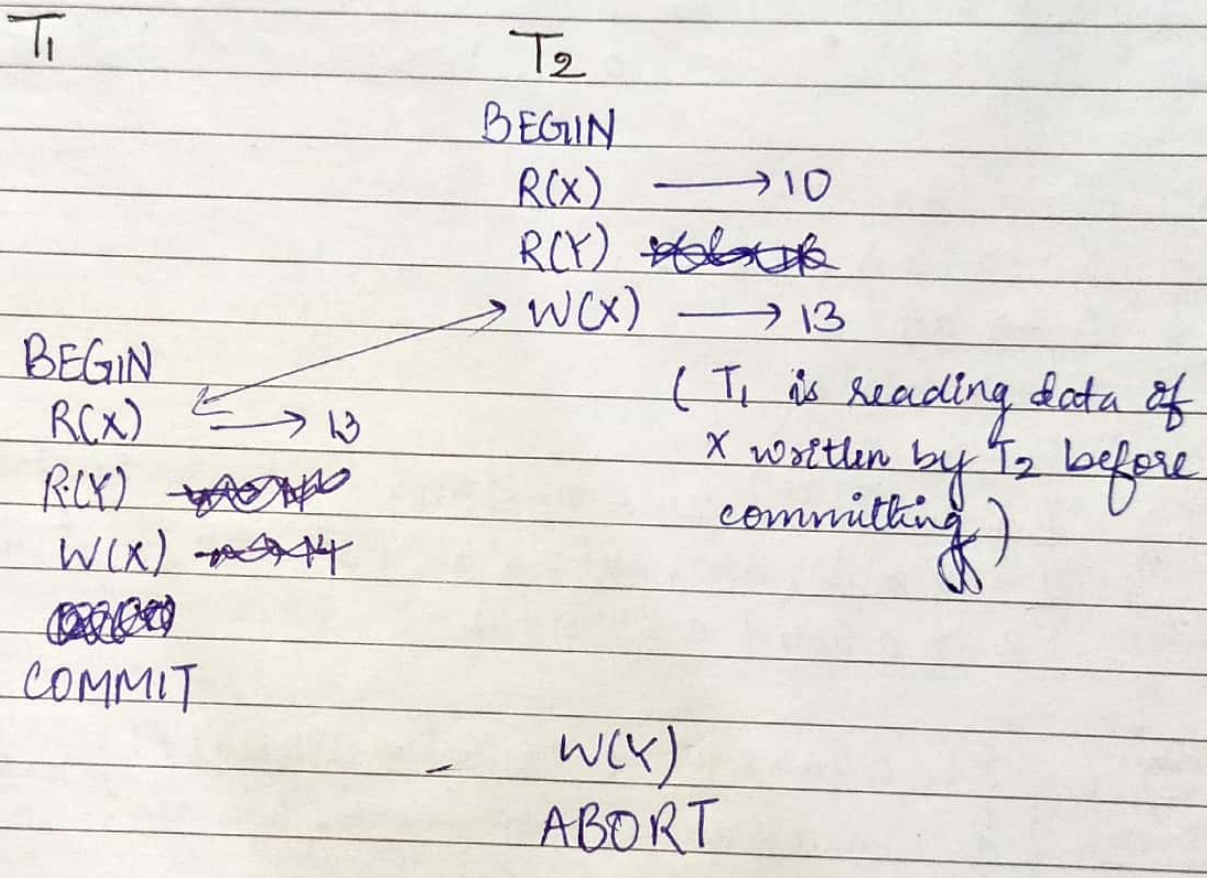
DBMS Assignment-3

Date: / /

Que-1 Consider a database with 2 objects X and Y and 2 transactions T_1 and T_2. T_1 performs the following operations : R(X), R(Y), W(X). T_2 performs the following operations : R(X), R(Y), W(X), W(Y).

i) Give an example schedule with T_1 and T_2 that results in a write-read conflict.



(ii) Give an example schedule that results in read-write conflict.

T_1	T_2
	BEGIN
	R(X)
	R(Y)
	W(X)
BEGIN	
R(X)	
R(Y) ←	
	→ W(Y)
	COMMIT.
W(X)	
COMMIT	

(iii) Give an example schedule that results in write-write conflict

T_1	T_2
	BEGIN
	R(X)
	R(Y)
BEGIN	
R(X)	
R(Y)	
	W(X)
W(X) ←	
COMMIT	~~COMM~~
	W(Y)
	COMMIT

(iv) for each of the three schedules, show that strict 2PL disallows the schedule.

If we use strict 2PL the locking and unlocking will be done in 2 phases and all the exclusive locks will be held till the transaction commits or aborts and shared locks can be released anytime during the second phase. If we apply strict 2PL, only serializable schedules will be allowed and all the 3 schedules listed above will be disallowed. Let us denote taking exclusive lock by x and shared lock by s, and release of lock by u, and commit by c. So, following will be the execution of schedules where strict 2PL will ensure serializability by not granting locks which may create conflicts.

1) $S_2(X), R_2(X), S_2(Y), R_2(Y), X_2(X), W_2(X), S_1(X)$ - request not granted, $X_2(Y), W_2(Y), U_2(X), U_2(Y), C_2$

$S_1(X), R_1(X), S_1(Y), R_1(Y), X_1(X), W_1(X), U_1(X), U_1(Y), C_1$

(2 → for T_2, 1 for T_1)

2) $S_2(x)$, $R_2(x)$, $S_2(Y)$, $R_2(Y)$, $X_2(x)$, $W_2(x)$, $S_1(x)$ – Request not granted, $X_2(Y)$, $W_2(Y)$, $U_2(x)$, $U_2(Y)$, C_2, $S_1(x)$, $R_1(x)$, $S_1(Y)$, $R_1(Y)$, $X_1(x)$, $W_1(x)$, $U_1(x)$, $U_1(Y)$, C_1, $S_2(Y)$,

3) $S_2(x)$, $R_2(x)$, $R_2(Y)$, $S_1(x)$ – Request not granted, $X_2(x)$, $W_2(x)$, $X_2(Y)$, $W_2(Y)$, $U_2(x)$, $U_2(Y)$, C_2, $S_1(x)$, $R_1(x)$, $S_1(Y)$, $R_1(Y)$, $X_1(x)$, $W_1(x)$, $U_1(x)$, $U_1(Y)$, C_1

Que=2 Consider the following classes of schedules: serializable, conflict serializable, view-serializable, recoverable, avoids-cascading-aborts, and strict. Classify the below schedules in one of the classes.

1) $R_1(x)$, $R_2(x)$, $W_1(x)$, $W_2(x)$

Not serializable, not conflict-serializable, not view-seriali-zable; It is recoverable and avoid cascading aborts; not strict.

2) $W_1(x)$, $R_2(Y)$, $R_1(Y)$, $R_2(x)$

It is serializable, conflict-serializable, and view-seriali-zable; It does not avoid cascading aborts, is not strict; We cannot decide whether it is recoverable or not, since the abort / commit sequence of these 2 transactions are not specified.

3) $R_1(x)$, $R_1(Y)$, $W_1(x)$, $R_2(Y)$, $W_3(Y)$, $W_1(x)$, $R_2(Y)$

It is not serializable, not conflict serializable, not view-serializable; It does not avoid cascading aborts, not strict; We cannot decide whether it's recoverable or not, since the abort / commit sequence of these transactions are not specified.

4) $R_1(X)$, $W_2(X)$, $W_1(X)$, $Abort_2$, $Commit_1$
It is serializable, conflict-serializable, and view-Serializable; It is recoverable and avoid cascading aborts; It is not strict

5) $R_1(X)$, $W_2(X)$, $W_1(X)$, $Commit_2$, $Commit_1$
It is serializable and view serializable, not conflict-serializable; It is recoverable and avoid cascading aborts; It is not strict.

6) $W_1(X)$, $R_2(X)$, $W_1(X)$, $Commit_2$, $Abort_1$
It is not serializable, not view-serializable, not conflict-Serializable; It is not recoverable, therefore not avoid cascading aborts, not strict.

7) $R_2(X)$, $W_3(X)$, $Commit_3$, $W_1(Y)$, $Commit_1$, $R_2(Y)$, $W_2(Z)$, $Commit_2$
It belongs to all above classes.

8) $R_1(X)$, $W_2(X)$, $W_1(X)$, $R_3(X)$, $Commit_1$, $Commit_2$, $Commit_3$
It is serializable and view-serializable, not conflict-Serializable; It is recoverable, but not avoid cascading aborts, not strict.

Que-3 Consider the following lock requests in Table.
SC.) and XC.) stand for 'shared lock' and 'exclusive lock' respectively.

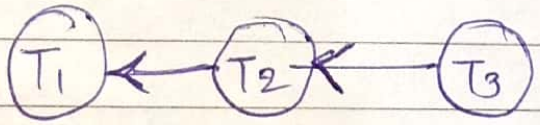
Lock Requests of three transactions: T_1, T_2 and T_3

Time	t_1	t_2	t_3	t_4	t_5	t_6	t_7
T_1	X(A)						S(C)
T_2			S(B)	S(C)		S(A)	
T_3		S(C)			X(B)		
LM	G	g	g	g	b	b	g

i) for the lock requests in Table 1 determine which lock will be granted or blocked by the lock manager.

- S(C) at t_2 : g.
- S(B) at t_3 : g
- S(C) at t_4 : g
- X(B) at t_5 : b
- S(A) at t_6 : b
- S(C) at t_7 : g

ii) Give the wait-for graph for the lock requests in Table at time tick t_7



(iii) Determine whether there exists a deadlock in the lock requests in Table, and explain why.

A deadlock does not exists because there is no cycle in the dependency graph.

Que-4 Consider the following lock requests in Table 2

Lock requests for four transactions : T_1, T_2, T_3, T_4

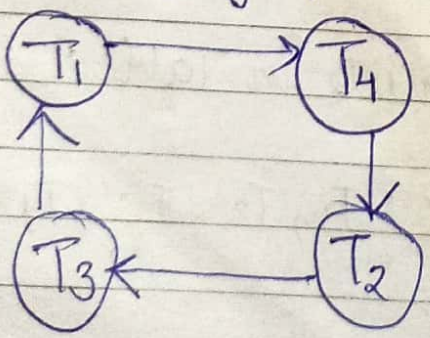
Date: / /

Time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
T_1	X(B)			S(A)				
T_2					X(D)	X(C)		
T_3			S(C)				X(B)	
T_4		X(A)						S(D)
LM	G	g	g	b	g	b	b	b

I) For the lock requests in Table 2, determine which lock request will be granted, blocked or aborted by the lock manager (LM), if it has no deadlock prevention policy.

- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : b
- X(D) at t_5 : g
- X(C) at t_6 : b
- X(B) at t_7 : b
- S(D) at t_8 : b

II) Give the wait-for graph for the lock requests in Table 2. Determine whether there exists a deadlock in the lock requests in Table 2 under LM, and explain why.
A deadlock exists because there is a cycle in the dependency graph.



III) To prevent deadlock, we use a lock manager (LM) that adopts the wait-Die policy. We assume the four transactions have priority: $T_1 < T_2 < T_3 < T_4$. Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a').

- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : a (T_1 aborts)
- X(D) at t_5 : g
- X(C) at t_6 : a (T_2 aborts)
- X(B) at t_7 : g (No locks held on B since T_1 was aborted)
- S(D) at t_8 : g (No locks held on D since T_2 was aborted)

IV) In this question, we use a lock manager (LM) that adopts the wound-wait policy. We assume the four transactions have priority: $T_1 < T_2 < T_3 < T_4$. Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a'); for 'abort'.

- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : b
- X(D) at t_5 : g
- X(C) at t_6 : b
- X(B) at t_7 : g (T_1 aborts)
- S(D) at t_8 : g (T_2 aborts)

Que=5 Perform the following operations in Btree and B+ tree

The tree can hold up to 4 pointers and 3 keys

a) Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

III) To prevent deadlock, we use a lock manager (LM) that adopts the wait-Die policy. We assume the four transactions have priority: $T_1 < T_2 < T_3 < T_4$. Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a').

- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : a (T_1 aborts)
- X(D) at t_5 : g
- X(C) at t_6 : a (T_2 aborts)
- X(B) at t_7 : g (No locks held on B since T_1 was aborted)
- S(D) at t_8 : g (No locks held on D since T_2 was aborted)

IV) In this question, we use a lock manager (LM) that adopts the wound-wait policy. We assume the four transactions have priority: $T_1 < T_2 < T_3 < T_4$. Determine which lock request will be granted ('g'), blocked ('b') or aborted ('a'); for 'abort'.

- X(A) at t_2 : g
- S(C) at t_3 : g
- S(A) at t_4 : b
- X(D) at t_5 : g
- X(C) at t_6 : b
- X(B) at t_7 : g (T_1 aborts)
- S(D) at t_8 : g (T_2 aborts)

Que:5 Perform the following operations in Btree and B+ tree

The trees can hold up to 4 pointers and 3 keys

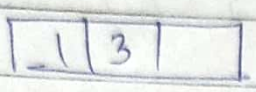
a) Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

Date: / /

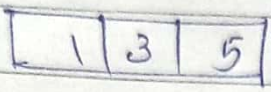
Btree

Insert 1

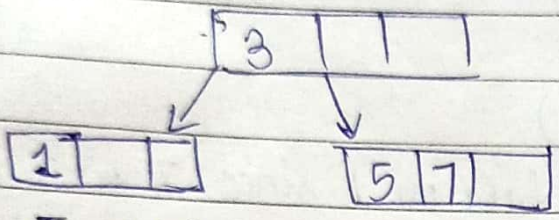
Insert 3



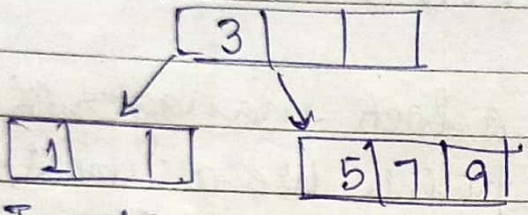
Insert 5



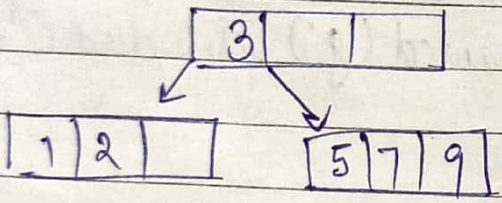
Insert 7



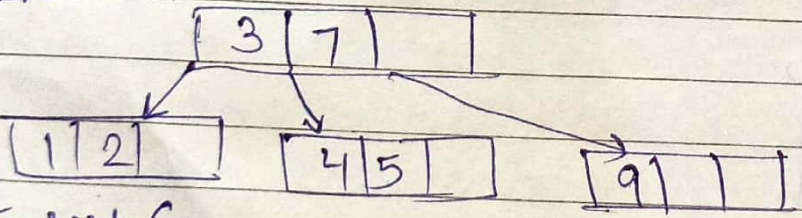
Insert 9



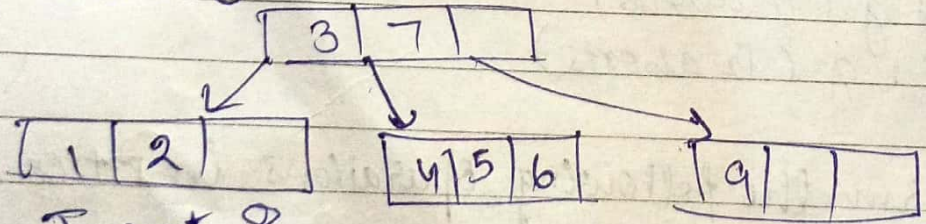
Insert 2



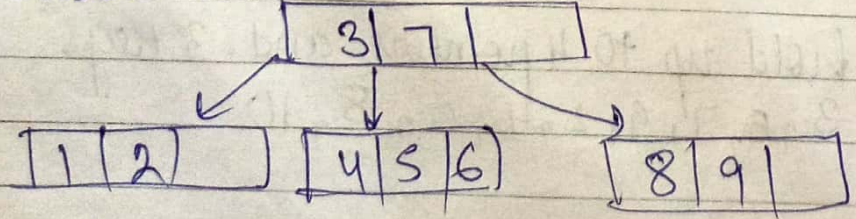
Insert 4



Insert 6

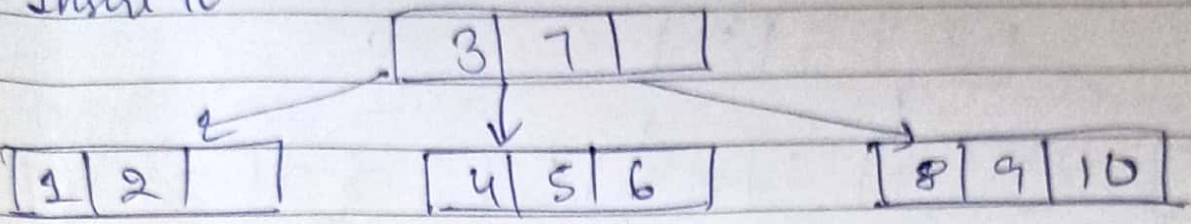


Insert 8



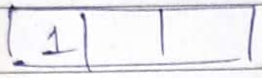
Date: / /

Insert 10

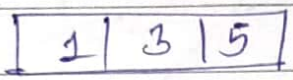


B+ tree

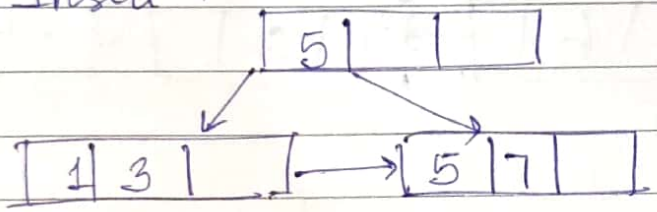
Insert 1



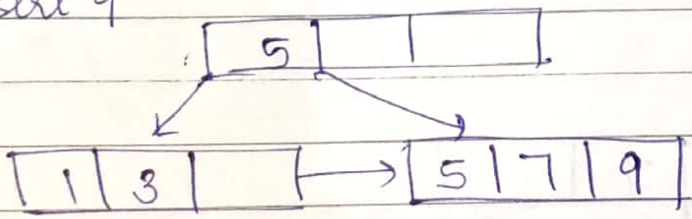
Insert 3,5



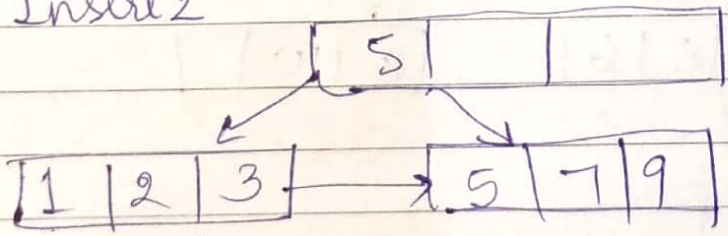
Insert 7



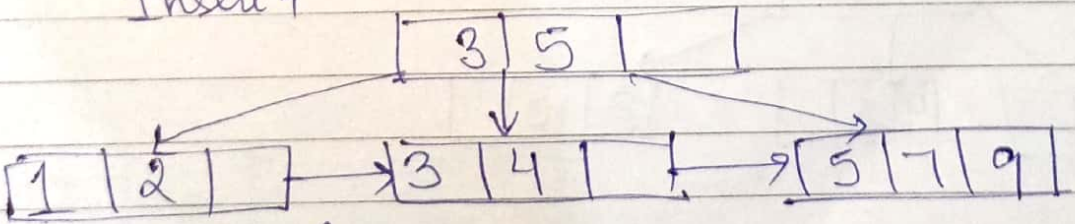
Insert 9



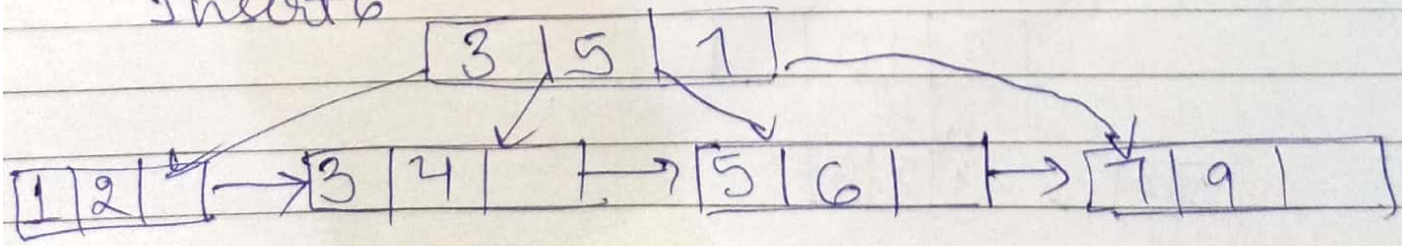
Insert 2



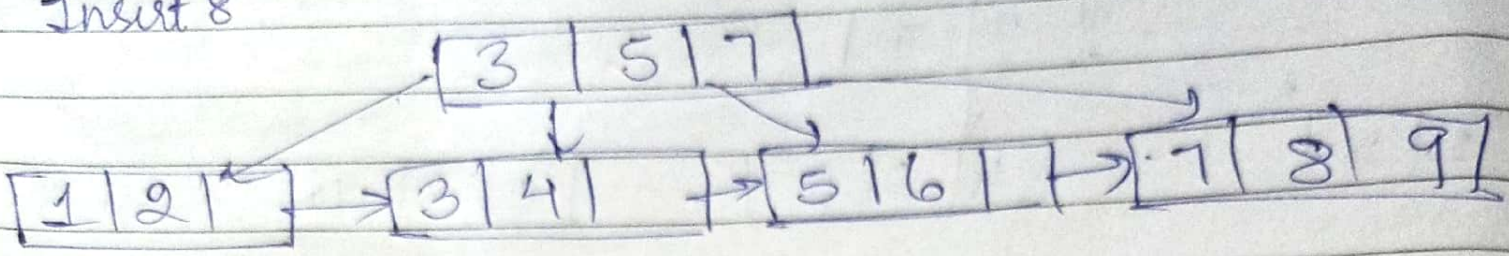
Insert 4



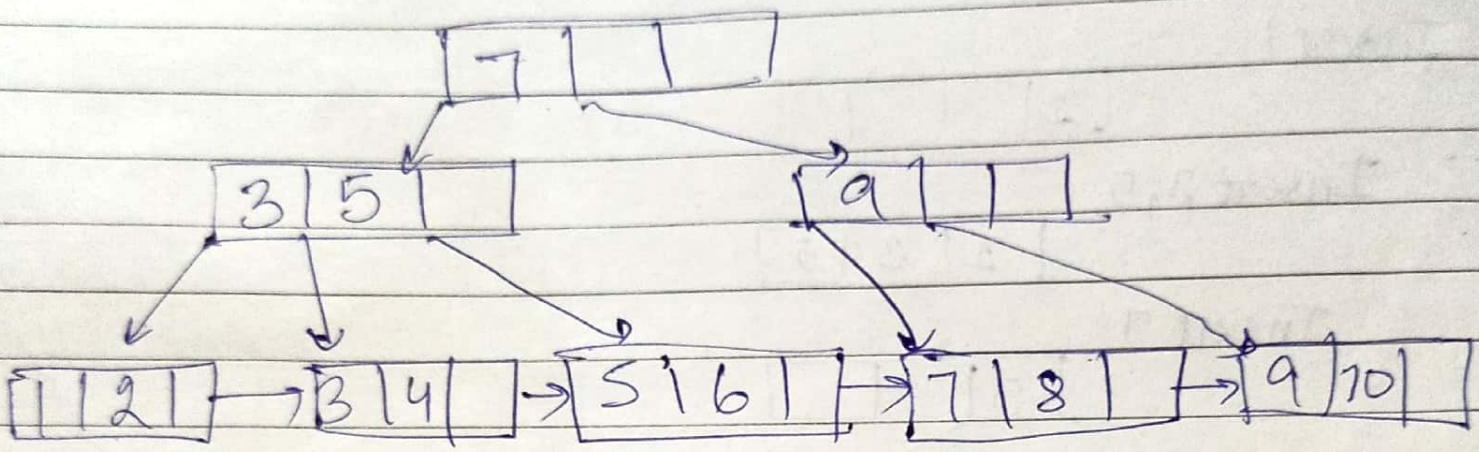
Insert 6



Insert 8

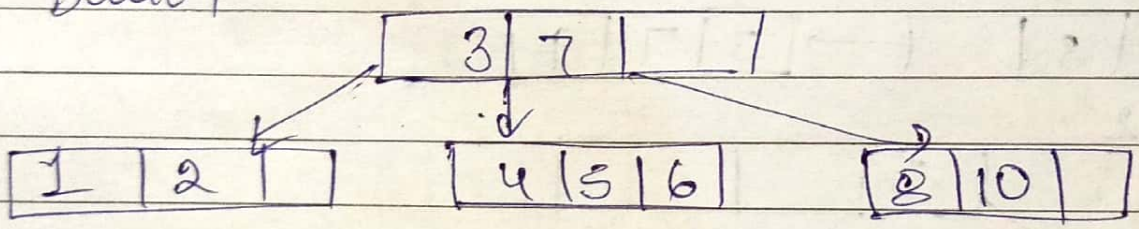


Insert 10

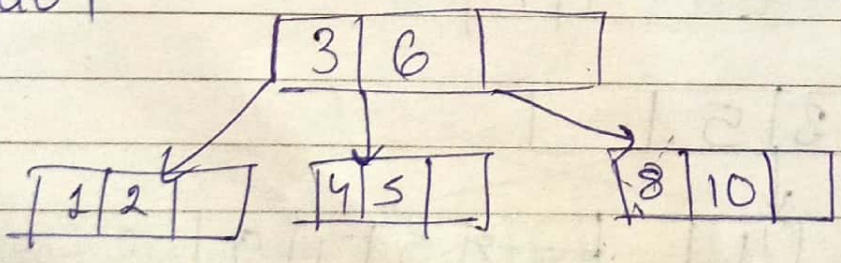


b) Delete 9, 7, 8

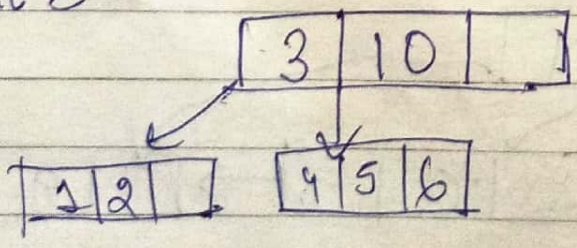
B tree
Delete 9



Delete 7

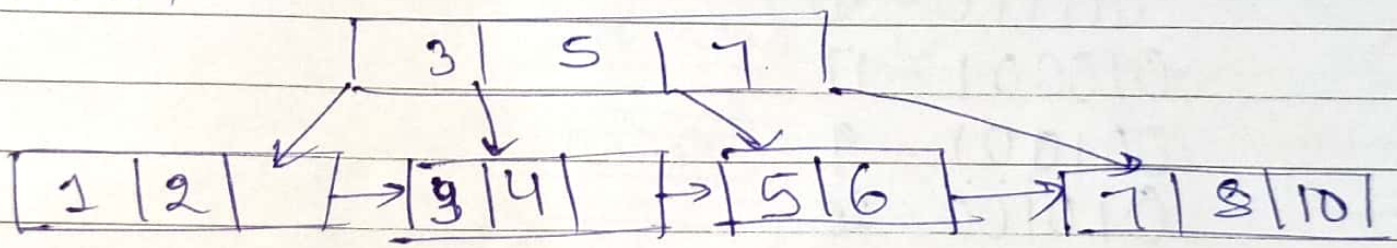


Delete 8

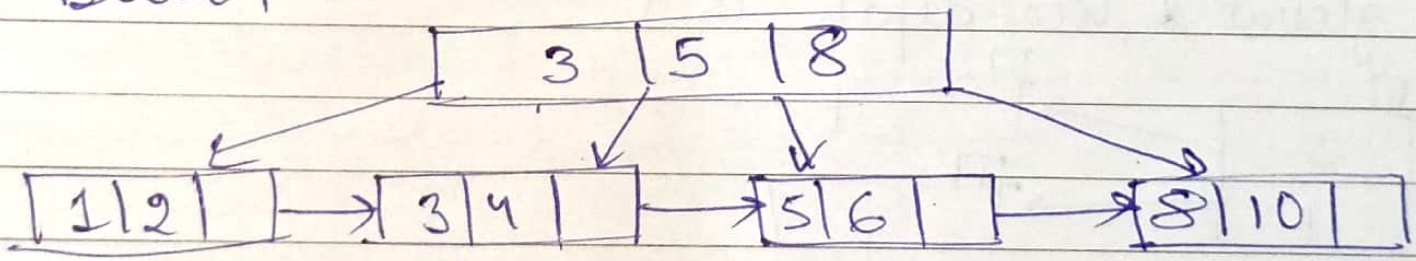


B+ tree

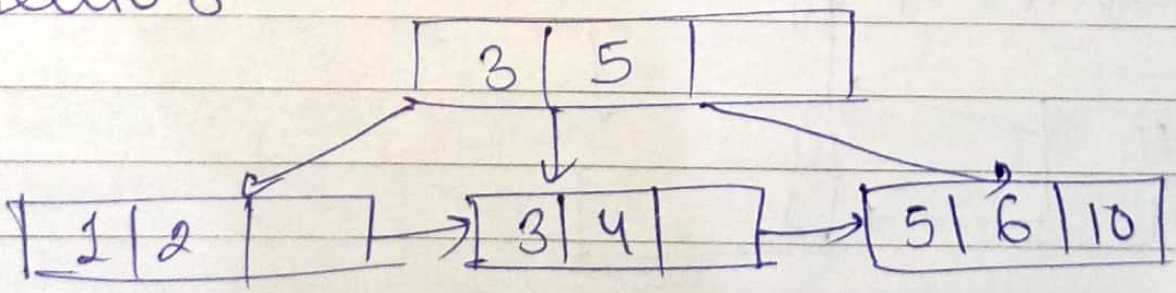
Delete 9



Delete 7



Delete 8



Que=6 Hash the following elements using extendible hashing : 1, 40, 61, 2, 4, 15, 30, 17, 9, 20, 26

Assumptions,

i) Bucket size = 3

ii) Suppose the global depth is X. Then the hash function returns X least significant bits.

Converting into binary form

1 → 000001 - 1

40 → 101000 - 40

111101 - 61

000010 - 2

000100 - 4

001111 - 15

011110 - 30

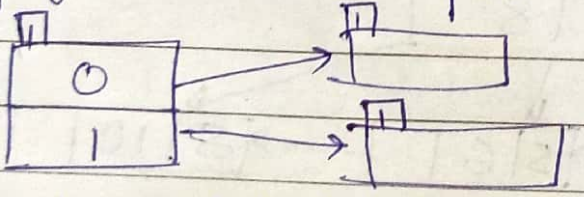
010001 - 17

001001 - 9

010100 - 20

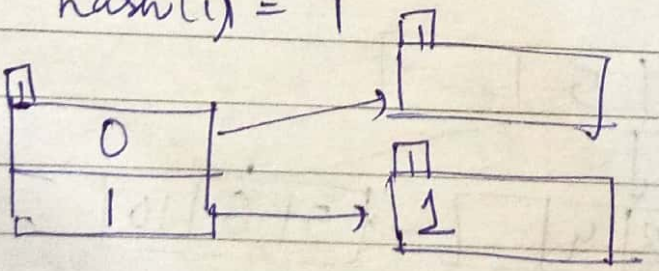
011010 - 26

Initially global & local depth is 1



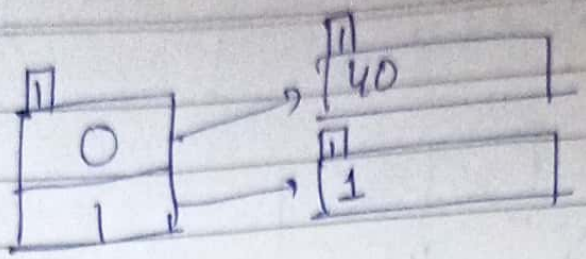
Inserting 1, global depth is 1

hash(1) = 1

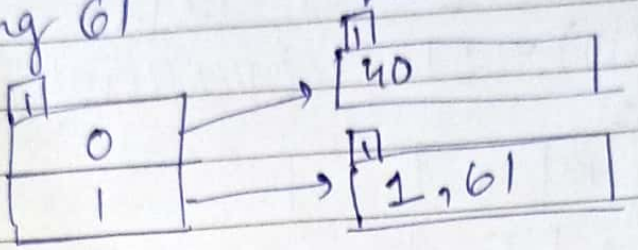


Inserting 40

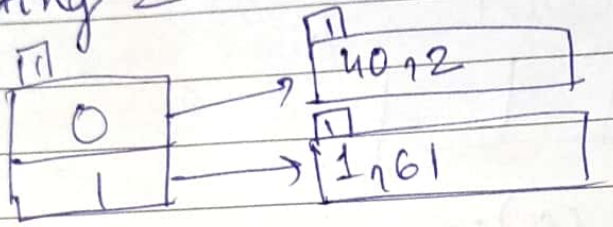
hash(40) = 0



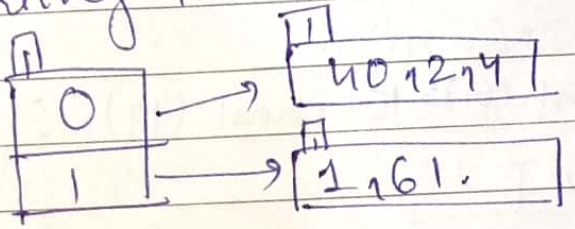
Inserting 61 hash(61) = 1



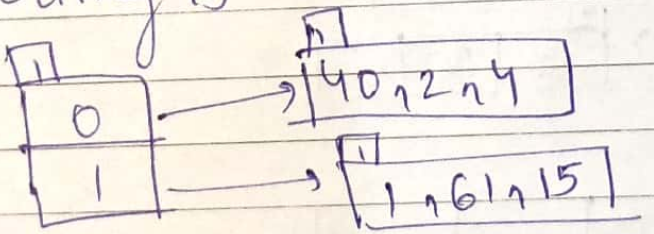
Inserting 2 hash(2) = 0



Inserting 4 hash(4) = 0

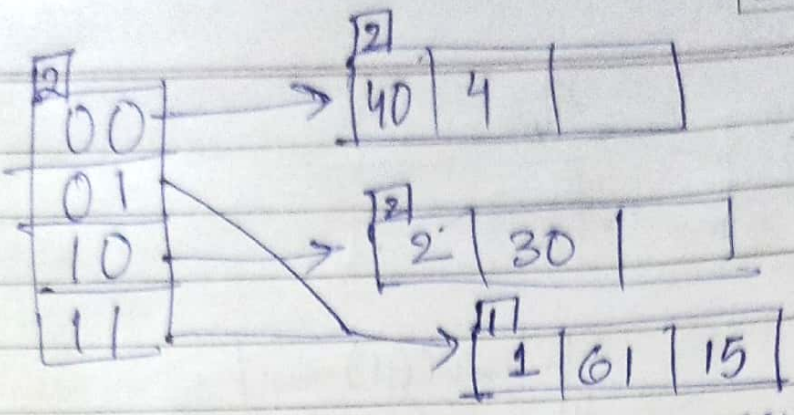


Inserting 15 hash(15) = 1

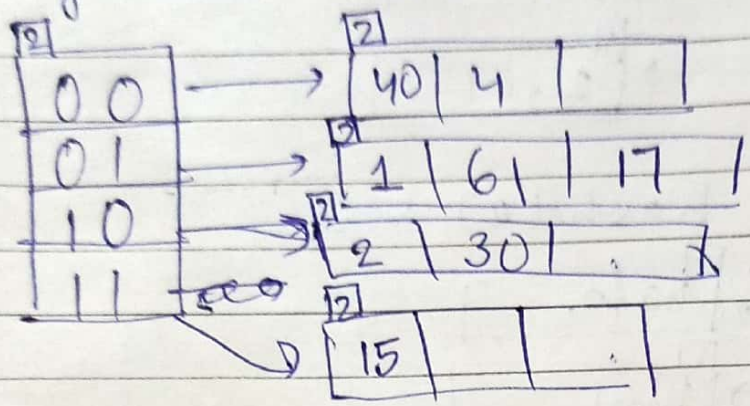


Inserting 30 , hash(30) = 0 ⇒ OVERFLOW!
Directory expansion & rehashing of all the numbers inserted upto now occurs
global depth = 1+1 = 2

hash(1) = 01 , hash(61) = 01
hash(40) = 00 , hash(2) = 10 , hash(4) = 00
hash(15) = 11 hash(30) = 10



Inserting 17 hash(17) = 01 OVERFLOW!

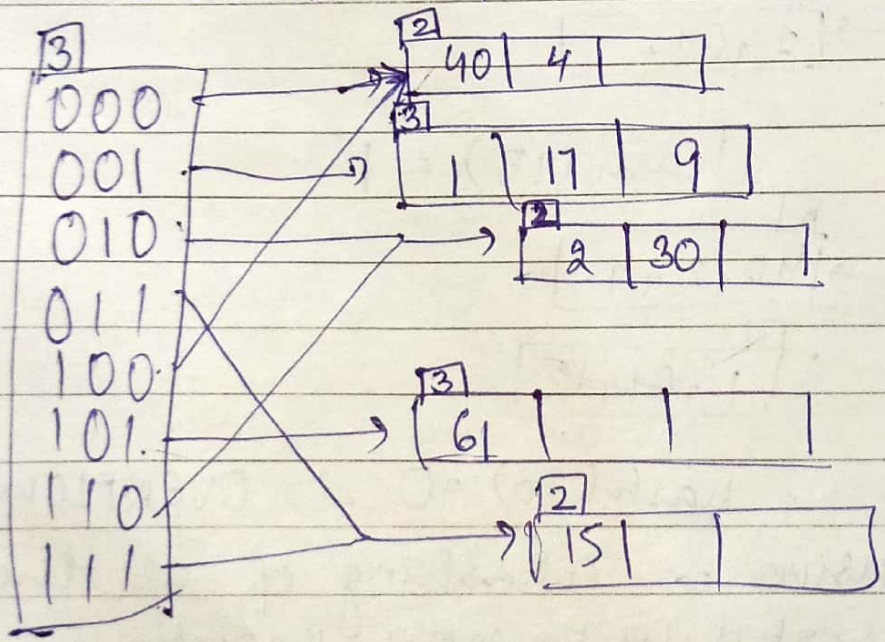


Inserting 9 hash(9) = 01 OVERFLOW!

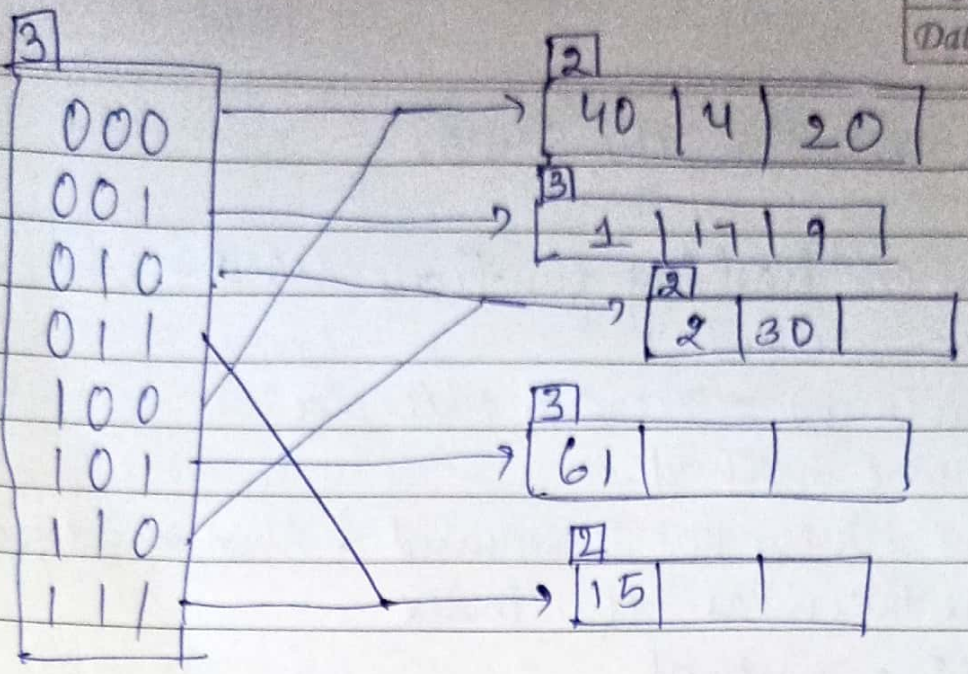
local depth = global depth

⇒ directory expansion & rehashing

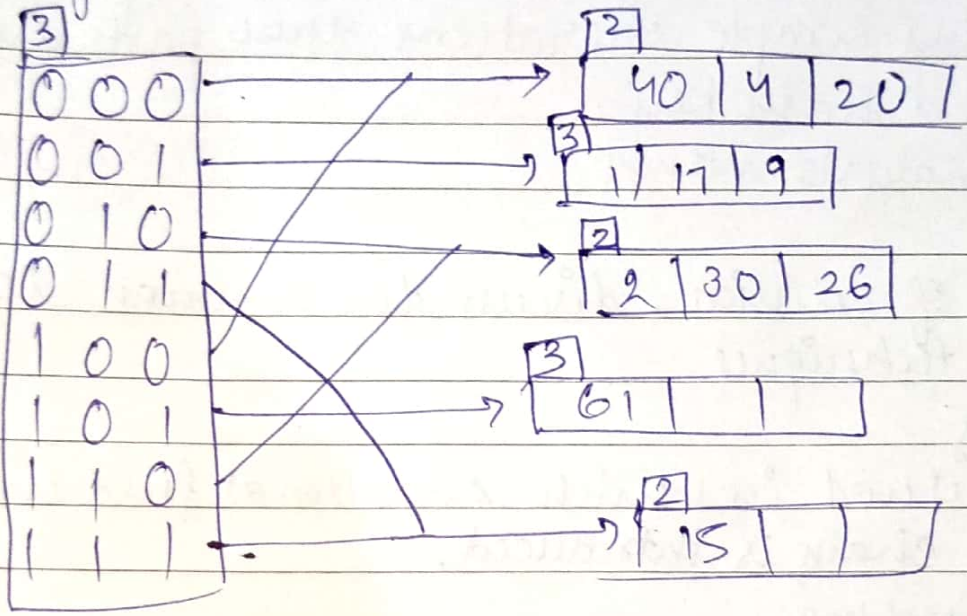
hash(1) = 001, hash(61) = 101, hash(17) = 001, hash(9) = 001



Inserting 20 hash(20) = 100



Inserting 26 hash(26) = 010



Que=7 Answer the following questions in terms of static hashing:

a) Define static hashing

In DBMS, hashing is a technique to directly search the location of desired data on the disk without using index structure. Data is stored in the form of data blocks whose address is generated by applying a hash function in the memory location where these records are stored known as data block or data bucket.

In static hashing, the resultant data bucket address

will always remain the same.

b) Most common hashing functions used

1) Division method

hash(key) = record % table-size

2) Mid-square method

In this method firstly key is squared & then mid part of the result is taken as the index.

3) Digit folding method

In this method the key is divided into separate parts and by using simple operations these parts are combind to produce a hash key

4) Digit Analysis method

c) In case of collision, discuss the common collision resolution techniques.

1) Chaining

It is a method in which additional field with data, i.e., chain is introduced.

2) Linear probing

In this, collision can be solved by placing the second record linearly down, whenever empty place is found.

hash(key, i) = (hash(K) + i) mod m

3) Quadratic probing

hash(key, i) = (hash(K) + i^2) mod m

4) Double hashing

hash(key, i) = (h_1(K) + ih_2(K)) mod m

d) Differentiate between static & dynamic hashing

STATIC HASHING	DYNAMIC HASHING
A hashing technique that allow users to perform lookups on a finalized dictionary (all objects in dictionary are final & not changing)	A hashing technique in which data buckets are added & removed dynamicaly & on demand.
Resultant data bucket address is always the same	Data buckets change depending on the records
less efficient	more efficient.