6.a

block size = 4096 bytes
block-address size = 9 bytes
block access time = 10 ms (micro seconds)
record size = 200 bytes
record key size = 12 bytes
Number of records = 10^8 million = 10^{14}
First, we find the order of the tree as $n \leq \frac{blocksize - |blockaddress|}{|blockaddress| + |key|}$

n <= (4096 -9)/ 9 + 12 = 194.619
n = 194

For calculating minimum height of tree, we need maximum fanout value, i.e. n + 1 = 195

So, the minimum height of tree with N = 10^{14} records
h >= $\log_{(n)}(N)$
h >= $\log_{(195)}(10^{14})$
h >= 6.1134447185
h_{min} = 7

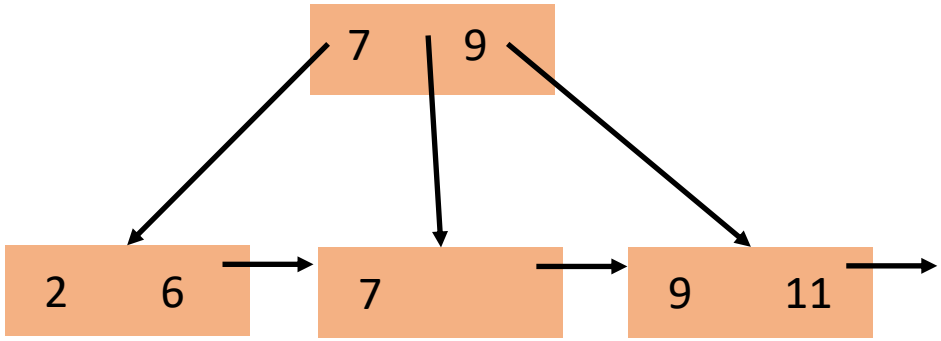
We need one more block access to get record into the data file. So total blocks needed to be accessed will be 7 + 1 and Minimum access to time will be 8 * 10 ms = 80 ms

6.b

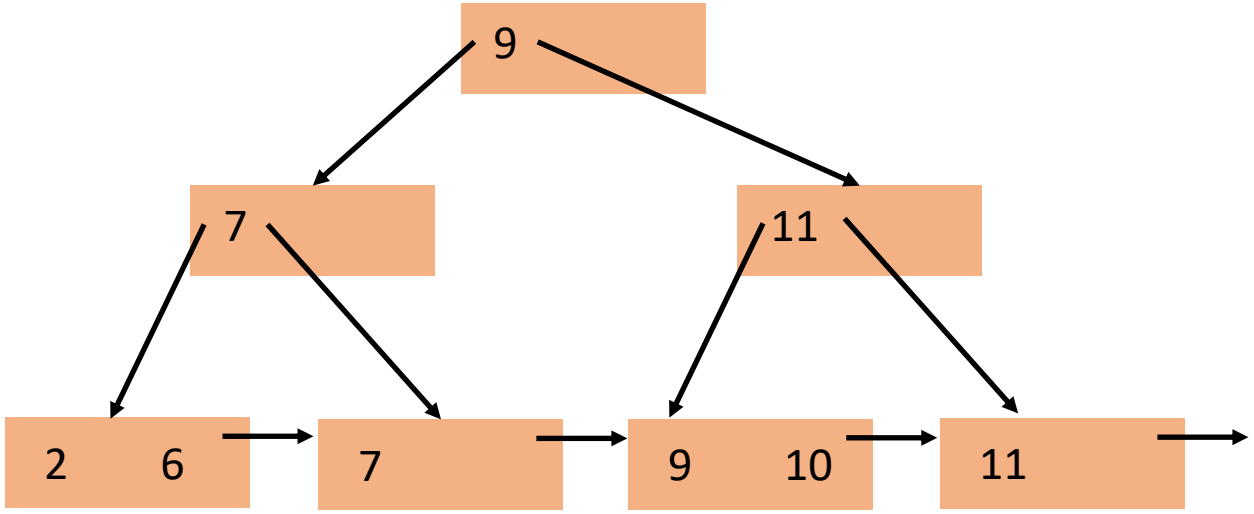
Maximum time is taken when we have minimum branching factor, so the value of n will be
n = ceil(194/2) + 1 = 98
Another point to notice is, we can have minimum branching factor of 2 at the root, so we will need to divide the value of N by 2
N = $10^{14}/2 = 5*10^{13}$.
Height of this tree will be
h >= $\log_{(98)}(5*10^{13})$
h >= 6.8796658358
h_{max} = 7
But we will need one more block access to get the record, so total time of access will be 8*10 = 80 ms.

7.a

#Inserting 6

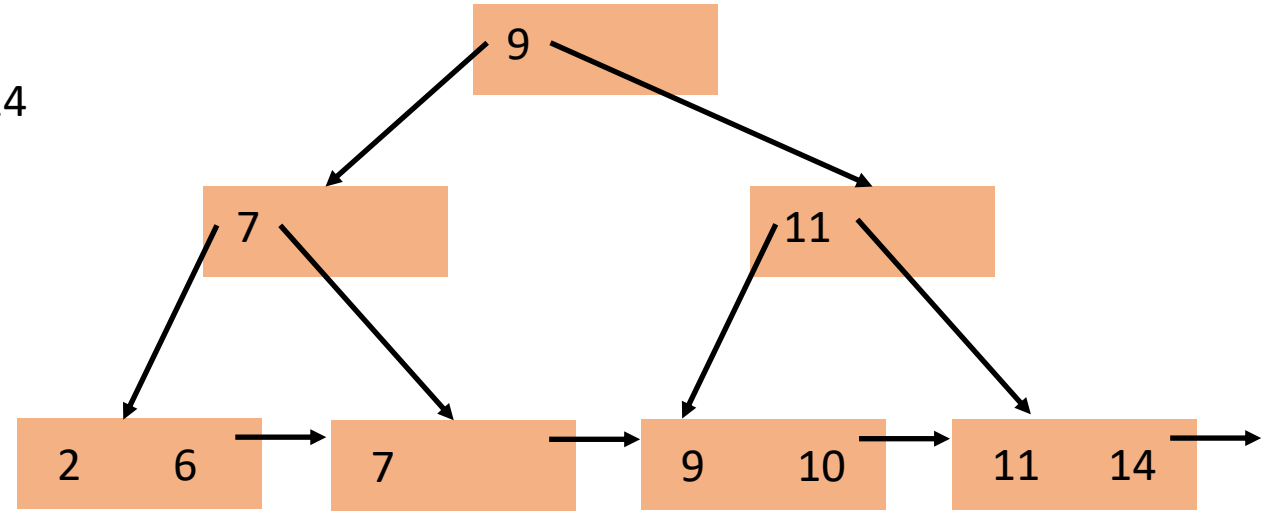


#Inserting 10

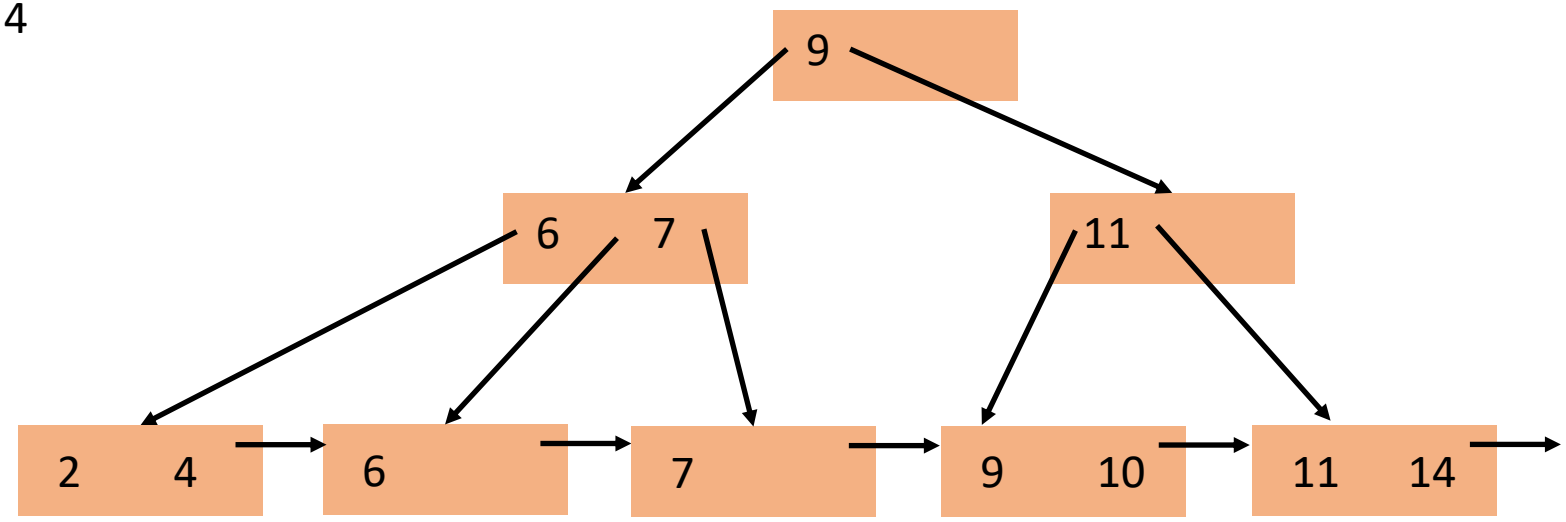


7.a

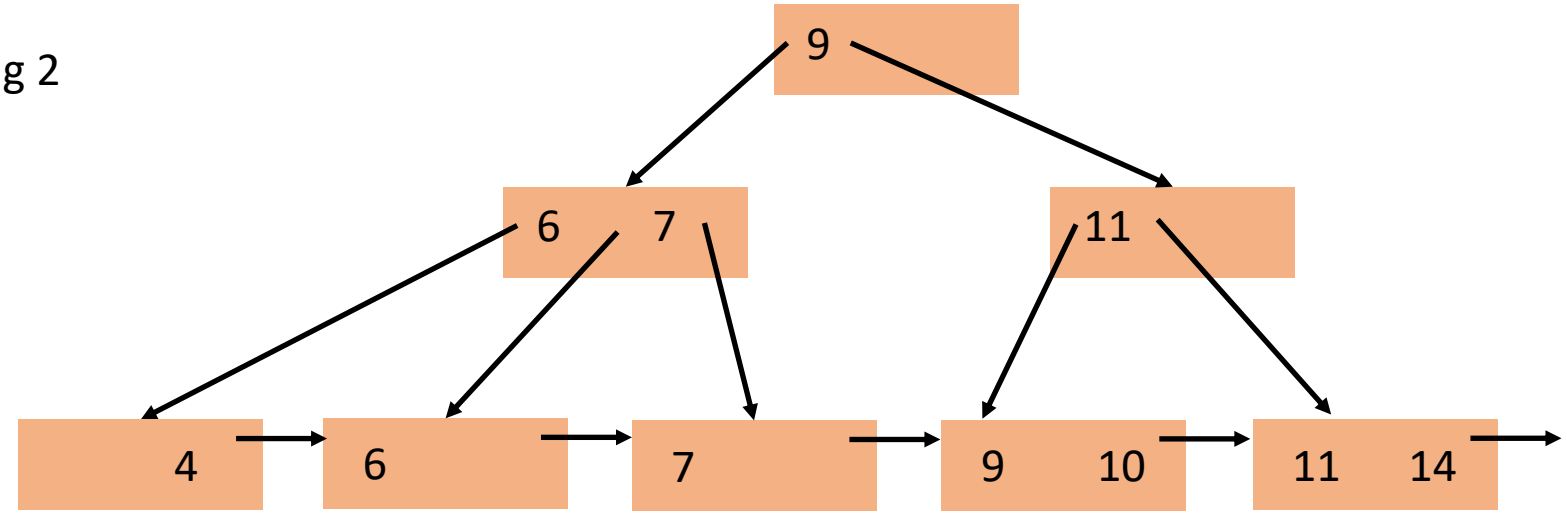
#Inserting 14



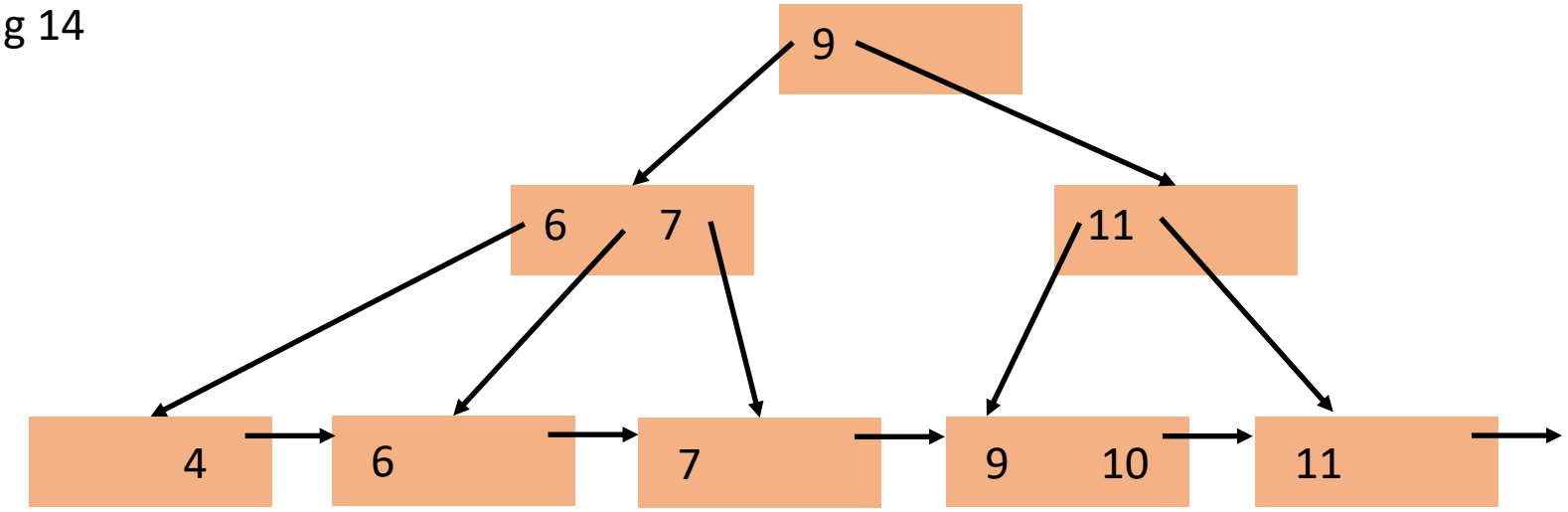
#Inserting 4



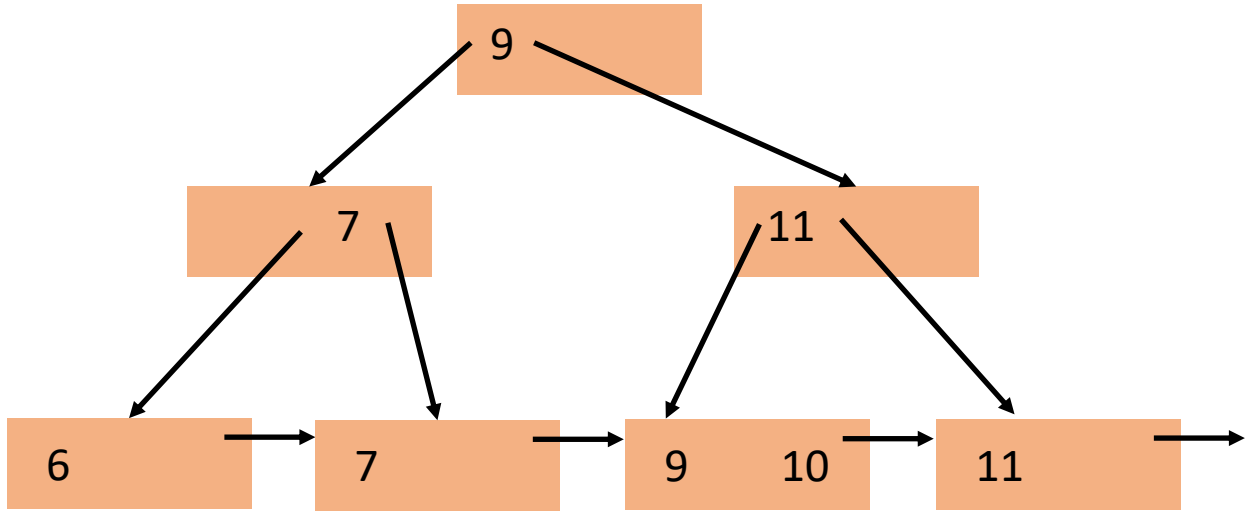
7.b #Deleting 2



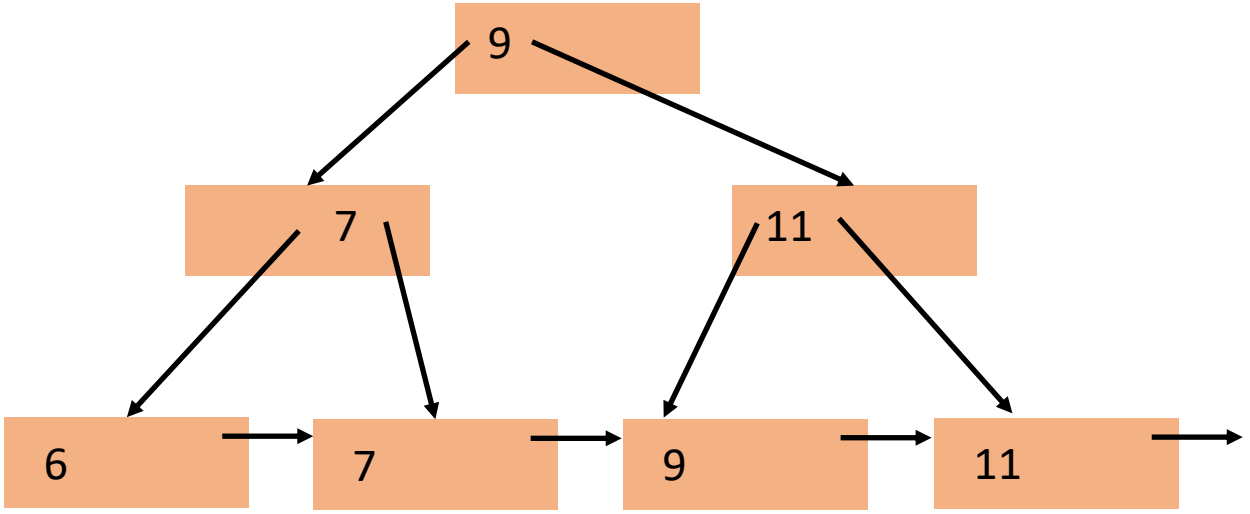
#Deleting 14



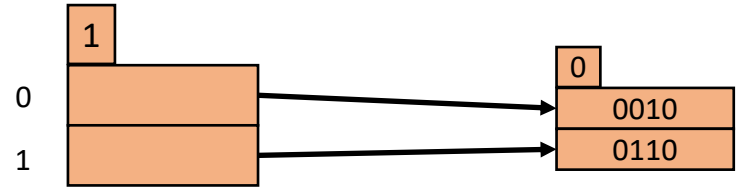
7.b #Deleting 4



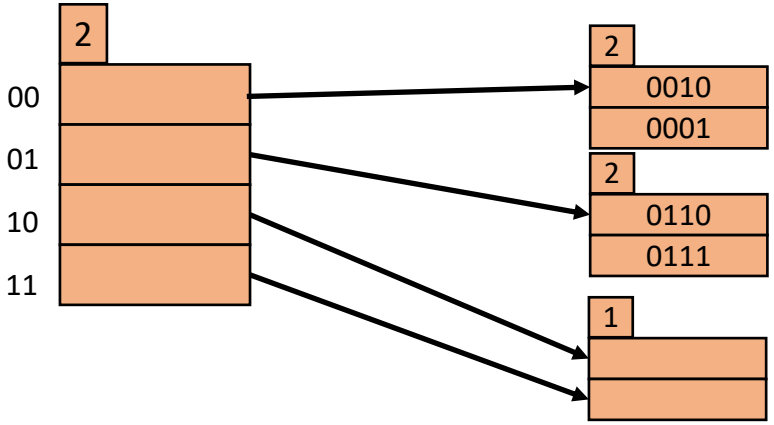
#Deleting 10



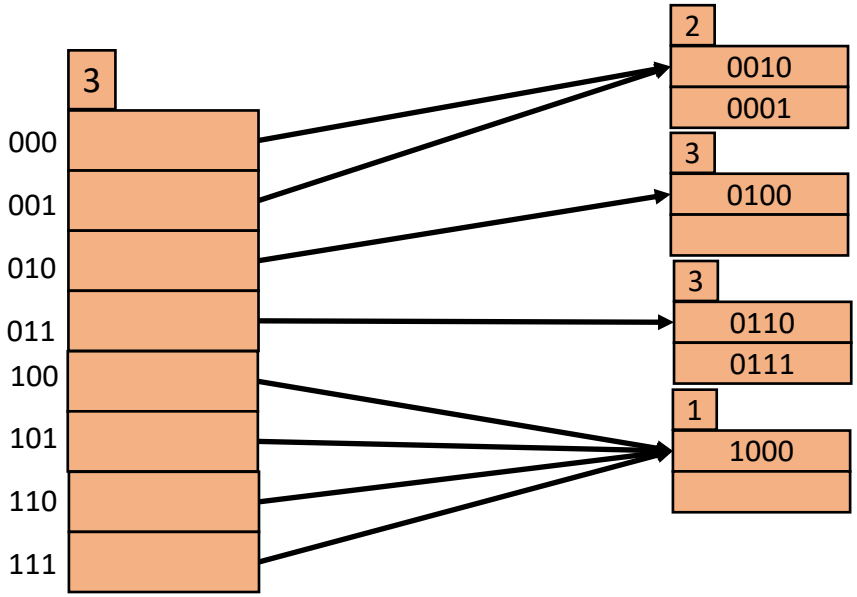
8.A i. inserting records with key 0010, 0110



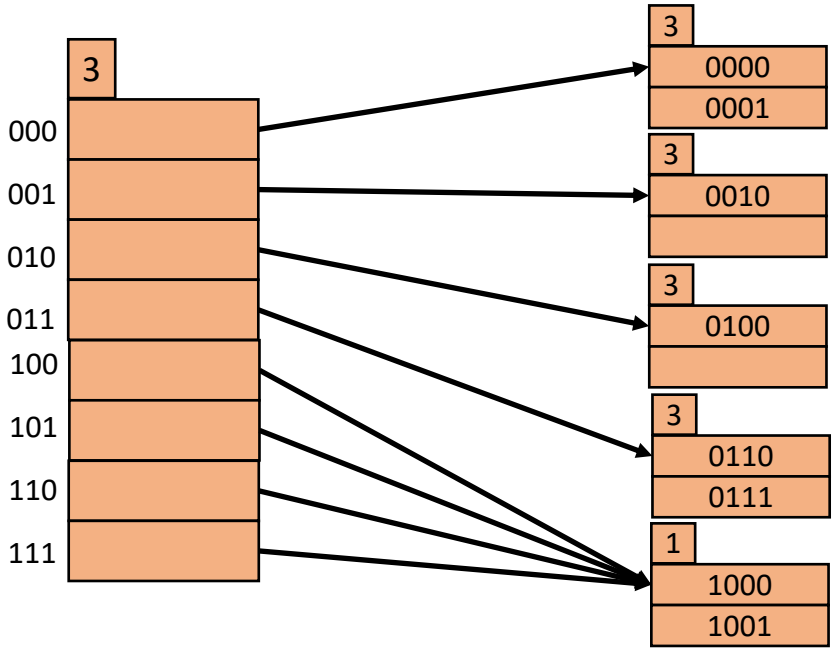
ii. inserting records with key 0001, 0111



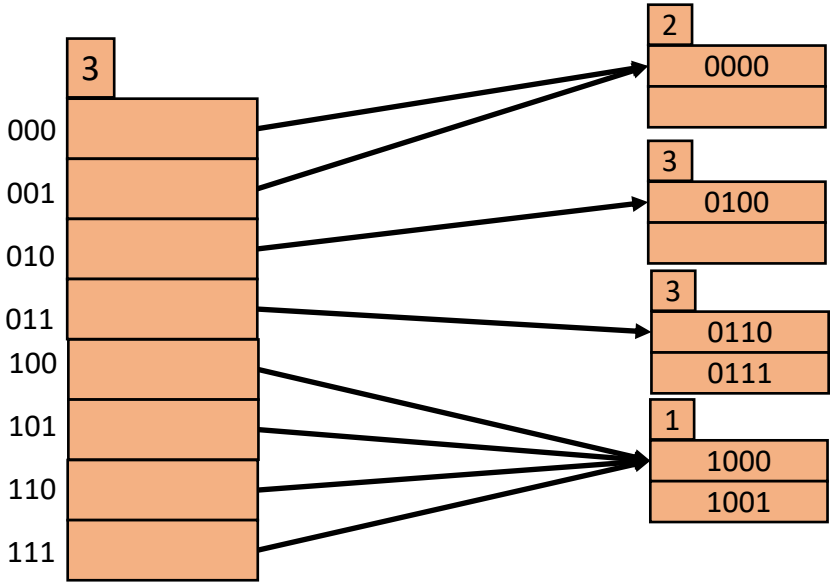
8.A. iii. inserting records with key 0100, 1000



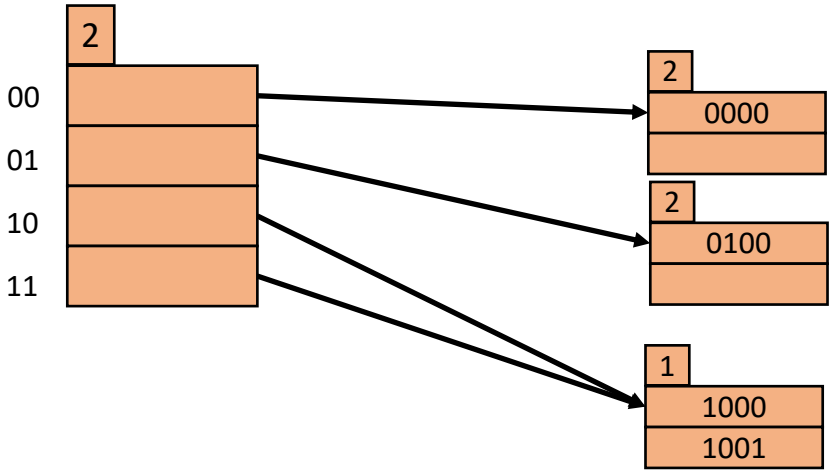
8.A iv. inserting records with key 0000, 1001



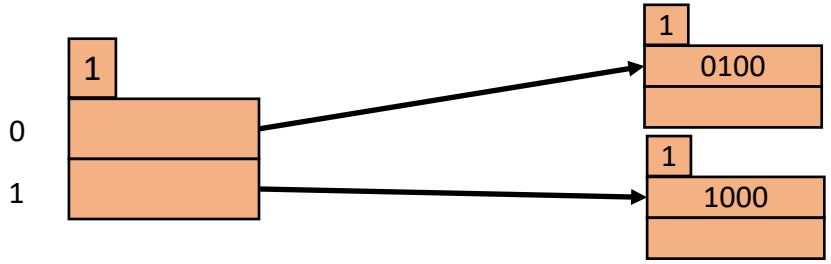
8.B. i. delete records with key 0001, 0010



8.B. ii. delete records with key 0110, 0111



8.B. iii. delete records with key 0000, 1001



9.A Number of records in R 1,500,000, and 30 records of R can fit in one block.
Number of blocks, b(R) = 1,500,000/30 = 50,000 blocks
Number of records in S 5,000 and 10 records of S can fit in one block.
Number of blocks, b(S) = 5,000/10 = 500 blocks
Main memory buffer = 101 blocks
Assuming that we keep one block of memory for output, we will have 100 blocks of memory as buffer memory.
M = 100.

Number of block IO's are necessary to perform R Natural join S using the block nested-loops join algorithm

Taking R as outer
IO = b(R) + (b(R)*b(S))/M
= 50,000 + (50,000 * 500)/ 100
= 50,000 + 250,000
= 300,000

Taking S as outer
IO = b(S) + (b(R)*b(S))/M
= 500 + (50,000 * 500)/ 100
= 500 + 250,000
= 250,500

9.B Number of block IO's are necessary to perform R Natural join S using the sort-merge join algorithm

Without skew
IO= b(R) + b(S) + 2*b(R)*ceil($\log_M$ (b(R))) + 2*b(S)*ceil($\log_M$ (b(S)))
= 50,000 + 500 + 2*50,000 * 3 + 2*500*2
= 352,500

9.c

Since we don't know the value of P in advance, we can't have a tight analysis for number of block IO's that are necessary to perform R natural join S.

First, we will perform sorting for all possible values of P and that would require

IO = $2*b(R)*\text{ceil}(\log_M (b(R))) + 2*b(S)*\text{ceil}(\log_M (b(S)))$

= 2*50,000 * 3 + 2*500*2

= 302,000

Now, for different values of P, we will have to perform block nested-loop join per occurrence of B-value.

So, for P = 1

Considering smaller relation S for the outer loop, we get block nested-loop IO requirement as

IO = b(S) + b(R) * b(S)/ M

= 500 + (50,000 * 500)/ 100

= 500 + 250,000

= 250,500

Total IO = 302,000 + 250,500 = 552,500

So, for P = 2, we will need two block nested loop joins between file size of 50000/2 = 25,000 and 500/2 = 250

Considering smaller relation S for the outer loop, we get block nested-loop IO requirement as

IO = b(S) + b(R) * b(S)/ M

= 250 + (25,000 * 250)/ 100

= 250 + 62,500

= 62,750

Total IO = 302,000 + 2* 62,750 = 427,500

So, for P = 3, we will need three block nested loop joins between file size of 50000/3 =16667 and 500/3=167.

Considering smaller relation S for the outer loop, we get block nested-loop IO requirement as

IO = b(S) + b(R) * b(S)/ M

= 167 + (16667 *167)/ 100

= 167 +27834

= 28,001

Total IO = 302,000 + 3* 28,001 = 386,003

In similar fashion we can calculate depending on different values of P.

9.D Number of block IO's are necessary to perform R Natural join S using the hash join algorithm

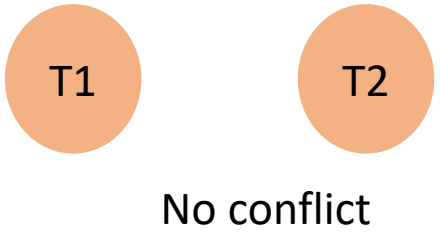
IO = 3*(b(R) + b(S))
= 3*(50,000 + 500)
= 151,500

10.(a). To check conflict serializable, I draw the precedence graph for each schedule and check for the presence of cycle in the graph, If the graph has cycle, then it is not conflict serializable.

S1 = R1(x)R2(y)R1(z)R2(x)R1(y).

There is no conflict between transaction T1 and T2, so This schedule is conflict serializable.

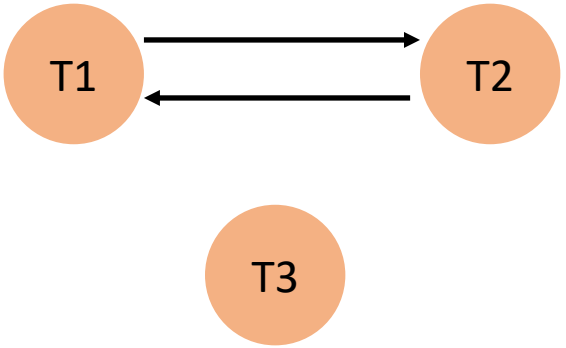
S1' = R1(x)R1(z)R1(y)R2(y)R2(x)



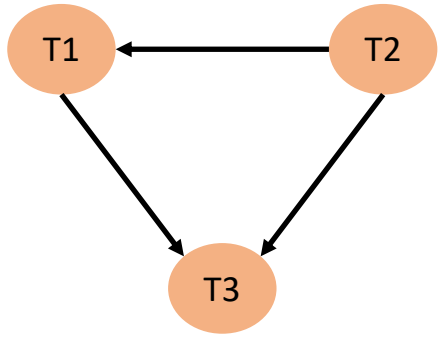
No conflict

10(b) S2 = R1(x)W2(y)R1(z)R3(z)W2(x)R1(y).

From the precedence graph we can see that there is a loop between transaction T1 and T2, so the presence of loop indicates that this schedule is not conflict-serializable.



10(c) S3 = R1(z)W2(x)R2(z)R2(y)W1(x)W3(z)W1(y)R3(x).



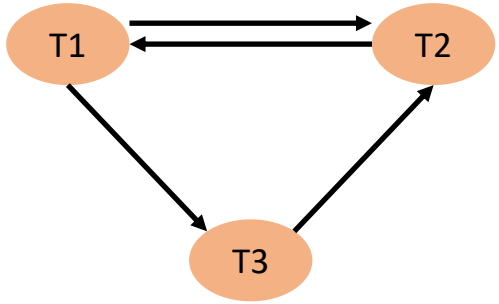
From the precedence graph we see that there is no cycle, which Means that this schedule is conflict-serializable.

S3' = T2; T1; T3.
= W2(x)R2(z)R2(y) R1(z) W1(y) W3(z) R3(x).

11. Edges (T1, T2), (T2, T1), (T1, T3), (T3, T2).

T1 = R1(x) R1(Y) R1(z)
T2 =W2(y) W2(x)
T3 = R3(z) R3(x) W3(x)

S = R1(x)R3(x)W3(x)w2(y)R1(z)R3(z)W2(x)R1(y)



Corresponding precedence graph

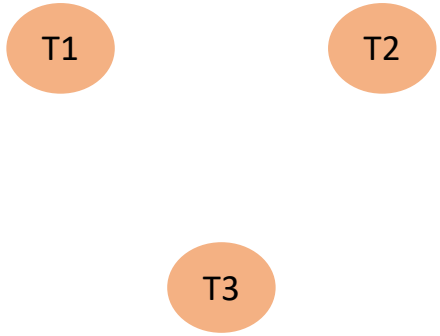
12. Schedule S which is in conflict-equivalence with all serial schedules for transaction T1, T2 and T3.

We can define transaction on different variables like this:
T1 = R1(x) w1(x)
T2 = R2(y) w2(y)
T3 = R3(z) w3(z) and
The serial schedule which is conflict equivalent to all serial schedules with T1, T2 and T3 is
S = R1(x) w1(x)R2(y) w2(y)R3(z) w3(z)

The serial schedule S is conflict equivalence with all serial schedules given below:

S1 = R1(x) w1(x)R2(y) w2(y)R3(z) w3(z)
S2 = R1(x) w1(x) R3(z) w3(z) R2(y) w2(y)
S3 = R3(z) w3(z) R2(y) w2(y) R1(x) w1(x)
S4 = R3(z) w3(z) R1(x) w1(x) R2(y) w2(y)
S5 = R2(y) w2(y) R1(x) w1(x) R3(z) w3(z)
S6 = R2(y) w2(y) R3(z) w3(z) R1(x) w1(x)

All of these are serializable to S.



Corresponding precedence graph for any
Of the Schedules from S1 to S6

13.a Consistency requirement: A=0 or B=0

There can be 2! ways of writing serial schedules including as given below:
S1 = T1 T2 and
S2 = T2T1.
The proof is shown below:

Schedule S1 = T1T2

T1	T2

read(A);
read(B);
if A = 0 then B := B+1;
write(B).	

	read(B);
	read(A);
	if B = 0 then A := A+1;
	write(A).

A	B
0	0
0	1
0	1
0	1

Consistency requirement satisfied.

This serial schedules serves the consistency requirement of the database.

13.a Consistency requirement: A=0 or B=0

Schedule S2 = T2T1

T1	T2
	read(B);
	read(A);
	if B = 0 then A := A+1;
	write(A).
read(A);	
read(B);	
if A = 0 then B := B+1;	
write(B).	

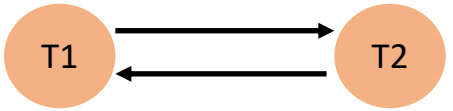
A	B
0	0
1	0
1	0
1	0

Consistency requirement satisfied.

Both the serial schedules serves the consistency requirement of the database.

13.B A non-serializable schedule involving T1 and T2.

S = R1(A) R2(B) R2(A) W2(A) R1(B) W1(B)



Corresponding precedence graph.

T1	T2
read(A);	

	read(B);
	read(A);
	if B = 0 then A := A+1;
	write(A).

read(B);
if A = 0 then B := B+1;
write(B).	

13.c

No, there is no non-serial schedule on T1 and T2, which can produce serializable results. For these transaction, T1 and T2, it is only possible to produce a serializable result if we swap the order of operations in T1 and T2, but this should not be done as it will change the semantic of transaction in general.