CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture 11: external sorting and query evaluation

(R&G ch. 13 and 14)

CMU SCS

Why Sort?

CMU SCS

Why Sort?

- select ... order by
  - e.g., find students in increasing *gpa* order
- *bulk loading* B+ tree index.
- *duplicate elimination* (select distinct)
- select ... group by
- *Sort-merge* join algorithm involves sorting.

CMU SCS

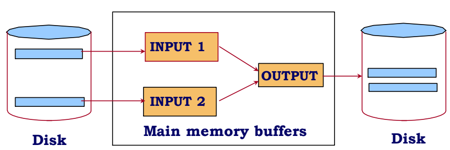
Outline

- ➡ two-way merge sort
- external merge sort
- fine-tunings
- B+ trees for sorting

CMU SCS

2-Way Sort: Requires 3 Buffers

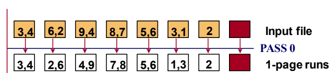
- Pass 0: Read a page, sort it, write it.
  - only one buffer page is used
- Pass 1, 2, 3, …, etc.: requires 3 buffer pages
  - merge pairs of **runs** into runs twice as long
  - three buffer pages used.



CMU SCS

Two-Way External Merge Sort

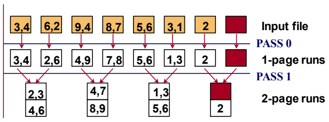
- Each pass we read + write each page in file.



CMU SCS

Two-Way External Merge Sort

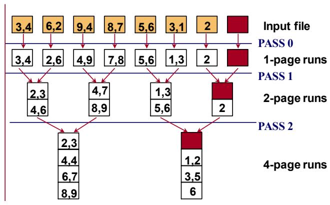
- Each pass we read + write each page in file.



CMU SCS

Two-Way External Merge Sort

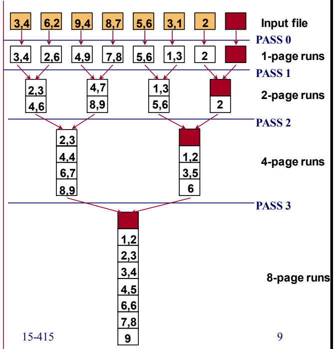
- Each pass we read + write each page in file.



CMU SCS

Two-Way External Merge Sort

- Each pass we read + write each page in file.



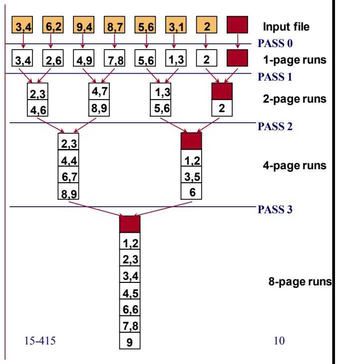
CMU SCS

Two-Way External Merge Sort

- Each pass we read + write each page in file.
- N pages in the file => $= \lceil \log_2 N \rceil + 1$
- So total cost is:

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

- *Idea:* ***Divide and conquer:*** sort subfiles and merge



CMU SCS

Outline

- two-way merge sort
- ➡ external merge sort
- fine-tunings
- B+ trees for sorting

CMU SCS

External merge sort

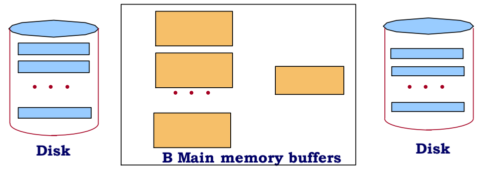
B > 3 buffers

- Q1: how to sort?
- Q2: cost?

CMU SCS

General External Merge Sort

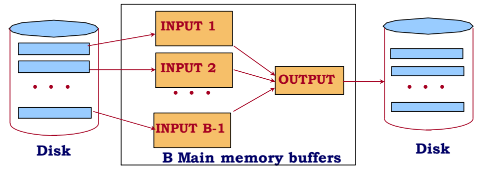
B>3 buffer pages. How to sort a file with N pages?



CMU SCS

General External Merge Sort

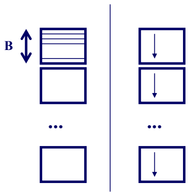
- Pass 0: use *B* buffer pages. Produce $\lceil N/B \rceil$ sorted runs of *B* pages each.
- Pass 1, 2, …, etc.: merge *B-1* runs.



CMU SCS

Sorting

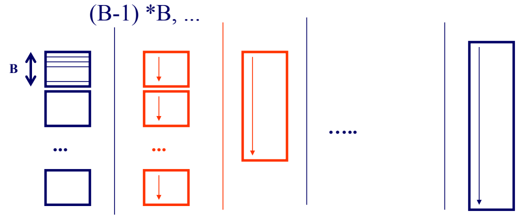
- create sorted runs of size B (how many?)
- merge them (how?)



CMU SCS

Sorting

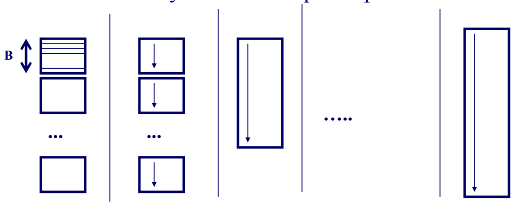
- create sorted runs of size B
- merge first B-1 runs into a sorted run of (B-1) *B, ...



CMU SCS

Sorting

- How many steps we need to do? 'i', where B*(B-1)^i > N
- How many reads/writes per step? N+N



CMU SCS

Cost of External Merge Sort

- Number of passes: $1+\lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Cost = 2N * (# of passes)

CMU SCS

Cost of External Merge Sort

- 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: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Formula check: $\lceil \log_4 22 \rceil$ = 3 … + 1 → 4 passes √

CMU SCS

Number of Passes of External Sort

(I/O cost is 2N times number of passes)

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

CMU SCS

Outline

- two-way merge sort
- external merge sort
- ➡ fine-tunings
- B+ trees for sorting

CMU SCS

Outline

- two-way merge sort
- external merge sort
- fine-tunings
  - ➡ which internal sort for Phase 0?
  - blocked I/O
- B+ trees for sorting

CMU SCS

Internal Sort Algorithm

- Quicksort is a fast way to sort in memory.
- But: we get B buffers, and produce 1 run of length B.
- Can we produce longer runs than that?

CMU SCS

Internal Sort Algorithm

- Quicksort is a fast way to sort in memory.
- But: we get B buffers, and produce 1 run of length B.
- Can we produce longer runs than that?

Heapsort:
- Pick smallest
- Output
- Read from **next** buffer

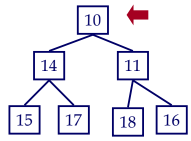
Internal Sort Algorithm

- Quicksort is a fast way to sort in memory.
- But: we get B buffers, and produce 1 run of length B.
- Can we produce longer runs than that?
- Alternative: “tournament sort” (a.k.a. “heapsort”, “replacement selection”)
- Produces runs of length ~ 2*B
- Clever, but not implemented, for subtle reasons: tricky memory management on variable length records

details

Reminder: Heapsort

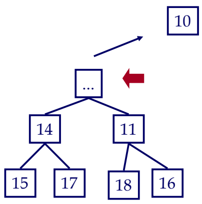
pick smallest, write to output buffer:



details

Heapsort:

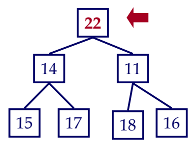
pick smallest, write to output buffer:



details

Heapsort:

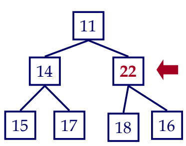
get next key; put at top and ‘sink’ it



details

Heapsort:

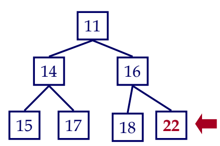
get next key; put at top and ‘sink’ it



details

Heapsort:

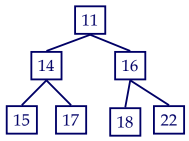
get next key; put at top and ‘sink’ it



CMU SCS

details

Heapsort:



When done, pick top (= smallest) and output it, if 'legal' (ie., >=10 in our example

This way, we can keep on reading new key values (beyond the B ones of quicksort)

Faloutsos 15-415 31

CMU SCS

Outline

- two-way merge sort
- external merge sort
- fine-tunings
  - which internal sort for Phase 0?
  - ➡ blocked I/O
- B+ trees for sorting

Faloutsos 15-415 32

CMU SCS

Blocked I/O & double-buffering

- So far, we assumed random disk access
- Cost changes, if we consider that runs are written (and read) sequentially
- What could we do to exploit it?

Faloutsos 15-415 33

CMU SCS

Blocked I/O & double-buffering

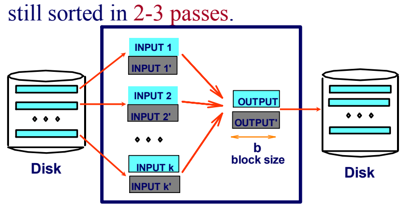
- So far, we assumed random disk access
- Cost changes, if we consider that runs are written (and read) sequentially
- What could we do to exploit it?
- A1: Blocked I/O (exchange a few r.d.a for several sequential ones)
- A2: double-buffering

Faloutsos 15-415 34

CMU SCS

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

Faloutsos 15-415 35

CMU SCS

Outline

- two-way merge sort
- external merge sort
- fine-tunings
- ➡ B+ trees for sorting

Faloutsos 15-415 36

CMU SCS

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

CMU SCS

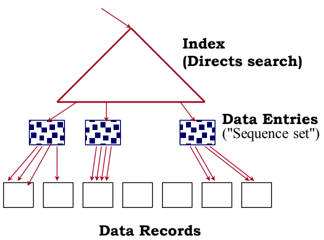
Using B+ Trees for Sorting

- 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!

CMU SCS

Clustered B+ Tree Used for Sorting

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

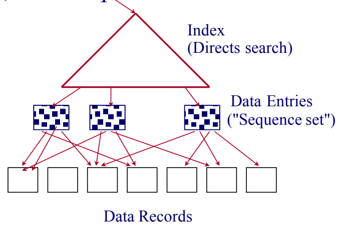


Always better than external sorting!

CMU SCS

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!*



CMU SCS

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.

CMU SCS

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.
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.

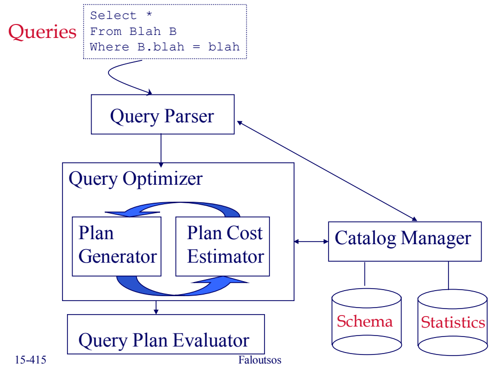
CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- (14.3.2) Hashing

CMU SCS

Cost-based Query Sub-System



CMU SCS

Schema

- What would you store?
- How?

CMU SCS

Schema

- What would you store?
- A: info about tables, attributes, indices, users
- How?
- A: in tables! eg.,
  - Attribute_Cat (attr_name: **string**, rel_name: **string**; type: **string**; position: **integer**)

CMU SCS

Statistics

- Why do we need them?
- What would you store?

CMU SCS

Statistics

- Why do we need them?
- A: To estimate cost of query plans
- What would you store?
  - NTuples(R): # records for table R
  - NPages(R): # pages for R
  - NKeys(I): # distinct key values for index I
  - INPages(I): # pages for index I
  - IHeight(I): # levels for I
  - ILow(I), IHigh(I): range of values for I
  - ...

CMU SCS

Outline

- (12.1) Catalog
- ➡ (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- (14.3.2) Hashing

CMU SCS

Operator evaluation

3 methods we'll see often:

CMU SCS

Operator evaluation

3 methods we'll see often:
- indexing
- iteration (= seq. scanning)
- partitioning (sorting and hashing)

CMU SCS

``Access Path''

- Eg., index (tree, or hash), or scanning
- Selectivity of an access path:
  - % of pages we retrieve
- eg., selectivity of a hash index, on range query: 100% (no reduction!)

CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- ➡ (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- (14.3.2) Hashing

CMU SCS

Algorithms

- selection:
- projection
- join
- group by
- order by

Algorithms

- selection: scan; index
- projection (dup. elim.):
- join
- group by
- order by

Algorithms

- selection: scan; index
- projection (dup. elim.): hashing; sorting
- join
- group by
- order by

Algorithms

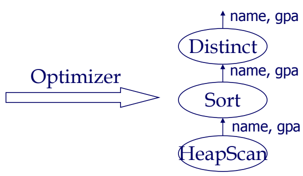
- selection: scan; index
- projection (dup. elim.): hashing; sorting
- join: many ways (loops, sort-merge, etc)
- group by
- order by

Algorithms

- selection: scan; index
- projection (dup. elim.): hashing; sorting
- join: many ways (loops, sort-merge, etc)
- group by: hashing; sorting
- order by: sorting

Iterator Interface

SELECT DISTINCT name, gpa
FROM Students



iterator

Iterators

- Relational operators: subclasses of **iterator**:

```
class iterator {
    void init();
    tuple next();
    void close();
    iterator &inputs[];
   // additional state goes here
  }
```

- iterators can be cascaded

CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- ➡ (12.6) Typical Q-optimizer
- (14.3.2) Hashing

15-415 Faloutsos 61

CMU SCS

Q-opt steps

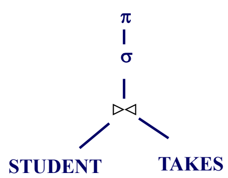
- bring query in internal form (eg., parse tree)
- … into 'canonical form' (syntactic q-opt)
- generate alt. plans
- estimate cost; pick best

15-415 Faloutsos 62

CMU SCS

Q-opt - example

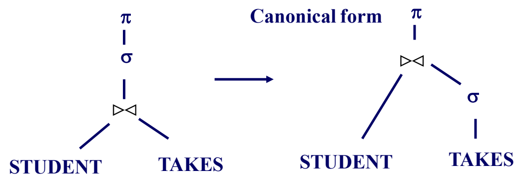
select name
from STUDENT, TAKES
where c-id='415' and
STUDENT.ssn=TAKES.ssn



15-415 Faloutsos 63

CMU SCS

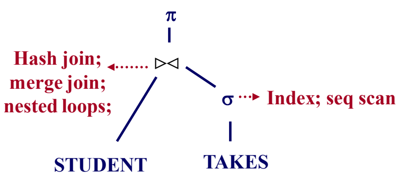
Q-opt - example



15-415 Faloutsos 64

CMU SCS

Q-opt - example



15-415 Faloutsos 65

CMU SCS

Outline

- (12.1) Catalog
- (12.2) Intro to Operator Evaluation
- (12.3) Algo's for Relational Operations
- (12.6) Typical Q-optimizer
- ➡ (14.3.2) Hashing

15-415 Faloutsos 66

CMU SCS

Grouping; Duplicate Elimination

select distinct ssn
from TAKES

- (Q1: what does it do, in English?)
- Q2: how to execute it?

CMU SCS

An Alternative to Sorting: Hashing!

- Idea:
  - maybe we don't need the *order* of the sorted data
  - e.g.: forming groups in GROUP BY
  - e.g.: removing duplicates in DISTINCT
- Hashing does this!
  - And may be cheaper than sorting! (why?)
  - But what if table doesn't fit in memory??

CMU SCS

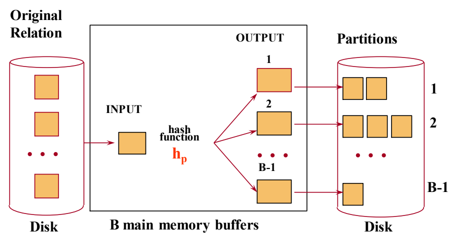
General Idea

- Two phases:
  - Phase1: Partition: use a hash function h_p to split tuples into partitions on disk.
    - We know that all matches live in the same partition.
    - Partitions are "spilled" to disk via output buffers

CMU SCS

Two Phases

- Partition:



CMU SCS

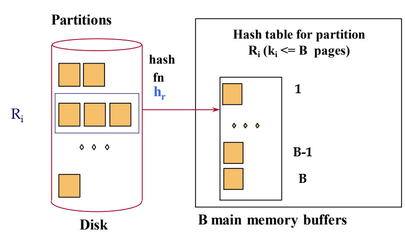
General Idea

- Two phases:
  - Phase 2: ReHash: for each partition on disk, read it into memory and build a main-memory hash table based on a hash function h_r
    - Then go through each bucket of this hash table to bring together matching tuples

CMU SCS

Two Phases

- Rehash:



CMU SCS

Analysis

- How big of a table can we hash using this approach?
  - B-1 “spill partitions” in Phase 1
  - Each should be no more than B blocks big

CMU SCS

Analysis

- How big of a table can we hash using this approach?
  - B-1 “spill partitions” in Phase 1
  - Each should be no more than B blocks big
  - Answer: B(B-1).
    - ie., a table of N blocks needs about sqrt(N) buffers
  - What assumption do we make?

CMU SCS

Analysis

- How big of a table can we hash using this approach?
  - B-1 “spill partitions” in Phase 1
  - Each should be no more than B blocks big
  - Answer: B(B-1).
    - ie., a table of N blocks needs about sqrt(N) buffers
  - Note: assumes hash function distributes records evenly!
    - use a ‘fudge factor’ f>1 for that: we need B ~ sqrt(f * N)

CMU SCS

Analysis

- Have a bigger table? Recursive partitioning!
  - In the ReHash phase, if a partition *b* is bigger than B, then recurse:
  - pretend that *b* is a table we need to hash, run the Partitioning phase on *b*, and then the ReHash phase on each of its (sub)partitions

CMU SCS

break

Real story

- Partition + Rehash
- Performance is very slow!
- What could have gone wrong?

CMU SCS

break

Real story

- Partition + Rehash
- Performance is very slow!
- What could have gone wrong?
- Hint: some buckets are empty; some others are way over-full.

CMU SCS

Hashing vs. Sorting

- Which one needs more buffers?

CMU SCS

Hashing vs. Sorting

- **Recall: can hash a table of size N blocks in sqrt(N) space**
- How big of a table can we sort in 2 passes?
  - Get N/B sorted runs after Pass 0
  - Can merge all runs in Pass 1 if $N/B \le B-1$
    - Thus, we (roughly) require: $N \le B^2$
    - We can sort a table of size N blocks in about space sqrt(N)
  - Same as hashing!

CMU SCS

Hashing vs. Sorting

- Choice of sorting vs. hashing is subtle and depends on optimizations done in each case ...
  - Already discussed some optimizations for sorting:

CMU SCS

Hashing vs. Sorting

- Choice of sorting vs. hashing is subtle and depends on optimizations done in each case ...
  - Already discussed some optimizations for sorting:
    - Heapsort in Pass 0 for longer runs
    - Chunk I/O into large blocks to amortize seek+RD costs
    - Double-buffering to overlap CPU and I/O
  - Another optimization when using sorting for aggregation:
    - "Early aggregation" of records in sorted runs
  - We will discuss some optimizations for hashing next...

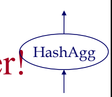
CMU SCS

Hashing: We Can Do Better!

- Combine the summarization into the hashing process - How?

CMU SCS

Hashing: We Can Do Better!



- Combine the summarization into the hashing process - How?
  - During the ReHash phase, don't store tuples, store pairs of the form <GroupVals, RunningVals>
  - When we want to insert a new tuple into the hash table
    - If we find a matching GroupVals, just update the RunningVals appropriately
    - Else insert a new <GroupVals, RunningVals> pair

CMU SCS

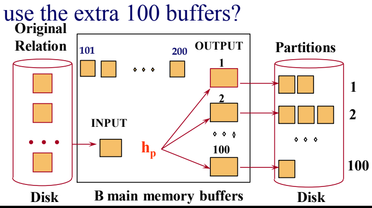
Hashing: We Can Do Better!

- Combine the summarization into the hashing process
- What's the benefit?
  - Q: How many pairs will we have to handle?
  - A: Number of distinct values of GroupVals columns
    - **Not** the number of tuples!!
  - Also probably "narrower" than the tuples

CMU SCS

Even Better: Hybrid Hashing

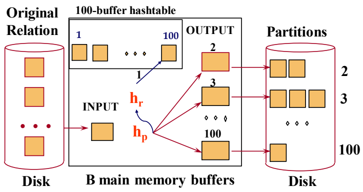
- What if B > sqrt(N)?
- e.g., N=10,000, B=200
- B=100 (actually, 101) would be enough for 2 passes
- How could we use the extra 100 buffers?



CMU SCS

Even Better: Hybrid Hashing

- Idea: hybrid! … keep 1st partition in memory during phase 1!
  - Output its stuff at the end of Phase 1.



CMU SCS

Even Better: Hybrid Hashing

- What if B=**300**? (and N=10,000, again)
- i.e., 200 extra buffers?

CMU SCS

Even Better: Hybrid Hashing

- What if B=**300**? (and N=10,000, again)
- i.e., 200 extra buffers?
- A: keep the first **2** partitions in main memory

CMU SCS

Even Better: Hybrid Hashing

- What if B=**150**? (and N=10,000, again)
- i.e., 50 extra buffers?

CMU SCS

Even Better: Hybrid Hashing

- What if B=**150**? (and N=10,000, again)
- i.e., 50 extra buffers?
- A: keep half of the first bucket in memory

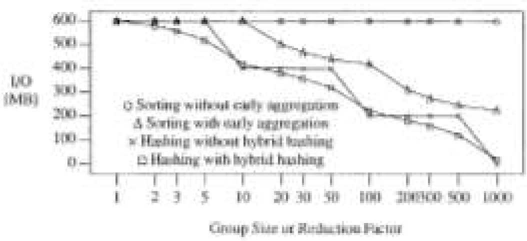
CMU SCS

Hybrid hashing

- can be used together with the summarization idea

CMU SCS

Hashing vs. Sorting revisited



Notes: (1) based on analytical (not empirical) evaluation
(2) numbers for sort do not reflect heapsort optimization
(3) assumes even distribution of hash buckets

Source: G. Graefe. ACM Computing Surveys, 25(2).

CMU SCS

So, hashing's better … right?

- Any caveats?

CMU SCS

So, hashing's better … right?

- Any caveats?
- A1: sorting is better on non-uniform data
- A2: ... and when sorted output is required later.

Hashing vs. sorting:

- Commercial systems use either or both

CMU SCS

Summary

- Query processing architecture:
  - Query optimizer translates SQL to a query plan = graph of iterators
  - Query executor "interprets" the plan
- Hashing is a useful alternative to sorting
  - Both are valuable techniques for a DBMS