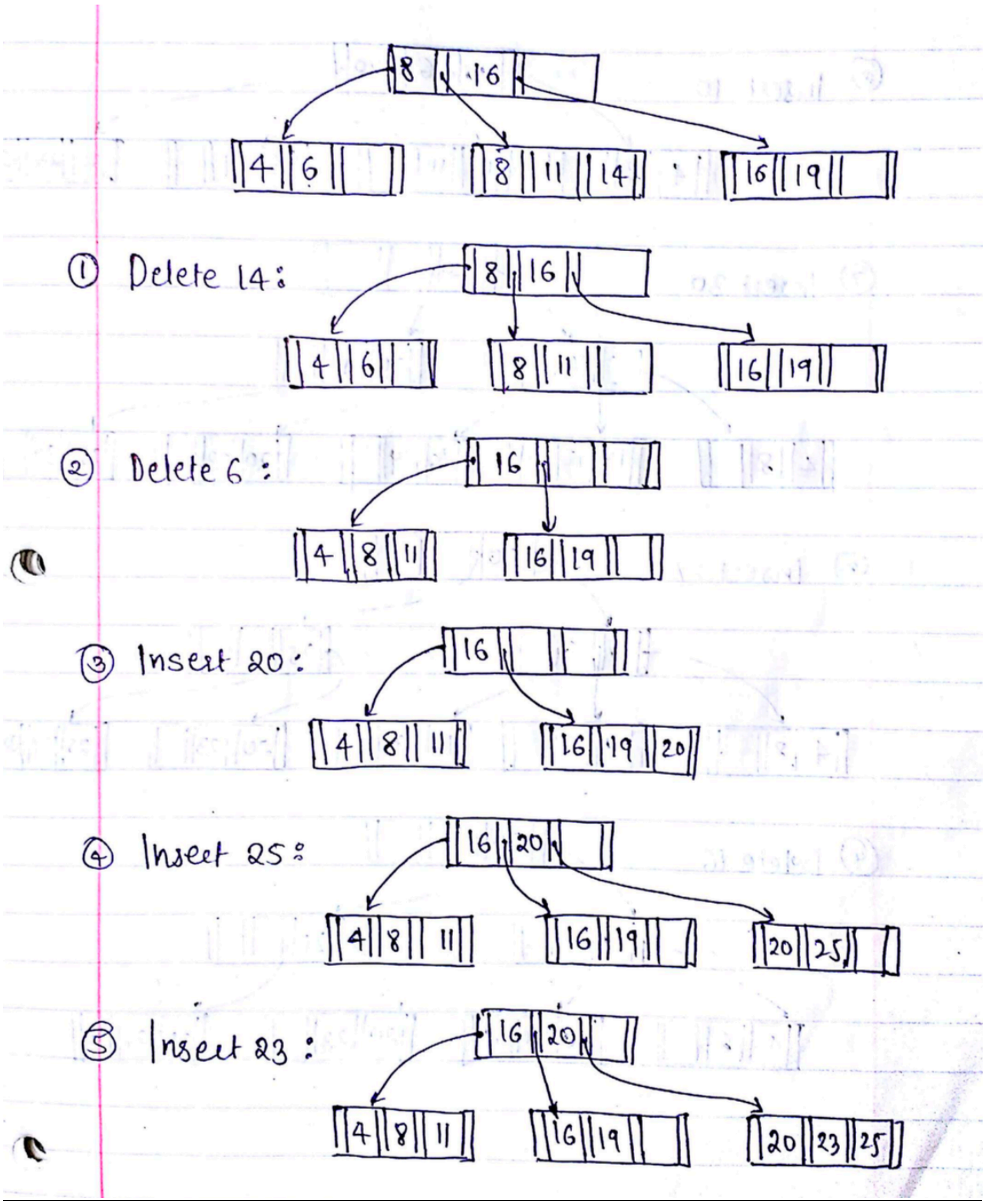
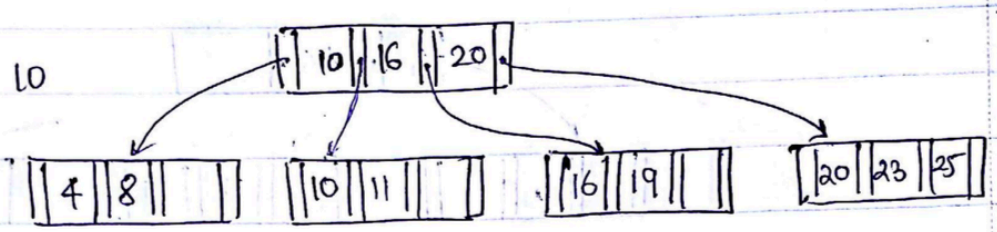
ASSIGNMENT 4

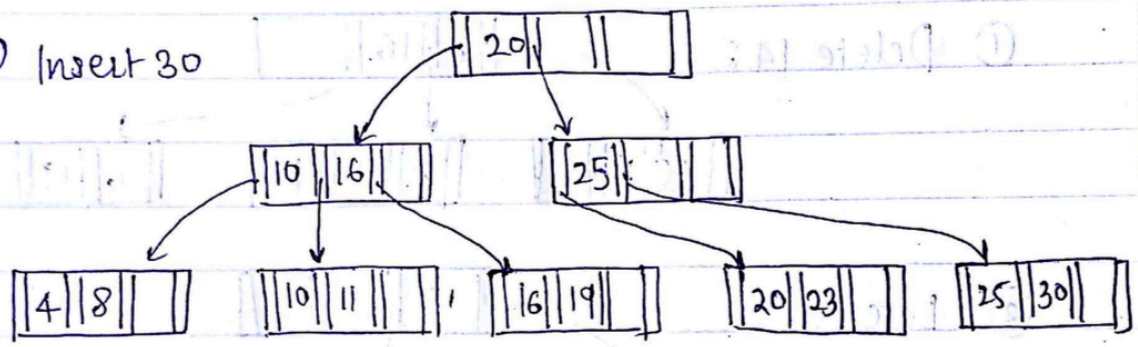
Problem 1:



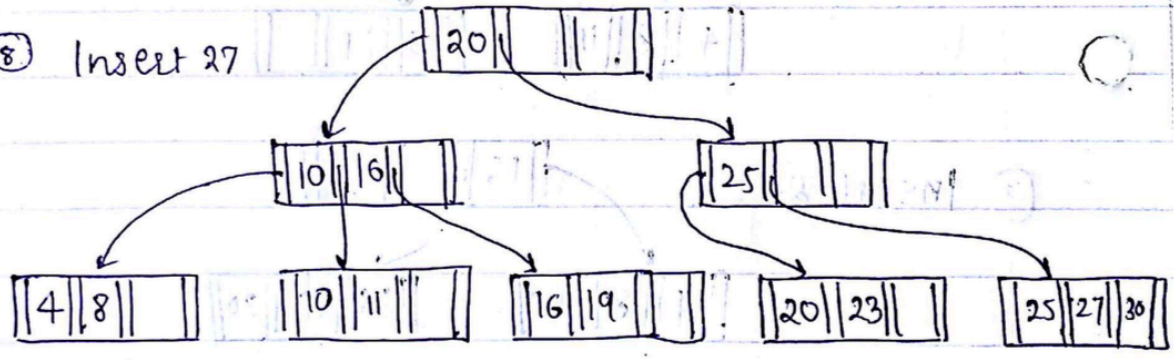
⑥ Insert 10



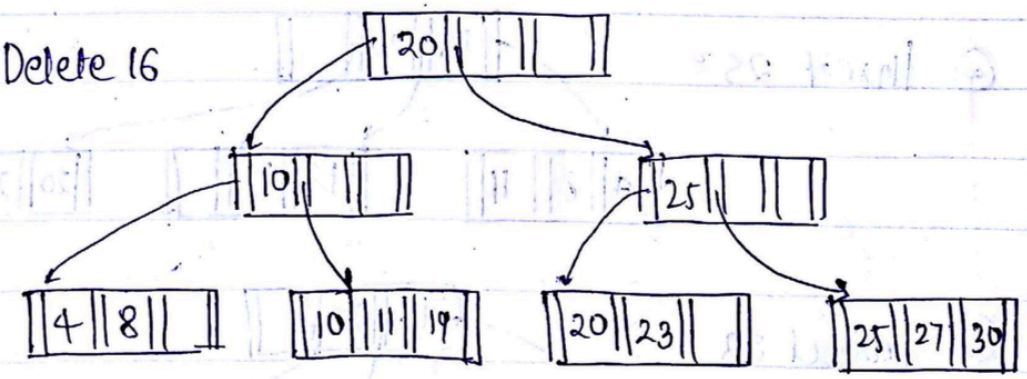
⑦ Insert 30



⑧ Insert 27



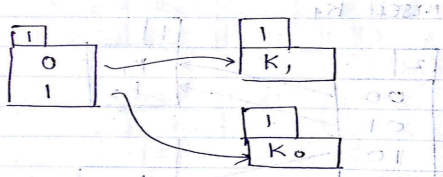
⑨ Delete 16



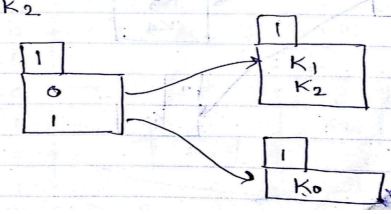
Problem 2:

Q2.

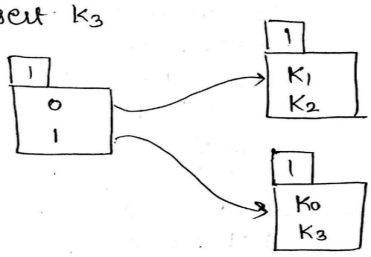
Keys	Hash values
K_0	11011000
K_1	01000101
K_2	01011101
K_3	10101010
K_4	10011001
K_5	00010100
K_6	10100101
K_7	01111001



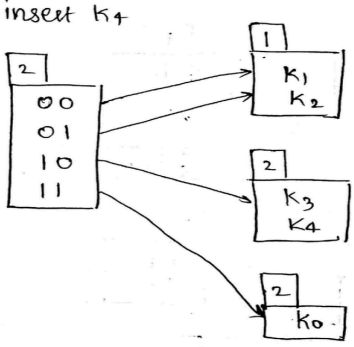
insert K_2

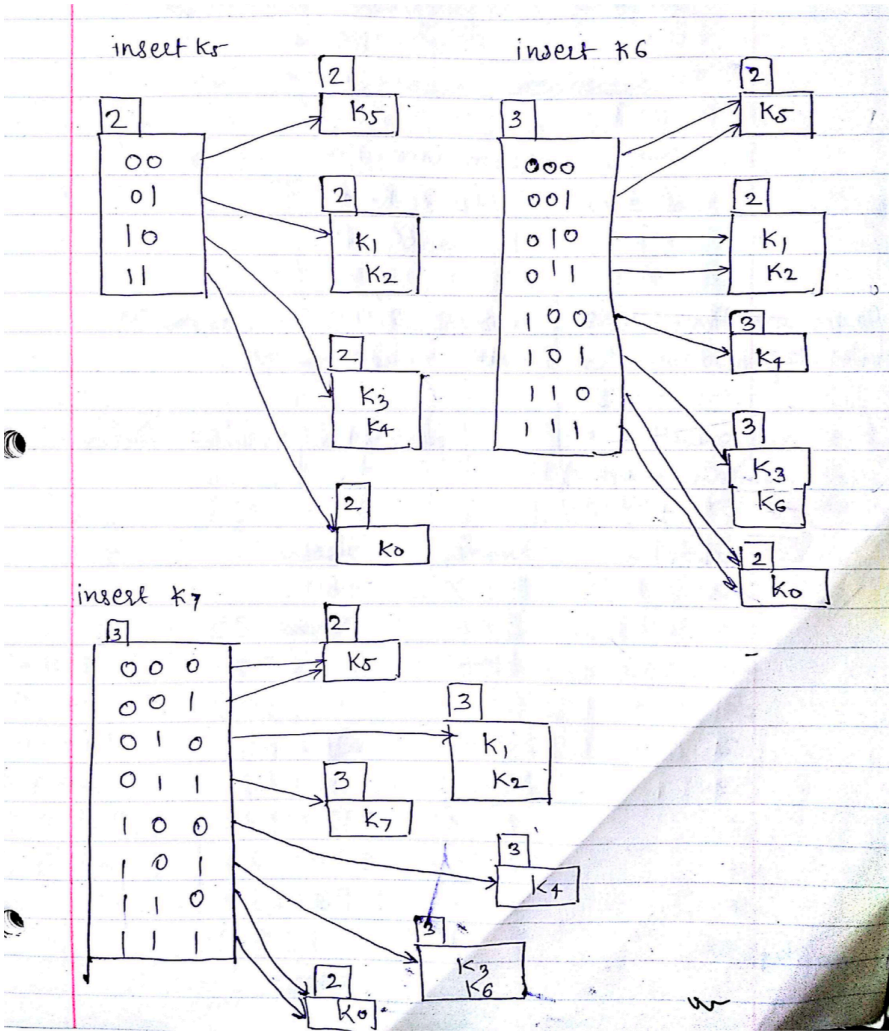


insert K_3



insert K_4




insert K5
2
00
01
10
11
2
K5
2
K1
K2
2
K3
K4
2
K0
insert K6
3
000
001
010
011
100
101
110
111
2
K5
2
K1
K2
3
K4
3
K3
K6
2
K0
insert K7
3
000
001
010
011
100
101
110
111
2
K5
3
K1
K2
3
K7
3
K4
3
K3
K6
2
K0

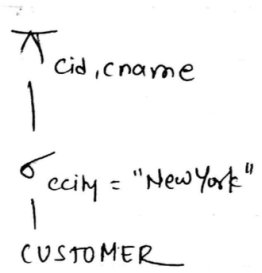
Problem 3:

a)

Query 1: List ids and names of all customers who stays in 'new York' city
Query 2: List ids and names of all customers who stays in 'new York' city and ordered on '2015-10-10'
Query 3: List ids and names of all customers who stays in 'Bunkford' city and ordered on '2015-10-10'

b)

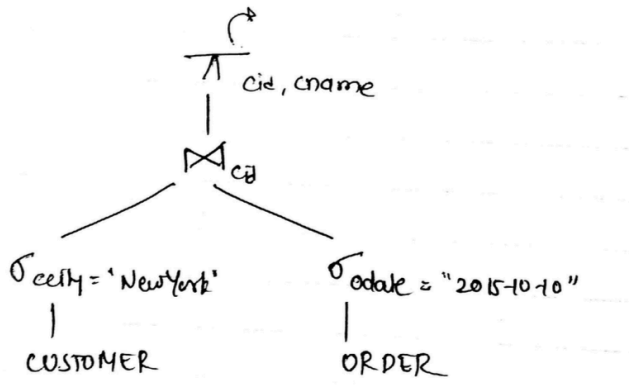
Query 1:



We are storing 500,000 customers i.e. 1% of customer live in NY, in the main memory which is 500,000*200 bytes ~ 1 GB. So there wont be any out of memory issue.

Now since there is no index we will scan the entire table of customers as we don't know when to stop the looking for customers: 50,000,000 * 200 bytes ~ 10GB with disk read at a speed of 100MB/s .The entire read can be done in around 10 ms + 10 GB/ 100 (MB/s) =100.01 s.

Query 2:

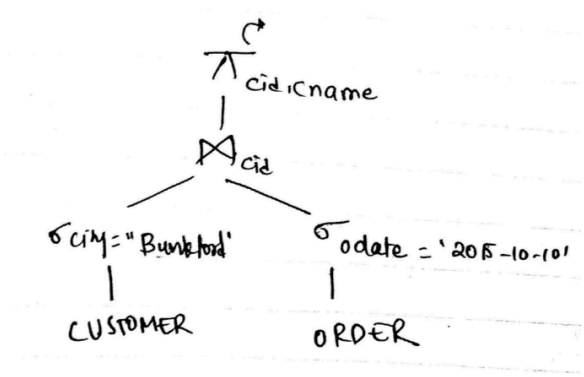


As stated 10 million orders were done on October 10, 2015 and 1% of those were by customers in NYC, we get 100,000 orders with each row about 300 bytes. Now if we do a direct scan here that will require 10 million * 300 bytes ~ 3GB which will lead to a out of memory storage as we have only 1GB of main memory to process.

So we first scan for order date October 10, 2015 which will give 1GB of data. Now since only 1% of customer i.e 500,000 customers live in New York so after scanning customer once only 0.1 GB of customer data will be available. Now we take only cid from order and cid, name from customer, thus filtered data will fit in main memory.

So scanning through customer of 10GB once will take 100.01 s
scanning through order table of 1000 GB(10 billion * 100 bytes) once will take 10ms+10000s ~10000.01s
So total time= 100.01+10000.01= 10100.02s.

Query 3:



As stated 10 million orders were done on the date 2015-10-10 and 100 customers live in Bunkford. Now if we do a direct scan from here that will lead to a out of memory storage as we have only 1GB of main memory to process.

So we first check how many tuples will survive the selection: we first scan for order date October 10, 2015 which will give 1GB of data. Only 100 customers live in Bunkford so scanning customer once , we will require only 20KB . if we take only cid from order and cid, cname from customer then it will fit in main memory.

So scanning through customer of 10GB once will take 100.01 s
scanning through order table of 1000 GB(10 billion * 100 bytes) once will take 10ms+10000s ~10000.01s
So total time= 100.01+10000.01= 10100.02s.

c)

Sparse clustered B+-tree index on ccity in the customer table

Each node has n pointers and n-1 keys. With 32 bytes per key, 8 bytes per pointer and assuming node to be of size 4096 bytes.
(n-1)*32+ n*8
= > n=103
Assuming 80% occupancy per node 103*(80/100)=82, each leaf node will contain 82 index entries and each internal node will have 82 children.

For the customer table, 20 records will fit into disk page (4000 bytes (node size) /200 bytes (tuple size of customer)). For 100% occupancy, a sparse index will have 1 entry per 20 records or a total of 2.5 million index entries.
Since there 82 index entries in each leaf node, there will be 2.5 million / 82~ 30488 leaf level nodes.
On the upper level, there will be 30488/82 ~ 372 nodes.
On the upper level, there will be 372/82 ~ 4 nodes.
And above that will be the root node.
So the B+ tree has 4 levels of nodes: the root, two internal levels, and the leaf level. Thus is takes about 5 * 10 ms = 50 ms to fetch a single record from the table using this index assuming no caching.
The size of the tree will be about (30488 + 372 +4 + 1) * 4KB ~123460KB ~ 12MB.

Dense unclustered B+-tree index on cid in the Order table

Each index entry has n-1 key nodes and n child pointers .now 8-byte key, and an 8--byte pointer and thus takes 16 bytes. Thus
16*n ~ 4096
=> n~256

assuming 80% occupancy we get about 256*(80/100)=204, each leaf node will contain 204 index entries and each internal node will have 204 children.

There are around 10 billion index entries, which mean there are about 49million leaf level nodes.
On the upper level , there will be 240,196 nodes.
On the upper level , there will be 1177 nodes.
On the upper level , there will be 5 nodes and
above this will be the root node.
Thus a total of 5 levels including the root,3 internal levels and the leaf level. Thus it will require 6*10 = 60 ms to retrieve a single record. The cost of the size is 49000000 + 240196+1177+5+1= 196965516KB = 197 GB.
The cost of fetching 50 records would involve 49 additional seeks into the underlying table, adding 49*10 = 490 ms to the 60ms for a single record thus totaling the total time to be 490 +60 =550 ms

d)

Query 1:

The ccity is sorted thus we just need to fetch first record from customer where ccity is New York. Since there are 1% of the total customers in NYC, cost of a complete scan of Customer relation would also require 1% of the cost i.e it will require 1 sec also we require 50ms to fetch a single record from disk .

The total cost to run the query will be 1050s.

Query 2:

We can reduce the current fetch time by choosing a clustered index on ccity in Customer and odate in Order table. Since there are 1% of the total customers in NYC,cost of a complete scan of Customer relation would also require 1% of the cost. I e it require 1050ms. Then we could scan Order table by odate but only fetch orders for 10th October 2015 through the index (10 million) which would take 10000 + 50= 10050ms.

The total cost to run the query will be 1050+10050=11100ms

Query 3:

We could choose the first index as clustered index on ccity to fetch customers in Bubkford city which will take 0.2 ms to scan. Then we will scan order table during the join but only fetching orders for October 10, 15. This will reduce the cost of accessing order table to 10050ms.

The total cost to run the query will be 50.2+10050=10100.2ms

Problem 4

a)The capacity of the disk is : 3 * 2 * 250,000 * 2000* 512 bytes ~ 1,536,000 MB ~ 1.536 TB.
Maximum rate of read using 5000 RPM (83.33 RPS) is 83.33 * 2000 * 512 bytes/s ~ 85.33 MB/s .
Average rotational latency (1/ 83.33) * 0.5 * 1000 ~ 6ms.

b)Note that 85.33 MB/s ~ 85.33 KB/ms . to read x KB of data it takes x/85.33 transfer time after initial 16 ms for seek and average rotational latency.

Block Model:

Read 4KB: t=16+ 4/85.33 = 16.04ms
Read 80KB: 20t = 320.93ms
Read 800KB: 200t= 3209.37ms ~ 3.2s
Read 80MB: 20000t= 320937ms ~ 320.93s

LTR Model:

Read 80KB: 16+80/85.33 = 16.93ms
Read 800KB: 16+800/85.33 = 25.37ms
Read 80MB: 16+80000/85.33 = 953.53ms

Conclusion: As the size of file increases predictions using LTR model are much faster than those of block model.

c)Phase 1: Repeat the following until all data Read:

Read 1GB of data (since main memory is of size 1GB) and sort it in main memory using any sorting algorithm. Write into a new file until all data is read.
Time to read 1GB ~ 16ms+1000000/85.33ms = 11735ms = 11.735s
To read and write 25 such files will take 11.735 * 2 * 25 = 586.76s

Phase 2: Merge the 25 files created in phase 1 in one pass. The main memory is divided into 25 input + 1 output i.e. total 26 buffers. Each buffer is of size 1024/26 = 39.38 MB.
For each buffer the read and write time is 16ms+39380/85.33ms=477.50ms.
25GB can be divided into 25*1024/39.38=650 pieces.
The total time is 477.50 * 650 * 2 = 620750ms ~ 620.75s

The total time for sorting the file in a single pass is about 586.76s + 620.75s = 1207.51s , almost 20 min.

d)

As we need to consider the two merge phase in this problem. As we need to merge 25 files of 1 GB each we can consider 5 way merge followed by another 10 way merge.

For 5 way merge:

Initially we divide the memory into 1024/6 = 170.67 MB.
Hence, for each buffer the read and write time = 16ms + 170.67 /85.33 sec = 2.016 sec.
Number of pieces will be 25*1000/170.67 = 150 pieces.
Therefore, we get time taken to be 150 * 2.016 * 2 =604.8sec

For 10 way merge:

Initially we divide the memory into 1024/11 = 93.09 MB.
Hence, for each buffer the read and write time = 16ms + 93.09 /85.33 sec = 1.1069 sec.
Number of pieces will be 25*1000/93.09 = 275 pieces.
Therefore, we get time taken to be 275 * 1.1069 * 2 =608.79sec

The total time will be: sort time + merge 1 + merge 2 = 586.76 + 604.8 + 608.79 = 1800.35s ~ 30 min.

Thus we conclude one merger phase is better than two-merge phase.