Samuel Pease 2017-12-03 HW #4

Q2) The number of I/Os for Method 1 with an external-memory merge sort is where B(R) is the number of disk blocks, 10,000, and M is the available memory which is 101. This comes out to = I/Os

The number of I/Os for method 2 is 100000+10000 because we simply need to look at the already sorted B tree and read every tuple from disk resulting in 100000 reads and then when enough tuples are read for a full block it is written. This results in 10000 writes as this is how many blocks of the data takes up. This means there are 110000 total I/Os for this method.

So Method 1 requires the least I/Os and is the winner.

Q3a)

natural join (

)

)

b) Since there are 60,000 orders over 2,000 distinct dates I will assume that the 2000 distinct date are the 2000 dates leading up to the present. I assume this since there are on average 30 orders a day I find it reasonable to believe there is not a day without a single order since they began.

So |Q| ≈ 60,000 \* 60/2000 = 1,800

c) Because the answer to (b):1800 is significantly larger than the number of distinct customer ID’s:1000 we will assume that all 1000 IDs are represented in this subset of orders. So |Q| ≈1000

d) |Q| ≈ 1,000 \* 1/50 = 20

e) |Q| ≈ 1,800 \* 20 \* 1/1000= 36

f) First I would read in all the tuples from order table which is has 60,000 tuples so it occupies 6,000 blocks. As each tuple is read in it is stored in memory if shipdate=:today. This will be approximately 2000/60000=30 tuples which is 3 bocks which fits in memory. So this process takes 6000 I/Os. Then all of Inventory is read in which takes 3,000 reads. It is saved in memory if it matches the book id of one of the tuples in memory and the block is pushed to disk when full. There are approximately 30\*30,000/600=1500 tuples in the final join so 150 blocks are pushed to disk. This means there are a total 6000+3000+150=9150 I/Os.

g) It is still most efficient to proceed as above for filling memory with tuples from Order with ShipDate=:today as there is no index on Shipdate. Then I would cache the root node of Inventory in a memory block. Then because the tree has height 3 (log\_100 3000 +1) it only takes 2 reads to retrieve each entry of Inventory. We will assume that the 30 records from Order in memory all have distinct BookIDs since the number of different bookIDs:600 is much larger. Then we will assume that the number of books in inventory matching these IDs is 30/600 \*30000 = 1500. Since the tree is sorted by BookID we will assume the tuples fill 150 block with tuples to be joined to tuples in memory. These tuples are joined and stored in memory until the final memory block is full then it is pushed to disk. This is a push of 150 blocks as discussed in (f). So this method is 6000+1+300+150=6451 I/Os.

X1a)The minimum number of blocks in memory to do it in one pass is 2000 because the distinct tuples need to be stored in memory as they’re read in order to compare to future blocks that are read.

B)Read in the 10,000 blocks one at a time and save distinct tuples in memory until all 1001 memory blocks are filled and then flush the last one to disk and continue reading the 10,000 blocks until they’re all read. This results in 10,000 unique tuples in the first 1000 blocks, save these to memory. Read in all blocks that were previously pushed to disk and save the distinct tuples in memory. Since all these tuples are distinct from the previous 10,000 and from each other we have found all 20,000 distinct tuples. It will take 9,000I/Os to read these tuples back in the worst-case scenario that the original 10,000 tuples were all distinct but that still sums up to 19000 I/Os excluding writing.