COMP9311 Assignment 3

Ivan Teong z3386180

Q1)

- Sort file with **N** pages
- **A** buffer pages available
- 1st pass: read in/write out blocks of **B** buffer pages
- Subsequent passes: read in blocks of **k** buffer pages, write out blocks of **n** buffer pages

The first pass produces $N_1 = \lceil N/B \rceil$ sorted runs of B pages each, and the number of page I/O is 2 x N.

In subsequent passes, we can merge at most F = $\lfloor \frac{A-n}{k} \rfloor$ runs in each pass.

The number of subsequent passes is therefore $\lceil \log_F N_1 \rceil$.

Total number of page I/Os of sorting the file using the external merge sort algorithm by the settings above = 2 x N x ($\log_F N_1$ + 1).

Q2)

Yes it is possible.

Deleting entries may cause a bucket to become empty, after which it can be merged with its 'split image' bucket referenced by the directory entry which shares the last (local depth-1) bits. These deletions may result in the directory's size being halved and hence, its global depth decreased. This is caused by buckets merging that causes all local depth to be strictly smaller than global depth, and when each directory element points to the same bucket as its 'split image'.

For example, we look at a directory with global depth d, where its value ((2^{d-3} * k) + i) points to bucket A_i , besides 2^{d-1} which points to bucket B. When an entry is deleted which leaves bucket B empty, all the value ((2^{d-3} * k) + i) in the directory will point to bucket A_i . Therefore, the directory can be halved 3 times, causing the global depth to be reduced by 3 after the entry is deleted.

Q3)

The relational algebra $\prod a,b$ ($\sigma_{(a>50,000)}$ (R)) query selects the values of a and b from relation R where the value of a>50,000.

To find out which approach (or combination thereof) is the cheapest, we first eliminate the approaches that are either too expensive or not relevant to the situation.

Using index to access data will likely be cheaper than accessing it directly, so we eliminate the approach of accessing the sorted file for R directly.

We also eliminate the approaches where hash indexes are used since they are only useful for equality comparisons and not range searches. This eliminates the approaches of using a linear hashed index on attribute R.a and using a linear hashed index on attributes (R.a, R.b).

This leaves the following 3 approaches which are B-tree indexes, usable for equality and range queries efficiently. We will investigate to find out which of these is most likely to be the cheapest:

Using an unclustered B+ tree index on attribute R.b.:

The DBMS can do a binary search of $\log_2^{500,000}$ = 19 to find the first record with R.b for an unclustered B+ tree index to limit the search range for R.a, but since there is no condition stated for attribute R.b, this search cannot be performed.

Records with R.a>50,000 are in the last 495,000 pages of the file. The DBMS cannot traverse the B+ tree to find the first record with an R.b attribute (since there is no condition stated), so the leaf nodes cannot be scanned to find all qualifying tuples which satisfy R.b from which only the ones with RIDs falling into the last 495,000 pages of the file are followed and only those pages of the file are read. Therefore, it is still expensive to use an unclustered B+ tree index in this case due to attribute R.b not having any condition to limit the search range.

Furthermore, clustered B+ tree is good for sorting and unclustered B+ tree is usually bad. Unclustered B+ trees usually cost more I/Os than clustered B+ trees, so this case is likely to be more expensive than the clustered B+ tree indexes.

Using a clustered B+ tree index on attributes (R.a, R.b) versus using a clustered B+ tree index on attribute R.a:

For the former, our index will organize records by R.a first followed by R.b. We identify the starting leaf page for the scan by traversing the tree to find the first record with R.a>50,000, where the process usually involves 2 or 3 I/Os. Then we scan the leaf pages to identify the records where R.a>50,000, going through all of the 4,950,000 entries. Each of these entries points to one of the last 495,000 pages of the file. For the latter, our index will not organize records by R.a followed by R.b since the index is only used on attribute R.a. Hence, it is cheaper for the algorithm to find a and b simultaneously through records organized by R.a followed by R.b.

In conclusion, using a clustered B+ tree index on attributes (R.a, R.b) is most likely to be the cheapest of all the approaches.

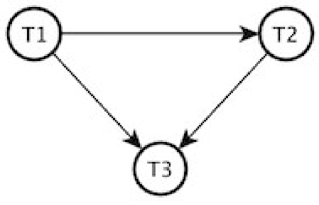
Q4)

a)

This schedule is serializable. There are 2 main justifications for this.

- Firstly, if the serializability graph is constructed as show below, it is acyclic and hence serializable. Non-serializable schedules are cyclic, but not this schedule.

Serializability Graph


$$\begin{tikzcd}
T1 \arrow[rr] \arrow[dr] & & T2 \arrow[dl] \\
& T3 &
\end{tikzcd}$$

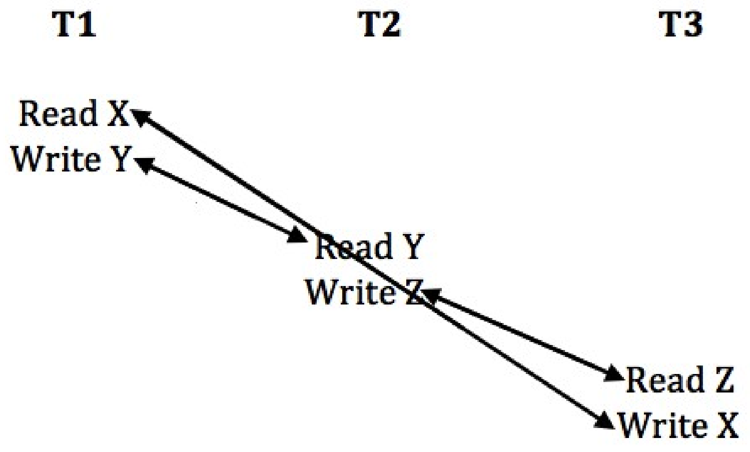
- Secondly, the schedule of the 3 transactions is a serial schedule, because the transactions are executed one after another (a running transaction is completed before another starts: “R1(X), W1(Y), R2(Y), W2(Z), R3(Z), W3(X)”) and not interleaved. Serial schedules are always serializable.

b)

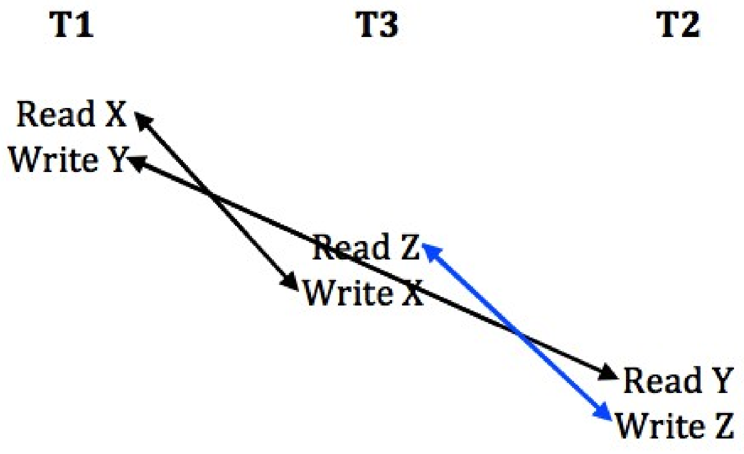
To check for possible serial schedules of the 3 transactions, we can switch the orders of the transactions (with the transactions non-interleaving/non-overlapping) and then see whether they are conflict-equivalent to S.

Conflict equivalence is when the order of any 2 conflicting operations is the same in both schedules (where both schedules involve the same transactions), which in this case is the schedule S and the possible serial schedules that we are looking for.

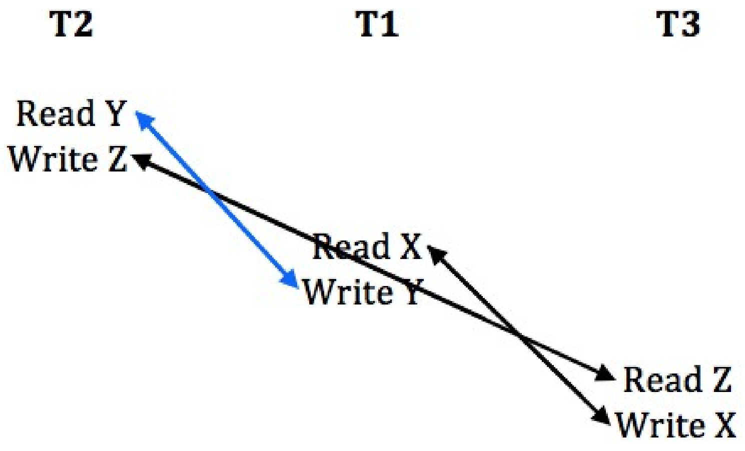
The conflicting operations in S:



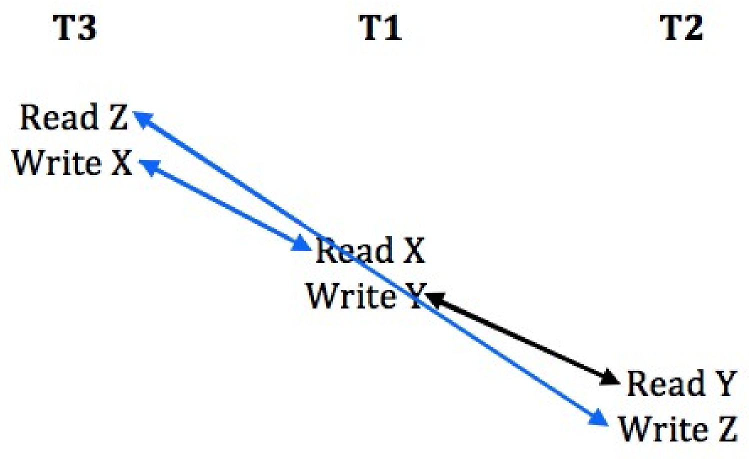
If we keep T1 as the first transaction and switch the orders of T2 and T3, one of the orders of conflicting operations will change. The order of conflicting operations will change from W2(Z) → R3(Z) to R3(Z) → W2(Z) (as shown in blue below). Hence, they are not conflict-equivalent.



If we let T2 be the first transaction, W1(Y) → R2(Y) will have its order of conflicting operations reversed (as shown in blue below) to R2(Y) → W1(Y), which will not be conflict-equivalent to S.



If we let T3 be the first transaction, R1(X) → W3(X) and W2(Z) → R3(Z) will have their orders of conflicting operations reversed (as shown in blue below) to W3(X) → R1(X) and R3(Z) → W2(Z) respectively, which will not be conflict-equivalent to S.



Hence, S has only 1 possible serial and conflict-equivalent schedule, which is exactly itself.