External Sorting

CS634
Lecture 10

Slides based on "Database Management Systems" 3rd ed, Ramakrishnan and Gehrke

Why is Data Sorting Important?

- Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- Sorting is first step in *bulk loading* B+ tree index
- Sorting useful for eliminating *duplicate copies*
  - Needed for set operations, DISTINCT operator
- *Sort-merge join* algorithm involves sorting

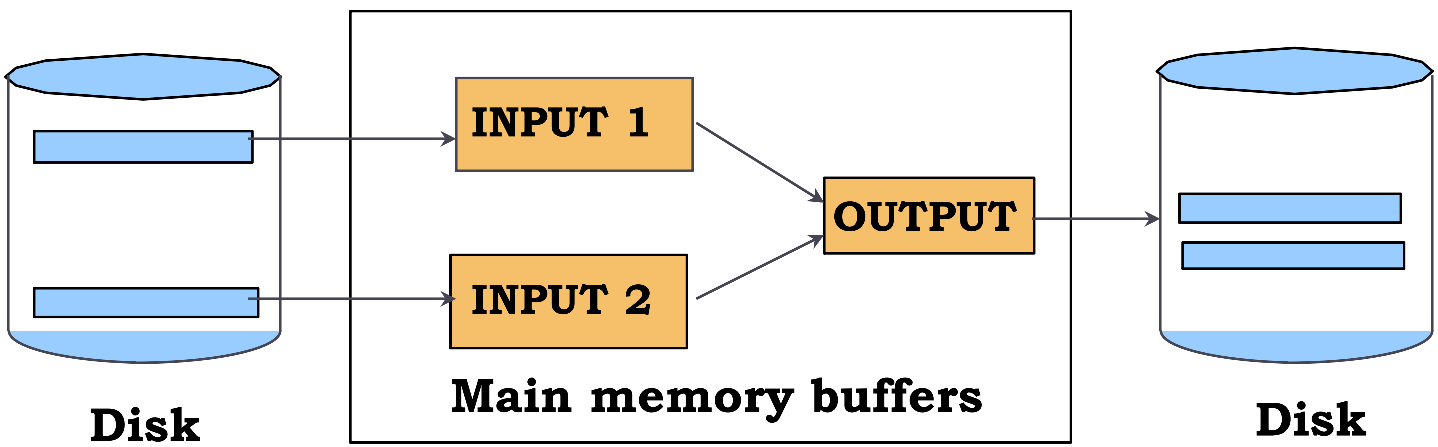
- Problem: sort 1Gb of data with 1MB of RAM, or 100MB
  - Sort is given a memory budget, can use temp disk as needed
  - Focus is minimizing I/O, not computation as in internal sorting

In-memory vs. External Sorting

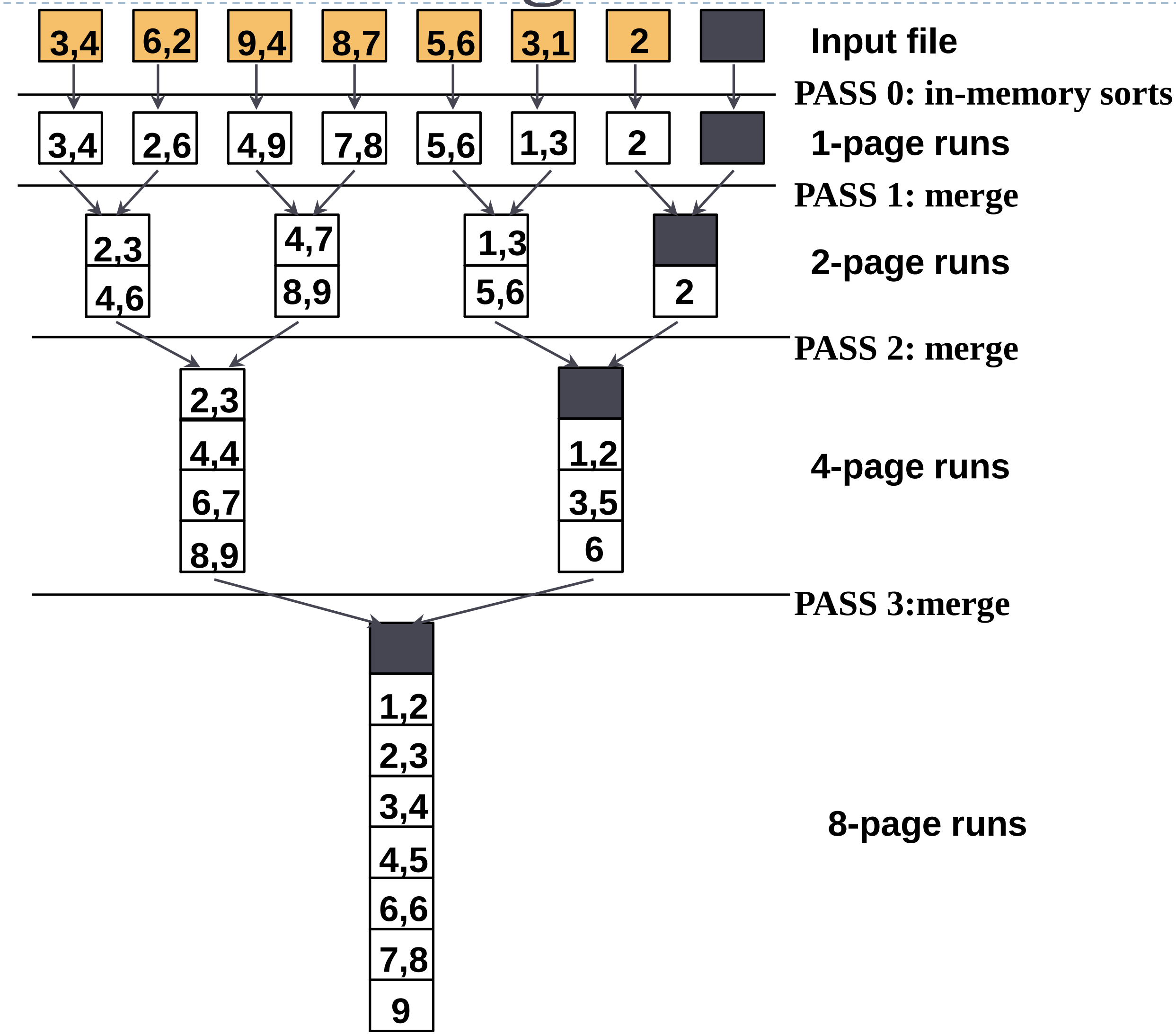
- If data fits in memory allocated to a sort, an in-memory sort does the job.
- Otherwise, need external sorting, i.e., sort batches of data in memory, write to files, read back, merge,…

2-Way External Sort: Requires 3 Buffers

- Pass 1: Read a page, sort it (in-memory sort), write it
  - only one buffer page is used
- Pass 2, 3, …, etc.:
  - three buffer pages used



Two-Way External Merge Sort



Two-Way External Merge Sort

- Each pass we read + write each page in file.
- Number of pages *N* in the file determines number of passes

  Ex: N = 7, round up to power-of-two 8 = 2^3, #passes = 4 (last slide)

  Here 3 = $\log_2 8$ = ceiling($\log_2 7$), so 4 = ceiling($\log_2 N$) + 1
- Total number of passes is, using ceiling notation:

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

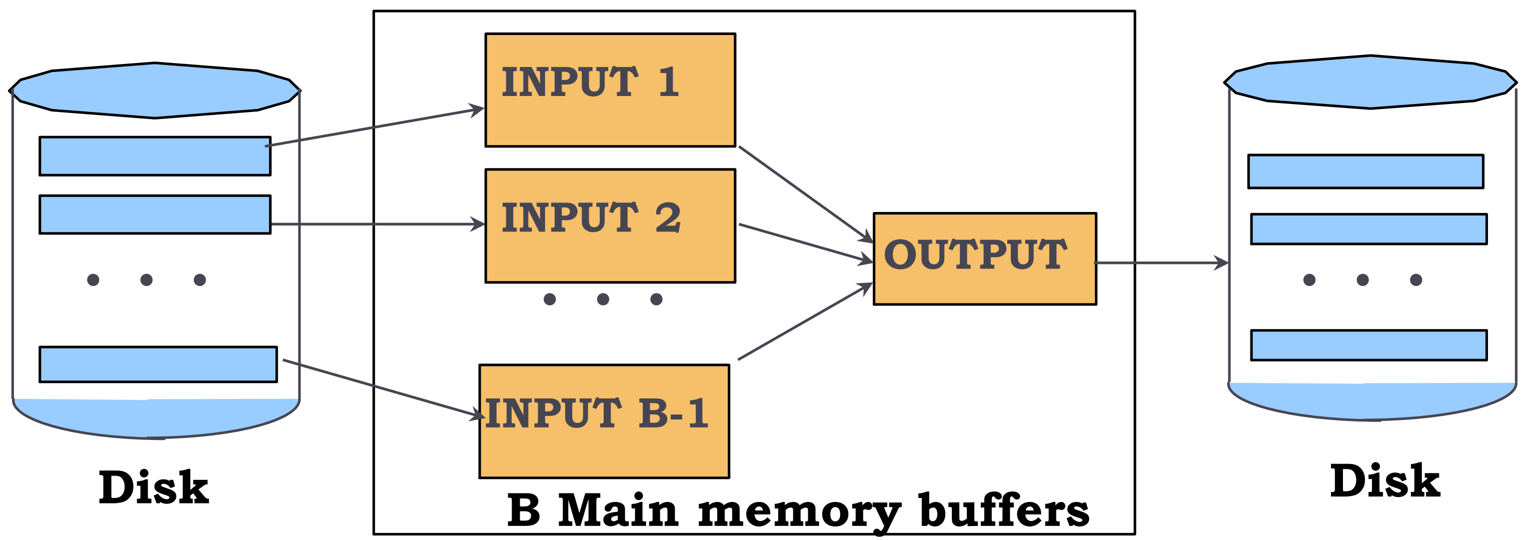
- Total cost is: write & read all N pages for each pass:

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

General External Merge Sort

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

- To sort a file with N > B pages* using B buffer pages:
  - Pass 0: use B buffer pages. Produce $\lceil N / B \rceil$ sorted runs of B pages each.
    - *Example: B=10, N=120, N/B = 12, so 12 runs of 10 pages*
  - Pass 1, …, etc.: merge B-1 runs.



*If N <= B, use an in-memory sort

Cost of External Merge Sort, as on pg. 427,
with yellow over over-simplistic conclusion: see next slide

- Example: with 5 buffer pages, 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: ceiling(6/4) = 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Merge 2 runs to produce sorted file of 108 pages

  Note 22 rounds up to power-of-4 64 = 4^3 so we see 3 passes of merging using (up to) 4 input runs, each with one input buffer.

  3 = ceiling($\log_4$ 22) where 4 = B-1 and 22 = ceiling(N/B)

  plus the initial pass, so 4 passes in all.

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

Cost = 2N * (# of passes) = 2*108*4 i/os

- This cost assumes the data is read from an input file and written to another output file, and this i/o is counted

Cost of External Merge Sort

- Example: with 5 buffer pages, sort 108 page file:
  - Pass 0: ceiling(108/4) = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: ceiling(22/4) = 6 sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: ceiling(6/4) = 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Merge 2 runs into sorted file of 108 pages

  Note 22 rounds up to power-of-4 64 = 4^3 so we see 3 passes of merging using (up to) 4 input runs, each with one input buffer.

  3 = ceiling($\log_4$ 22) where 4 = B-1 and 22 = ceiling(N/B)

  plus the initial pass, so 4 passes in all.
- Number of passes: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- But the passes are not always all the same cost: look at writes and reads over whole run (including any reading input from a file and/or writing the output of the sort to a file, if not pipelined)
  - [Read N],write N, read N, write N, read N, write N, read N, [write N]
  - The bracketed amounts depend on whether or not the data is read from a file at the start and written to a file at the end, or pipelined in and/or out.
- That's 6N, 7N, or 8N i/os, not always the 8N as given in the book's formula
- Cost = N * (# of read/writes of N) = 2N(#passes - 1) up to 2N(#passes)

Cost of External Merge Sort, bigger file

- Number of passes (N>B): $$1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$$
- Cost = 2N * (# of passes) = O(NlogN) like other good sorts
- Example: with 5 buffer pages, sort 250 page file, including reading the input data from a file and writing the output data to another file.
  - Pass 0: ceiling(250/5) = 50 sorted runs of 5 pages each
  - Pass 1: ceiling(50/4) = 13 sorted runs of 20 pages each (last run is only 10 pages)
  - Pass 2: ceiling(13/4) = 4 sorted runs, 80 pages and 10 pages
  - Pass 3: Sorted file of 250 pages

Note 50 again rounds up to power-of-4 64 = 4^3 so we see 3 passes of merging using (up to) 4 input runs, plus the initial pass, so 4 passes again

Cost = 2*250*4 i/os

But 50 is getting up in the vicinity of 64, where we start needing another pass

Cases in sorting

- N <= B: data fits in memory: in-memory sort
- B < N <= B*(B-1): 2-pass external sort
  - (create up to B-1 runs of B pages, do one big merge)
- B*(B-1) < N <= $B*(B-1)^2$ 3-pass external sort
  - (create up to $(B-1)^2$ runs of B, do merge to B-1 runs, do second merge pass)

- If B = 10K (80MB with 8KB blocks, ordinary size now)
  - B*(B-1) = 10K*10K = 100M blocks = 800MB: max for 2-pass sort
  - $B*((B-1)^2$ = 1000G = 1T = 8TB: max for 3-pass sort
- So rare to see 4-pass sort today
- We made a graph of showing Cost = 2*N for N range of 2-pass sort, Cost = 4*N for higher N, causing jump in Cost at start of 3-pass region

Number of Passes of External Sort (from text pg, 428)

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

All these B values look tiny today!

Internal Sort Algorithms

- Quicksort is a fast way to sort in memory.
- An alternative is "tournament sort" (a.k.a. "heapsort")
- Radix sort is another
- This is studied in data structures

I/O for External Merge Sort

Assumed so far that we do I/O a page at a time (say 4KB or 8KB)

But larger-block disk I/O is much faster (not on SSD, however)

Ex: 4KB takes 4ms, 100KB takes 5ms (approx.)

- In fact, we can read a *block* of pages sequentially!
- Suggests we should make each buffer (input/output) be a *block* of pages
  - Need cooperation with buffer manager, or own buffer manager
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 1-2 passes

HDD vs. SSD

- HDD typical values:
  - 100 io/s random reads/writes
  - 100 MB/s sequential read or write
  - Means 100*8KB/s = 800 KB/s = .8MB/s using 8KB random reads
  - That's less than 1% of sequential reading speed!
  - In DB case with tablespaces, not really "random i/o", so say multiblock i/o is 25x faster than block i/o.
- SSD typical values: 5x faster sequential i/o, 125x faster on multiblock i/o, but 10x cost/GB.
  - 500 MB/s sequential read, also write on new SSD
  - Writes slow down on full disk (needs to erase before write)
  - 8KB ios: (500MB/s)/8KB = 64K io/s
  - See higher numbers in product literature, but need many i/os in progress to do that.

Example of a Blocked I/O Sort

Example: N=1M blocks, B=5000 blocks memory for sort

Use 32 blocks in a big buffer, so have 5000/32 = 156 big buffers

File is 1M/32 = 31250 big blocks

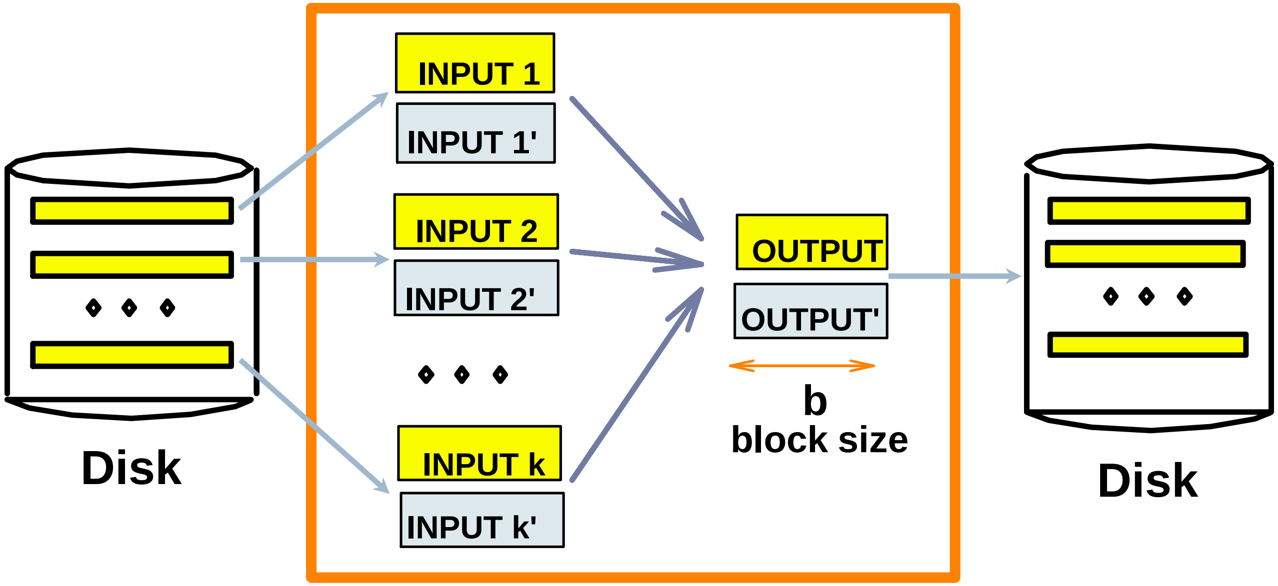
- Pass 0: sort using 156 big buffers to first runs: get ceiling(31250/156) = 201 runs
- Pass 1: merge using 155 big buffers to 2 runs
- Pass 2: merge 2 runs to final result

See 3 passes here, vs. 2 using "optimized" sort, pg. 431

- Cost = 2N*3 = 6N, vs. 4N using ordinary blocks
- But I/O is 4ms vs. (5/32)ms, so 6*(5/32)=1 vs. 4*4 = 16, a win.

Prefetching to speed up reading

- 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

Prefetching, tuning i/o

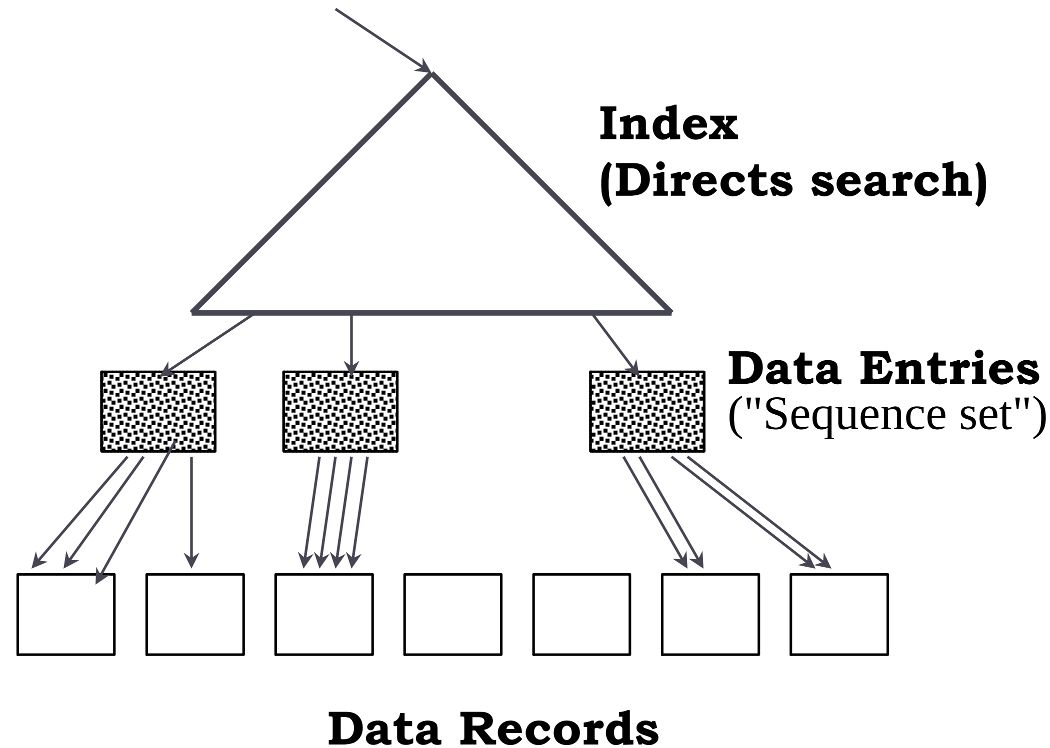
- Note this is a general algorithm, not just for sorting
- Can be used for table scans too
- Database have I/O related parameters
- **Oracle:**
- DB_FILE_MULTIBLOCK_READ_COUNT
- Says how many blocks to read at once in a table scan

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!***

(Already existent) Clustered B+ Tree Used for Sorting

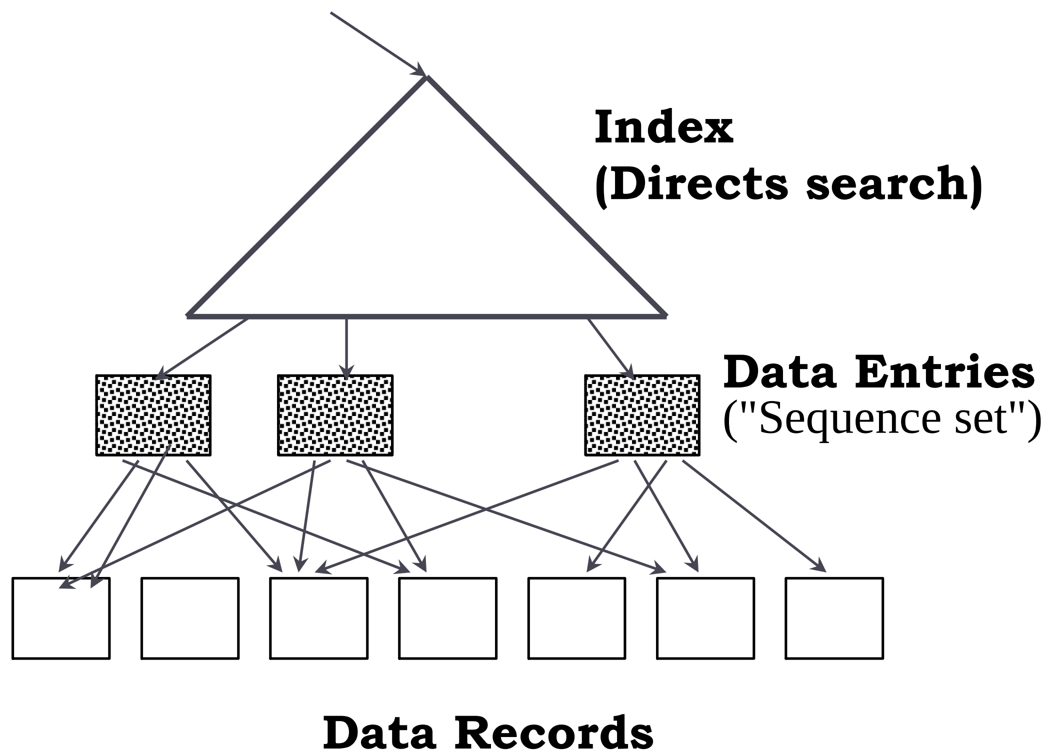
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used, additional cost of retrieving data records: each page fetched just once



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 (*p=100 is the more realistic value*)**
- ***B=1,000 and block size=32 for sorting***
- ***Assumes the blocks are never found in the buffer pool***

Sorting Records: Benchmarks

- Parallel sorting benchmarks/competitions exist in practice
- Datamation: Sort 1M records of size 100 bytes (considered obsolete now at [http://sortbenchmark.org/)](http://sortbenchmark.org/)
  - Typical DBMS: 5 minutes
  - 2001: .48 sec. at UW (most recent I could find)
  - Oracle on dbs3: sorts 80MB in 32 sec. with 8GB for PGA.
- Newer benchmarks:
  - Minute Sort: How many TB can you sort in 1 minute?
    - 2016: 37TB/55TB Tencent Sort at Tencent Corp., China
  - Cloud Sort: How much in USD to sort 100 TB using a public cloud

    2015: $451 on 330 Amazon EC2 r3.4xlarge nodes, by profs at UCSD.

    2016: $144 using Alibaba Cloud, by profs at Nanjing U, others

Oracle on dbs3: .5 min to sort 1M records, 11 min to sort 250M records

- Select median(k250k) from bench250;
  - 250M records, roughly 2GB data (250M*8bytes/row)
- Suppose Oracle allots 100 MB for this sort
  - Then 2GB/100MB= 20 runs in pass 0
  - 100MB/8KB = 12800 pages of buffer (B=12800)
- So pass 1 merges 20 runs into final sorted output
- The DB reads the table (1.7 min) then writes/reads the 2GB, then the output is processed on the fly.
  - 2*2GB/8KB = (1/2)M i/o, at about 500 i/o/s
  - (1/2)Mi/os/(500 i/os/s) = 250 s = 4.2 min, plus 1.7 = 6 min
- Works out roughly. First reads = table scan, faster because of contiguous data, prefetching
- Additional 5 min or so for in-memory sorting, CPU bound

Pipelined Sort Engine

- How it works: stream of tuples in, stream of tuples out:
  - Initialize/create sort object: given B, number memory buffers
  - Put_tuple, put_tuple, put_tuple,… add data
  - Get_tuple: hangs for a while, returns first sorted tuple
  - Get_tuple, get_tuple, … rest of sorted tuples
  - Done!
- The sort doesn't need to know how many tuples will be added!
- It just fills B buffers, sorts, outputs run, fills again, …
- When it sees get_tuple, it does know how much data is involved, can plan a multi-pass sort if needed.
- This possibility of pipelined sort is mentioned on pg. 496, but usually the authors assume file-to-file sort
- This adds write N, read N to the plan, 2N to cost.

Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting! Oracle: separate memory area
- 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, for properly managed database and decent sized memory.
  - Using SSD: 5x faster for this needed sequential i/o, but writes may slow down over time.

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, we're still improving!
- Clustered B+ tree is good for avoiding sorting; unclustered tree is usually useless.