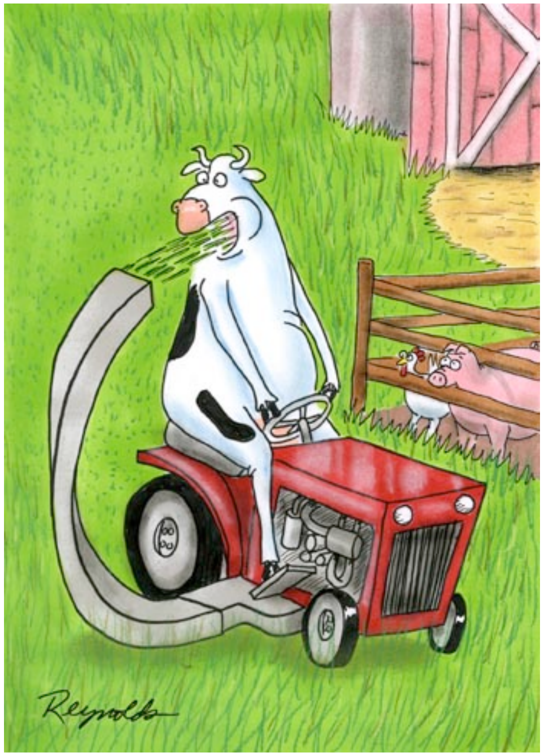
External Sorting

Chapter 13

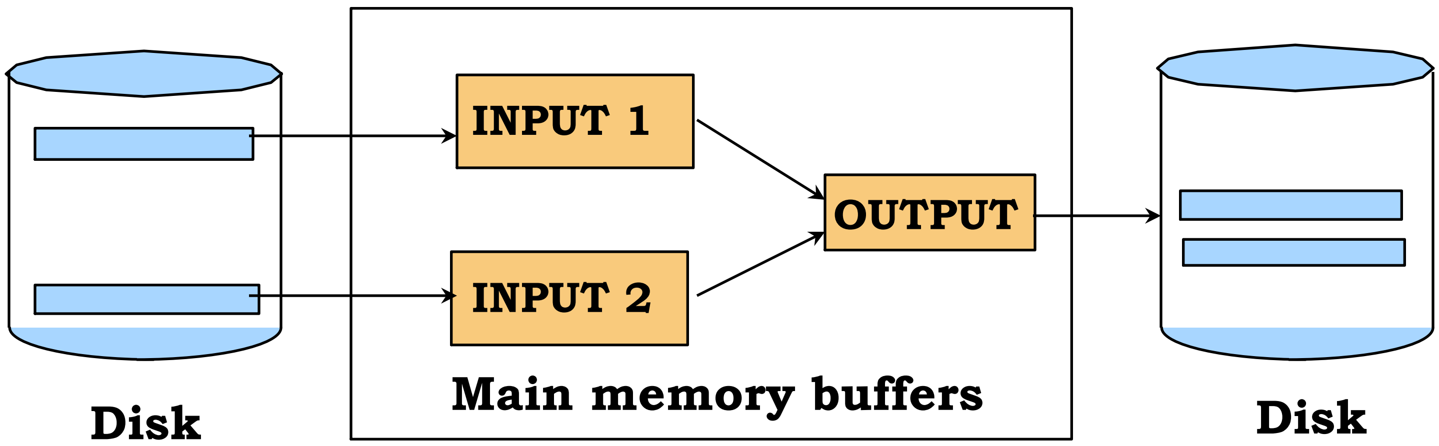


Why Sort?

- A classic problem in computer science!
- Advantages of requesting data in sorted order
  - gathers duplicates
  - allows for efficient searches
- Sorting is first step in *bulk loading* B+ tree index.
- *Sort-merge* join algorithm involves sorting.
- Problem: sort 20Gb of data with 1Gb of RAM.
  - why not let the OS handle it with virtual memory?

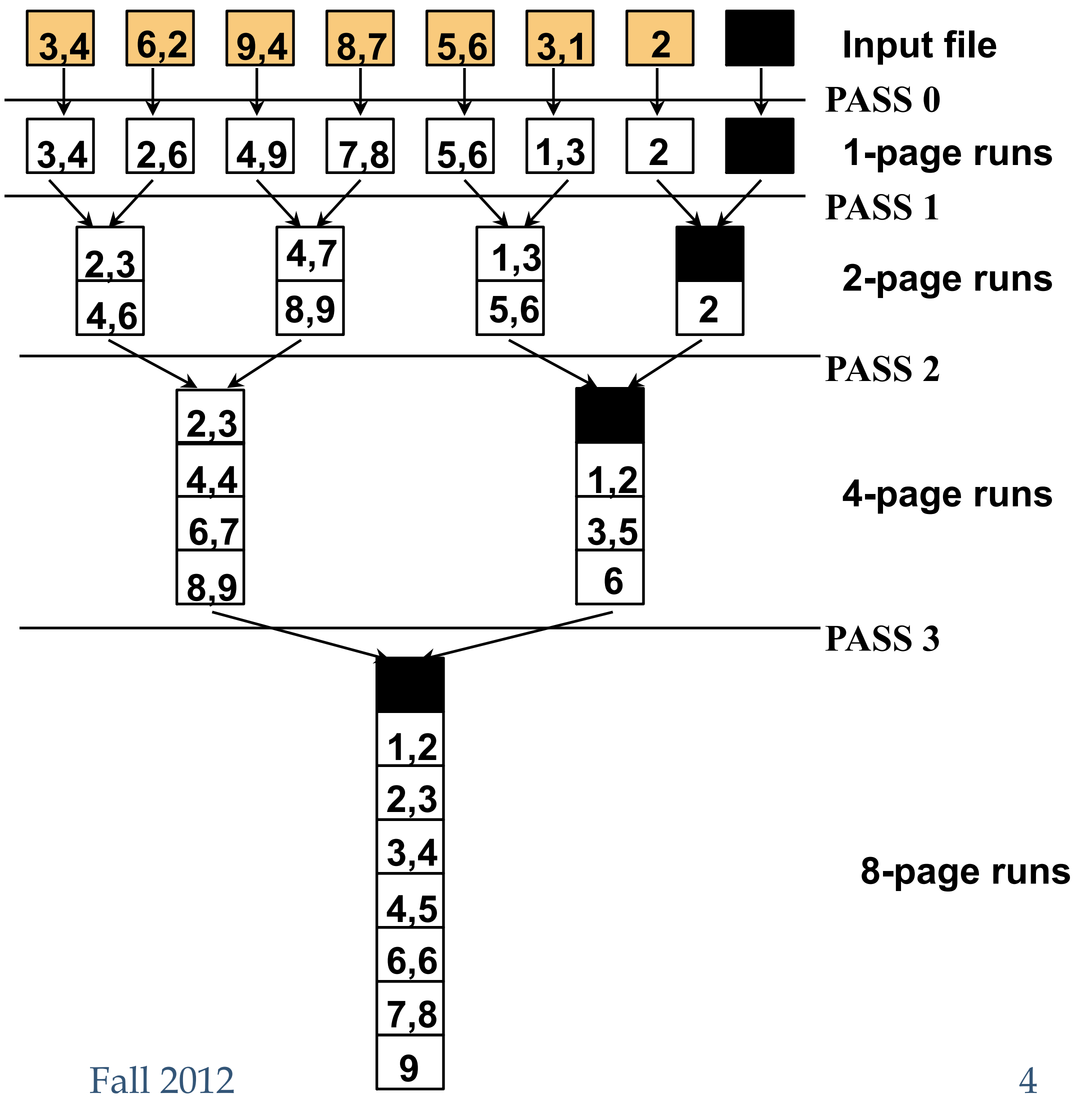
2-Way Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it, write it.
  - only one buffer page is used
- Pass 2, 3, …, N etc.:
  - Read two pages, merge them, and write merged page
  - Requires three buffer pages.



Two-Way External Merge Sort

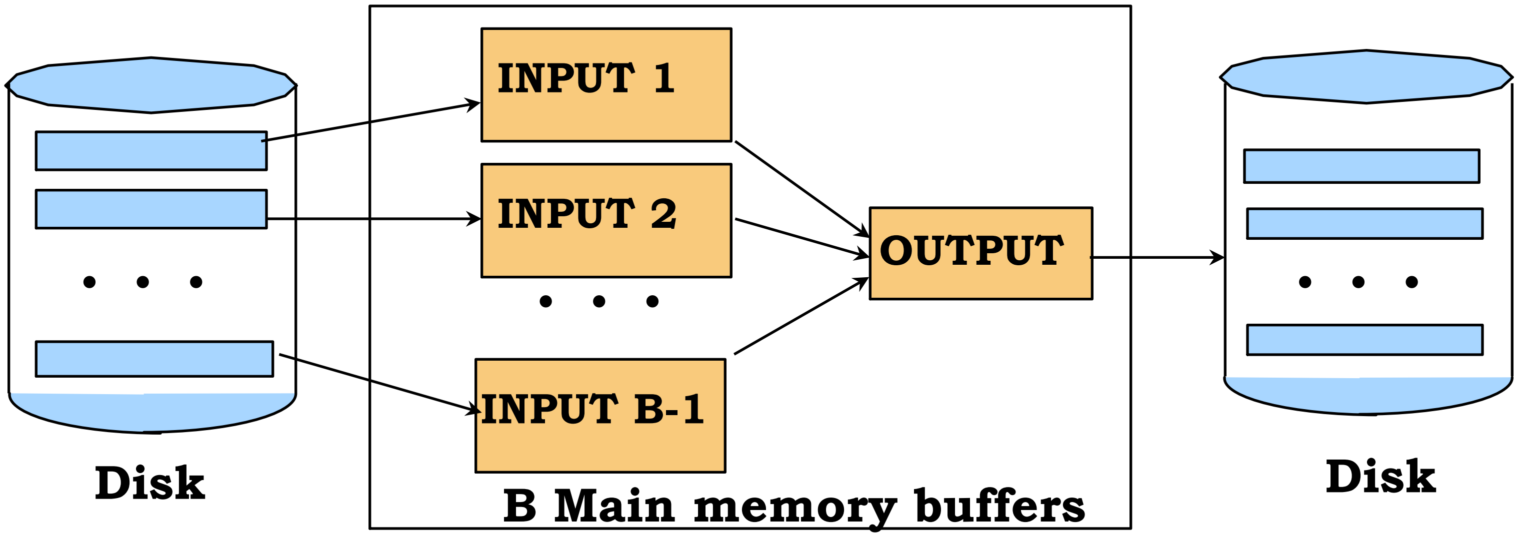
- Each pass we read + write each page in file.
- N pages in the file => the number of passes
  $= \lceil \log_2 N \rceil + 1$
- So toal cost is:
  $$2N\left(\lceil \log_2 N \rceil + 1\right)$$
- *<u>Idea:</u>* ***Divide and conquer:*** sort pages and merge



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 2, …, etc.: merge B-1 runs.



General External Merge Sort

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

- Key Insight #1: We can merge more than 2 input buffers at a time… affects fanout → base of log!
- Key Insight #2: The output buffer is generated incrementally, so only one buffer page is needed for any size of run!
- 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 2, …, etc.: merge *B-1* runs, leaving one page for output.

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: $\lceil 6/4 \rceil$= 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Number of Passes of External Sort

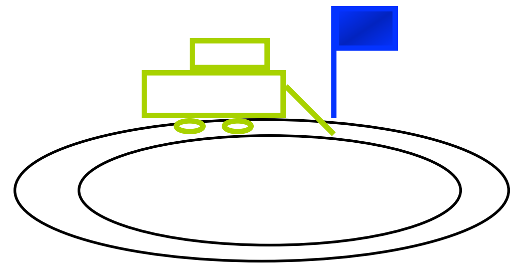
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Internal Sort Algorithm

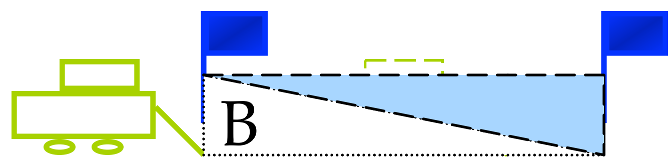
- Quicksort is a fast way to sort in memory.
  - Very fast on average
- Alternative is "replacement sort"
  - **`Top:`** Read in *B* blocks
  - **`Fill:`** Find the smallest record greater than the largest value to output buffer
    - add it to the end of the output buffer
    - fill moved record's slot with next value from the input buffer, if empty refill input buffer
  - else **`end run`**
  - goto **`Fill`**

More on Replacement Sort

- Fact: average length of a run is *2B*
- The "snowplow" analogy
  - Imagine a snowplow moving around a circular track on which snow falls at a steady rate.



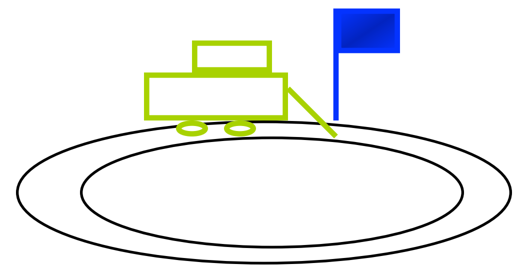
  - At any instant, there is a certain amount of snow S on the track. Some falling snow comes in front of the plow, some behind.



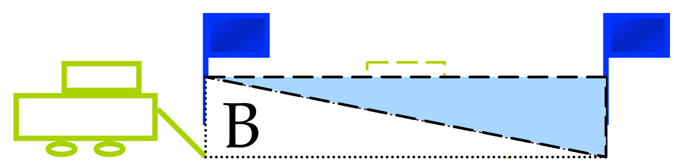
  - During the next revolution of the plow, all of this is removed, plus 1/2 of what falls during that revolution.
  - Thus, the plow removes 2S amount of snow.

More on Replacement Sort

- Fact: average length of a run in heapsort is *2B*
  - The "snowplow" analogy
- 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 ...

I/O for External Merge Sort

- … longer runs imply fewer passes!
- Actually, do I/O a page at a time
- In fact, read a *block* of pages sequentially!
- Suggests we should make each buffer (input/ output) be a *block* of pages.
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 2-3 passes.

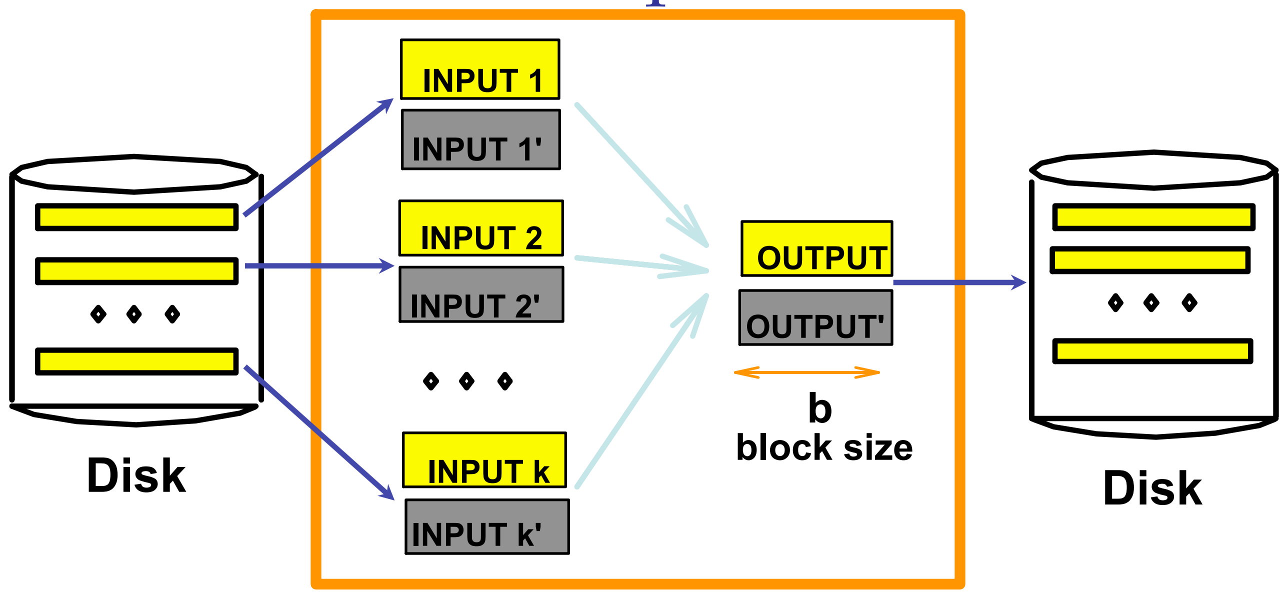
Number of Passes of Optimized Sort

N	B=1,000	B=5,000	B=10,000
100	1	1	1
1,000	1	1	1
10,000	2	2	1
100,000	3	2	2
1,000,000	3	2	2
10,000,000	4	3	3
100,000,000	5	3	3
1,000,000,000	5	4	3

☛ *Block size = 32, initial pass produces runs of size 2B.*

Double Buffering

- To reduce wait time for I/O request to complete, can *prefetch* into a "shadow block".
- Potentially, more passes; in practice, most files *still* sorted in 2-3 passes.



B main memory buffers, k-way merge

Sorting Records!

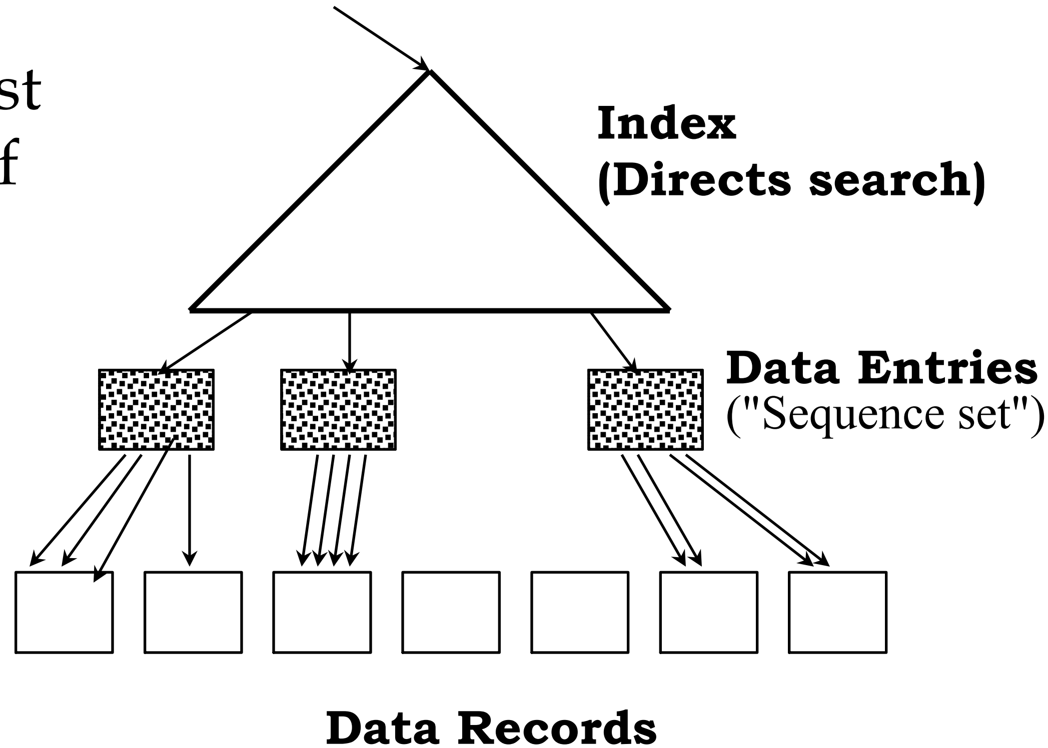
- Sorting has become a blood sport!
  - Parallel external sorting is the name of the game …
- 2005 IBM Almaden
  - Sort 1Tbyte of 100 byte records
  - Typical DBMS: > 5 days
  - World record: ***17 min, 37 seconds***
    - RS/6000 SP with 488 nodes
    - Each node: 4, 332MHz 604 processors, 1.5GB of RAM, and a 9GB SCSI disk
- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?

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

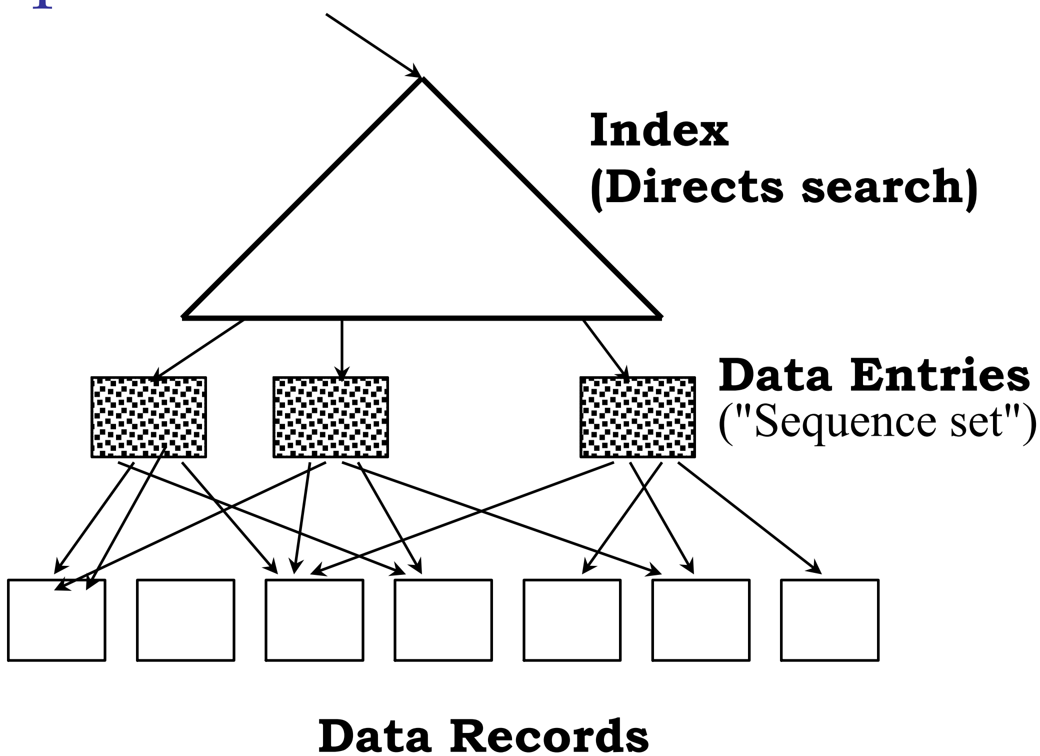
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.
- Fill factor of < 100% introduces a small overhead extra pages fetched



☛ *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; DBMS may dedicate part of buffer pool just for sorting!
- 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 runs rarely more than 2 or 3.

Summary, cont.

- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Replacement sort: slower (2x), but with longer runs
- The best sorts are wildly fast:
  - Despite 40+ years of research, we're still improving!
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.