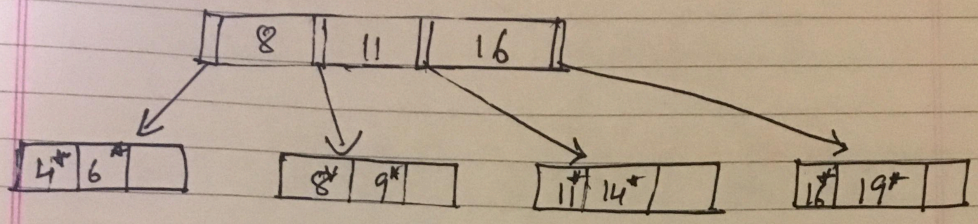
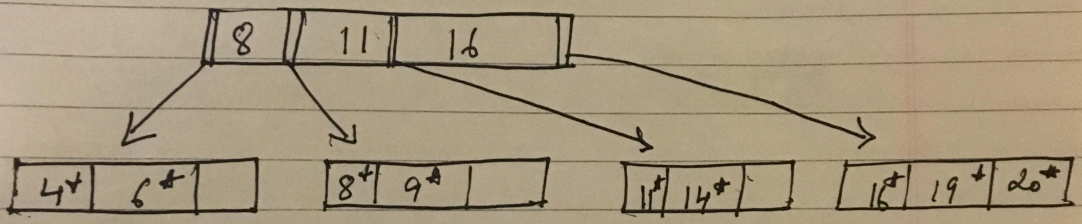
Question 1:

Ques 1

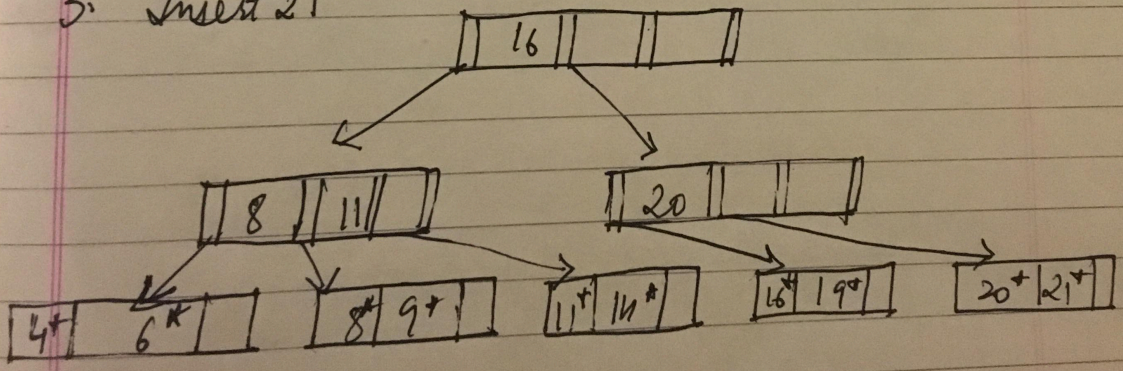
1. Insert 9



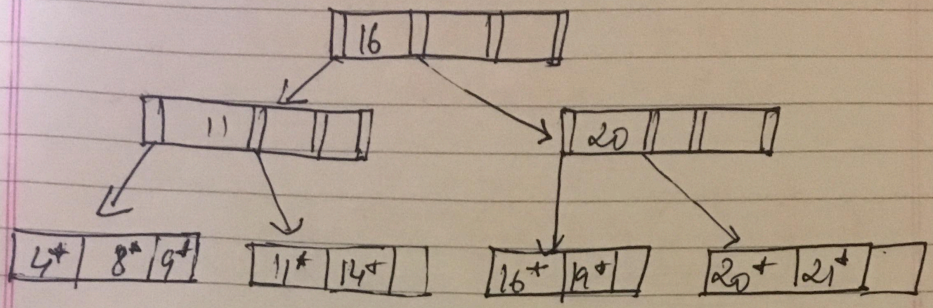
2. Insert 20



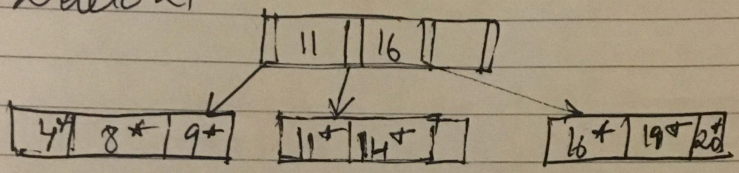
3. Insert 21



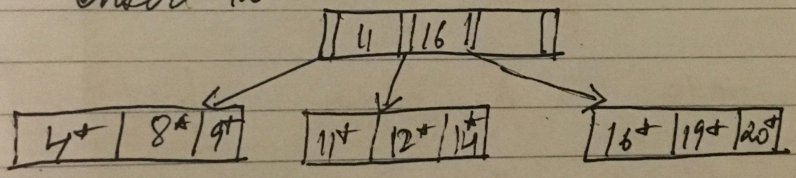
④ Delete 6 :



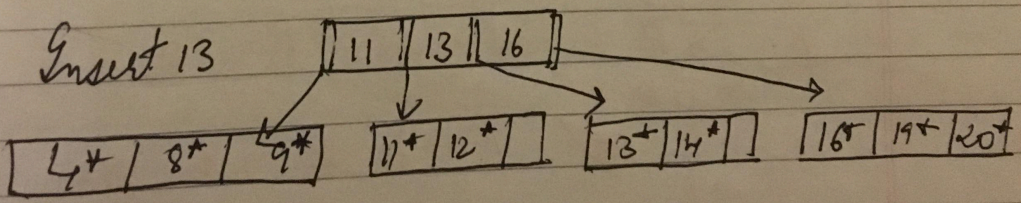
⑤ Delete 21



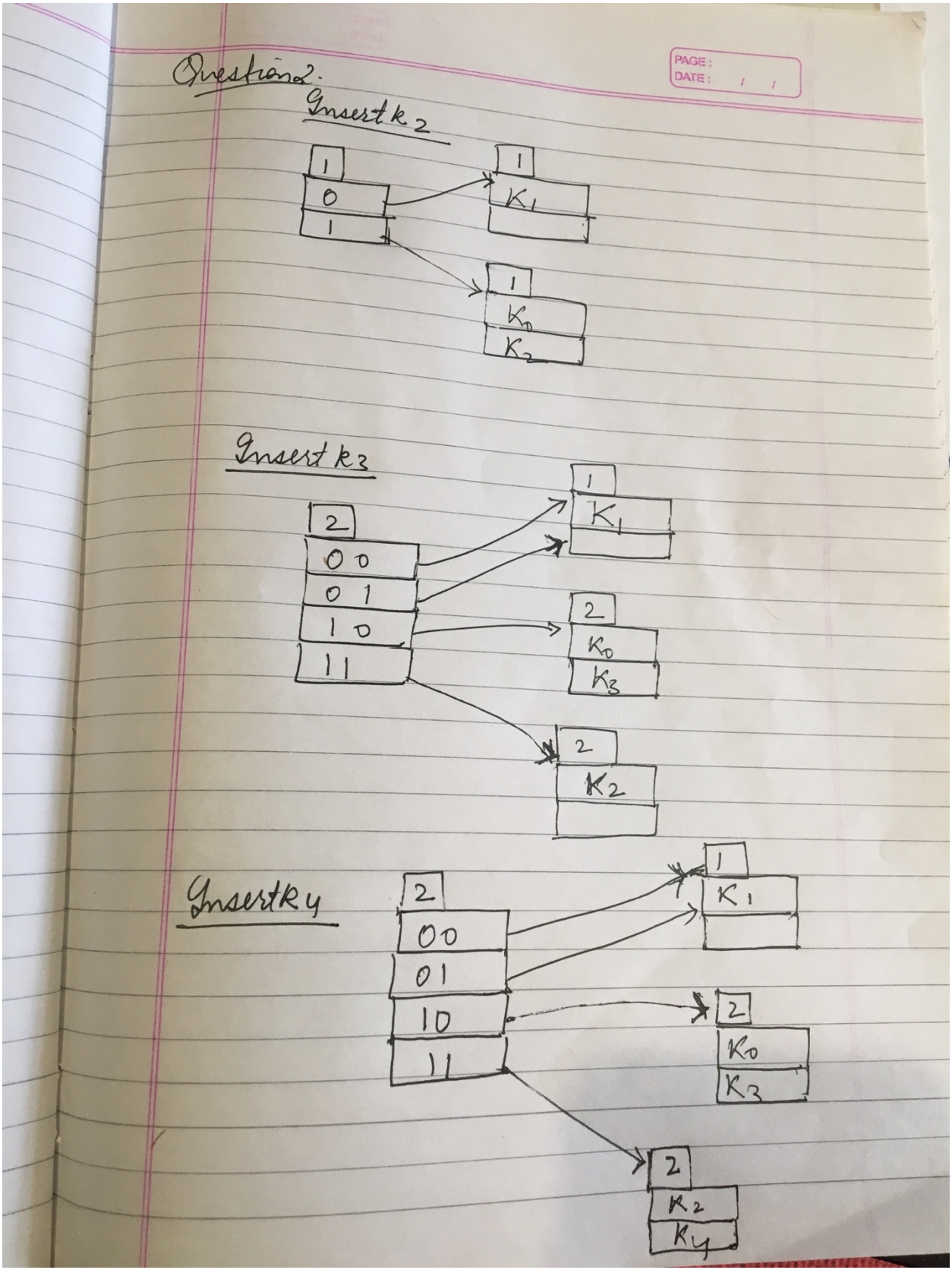
⑥ Insert 12

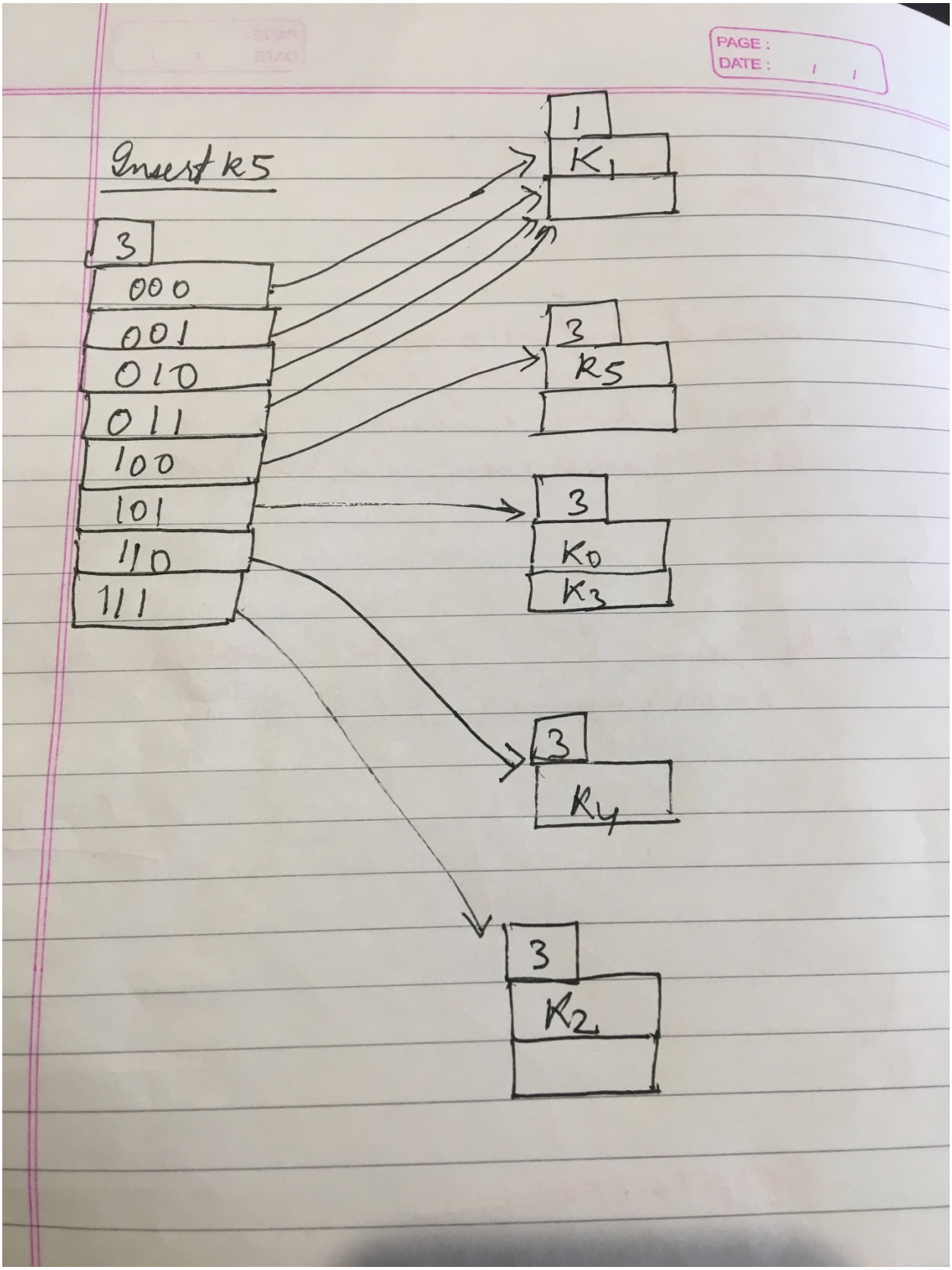


⑦ Insert 13

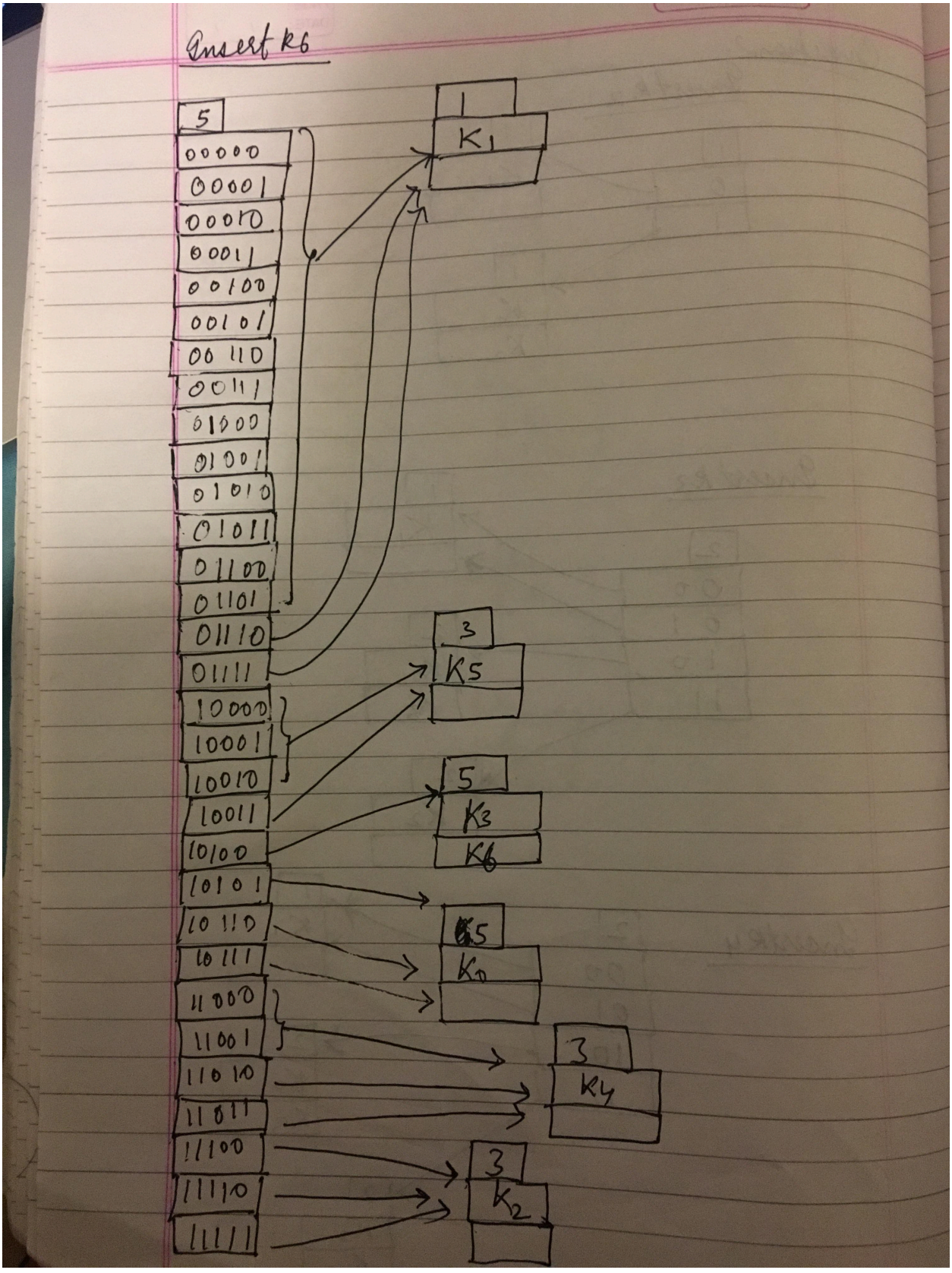


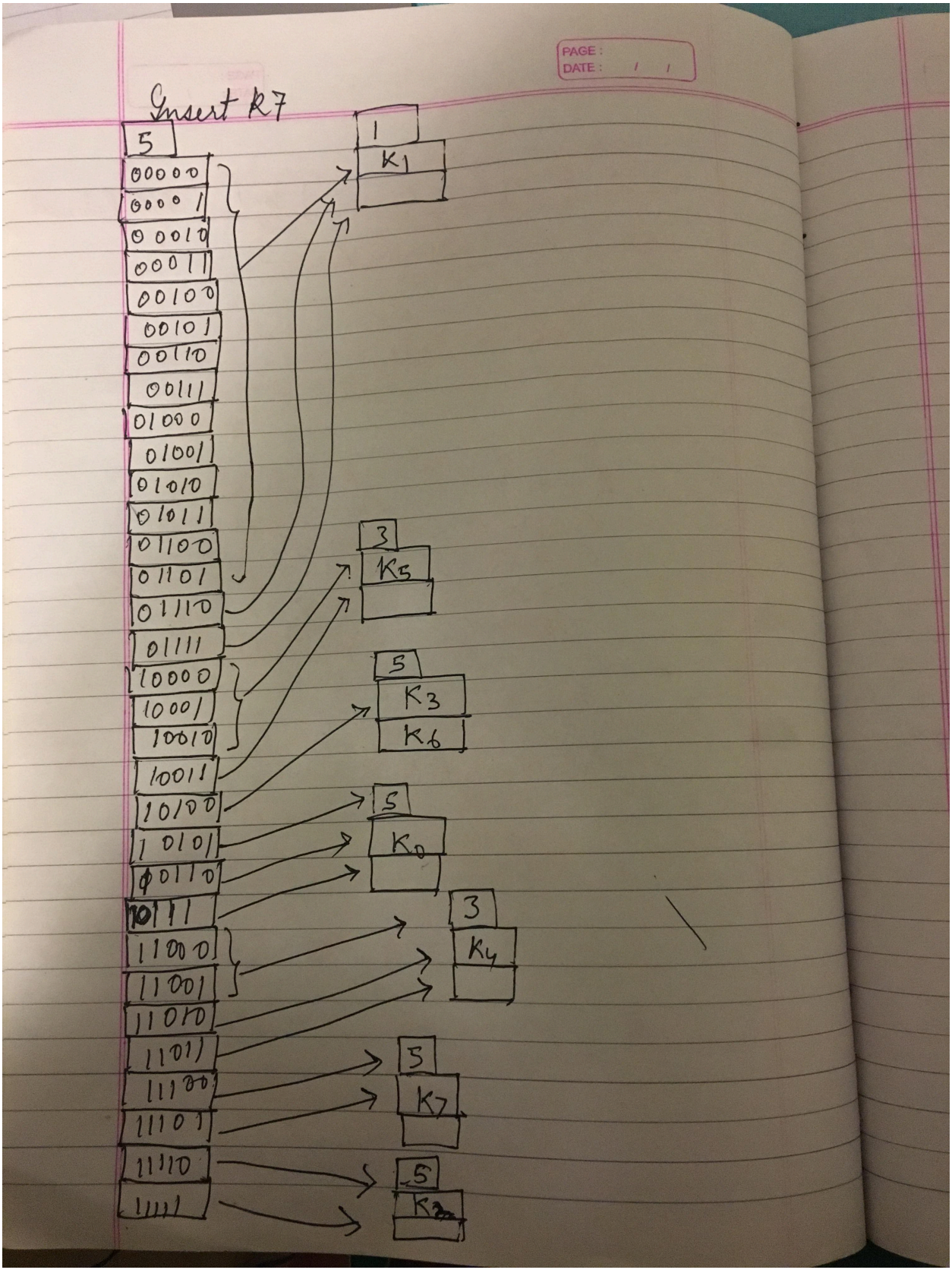
Question 2:


Question 2:
PAGE :
DATE : / /
Insert k2
1
0
1
1
K1
1
K0
K2
Insert k3
2
00
01
10
11
1
K1
2
K0
K3
2
K2
Insert k4
2
00
01
10
11
1
K1
2
K0
K3
2
K2
K4


Insert k5
3
000
001
010
011
100
101
110
111
1
K1
3
K5
3
K0
K3
3
K4
3
K2

Insert K6


5
00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001
10010
10011
10100
10101
10110
10111
11000
11001
11010
11011
11100
11110
11111
1
K1
3
K5
5
K3
K6
K5
K0
3
K4
3
K2


PAGE :
DATE : / /
Insert K7
5
00000
0000 1
0 0010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001
10010
10011
10100
1 0101
00110
10111
11000
11001
11010
11011
11100
11101
11110
11111
1
K1
3
K5
5
K3
K6
5
K0
3
K4
5
K7
5
K2

Question 3:

(a)
Query1: list the ids and the corresponding artist ids of tracks have been played in 11/29/2017.
Query2: list the ids and the corresponding artist ids of tracks which the artist's genre is Jazz and have been played in 11/29/2017.
Query3: list the ids and names of all users living in Miami who have played any song has duration less than 10s.

(b)

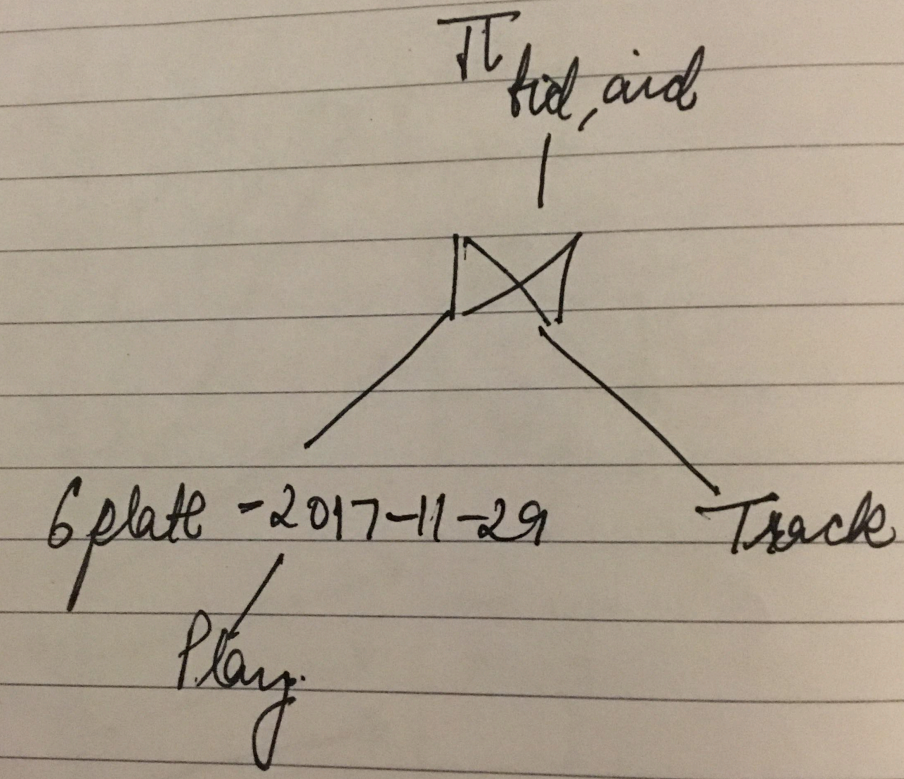
Query Plan Tree –

The query processing plans are shown below. Note that sort and hash-based joins are only used when even the smaller of the two inputs is much larger than memory, which in this case never happens. Thus, all joins are blocked nested-loop joins.

Query1:
Table size: (Sanning size → Scanning time**)**

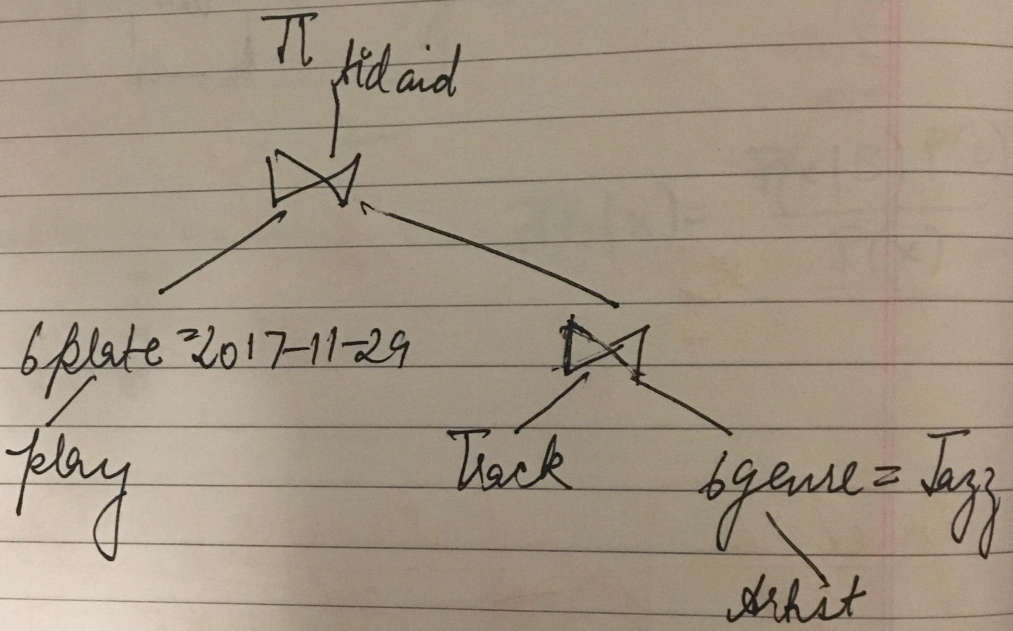
- User: 100 m * 100 bytes = 10 b bytes = 10 GB
- Track: 5 m * 100 bytes = 500 m bytes = 0.5 GB
- Artist: 1 m * 100 bytes = 100 m bytes = 0.1 GB
- Play: 20 b * 40 bytes = 800 b bytes = 800 GB

①

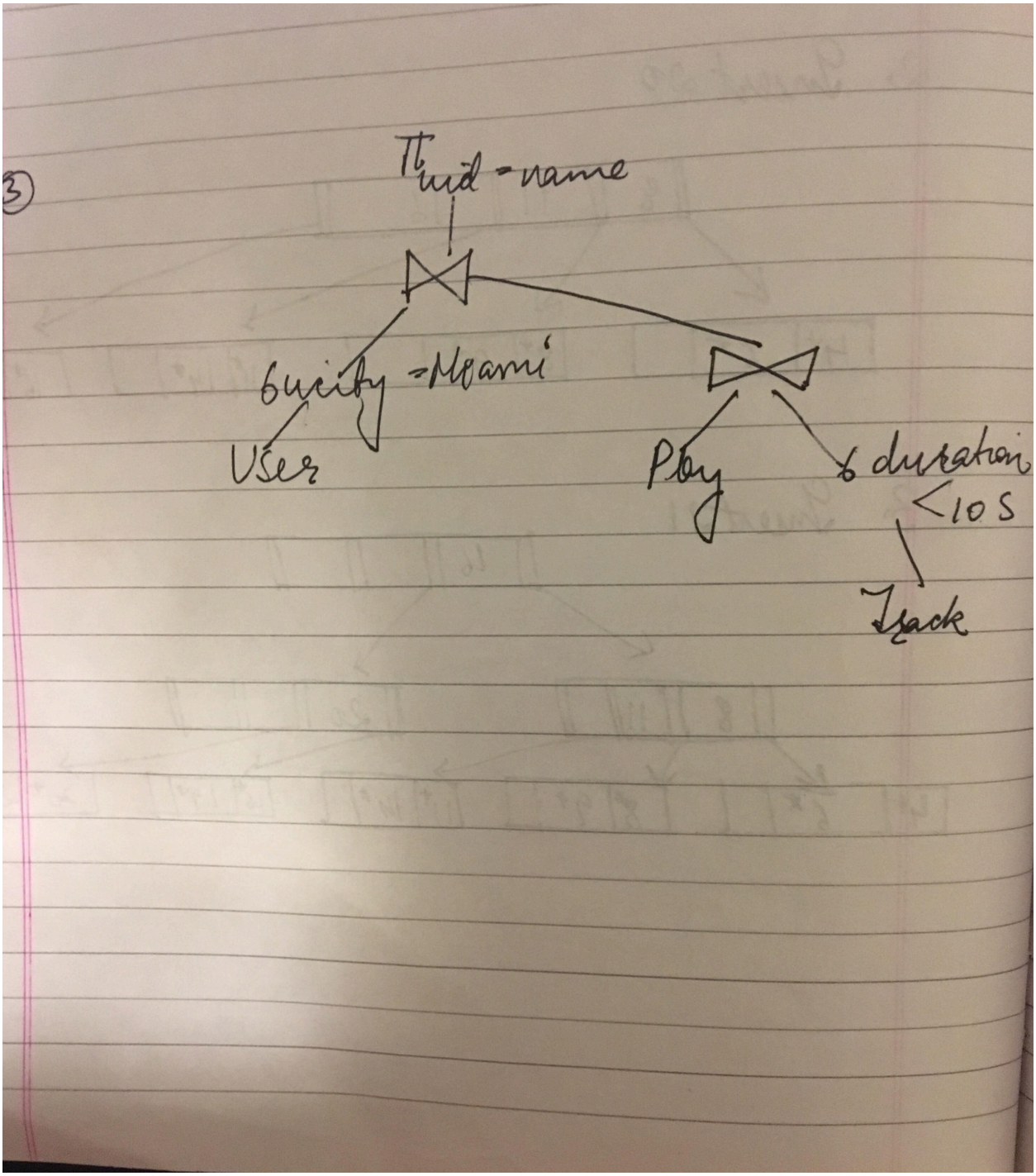


There are 8GB of play records in November 29, 2017(200 million such playing records since playing records distributed evenly and independently during 100 days) which larger than the main memory(1 GB). However, we only need the tid and tdate of each record, which would probably fit in about 20 of the 40 bytes of a play tuple(8 bytes for tid and 12 bytes for pdate). So this brings size down to maybe 4GB, meaning that 1/4 of the required data from Play fits in main memory. Thus, we perform a block-nested loop join where we read first 1/4 of Play, then scan Track once, and then read the other 1/4 of Play, and scan Track one more time, and repeat this process twice more. So the total amount of data to be read is 800GB + 4*0.5GB = 802 GB, costing 8020 seconds.

Query 2:


π
tid aid
σplate='2017-11-29'
play
Track
σgenre='Jazz'
Artist

We scan Artist to find whose genre is Jazz. There are 10000 such artist records(1% * 1 million artist). On average, every artist has 5 tracks(5 million / 1 million since data is evenly and independently distributed), so we would expect about 50000 tuples in the Track table to join with these 10000 artists. We do the join by placing the 10000 artist tuples in main memory and then scanning Track once, giving us 50000 tuples that clearly also fit in main memory. (Note that we actually only need the tid attributes of these tuples for the next join.) So, we place these in main memory. Then we scan Play once, checking which plays were done on November 29, 2017, and then looking if those tuples match any of the surviving Track tuples. Thus, we scan Artist, Track and Play once, for a total of 0.1 GB + 0.5 GB + 800 GB = 800.6 GB, costing 8006 seconds.



After scanning Track once, we have 50000(1% * 5,000,000) tuples of tracks have a duration shorter than 10 seconds. Of course, we can project away everything from the resulting records except the tid, so realistically the size per record might be only, say, about 8 bytes, or 8 * 50000 = 400 KB total.

Assume that data is evenly and independently distributed, so we got about 20 billion / 5 million = 400 plays/per song. Thus, after joining the 50,000 Track tuples with Play, by scanning Play once, we have 400 * 50,000 = 20 million tuples that need to be joined with User.

C)

C1:
n = 4096 / (8 + 12) ~ 205
Entries per node = 205 * 80% = 164
For the dense index on Artist, there will be 1 million index entries, in 1,000,000 / 164 = 6098 leaf nodes, and 6098 / 164 = 37 on the next level, and then the root. So this tree has 3 levels of nodes, and 4 * 5ms = 20 ms are needed to fetch a single record from the table. The size of the tree is dominated by the leaf level, which is about 6098 * 4KB = 24.392 MB

C2:

n = 4096 / (8 + 12) ~ 205
Entries per node = 205 * 80% = 164
For dense clustered B+-tree index on tid in the Play table, there will be 20 billion entries, in 20,000,000,000 / 164 = 121,951,220 leaf nodes, and 121,951,220 / 164 = 743,605 on the next level, then 4,534, then 28, then the root. Thus, this tree has 5 levels and 6 * 5 ms = 30 ms are needed to fetch a single record from the table. The cost of fetching 50 records would involve 49 additional seeks into the underlying table, adding 49*5 = 245 ms to the 30ms for a single record which is 275 ms. The size of the tree is dominated by the leaf level, which is about 121,951,220 * 4KB or about 487 G

D)

Query1: We would choose a clustered index on pdate in Play, to accelerate the fetching of plays has been done on November 29, 2017, which would take only 1% of the cost of a complete scan of Play. So we would basically be left with the cost of scanning Track once in the final join, about 5 seconds. So the total cost is down to 85 seconds.

Query2:We choose clustered indexes on genre in Artist and on pdate in Play. We would use the first index to fetch artist whose genre is Jazz, which would take only 1% of the cost of a complete scan of Artist, and accelerate the initial selection to 0.01 s. We would use the second index to fetch tracks played on 2017-11-29, which would take only 1% of the cost of a complete scan of Play and accelerate the initial selection to 80 s.

Query 3: For the third query, we can choose a clustered index on ucity in User, to accelerate the initial selection by a factor of 1000 (since 0.1% of all users live in Miami), to 0.1 seconds. Then we can accelerate the join between Play and tracks which duration is less than 10s by using a clustered index on tid in Play. Resulting in one lookup into Play for each tuple in selected Track. This would be 50,000 lookups of 30ms each in the case of a clustered index, or about 1,500 seconds.

Question 4

Ques 4

a) Capacity of the disk =
$3 \times 1 \times 200{,}000 \times 1000 \times 1024 \simeq 614{,}400\ MB \simeq 614.4\ GB$
$\sim 600\ GB$

Maximum rate at which data can be read from disk (using 12000 RPM = 200 RPS)

$= 200 \times 1000 \times 1024$ byte/s $\sim 200\ MB/s$

Average rotational latency $= \frac{60}{12000} \times \frac{1}{2}$

$= 2.5$ ms.

b) Note that 200 MB/s ~ 200 KB/ms, so it takes $\frac{x}{200}$ ms transfer time to read x KB of data after the initial 4+2.5 = 6.5 ms for seek & avg. rotational latency.

Block model:

Read 4 KB : $t = 6.5 + \frac{4}{200}$ ms $= 6.52$ ms

200 KB : $T = 50t = 326$ ms

20 MB : $T = 5000t = 32.6$ s.

2 GB : $T = 500000t = 3260$ s

LTR model:

Read 200 KB = $6.5 + \frac{200}{200} = 6.5$ ms

20 MB = $6.5 + \frac{20000}{200} = 106.5$ ms

2 GB = $6.5 + \frac{2000000}{200}$ ms

$= 1,000,006.5 \times 10^{-2} \times 10^{3}$ s

$= 10.0065$ s.

Thus, the predictions by the LTR model are much faster, & more accurate than those for the block model when reading large files.

c) Phase 1: Repeat the following until all data is read. Read 4 GBs of data & sort it in main memory using any sorting algorithm. Write it into a new file until all the data is read. The time to read 4 GBs

$$6.5 + \frac{4^2 \times 10^4}{200} \text{ ms}$$

$$6.5 + 20000 \text{ ms}$$

$$= 20006.5 \text{ ms}$$

$$= 20.0065 \text{ s}$$

To read & write $\frac{256}{4} = 64$ such files takes

$$= 64 \times 20.0065 \times 1$$

$$= 1280.416 \text{ s.}$$

$$\sim 1280 \text{ s.}$$

Phase 2: Merge 64 files created in Phase 1 in one pass. The main memory is divided in 65 buffers

Each buffer is of size $\frac{1024}{65} = 15.75$ MB

For each buffer, the read & write time is

$$6.5 + \frac{15.75}{200} \text{ s}$$

$$= 85.2 \text{ ms}$$

256 GB can be divided into 256×1024

$$\sim \frac{16644}{3076} \approx 3900 \text{ pieces}$$

The total time is 85.2×3076 ms

$$= 262075.2 \text{ ms}$$

$$\approx 262 \text{ s}$$

Total time for sorting the 256 GB file in a single pass is about $1280 \text{ s} + 262 \text{ s}$

$$\approx 1542 \text{ s}$$

merge. So we need 17 buffers.
so each buffer is of size
and since we have 4 GB of main memory

d) For merging 64 files in 2 merge passes, we can have 2 8-way merge. So we need 9 buffers; each of size = $\frac{1024}{9} \sim 114$ MB.

Reading 1 buffer takes $\frac{114}{200}$ s ~ 0.57 s.

and reading all 256 GB (64 files) takes close to 145.92 s

Since, the cost of seeks, is very small compared to the costy transfer.

A precise analysis would show that this option is worse than the option in (C) (however this option is better than other combinations like (16 x 4) or (32 x 2) etc.