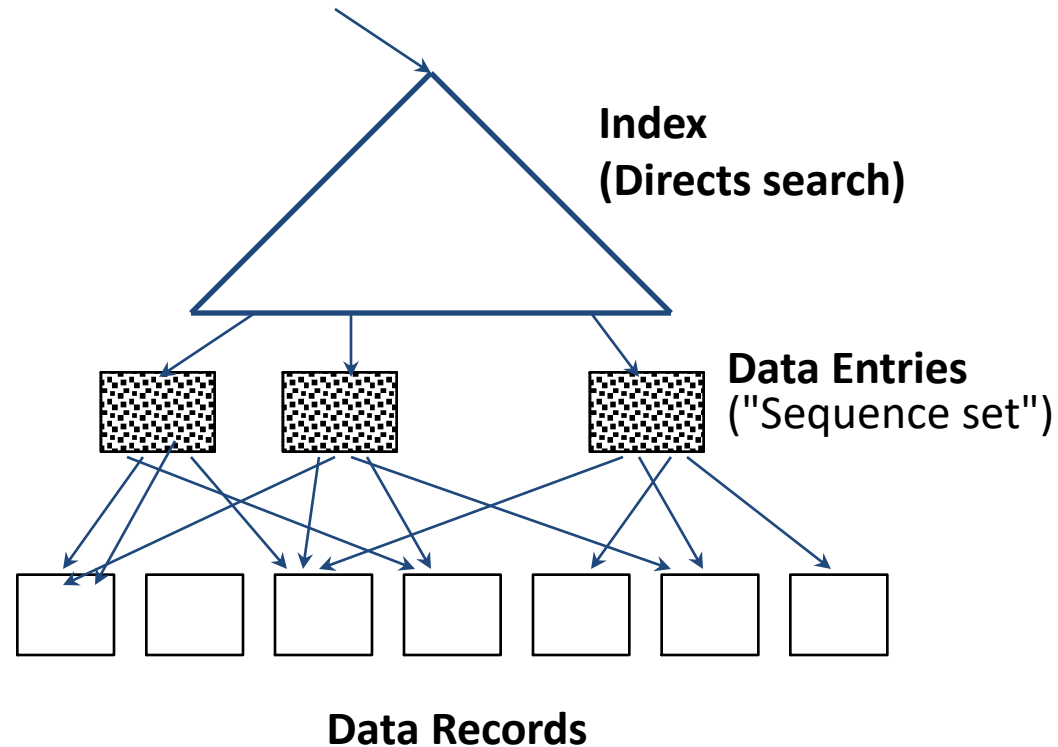


**➡ Always better than external sorting!**



# 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!**



# External Sorting vs. Unclustered Index

if  $B \geq N$  then  
only quick sort!

N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

Special case, that the tree is always behaving like a clustered tree

- $p$ : # of records per page
- $B=1,000$  and block size=32 for sorting
- $p=100$  is the more realistic value.

# Summary

External sorting is used for many different operations in DBs

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 **fewer 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, still improving!

Clustered B<sup>+</sup> tree is good for sorting

Unclustered tree is usually very bad