| Size n of relation S | Avg execution time to scan S(in ms) | Avg execution time to sort S (in ms) |
|---|---|---|
| 10 | 0.028 | 0.044 |
| 100 | 0.056 | 0.091 |
| 1000 | 0.290 | 0.688 |
| 10000 | 2.031 | 7.511 |
| 100000 | 26.252 | 166.614 |
| 1000000 | 202.440 | 1342.876 |
| 10000000 | 2220.582 | 16748.270 |
| 100000000 | 14,756.498 | 81476.678 |

3.a The above table determines the query plan for scanning and sorting as per the observation. Execution time for scanning is less compared to that of the execution time for sorting. The scanning and sorting time increases as the size n of the relation S increases.

3.b As the size of the relation S increases the execution time for sorting increases.

3.c Yes, it conforms with the formal time complexity of external sorting as it takes O(nlogn) time. The file is subdivided into subparts of B blocks and each of these subfiles will be sorted in a parallel way and will be Merged. We will need to merge B Sorted files which will add up to 2N($\log_B$(N)). Hence it conforms with the formal time complexity of external sorting.

3.d Set work_mem='64kB'

| Size n of relation S | Avg execution time to scan S(in ms) | Avg execution time to sort S (in ms) |
|---|---|---|
| 10 | 0.036 | 0.026 |
| 100 | 0.041 | 0.074 |
| 1000 | 0.232 | 1.406 |
| 10000 | 2.183 | 10.324 |

| | | |
|---|---|---|
| 100000 | 23.916 | 128.327 |
| 1000000 | 256.360 | 1079.581 |
| 10000000 | 2882.076 | 11936.966 |
| 100000000 | 14,882.586 | 112318.732 |

Set work_mem='1GB'

| Size n of relation S | Avg execution time to scan S(in ms) | Avg execution time to sort S (in ms) |
|---|---|---|
| 10 | 0.016 | 0.034 |
| 100 | 0.038 | 0.085 |
| 1000 | 0.212 | 0.847 |
| 10000 | 1.883 | 7.004 |
| 100000 | 20.081 | 82.512 |
| 1000000 | 193.151 | 926.272 |
| 10000000 | 2064.654 | 10366.519 |
| 100000000 | 15018.812 | 114788.745 |

3.e

| Size n of relation S | Avg execution to create index indexedS(in ms) | Avg execution time for range query (in ms) |
|---|---|---|
| 10 | 0.613 | 0.55 |
| 100 | 3.77 | 5.5 |
| 1000 | 28.7 | 30.23 |
| 10000 | 109 | 104.88 |
| 100000 | 800.34 | 700.55 |
| 1000000 | 9000 | 2300 |
| 10000000 | 88780 | 16450 |

|  |  |  |
|---|---|---|
| 100000000 | 970455 | 21340 |

Average execution time for range query decreases compared to other above algorithms when using b+ indexing.

7. a. Size of node n can be calculated using the below
N <= (Block_size - |blockaddress| ) / (|blockaddress| + |key|)
N <= (4096 – 9) / (9+8)
N <= 4087/17
N <= 240
N = N+1 = 241 (adding 1 to N)
Because each block can fit a maximum of 240 records.
Minimum Time :- ([log241(10^9+1)] + 1) * 10ms
= (9 (log24110+1) + 1 ) * (10)ms
= 4 * 10 ms
= 40 ms

b. Calculating the maximum time to insert a record with a key will be taken when branching factor is minimum.
(240/2) +1 = 121
Height of the tree = log121(10^9+1) * 10ms
= 5 * 10ms = 50ms

c. For Two Levels it will be= 4096+(241*4096)
= 991232 bytes
= 1MB

For Three Levels it will be= 1MB + (241)^2 * 4096
= 238899768
= 238MB

10. Size = 400
Execution-TIme: 0.280 ms

Size = 4000
Execution-Time: 1.280 ms

Size = 40000
Execution-Time: 8.258 ms

11. Size = 400
    Execution-Time: 0.426 ms

    Size = 4000
    Execution-Time: 3.175 ms

    Size = 40000
    Execution-Time: 13.755 ms

12. Size = 400
    Execution-Time: 0.285 ms

    Size = 4000
    Execution-Time: 0.963 ms

    Size = 40000
    Execution-Time: 4.052 ms

13. **For Join Or IN**
    Size = 400
    Execution-Time: 0.484 ms

    Size = 4000
    Execution-Time: 5.066 ms

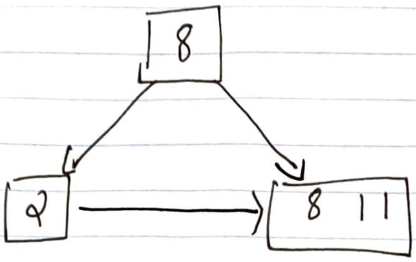
    Size = 40000
    Execution-Time: 30.397 ms

    **For Anti-Join Or Not-IN**
    Size = 400
    Execution-Time: 0.614 ms

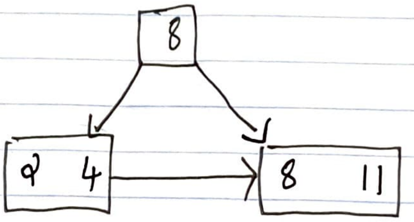
    Size = 4000
    Execution-Time: 9.029 ms

    Size = 40000
    Execution-Time: 30.805 ms

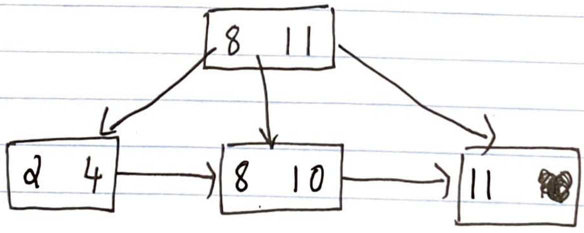
8]



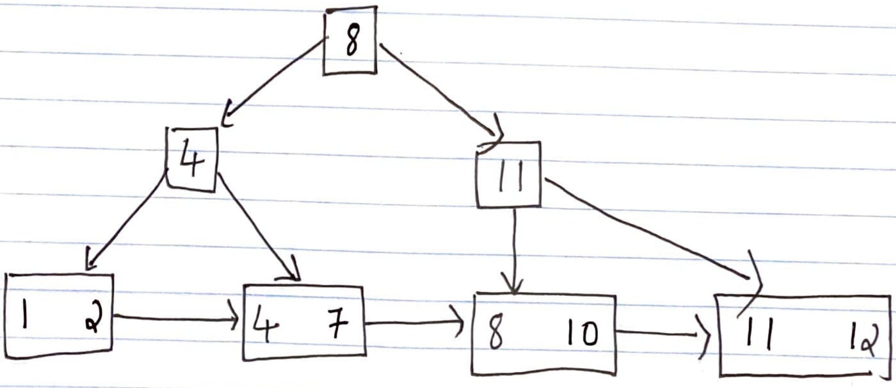
a] Inserting 4



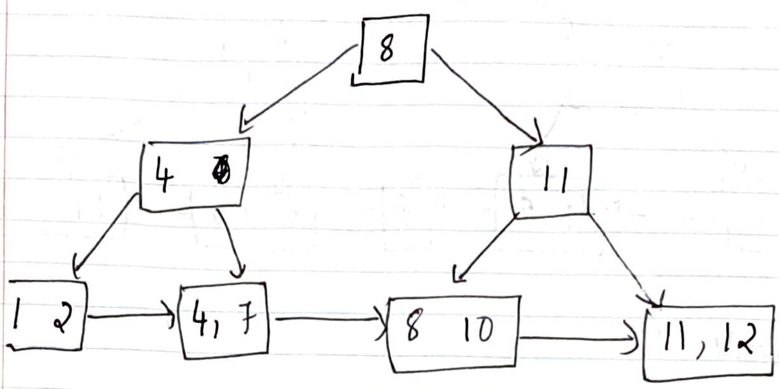
Insert 10



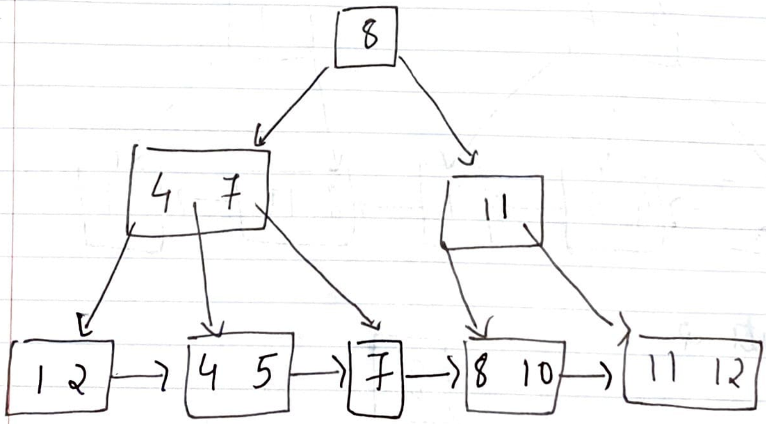
Insert 12 and 1



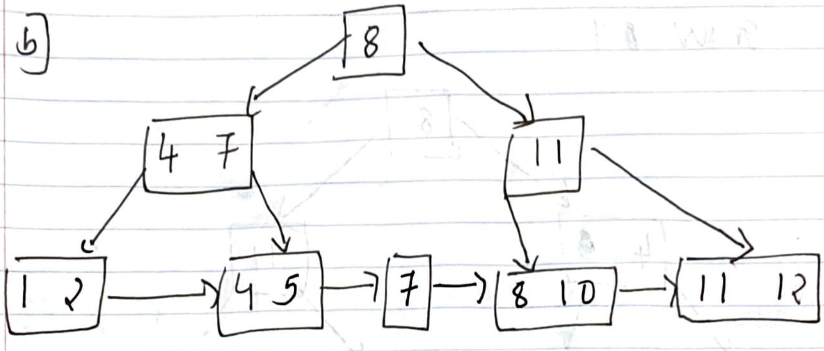
Insert 7



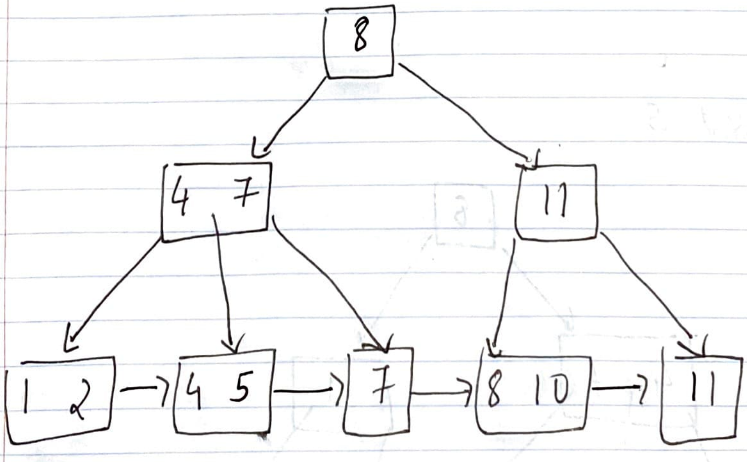
Insert 5



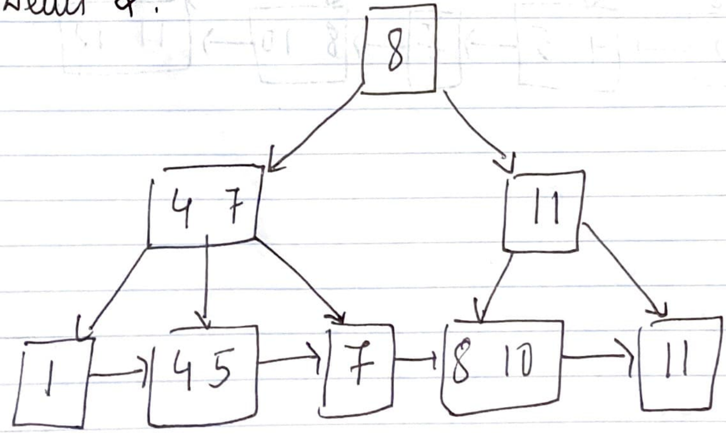
8] b]



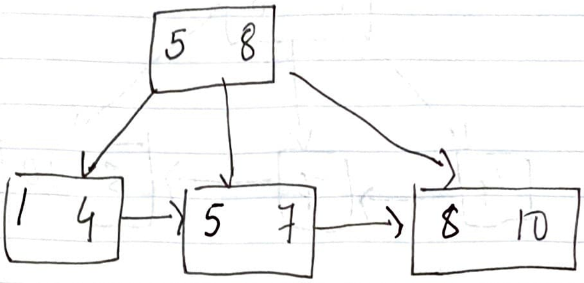
Delete 12

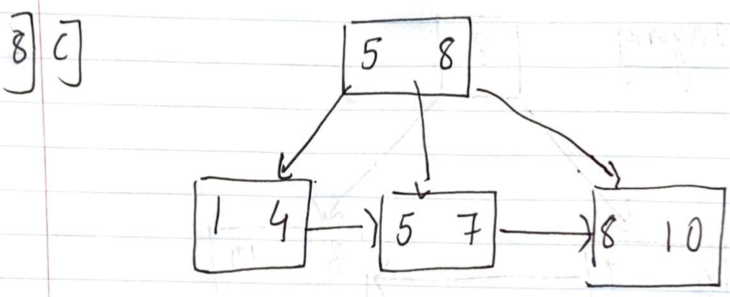


Delete 2 :

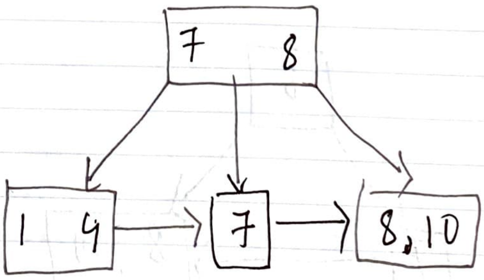


Delete 11

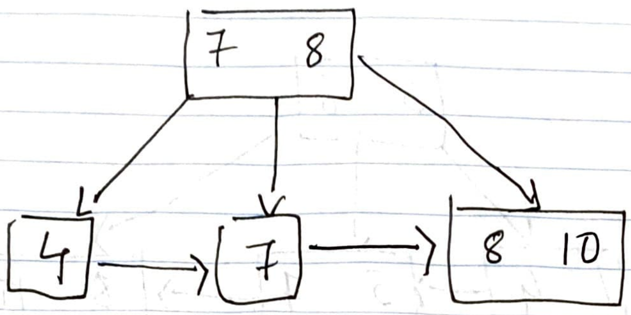




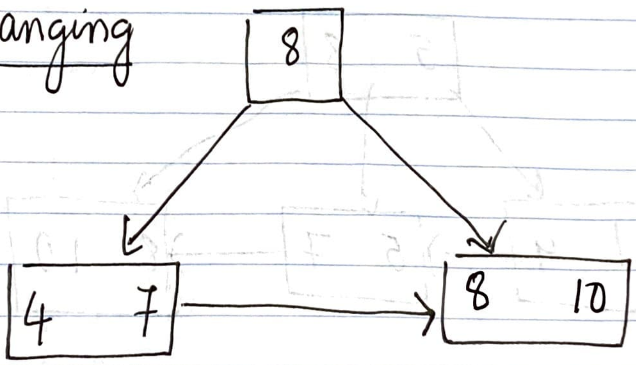
Delete 5



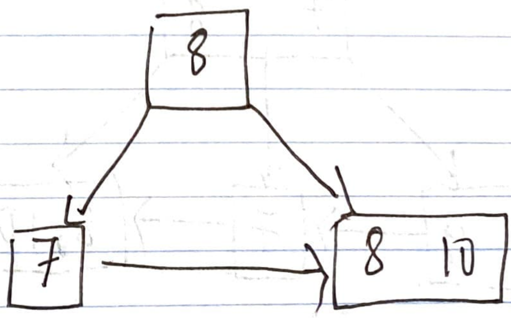
Delete 1



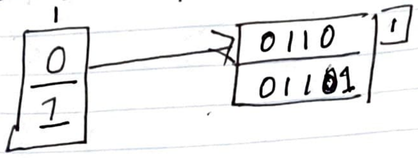
Rearranging



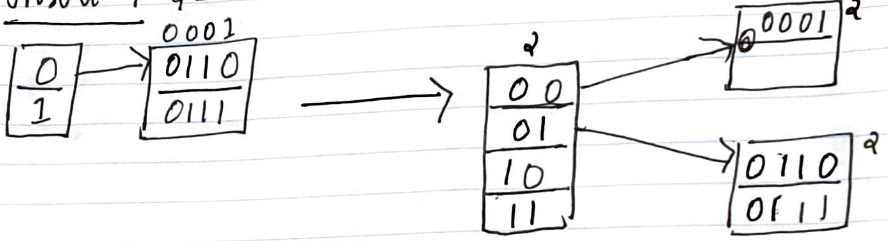
Delete 4



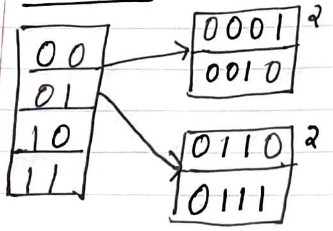
9] 9] Insert 6 & 7



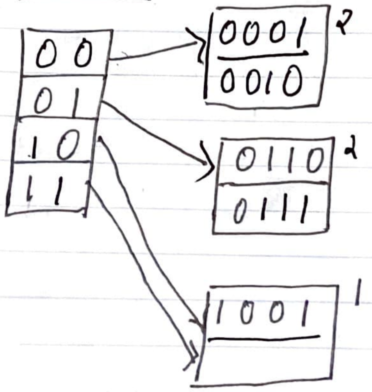
Insert 7 & 1



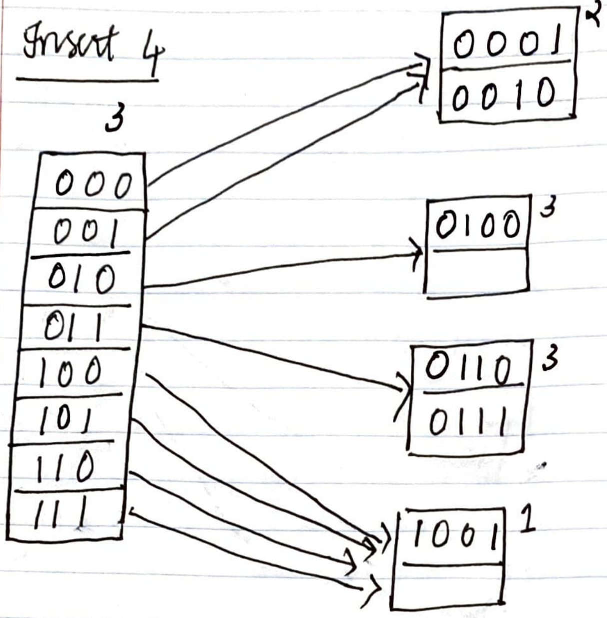
Insert 2



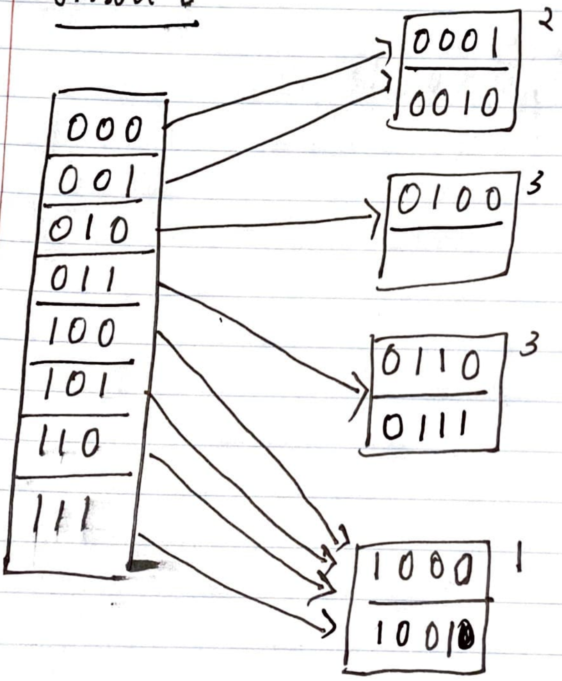
Insert 9



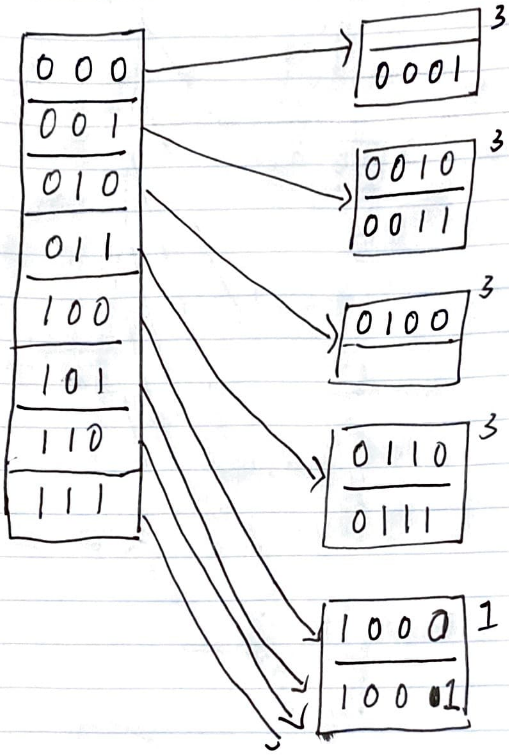
Insert 4



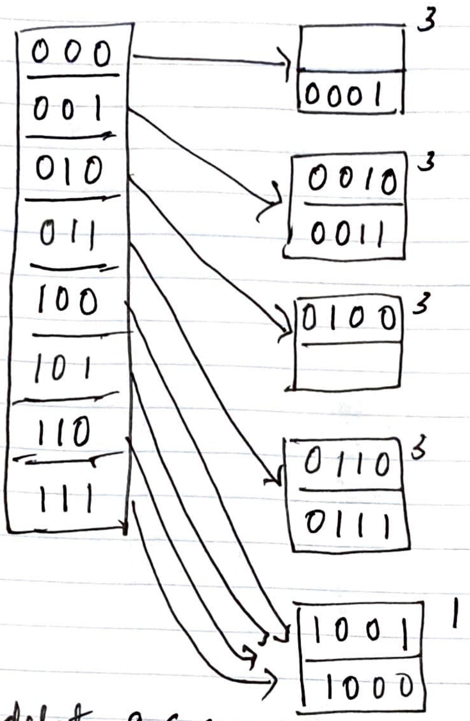
Insert 8



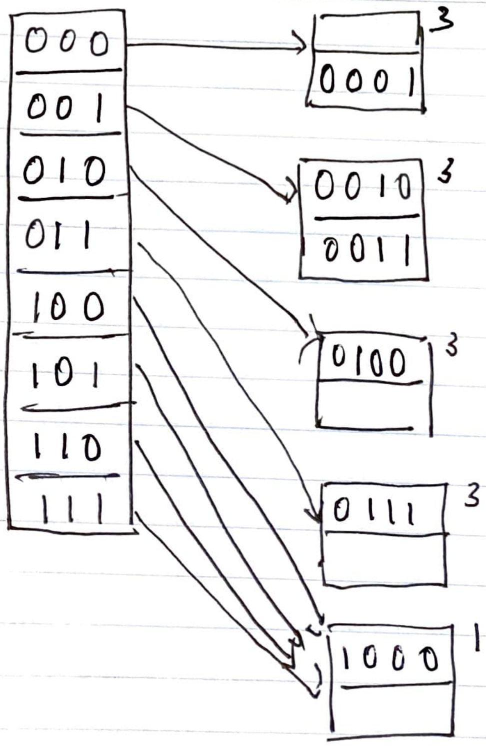
Insert 3


000
001
010
011
100
101
110
111
3
0001
3
0010
0011
3
0100
3
0110
0111
1
1000
1001

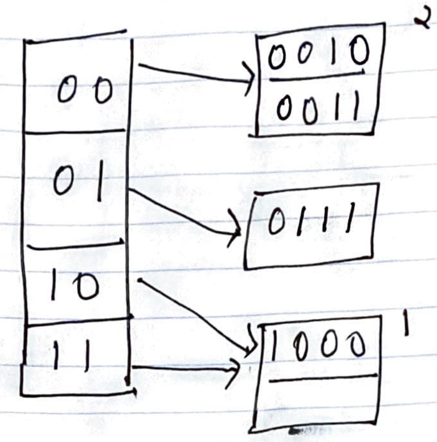
9] b]


000
001
010
011
100
101
110
111
3
0001
3
0010
0011
3
0100
3
0110
0111
1
1001
1000

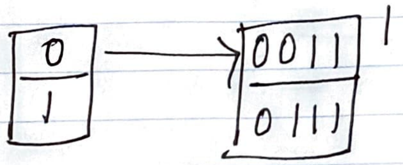
delete 9 & 6


000
001
010
011
100
101
110
111
3
0001
3
0010
0011
3
0100
3
0111
1
1000

Delete 4 and 11



Delete 2 and 8



0
1
0011
0111
1