INDEX

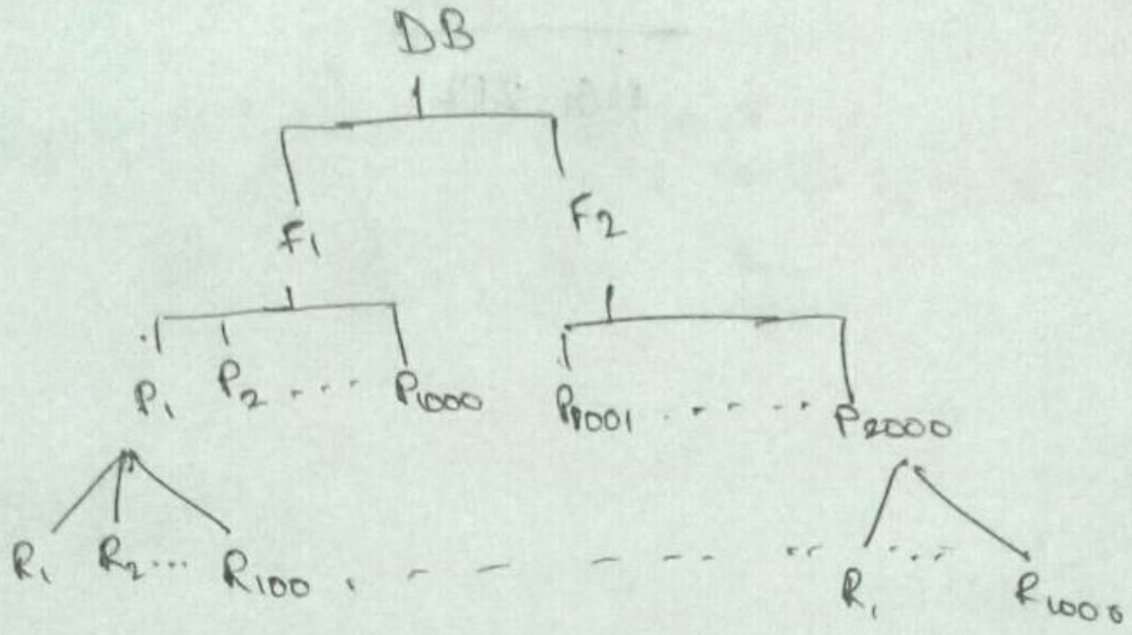
NAME: K. Gowtham STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		DBMS		
		Multi Granularity (Small part in 7th unit)		
		8-File organization & Indexing		
		DLD		
		1. Boolean Algebra & Logic Gates		
		2. k-maps		
		3. Number system & signed data representation		
		4. Combinational Circuits		
		5. Sequential Circuits.		
		English		
		Little bit of introduction		

DBMS

Multiple Granularity:

This protocol needs a mechanism to allow the system to define multiple levels granularity where the small granularities are nested within larger ones. Such a hierarchy can be represented as shown below:



→ If child is already locked, its ancestor can't be locked in conflicting mode.

→ If parent is already locked, its child can't be locked in conflicting mode.

* This protocol can be implemented using the additional lock modes called Intention lockmodes. They are

(i) Intention shared (Is)

It indicates that a shared lock will be requested on some descendant nodes.

(ii) Intension Exclusive mode (Ix)

It indicates that an exclusive lock will be requested on some descendant nodes.

(iii) Shared & Intension Exclusive (SIx)

It indicates that the current node is locked in shared mode and exclusive locks will be requested on some descendant nodes by the same transaction.

Q: Indicate the lock & unlock requests of the following transacti

T_1: Read P_{3000} (R_1)

G ↓
- IS(DB)
- IS(F_2)
- IS(P_{2000})
- S(R_1)

S ↓
- U(R_1)
- U(P_{2000})
- U(F_2)
- U(DB)

MG-2PL

T_2: update(F_2)

g ↓
- IX(DB)
- X(F_2)

S ↓
- U(F_2)
- U(DB)

MG-2PL

g - growing phase
S - shrinking phase

→ This protocol requires locking being done from root to leaves and unlocking being done from leaves to root.

→ As the locking & unlocking being done in two phases, this protocol is called Multiple Granularity 2 Phase locking protocol.

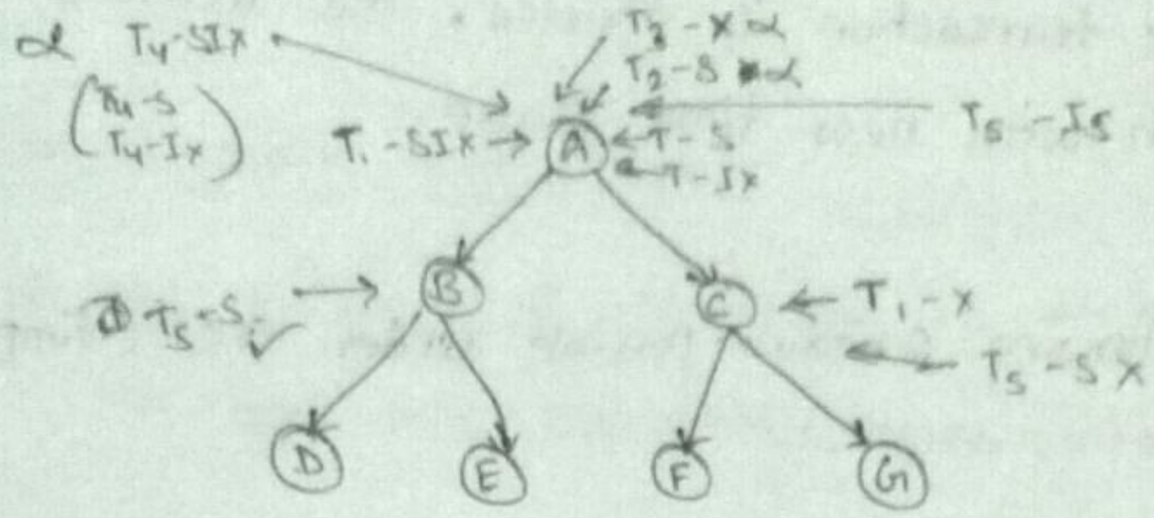
→ This protocol is best suitable for the database system that uses both & short & long transactions.

Q: Suppose a transaction (T_i of previous tree) wants to read all the records of page P_{1000}. Which of the following is the correct locking sequence

a) IS(DB) S(F_1) S(P_{1000})
b) IX(DB) X(F_1) X(P_{1000})
c) IS(DB) IS(F_1) S(P_{1000}) ✓
d) all of the above

Q: Suppose a transaction T_i locks a node in the database tree in SIX mode then which of the following locks can be requested on the same node by the different transactions.

a) S b) X c) SIX d) IS ✓



Lock compatibility:

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	X
IX	✓	✓	X	X	X
S	✓	X	✓	X	X
SIX	✓	X	X	X	X
X	X	X	X	X	X

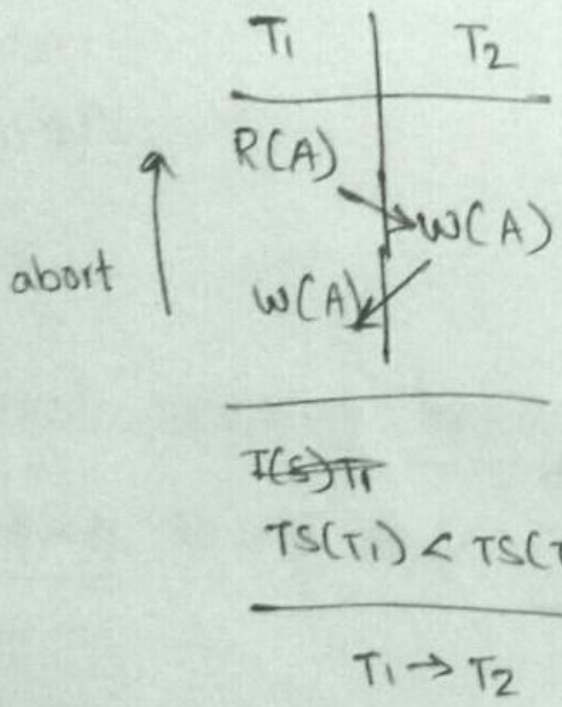
Time Stamp based Protocols:

* It is a deadlock free protocol.
* The order of serializability is in the order of their time stam
  i.e, if TS(Ti) < TS(Tj) then the resultant schedule must be equilant to $T_i \rightarrow T_j$

(i) Time stamp ordering protocol :

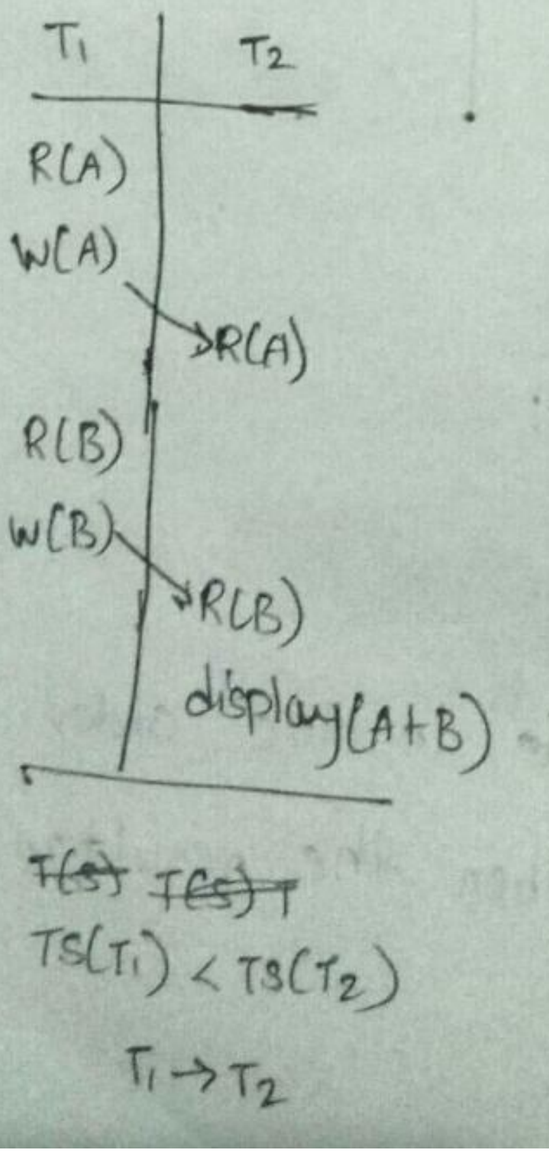
It requires that all the conflicts must be processed in the order of their time stamps. If any transactions operation not executes in the order of their time stamps such operation is rejected and the transaction is aborted. The aborted transaction restart in a system with new time stamp.

Q: verify if the following schedule possible under TOP (time stamp ordering protocol):



∴ This schedule is not possible under TOP.

Q: Verify if the following schedule possible under TOP.



∴ This schedule is possible under TOP.

- Advantages:

  * Ensures conflict serializability
  * Free from deadlocks.

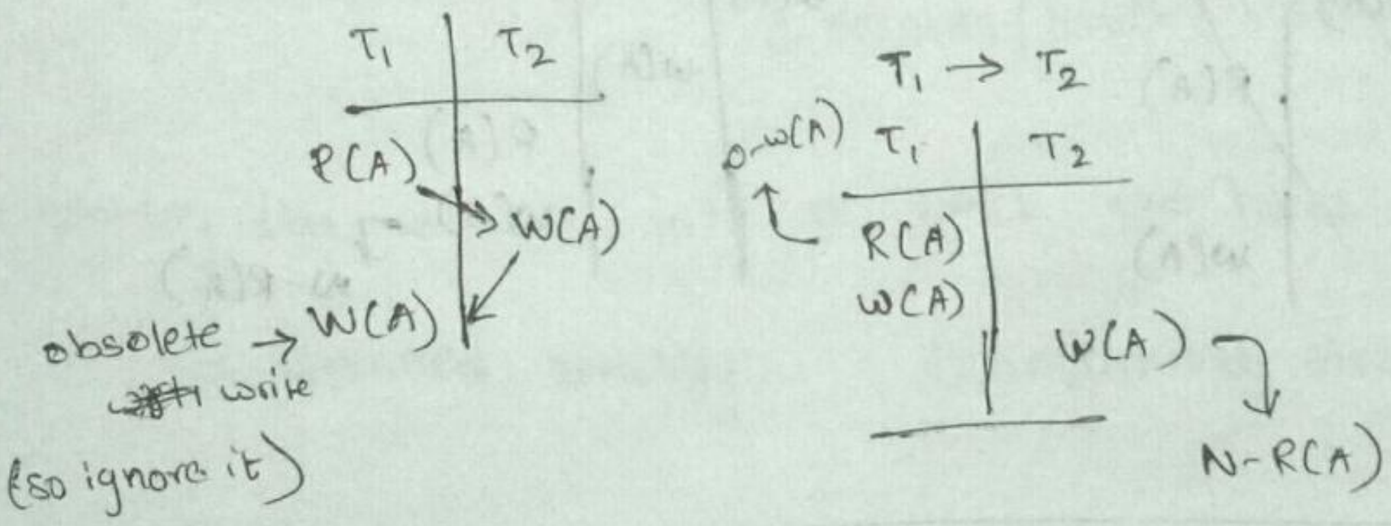
Disadvantages:

  * Starvation may occur.

(ii) Thomas write rule (TWR):

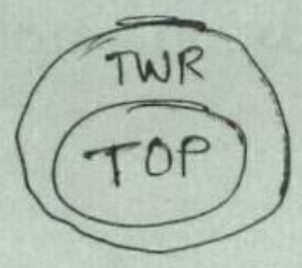
→ The problem of starvation is minimized by ing ignoring obsolete write operations (Useless writes).

Q: verify if the following schedule is possible under TWR



∴ This schedule is possible under TWR

→ Every TOP is also TWR but reverse is is not true.



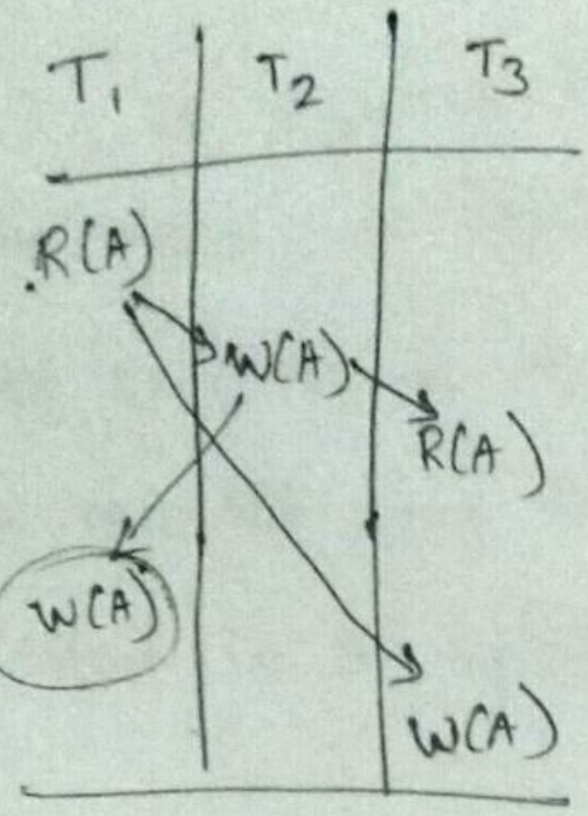
→ TWR uses view serializability.

Q: Consider the following schedule

$R_1(A)$ $W_2(A)$ $R_3(A)$ $W_1(A)$ $W_3(A)$

The above schedule is possible under ____.

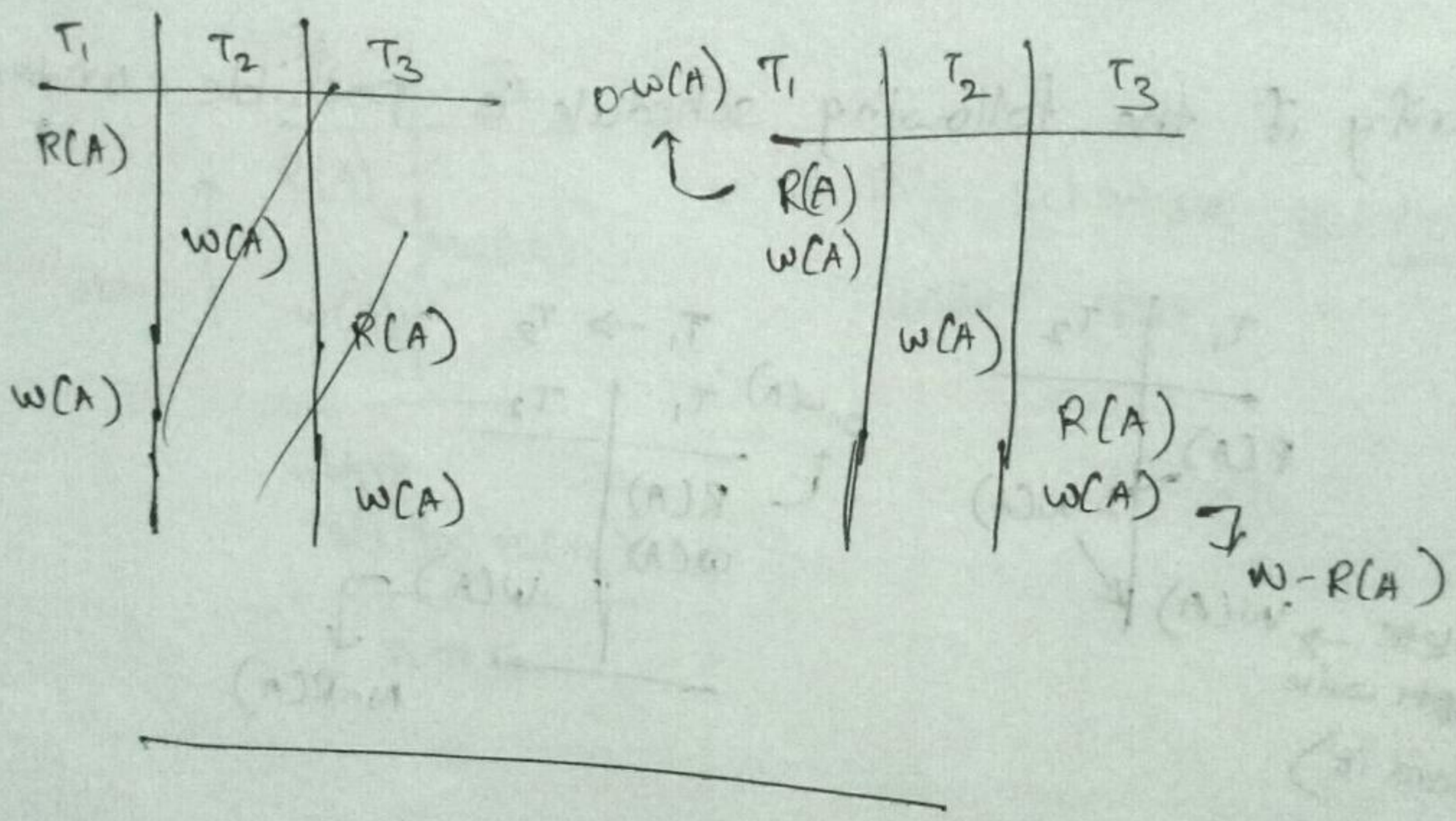
a) TOP b) TWR ✓ c) both d) none



Time stamp order: T_1 T_2 T_3

It is clear that the schedule is not under TOP

Consider serial schedule $T_1 \rightarrow T_2 \rightarrow T_3$



Problems:

(25) Time-stamp ordering

8. File Organization & Indexing:

Database is stored as collection of files and each file is a collection of records stored into memory blocks. Each record is a collection of data.

Blocking factor: It is the avg no of records that can be stored into a block

Eg: let block size, B = 100 bytes

record size, R = 30 bytes

Blocking factor = 3.33 records / block (spanned strategy)

(or)

3 records / block (unspanned strategy)

For storing the records into a block we have two strategies:

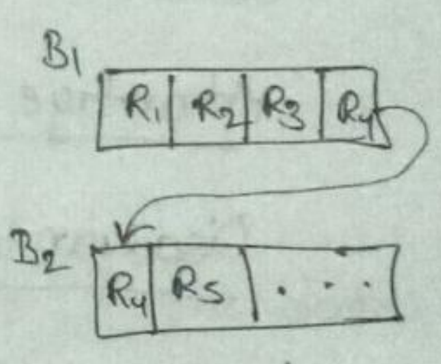
(i) spanned strategy (ii) unspanned strategy

Spanned strategy:

It allows partial partial part of a record can be stored into a block.

Advantage: No wastage of memory.

Disadvantage: More no of blocks to access.



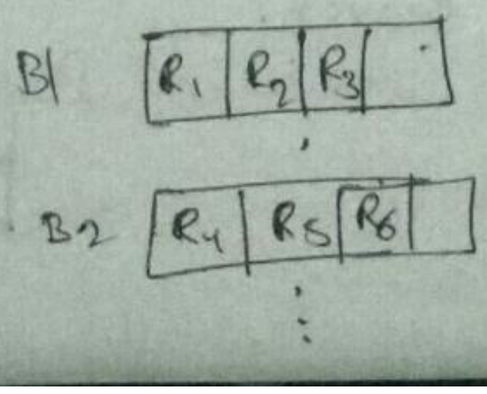
(This diagram for above example)

Note:

This is suitable for variable length records.

Unspanned Strategy:

No record can be stored in more than one block.



Advantage: Less blocks to be accessed

Disadvantage: Memory wastage.

Note:

Used for storing fixed length records.

Organization of records into a file:

(i) Ordered file organization:

The records are stored in the order of some key value. Usually on its primary key.

*It uses binary search to search records since records are ordered.

Advantage: Searching is effficient.

Disadvantage: Insertion is expensive.

(ii) Unordered file organization:

Any record can be stored anywhere in the file where there is a place for the record. Usually the records are stored at the end of the file.

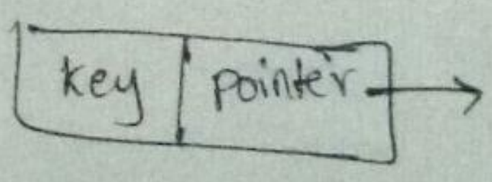
Advantage: Insertion is easier.

Disadvantage: Searching takes more time.

Index.

* Index is used to speed up the searching process.

* Index is ordered file with two fields key & pointer on which we perform binary search.



Pointer
- record pointer
- block pointer

Index classification

① Single level index
- (i) Primary index (PK + ordered)
- (ii) clustered index (NK + ordered)
- (iii) Secondary index ($\frac{NK}{CK}$ + unordered)

② Multi level index
- a) B-Tree
- b) B^+ - Tree

PK (Primary key)
- * no duplicates
- * no nulls

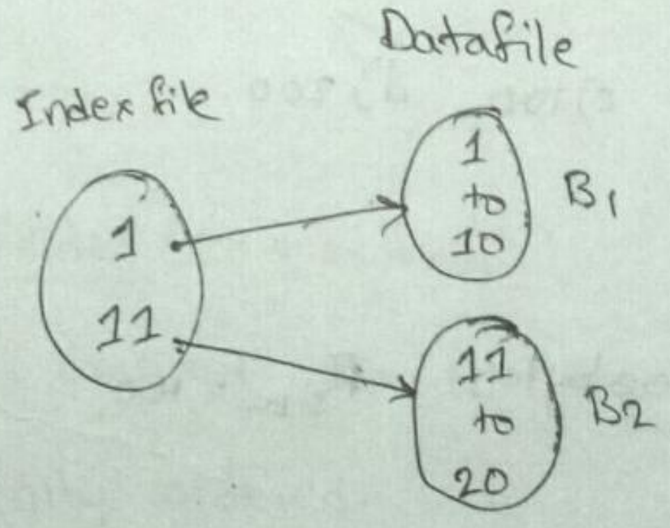
NK (Non-key)
- * with duplicates

CK (candidate key)
- * no duplicates

Other classification of index:

(i) sparse index:

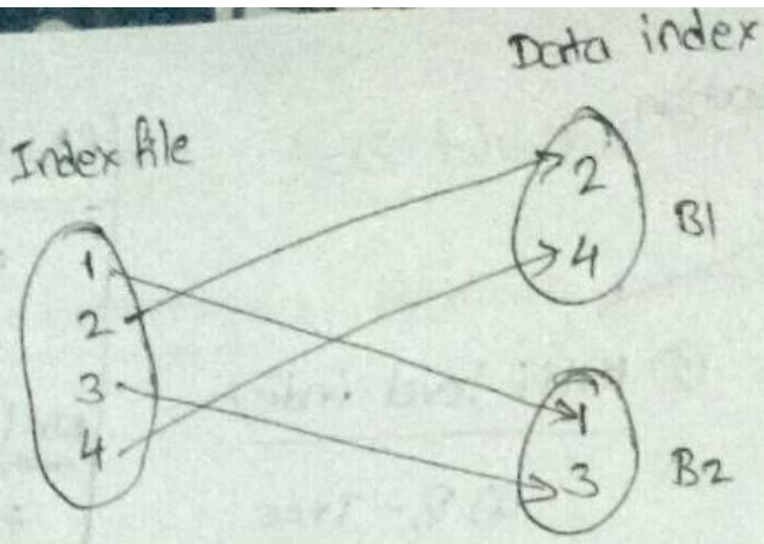
It has an index record only for some records in data file



→ An index record is created for each block in the data file
→ The no of index records are equal to no of blocks in the datafile.
→ It uses block pointer (points to block)

(ii) Dense index:

* An index record is created for each record in datafile.
→ The no of index records are equal to no of records in datafile.
→ It uses ~~reo~~ record pointer
→ It provides logical ordering of datafile.



Q: Consider a file of 10,000 records stored in the blocks of size 1000 bytes each and all the ~~rec~~ recoreds are fixed length of size 100 bytes of which key field of record is 12 by and ~~bt~~ both block and record pointer takes 8 bytes.

(i) Find the no of blocks needed for sparse index on the data file.

a) 10 b) 20 ✓ c) 100 d) 200

(ii) Find the no of block needed for dense index on the datafile.

a) 10 b) 20 c) 100 d) 200 ✓

$n = 10000$, $B_{size} = 1000$ bytes $R_{size} = 100$ $k_{size} = 12$, ~~$B_p = R_p =$~~

$B_p = R_p = 8$

(i.A) 10 ~~blocks for records~~ 1 block can store 10 records

∴ we need 1000 blocks to store all records

~~1 index record = 12 + 8 = 20~~

~~1 block can store 50 index records~~

∴ we need 1000 index records

1 index record = 12 + 8 = 20 bytes

1 block → 50 index records

∴ we need 20 blocks for sparse ind

Ans:

(a) Blocking factor = $\frac{1000}{100}$ = 10 records / block

no of datablocks needed = $\frac{10000}{10}$ = 1000

Size of index record = 12+8 = 20 bytes

Blocking factor (index file) = $\frac{1000}{20}$ = 50 index records / block

Sparse:

no of index records = 1000

no of index blocks = $\frac{1000}{50}$ = 20

Dense:

no of index records = 10,000

no of index blocks = $\frac{10000}{50}$ = 200

Single level index:

(i) Primary Index (PK + ordered)

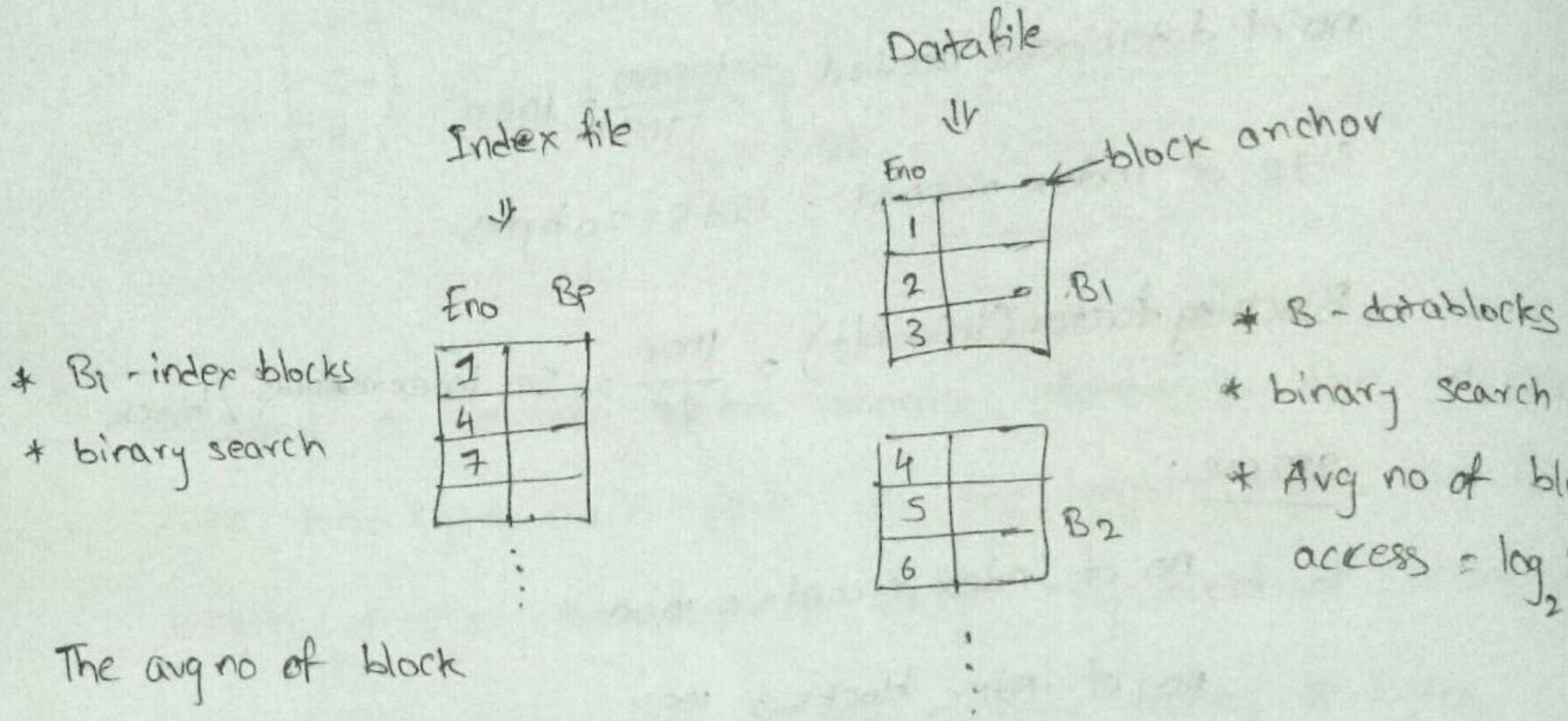
It is created on primary key of data file if the file is physically ordered.

→ Primary index is an ordered file with two fields. First field is primary key of the datafile and second field is a block pointer.

→ An index record is created for the first record of each block called block anchor

→ The no of index records are equals to no of block in the datafile.

→ This type of index is called sparse index.



The avg no of block access = $\log_2 B_i + 1$

Q1: Suppose a file contains 30,000 records stored on a

(1) disc blocks are of size 1024 bytes and all the records are fixed length of size 100 bytes of which key field is of size 9 bytes and both block & record pointer takes 6 bytes. Find the avg no of blocks to access if we create a primary index on key field of the file.

$$\text{Blocking factor} = \frac{1024}{100} = 10$$

no of ^{data} blocks = 30000/10 = 3000

index record size = 9+6 = 15

no of index records = 3000

~~no of index blocks = $\frac{1024}{15}$~~ = blocking factor (index) = $\frac{1024}{15} = 68.26$

no of ^{index} blocks = $\frac{3000}{68} = \frac{1000}{23} = 44.11$ ∴ 45 index blocks

Avg no of blocks to access $= \log_2 45 + 1$

$= \lceil 5.(5) \rceil + 1$

$= 6 + 1 = \underline{7}$ blocks

(iii) Clustered Index (Non key ordered)

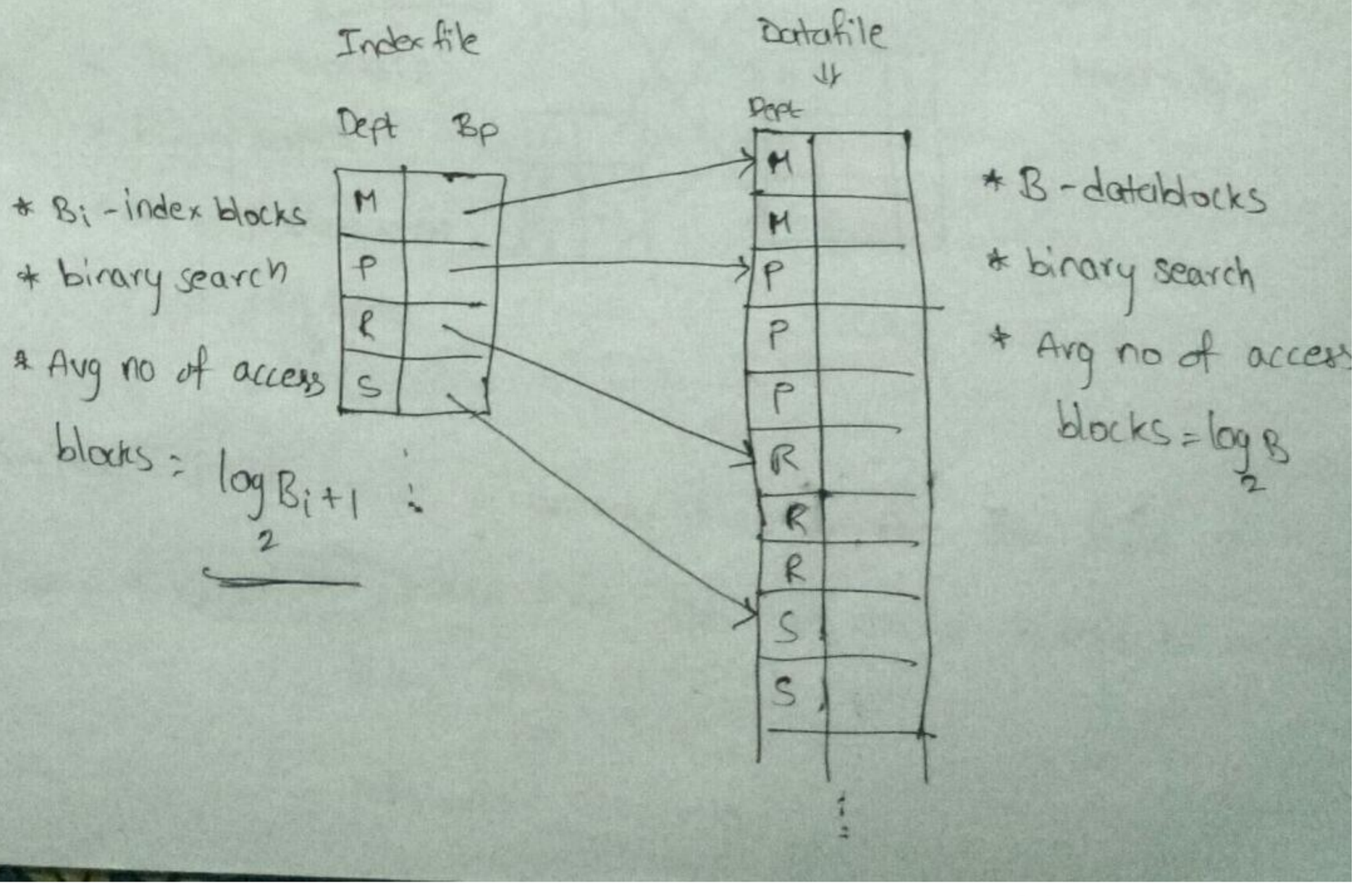
→ If the file is physically ordered on the non key field, then we create a special type of index called a clustered index and the non-key of the file is called clustering field.

→ A clustered index is an ordered file with two field. key (clustering field) & pointer.

→ An index record is created for each cluster.

→ The no of index records are equals to the no of clusters in the datafile.

→ This type of index is called sparse sparse index.



Q:

(2) Suppose if we create a clustered index on the non-key field of the file of (Q1) then find the avg no of block to access. A

Assume that there are 1000 distinct non-key values

no of datablocks = $\frac{300000}{1000}$ = ~~30~~ 1000

index record size = 9+6 = 15

no of index records = 1000

no of index blocks =

blocking factor (index file) = $\frac{1000}{15}$ = 68

no of index blocks = $\lceil \frac{1000}{68} \rceil$ = 15

avg no of block access = $\lceil \log_2 15 \rceil + 1$

= 4+1 = 5 //

iii) Secondary index:

→ secondary index provides secondary means of accessing a file on which some primary access already exists.

→ secondary index may be created on a non-key or on a candidate key of a datafile

Secondary index (ckt unordered):

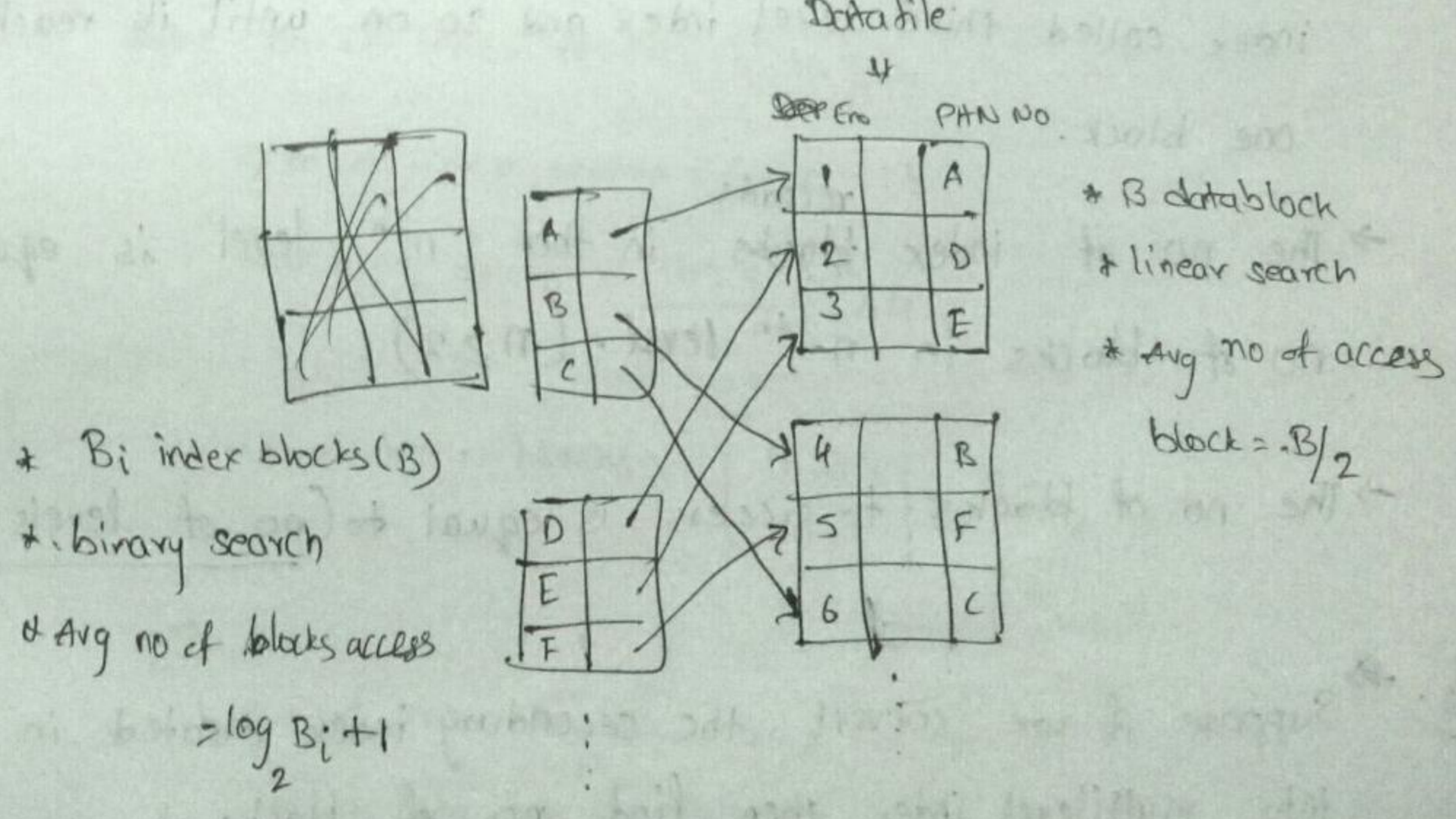
→ It is an ordered file with two fields.

→ First field is candidate key of the data file.

→ Second field is record pointer.

→ An index record is created for each record in the datafile
ie, no of index records = no of records in the datafile

→ This type of index is called dense index



Q: (3) Suppose if we create secondary index on the key field of the file of question i then find the avg no of blocks to access.

$$B.F = \frac{1024}{100} = 10$$

index record size = 15

no of data blocks = $\frac{30000}{10} = 3000$

no of index ~~blocks~~ records = 30000

no of BF (index) = $\frac{1000}{15} = 68$

no of index blocks = $\left\lceil \frac{30000}{68} \right\rceil = 442$

avg no of block accesses = $\left\lceil \frac{\log 442}{2} \right\rceil + 1$

$= 9+1 = 10$

Multilevel Index:

It is an index over an index

→ On any of the single level index we create a primary index called second level index on which we create a primary index called third level index and so on until it reaches to one block.

→ The no of index ~~blocks~~ records in the n^{th} level is equal is to no of blocks in $n-1^{th}$ level. $(n \geq 2)$

→ The no of blocks to access is equal to (no of levels + 1)

Q: → Suppose if we convert the secondary index created in (Q3) is into multilevel index then find no of blocks to access.

∴ no of index blocks in lvl 1 = 442

no of index records in lvl 2 = 442

no of index blocks in lvl 2 = $\left\lceil \frac{442}{68} \right\rceil = 07$

no of index records in lvl 3 = 7

no of index blocks in lvl 3 = $\lceil \frac{7}{68} \rceil = 1$

no of block access = 3+1 = 4 //

③ $r = 16384$ $R = 32$ $k = 6$ $B = 1024$ $B_p = 10$

non-key ordered, unspanned

Blocking factor = $\frac{1024}{32} = 32$ bytes

size of index record = 16 bytes

no of blocks in data file = $\frac{16384}{32} = 2^{19}$

no of index record = 16384

no of ~~index blocks~~ = $\frac{16384}{16} = 2^{20}$

~~Size~~ no of index records = 16384

size of index record = 6 + 10 = 16

blocking factor = $\frac{1024}{16} = 64$

no of index blocks = $\lceil \frac{16384}{64} \rceil = 256$ //

and

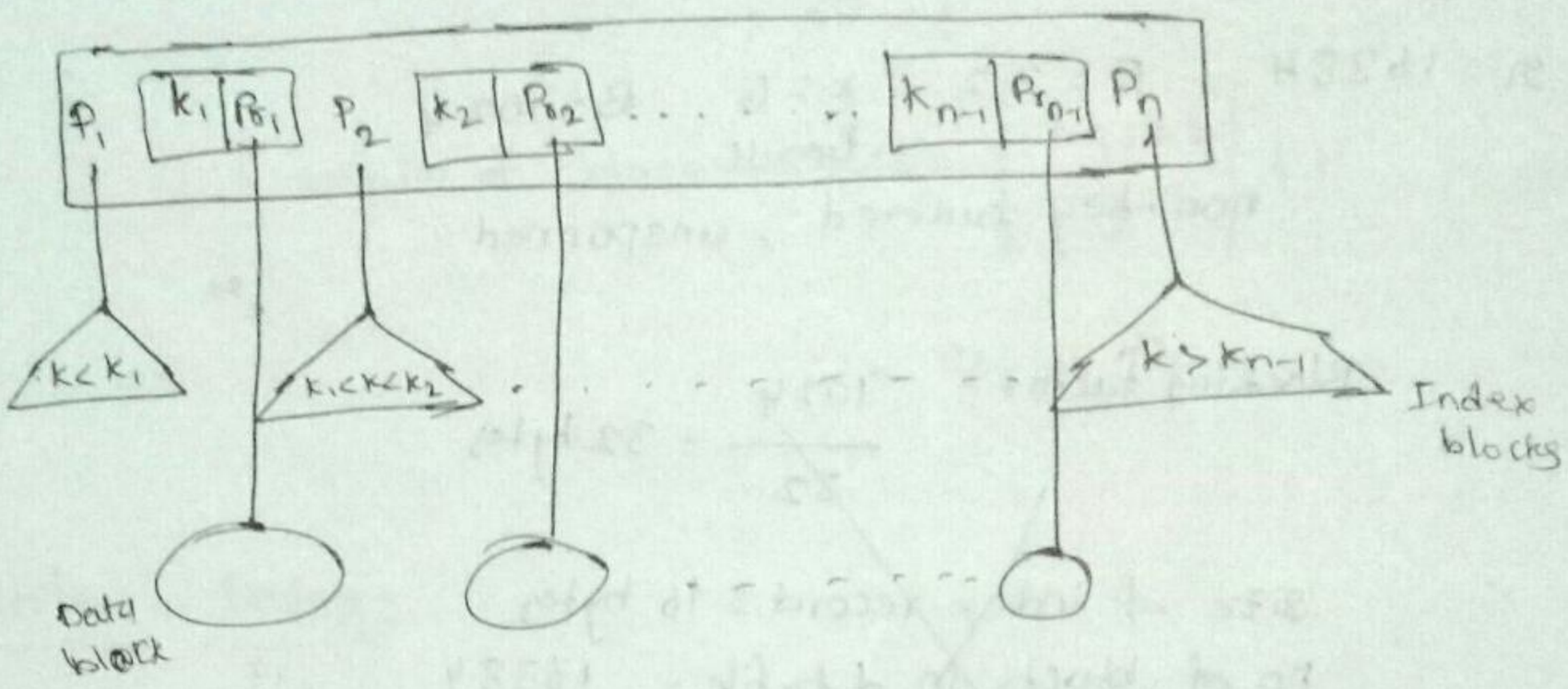
no of index records = 256

no of index block = $\lceil \frac{256}{64} \rceil = 4$ //

Implementation of multilevel index Using B-Tree (~~ordered index~~ i.e Direct access)

→ It is a height (ie) balanced search tree.

→ Each block in a multilevel index is treated as one node in B-tree



$P_1 \cdot P_2 \cdots P_{n-1}$ ← Block pointer / Tree pointer / index pointer

$k_1 < k_2 \cdots < k_{n-1}$ ← keys

$Pr_1\ Pr_2 \cdots Pr_{n-1}$ ← record / data pointers

→ B-Tree provides direct access to data items.

Order of B-tree (Let 'n'):

It is the maximum no of tree pointers in a node

n ← Tree pointers

(n-1) ← (key + record ptr)

$$n*p + (n-1)*(k+R) \leq B$$

Q: Calculate the order of B-Tree, suppose the key field of the file is 9 bytes, record ptr is 7 bytes, block ptr is 6 bytes and block of size 512.

$$6n + (n-1)*(9+7) \leq 512$$

⑥ $22n \leq 528$

$\boxed{n \leq 24}$

$\therefore$ order = 24

tree

Q: Calculate the no of records that can be stored in a B-tree of 3-levels for the above question. Assume that each node is 69 % full.

order = $24 * 0.69 = 16$

		16 pointers	15 index records
root :	1 node		
level1 :	16	$16 \times 16 = 256$	$16 \times 15 = 240$
level2 :	256	$256 \times 16 = 4096$	$256 \times 15 = 3840$
level3 :	4096	$4096 \times 16 = 65,536$	$4096 \times 15 = 61,440$

Total index records = $15 + 240 + 3840 + 61,440$

$= 65,535$

$\boxed{n = O^{l+1} - 1}$ $\quad O \rightarrow$ Order

Problems:

④

$5P + (P-1)(18) \leq 512$

$23P \leq 530$

~~$P \leq 23$~~

$P \leq \left\lfloor \frac{530}{23} \right\rfloor$

$P = 23$

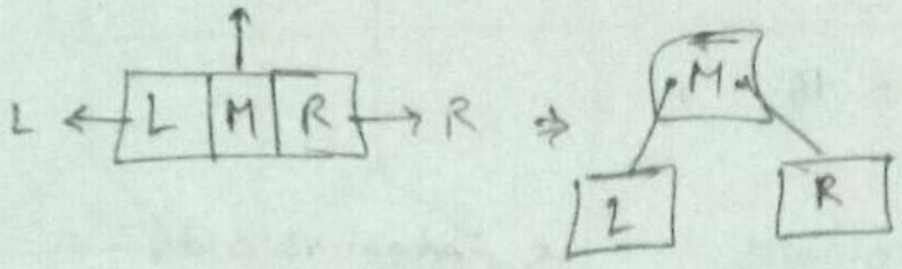
23) 530 (23
46

70
69

1

B-tree insertion:

Insert the key values always at the leaf level. If a node is full then you split the node into two nodes moving the middle element to its higher level and all the keys less than the middle key move to ~~this~~ the left subtree and greater than the middle move to the right subtree.



order : n

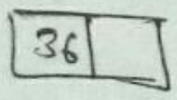
max : np , (n-1)k

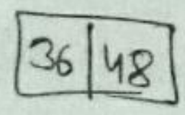
min : $\left\lceil \frac{n}{2} \right\rceil p , \left(\left\lceil \frac{n}{2} \right\rceil - 1\right)k$

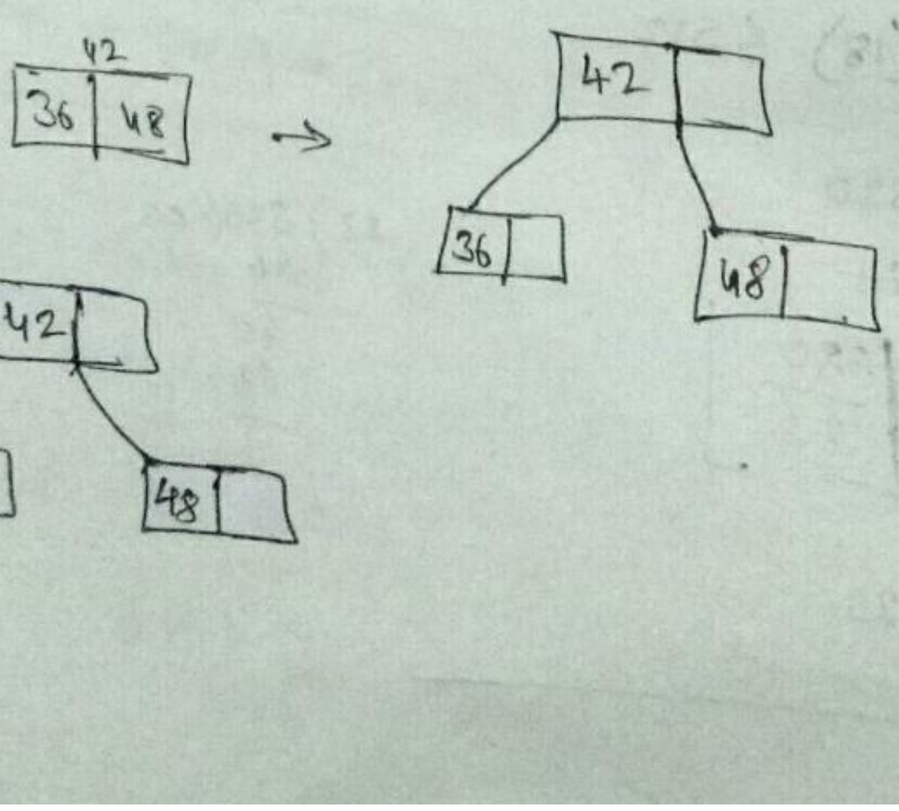
except for root node (but contains atleast 2 block pointers)

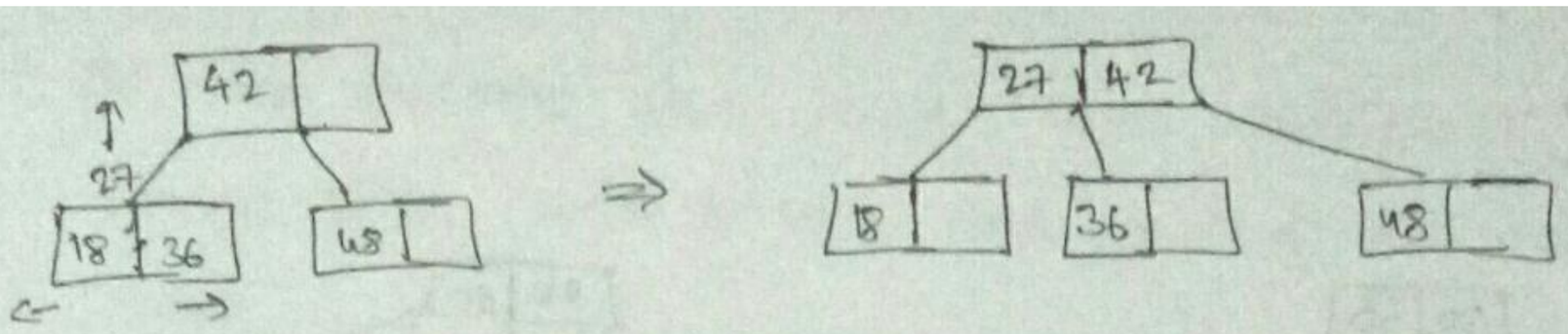
Q: Insert the following keys in a b-tree of order 3

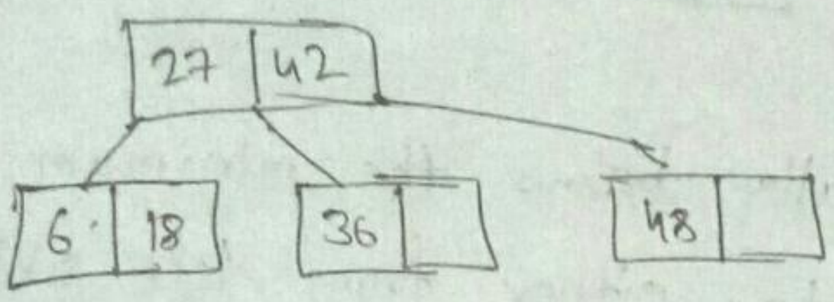
Keys : 36, 48, 42, 18, 27, 6, 24

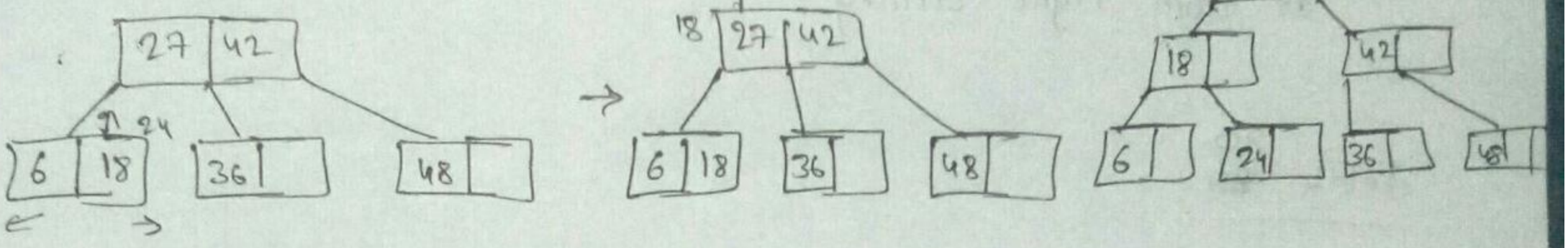












Expected type of questions:

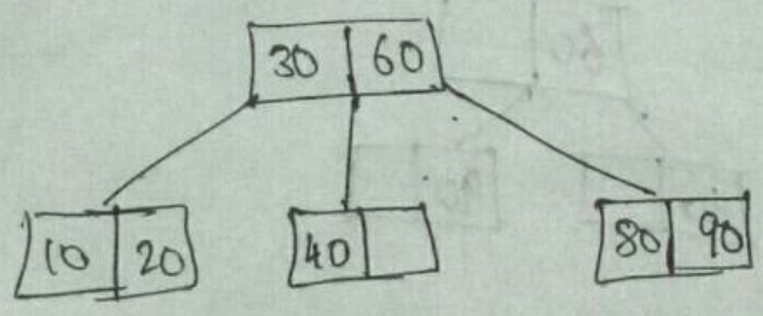
No of leaf splits = 3

No of root splits = 2

Total splits = 4

no of nodes 100% = 0

B-tree Deletion:

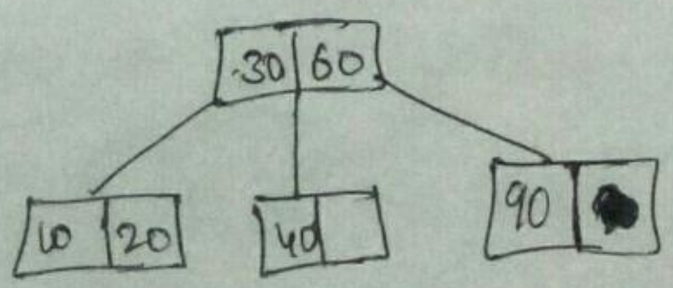


Here order = 3

max : 3P, 2k

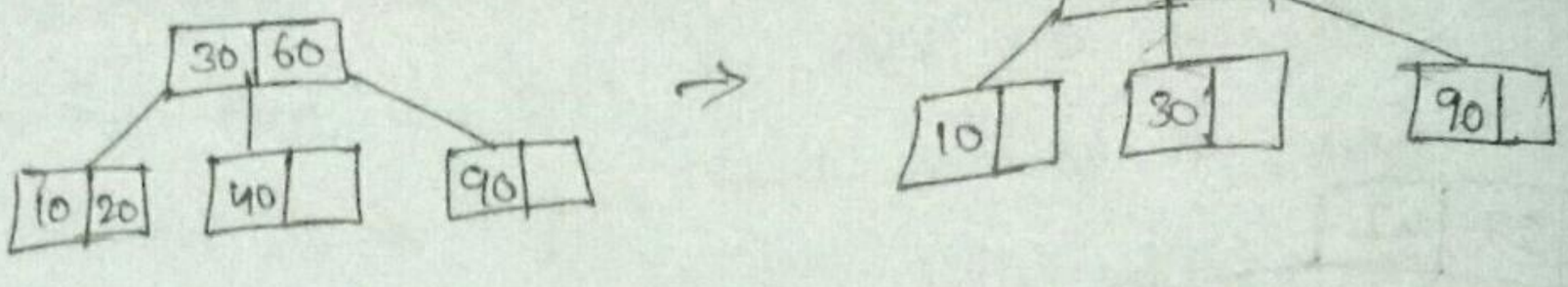
min : 2P, 1k

delete 80



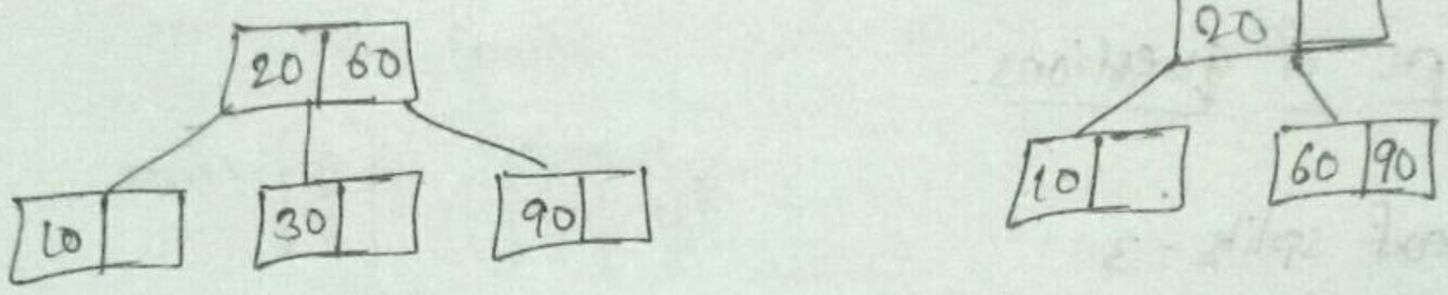
* If the node contains more than or equal to minimum keys then the deletion of a do not require any modifications

delete 40:



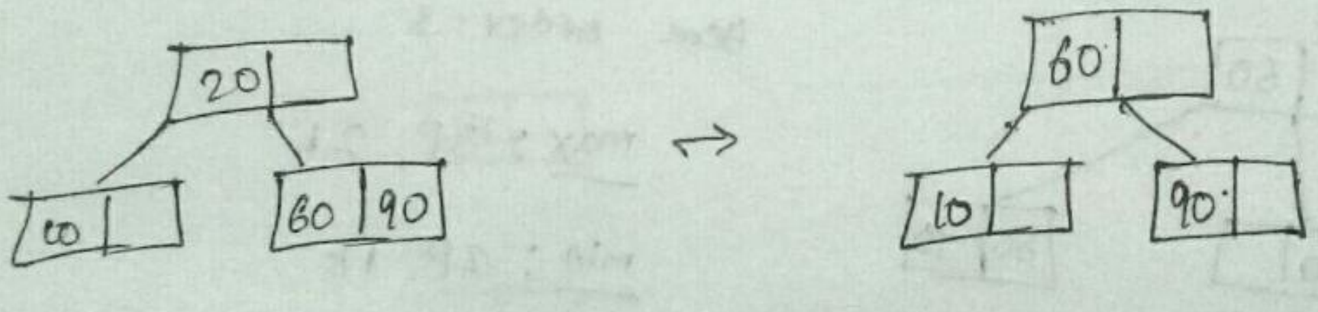
* If the deletion of a key falls below the minimum no of keys then borrow the key either from left sibling or from right sibling

delete 30:



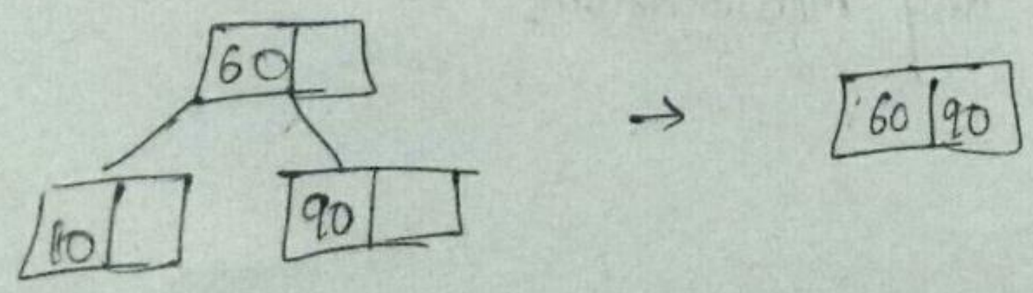
* If siblings do not have sufficient keys then borrow the keys from parent and merge with sibling

delete 20:



* If the key is deleted from the internal node then replace it with either largest key from left child or smallest from right subtree

delete 10:



* If ~~the~~ parents also don't have sufficient keys then merge the node parent and sibling to make one node.

B+-tree: (ordered access)

It is a modification of B-Tree data structure and there are two modifications suggested:

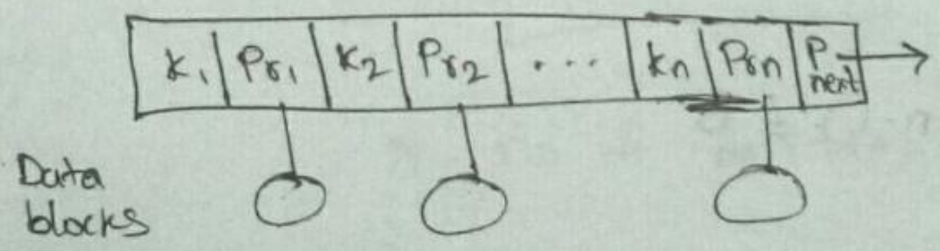
(i) Leaf node:

Remove tree pointers and contains keys and record pointer pairs.

(ii) Internal node:

Remove record pointers and contains keys and tree pointers.

Structure of leaf node:



$k_1 < k_2 \cdots < k_n$ ← keys

$P_{r1}, P_{r2} \cdots P_{rn}$ ← record / data ptr

P_{next} ← block / tree / index ptr

order of leaf node (Let 'n'):

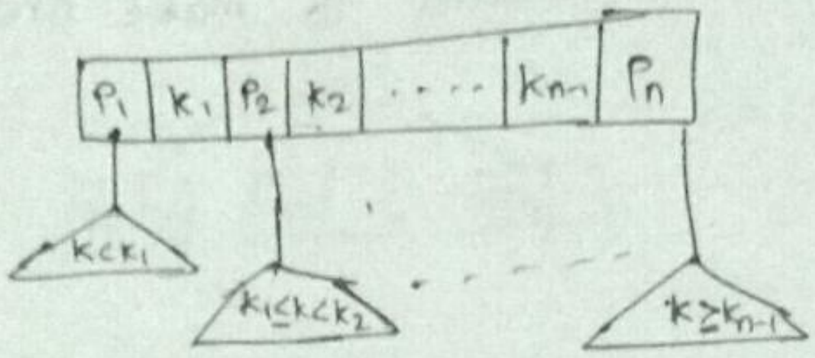
* It is the maximum no of index records in a node

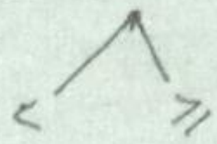
$n \leftarrow$ (key + Record ptr)

$1 \leftarrow P_{next}$

$$n*(k+P_r)+P_{next} \leq B$$

Structure of internal node (~~ordered access~~):





$P_1, P_2 \ldots P_n$ ← block/tree/index ptr

$K_1, K_2 \ldots K_{n-1}$ ← keys

Order of internal node (Let 'n'):

* It is the max no of tree pointers in a a node.

n ← tree pointers

(n-1) ← keys

$$n*P + (n-1)\,K \leq B$$

Q: Calculate order of B^+-tree suppose the key f file is 9 byte

$K=9,\ P_r=7,\ P=6,\ B=512.$

internal node:

① $n*P + K(n-1) \leq B$

$$6n + 9n - 9 \leq 512$$

$$15n \leq 521$$

$$n \leq \lceil \frac{521}{15} \rceil \qquad n \leq \frac{521}{15}$$

$$n \leq \lceil 34.7 \rceil \qquad n \leq 34.() \Rightarrow n = 34$$

leaf node:

$$n*(K+P_r) + P \leq B$$

$$n(9+7) + 6 \leq 512$$

$$16n \leq 506$$

$$n = 31$$

Q: calculate no of index records that can be stored in a B^+ tree of 3 levels for the above question. Assume that each node is 69% full.

- order of leaf node = $31 \times 0.69 = 21$
- order of internal node = $34 \times 0.69 = 23$

Root: 1 node — 23 tree pointers

lvl1 : 23 nodes — $23 \times 23 = 529$

lvl2 : 529 nodes — $529 \times 23 = 12,167$

lvl3 : 12,167 nodes — $12,167 \times 21 = 2,55,507$ records
(leaf level)

(compare ~~sem~~ the problem with the same problem dealt on B-tree with same conditions)

~~$n = d^l$~~ $\boxed{n = I^l \times L}$

n - no of ~~re~~ index records

I - ~~no of~~ order of internal node.

L - order of leaf node

l - ~~lev~~ level of tree without root node.

Problems

(8) $k=8$ $P=2$ $B=512$

$np + (n-1)k \leq B512$

$2n + 8n - 8 \leq$

$10n \leq 520$

$n = 52$

(9) $B = 1024 \quad P_R = 7 \quad k = 9 \quad P = 6$

$$n(k + P_R) + P \leq 1024$$

$$16n + 6 \leq 1024$$

$$16n \leq 1018$$

$$n \leq 63$$

(10) $k = 12 \quad B = 1024 \quad P_R = 10 \quad P = 8$

$$np + (n-1)k \leq 1024$$

$$8n + (n-1)12 \leq 1024$$

$$20n \leq 1034$$

$$n \leq 51.(0)$$

$$n = 51$$

$\therefore$ max no of keys $= 51 - 1 = 50$

(1) $B = 512 \quad k = 9 \quad P = 6 \quad P_R = 7$

lvl 3 B+ tree

From (Q1)

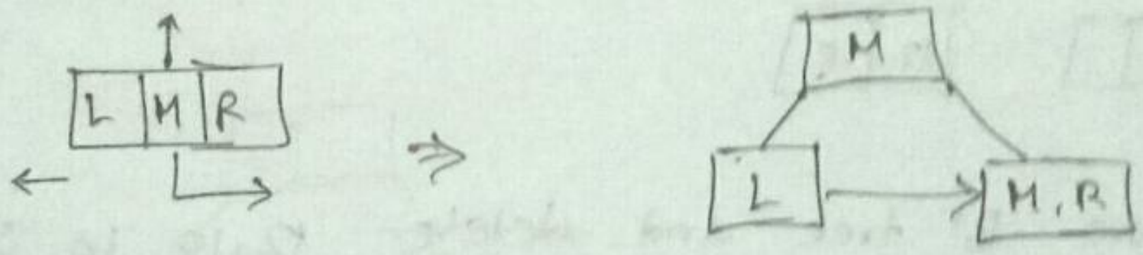
order of internal node $= 34$

$$n = 8 I^{l} \times L$$

$$n = 34^{3} \times 31$$

B+ tree insertion:

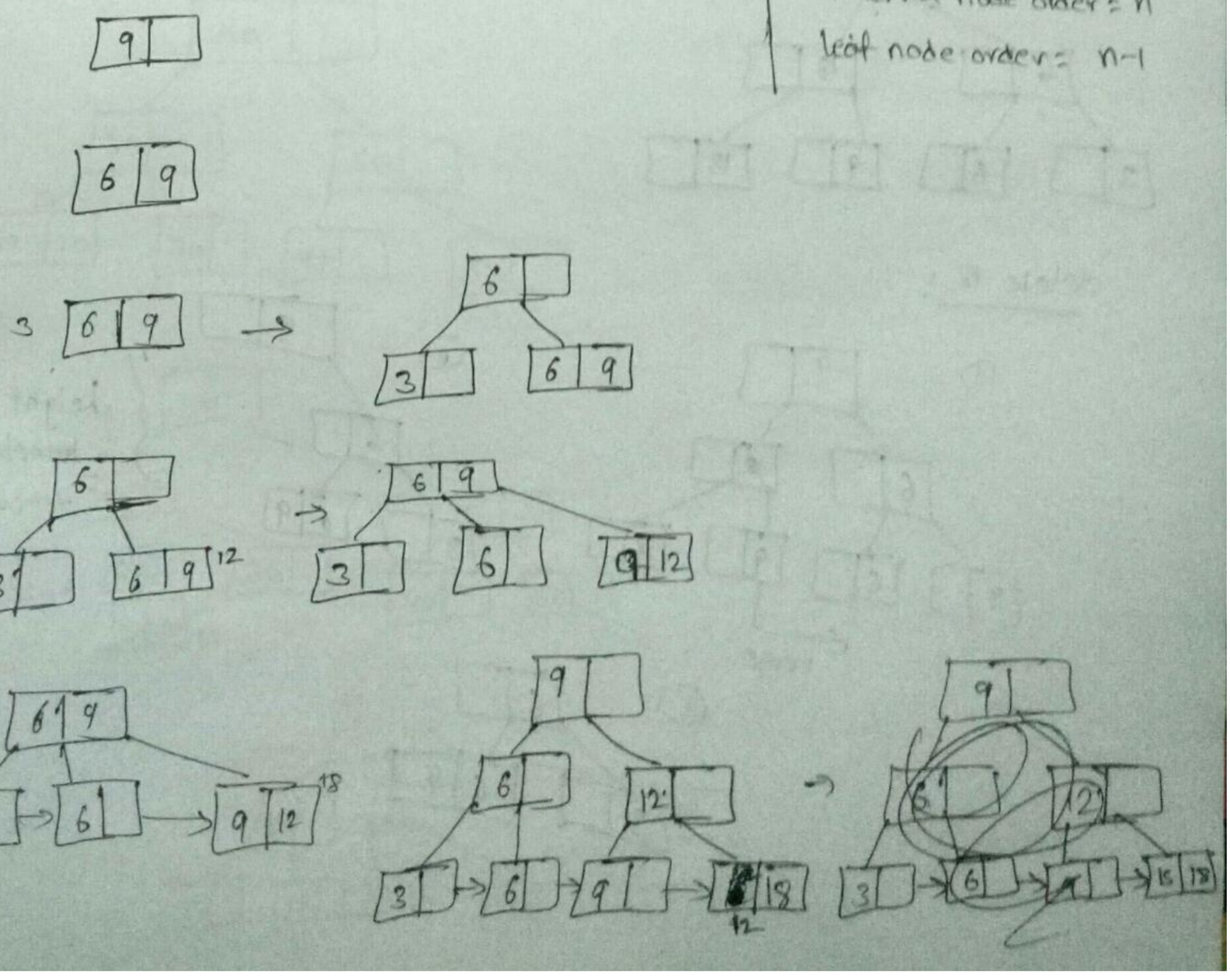
Inserting the key values into B+ tree is same as inserting the key values into B-tree except the leaf node split. i.e., If a node is full then we split the node into two nodes moving the copy of middle to the upper level and all the keys less than the middle move to the left subtree and greater than or equal to middle move to the right subtree

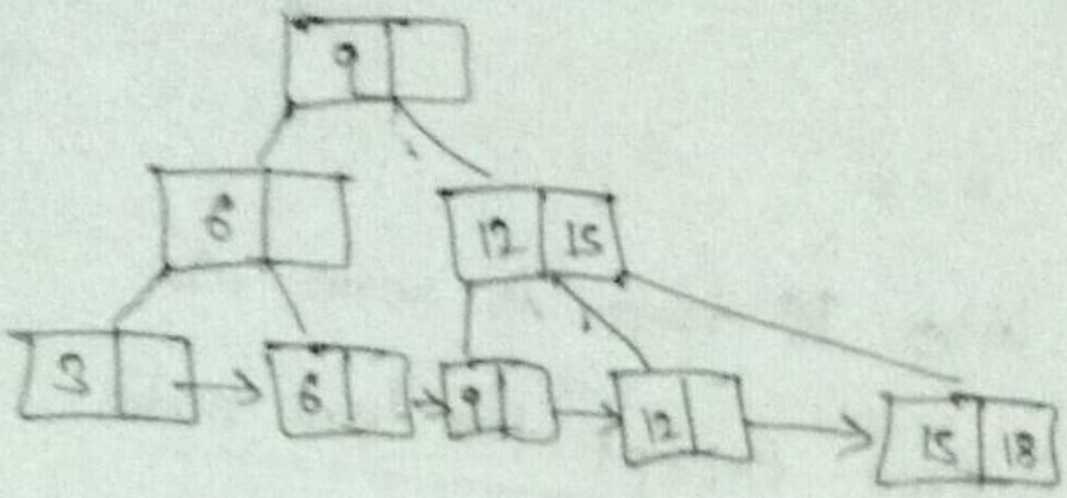


Q: Insert the following key in a B+-tree of order 3.

keys: 9, 6, 3, 12, 18, 15

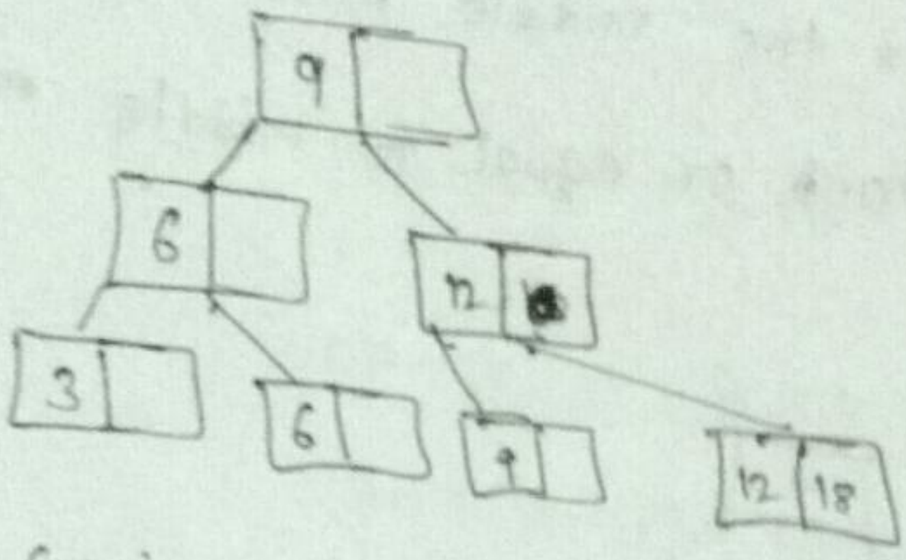
if order is n
Internal node order = n
leaf node order = n-1





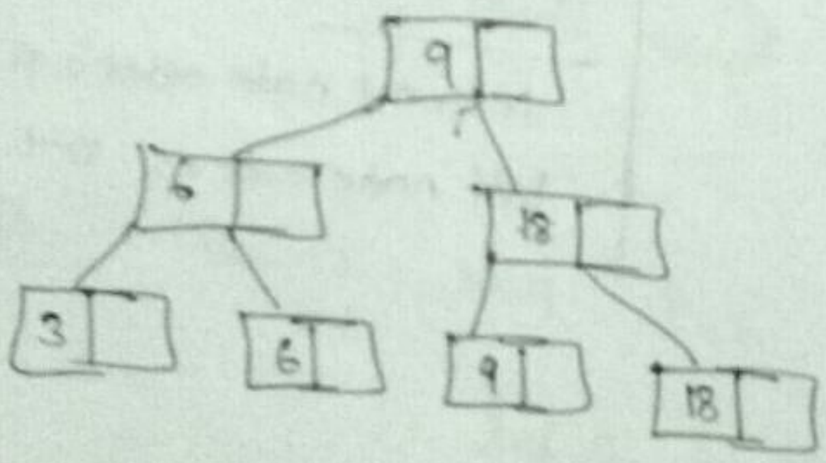
B+ tree - Deletion:

Q:

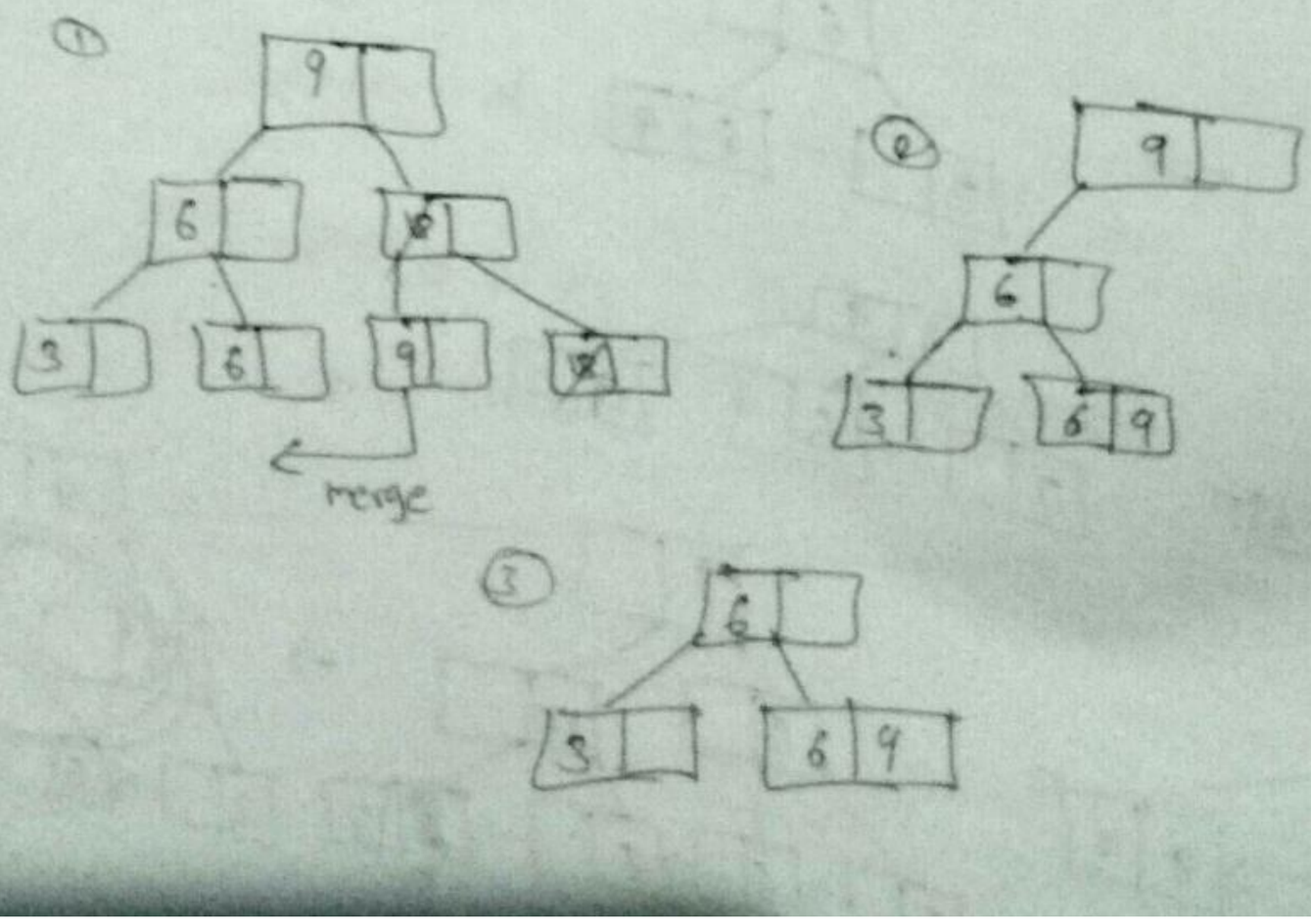


Consider the above B+ tree and delete 12, 18 in sequence

delete 12:



delete 18:



Problems:

(12)

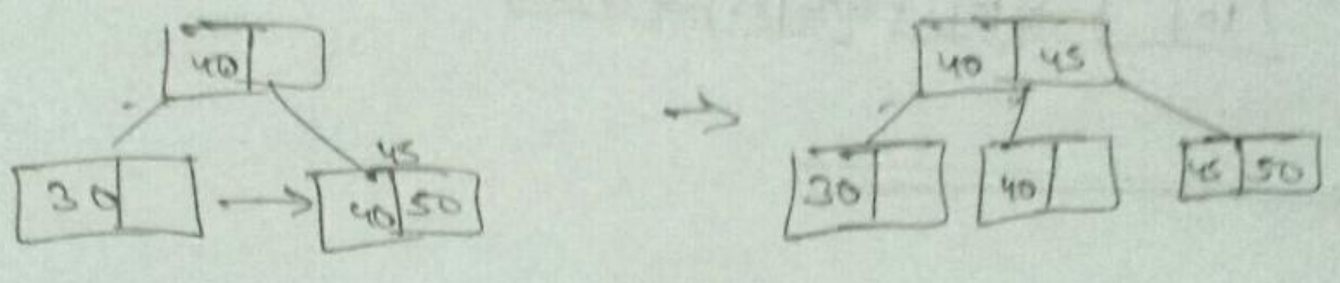
max keys = 5

order = 6

min pointers $= \left\lceil \frac{6}{2} \right\rceil = 3$

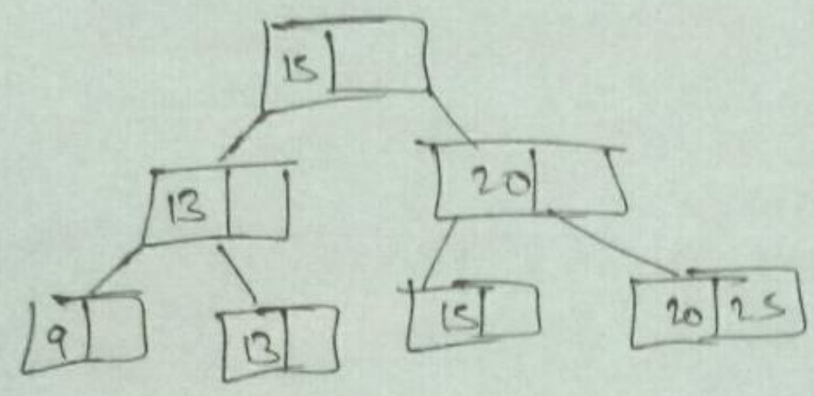
∴ min keys = 3 − 1 = 2 keys

(13)



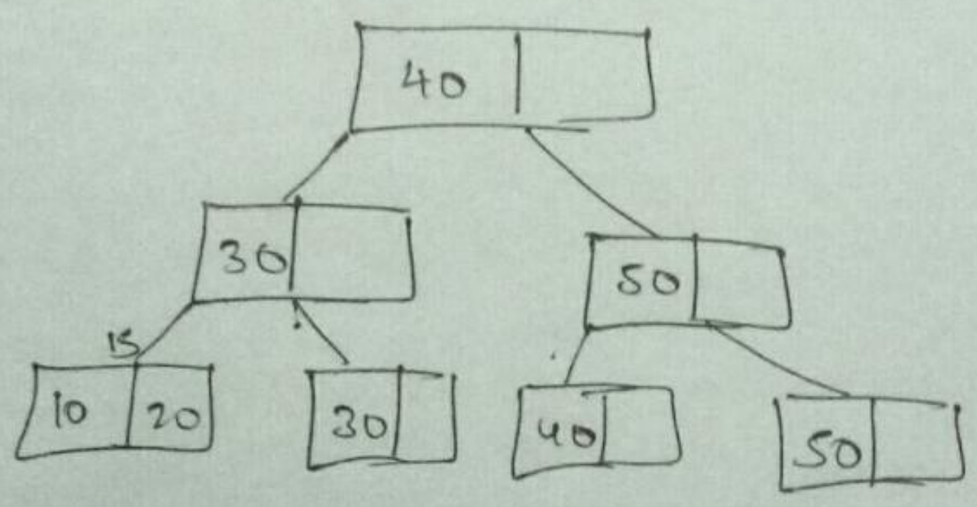
∴ none

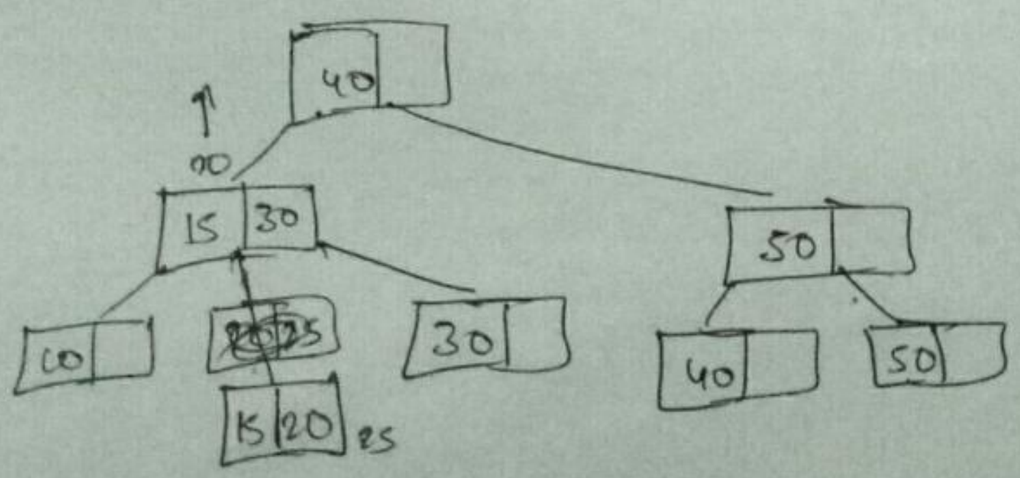
(14)

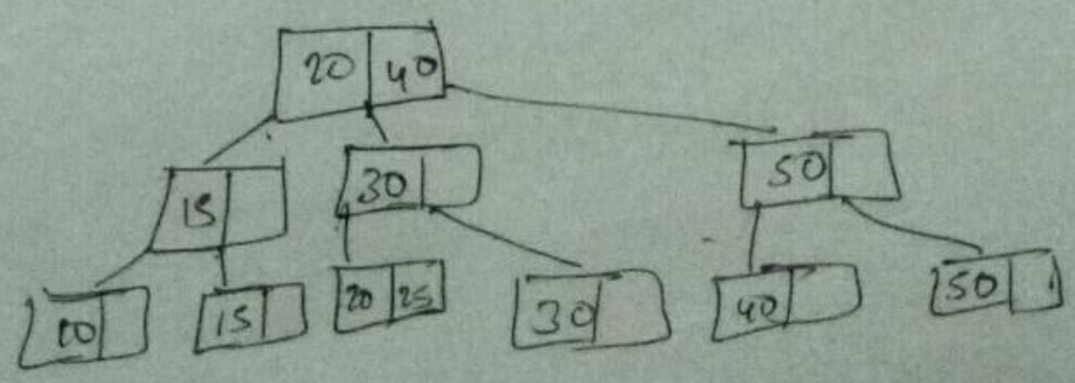


Ans: d

(15)







∴ 1

(16) delete 50