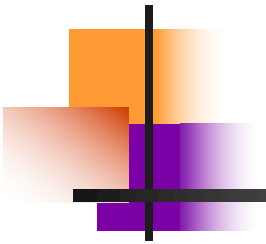
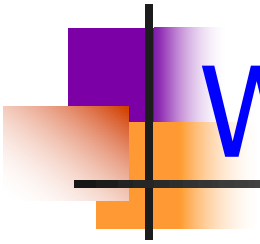


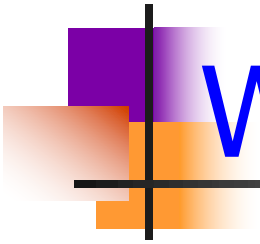
External Sorting

Chapter 13



Why Sort?

- User wants query answers in some order
  - E.g., decreasing order of age
- First step to bulk-loading B+ Tree
- Eliminate duplicate records
  - SELECT DISTINCT
- Sort-merge join algorithm (later)



Why bother?

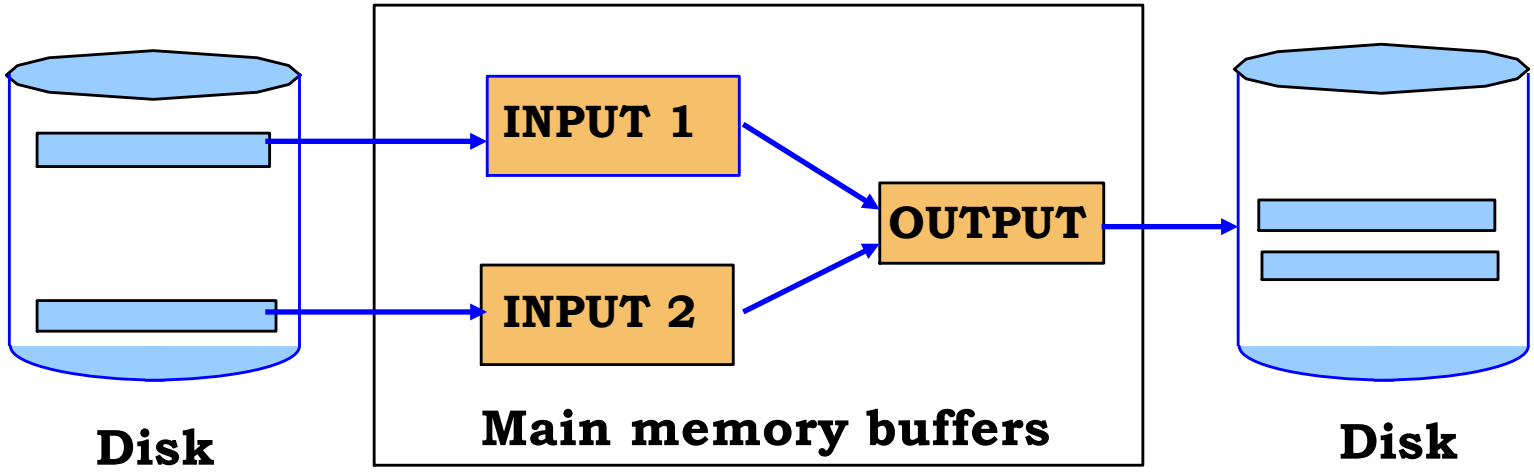
- But we already know how to sort…
- New Problem: How to sort 100 GB of data with 1 GB RAM?
  - External Sort – Minimize disk access cost
  - Why not use virtual memory?

Sorting Records! Sortbenchmark.org

- Sorting has become a blood sport!
- Results from 2014:
- Gray Sort: How many bytes/min, while sorting at least 100TB?
  - Daytona: TritonSort/UCSD (100TB in 1378 sec), Apache Spark (100TB in 1,406 sec)
  - Indy: BaiduSort (100TB in 716 sec)
- Minute Sort: How many records can you sort in 1 minute?
  - Daytona: DeepSort (3.7 TB/min)
  - Indy: BaiduSort (7 TB/min)
- Penny Sort: How many can you sort for a penny? (since 2011)
  - Daytona: Psort/Univ of Padova (286GB)
  - Indy: Psort/Univ of Padova (334GB)

Two-Way External Merge Sort

- Pedagogical use only!
- Requires 3 Buffer pages
- Pass 1: Read a page, sort it, write it (a run).
  - only one buffer page is used
- Pass 2, 3, …, etc.:
  - three buffer pages used.



Two-Way External Merge Sort

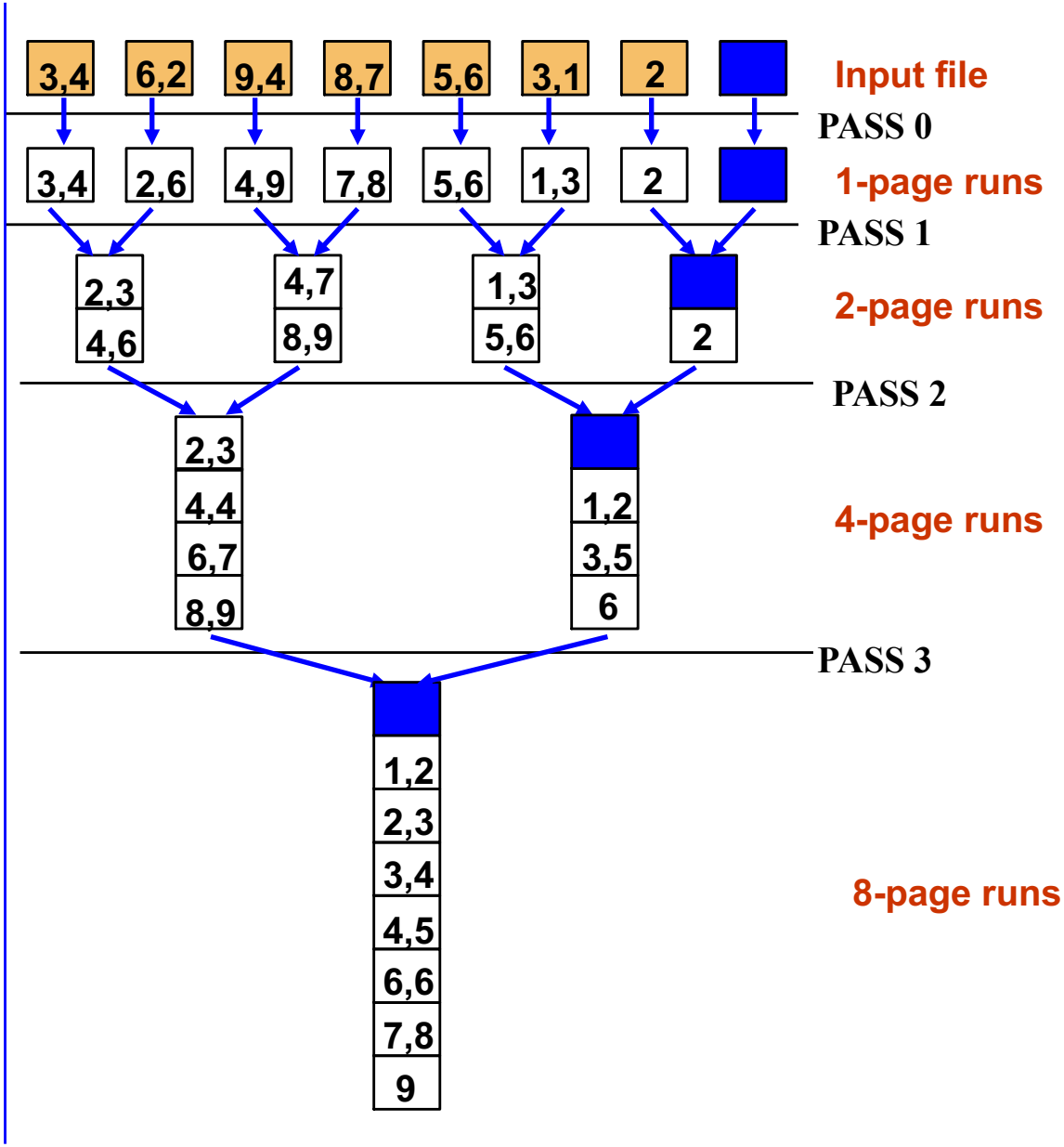
- Read & write entire file in each pass
- **Divide and conquer**
- #of passes:

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

- Cost (# of page I/Os):

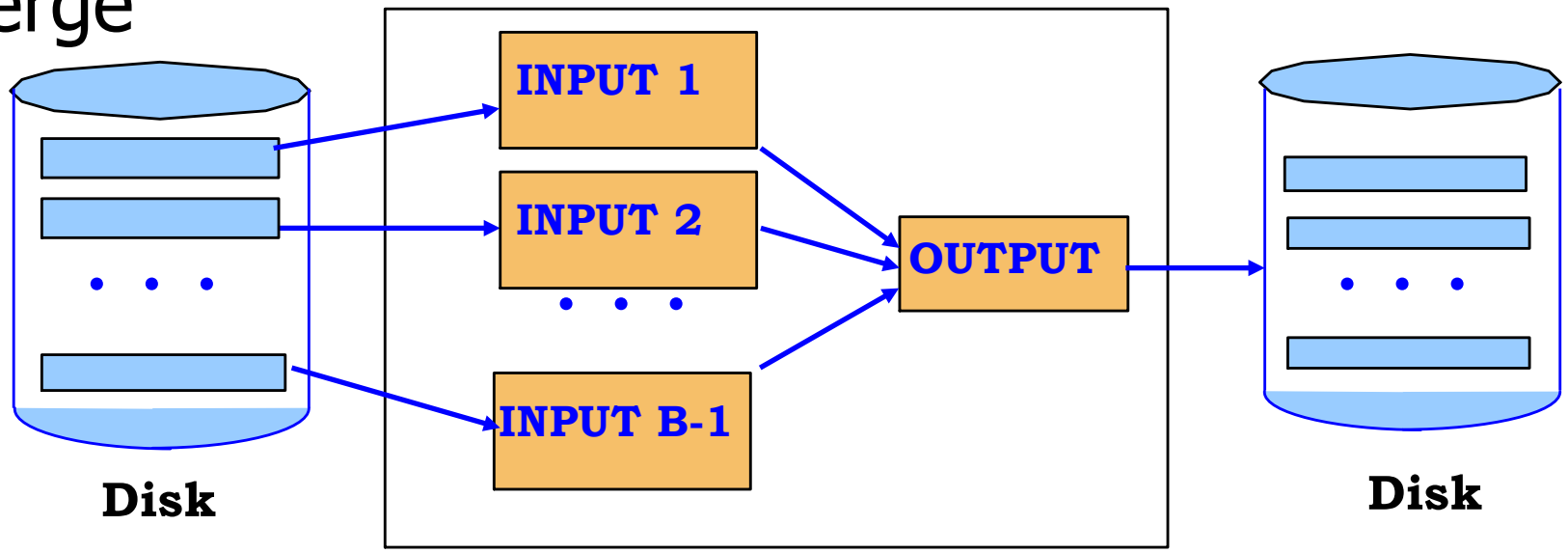
$$2N(1+\lceil \log_2 N \rceil)$$

- In this example:
  - 56 pages



General External Merge Sort

- Sort a file with *N* pages using *B* buffer pages:
  - Pass 0: use *B* buffer pages and sort them internally, producing $\lceil N/B \rceil$ sorted runs (each B-page long)
  - Pass 2, 3, …: merge ***B-1*** runs, using (B-1)-way merge



B-1 way merge.
Total buffer pages: B

Cost of External Merge Sort

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

  - Cost = 2N * (# of passes)
  - With 11 buffer pages, how many page I/Os are needed to sort a 1000-page file?
    - Pass 0: ⌈ (1000/11) ⌉ = 91 runs of 11 pages each (Note: the last run is 10 pages)
    - Pass 1: ⌈ 91/10 ⌉ = 10 sorted runs of 110 pages each (last run is only 10 pages)
    - Pass 2: ⌈ 10/10 ⌉ = 1 sorted run of 1000 pages.

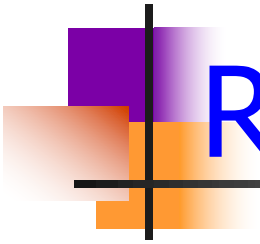
    Sorted File in 3 passes.
    Total I/O: 2000 pages/pass or 6000 pages.

Number of Passes of External Sort

N (# of pages)	B=3	B=17	B=257
100	7	2	1
10,000	13	4	2
1,000,000	20	5	3
10,000,000	23	6	3
100,000,000	26	7	4
1,000,000,000	30	8	4

32K pg size, 32TB relation

@1ms per read, 1111 hours = **46 days!**



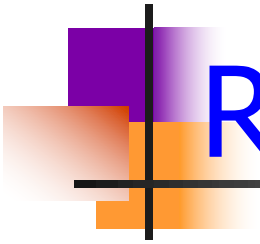
Reducing number of initial runs

- In the basic scheme, with B memory pages, you will produce approx. N/B runs initially because you will sort B pages, creating a run.
- The eventual sorting cost is lower if we have fewer runs
- Can we produce fewer runs?
- Yes!: Replacement sort can produce runs longer than B pages on the average!
  - Longer runs may mean fewer passes (less I/O)

Replacement Sort (for Pass 0 only)

- One page each is dedicated as an input buffer and an output buffer
- Remaining B-2 (call it M) pages are called the **current set**

Input buffer	Current Set	Output buffer
13 5	2 7 10 20 30 40	3 6 …



Replacement Sort (for Pass 0 only)

- Start by reading a page from file into the input buffer. Copy records from input buffer to current set
- Repeatedly pick smallest value from current set that is greater than largest value in output buffer
  - Write to output buffer (run). If buffer full, output
- Start a new run when no value in current set is larger than all values in output
- On average, produces runs of size 2M, i.e., 2*(B-2) pages.

Internal Sort Algorithm: Replacement Sort

Example: M = 2 pages, 2 tuples per page. Input Sequence: 10, 20, 30, 40, 25, 35, 9, 8, 7, 6, 5, …

	Current set: 10, 20, 30, 40
Read 25, Output 10.	Current set: 20, 25, 30, 40
Read 35, Output 20.	Current set: 25, 30, 35, 40
Read 9, Output 25.	Current set: 9, 30, 35, 40
Read 8, Output 30.	Current set: 8, 9, 35, 40
Read 7, Output 35.	Current set: 7, 8, 9, 40
Read 6, Output 40.	Current set: 6, 7, 8, 9
Read 5, Flush output, Start new run.	In-memory …
On Disk: 10, 20, 25, 30, 35, 40	

- Average length of a run in replacement sort is *2M Pages.*

Blocked I/Os

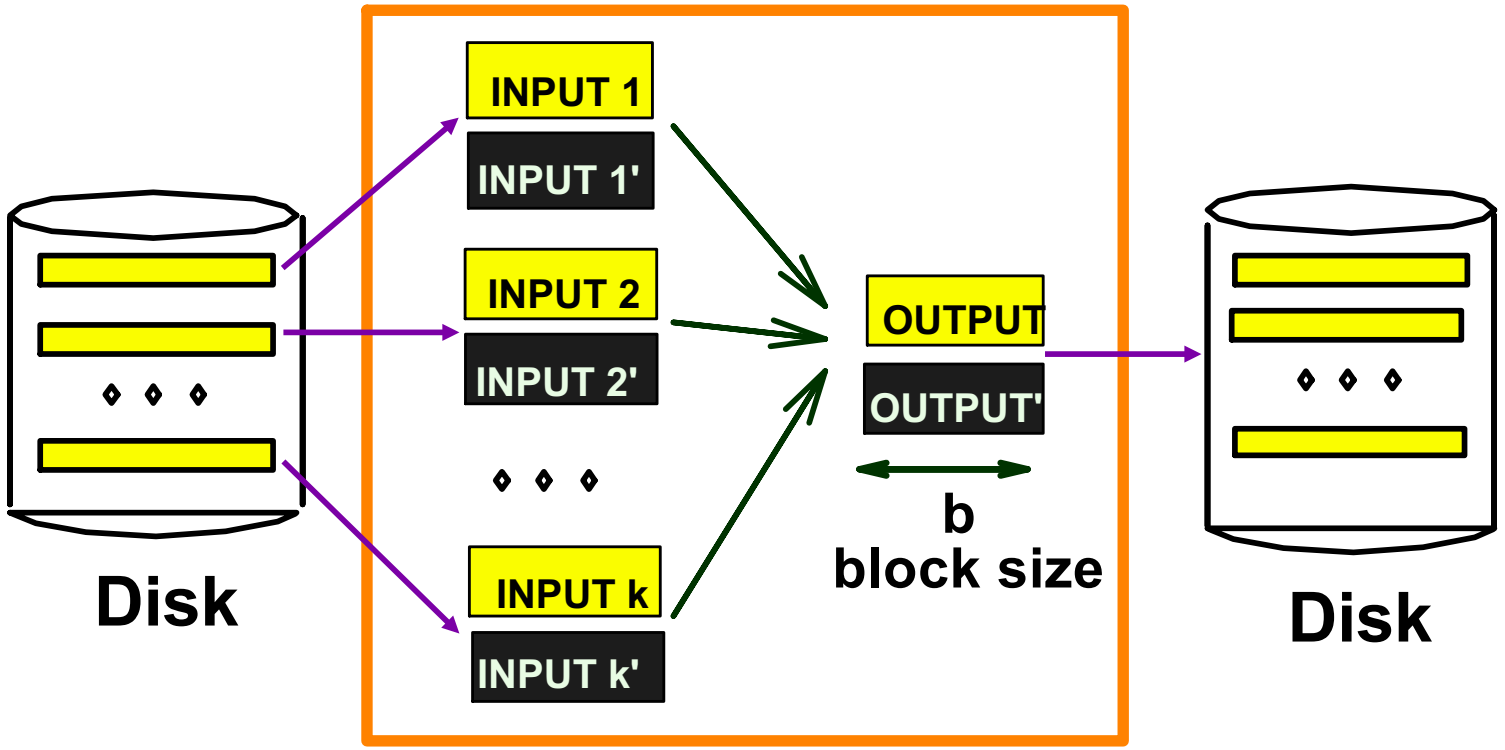
- Single request to read a *block* of pages often cheaper than independent requests for each page – *Why?*
- Make each buffer a block of q pages instead
  - Reduces cost per page I/O
  - First Pass: Each run **2(B-2)** pages, **$\lceil N/2(B-2) \rceil$** runs (where **B** is the size of the buffer pool in #pages)
    - *Assuming we use replacement sort optimization…*
  - # of runs merged in each pass (fanout): **$F = \lfloor (B-q)/q \rfloor$**
  - # passes: **$\lceil \log_F(\text{\# of runs from first pass}) \rceil + 1$**
  - Cons: The fanout is lower and thus # of passes could increase,
  - Pros: Each pass doing more efficient I/O.

  - **This could be cheaper or more expensive – need to do the math**
  - **In practice, often 2-3 passes are sufficient.**

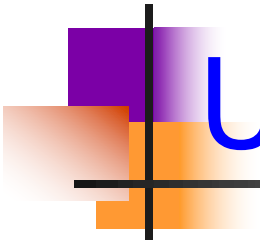
Double Buffering

Reduces response time.
What about throughput?
- Not much difference.

- Overlap CPU and IO processing
- *Prefetch* into shadow block.
  - Potentially, more passes; in practice, 2-3 passes.



B main memory buffers, k-way merge

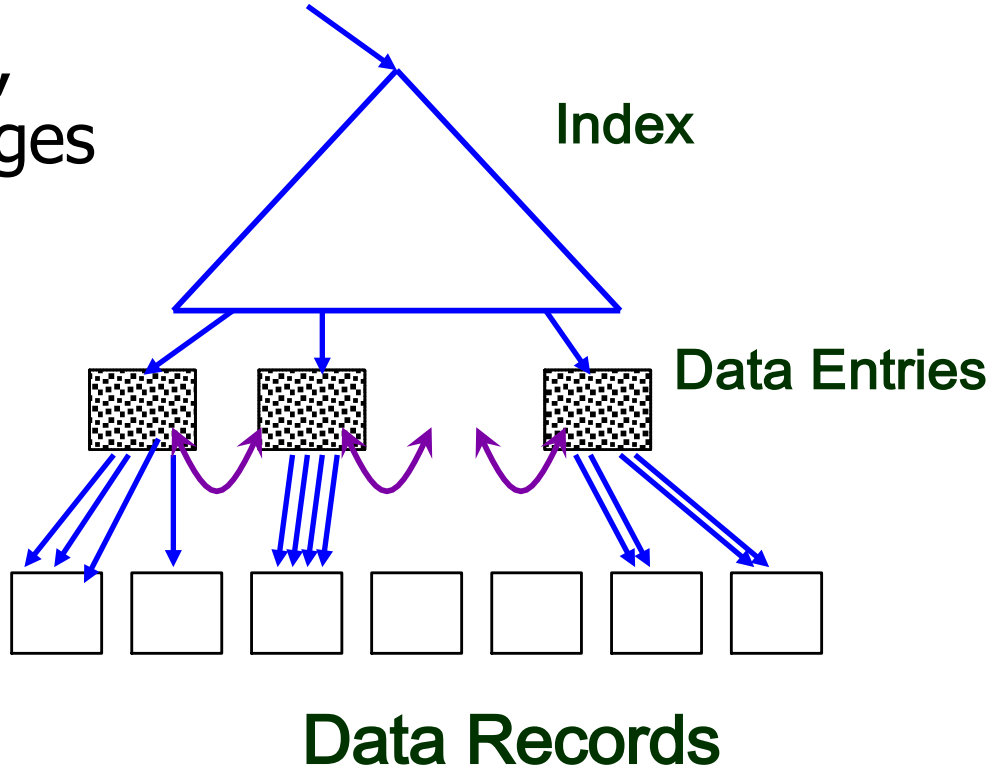


Using B+ Trees for Sorting

- 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

- Go to the left-most leaf, then retrieve all leaf pages
- Alt 1: Done!
  - # pages?
- Alt 2: Retrieving data records, each page fetched just once



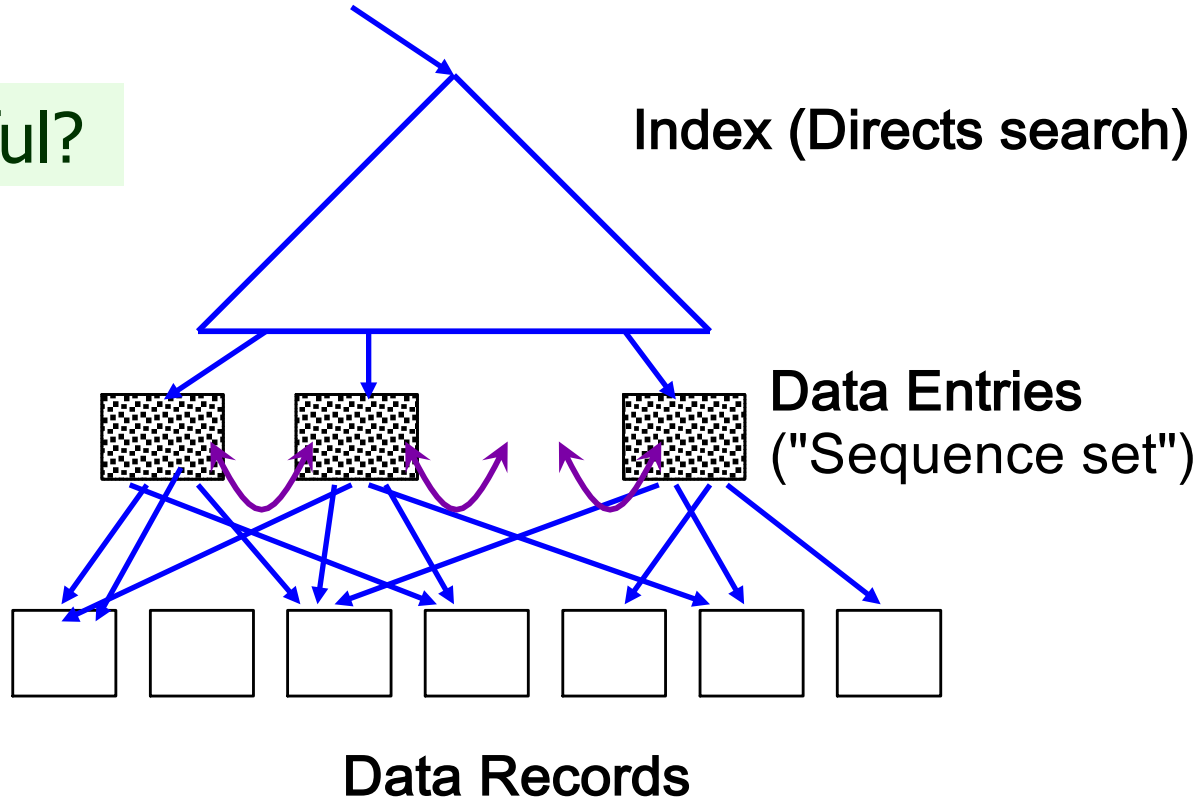
➢ **Faster than external sorting!**

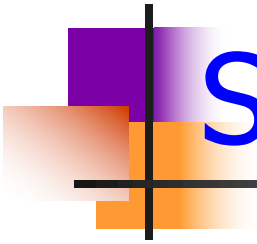
Why not scan the data file directly?

Unclustered B+ Tree Used for Sorting

- Alternative (2): In general, one I/O per data record!

When can this be useful?





Summary, External Sort

- Important operation
- Minimize disk I/O cost, use the (large) buffer pool:
  - Larger runs
  - Fewer merges
  - Blocked IOs
  - Double Buffering
- Choice of internal sort algorithm may matter
  - Pass 0: Run size B or 2B
- Can use indices
  - Clustered Index: Great! Always better than external sort
  - Unclustered Index: Use with caution