Topic: External Sorting

Assigned reading: Sections 13.1 and 13.2 from Raghu Ramakrishnan's book

Sorting is an important operation in DBMS. Sorting is required for many operations such as Selection, Joins, Order-by, and Duplicate elimination. In your algorithms class, you must have studied many sorting algorithms. Below is a comparison of these algorithms based on various aspects.

Algorithm	Time complexity	Space complexity	Stability	Ease of parallelization	Partial output available in between
Bubble	Bad	Good	Yes	Bad	Yes
Insertion	Bad	Good	Yes	Bad	No
Quick	Good except for worst case	Good	Yes only if additional linear space is used. Otherwise no.	Yes	No
Heap	Good	Good	No	No	Yes
Merge	Good	Bad	Yes	Yes	No

Note that there is no best sorting algorithm. When you analyzed these algorithms in your Algorithms class, you made two assumptions. First, the data fits into the main memory. Second, you focus only on compute-cost. However, both these assumptions are not true in the context of DBMS. A DBMS often has to deal with the data that might not fit into the main memory and the compute-cost is insignificant as compared to the I/O cost. In such scenario, sorting operation is referred as external sorting.

Let us start with a simple external sorting algorithm based on merge sort. We will assume that only three buffer pages are available in the main memory. Refer to Figures 13.2 and 13.3 from the Ramakrishnan's book.

The algorithms proceeds in the form of multiple passes over the data file. In Pass 0, we will read each page of the data file into the main memory, sort it, and then write it back to the disk.

Let us assume that the size of data is N pages. In the context of external sort, a run is a sorted subfile of the data. At the end of Pass 0, we have created N runs, each of size 1 page. In each successive pass, we will double the size of run and reduce the number runs to the half. Each pass needs I/O of 2N pages (reading N pages and writing them back). Total cost of sorting will be $2N(\text{Log}_2 N + 1)$. In each pass, we merge two consecutive runs. Each participating run is assigned one buffer page and one page is reserved for storing the temporary output. Thus, we need only three buffer pages.

Next, we will see how to make this sorting more efficient by utilizing more than three buffer pages.