Comp115: Databases

External Sorting

Instructor: Manos Athanassoulis

External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

Using B^+-Trees for sorting

Why Sort?

a classic problem in computer science!

but also a database specific problem:

(i) data requested in sorted order

e.g., find students in increasing *gpa* order

(ii) *bulk loading* B+ tree index

(iii) eliminating *duplicate* (Why?)

(iv) summarizing groups of tuples

(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 over data, calculate *something useful*, and write back on disk

Streaming Data Through RAM

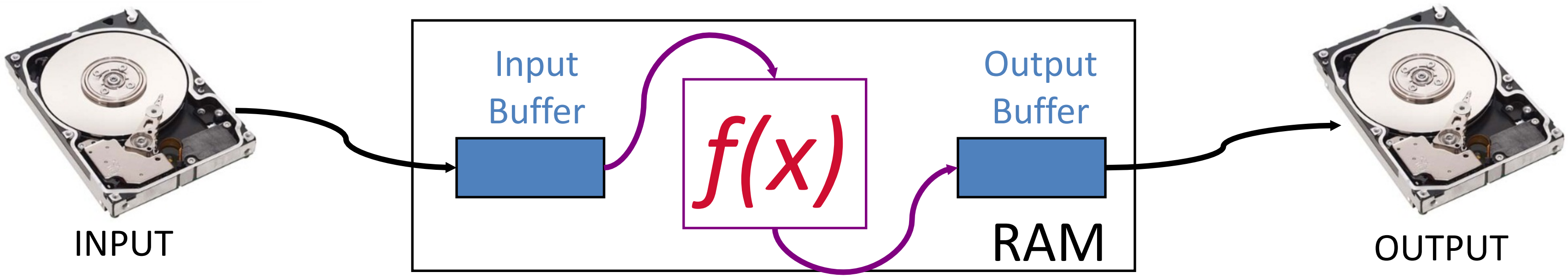
An important method for sorting & other DB operations

Simple case:

- Compute *f(x)* for each record, write out the result
- Read a page from INPUT to Input Buffer
- Write *f(x)* for each item to Output Buffer
- When Input Buffer is consumed, read another page
- When Output Buffer fills, write it to OUTPUT

Reads and Writes are *not* coordinated

- E.g., if f() is Compress(), you read many pages per write.
- E.g., if f() is DeCompress(), you write many pages per read.



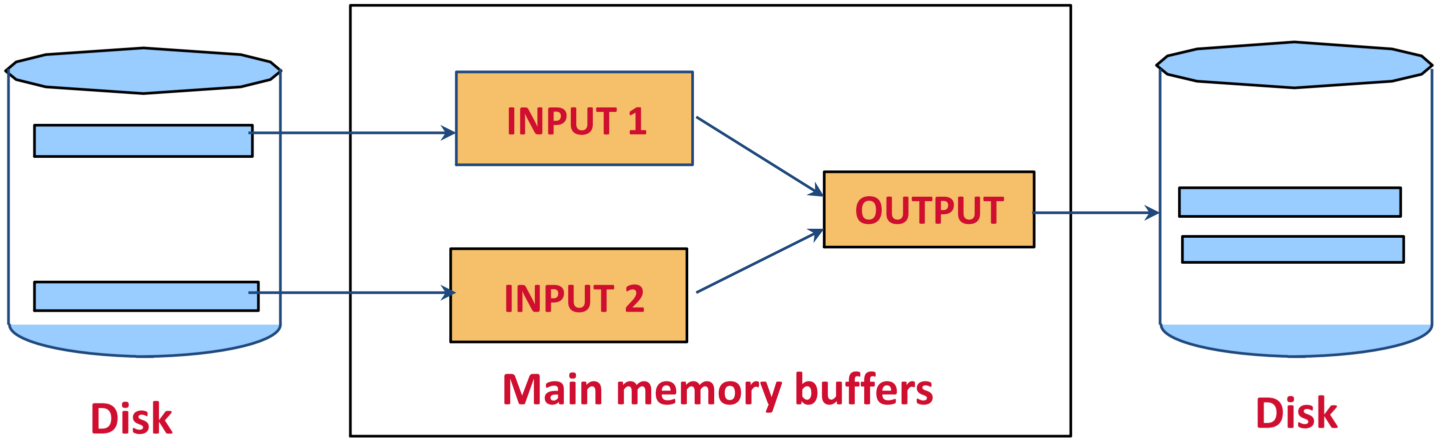
2-Way Sort: Requires 3 Buffers

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.

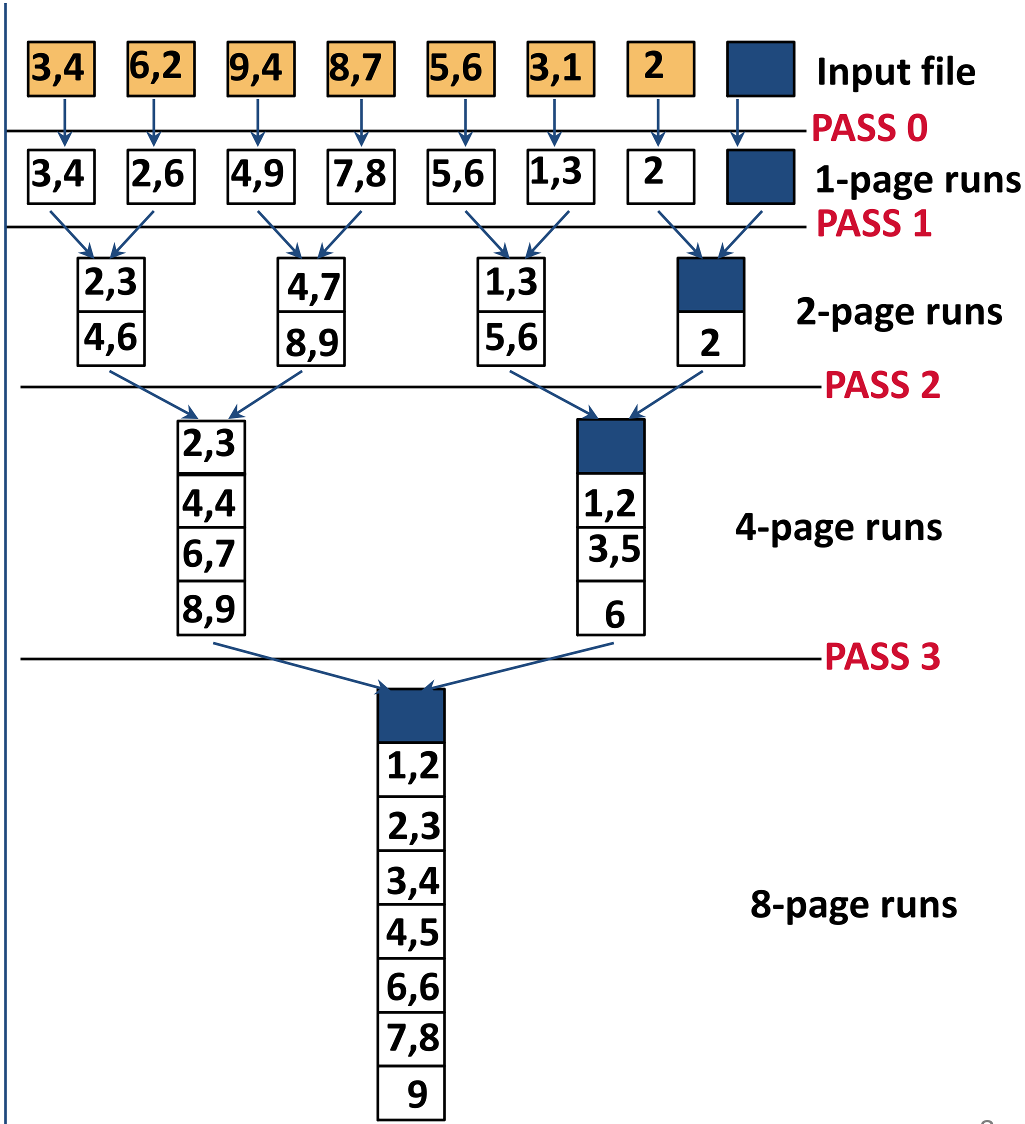
N pages in the file => the number of passes

$$= \lceil \log_2 N \rceil + 1$$

So total cost is:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

<u>Idea:</u> Divide and conquer: sort subfiles and merge



External Sorting

Intro & 2-way external sorting

General external sorting & performance analysis

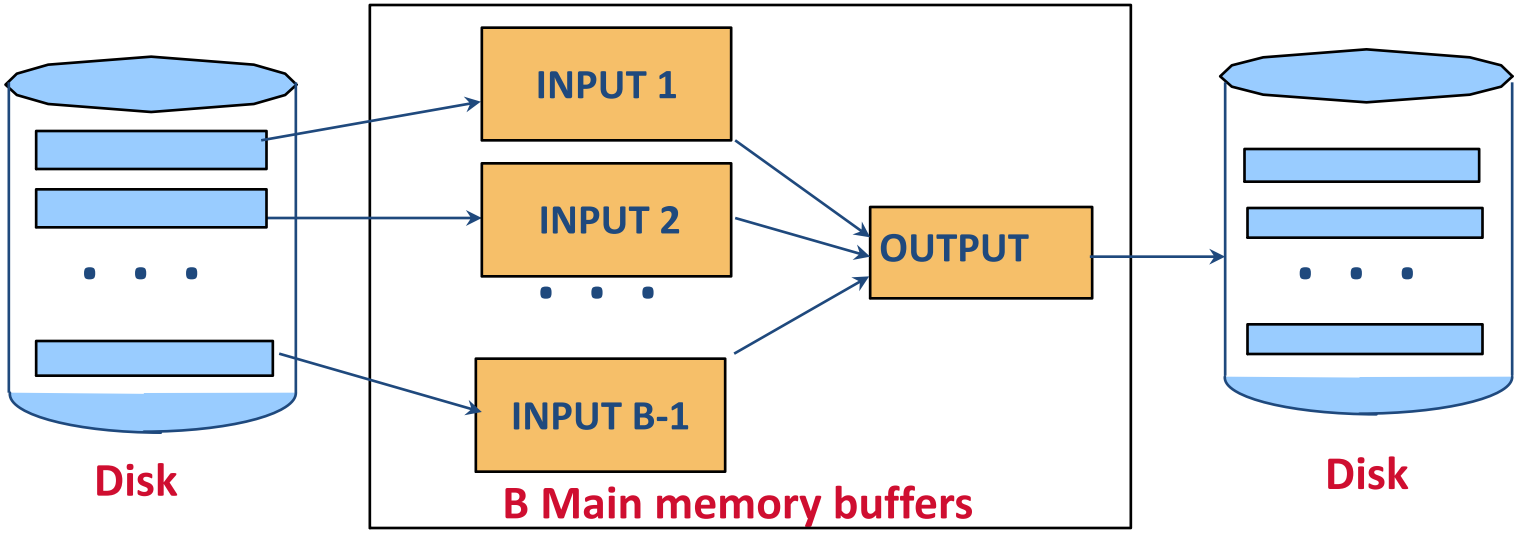
Using B^+-Trees for sorting

General External Merge Sort

☛ ***More than 3 buffer pages. How can we utilize them?***

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

N = 108 pages

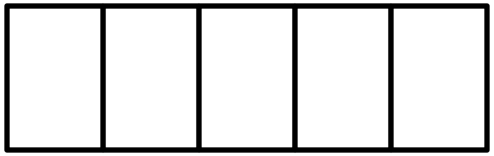
0: 5 | 5 | … $\lceil 108/5 \rceil = 22$ sorted runs of 5 pages each (last run 3 pages) | 3

1: 20 | 20 | … $\lceil 22/4 \rceil = 6$ sorted runs of $5 \cdot 4 = 20$ pages each (last run 8) | 8

2: 80 | … $\lceil 6/4 \rceil = 2$ sorted runs of $20 \cdot 4 = 20$ pages (last run 28) | 28

3: Sorted File!

B=5 buffer pages



Cost of External Merge Sort

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

Cost = 2N * (# 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: $1+\lceil log_{B-1}\lceil N/B \rceil \rceil = 1+\lceil log_4 22 \rceil$ = 1 + 3

Number of Passes of External Sort

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

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

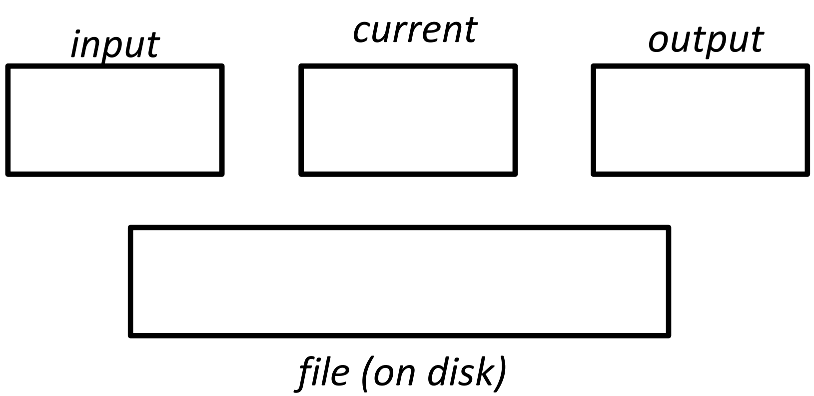
N = 7 pages (file), B = 3 pages (buffers)

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

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

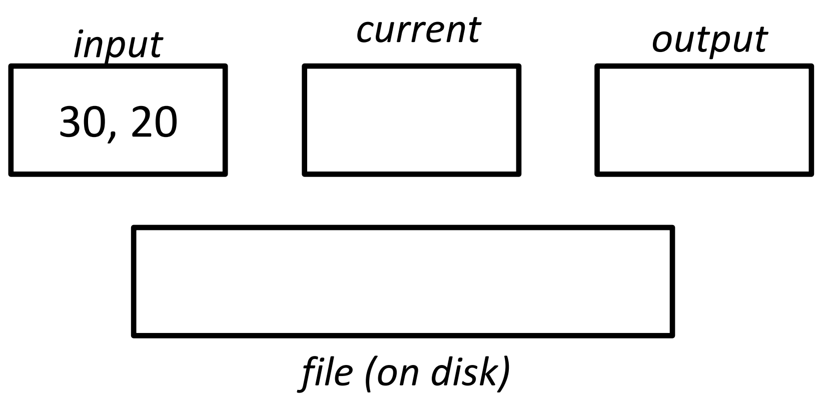
N = 7 pages (file), B = 3 pages (buffers)

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

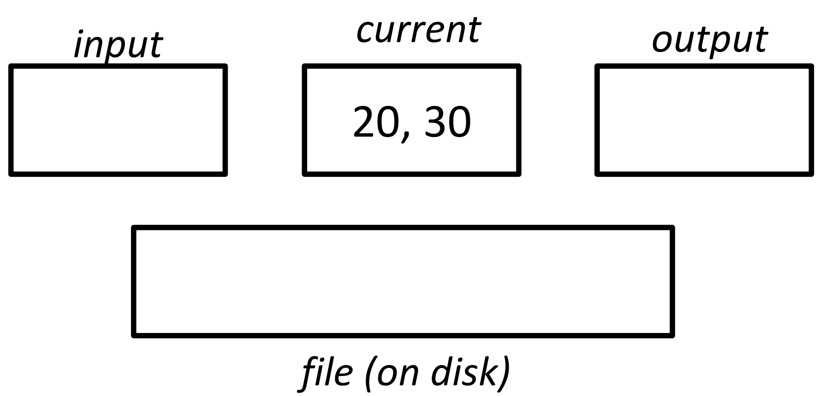
N = 7 pages (file), B = 3 pages (buffers)

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

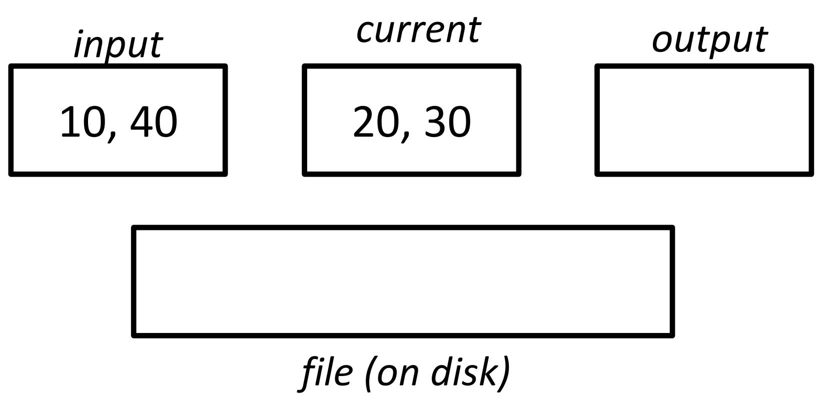
N = 7 pages (file), B = 3 pages (buffers)

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

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

input: 40

current: 20, 30

output: 10

file (on disk)

In-memory Heapsort

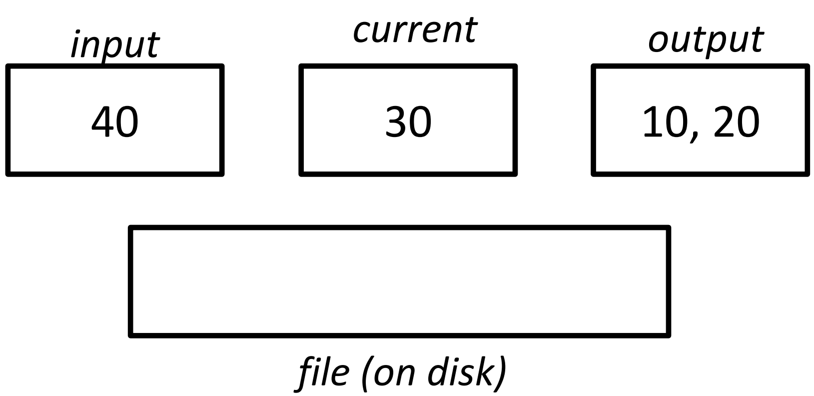
N = 7 pages (file), B = 3 pages (buffers)

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

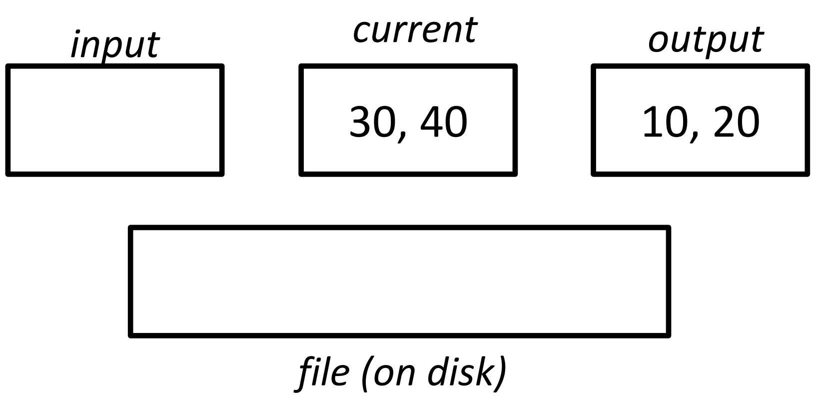
N = 7 pages (file), B = 3 pages (buffers)

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

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

input: 22, 17　*current*: 30, 40　*output*: 10, 20

file (on disk)

In-memory Heapsort

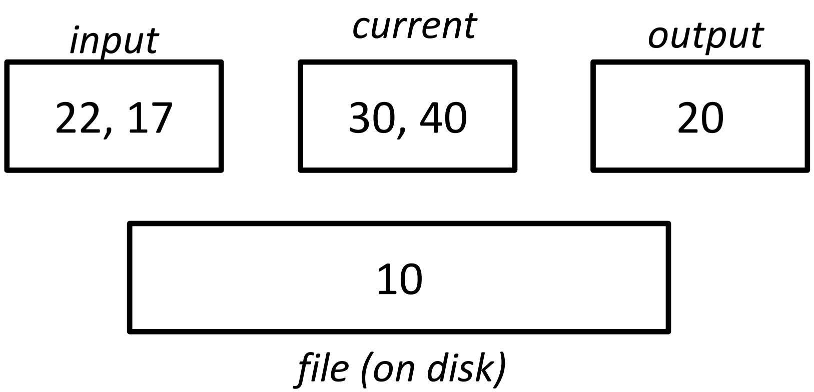
N = 7 pages (file), B = 3 pages (buffers)

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

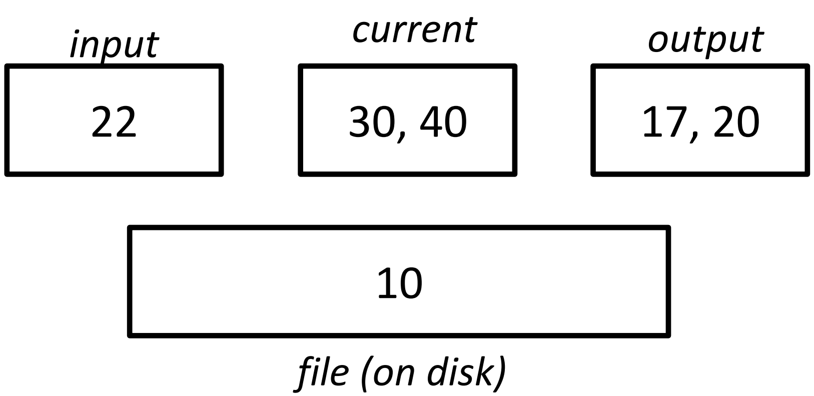
N = 7 pages (file), B = 3 pages (buffers)

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

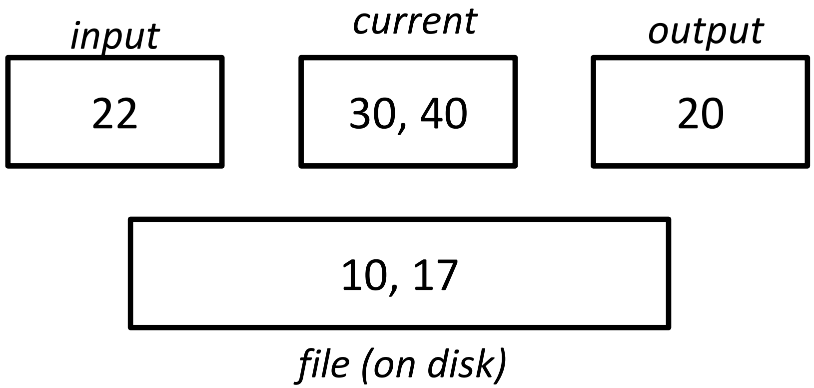
N = 7 pages (file), B = 3 pages (buffers)

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

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

input: (empty)

current: 30, 40

output: 20, 22

10, 17

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

input 25, 73

current 30, 40

output 20, 22

10, 17

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

input	*current*	*output*
25, 73	30, 40	

10, 17, 20, 22

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

input: (empty)

current: 40, 73

output: 25, 30

10, 17, 20, 22

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

input: 16, 26

current: 40, 73

output: 25, 30

10, 17, 20, 22

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

input	*current*	*output*
16, 73	30, 40	25, 26

10, 17, 20, 22

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

input: 16, 73

current: 30, 40

output: (empty)

10, 17, 20, 22, 25, 26

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

input: 21, 13

current: 73, 16

output: 30, 40

10, 17, 20, 22, 25, 26

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

input	*current*	*output*
21, 13	73, 16	

10, 17, 20, 22, 25, 26, 30, 40

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

input	*current*	*output*
21	13, 16	73

10, 17, 20, 22, 25, 26, 30, 40

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

input	*current*	*output*
21	13, 16	

10, 17, 20, 22, 25, 26, 30, 40, 73

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

input

current
21

output
13, 16

10, 17, 20, 22, 25, 26, 30, 40, 73

file (on disk)

new 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

input	*current*	*output*
22, 24	21	

10, 17, 20, 22, 25, 26, 30, 40, 73

file (on disk)

13, 16

new 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

input

current

24

output

21, 22

10, 17, 20, 22, 25, 26, 30, 40, 73

file (on disk)

13, 16

new 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

input [ ] *current* [24] *output* [ ]

[10, 17, 20, 22, 25, 26, 30, 40, 73]

file (on disk)

[13, 16, 21, 22]

new 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

input | *current* | *output*: 24

10, 17, 20, 22, 25, 26, 30, 40, 73

file (on disk)

13, 16, 21, 22

new 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

input *current* *output*

10, 17, 20, 22, 25, 26, 30, 40, 73

file (on disk)

only 2 (longer) sorted runs!

13, 16, 21, 22, 24

new file (on disk)

More on Heapsort

Fact:

average length of a run in heapsort is *2(B-2)*

Worst-Case:

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

Best-Case:

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

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

External Merge Sort Summary

unsorted file of N pages

0: B | B | ... | B — $\lceil N/B \rceil$ sorted runs of B pages each (or, fewer of $2(B-2)$ each)

1: $B(B-1)$ | $B(B-1)$ | ... | $B(B-1)$ — $\frac{\lceil N/B \rceil}{B-1}$ sorted runs of $B(B-1)$ pages each

2: $B(B-1)^2$ | $B(B-1)^2$ | ... | $B(B-1)^2$ — $\frac{\lceil N/B \rceil}{(B-1)^2}$ sorted runs of $B(B-1)^2$ pages each

...

$\log_{B-1}\left(\left\lceil \frac{N}{B} \right\rceil\right)$: $\frac{\lceil N/B \rceil}{(B-1)^{\log_{B-1}(\lceil N/B \rceil)}} = 1$ sorted run! of $B \cdot (B-1)^{\log_{B-1}(\lceil N/B \rceil)} = B \cdot \left\lceil \frac{N}{B} \right\rceil = N$ pages

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!

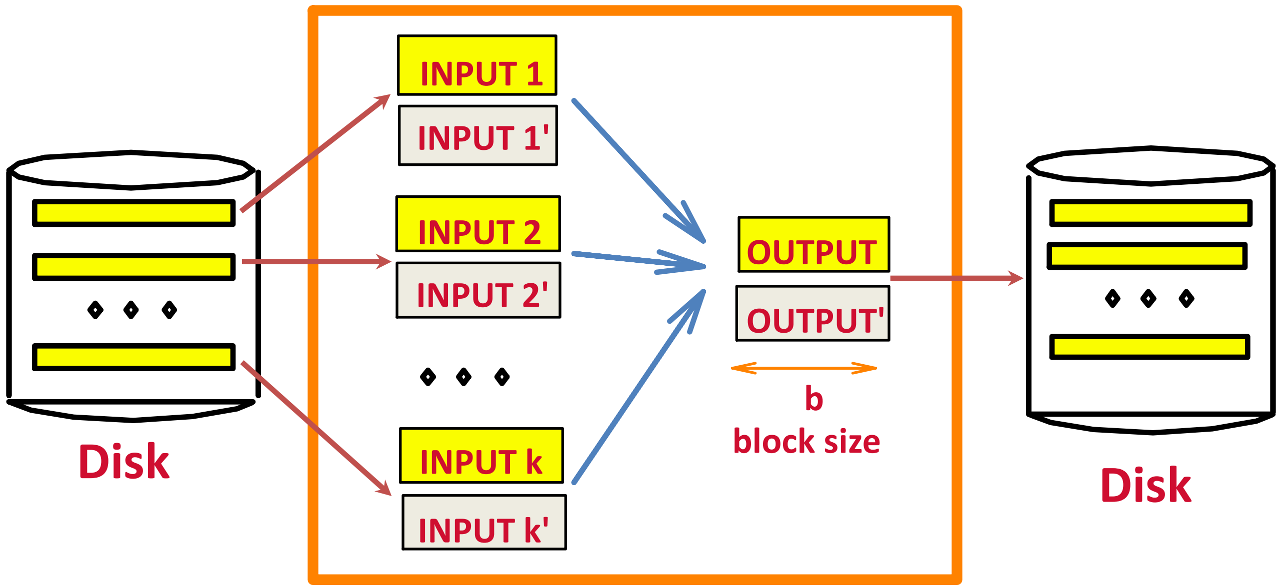
Suggests we should make each buffer (input/output) be a *block* of pages.

- 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^+-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?

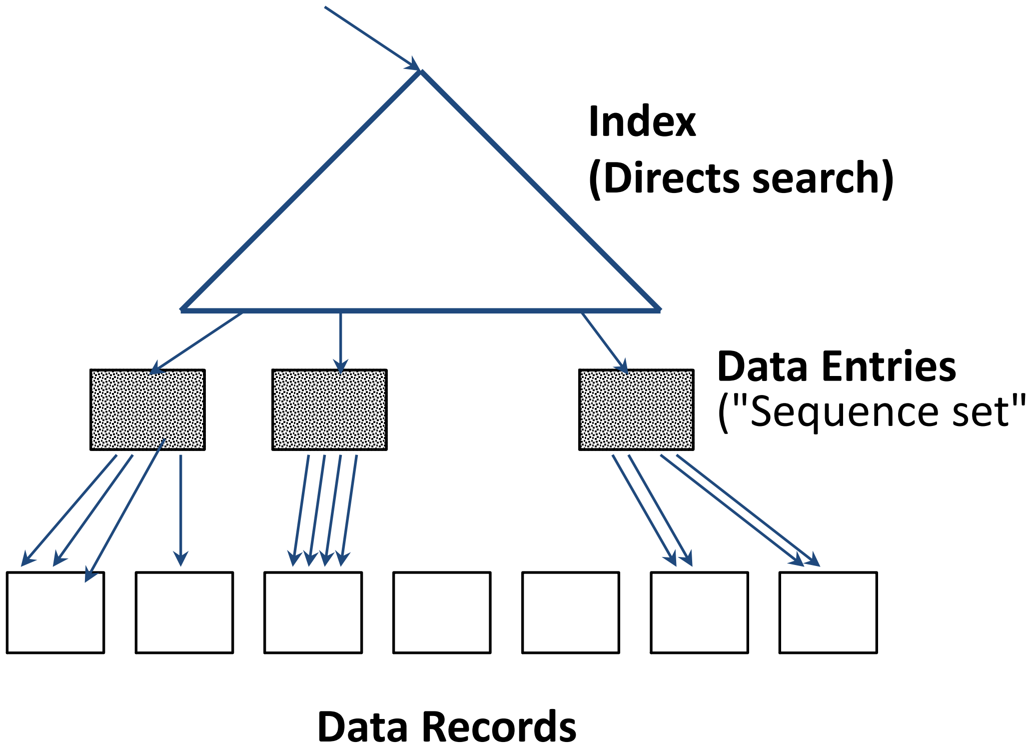
Cases to consider:

- 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)

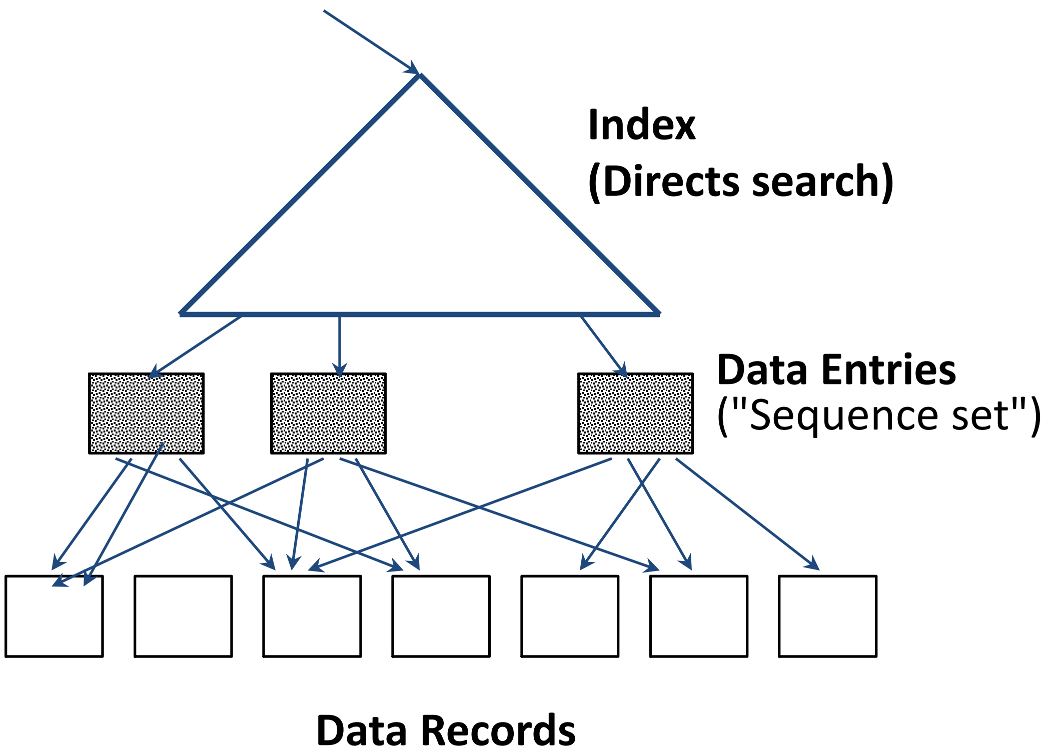
If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.



☛ ***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

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

- ***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 important

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 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, still improving!

Clustered B+ tree is good for sorting

Unclustered tree is usually very bad