Review (1)

- Consider a file with 6 million records of 200 bytes each. Suppose we have to perform 10,000 single-record accesses, and 100 range queries of 0.005% of the file.
  - Use hashing (with key-to-address transformation of the form x mod y). Suppose the hash table has a load factor of 70% and the bucket size is 4096 bytes. Moreover, assume that records are stored in the bucket, and there is no overflow of buckets.
  - Use B+-tree. Suppose each node is 70% full, and the sizes of a node, key and address are 4096, 8 and 4 bytes respectively.
- Which of the above two methods is better for the application?
- Under what circumstance will the "loser" outperform the "winner"?

Review (2) B+-tree

- Assume that (key,ptr) pairs are stored in leaf nodes. each node = 4096 bytes. let order be d => 2d*8 + (2d+1)*4 ≤ 4096 => d = 170 => each node can store at most 340 keys.
- since each node is 70% full, we have each node storing 238 keys (and 239 pointers).

- => at leaf level, we have 6,000,000/238 = 25211 nodes
- => at level above leaf, we have 25211/239 = 105 nodes
- => next level is the root.
- => the tree has 3 levels.

- for 10,000 single-record accesses, cost = 10,000*(3+1) = 40,000
- for each range query, we need to traverse 2 leaf nodes, and 22 data nodes (assuming data are clustered).
- so, the cost for 100 range queries = 100*(3+1+22) = 2600

- total = 42,600

Review (3) Hash method

- We have 6,000,000 records, each 200 bytes, 10,000 single-record accesses, 100 range queries, each accessing 0.005% of the file, i.e., 300 records.
- bucket size = 4096 bytes = 20 records
- since no overflow, and 70% load factor ==> each bucket contains 14 records only. there are 6,000,000/14 = 428,572 buckets.

- for 10,000 single-record accesses, cost = 10,000 I/O (i.e., 1 I/O per access).

- for each range queries, we need to access the entire file. So, total cost = 100*438,572 I/O

Review (4)

- B+-tree = 40,000 + 2,600

- Hash index = 10,000 + 100*438,572

- clearly, the winner is B+-tree.

- if the range queries cover almost the entire file, or the workload has few range queries, then hashing technique will win.

External Sort

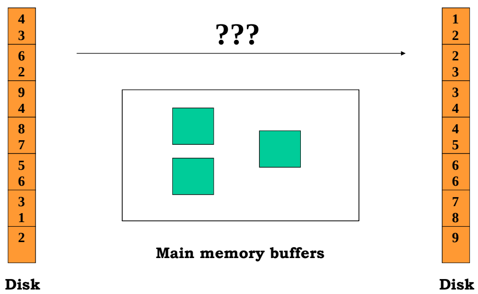
"There it was, hidden in alphabetical order."

Rita Holt

External Sorting

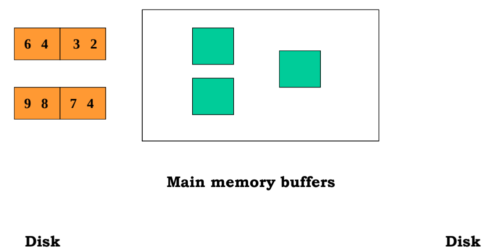
- A classic problem in computer science!
- Data requested in sorted order
  - e.g., find students in increasing *cap* order
- Sorting is used in many applications
  - First step in *bulk loading* operations.
  - Sorting useful for eliminating *duplicate copies* in a collection of records (How?)
  - *Sort-merge* join algorithm involves sorting.

Challenge: Sort 1Gb of data with 1Mb of RAM



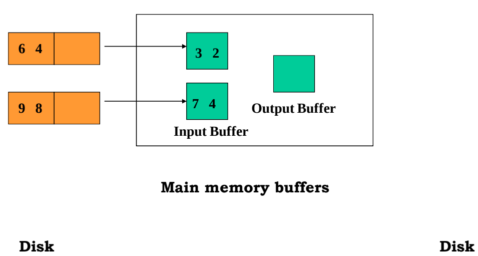
CS5208 7

A Simpler Problem: Combine Sorted Files



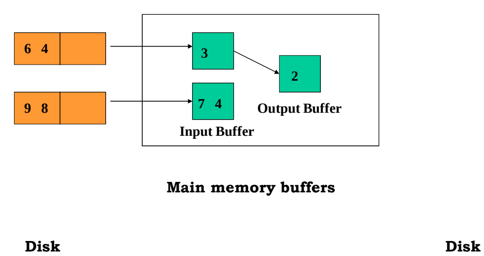
CS5208 8

A Simpler Problem: Combine Sorted Files



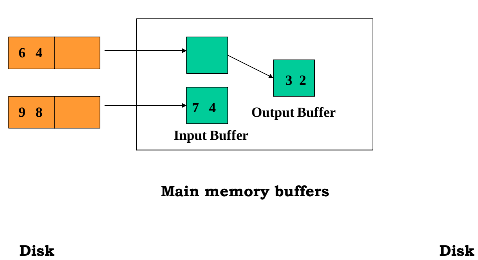
CS5208 9

A Simpler Problem: Combine Sorted Files



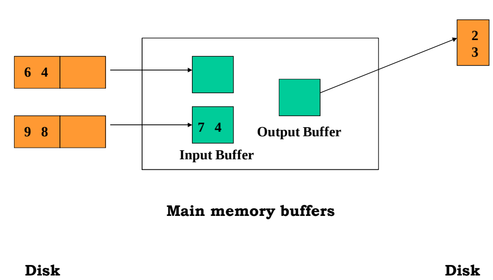
CS5208 10

A Simpler Problem: Combine Sorted Files



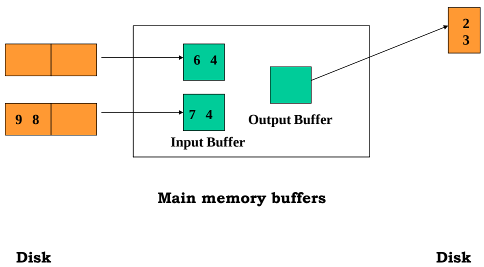
CS5208 11

A Simpler Problem: Combine Sorted Files

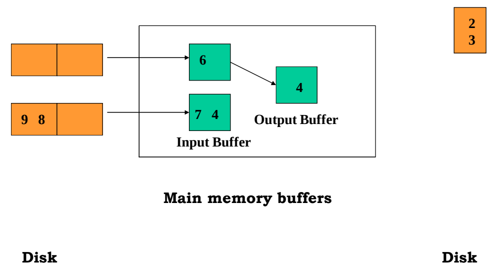


CS5208 12

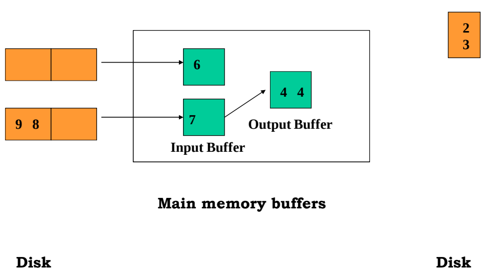
A Simpler Problem: Combine Sorted Files



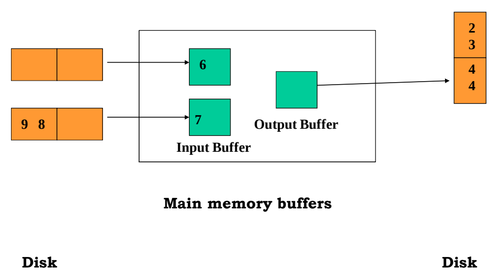
A Simpler Problem: Combine Sorted Files



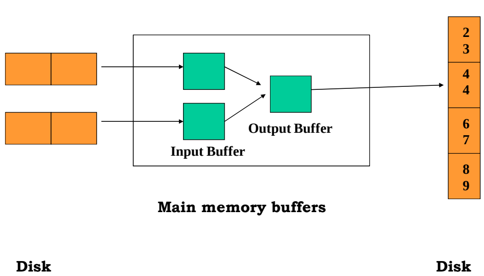
A Simpler Problem: Combine Sorted Files



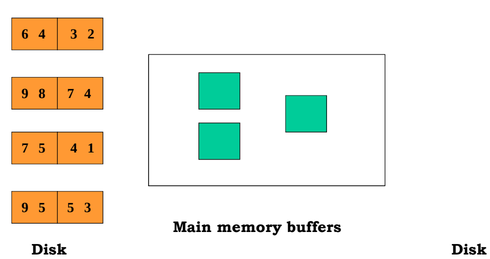
A Simpler Problem: Combine Sorted Files



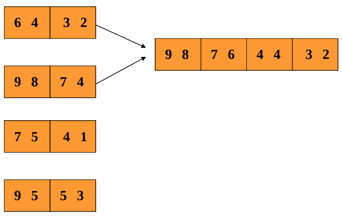
A Simpler Problem: Combine Sorted Files



What if there are many more runs?

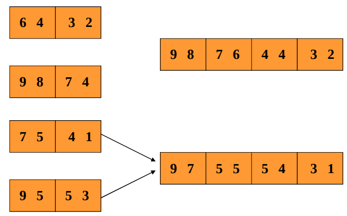


What if there are many more runs?



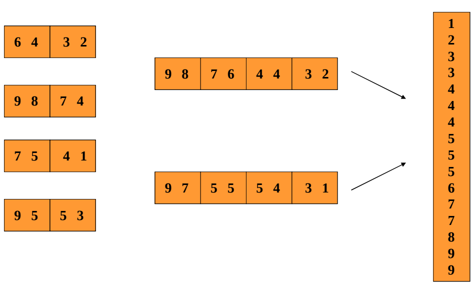
CS5208 19

What if there are many more runs?



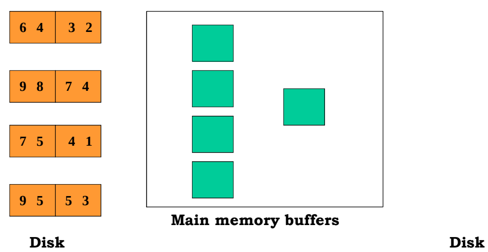
CS5208 20

What if there are many more runs?



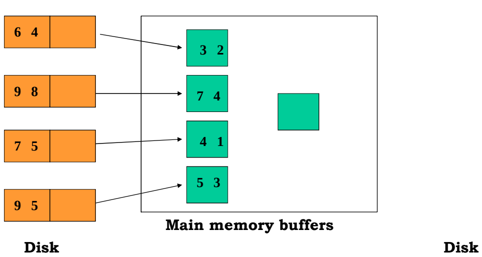
CS5208 21

What if there are more memory?



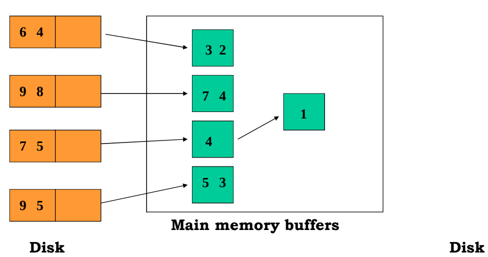
CS5208 22

What if there are more memory?



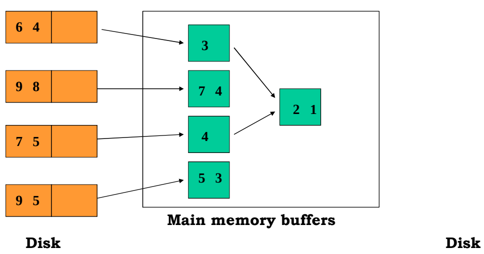
CS5208 23

What if there are more memory?

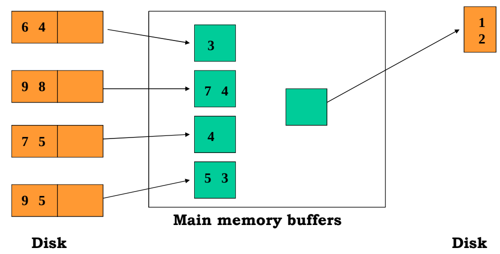


CS5208 24

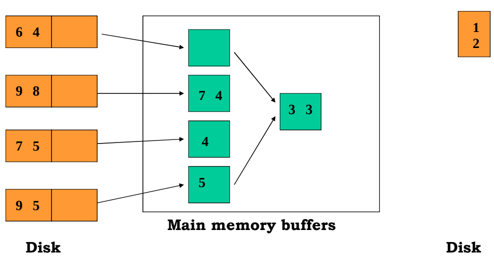
What if there are more memory?



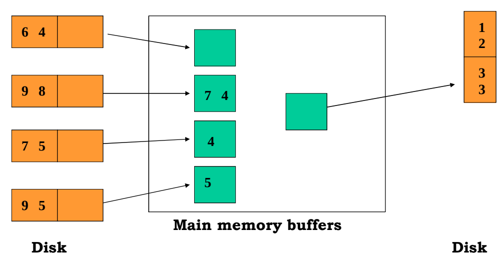
What if there are more memory?



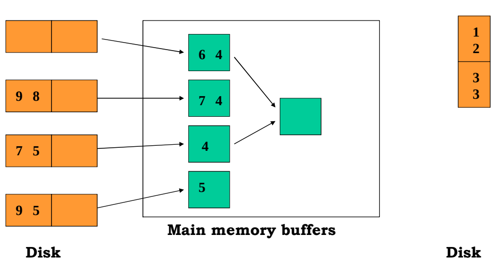
What if there are more memory?



What if there are more memory?



What if there are more memory?



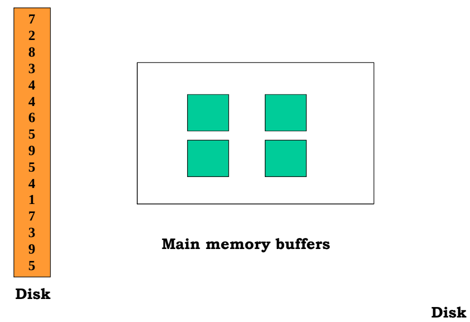
Multi-way Merge Sort

- Given *k* sorted files (runs), we can *merge* them into larger sorted runs, and eventually produce *one* single sorted file.
- To sort a very large file, we can do it in 2 steps
  - Generate sorted runs
  - Merge sorted runs (we already know how to do this)

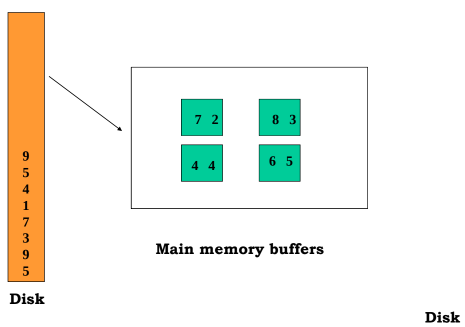
How to generate sorted runs?

- Read as many records as possible into memory
- Perform in-memory sort
- Write out sorted records as a sorted run
- Repeat the process until all records in the unsorted files are read

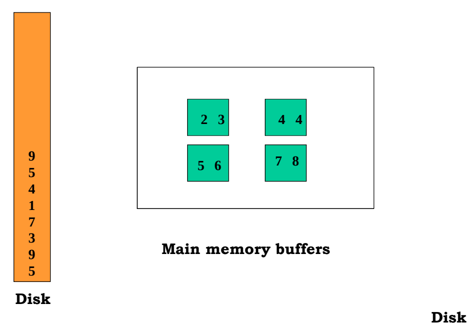
How to generate sorted runs?



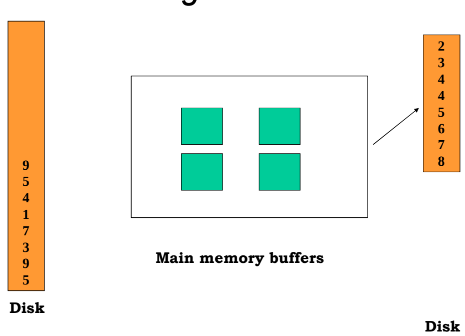
How to generate sorted runs?



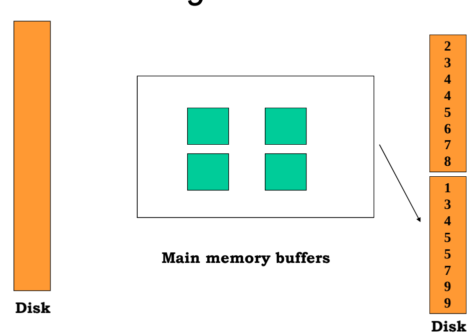
How to generate sorted runs?

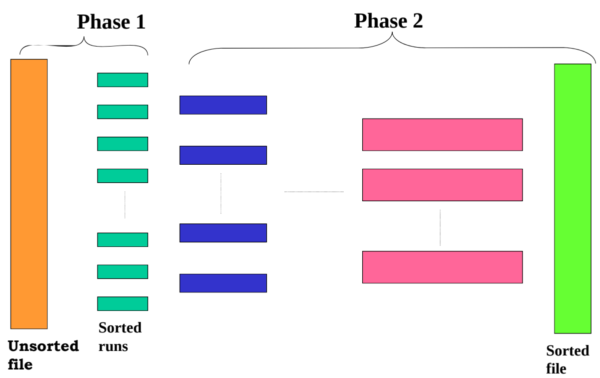


How to generate sorted runs?



How to generate sorted runs?





Multi-way Merge Sort

- To sort a file with *N* pages using *B* buffer pages:
  - Phase 1: use *B* buffer pages. Produce ⌈N / B⌉ sorted runs of *B* pages each.
    - 1 pass (read + write) over the file
  - Phase 2: merge *B-1* runs each time
    - ⌈ $\log_{B-1}$ ⌈N / B⌉ ⌉ passes

Cost of Multi-way Merge Sort

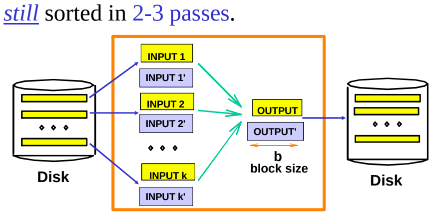
- Number of passes: 1 + ⌈ $\log_{B-1}$ ⌈N / B⌉ ⌉
- Cost = 2N * (# of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Phase 1 (pass 0): ⌈108 / 5⌉ = 22 sorted runs of 5 pages each (last run is only 3 pages)
  - Phase 2:
    - Pass 1: ⌈22 / 4⌉ = 6 sorted runs of 20 pages each (last run is only 8 pages)
    - Pass 2: 2 sorted runs, 80 pages and 28 pages
    - Pass 3: Sorted file of 108 pages

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

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

Internal Sort Algorithm

- Quicksort is a fast way to sort in memory.
- An alternative is ***replacement selection***

```
Read B blocks into memory
Output:  move smallest record, say s, to output buffer
Read in a new record r
if r > s, then GOTO Output
   else freeze r
if all records in memory are frozen, then all records that have
   been output constitute a run; unfreeze all records and start a
   new run
GOTO Output
```

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92
- suppose each block contains one record and B=5

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 34 68 45

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 45 → 34

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 45 → 34
60

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 ~~45~~ → 34
60 → 34 45

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 49 ~~34~~ 68 ~~45~~ → 34
60 → 34 45
2

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 ~~49~~ ~~34~~ 68 ~~45~~ ⟶ 34
60 ⟶ 34 45
2 ⟶ 34 45 49

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 ~~49~~ ~~34~~ 68 ~~45~~ ⟶ 34
~~60~~ ⟶ 34 45
2 ⟶ 34 45 49
38 ⟶ 34 45 49 60

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

109 ~~49~~ ~~34~~ ~~68~~ ~~45~~ ⟶ 34
~~60~~ ⟶ 34 45
2 ⟶ 34 45 49
38 ⟶ 34 45 49 60
28 ⟶ 34 45 49 60 68

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

~~109~~ ~~49~~ ~~34~~ ~~68~~ ~~45~~ ⟶ 34
~~60~~ ⟶ 34 45
2 ⟶ 34 45 49
38 ⟶ 34 45 49 60
28 ⟶ 34 45 49 60 68
47 ⟶ 34 45 49 60 68 109

Replacement Selection (Example)

- 109, 49, 34, 68, 45, 60, 2, 38, 28, 47, 16, 19, 35, 59, 98, 78, 76, 40, 35, 86, 10, 27, 61, 92

~~109~~ ~~49~~ ~~34~~ ~~68~~ ~~45~~ ⟶ 34
~~60~~ ⟶ 34 45
2 ⟶ 34 45 49
38 ⟶ 34 45 49 60
28 ⟶ 34 45 49 60 68
47 ⟶ 34 45 49 60 68 109
16 38 28 47 2 ⟶ start a new run

Replacement Selection (Cont.)

- Final results:
  - 34 45 49 60 68 109
  - 2 16 19 28 35 38 47 55 76 78 86 98
  - 10 27 35 40 61 92
- Would have been 5 runs using Quicksort

More on Replacement Selection

- Fact: average length of a run is *2B*
- Worst-Case
  - What is the 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!
  - Is replacement selection always better?

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.

Sequential vs Random I/Os

- Is minimizing passes *optimal*? Is merging as many runs as possible the best solution?
- Suppose we have 80 runs, each 80 pages long and we have 81 pages of buffer space.
- We can merge all 80 runs in a single pass
  - each page requires a seek to access (Why?)
  - there are 80 pages per run, so 80 seeks per run
  - total cost = 80 runs X 80 seeks = 6,400 seeks

Sequential vs Random I/Os (Cont)

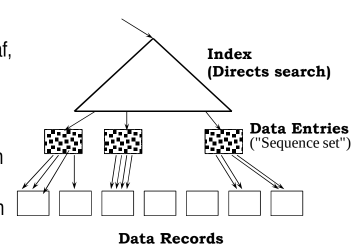
- We can merge all 80 runs in two steps
  - 5 sets of 16 runs each
    - read 80/16=5 pages of one run
    - 16 runs result in sorted run of 1280 pages
    - each merge requires 80/5X16 = 256 seeks
    - for 5 sets, we have 5X256 = 1280 seeks
  - merge 5 runs of 1280 pages
    - read 80/5=16 pages of one run => 1280/16=80 seeks in total
    - 5 runs => 5X80 = 400 seeks
  - total: 1280+400=1680 seeks!!!
- Number of passes increases, but number of seeks decreases!

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

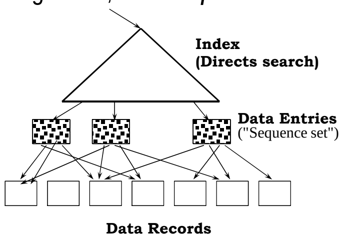
- Cost: root to the left-most leaf, then retrieve all leaf pages (<key,record> pair organization)
- If <key, rid> pair organization is used? Additional cost of retrieving data records: each page fetched just once.



Always better than external sorting!

Unclustered B+ Tree Used for Sorting

- each data entry contains <key,*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.***

Sorting Records!

- Sorting has become a blood sport!
  - Parallel sorting is the name of the game ...
- Datamation: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - World record: 3.5 ***seconds***
    - 12-CPU SGI machine, 96 disks, 2GB of RAM
- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?
  - Joule Sort: How many can you sort with 1 Joule?

Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted *runs* of size *B* (# buffer pages). Subsequent 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.
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.