

CSE 132C

Database System Implementation

Arun Kumar

Topic 3: External Sorting

Chapter 13 of Cow Book

Slide ACKs: Jignesh Patel, Paris Koutris

External Sorting: Outline

- ❖ **Overview and Warm-up** 
- ❖ **Multi-way External Merge Sort (EMS)**
- ❖ **EMS and B+ trees**

Motivation for Sorting

- ❖ User's SQL query has ORDER BY clause!
- ❖ First step of bulk loading of a B+ tree index
- ❖ Used in implementations of many relational ops: project, join, set ops, group by aggregate, etc. (next topic!)

Q: But sorting is well-known; why should a DBMS bother?

Often, the file (relation) to be sorted will not fit in RAM!

“External” Sorting

External Sorting: Overview

- ❖ **Goal:** Given relation **R** with N pages, **SortKey** A , M buffer pages (often, $M \ll N$), sort **R** on A to get sorted **R'**
- ❖ **Idea:** Sorting algorithm should be disk page I/O-aware!
- ❖ **Desiderata:**
 - ❖ High efficiency, i.e., low I/O cost, even for very large N
 - ❖ Use sequential I/Os rather than random I/Os AMAP
 - ❖ Interleave I/O and comp. (DMA); reduce CPU cost too

NB: I/O-aware external sorting is also a key part of the implementation of MapReduce systems (Hadoop, Spark, etc.)!

Warm-up: 2-way External Merge Sort

Idea: Make Merge Sort I/O-aware!

1. **Sort phase:** Read each page into buffer memory; do “internal” sort (use any popular fast sorting algorithm, e.g., quicksort); write it back to disk (a sorted “**run**”)
2. **Merge phase:** Read 2 runs, merge them on the fly, write out a new *double-length* run; recurse till whole file is a run!

NB: Sort phase is 1-pass; merge phase is often multi-pass!

Warm-up: 2-way External Merge Sort

Number of passes:

Sort phase: 1

Merge phase: $\lceil \log_2(N) \rceil$

Each pass does 1 read and
1 write of whole file:

$2N$ page I/Os per pass

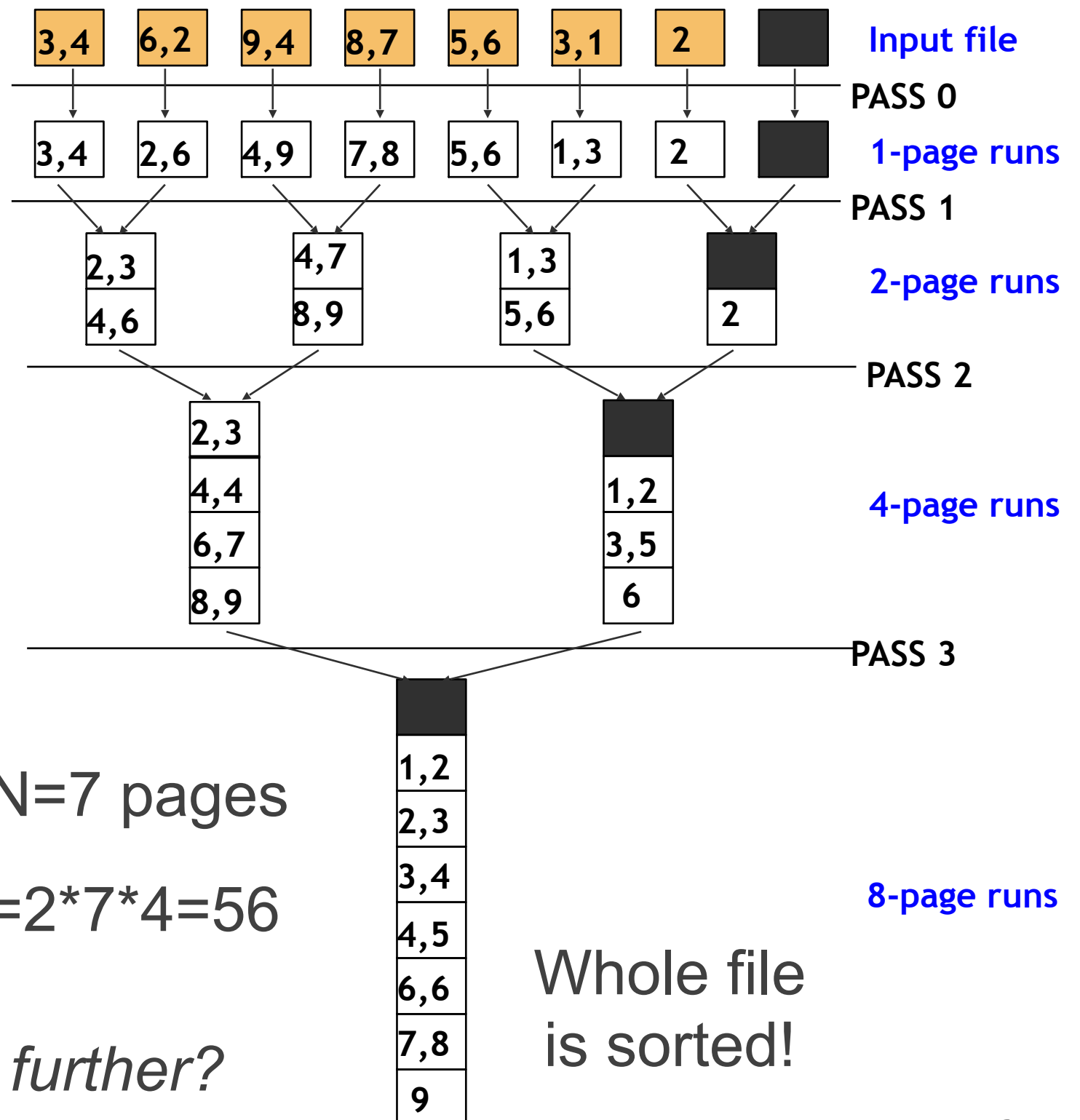
I/O cost of 2-way EMS:

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

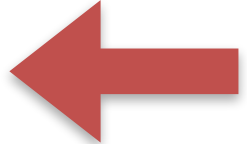
$N=7$ pages

$$=2*7*4=56$$

Q: How to reduce this cost further?



External Sorting: Outline

- ❖ Overview and Warm-up
- ❖ Multi-way External Merge Sort (EMS) 
- ❖ EMS and B+ trees

Multi-way EMS: Motivation

Q: How many buffer pages does 2-way EMS use?

Sort phase: 1 or 2 (1 each for R and W / 1 for in-place sort)

Merge phase: 3 (1 for each run input; 1 for merged output)

So, 2-way EMS uses only 3 buffer pages!

Idea: Why not exploit more buffer pages (say, $B \gg 3$)?

Sort phase:

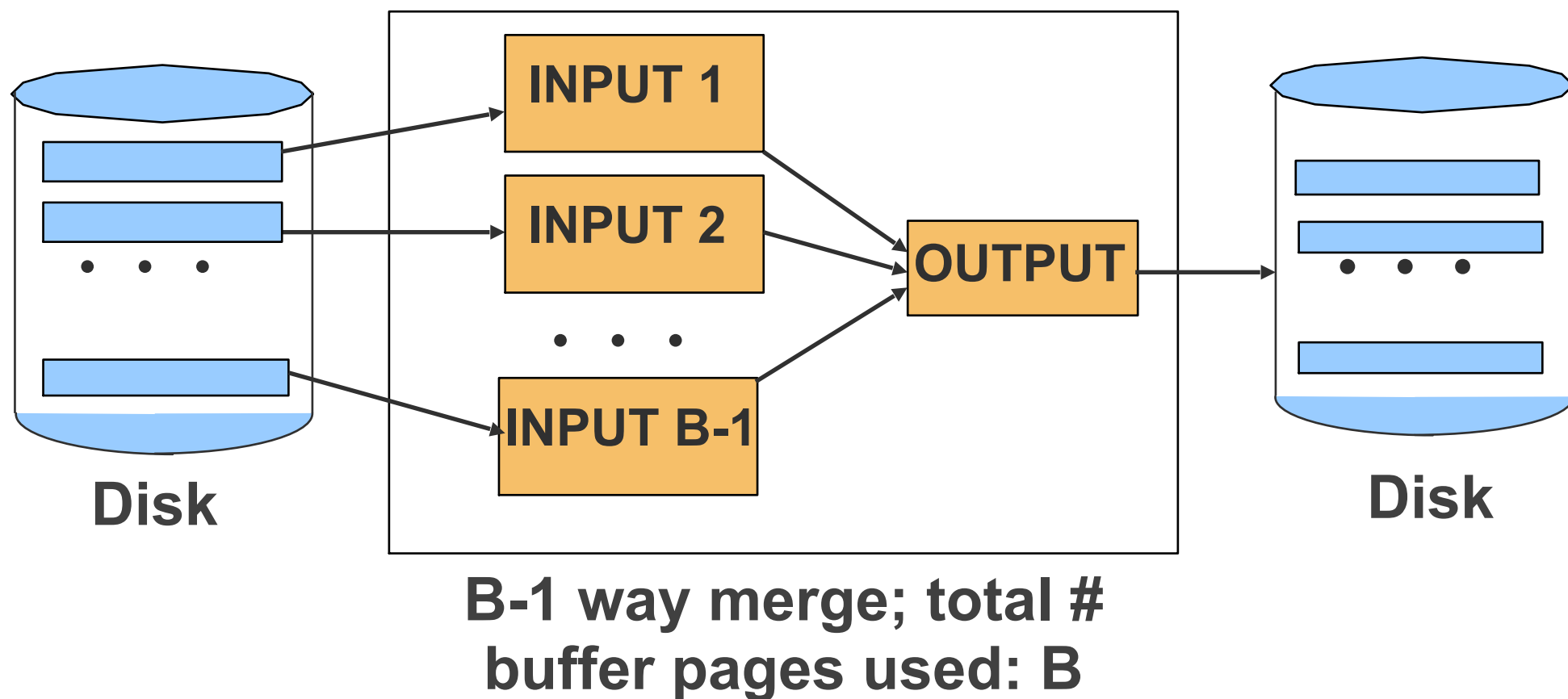
Read B pages at a time (not just 1 at a time)!

Write out $\lceil N/B \rceil$ sorted runs of length B each (not just 1)

But I/O cost of sort phase is still the same!

Multi-way EMS: B-way Merge Phase

Idea: Merge B-1 sorted runs at a time instead of merging just 2 sorted runs at a time as in 2-way EMS



passes for Merge Phase reduces to: $\lceil \log_{B-1}(\lceil N/B \rceil) \rceil$

Multi-way EMS I/O Cost

$$\text{Overall, \# passes} = 1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil$$

$$\text{I/O cost per pass} = 2N$$

$$\text{Total I/O cost of EMS} = 2N(1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil)$$

Example: File with 100M records of length 0.5KB each; page size is 8KB; number of buffer pages for EMS $B=1000$

$$\text{Number of pages } N = 100M * 0.5KB / 8KB = 6.25M$$

$$\begin{aligned} \text{Total I/O cost of EMS} &= 2 \times 6.25M \times (1 + \lceil \log_{999}(6250) \rceil) \\ &= 2 \times 6.25M \times (1 + 2) \\ &= 37.5M \end{aligned}$$

Only need the ceil!

Multi-way EMS I/O Cost

$$\text{Total number of passes} = 1 + \lceil \log_{B-1}(\lceil N/B \rceil) \rceil$$

With 8KB page, 800MB

N	Naive 2-way EMS	B=1K	B=10K	B=100K
1M	21	3	2	2
10M	25	3	2	2
100M	28	3	3	2
1B	31	4	3	2

With 8KB page, 8TB!

Only 2 passes to sort up to ~79.9TB!
(2 is the lower bound for EMS!)

Multi-way EMS: Improvements

- ❖ While already efficient, some key algorithmic+systems-oriented improvements have been made to multi-way EMS to reduce *overall runtime* (not just counting I/O cost)
- ❖ Three prominent improvements:
 1. Replacement sort (aka heap sort) as internal sort
 2. “Blocked” I/O
 3. Double Buffering

Improvement 1: Replacement Sort

- ❖ In standard EMS, quick sort used during Sort Phase
- ❖ Produces $\lceil N/B \rceil$ runs of length B pages each
- ❖ Replacement sort is an alternative for Sort Phase
- ❖ Produces runs of *average* length 2B pages each
- ❖ So, number of runs reduced on average to $\lceil N/2B \rceil$
- ❖ Maintains a sorted heap in B-2 pages; 1 page for reading; 1 for sorted output
- ❖ Slightly higher CPU cost; but significantly lower I/O cost

(We are skipping the details of this algorithm)

$$\text{New total I/O cost} = 2N(1 + \lceil \log_{B-1}(\lceil N/2B \rceil) \rceil)$$

Improvement 2: “Blocked” I/O

- ❖ Merge Phase did *not* recognize distinction between sequential I/O and random I/O!
- ❖ Time difference *not* fully reflected in just counting I/O cost
- ❖ **Idea:** Read a “block” of b pages of each run at a time!
- ❖ So, only $\lfloor B/b \rfloor - 1$ runs can be merged at a time
- ❖ b controls trade-off of # passes vs time-per-pass

$$\text{“Fan-in” of Merge Phase} = F = \lfloor B/b \rfloor - 1$$

$$\text{New total I/O cost} = 2N(1 + \lceil \log_F(N') \rceil)$$

$$N' = \lceil N/B \rceil \quad \text{or} \quad N' = \lceil N/2B \rceil$$

Improvement 3: Double Buffering

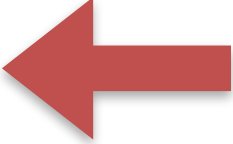
- ❖ Most machines have DMA; enables I/O-CPU parallelism
- ❖ Trivially feasible to exploit DMA in the Sort Phase
- ❖ But in the Merge Phase, CPU blocked by I/O for runs
- ❖ **Idea:** Allocate double the buffers for each run; while CPU processes one set, read pages (I/O) into other set!
- ❖ So, only $\lfloor B/2b \rfloor - 1$ runs can be merged at a time

New fan-in of Merge Phase = $F = \lfloor B/2b \rfloor - 1$

New total I/O cost = $2N(1 + \lceil \log_F(N') \rceil)$

$N' = \lceil N/B \rceil$ or $N' = \lceil N/2B \rceil$

External Sorting: Outline

- ❖ **Overview and Warm-up**
- ❖ **Multi-way External Merge Sort (EMS)**
- ❖ **EMS and B+ trees** 

Using B+ Tree for EMS

- ❖ Suppose we already have a B+ tree index with the SortKey being equal to (or a prefix of) the IndexKey
- ❖ Data entries of the B+ tree are already in sorted order!

Q: Is it a “good” idea to simply read the leaf level of the B+ tree to achieve the EMS then?

It depends!

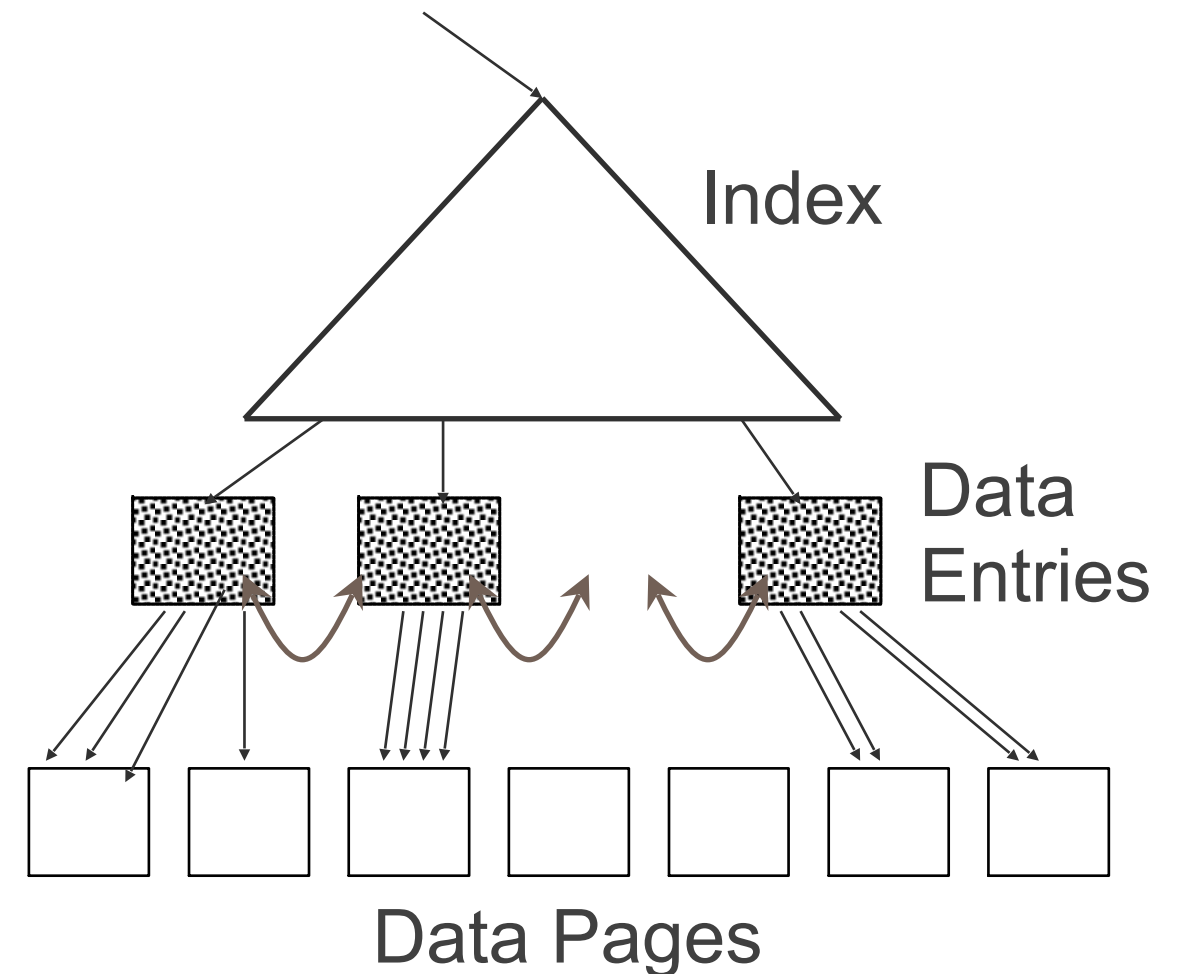
On whether the index is clustered or not

Good idea!

Might be really bad!

Using Clustered B+ Tree for EMS

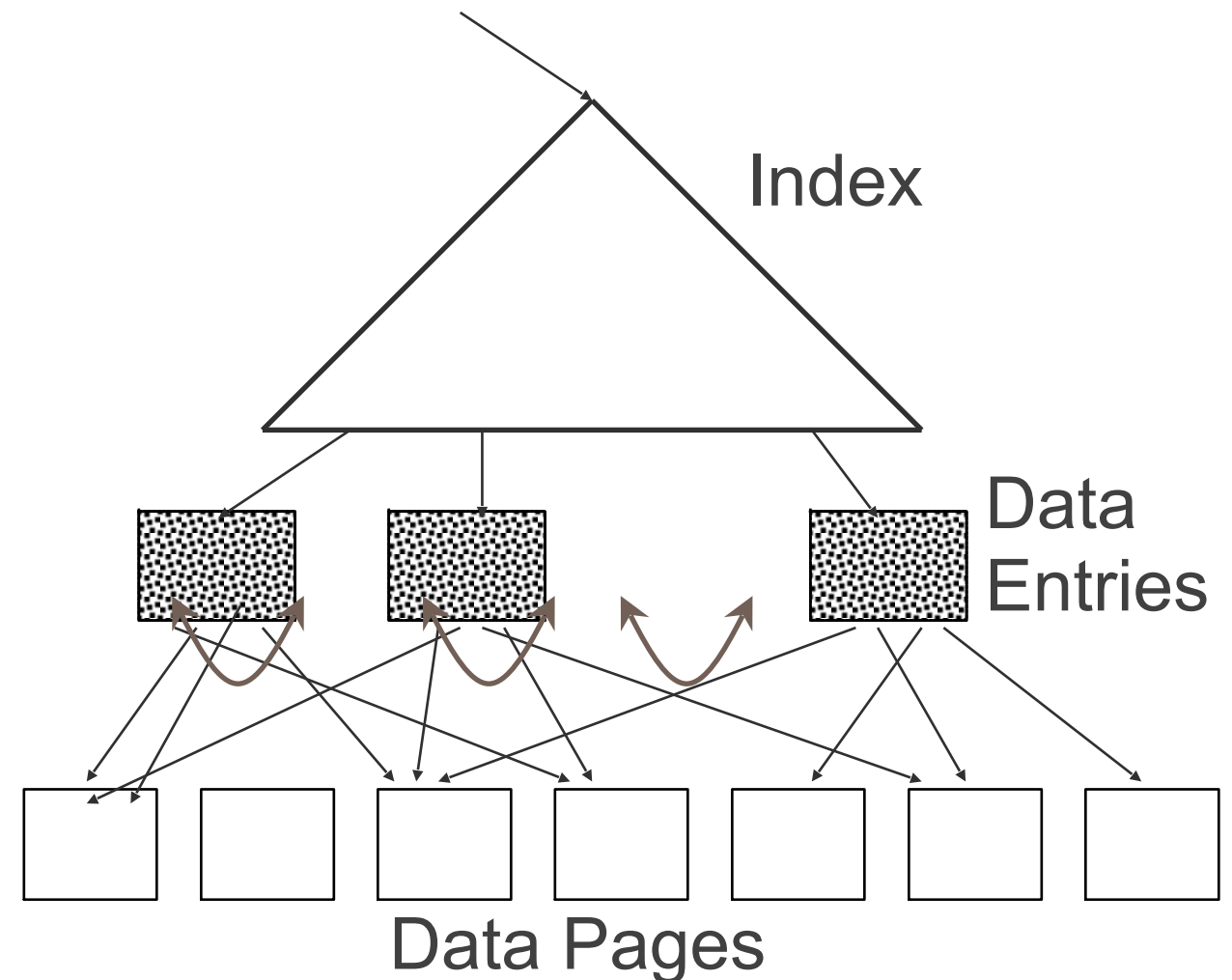
- ❖ Go down the tree to reach left-most leaf
- ❖ Scan leaf pages (data entries) left to right
- ❖ If AltRecord, done! O/W, retrieve data pages pointed to by successive data entries



- ❖ I/O cost if AltRecord: height + # leaf pages
- ❖ I/O cost otherwise: height + # leaf pages + # data pages
- ❖ Either way, I/O cost often $\ll$ from-scratch EMS!

Using Unclustered B+ Tree for EMS

- ❖ Unclustered means not AltRecord! Why?
- ❖ Same procedure as for clustered B+ tree
- ❖ Same I/O “cost” as for clustered tree with AltRID/AltRIDlist but many back-to-back *random* I/Os; thrashing!



Q: But when is this faster than from-scratch EMS?

- ❖ Usually, much slower than from-scratch EMS!

External Sorting as Competitive Sport!

The geekiest “sport” in the world: sortbenchmark.org

	Daytona	Indy
Gray	2016, 44.8 TB/min Tencent Sort 100 TB in 134 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub	2016, 60.7 TB/min Tencent Sort 100 TB in 98.8 Seconds 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub
Cloud	2016, \$1.44 / TB NADSort 100 TB for \$144 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk) Qian Wang, Rong Gu, Yihua Huang Nanjing University Reynold Xin Databricks Inc. Wei Wu, Jun Song, Junluan Xia Alibaba Group Inc.	2016, \$1.44 / TB NADSort 100 TB for \$144 394 Alibaba Cloud ECS ecs.n1.large nodes x (Haswell E5-2680 v3, 8 GB memory, 40GB Ultra Cloud Disk, 4x 135GB SSD Cloud Disk) Qian Wang, Rong Gu, Yihua Huang Nanjing University Reynold Xin Databricks Inc. Wei Wu, Jun Song, Junluan Xia Alibaba Group Inc.
Minute	2016, 37 TB Tencent Sort 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub	2016, 55 TB Tencent Sort 512 nodes x (2 OpenPOWER 10-core POWER8 2.926 GHz, 512 GB memory, 4x Huawei ES3600P V3 1.2TB NVMe SSD, 100Gb Mellanox ConnectX4-EN) Jie Jiang, Lixiong Zheng, Junfeng Pu, Xiong Cheng, Chongqing Zhao Tencent Corporation Mark R. Nutter, Jeremy D. Schaub
	2013, 168,242 Joules	2013, 168,242 Joules

External Sorting as Competitive Sport!

People are still at it!



Ray breaks the \$1/TB barrier as the world's most cost-efficient sorting system

CloudSort Benchmark

Cost to sort 100 TeraBytes of data

2016
Record

\$1.44 per TB

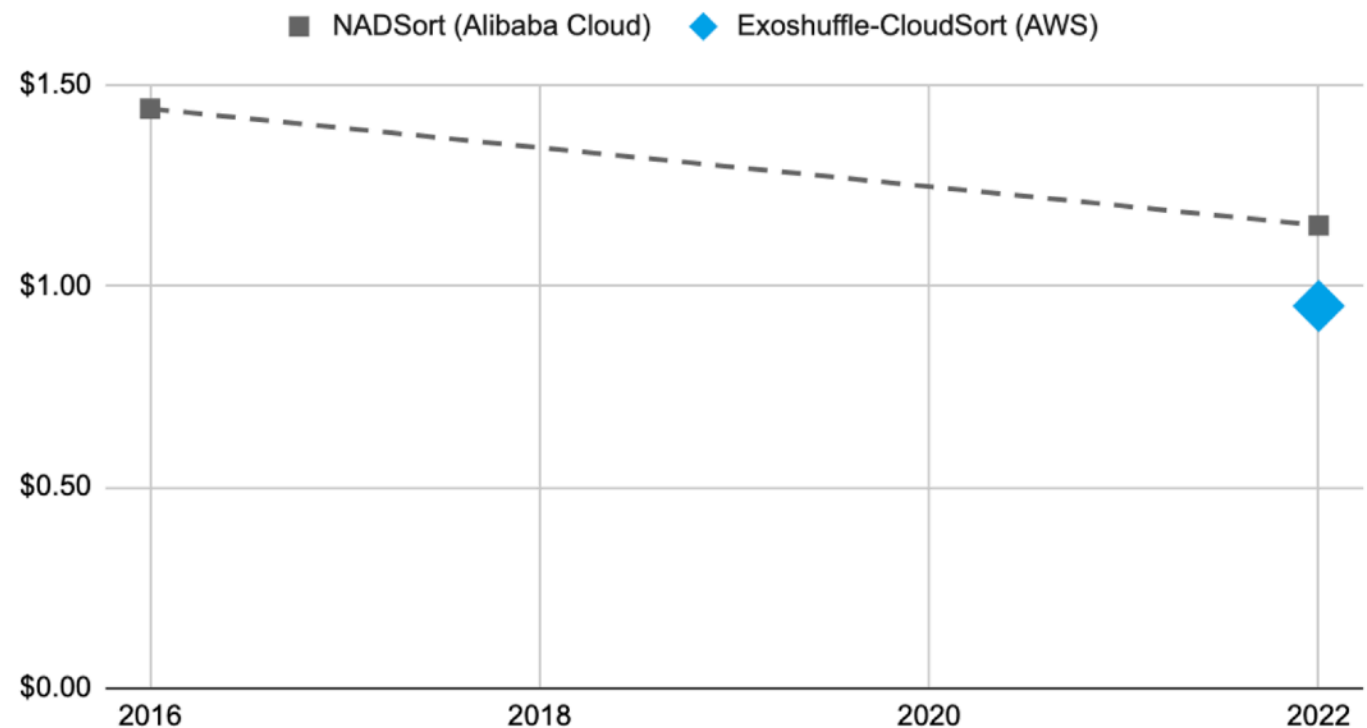
APACHE
Spark
on Alibaba Cloud

2022
Record

\$0.97 per TB

RAY on aws

CloudSort Cost over Years (\$/TB)



Review Questions

1. Briefly explain 2 use cases for external sorting in an RDBMS.
2. Which classical internal sorting algorithm does EMS build on?
3. Why bother optimizing EMS when its time complexity is already at the theoretical lower bound of $O(N \log(N))$?
4. Which EMS optimization(s) can actually raise the total I/O cost?
5. Given a relation of size 100GB, 8GB buffer memory, and 8KB page size, what is the I/O cost (in GB) for sorting this relation?
6. Why do companies/labs/academics still compete in the sorting benchmark competition? :)