Anish Sachdeva
DTU/2K16/MC/13

Database Management System (MC-302)
Assignment-3
Delhi Technological University

Q5] Perform the following operations on B tree and B+ tree.

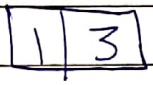
Be tees have 4 pointers and 3 keys

a) insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10
b) delete 9, 7, 8

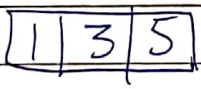
B-Tree

i) insert 1

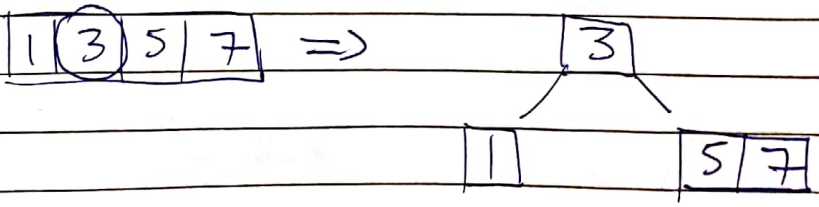
ii) insert 3



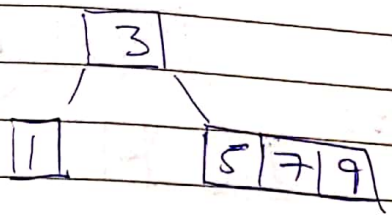
iii) insert 5



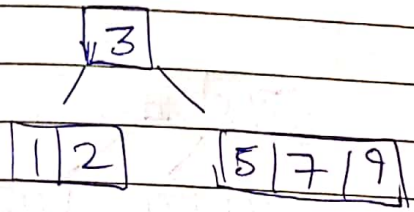
iv) insert 7



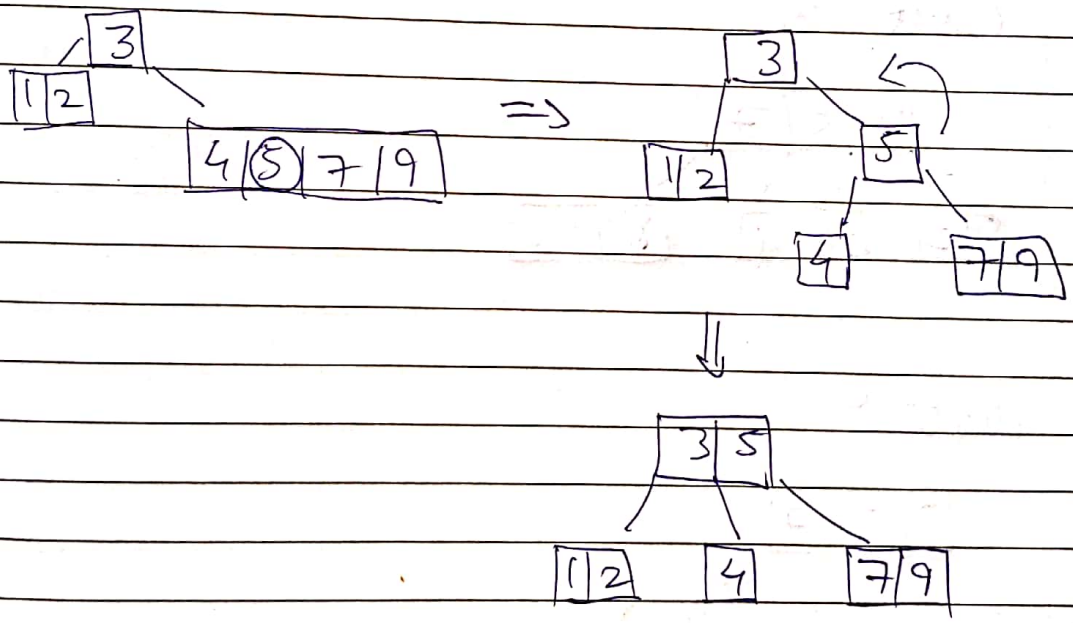
vi) Insert 9



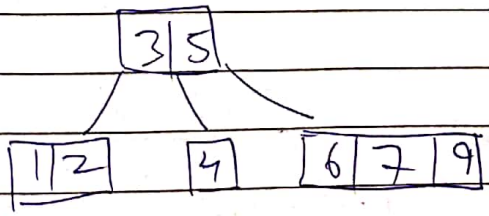
vii) Insert 2



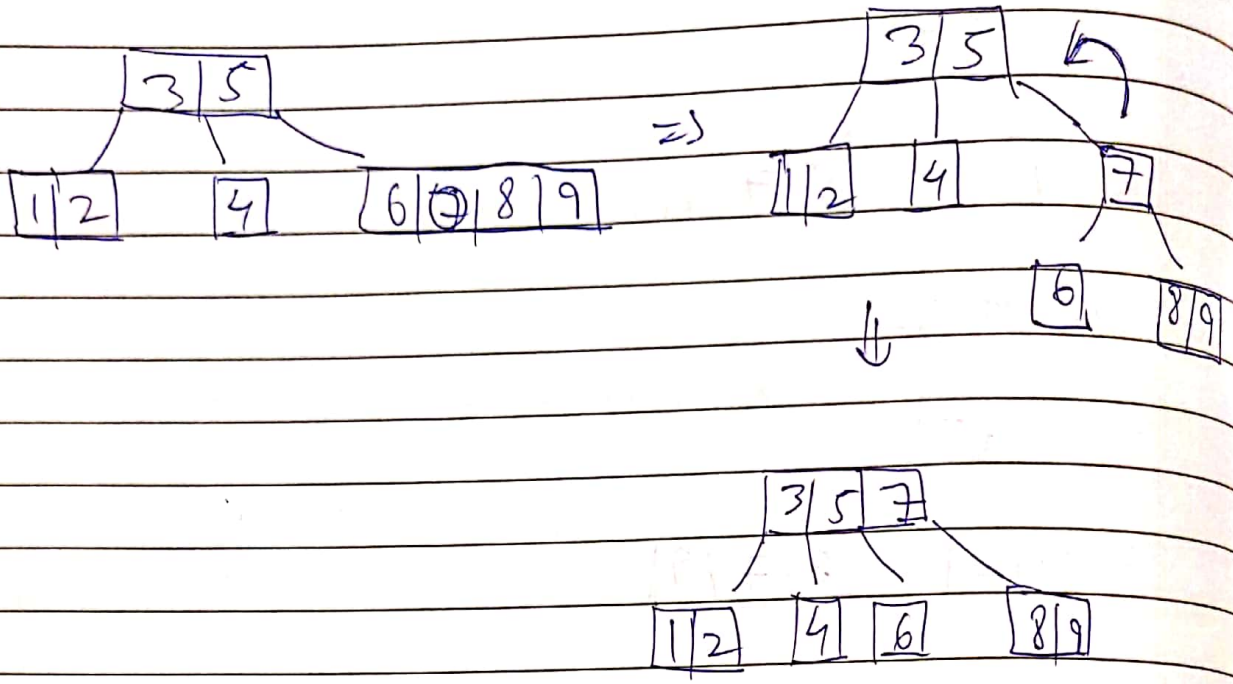
viii) Insert 4



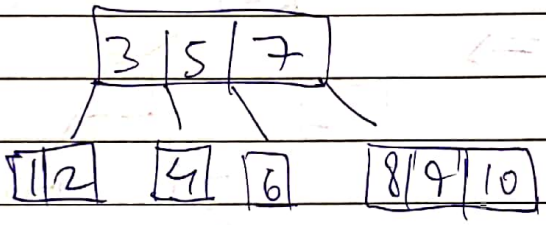
viii) Insert 6



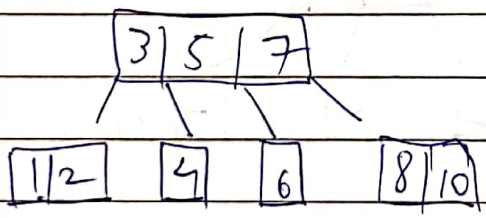
ix) insert 8



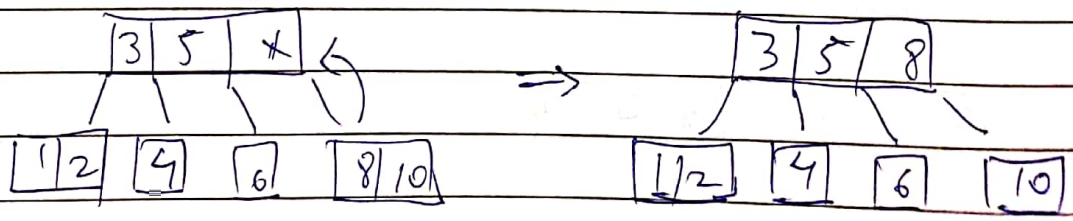
x) insert 10



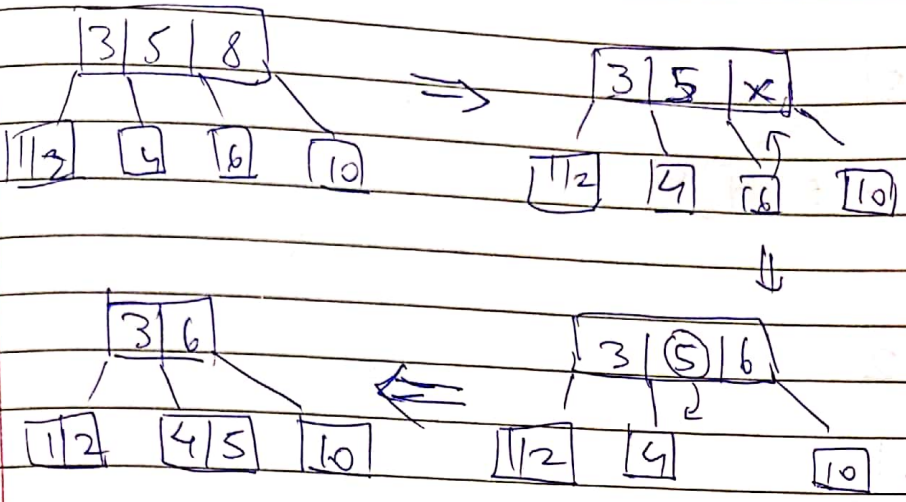
xi) delete 9



xii) delete 7



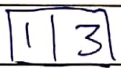
xiii) delete 8



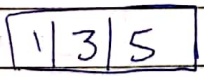
B + Tree

i) insert 1

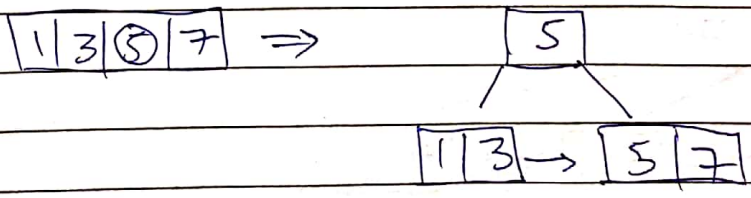
ii) insert 3



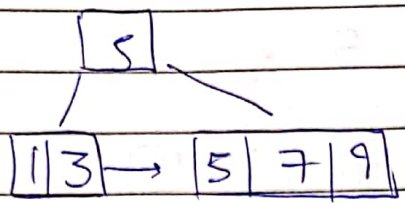
iii) insert 5



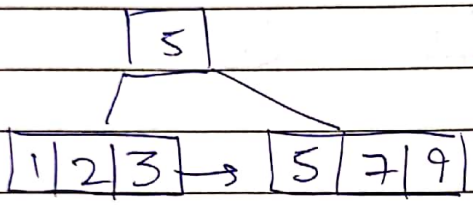
iv) insert 7



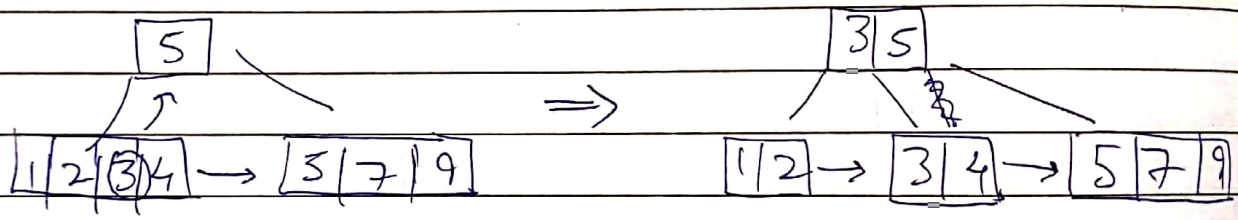
vi) Insert 9



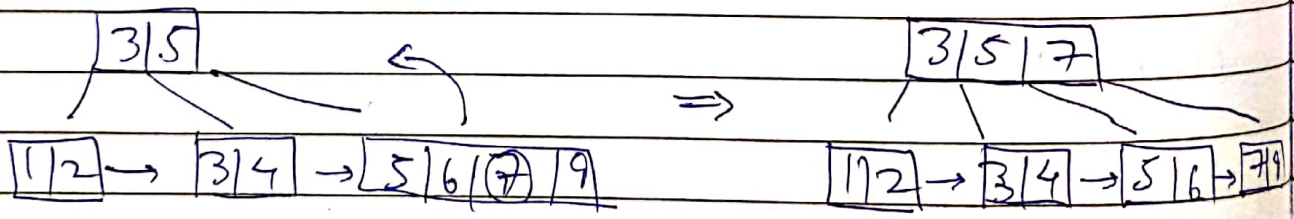
vii) insert 2



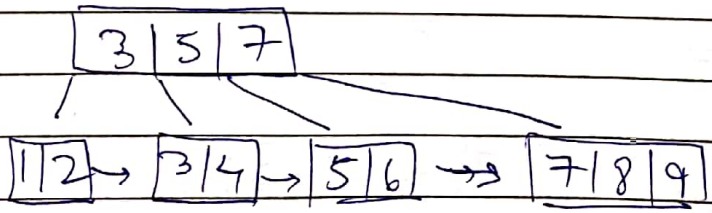
viii) insert 4



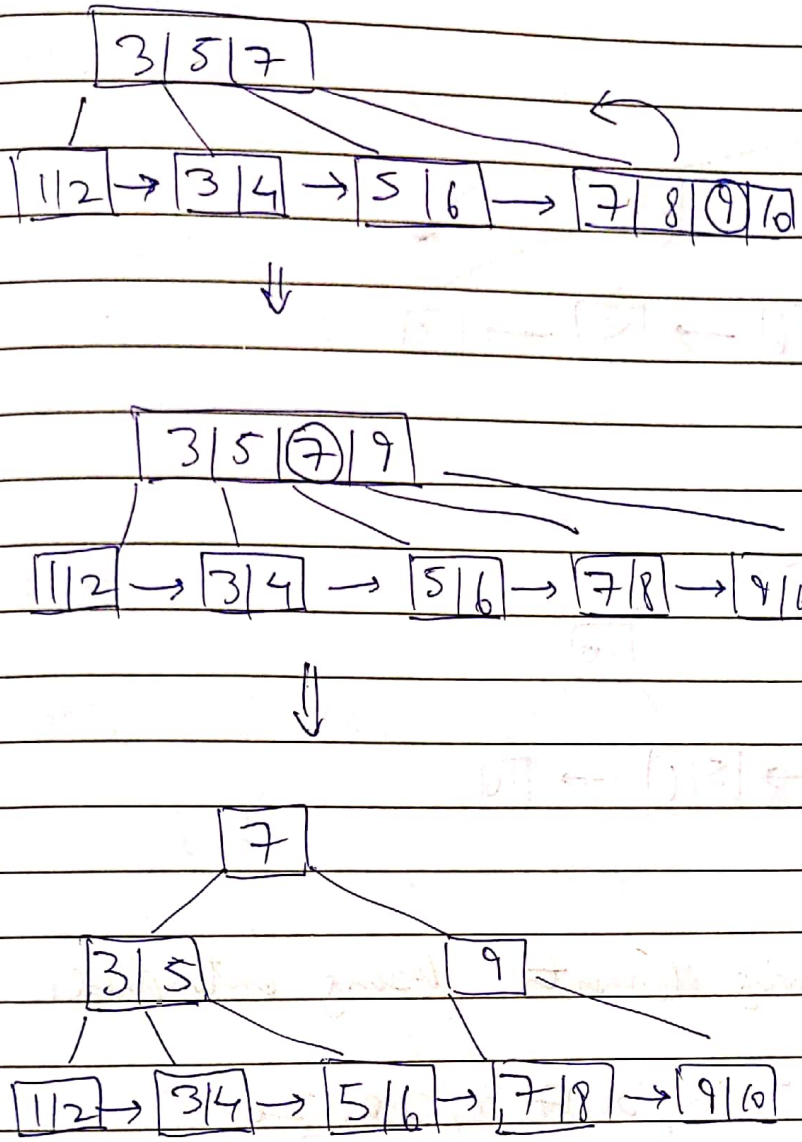
viii) insert 6



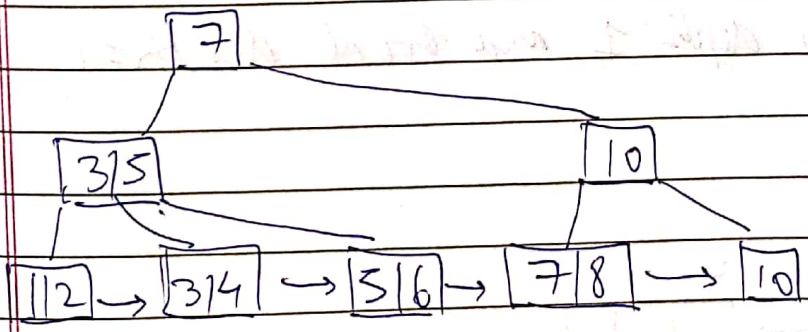
ix) insert 8



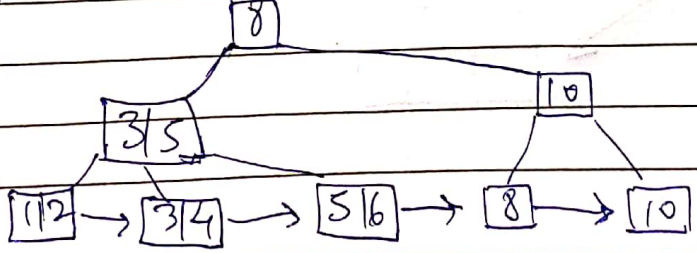
x) insert 10



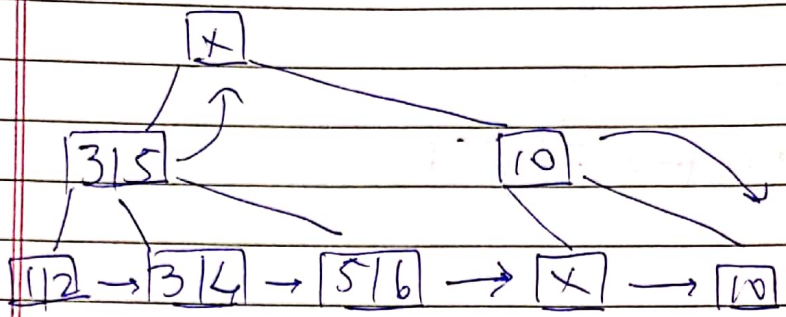
xi) delete 9

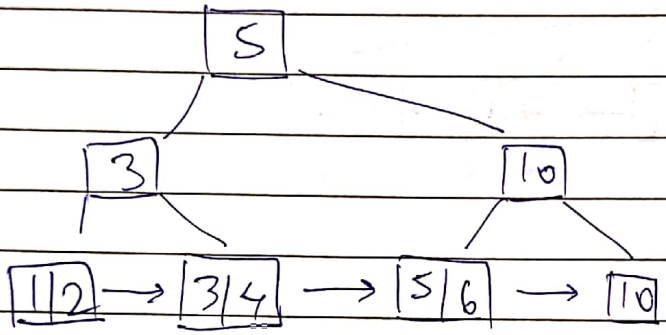


xii) delete 7



(iii) delete 8





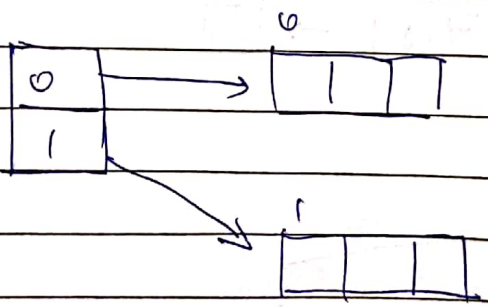
Q6) Hash the following elements using extendable hashing

1, 40, 61, 2, 4, 15, 30, 17, 9, 20, 26

Bucket size : 3

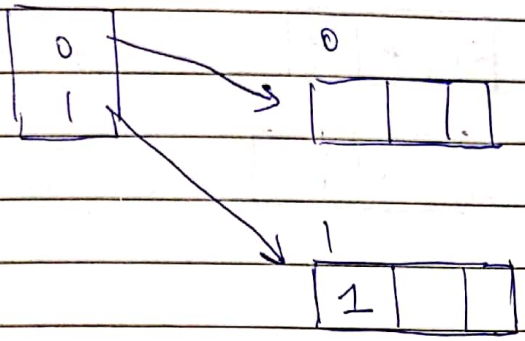
Initially global depth: 1 and local depth = 1
bucket size = 3

i) Initial state



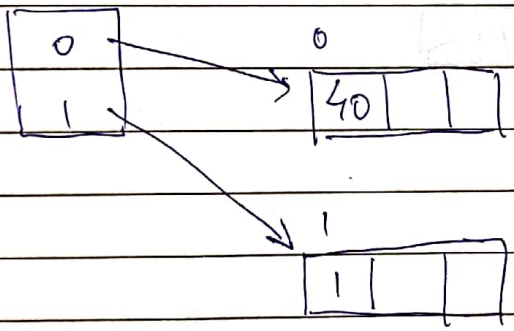
iii) insert 1

$(1)_{10} = (0001)_2$



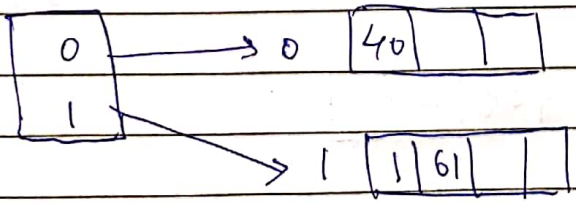
iii) insert 40

$(40)_{10} = (101000)_2$



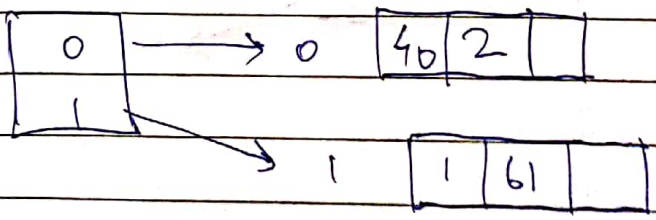
iv) insert 61

$(61)_{10} = (111101)_2$



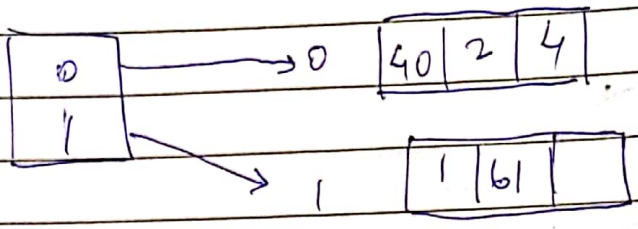
v) insert 2

$(2)_{10} = (0010)_2$

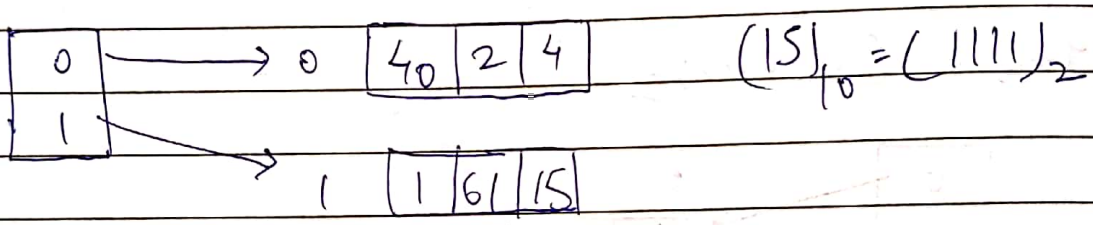


vii) insert 4

$(4)_{10} = (0100)_2$



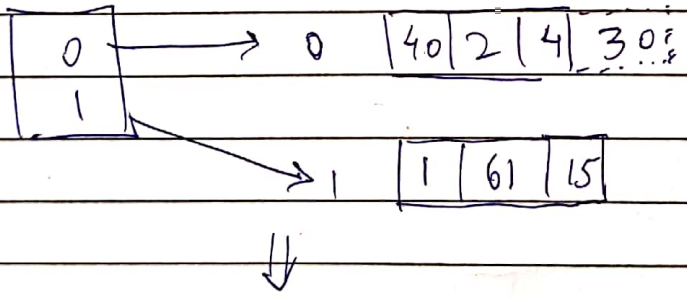
viii) insert 15

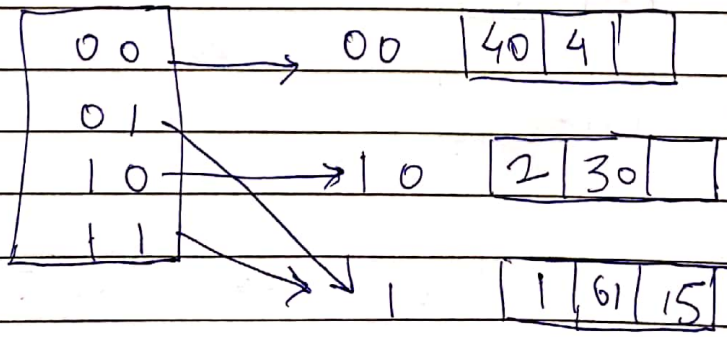


$(15)_{10} = (1111)_2$

viii) insert 30

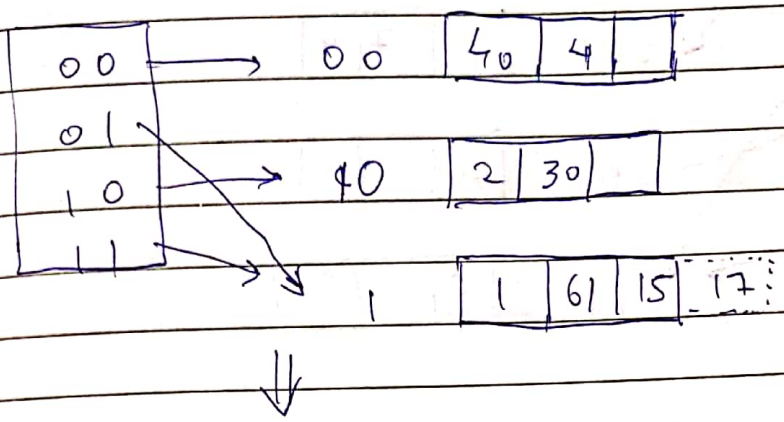
$(30)_{10} = (11110)_2$

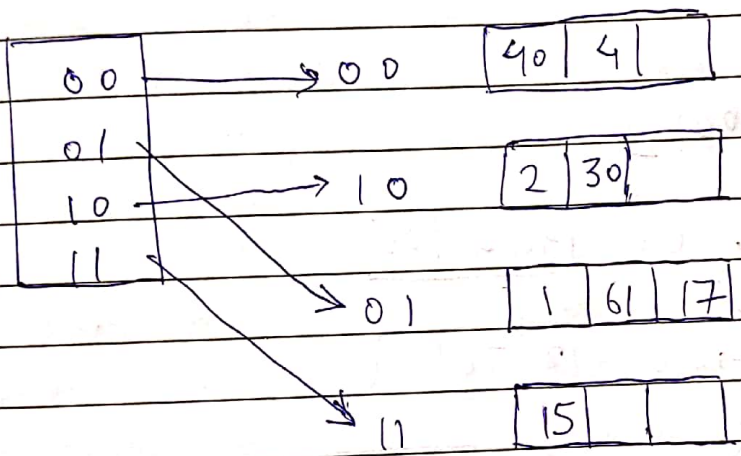




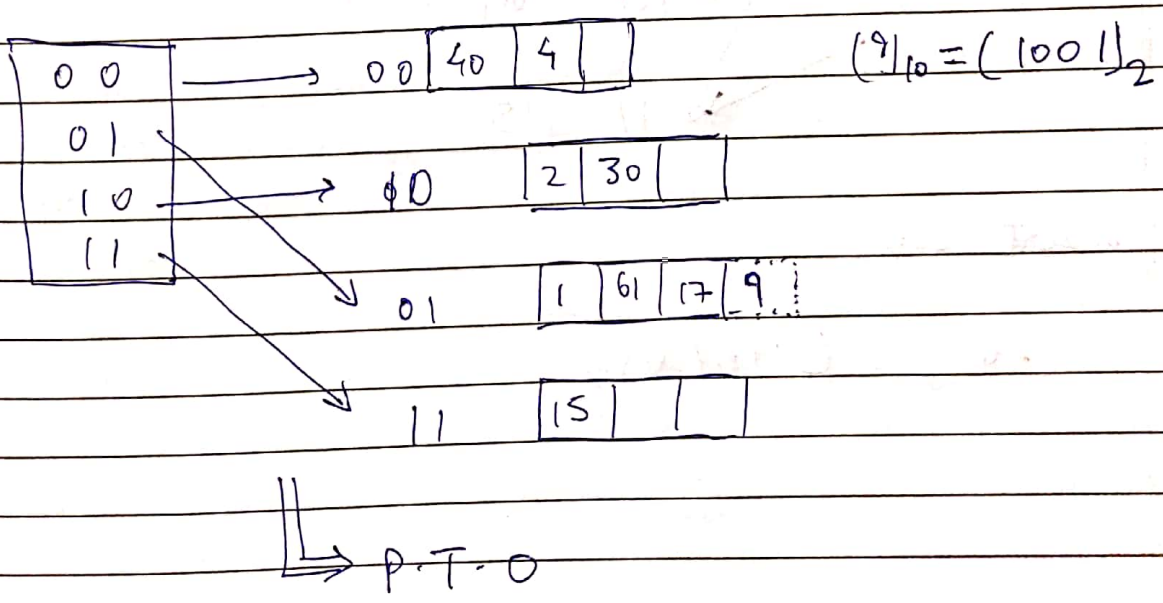
ix) insert 17

$(17)_{10} = (10001)_2$



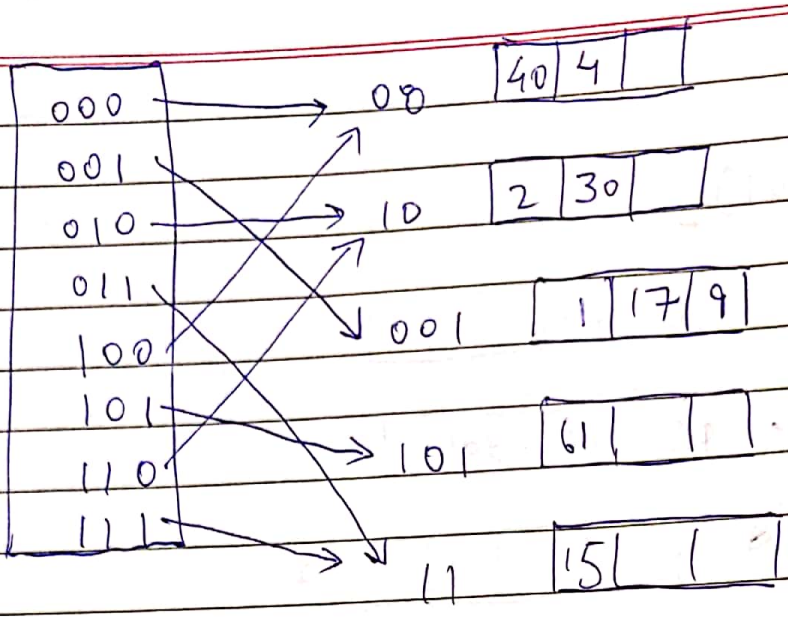


x) insert 9



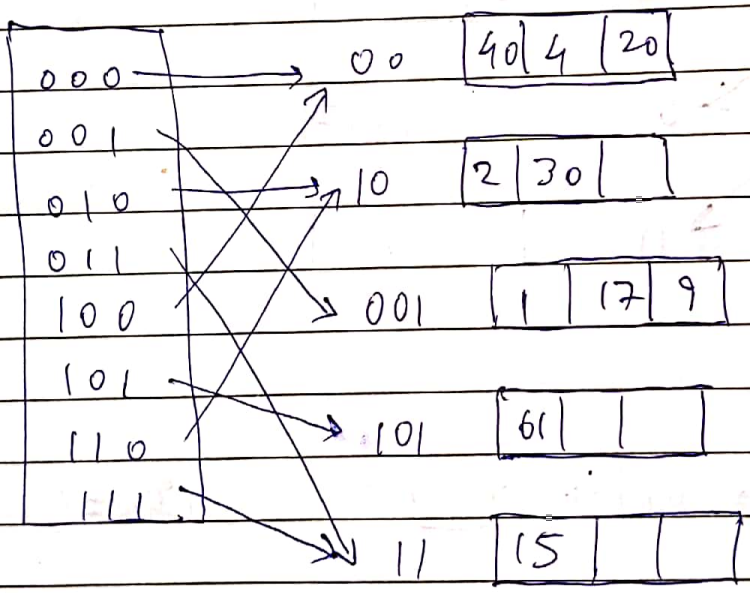
$(9)_{10} = (1001)_2$

p.T.O



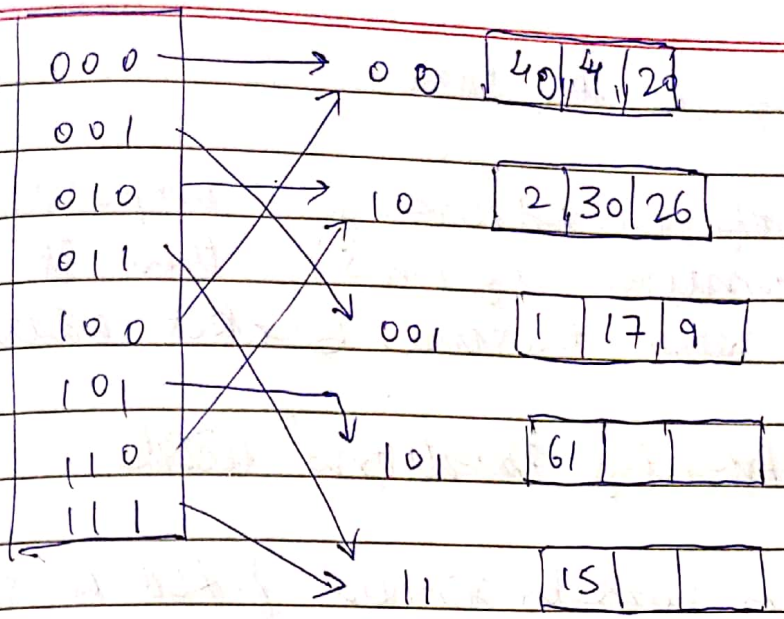
xi) insert 20

$(20)_{10} = (10100)_2$



xii) insert 26

$(26)_{10} = (11010)_2$



(iii) This will be the resultant hashmap based on extendible hashing.

(7) a) Define static Hashing.

In static hashing, we shall use the term bucket to denote a unit of storage that can store one or more records. A bucket is typically a disk block, but could be chosen to be smaller or larger than a disk block.

Formally, let K denote the set of all search-key values, and let B denote the set of all bucket addresses. A hash function h is a function from K to B.
$h : K \rightarrow B$.

To insert a record with search key k_i, we compute $h(k_i)$, which gives the address of the bucket for that record. Then the record is stored in that bucket.

In static hashing the address generated by the hash function for the bucket from the

key will always be the same.

eg. We generate an address for emp-id = 103 using hash function '% 5' (mod 5), then it will always return the same bucket address = 3.

5) Most common hashing Functions used

The number of buckets remains fixed in static hashing.
There are 2 types of static hashing :-

i) Internal Hashing
ii) External Hashing

Some problems with static hashing are :-

i) overflow - The buckets can only accomodate a finite entries and with finite buckets this can soon overflow.

ii) Underflow - We can create many buckets initially for the hashmap and then the actual elements (entries) that we need to insert would be very less leading to an underflow of data.

b) Some common hashing algorithms are :-

A) SHA-2, SHA-1, MD5, NLTM, LANMAN

i) MD5: This is the 5th version of the message digest algorithm. MD5 creates 128-bit outputs. MD5 was commonly used hashing algorithm. That was until weakness in the algorithm started to surface. Most of the weakness manifested themselves as collisions. Because of this MD-5 started to phase out.

ii) SHA-2: This is the second version of the secure hash algorithm standard, SHA-0 being the first. SHA-1 creates 160-bit outputs. SHA-1 is one of the main algorithms that began to replace the MD5, after vulnerabilities were found.

SHA-1 gained widespread use and acceptance. It was designated to a FIPS-140 compliant algorithm.

iii) SHA2: This is actually a suite of hashing algorithms. The suite contains SHA-224, SHA-256, SHA-384 and SHA-512. Each algorithm is represented by the length of it's output. SHA-2 algorithms are more secure than SHA-1 algorithms, but SHA-2 hasn't gained widespread use.

iv) LANMAN: Microsoft LANMAN is the Microsoft LAN manager hashing algorithm. LANMAN was used by legacy windows systems to store passwords. LANMAN used DES to create the hash. The problem is that LANMAN's implementation of the DES algorithm isn't very secure, and therefore, LANMAN is susceptable to attacks.

LANMAN password hashes can be cracked in just a few hours. Microsoft no longer uses LANMAN as the default storage mechanism.

It is available, but it is no longer turned on by default.

v) NLTM: This is the NT LAN manager algorithm. The NLTM algorithm is used for password hashing during authentication. It is the successor of the LANMAN algorithm. NLTM was followed NLTMv2. NLTMv2 uses a HMAC-MD5 algorithm for hashing.

c) The 2 types of collision resolution techniques that are used are:-

1) Open Addressing
2) Chaining

Open Addressing: In this technique a hash table with pre-identified size is considered. All items are stored in the hash table itself. In addition to the data, each hash bucket also maintains three states EMPTY, OCCUPIED and DELETED.

While inserting if a collision occurs, alternative cells are tried until an empty bucket is found. For which one of the following techniques is adopted.

1. Linear Probing (This is prone to clustering of data + some other constraints)

2. Quadratic probing

3. Double hashing (in such cases of collision another hashing function is used with the key value as input to identify where in the open addressing scheme the data should actually be stored.)

Chaining: Open hashing, is a technique in which data is not directly stored at the hash key index (K) of the hash table. Rather the data at the key index (K) in the hash table is a pointer to the head of the data structure where the data is actually stored. In the most simple and common implementations the data structure adopted for storing the elements is a linked list.

d)

Static Hashing	Dynamic Hashing
Number of buckets are fixed.	Number of buckets are not fixed (dynamic) - change over time
As the file grows, performance decreases.	Performance doesn't degrade as no. of entries stored in hashmap increases.

There is more space overhead	There is less space overhead
Bucket address table isn't used	Bucket address table is used.
Some forms of static hashing are Open hashing and closed hashing.	Some forms of Static dynamic hashing are extendable hashing and linear hashing.
Implimentation is relatively simple	Implimentation is relatively complex.
It isn't very desirable to use given it's less than adequate performance.	It is very desirable and used in most applications due to performance benefits
The system directly interacts with the Bucket table / and then the bucket.	The bucket address is used in this implimentation
Overflowing chaining is used	Overflowing chaining isn't used

Q4) i)

T_1	T_2
	BEGIN
	R(X) → 10
	R(Y)
	W(X) → 13
BEGIN	T_1 is reading data of X written by T_2 before committing
R(X) → N	
N(X)	
COMMIT	
	W(Y)
	ABORT

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 of a 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	
	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 any time during the second phase

If we apply strict 2PL, only serializable schedules will be allowed and all 3 schedules listed above will be disallowed. Let us denote exclusive locks by X and shared lock by S, and Release

y lock by V (unlock), commit by C.

Following are the execution schedules whose strict 2PL will ensure Serializability or 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 won't be 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) $S_2(X), R_2(X), S_2(Y), R_2(Y), X_2(X), W_2(X), S_1(X)$ – request won't be 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) Consider the following classes of schedule: serializable, conflict serializable, view-serializable, recoverable, avoids-cascading-aborts and strict. Classify below schedules in one of the classes.

The actions are listed in the order they are scheduled and subscripted with the transaction name. If a commit or abort is not shown, the schedule is complete; assume the abort or commit must follow all the listed actions.

1. $R_1(X) R_2(X) W_1(X) W_2(X)$

There can be a conflict in writing on different processes on same variable x. Hence this is is non-serializable, non conflict-serializable and not

View serializable

Although it is recoverable, hence also avoid cascading aborts. It is non-strict

2) $W_1(X)\ R_2(Y)\ R_1(Y)\ R_2(X)$
All the reads are happening after the write, hence not serializable, not conflict-serializable, not view serializable. It is recoverable and avoid cascading aborts. It is also non-strict.

3) $R_1(X)\ R_1(Y)\ W_1(X)\ R_2(Y)\ W_3(Y)\ W_1(X)\ R_2(Y)$
It is non-serializable, non conflict-serializable, not view-serializable. It is does not avoid cascading aborts, it is non-strict. We cannot decide whether it's recoverable since the abort commit sequence of these transactions are not specified

4) $R_1(X)\ W_2(X)\ W_2(X)\ W_1(X)$ Abort$_2$ Commit$_1$
This sequence is serializable, conflict serializable and view serializable. It is also recoverable and avoids cascading aborts. But, it is not strict

5) $R_1(X)\ W_2(X)\ W_1(X)$, Commit$_2$, Commit$_1$
It is serializable and view serializable, view conflict serializable, It is recoverable and avoid cascading aborts; It is non-strict

6) $W_1(X)\ R_2(X)\ W_1(X)\ \text{commit}_2\ \text{Abort}_1$
It is not serializable, non view-serializable, not conflict-serializable. It is not recoverable, therefore not avoid cascading aborts, It is also non-strict.

7) $R_2(X)\ W_3(X)$, $\text{Commit}_3\ W_1(Y)$, Commit_1, $R_2(Y)$, $W_2(Z)$, commit_2
It belongs to all mentioned classes above.

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, non-conflict serializable. It is recoverable, but not cascading aborts, and is non-strict

8) $R_1(X)\ W_2(X)\ W_1(X)\ R_3(X)\ \text{Commit}_1\ \text{Commit}_2$ Commit_3
It is serializable and view-serializable, non-conflict serializable. It is recoverable, but not avoid cascading aborts, non-strict.

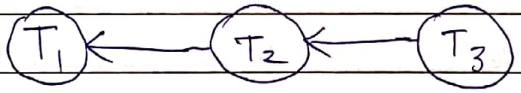
Q3) Consider the following lock requirements in Table. S(·) and X(·) stand for shared lock and Exclusive lock respectively.

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 I, determine which lock will be granted or blocked by the lock manager. Write 'g' in the LM row to indicate the lock is granted and 'b' to indicate lock is blocked.

ii) Give the wait for graph for the lock requests in Table-I and time-tick t_7.



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

There is no deadlock as there is no cyclic wait in the dependency graph

Q4) Consider the following lock requests in Table 2

Lock requests for four transactions : T_1 T_2 T_3 T_4

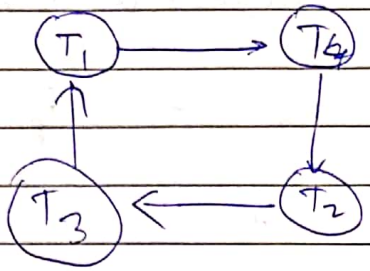
Time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
T_1	X(B)				S(A)			
T_2						X(D)	X(C)	
T_3			S(C)					X(B)
T_4		X(A)						S(D)
LM	G	G	G	B	G	B	B	B

i) For locks in table, determine which lock request will be granted, blocked or aborted by the Lock manager (LM). If it has no deadlock prevention policy. Write G for grant B for Block and 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 : 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?

The wait for graph will look like :-



As there is a cyclic wait and deadlock does exist.

iii) To prevent deadlock, we use 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 aborted)
- X(D) at t_5 : G
- X(C) at t_6 : A (T_2 aborted)
- X(B) at t_7 : G (No lock on B since T_1 was aborted)
- S(D) at t_8 : G (No lock held on D since T_2 aborted)

iv) In this question, we use lock manager (LM) that adopts the wait wound-wait policy. We assume that 4 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 : 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)