CS186 Vitamin 1 Solutions

# External Hashing and Sorting

## **Q1: How many passes are required to fully sort a 8192 MB file with external merge sort?** ( +2 points for correct )

We’ll start by figuring out how many pages our buffer contains and how many pages the file contains.

We perform the following calculation to find the number of buffer pages we have allocated:

B = # buffer pages = 32 MB \* (1024 KB / 1 MB) \* (1 page / 128 KB) = 256 pages.

We perform the following calculation to find the number of pages in the file:

N = # pages in file = 8192 MB \* (1024 KB / 1 MB) \* (1 page / 128 KB) = 65,536 pages.

Let’s use the formula for number of passes in an external merge sort: . Plugging in N = 65536 pages and B = 256 pages, we see that **3 passes are required**.

## **Q2: What's the I/O cost of fully sorting a 8192 MB file with external merge sort?** ( +2 points for correct )

The I/O cost follows directly from the number of passes it takes to sort the file. In particular, it will take 2 \* (# of pages in the file) \* (# of passes).

Hence, we get 2 \* 65536 \* 3 = **393,216 I/Os**

## **Q3: Suppose we double the size of our buffer, to 64 MB. What is the largest file size (in MB) that we can externally sort in two passes?** ( +2 points for correct )

In this scenario, B = 512. B\*(B-1) = 512 \* 511 = 261,632 pages = **32,704 MB**.

## **Q4: Generalizing Q3, if we double the size of our buffer, approximately how much larger of a file can we externally sort in k passes?** ( +2 points for correct )

The largest file we can sort in 2 passes is B\*(B-1). The largest file we can sort in *k* passes is B\*(B-1)k-1 (can you reason why?). Thus, doubling B will increase the max file size for *k* passes by a factor of **2k**.

## **Q5: You decide to separate your dataset with external hashing. How does the I/O cost of externally hashing the file compare with the I/O cost of externally merge sorting the file?** ( +2 points for correct )

Assume that the data is uniformly distributed on the hashed key and that your hashing function distributes the records into partitions evenly.

Given the conditions presented for external hashing, in particular the perfectly distributed hash, we see that it will take 3 passes to externally hash the whole file. This results in **the same I/O cost**.

To verify this, let’s step through this pass by pass. After pass one, we have 255 partitions, each at least 257 pages large. This requires recursive partitioning on all partitions - another complete pass. At this point we would conquer each partition, which is a final and third pass.

## **Q6: Suppose you are hashing a file and one of the partitions is 36 MB after the first pass (all other partitions can fit in the 32 MB buffer). How much larger is the I/O cost of externally hashing this file, compared to a scenario (with the same file) in which no partitions are ever oversized?** ( +2 points for correct )

Assume that a new hashing function is chosen for the second pass such that the records are distributed in a way that guarantees subsequent partitions to be under 32 MB.

Compared to a scenario with no oversized partitions, we need to recursively partition the 36 MB partition. Note that we do not need to do anything to the other partitions before conquering, and the IO cost of the conquer pass will not change - it does not depend on the number of partitions.

Thus, we take a single pass over 36 \* 1024 / 128 = 288 pages, which is 2 \* (288) = **576 I/Os** (a pass involves reading and writing each page once - 2 I/Os per page).

# 

# 

# Basic SQL Queries

Assume there exists a table called "Songs" with the following columns:

song\_id (Int, Primary Key), artist\_name (Text), title (Text), year\_released (Int), length\_seconds (Int), rating (Float)

An example record could look like the following:

(1, 'D.O.D.', 'Crazy Concurrency', 2007, 188, 10.0)

## **Q7: Which SQL query (or queries) will get the number of songs released after 2010 with a rating of at least 9.0?** ( +1 point for each correct query, -1 point for each incorrect query -- minimum of 0 points )

SELECT COUNT(\*) FROM Songs WHERE year\_released > 2010 AND rating >= 9.0;

~~SELECT COUNT(\*) FROM Songs GROUP BY year\_released, rating HAVING year\_released > 2010 AND rating >= 9.0;~~

This is incorrect. Suppose the Songs table contains only 5 songs - 2 songs with (year\_released, rating) = (2011, 9.5), and 3 songs with (year\_released, rating) = (2012, 9.0). Then, the output of the query will be a column with elements 2 and 3.

~~SELECT COUNT(\*) FROM Songs WHERE rating >= 9.0 GROUP BY year\_released HAVING year\_released > 2010;~~

This is incorrect. We can use the same counterexample to show that the output of the query could contain multiple elements.

SELECT COUNT(song\_id) FROM Songs WHERE year\_released > 2010 AND rating >= 9.0;

## **Q8: Which SQL query (or queries) will get the list of artists, without duplicates, who have produced at least one song more than 5 minutes long?** ( +1 point for each correct query, -1 point for each incorrect query -- minimum of 0 points )

SELECT DISTINCT artist\_name FROM Songs WHERE length\_seconds > 300;

SELECT artist\_name FROM Songs WHERE length\_seconds > 300 GROUP BY artist\_name;

~~SELECT artist\_name FROM Songs WHERE length\_seconds > 300 GROUP BY artist\_name, length\_seconds HAVING COUNT(\*) >= 1;~~

This is incorrect. Suppose the Songs table contains only 3 songs - 2 songs with (artist\_name, length\_seconds) = ('Led Zeppelin', 329), and 1 song with (artist\_name, length\_seconds) = ('Led Zeppelin', 482). Then, the output of the query will be a column with element 'Led Zeppelin' appearing twice.

~~SELECT artist\_name FROM Songs GROUP BY artist\_name, length\_seconds HAVING length\_seconds > 300;~~

This is incorrect. We can use the same counterexample to show that the output of the query could contain duplicate elements.