File organisation

Data base is a Collection of files, each file is a Collection of records each record is a sequence of fields.

Database
|
Files
|
Records
|
Fields.

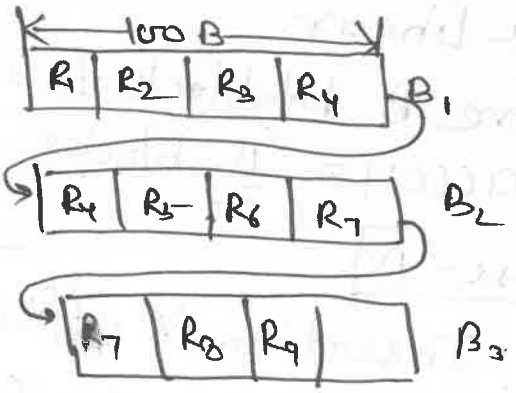
Blocking factor: Blocking factor is the average no. of records per block.

Strategies for storing file of records into Block:

① **Spanned strategy:** It allow, Partial Part of record can be stored in a block.

Adv: No wastage of memory.
Dis: Block access increases.

It is suitable for variable length record.



Un-Spanned strategy: No Record can be stored in more than 1 Block.

* **Dis:** Wastage of memory.
Advantage: Block accesses reduced.
* Suitable for fixed length Record.

rganization of records in a file:-

① **Ordered file organization**: All records of file are ordered based on some search key value

Searching: Binary.

If we have B data blocks to accen record. the average no. of block accen = $\lceil \log_2 B \rceil$

Adv: searching is efficient.

Disadvantage: Instertion is costly due to re-organization of the enitre file.

② **Un-ordered file organization**: All file of records are inserted at whene ever the place is available. usually at the end of file

Searching :- Linear.

If we have B-datablock to accen a record the avg # of block accen = $\frac{B}{2}$ blocks.

$\boxed{\text{Worst Case} = B}$

Advantage: Insertion is efficient.

Disadvantage:- Searching is inefficient compared to ordered file organization.

File organisation

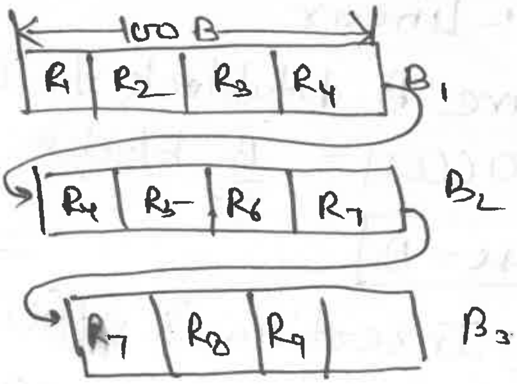
Data base is a Collection of files, each file is a Collection of records each record is a sequence of fields.

Database
|
Files
|
Records
|
Fields.

Blocking factor: Blocking factor is the average no. of records per block.

Strategies for storing file of records into Block :

① **Spanned strategy** :- It allow, Partial Part of record can be stored in a block.



Adv :- No wastage of memory.

Dis :- Block access increases.

It is suitable for variable length record.

Un-Spanned strategy :- No Record can be stored in more than 1 Block.

* Dis : Wastage of memory.

Advantage : Block accesses reduced.

* Suitable for fixed length Record.

Organization of records in a file:-

① **Ordered file organization:** All records of file are ordered based on some search key value.

Searching: Binary.

If we have B data blocks to access record, the average no. of block access $= \lceil \log_2 B \rceil$

Adv: Searching is efficient.

Disadvantage: Instertion is costly due to re-organization of the enitre file.

② **Un-ordered file organization:** All file of records are inserted at where ever the place is available, usually at the end of file.

Searching:- Linear.

If we have B-data block to access a record the avg # of block access $= \frac{B}{2}$ blocks.

$\boxed{\text{Worst Case} = B}$

Advantage: Insertion is efficient.

Disadvantage:- Searching is inefficient compared to ordered file organization.

Index :- Index are used to improve the searching efficiency.
Index is a record consists of two fields :

Key	Block Ptr.

Pointer to a Block where key is available.

* index is an ordered file.
* Searching :- Always binary.
* To access a record using index the avg no. of block access $= \lceil \log_2 B \rceil + 1$.
* Index can be created on any field of relation. (Primary key, Non-key).

Index

- Single level Index
  - Primary index (P.K + Ordered data)
  - Clustered index (Non-key + Ordered)
  - Secondary index (N.K/C.K + Un-ordered)
- Multilevel index
  - B-tree
  - B^+-Tree

Classification of indexes: ① Dense index
② sparse index.

Dense index: If an index entry is created for every search key value that index is called dense index.

Spars Index: if an index entry is created only for some search key values it is called sparse index.

Primary index:- (P.K + Ordered) A primary index is an ordered file whose records are of fixed length with two fields the first field is same as primary key of datafile and the second field is a pointer to data block. where the key is available.

* index entry is created for first record of each block called "block anchor".
* The no. of index entries is equal to the no. of datablocks.
* The avg no of block accen using index $= \lceil \log_2 B_i \rceil + 1$, where B_i index block.
* The type of index is called sparse index because it is indexing only first record of each block.

ex:- Suppose that we have an ordered file of 30,000 records stored on a disk. with block size 1024 B. File records are of fixed length and are un-spanned of size 100 B and suppose that we have created a primary index on the key field of the file of size 9B and a block pointer of size 6 B then find the avg no. of blocks to search for a record using with and without index?

Ans: Records $= 30{,}000$ ordered.

Blocksize $= 1024$ B

Record size $= 100$ B

$B.F = \left\lfloor \frac{1024}{100} \right\rfloor = 10$ record/block.

no. of data blocks $= \left\lceil \frac{30000}{10} \right\rceil = 3000$ blocks.

Avg no. of block access $= \lceil \log_2 3000 \rceil = 12$ (without indexing)

Size of index blocks $= 9+6 = 15$ B

No. of index records/Block $= \left\lfloor \frac{1024}{15} \right\rfloor = 68$

no. of index records $=$ no of datablocks $= 3000$

no. of index Block $= \left\lceil \frac{3000}{68} \right\rceil = 45$

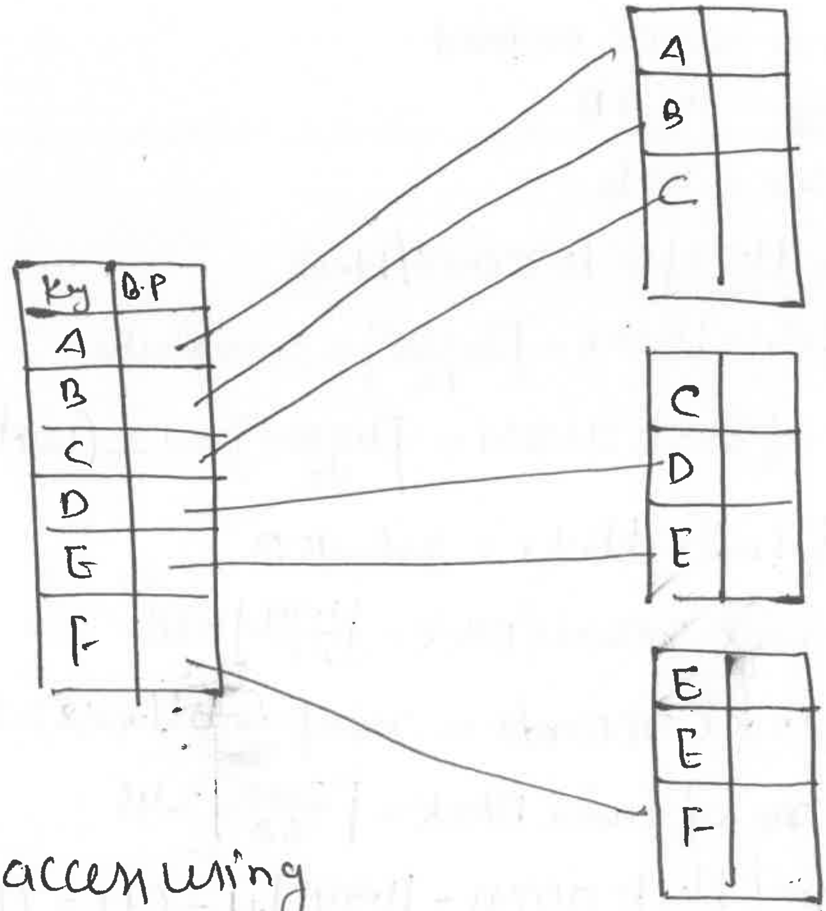
Avg. no. of block access $= \lceil \log_2 45 \rceil + 1 = 6+1 = 7$ (with indexing)

Clustered Index (N.K + ordered):

clustered index is an ordered file with two fields, the first field is same as the clustering fields is called non-key and the second field is a block pointer

Cluster index is created on data file whose file records are physically ordered on a non-key field which doesnot have a distinct value for each record that field clustering field.

Index entry is created for each distinct value of a clustering field. The block pointer points to the first block in which the key is available, type of index is dense



\# of block access using cluster index $\geq \lceil \log_2 B_i \rceil + 1$

<u>Secondary index</u> (N.K/C.K + Un-ordered)

Secondary index provides a secondary means of accessing a file for which some primary access already exists. Secondary index may be on a non-key or a C.K.

* index entry is created for each record in a datafile
* No. of index entries = No. of records.
* Type of secondary index is Dense

Q:- Previous data

Soln: records = 30000 un-ordered

B.Size = 1024B

Record size = 100 B

K = 9B, B_p = 6B

$B.F = \left\lfloor \frac{1024}{100} \right\rfloor = 10\ r/B$

no. of data blocks $= \left\lceil \frac{30000}{10} \right\rceil = 3000$ blocks

the avg no. of block. accen $= \frac{3000}{2} = 1500$

Size of index record = 15B

no. of index r/B $= \left\lfloor \frac{1024}{15} \right\rfloor = 68$

no. of index records = 30000 = Data records.

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

Avg no. of Block accen $= \lceil \log_2 442 \rceil + 1 = 9+1$ blocks.

Multilevel index: As single level index is an ordered file we can create a Primary index to the index itself. In this case the original file is called I^st level index and the index to index is called 2nd level index

We can repeat the above Procen untill all index entries fil in One diskblock.

Q. Find the avg. no of block accesses required to search for a record if multilevel index is created on the datafile of.

Ans: $B.F = 10\ r/B$

no of data block = 3000

I^{st} level : no. of index block = 442

2^{nd} level : no. of index records = 442 (no. of I^{st} level block)

no. of blocks = $\lceil \frac{442}{68} \rceil = 7$

3^{rd} level : Block no. of index record = 7.

no. of blocks = $\lceil \frac{7}{68} \rceil = 1$

Avg no. of block access = 1+1+1+1

Note:- If there n levels in multilevel index, the no. of block access to search for a record = n+1

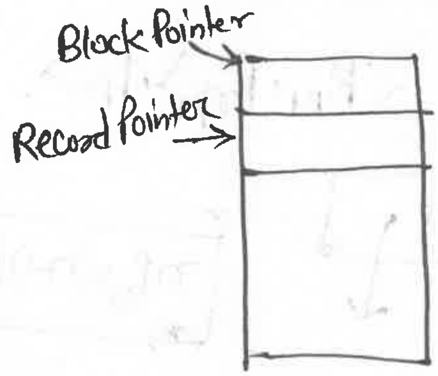
B-Trees and B^+-Trees

→ Generalization of Multilevel indexing.

→ These are multilevel indexes with small modifications.

Terminology: (i) node Pointer (or) block Pointer.

(ii) Record Pointer :- Record Pointer Points to the record we are searching for.



→ In B-Trees at every level we are going to have key and Data Pointer, and that data Pointer actually Pointing to either block or record.

Properties of B-Trees:

Root: A root of the B-Tree can have children between 2 and P.

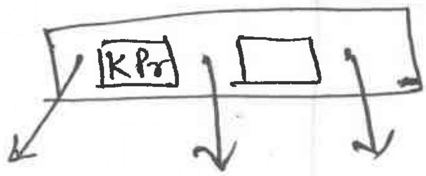
- where P is ~~Killed~~ Called as order of tree
- Order of trees means maximum number of children a node can have.

Internal nodes: → Internal node can have children between "$\lceil \frac{P}{2} \rceil$" and "P"

→ Internal node can have keys between "$\lceil \frac{P}{2} \rceil - 1$" and "$P-1$".

→ Internal node is arranged as below.

$$\langle P_1 \langle K_1, Pr_1 \rangle P_2 \langle K_2, Pr_2 \rangle \ldots \langle K_{P-1}, Pr_{P-1} \rangle, P_P \rangle$$



$$* \boxed{nP_B + (n-1)(K+P_r) \le B}$$

Leaf node:-

⇒ A ~~node~~ Leaf node can have keys between $\lceil \frac{P}{2} \rceil - 1$ and $P-1$.

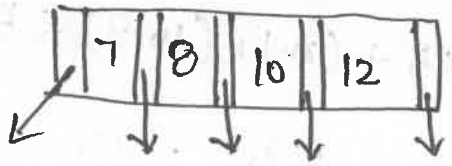
Example Take $P=5$

	min # of keys	maxi # of keys
Root	1	4
Internal node	2	4
Leaf	2	4

Underflow and overflow in B-trees:

⇒ Consider B-tree of order P, if number of search key Values in a B-tree node exceeds "P-1", then this condition is called overflow.

ex: Consider a B-tree of order 5



Insert 9

$$\#\text{of search key value} < \lceil \frac{P}{2} \rceil - 1 \quad \text{(underflow)}$$

Search: ⇒ Searching in a B-tree is similar to BST, however rather than moving left or right at each node we need to Perform a "P-way" search. ~~to see which subtree to Bob.~~

$$\text{Time complexity} = \log_P n$$

n - # of search key

P - order of B-Tree

Q: Consider a B-tree with Key size 10 B, block size 512 B, data Pointer is 8 B and block Pointer is 5 B. What is order of B-tree is ?

$$nP_B + (n-1)[K+P_r] \leq 512$$

$$n*5 + (n-1)(10+8) \leq 512$$

$$n = \frac{530}{23} = 23.$$

Q: Suppose order of B-tree is 23. Then how many max index records stored in 4-levels [Including root as 1-level] across the B-tree?

	Index Record	Node
1	22	1
	+	
2	22X23	23
	+	
3	$22*23^2$	23^2
	+	
4	$22X23^3$	23^3

Insertion: 1. Search to determine which leaf ~~ern~~ node will hold a key.

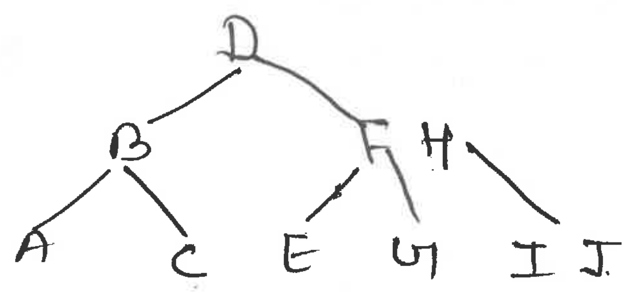
2. If leaf node have space, insert key in ascending order.

3. Otherwise split leaf nodes key into two Parts, and Promote median key to the Parent.

4. If the Parent node is full, recursively split and Promote median key to it's Parent.

5. If a Promotion is made to a full root node, split and create a new Root node holding only the Promoted median key.

ex:- Inserting A, B, C, D, E, F, G, H, I, J consider order of B-tree 4; max # of keys = 3
min # of keys = 1



ex:- Insert the following keys in B-Tree of order-4

Keys:- 2, 5, 10, 11, 1, 6, 9, 4, 3, 12, 18, 20, 25.

Deletion from a B-Tree

Steps: 1. If the key to be deleted is not in a leaf, swap it with successor (or Predecessor) under the natural order of keys. The delete key from the leaf.

2. If leaf contains more than the minimum number of keys then one can be deleded with no further action.

3. Otherwise, if the node contains the minimum number of keys consider the two immediate siblings of the node.

4. If one of the siblings has more than the minimum number of keys, the redistribute one key from this sibling to the ~~Pa~~ Parent node, and one key from the Parent to the deficient node.

5. If both immediate siblings have exactly the minimum # of keys, then merge deficient node with one of the sibling node, and ~~the~~ one entry from the Parent node.

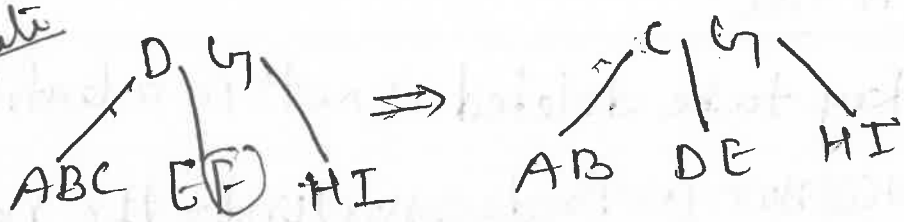
6. If this leaves the Parent node with too few keys then the Process is propogate upward.

ex:

Removing a key from a leaf node leaving l-keys in the leaf node If $l \geq \lceil \frac{P}{2} \rceil - 1$ then we can stop. If $l < \lceil \frac{P}{2} \rceil - 1$, we must reblance tree to correct this.

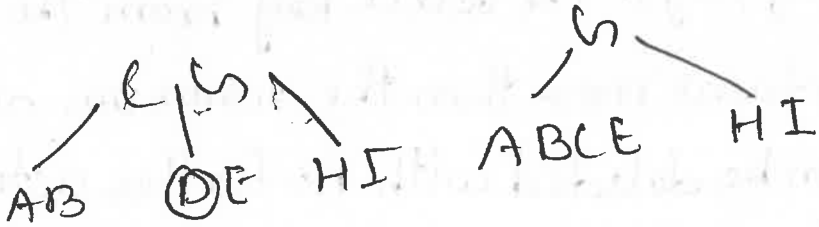
Borrow: Consider order-5 B Tree

F Delete



Coalesce:

Delete D



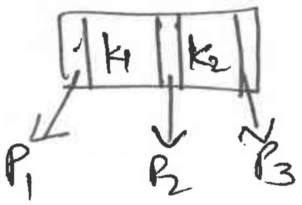
B+ Trees :- Order - P - B+ tree

(8)

⇒ Internal nodes has "$\lceil \frac{P}{2} \rceil$" to 'P' children.

⇒ Internal node has "$\lceil \frac{P}{2} \rceil - 1$" to "P-1" search key values.

Case for $n = 3$

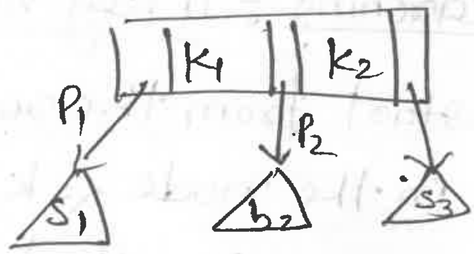


Leaf

Non-leaf node :- Each Key-Search values in subtrees S_i Pointed by P_i, $K_i >= K_{i-1}$

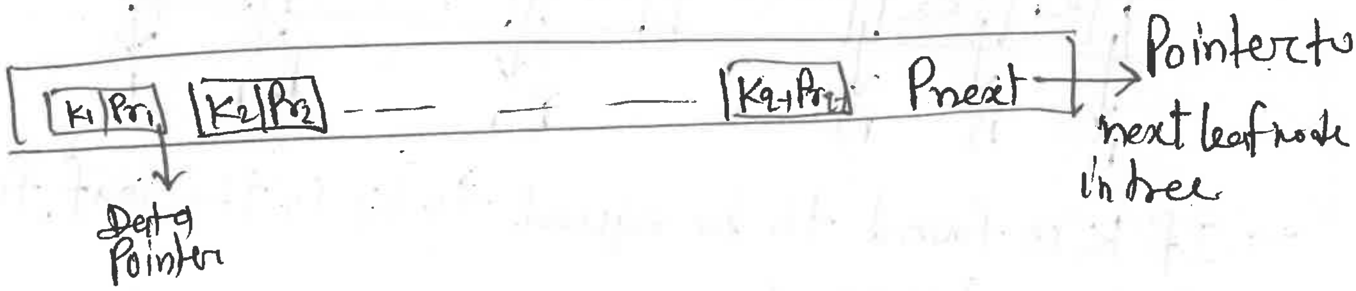
Key values in $S_1 < K_1$

$K_1 \leq$ Key values in $S_2 < K_2$



Leaf-node : 1. Each Leaf node is of the form

$<<K_1, Pr_1>, <K_2, Pr_2> \ldots\ldots <K_{q-1}, Pr_{q-1}>>$



ex: Search key field is V = 9B, the block size 512B, a record Pointer P_r = 7B, A block Pointer P = 6B. what is the order of internal node and leaf node?

Soln: $n P_B + (n-1)K \leq 512$

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

$n = 34$

$n[P_r + K] + P_B \leq 512$

$n[9+7] + 6 \leq 512$

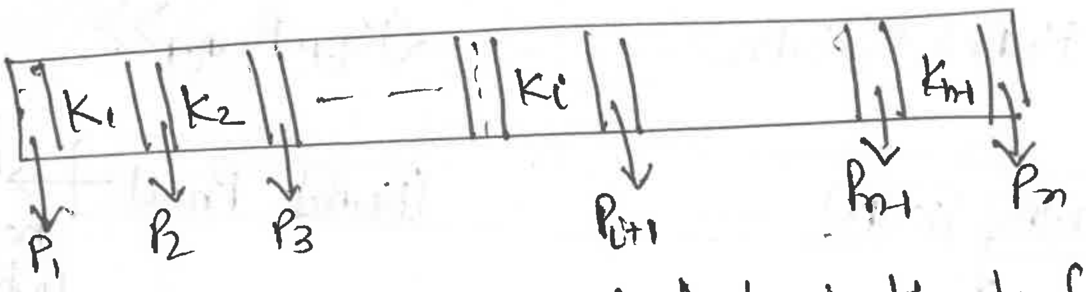
$n = 31$

Searching: a key-value k:

→ start from the root, look for the largest key value (K_i) in the node <= K.

→ Follow the Pointer P_{i+1} to next value, untill reach the leaf node

$K_i <= K < K_{i+1}$



→ If K is found to be equal to K_i in the leaf, follow P_{ri} to search record.

$\boxed{\text{T. Complexity} = \log_P n}$

Overflow: When # of search key values exceed "P-1"

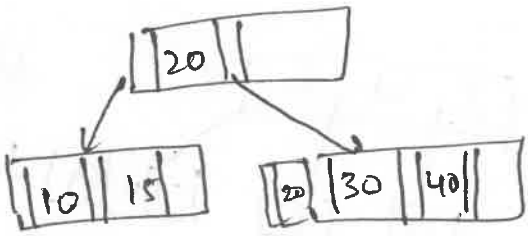
Leaf node: Split into two nodes

- 1st node contain $\lceil \frac{(P-1)}{2} \rceil$ values.
- 2nd node contain remaining value.
- Copy the smallest search key value of the 2nd node to the Parent node

Internal node: order-5 B^+ tree

ex: | 10 | 20 | 30 | 40 |

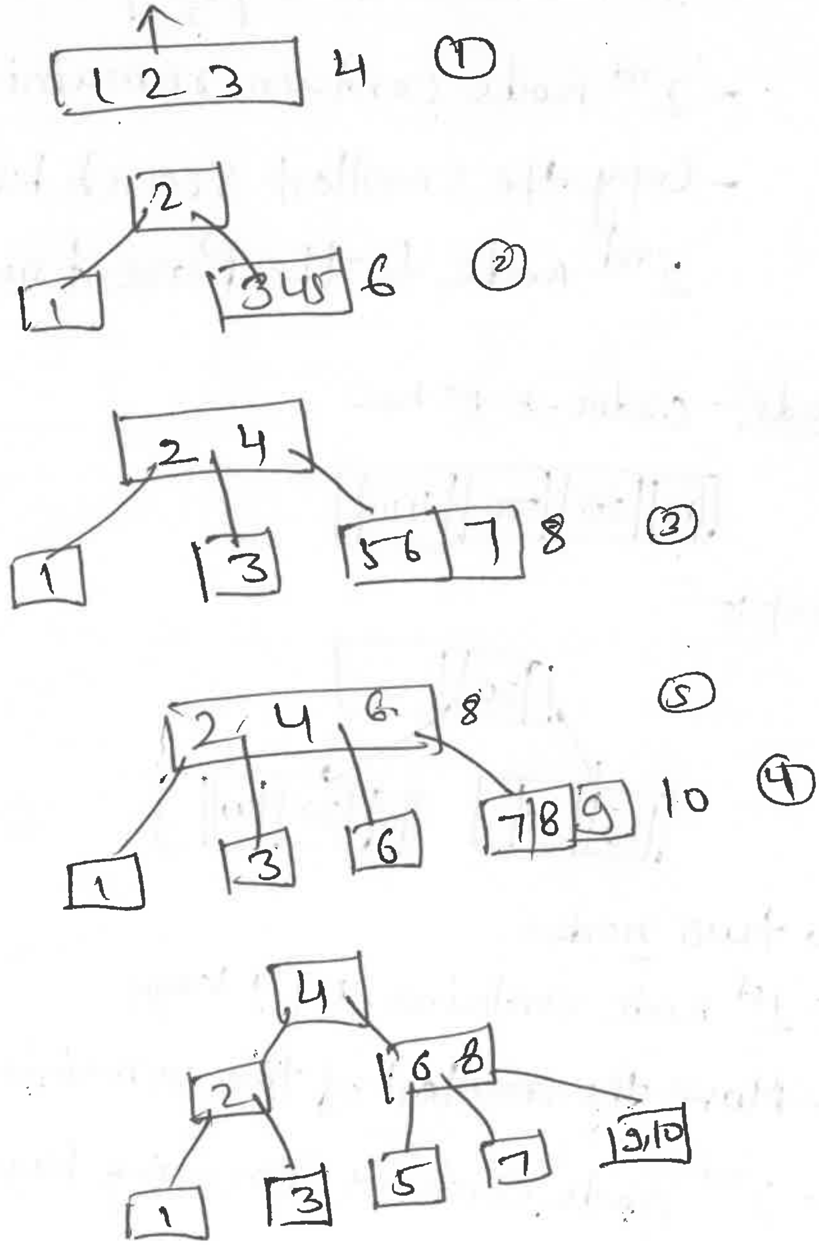
Insert 15



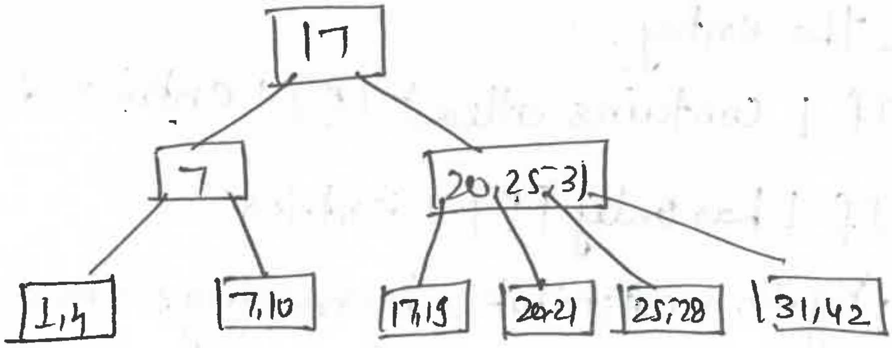
→ Split into two nodes

- 1st node contains $\lceil \frac{n}{2} \rceil - 1$ keys.
- Move the smallest of the remaining keys to the Parent.
- 2nd node contains remaining keys.

Q. [Imp] A B-Tree of order 4 is built from scratch by 10 successive insertions. What is the maximum # of node splitting operations that may take place?



Example: Construct a B^+ tree for (1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42) with P=4 (order is 4).



Note:- Why B/B^+ Trees Preferred over Binary Search Trees like (AVL, Red black .. etc) for storing index records and accessing them.

⇒ In Binary trees every node can have only 2 children therefor they tend to grow in height.

⇒ Because of it even database is small, the number of levels we might have access before we get to the final level is actually large.

Deleting a Data entry from B^+-tree :

⇒ start at root, find leaf L where entry belongs

⇒ Remove the entry.

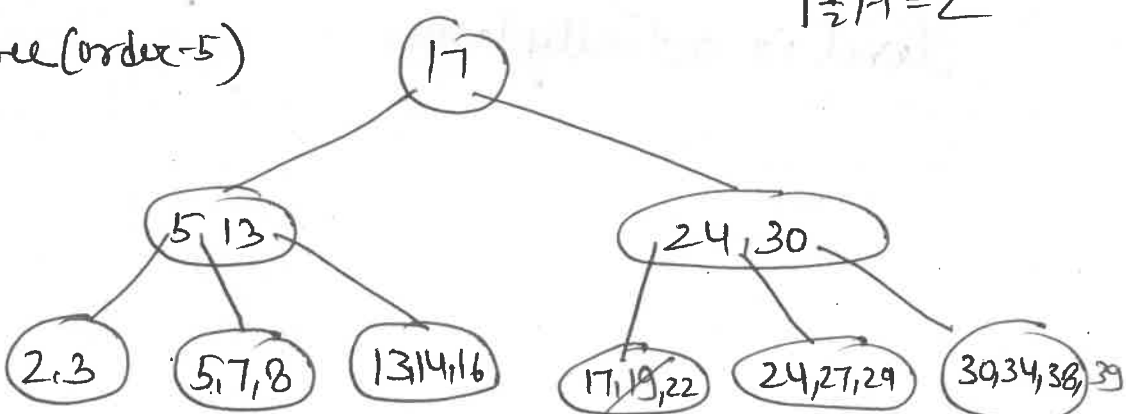
- ⇒ If L contains atleast $\lceil \frac{p}{2} \rceil - 1$ entries, done!
- ⇒ If L has only $\lceil \frac{p}{2} \rceil - 2$ entries
  - Try to redistribute, borrowing from sibling (adjacent node with same parent as L).
  - If redistribution fails, merge L and a sibling.
- ⇒ If merge occured, then corresponding entry from parent must be deleted.

Merge could propagate to root decreasing height.

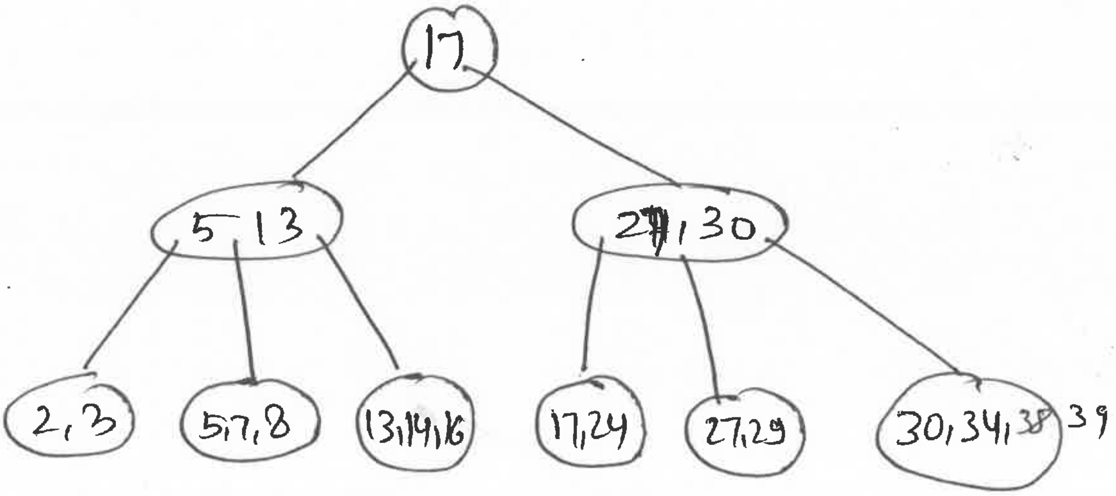
If the deleted entry present in internal node replace it with in order successor.

$\lceil \frac{p}{2} \rceil - 1 = 2$

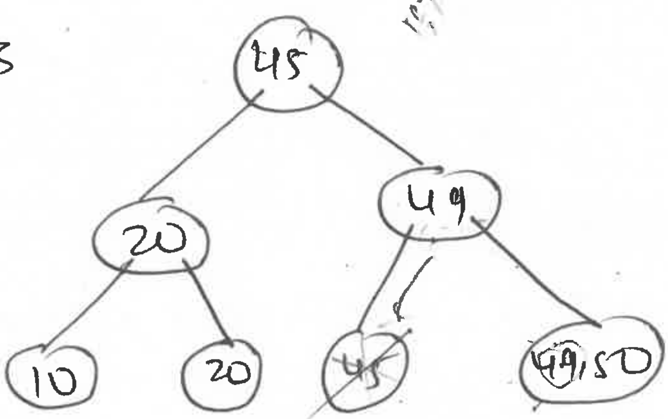
ex:- Consider B^+-tree (order-5)



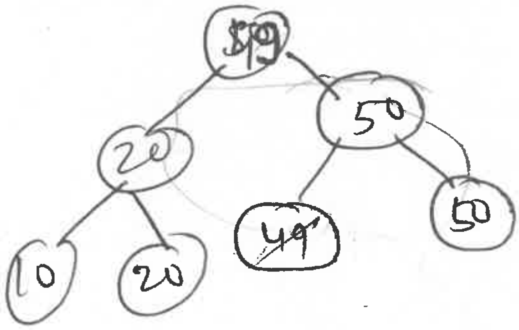
Delete 19, 22



ex:- order-3



Delete 45



Delete-49

Delete

Q.1 Which One of the following Statements in Not Correct about the B+ tree data structure used for creating an index of a relational database table ?

(a) Each leaf node has a Pointer to the next leaf node.
(b) Non leaf node have Pointer to data str.
(c) B^+ Tree is a height-balanced tree
(D) Key values in each node are kept in sorted order.

Q.2 In a B^+-Tree, if the Search-key value is 8 bytes long, the block size is 512 bytes and the block Pointer size is 2B then the maximum order of B^+ tree is ___

$$nP_B + (n-1)K \le 512$$

$$n \times 2 + (n-1)8 \le 512$$

$$n = 52.$$

Q.3 B^+ Trees are considered BALANCED because

(a) the lengths of the Paths from the root to all leaf nodes are all equal.
(b) the lengths of the Paths from the root to all leaf nodes differ from each other by at most 1.
(c) the number of children of any two non-leaf sibling nodes differ by at most 1.
(d) the number of records in any two leaf nodes differ by at most 1

④ Consider a B^+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 byte long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is ____

$$(n)(K+P_r)+P_B \le 1024$$

$$n[12+10]+8 \le 1024$$

$$n \le \frac{1024-8}{22}$$

$$n \le 46.18$$

$$n = 46$$

$$nP_B+(n-1)K \le 1024$$

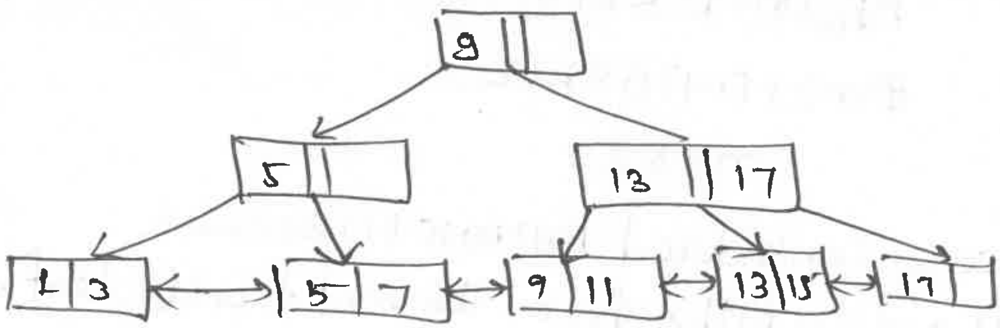
$$n8+12n-12 \le 1024$$

$$n \le \frac{1036}{20}$$

$$n = 51$$

$$keys = 51-1 = 50$$

⑤ With reference to the B^+ tree index of order 1 shown below, the minimum number of nodes (including the Root node) that must be fetched in order to satisfy the following query: "Get all records with a search key greater than or equal to 7 and less than 15" is __5.__



(6) A file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index. Then that index is called.

(a) Dense (b) Sparse (c) Clustered (d) Unclustered

(7) Consider a B^+ tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node?

(a) 1 (b) 2 (c) 3 (d) 4

(8) A clusting index is defined on the fields which are of type.

(a) Non-key and ordering
(b) Non-key and non-ordering
(c) key and ordering
(d) key and non-ordering.

(9) The order of a leaf node in a B^+ tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1KB, data record pointer is 7 bytes long, the value field is 9B and a block pointer is 6 byte long. What is the order of leaf node?

(a) 63 (b) 64 (c) 67 (d) 68

$$n[K+P_r]+P_B \leq 1024$$

$$n[9+7]+6 \leq 1024$$

$$n \leq \frac{1018}{16}$$

(10) ~~In a database file structure, the search key field is~~ -

A B-tree used as an Index for a large database table has four levels including the root node. If a new key is inserted in this index, then the maximum number of nodes that could be newly created in the Process are

(a) 5 (b) 4 (c) 3 (d) 2

Q.11 Which of the following is correct?

(a) B-trees are for storing data on disk and B^+ trees are for main memory.

(b) Range queries are faster on B^+-trees.

(c) B-trees are for Primary indexes and B^+-trees are for Secondary indexes.

(d) The height of a B^+-tree is independent of the # of records.

Q.12 B^+-trees are Preferred to binary trees in database.

(a) Disk capacities are greater than memory.

(b) Disk access is much slower than memory access

(c) Disk data transfer rates are much less than memory data transfer rates

(d) Disks are more reliable than memory.

Q.13 A B^+ tree index is to be built on the Name attribute of the relation STUDENT. Assume that all student names are of length 8 B, disk blocks are of size 512 B and index pointers are of size 4 B. Given this scenario, what would be the best choice of the degree (i.e. the number of pointer/Node) of the B^+ tree?

(a) 16 (b) 42 (c) 43 (d) 44

$K = 8B$, $B.S = 512B$

$P_B = 4B$

$$n \cdot 4 + (n-1)8 \le 512$$

$$n \le \frac{520}{12} = 43$$

Q.14 The order of an internal node in a B^+ tree index is the maximum # of children it can have. Suppose that a child pointer takes 6B, the search field value takes 14 bytes and the block size is 512 B. What is the order of the internal node?

(a) 24 (b) 25 (c) 26 (d) 27

$$6n + (n-1)14 \le 512$$

$$n \le \frac{526}{20} = 26$$

Q.15 Which one of the following is a key factor for preferring B^+ trees to binary search trees for indexing database relations?

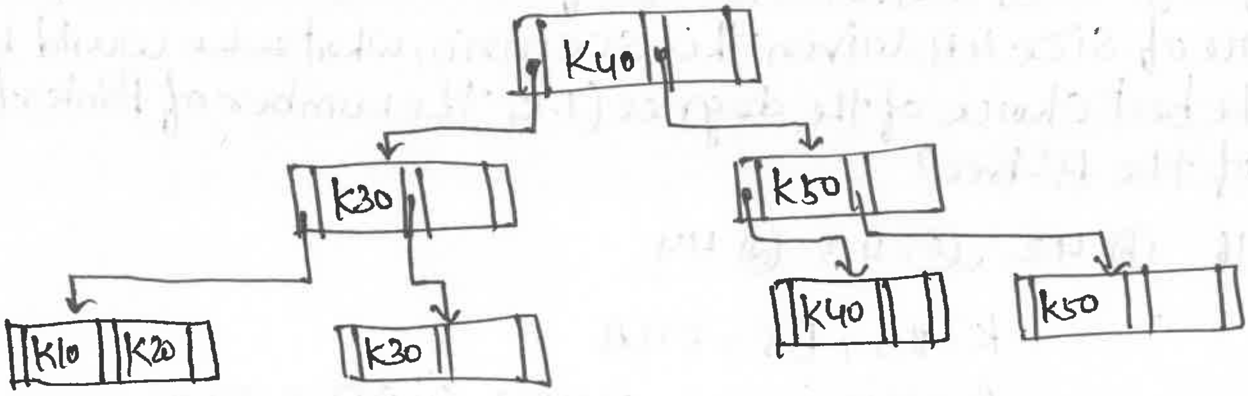
(a) Database relations have a large # of record.

(b) Database relations are sorted on the primary key.

(c) B^+ trees requires less memory than binary search trees.

(d) Data transfer from disk is in blocks.

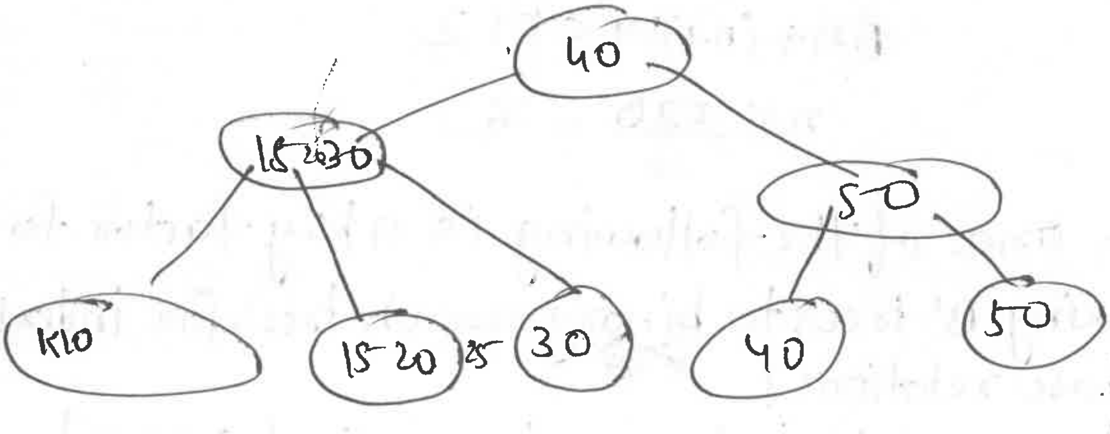
Q.16 Consider the B^+-tree in the adjoining figure, where each node has atmost two keys and three links.

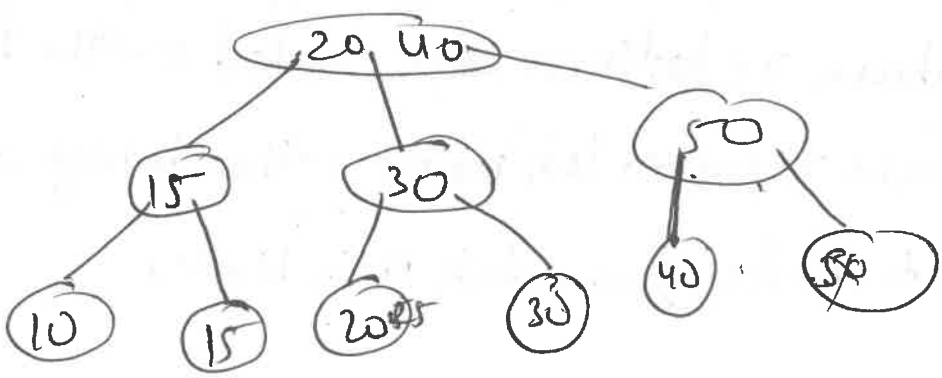


Keys K15 and then K25 are inserted into this tree order. Exactly how many of the following nodes (disregarding the links) will be present in the tree after the two insertion?

| K30 | K50 | | K25 | K30 | | K20 | K25 | | K15 | K20 |

(a) 1 (b) 2 (c) 3 (d) 4





(17) Now the key K50 is deleted from the B+-Tree resulting after the two insertions made earlier. Consider the following statements about the B-tree resulting after this deletion.

✓(i) The height of the tree remains the same
✓(ii) The node (disregarding the links) is present in the tree.
✗(iii) The root node remains unchanged (disregarding the links)

(18) Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 B. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 B and the size of a block pointer is 10 B. If the secondary index is built on the key field of the file, and a multilevel index scheme is used to store the secondary index, the # of first-level and second-level blocks in the multilevel index are respectively

(a) 8 and 0 (b) 128 and 6 (c) 256 and 4 (d) 512 and 5

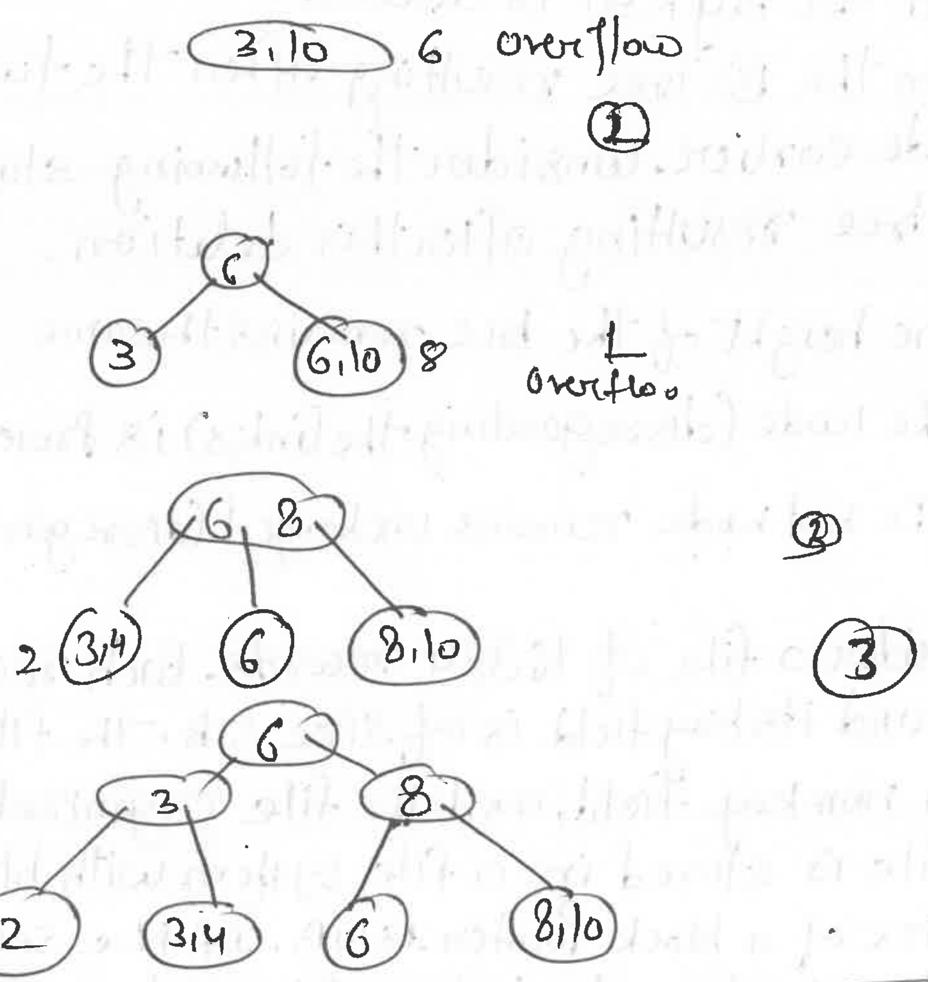
(19) The following key values are inserted into a B+-tree in which order of the internal nodes is 3 and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum # of tree pointer in each node, and the order of leaf nodes is the maximum # of data items that can be stored in it. The B+-tree is initially empty.

10, 3, 6, 8, 4, 2, 1

The maximum # of times leaf nodes would get split up as a result of these insertion is

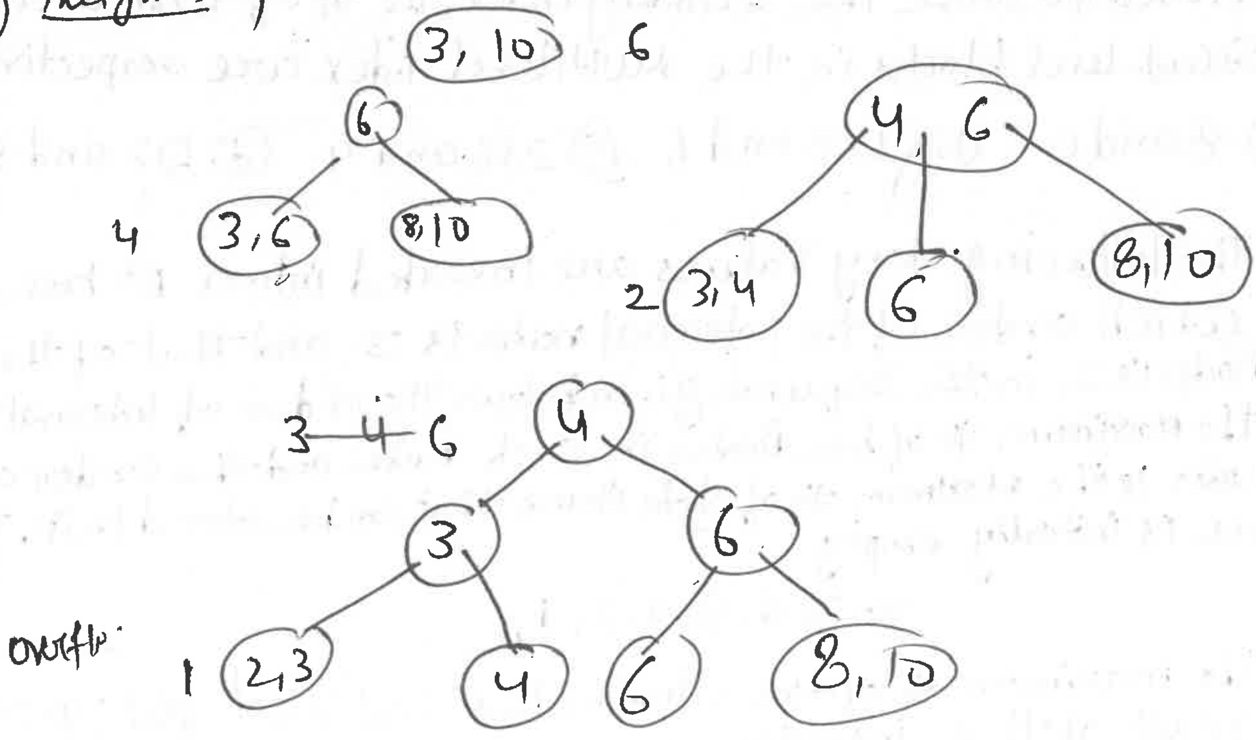
(a) 2 (b) 3 (c) 4 (d) 5

Sol^n:



This gives 3

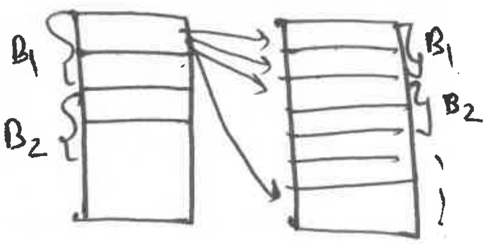
4 This gives 4



20: A database Table T_1 has records 2000 and occupies 80 disk blocks. Another table T_2 has 400 records and occupies 20 disk blocks. These two tables have to be joined as per a specified join condition that needs to be evaluated for every pair of records from these two tables. The memory buffer space available can hold exactly one block of records for T_1 and one block of records for T_2 simultaneously at any point in time. No index is available on either table.

(1) If Nested-loop Join algorithm is employed to perform the join, with the most appropriate choice of table to be used in outerloop, the number of block access required for reading the data are.

(a) 80000 (b) 40080 (c) 32020 (d) 100



Here we will take T_2 as the outer table in nested join algorithm. The number of block access then will be $20 + 400 \times 80 = 32020$

(2) If, instead of Nested-loop Join, Block nested-loop Join is used, again with the most appropriate choice of table in the outerloop, the reduction in the # of block accesses required for reading the data will be.

(a) 0 (b) 30400 (c) 38400 (d) 798400

Soln: ~~for every~~ block ~~of~~ in T_1 we need to load all blocks of T_2, so # of block access is $80 \times 20 + 20 = 1620$

So, difference is $32020 - 1620 = 30400$