m-way Search Tree
(or)
Multi-way Search Tree

→ m-way search trees are generalised versions of Binary Search Tree.

→ In m-way Search Tree, m stands for order of tree. Order of tree is the maximum number of children of a node.

For example: Binary Tree is a 2-way tree because every node in a Binary Tree has maximum two children.

→ m-way search tree has m children / pointers / sub-trees and m-1 keys in one node.

Properties of m-way Search Tree

(1) Each node can point at most m subtree.

(2) The number of keys in each node are maximum m-1 and each node can have maximum m pointers.

(3) m-way search tree representation :

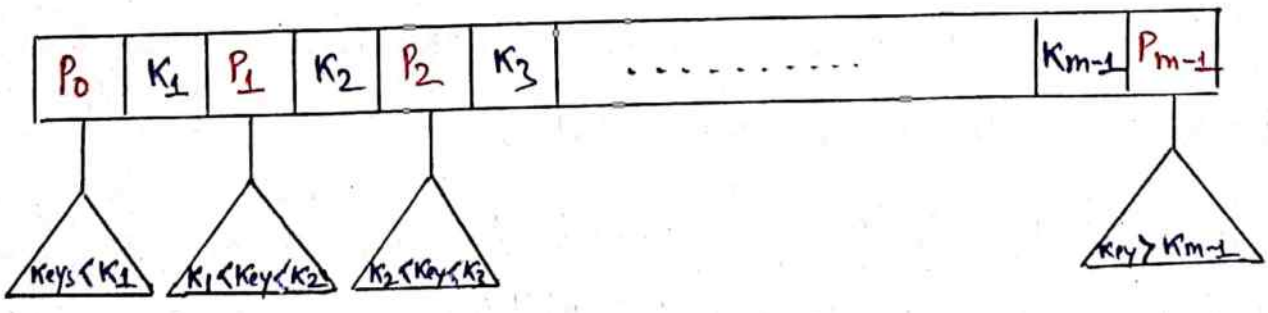


fig. m-way search tree Node Structure.

$K_1 < K_2 < K_3 \cdots\cdots K_{m-1}$

P_i is a pointer to subtree or node and $0 \le i \le m-1$.

(4) All the Keys values in the subtree pointed by P_i will be less than K_{i+1} and Keys pointed by P_{i+1} will be greater than K_i

(5) The sub-trees are the m-way search trees.

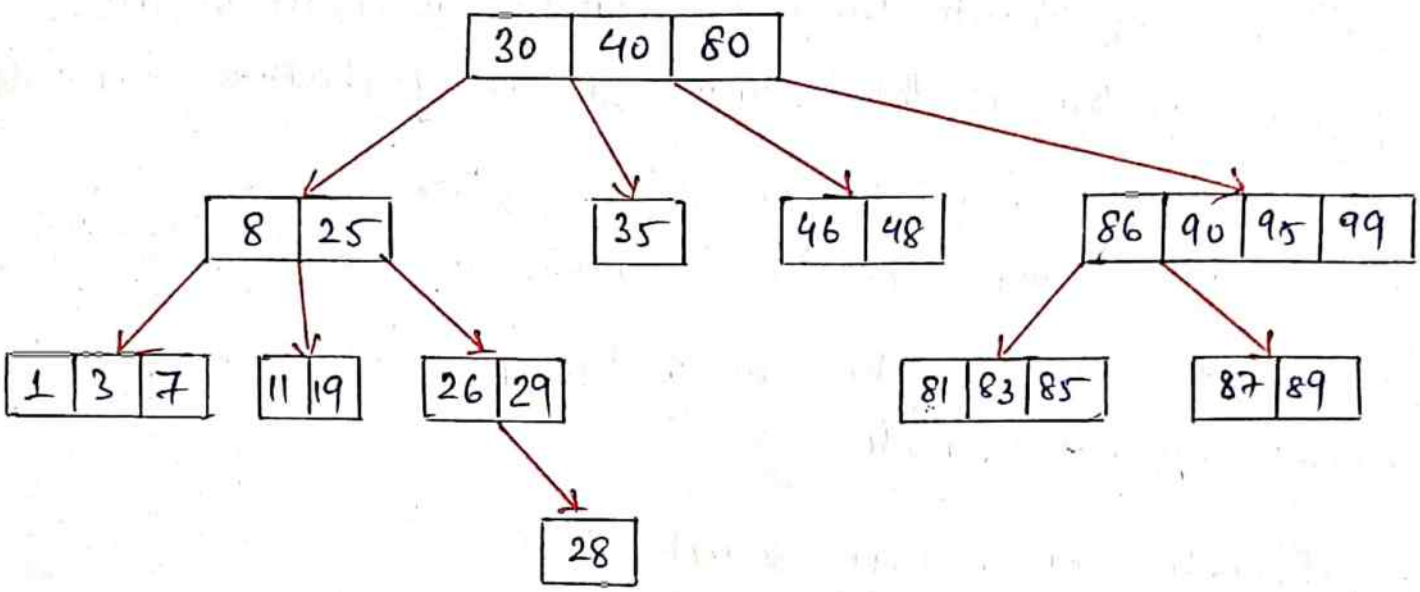


fig: multi-way search tree of order 5
or
5-way search tree

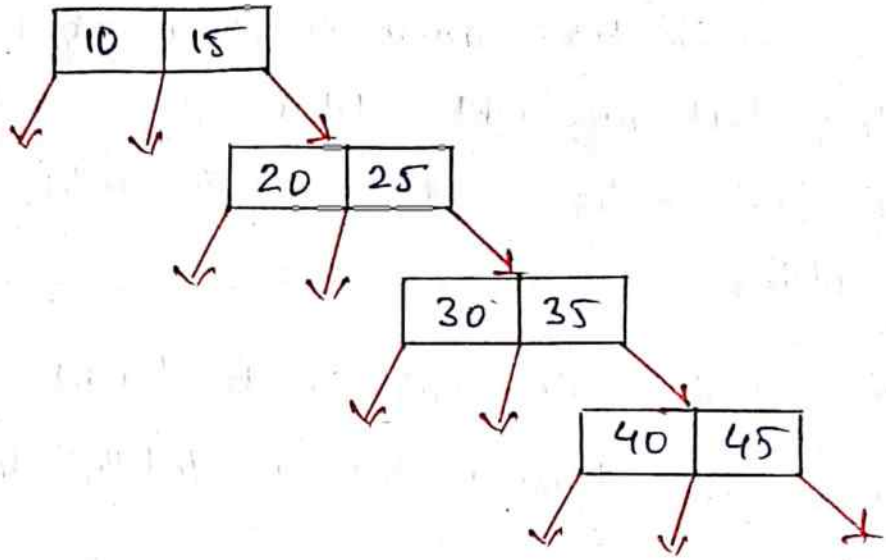
The problem with m-way search tree is that the tree is not balanced. ie leaf nodes are on different levels.

Example:→ Construct a 3-way search tree for the list of Keys in the order shown below:

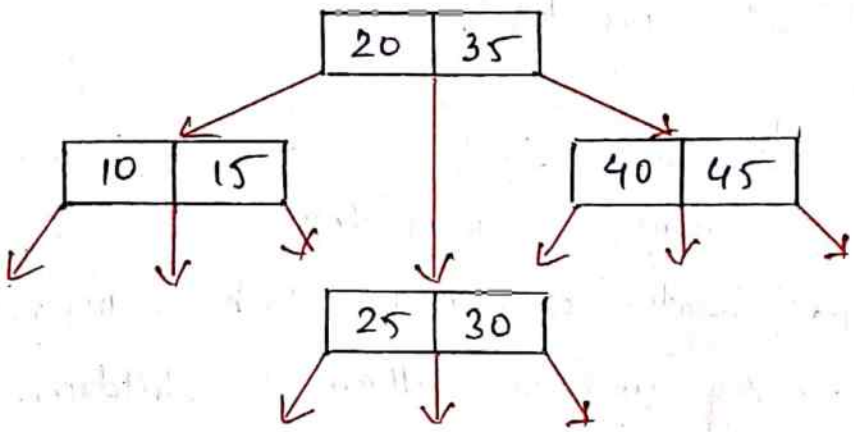
List A: 10, 15, 20, 25, 30, 35, 40, 45

List B: 20, 35, 40, 10, 15, 25, 30, 45

Soln 3-way search tree for list A.



3-way search tree for list B



So to balance m-way search tree, we have two data structures (1) B-tree

(2) B+ tree

NOTE (1) B tree and B+ tree are also called Balanced m-way Search tree.

(2) It is used to reduce the height of m-way search tree.

(3) These are used for file indexing for storing, retrieval of data in secondary memory.

B-tree

In m-way search tree, many nodes might have only left subtree but no right subtree.
Similarly some nodes might have only right subtree but no left subtree.

∴ Insertion of keys also increases the height of tree.
As we know access time in tree is totally dependent on level of tree.

∴ To minimize the access time, we need to take all the leaf nodes at same level.

Important Points

- B-tree is a balanced m-way tree.
- It is a Generalization of BST in which a node can have more than one key and more than 2 children.
- B-tree maintains sorted data.
- In B-tree, all leaf nodes are at same level.
- B-tree of order m has following properties:
  (i) Every node has maximum m children.
  (ii) Minimum children: Leaf = 0
  Root = 2
  Internal nodes = $\lceil \frac{m}{2} \rceil$

(iii) Every node has maximum $m-1$ keys

(iv) Minimum keys: Root node = 1

All other nodes = $\lceil \frac{m}{2} \rceil - 1$

Insertion in B Tree

Q1 Create a B tree of order 3 by inserting values from 1 to 10.

Soln Order = 3

∴ Every node has maximum 3 children

Every node has maximum 2 keys

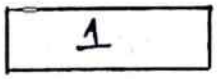
Minimum children = $\lceil \frac{3}{2} \rceil = \lceil 1.5 \rceil = 2$

Minimum keys = $\lceil \frac{3}{2} \rceil - 1$

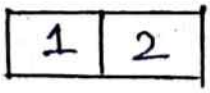
$= 2 - 1 = 1$

Insert values from 1 to 10 means insert values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

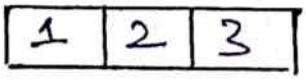
Insert 1



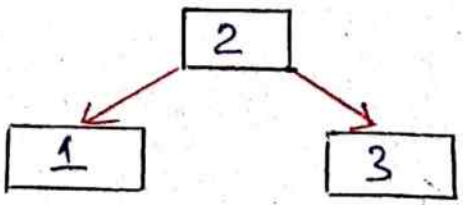
Insert 2



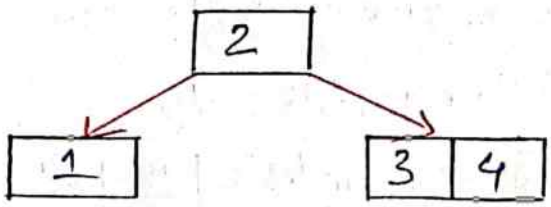
Insert 3



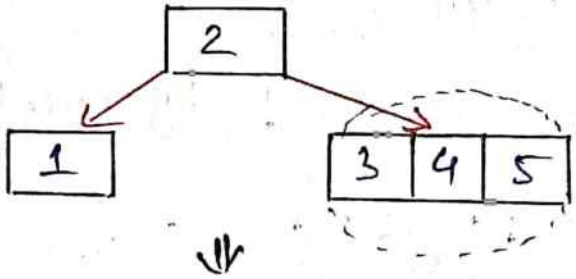
Node is Full ∴ Split



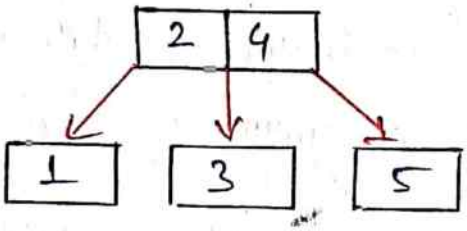
Insert 4



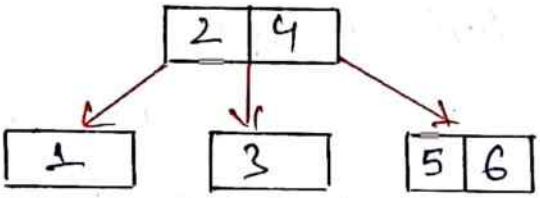
Insert 5



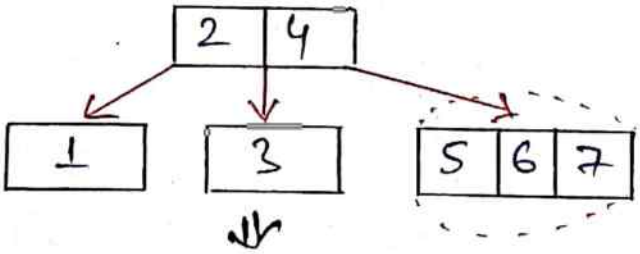
Node is Full
∴ Split



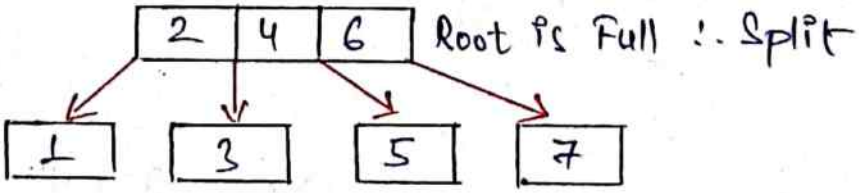
Insert 6



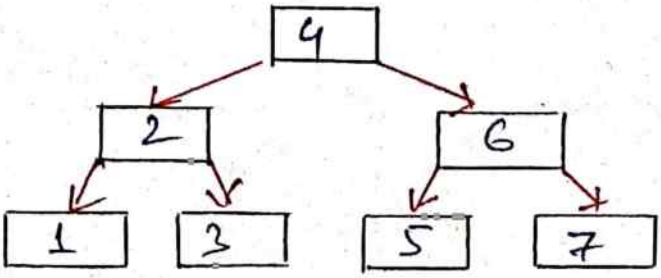
Insert 7



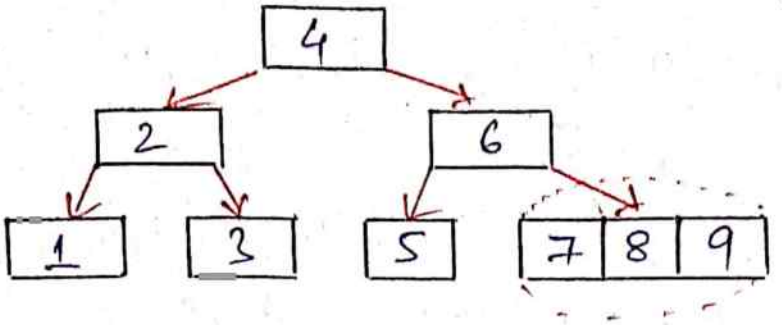
Node is Full
∴ Split



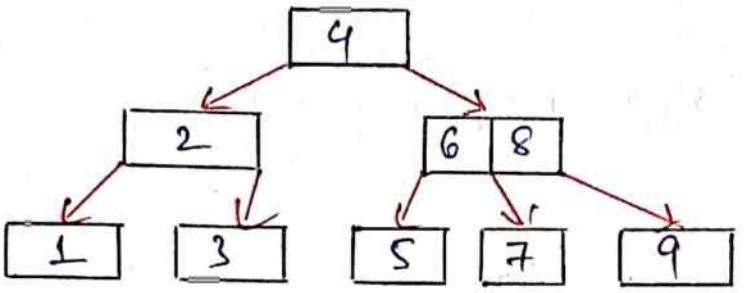
Root is Full ∴ Split



Insert 8, 9



Node is Full ∴ Split



Insert 10

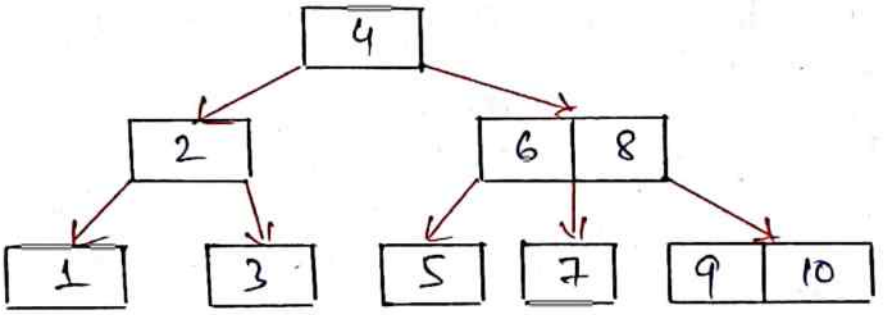


fig. B-tree of order 3

Q2 Construct a B-tree of order 4 with following set of data:

5, 3, 21, 9, 1, 13, 2, 7, 10, 12, 4, 8

Soln

Order = 4

∴ Every node has maximum 4 children.

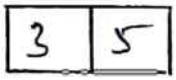
Every node has maximum 3 Keys

Minimum children $= \lceil \frac{4}{2} \rceil = 2$

Minimum Keys $= \lceil \frac{4}{2} \rceil - 1 = 2 - 1 = 1$

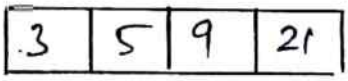
Insert 5

Insert 3



Insert 21

Insert 9

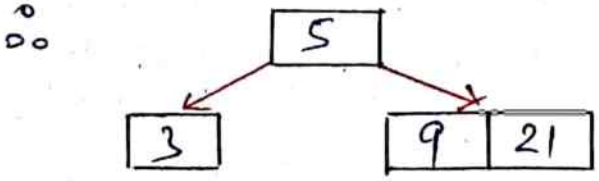


Node is full ∴ Split

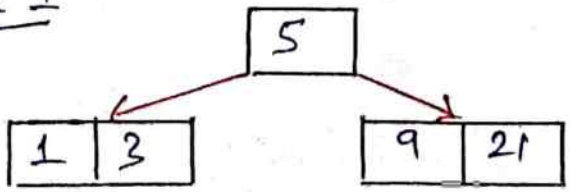
If 5 is Root ⇒ Left-Biased B-tree

If 9 is Root ⇒ Right-Biased B-tree

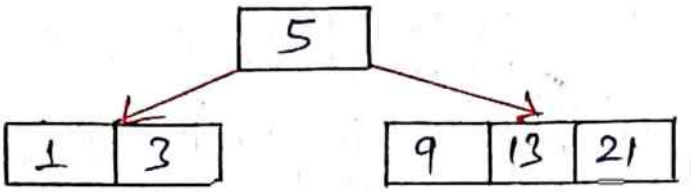
Lets follow Left Biasing.



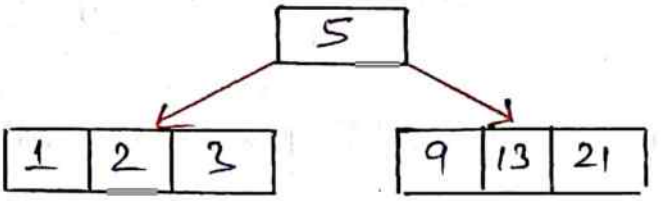
Insert 1



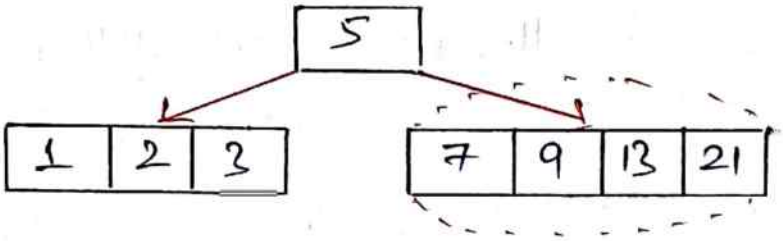
Insert 13



Insert 2

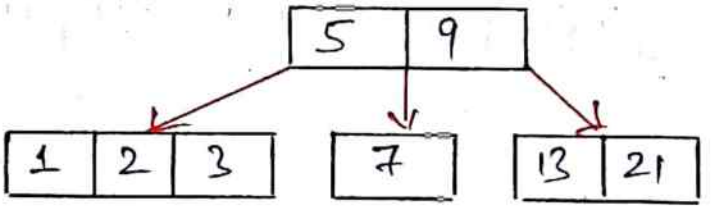


Insert 7

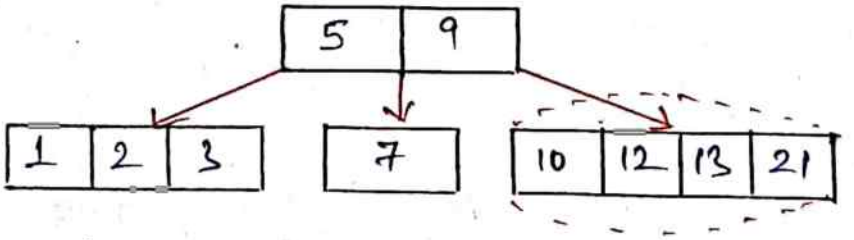


Node is full
∴ Split

⇓ Follow Left Biasing

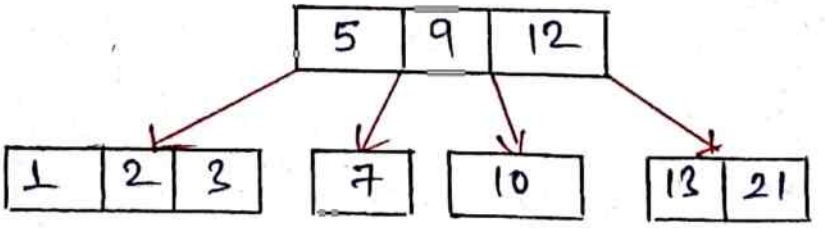


Insert 10, 12

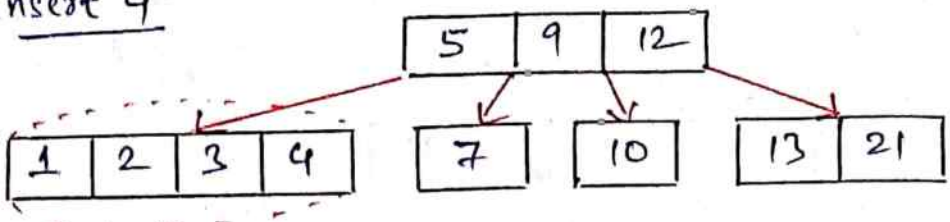


Node is full
∴ Split

⇓ Follow Left Biasing

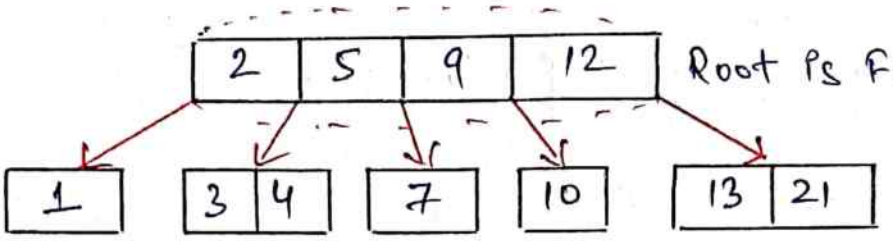


Insert 4



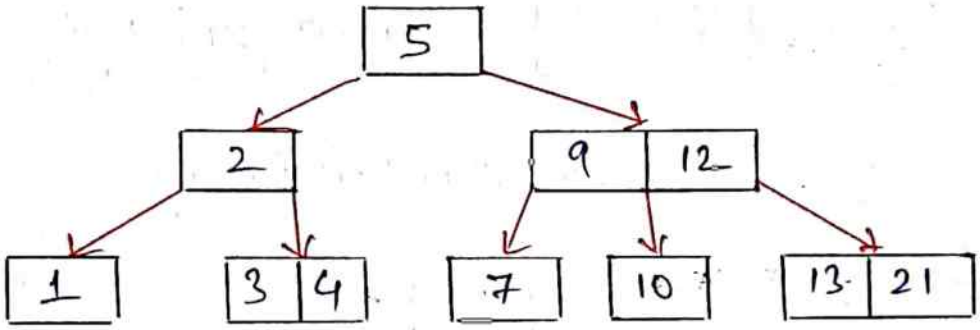
Node is Full
∴ Split

⇓ Follow Left Biasing



Root is Full. ∴ Split

⇓ Follow Left Biasing



Insert 8

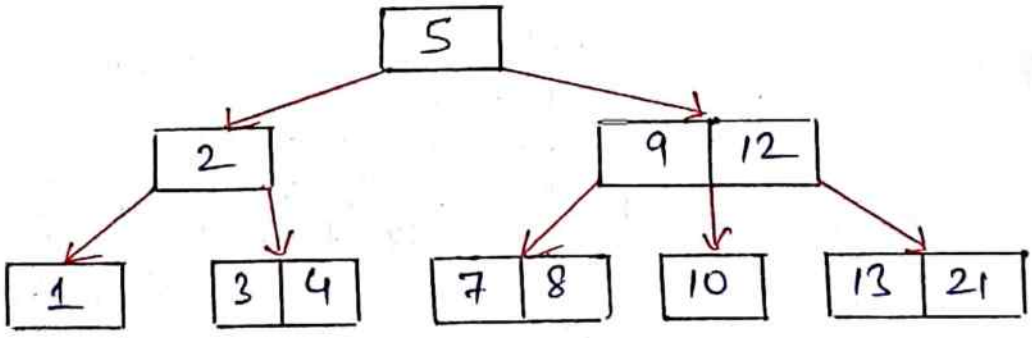


fig B-tree of order 4

Q3 Create a B-tree of order 5 for following keys:

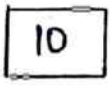
10, 20, 50, 60, 40, 80, 100, 70, 130, 90, 30, 120, 140, 25, 35, 160, 180

Soln Order = 5

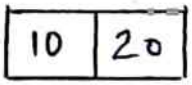
∴ Every node has maximum 5 children.

Every node has maximum 4 keys

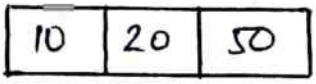
Insert 10



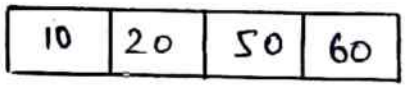
Insert 20



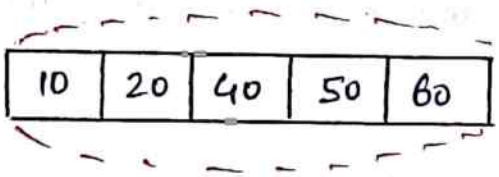
Insert 50



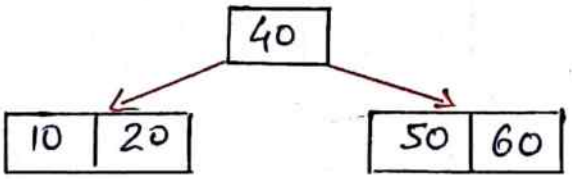
Insert 60



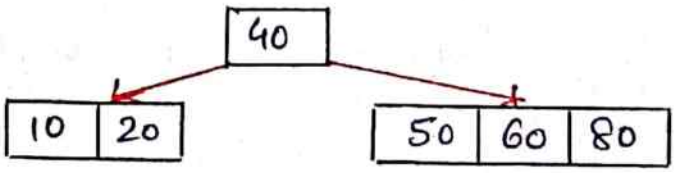
Insert 40



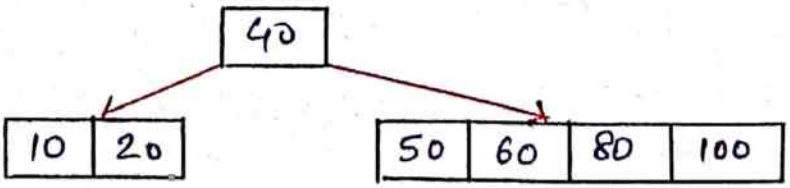
Overflow ∴ Split



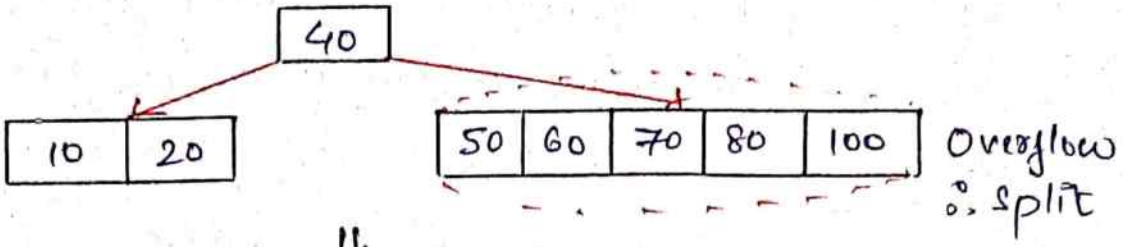
Insert 80

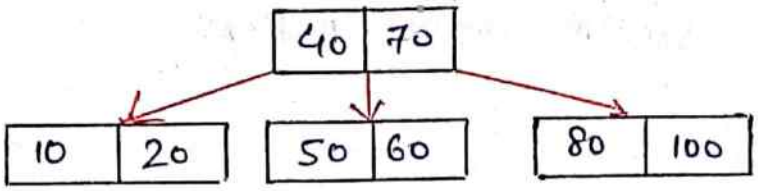


Insert 100

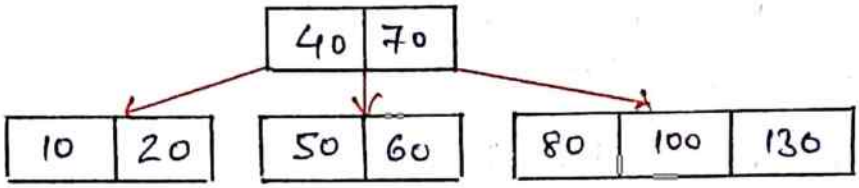


Insert 70

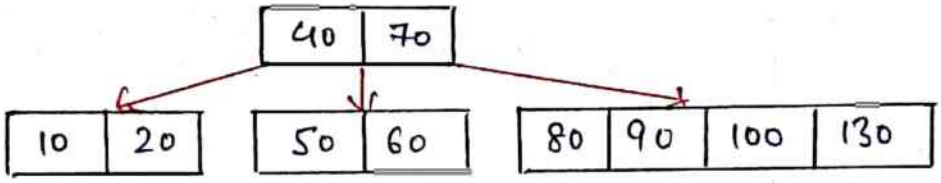




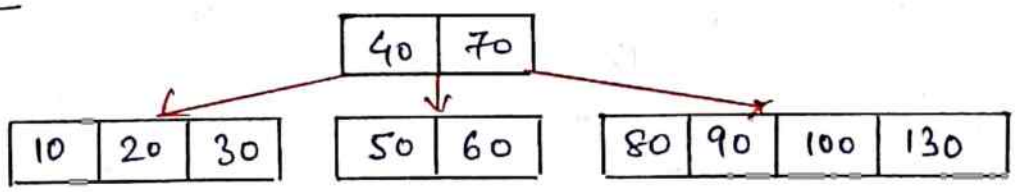
Insert 130



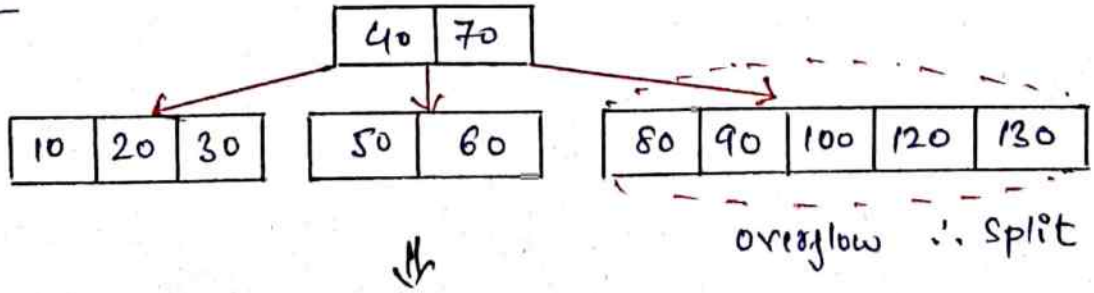
Insert 90

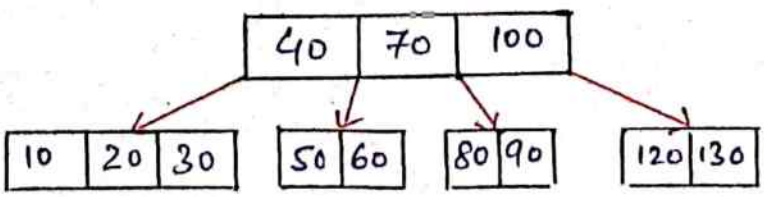


Insert 30

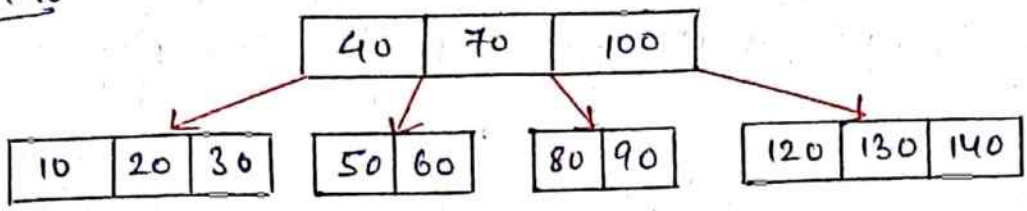


Insert 120

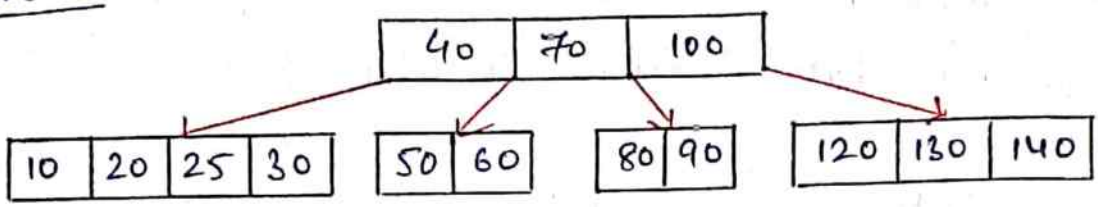




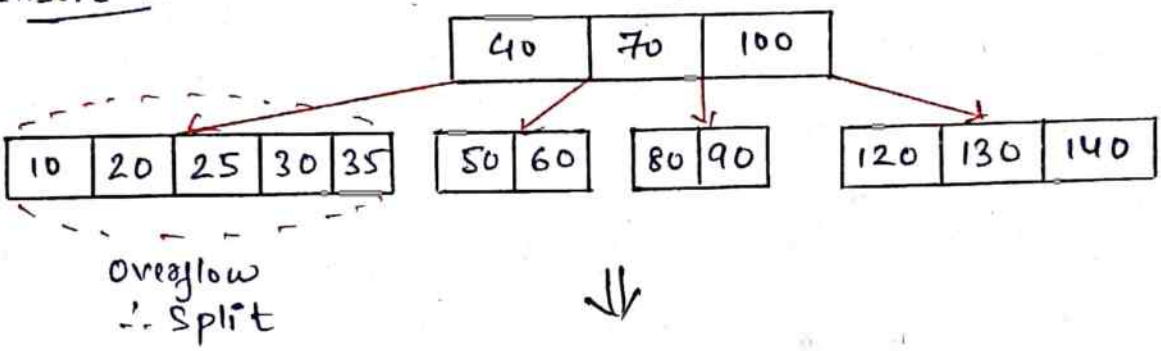
Insert 140

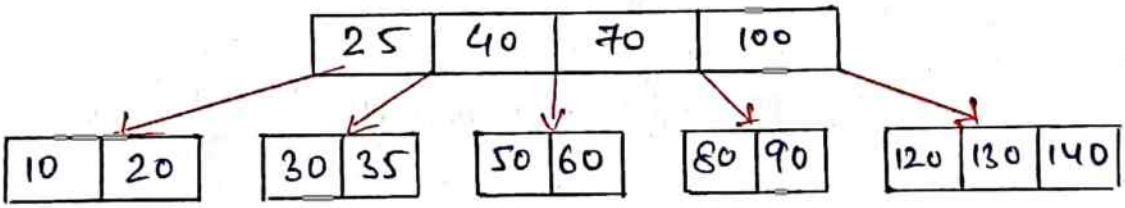


Insert 25

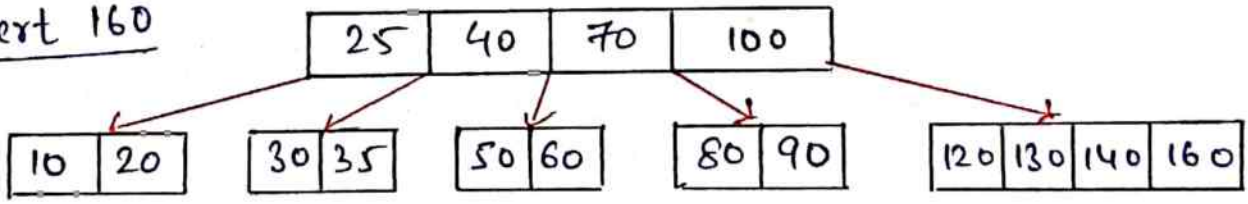


Insert 35





Insert 160



Insert 180

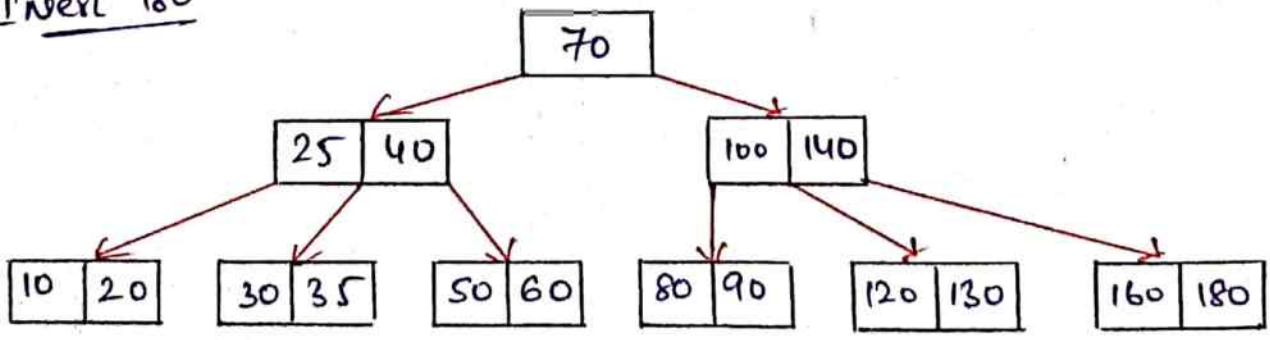


fig. B-tree of order 5

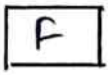
Q4 Show the result of inserting the keys F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B in order into an empty B-tree of order 5.

Soln Order = 5

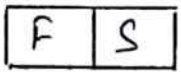
∴ Every node has maximum 5 children.

Every node has maximum 4 keys.

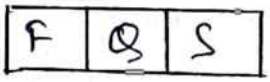
Insert F



Insert S

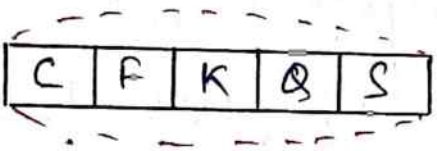


Insert Q

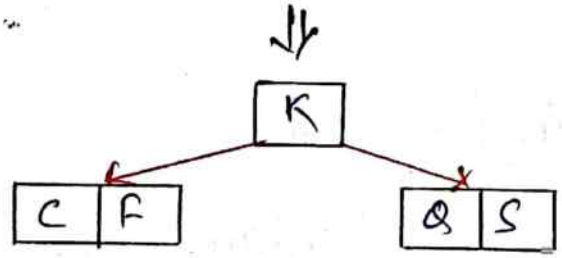


Insert K

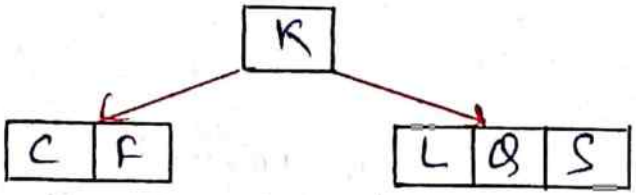
Insert C



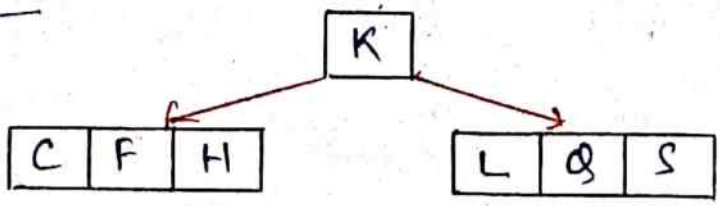
Overflow ∴ Split



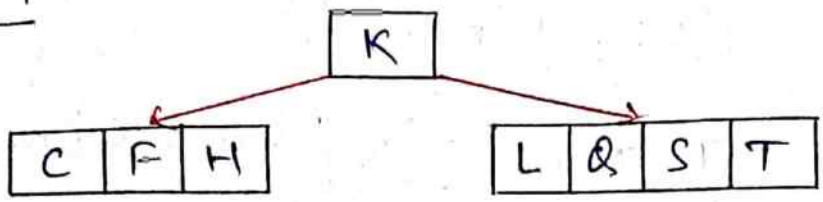
Insert L



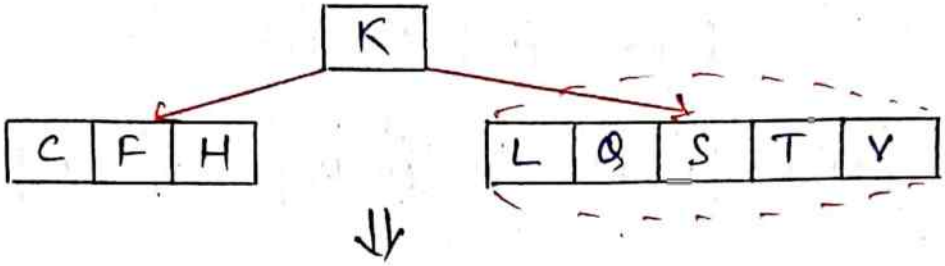
Insert H



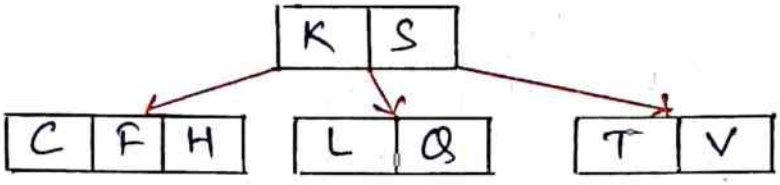
Insert T



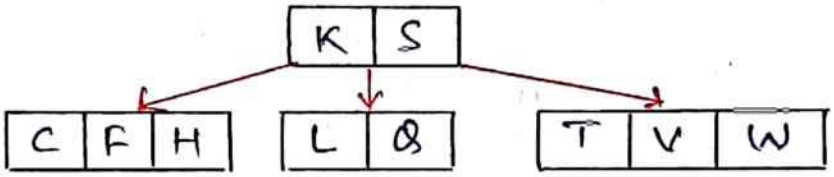
Insert V



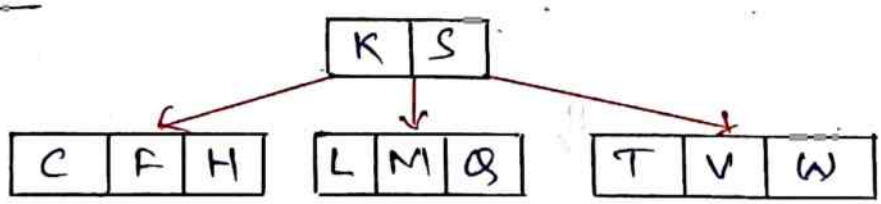
Overflow
∴ Split



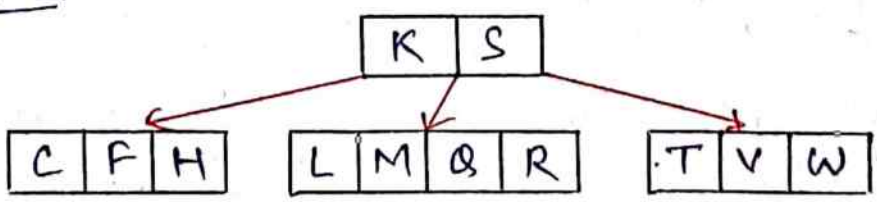
Insert W



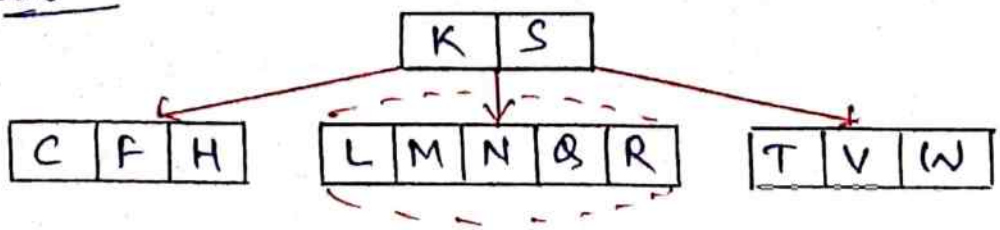
Insert M



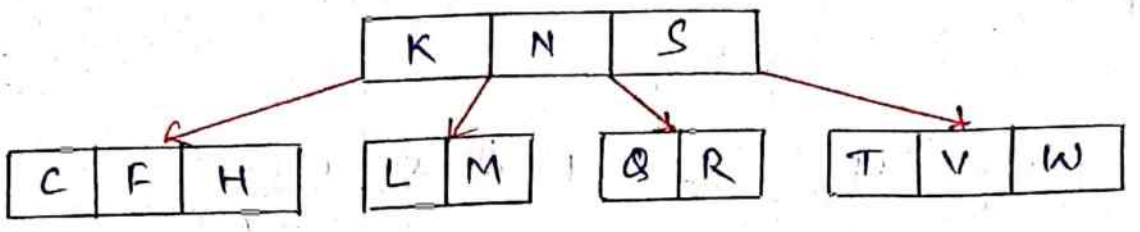
Insert R



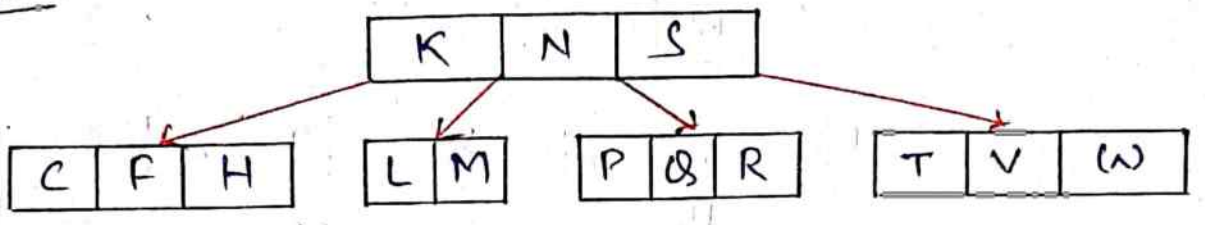
Insert N



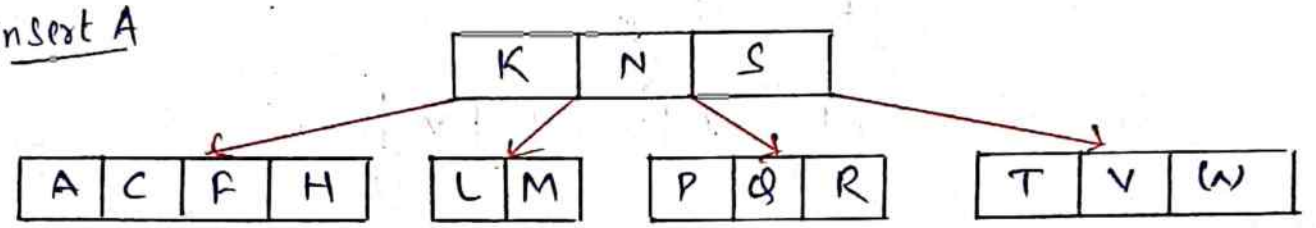
Overflow ∴ Split



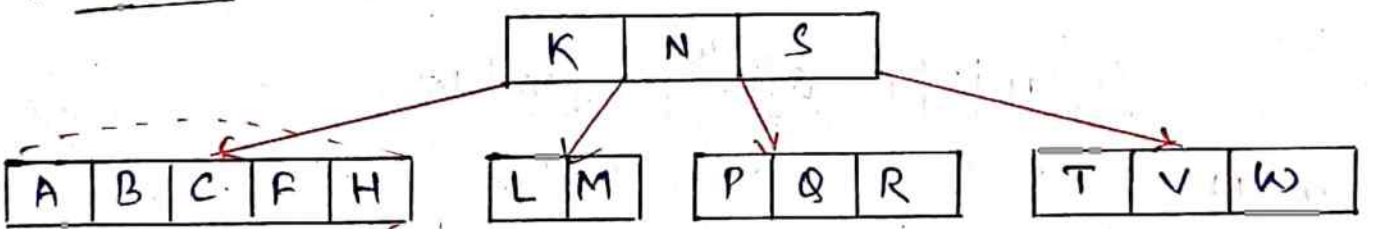
Insert P



Insert A



Insert B



overflow
∴ Split

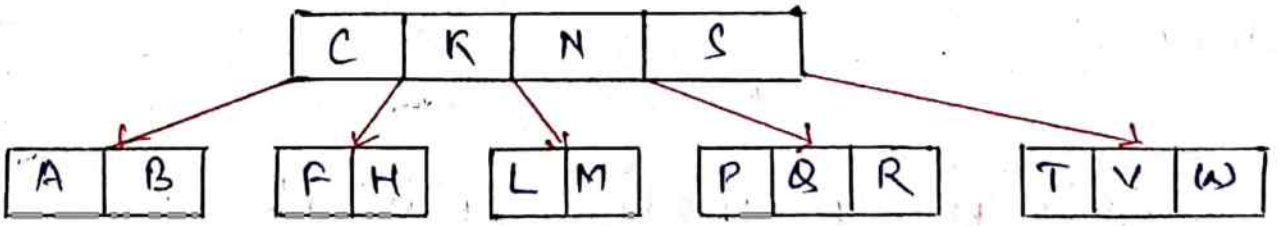


fig. B-tree of order 5

Q5 Construct a B-tree of order 5 with the following set of data:

D, H, Z, K, B, P, Q, E, A, S, W, T, C, L, N, Y, M

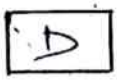
Soln

Order = 5

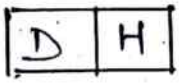
∴ Maximum children in each node = 5

Maximum Keys in each node = 4

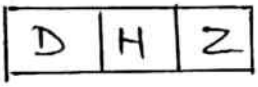
Insert D



Insert H

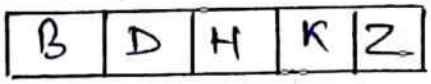


Insert Z

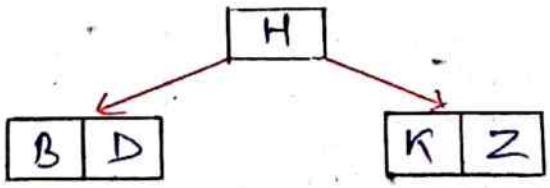


Insert K

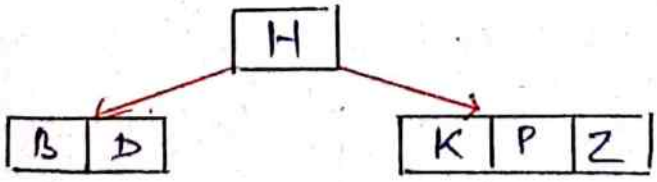
Insert B



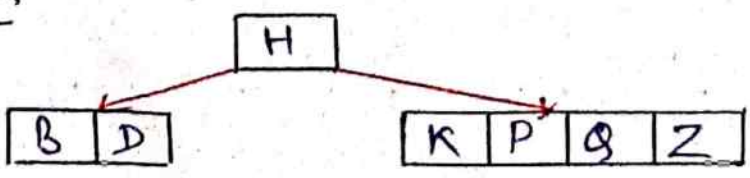
⇒ overflow ∴ split



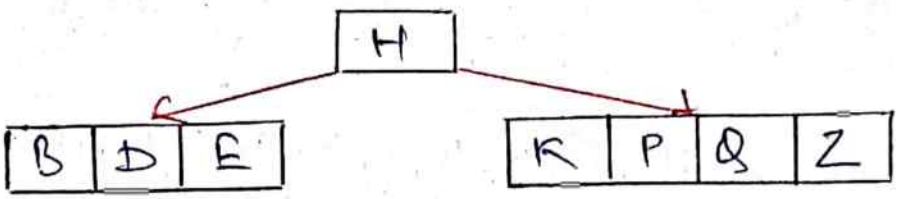
Insert P



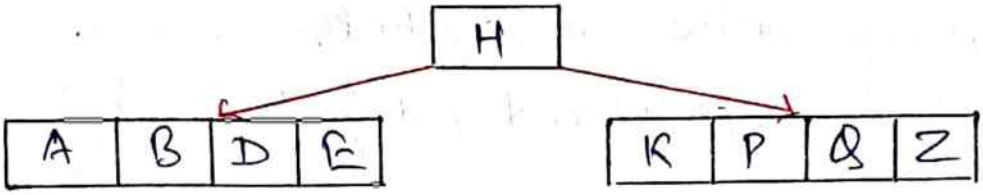
Insert Q



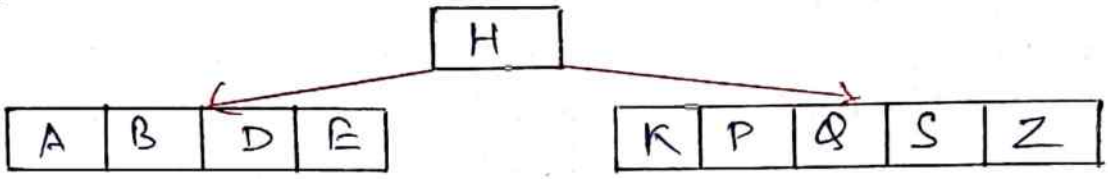
Insert E



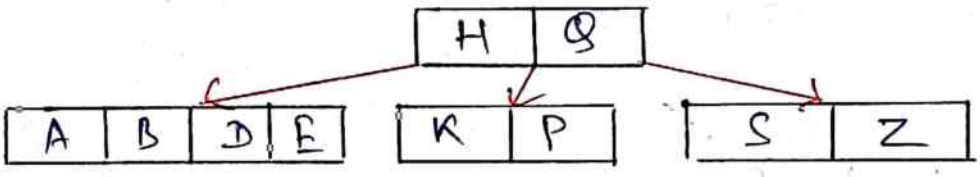
Insert A



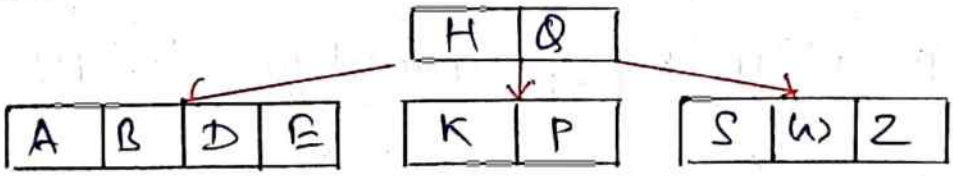
Insert S



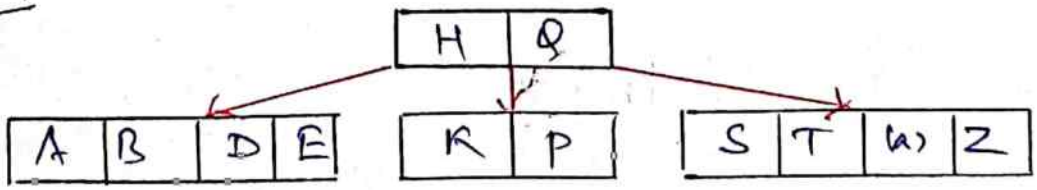
overflow ∴ Split



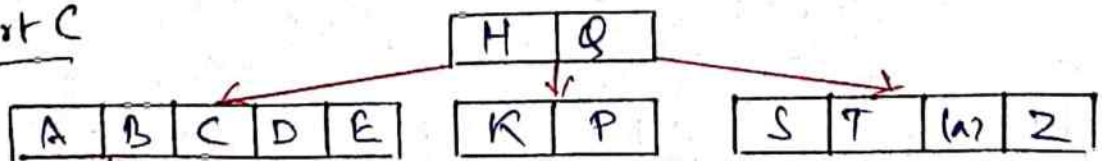
Insert W



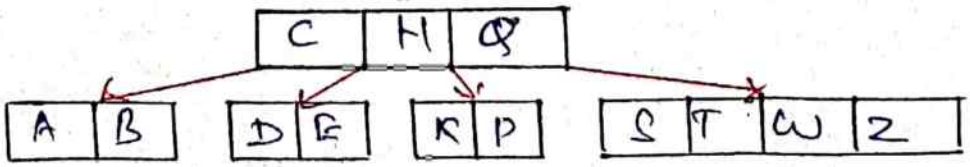
Insert T



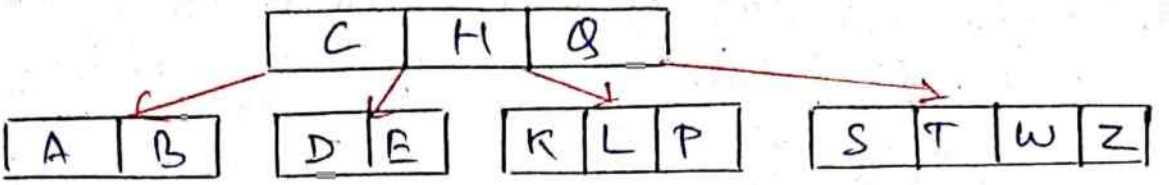
Insert C



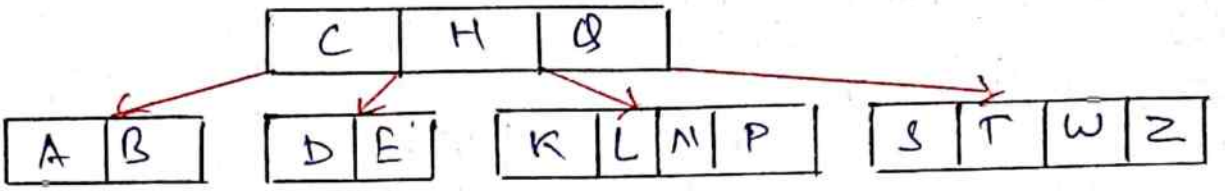
overflow. ∴ Split



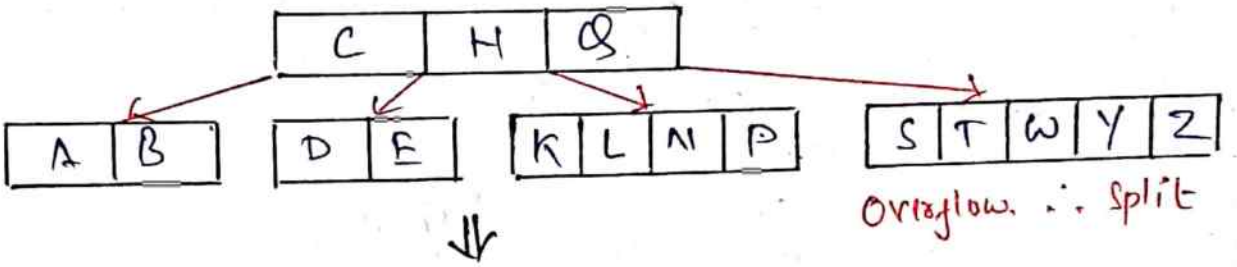
Insert L

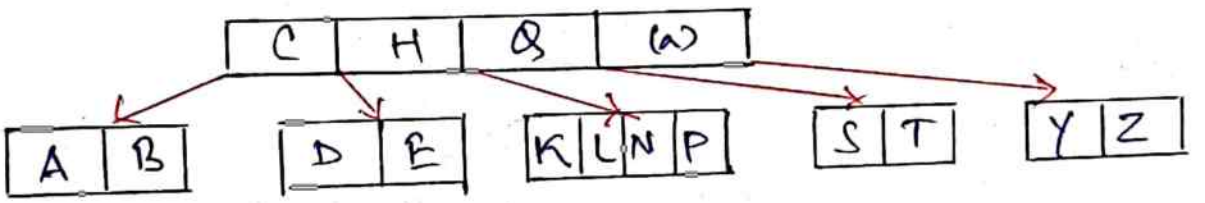


Insert N

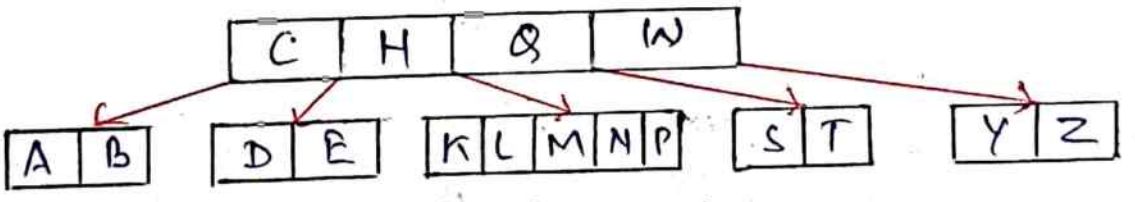


Insert Y





Insert M



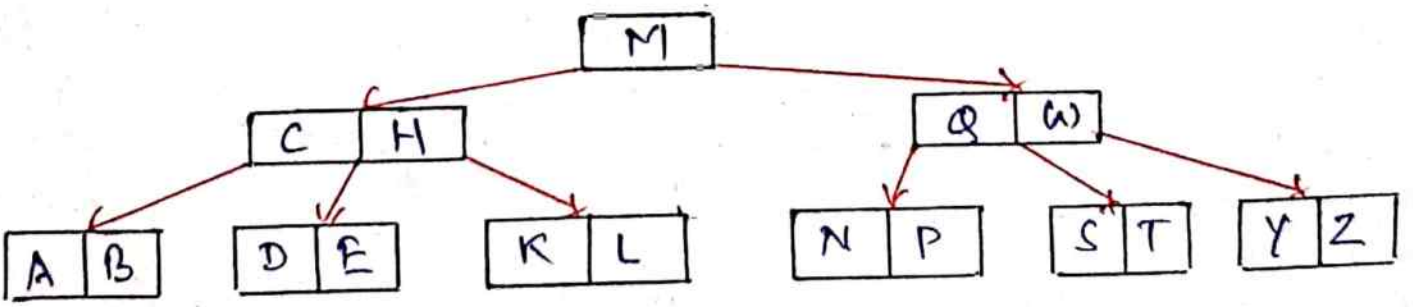


fig B-tree of order 5

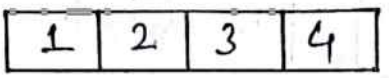
Q6 Create a B-tree of order 5 by inserting values from 1 to 20.

Soln Order = 5

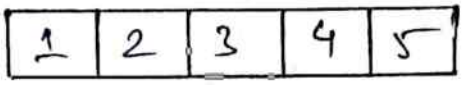
∴ Maximum children in each node = 5

Maximum Keys in each node = 4

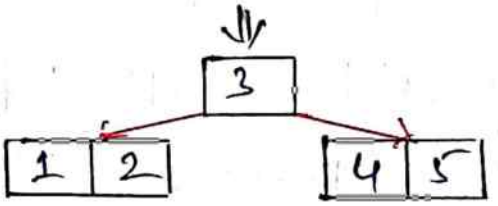
Insert 1,2,3,4



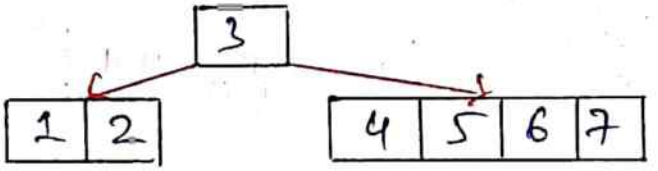
Insert 5



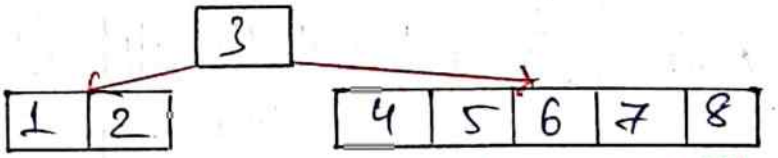
overflow. ∴ Split



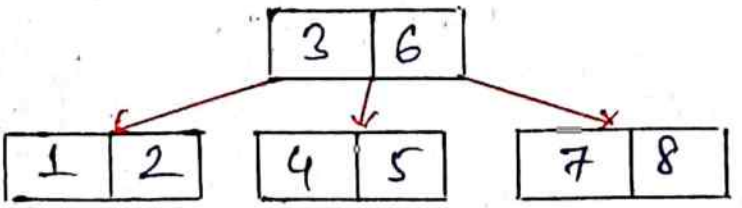
Insert 6,7



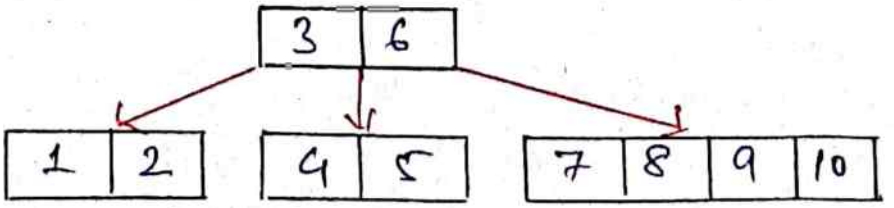
Insert 8



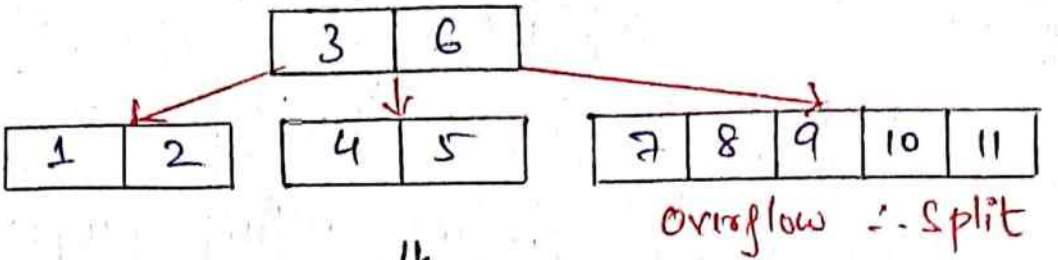
overflow ∴ split



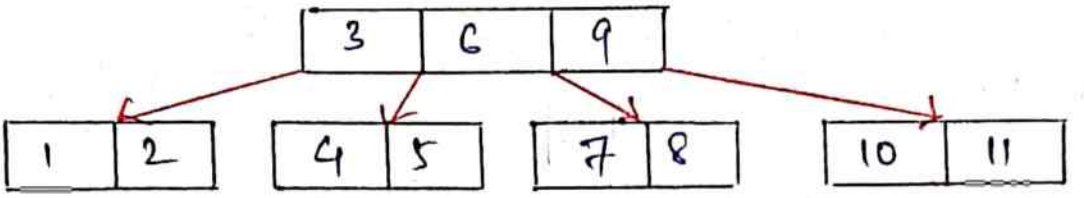
Insert 9,10



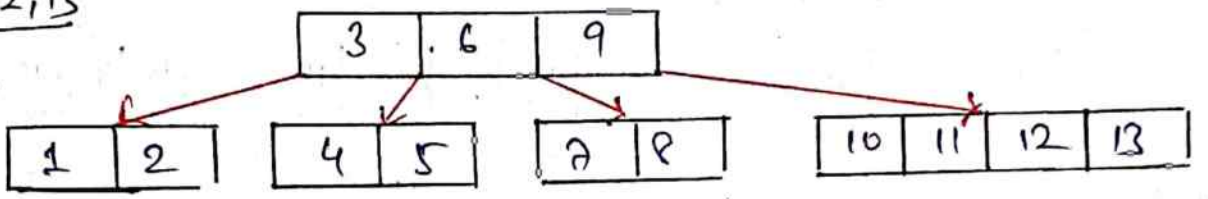
Insert 11



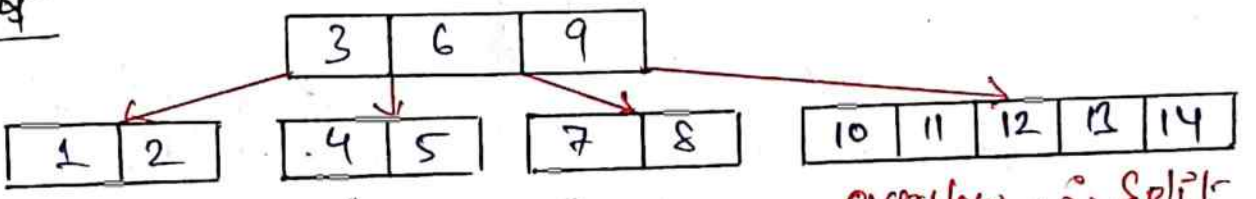
Overflow ∴ Split



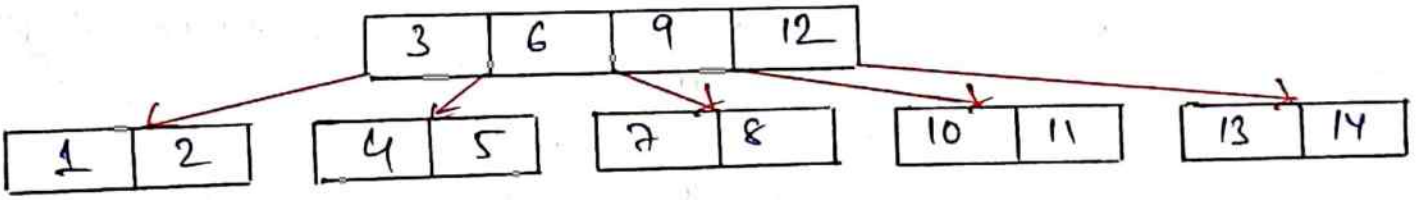
Insert 12, 13



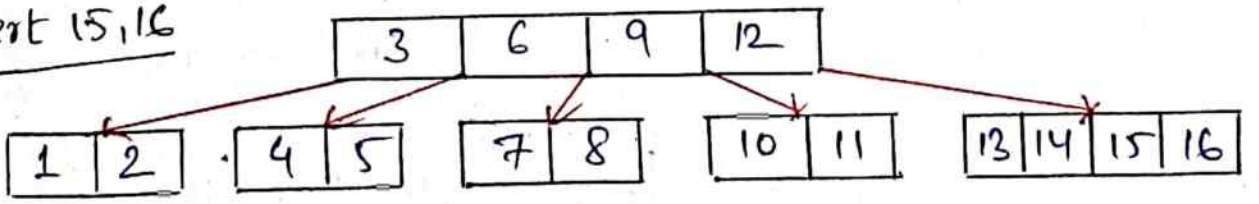
Insert 14



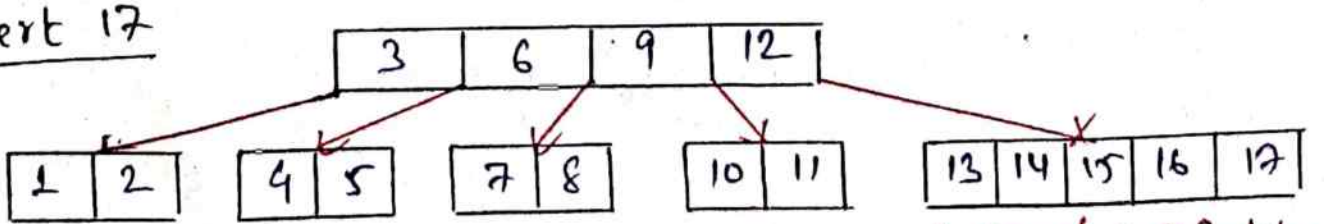
Overflow ∴ Split



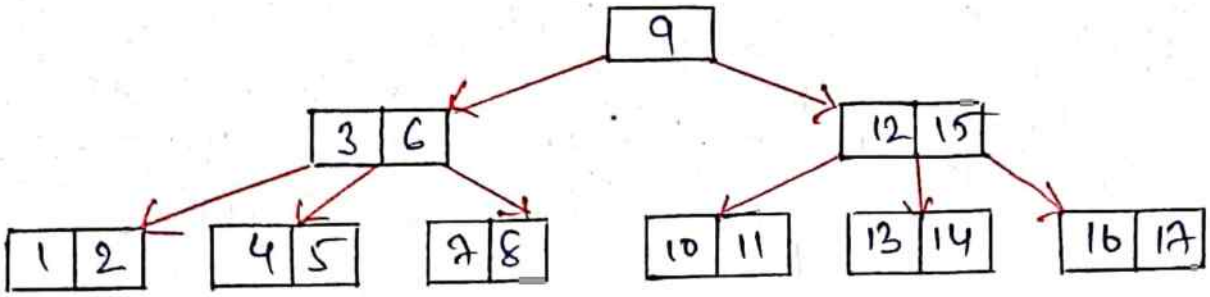
Insert 15, 16



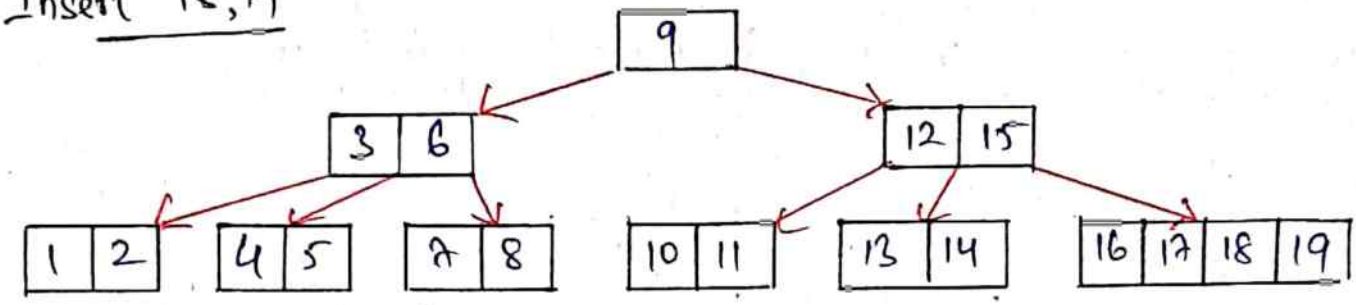
Insert 17



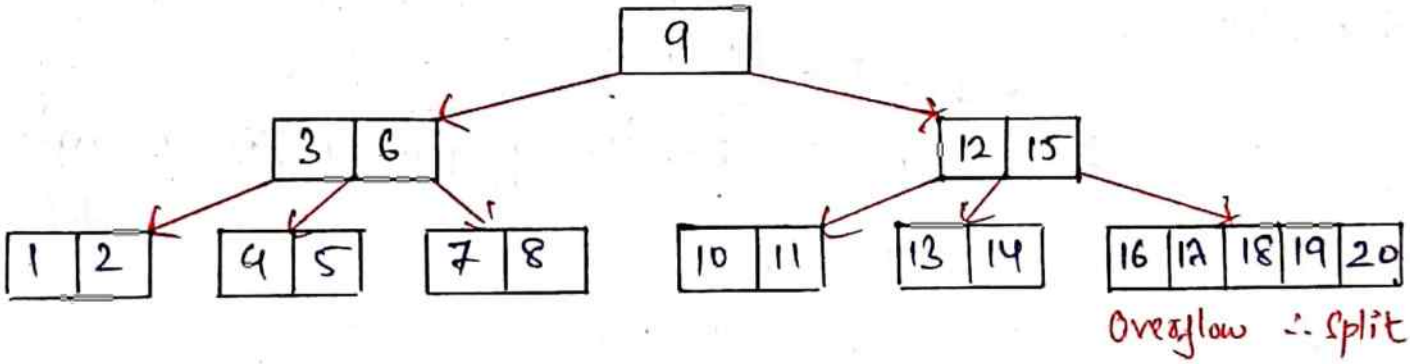
∴ Overflow ⇒ Split



Insert 18, 19



Insert 20



Overflow ∴ Split

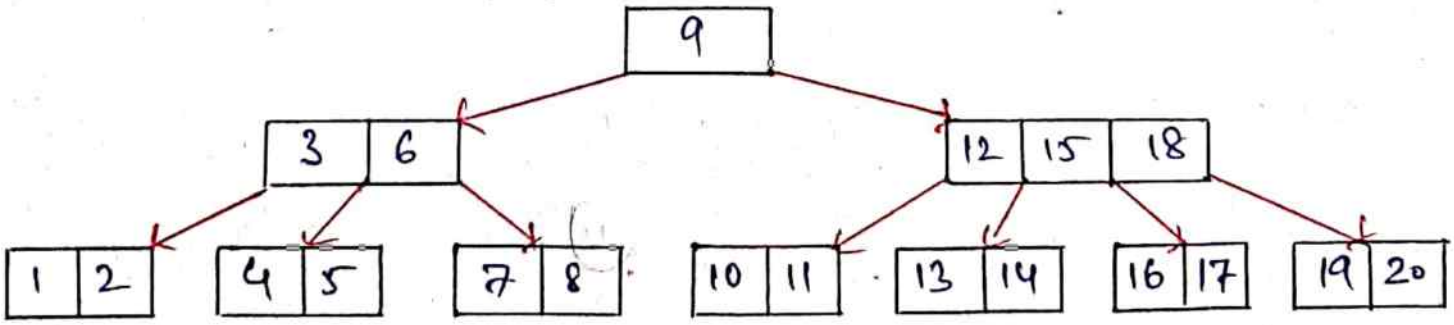


fig. B-tree of order 5

Q7 Consider the following sequence of nodes and show the growth of the B-tree of order 4.

Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V.

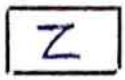
Soln

Order = 4 = m

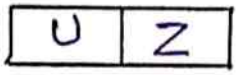
∴ Maximum children in each node = m = 4.

Maximum keys in each node = m - 1 = 3

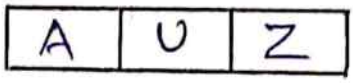
Insert Z



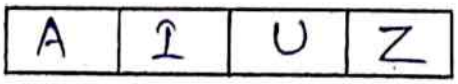
Insert U



Insert A

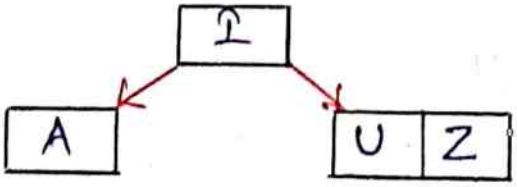


Insert I

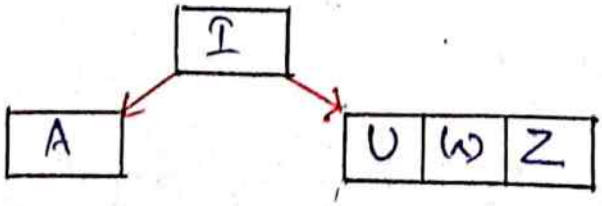


Overflow ∴ Split

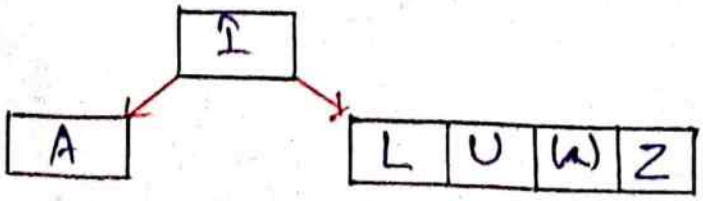
⇓ (Left Biasing)



Insert W

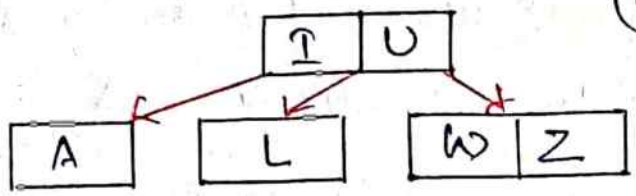


Insert L

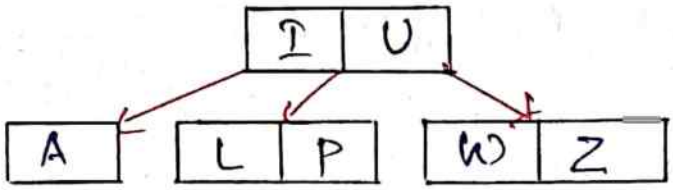


Overflow ∴ Split

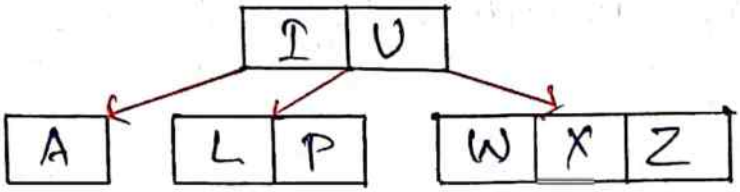
(Left-Biasing)



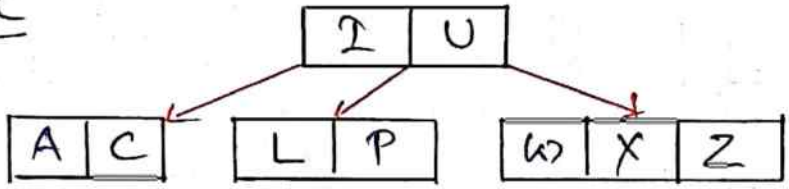
Insert P



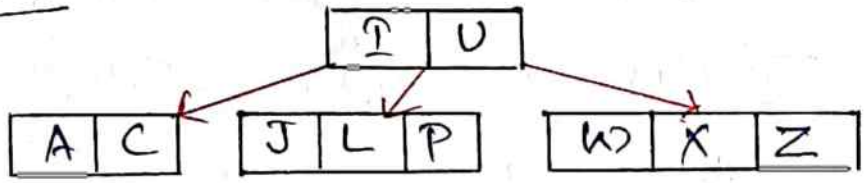
Insert X



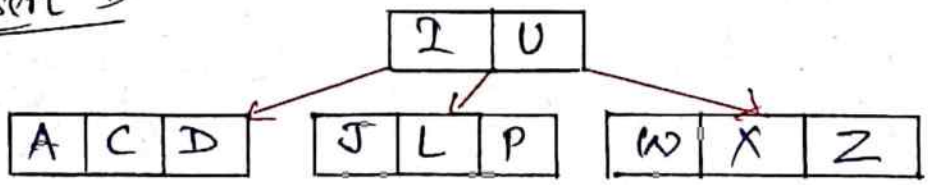
Insert C



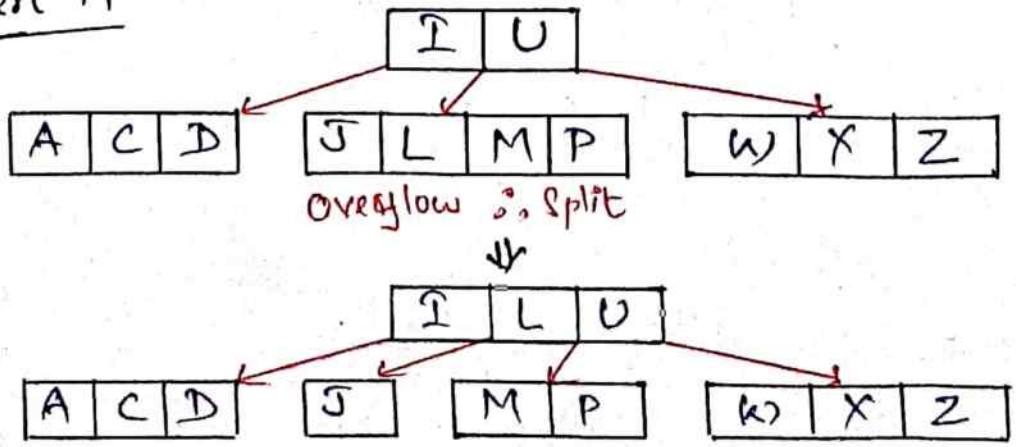
Insert J



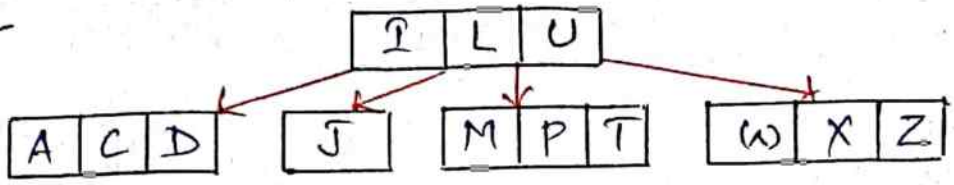
Insert D



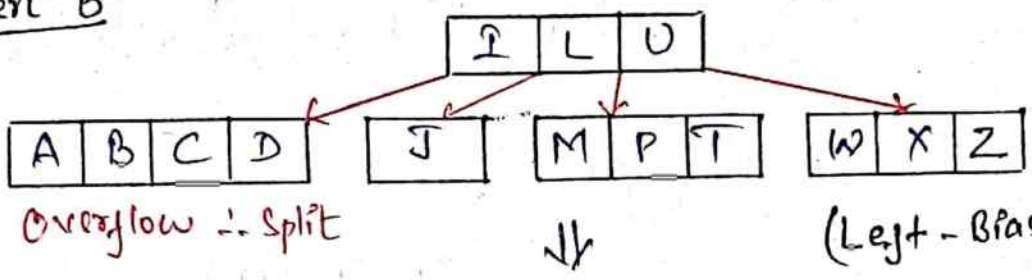
Insert M



Insert T

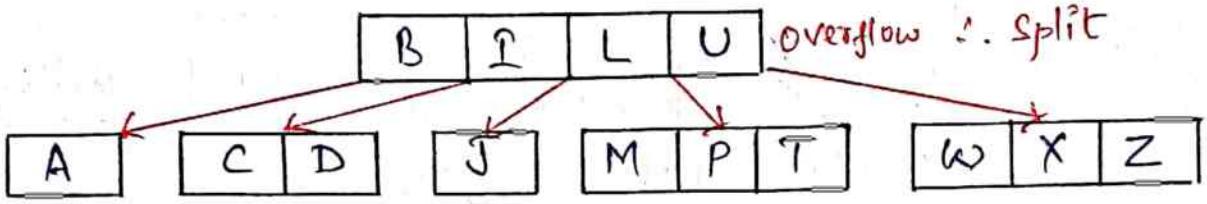


Insert B



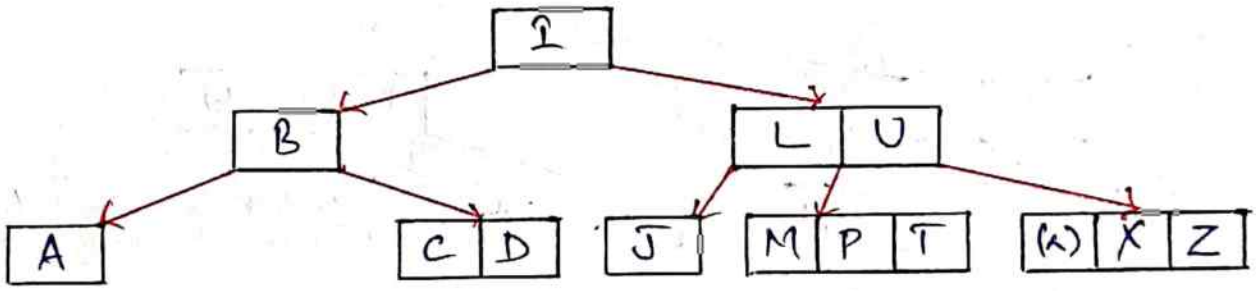
Overflow ∴ Split

(Left - Biasing)

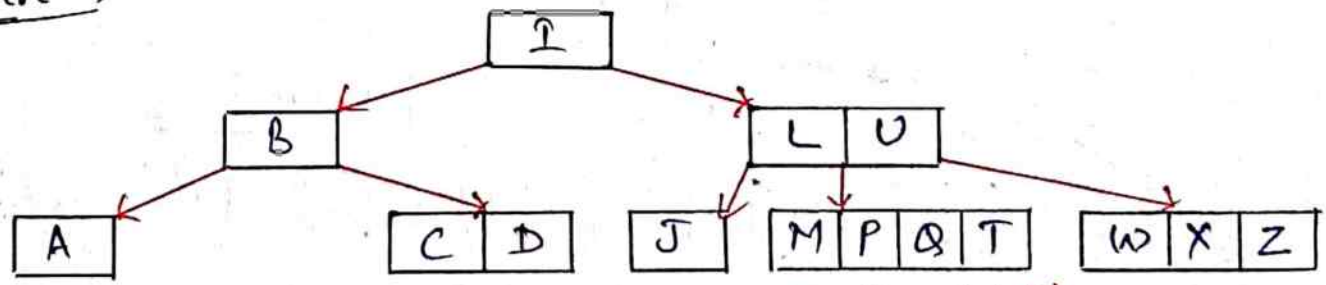


Overflow ∴ Split

(Left - Biasing)

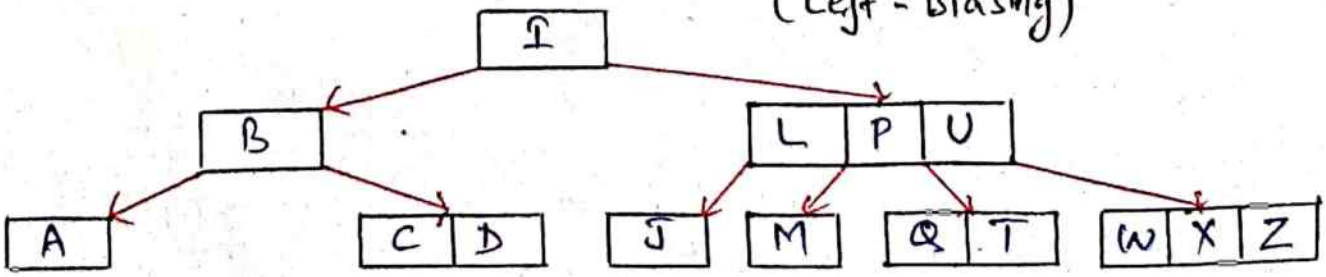


Insert Q

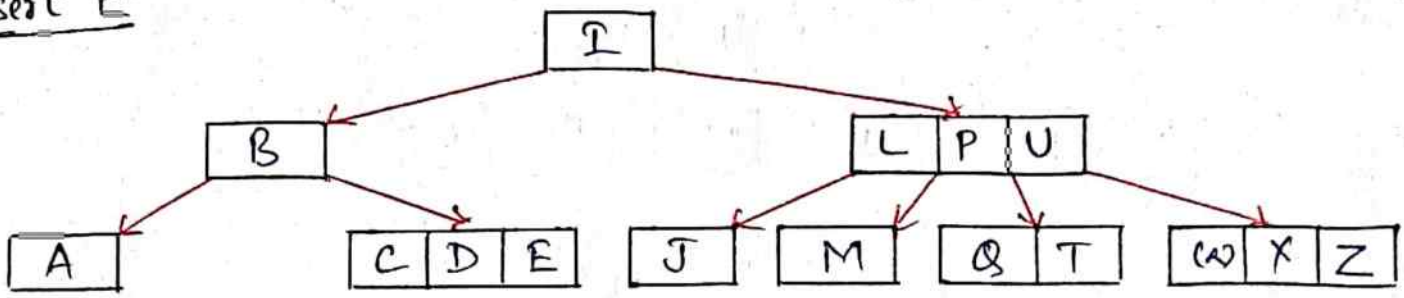


Overflow ∴ Split

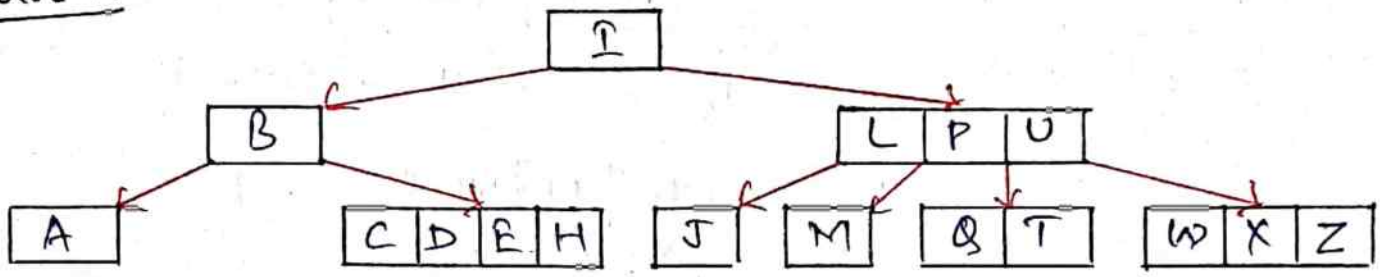
(Left - Biasing)



Insert E

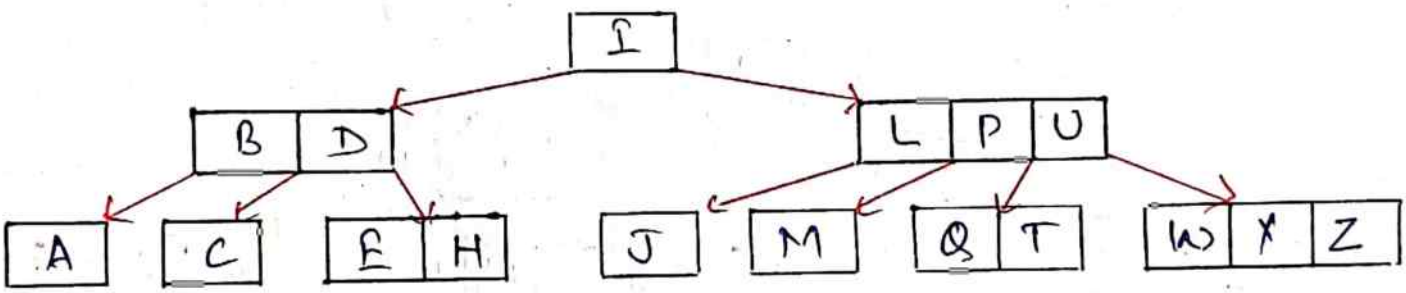


Insert H

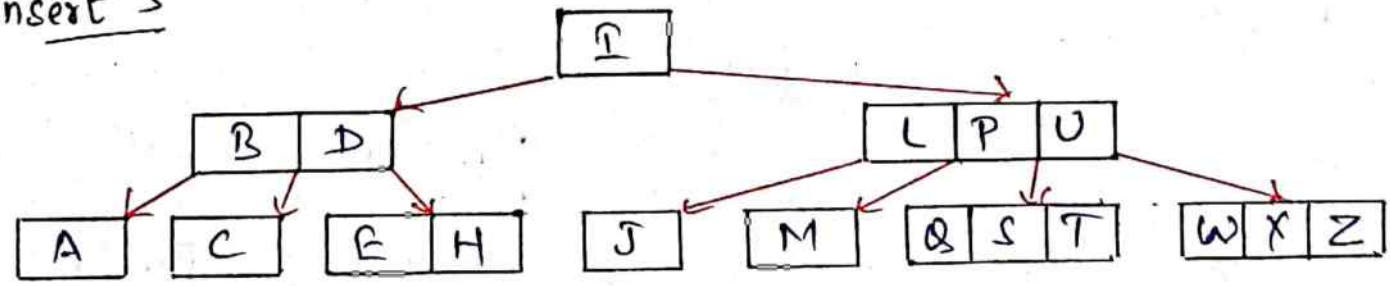


overflow ∴ split

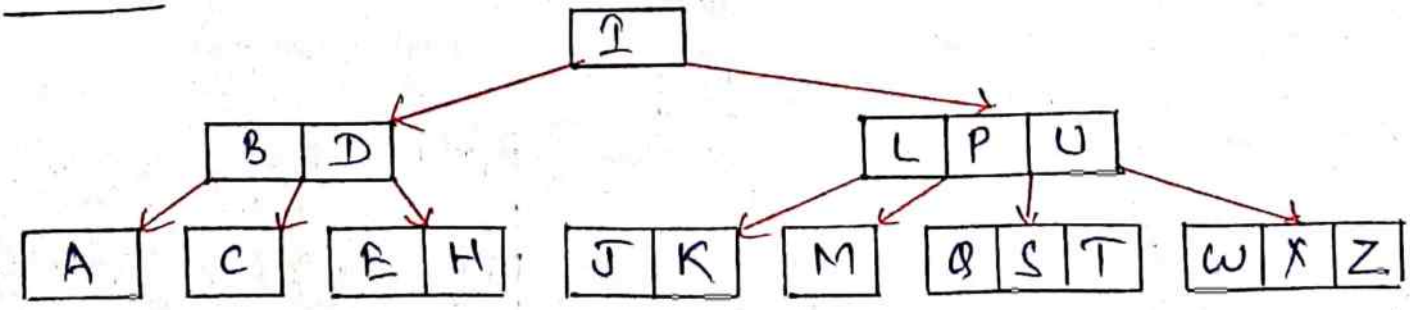
(Left-Biasing)



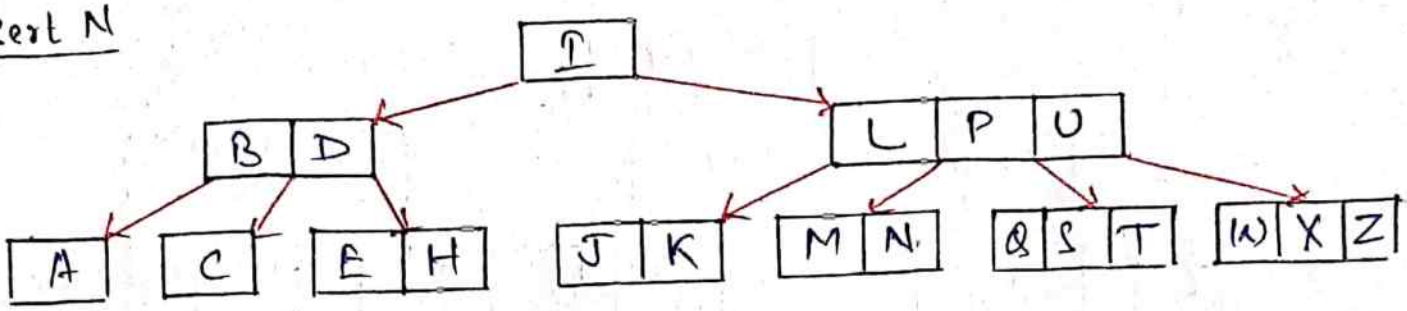
Insert S



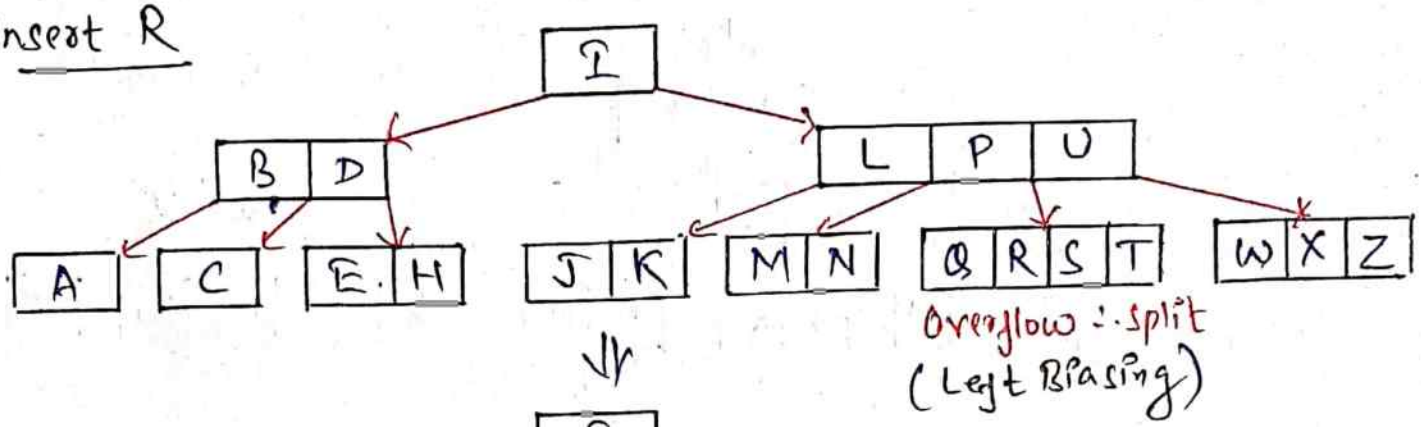
Insert K



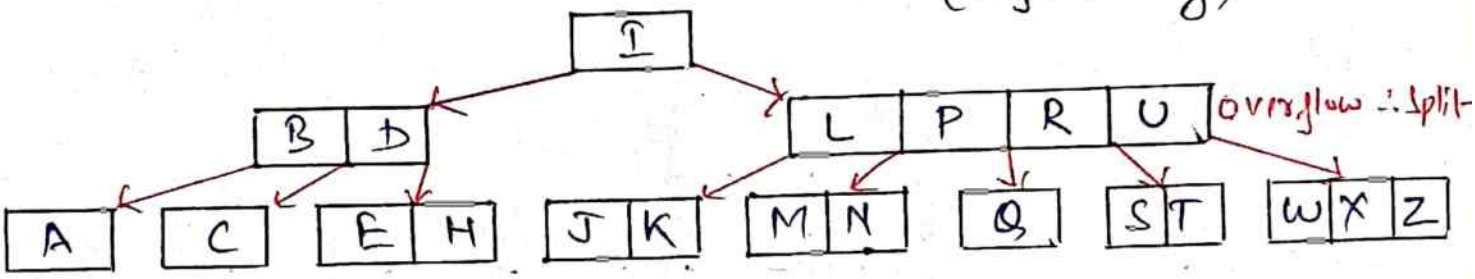
Insert N



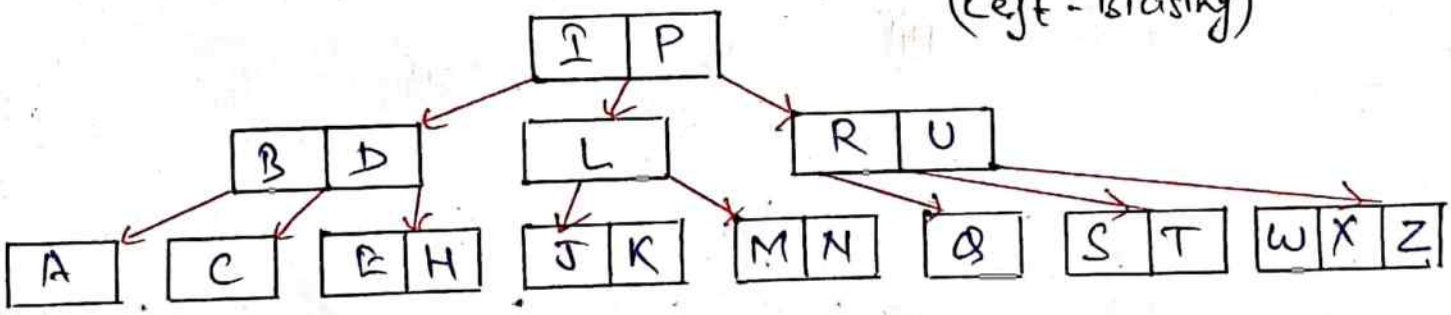
Insert R



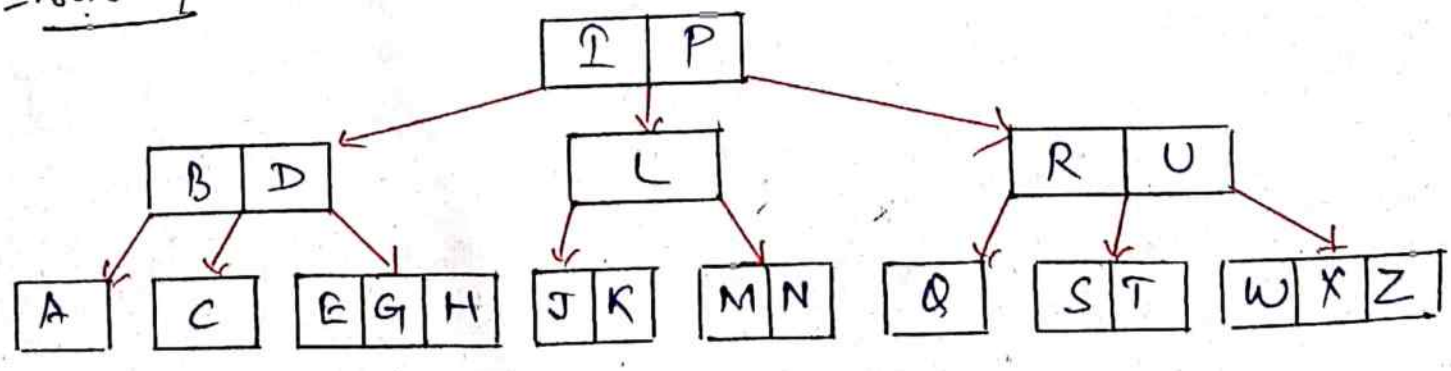
Overflow :. split
(Left Biasing)



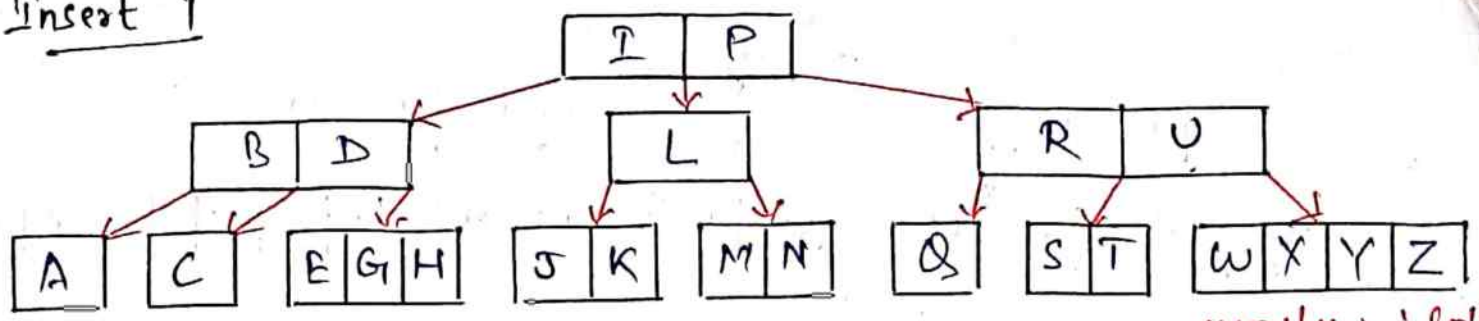
(Left - Biasing)



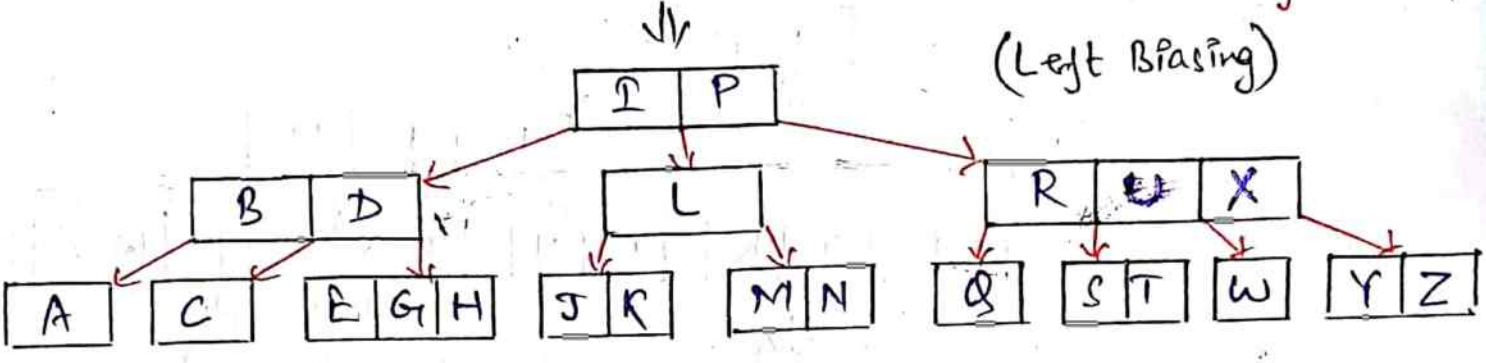
Insert G



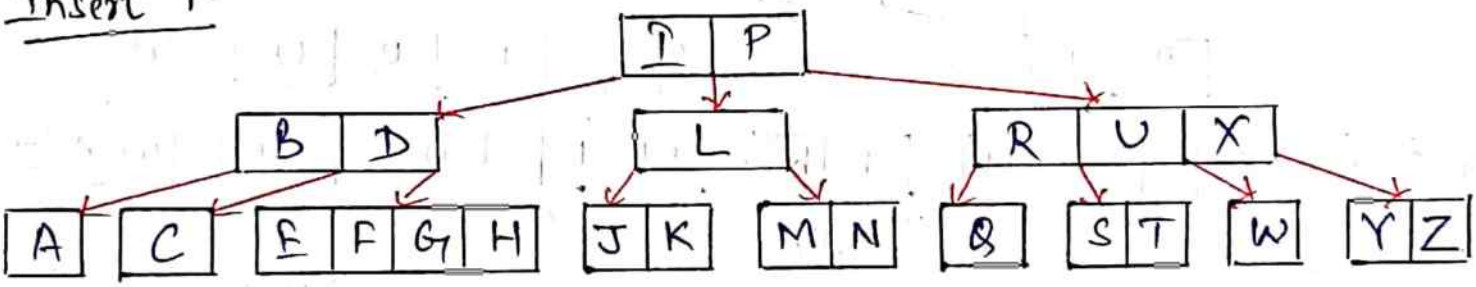
Insert Y



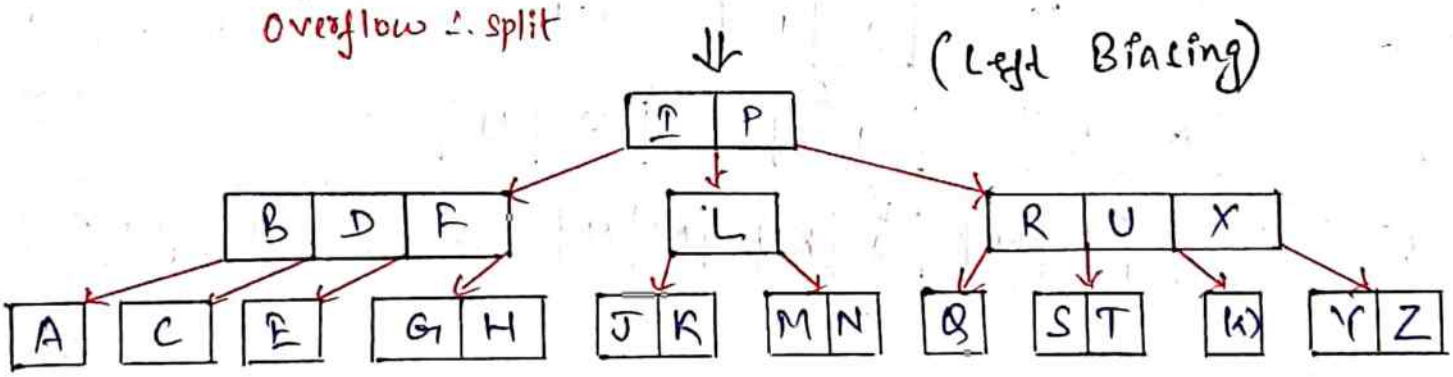
overflow ∴ Split



Insert F



Overflow ∴ split



Insert O, V

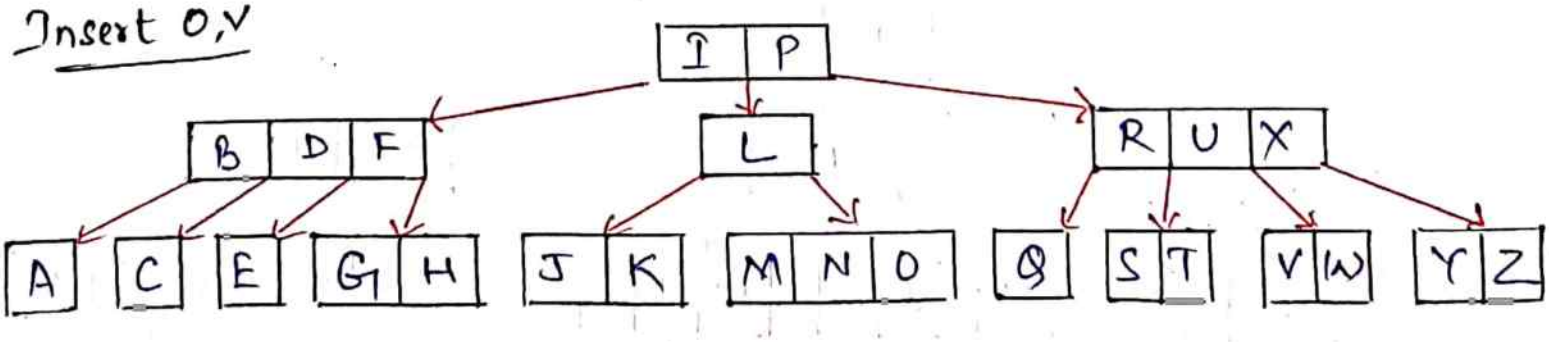


fig B-tree of order 4

B-tree Deletion

(1) If target Key is in leaf node.

(2) If target Key is in internal node.

Case(1) :=> If target Key is in leaf node.

- that leaf node contains more than minimum number of Keys.
  - So No problem
  - Solution is : simply delete that node
- that leaf node contains minimum number of Keys.
  - Problem : Underflow
  - Solution is Borrow Key from sibling if possible.
    - either → Borrow Key from immediate left sibling.
    - or → Borrow Key from immediate right sibling.
    - If Key borrow is not possible, then go for node combine.

Case(2) :=> If target Key is in internal node.

Solution is
- either → Replace by its inorder predecessor
- or → Replace by its inorder Successor
- Node Combine

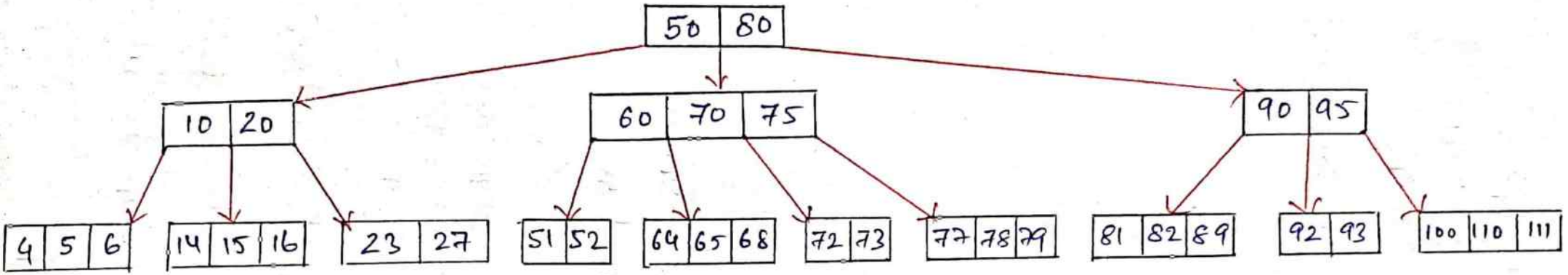


fig. B-tree of order 5

Order (m) = 5

∴ Maximum children in each node = m = 5

Maximum Keys in each node = m − 1 = 4

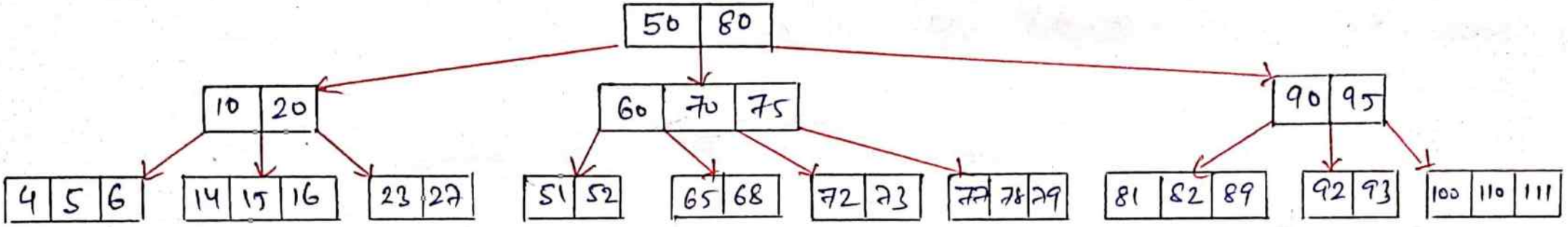
∴ Minimum children in each node = $\lceil \frac{m}{2} \rceil = \lceil \frac{5}{2} \rceil = \lceil 2.5 \rceil = 3$

Minimum Keys in each node = $\lceil \frac{m}{2} \rceil - 1 = 3 - 1 = 2$.

Delete 64, 23, 72, 65, 20, 70, 95, 77, 80, 100, 6, 27, 60, 16, 50.

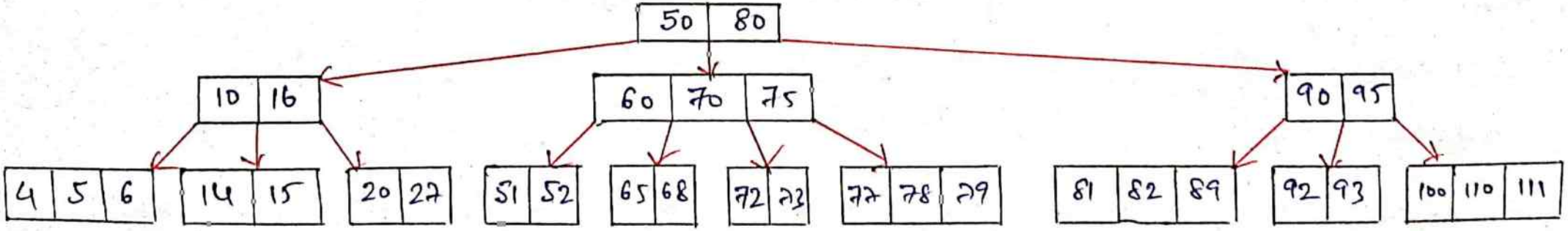
Delete 64

Here, Key 64 is in leaf node and that leaf node contains more than minimum number of keys. So simply delete 64.



Delete 23

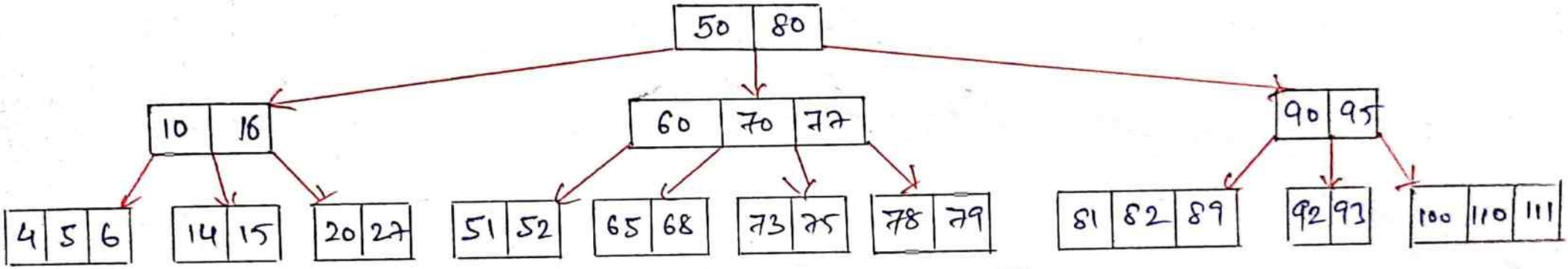
Here, Key 23 is in leaf node and that leaf node contains minimum number of keys. After deleting Key 23, the problem arises is Underflow. The solution is Borrow key from immediate left sibling.



After deleting Key 23, the Greatest Key 16 from left sibling moves up and parent Key 20 moves down.

Delete 72

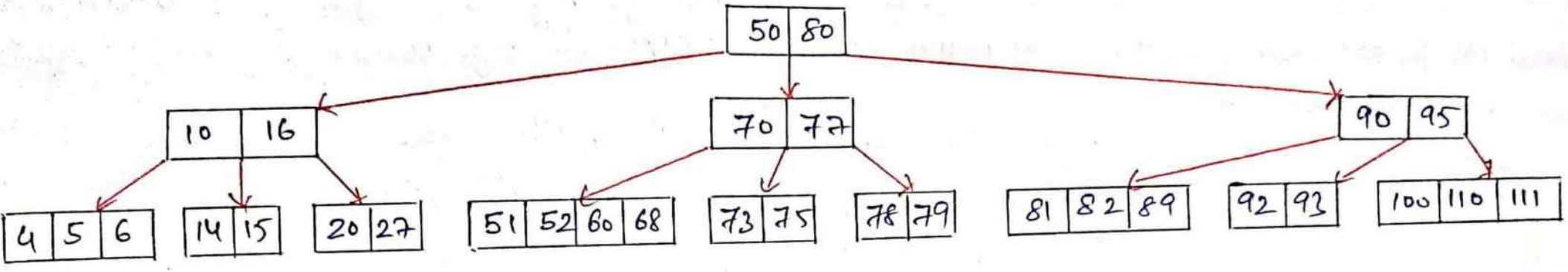
Here, the Key 72 is in leaf node and that leaf node contains minimum number of keys. After deleting key 72, the problem arises is Underflow. The Solution is Borrow key from immediate right sibling.



Delete 65

Here, the Key 65 is in leaf node and that leaf node contains minimum number of keys. After deleting Key 65, the problem arises is Underflow. Borrowing key from immediate left sibling and immediate right sibling is not possible.

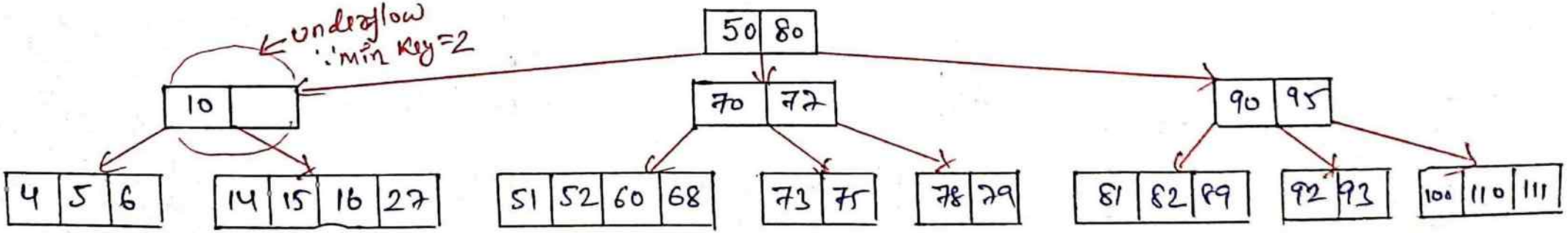
∴ Solution is Bring corresponding parent down, and combine the nodes.



Delete 20

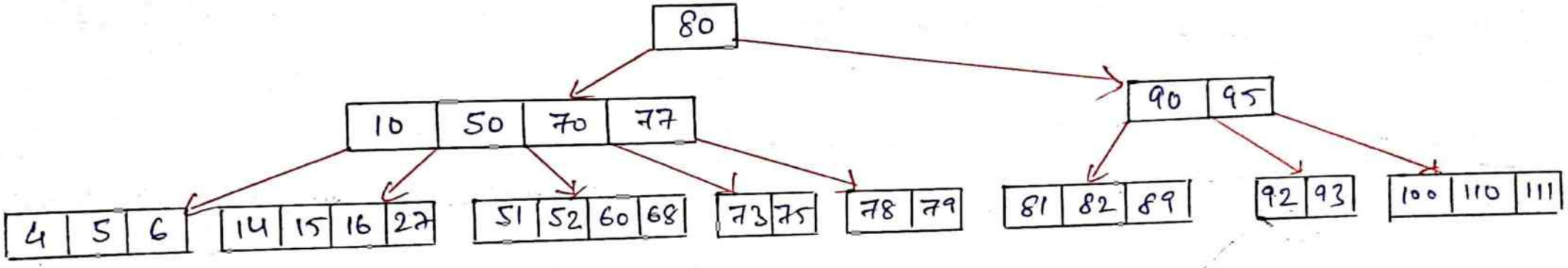
Here, the Key 20 is in leaf node and that leaf node contains minimum number of Keys. After deleting Key 20, the problem arises is underflow. Borrowing Key from immediate left sibling is not possible.

∴ Solution is Bring corresponding parent down, and combine the nodes.



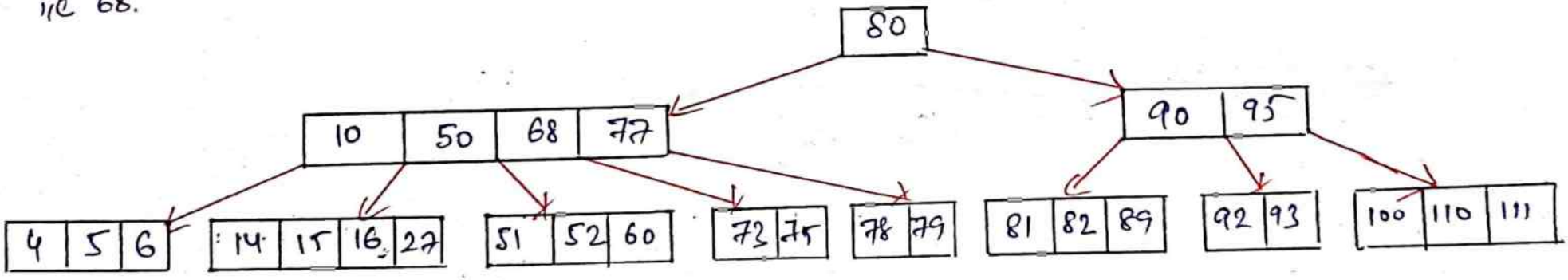
Here, after deleting Key 20, the internal node having Key 10 suffers from underflow. Since its right sibling contains minimum number of keys. So Key borrowing from its right sibling is not possible.

∴ Solution is Bring parent key 50 down and combine the nodes.



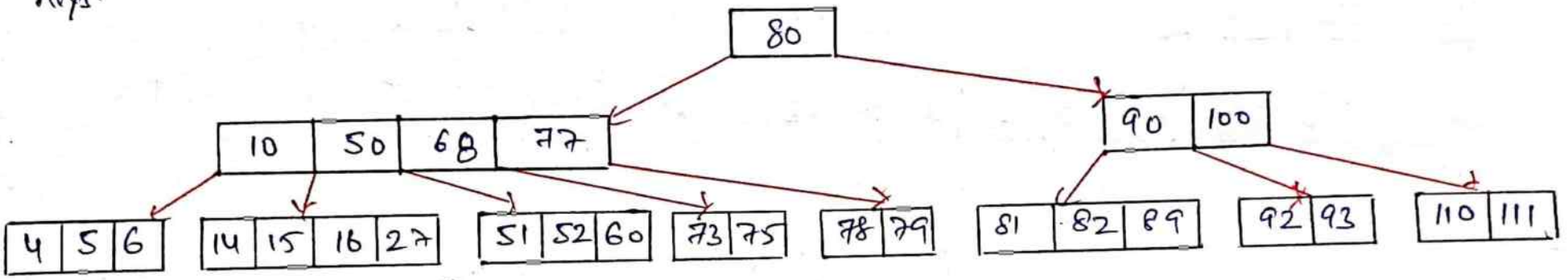
Delete 70

Here, the Key 70 is in internal node. So Replace this with its inorder predecessor Key i/e 68.



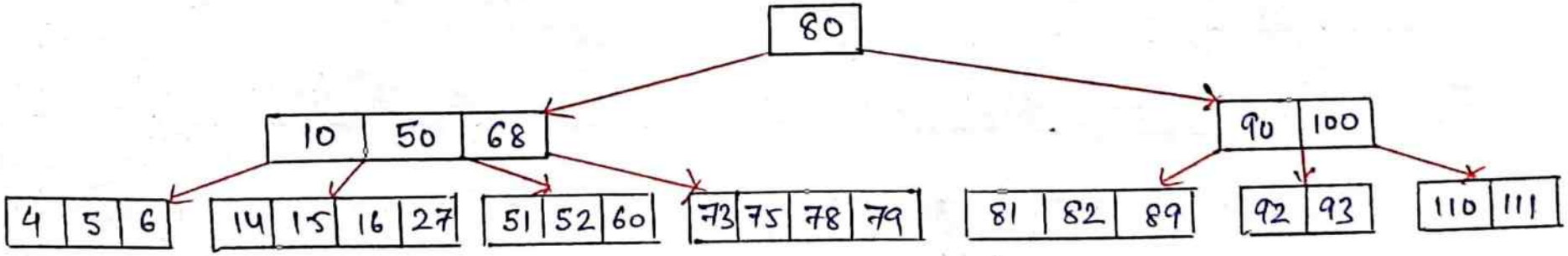
Delete 95

Here, the Key 95 is in internal node. So Replace this with its inorder successor Key 100. We choose inorder successor as inorder predecessor is having minimum number of keys.



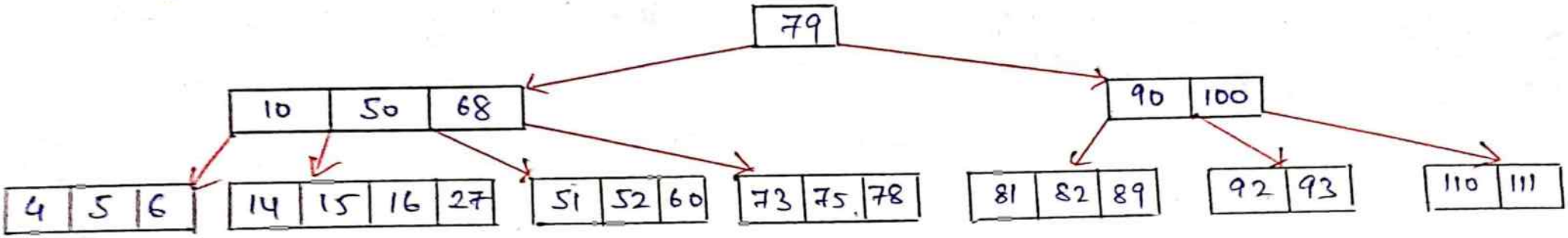
Delete 77

Here, the Key 77 is in internal node. For Key 77, both inorder predecessor and inorder successor is having minimum number of keys. Solution is delete Key 77 and combine its children.



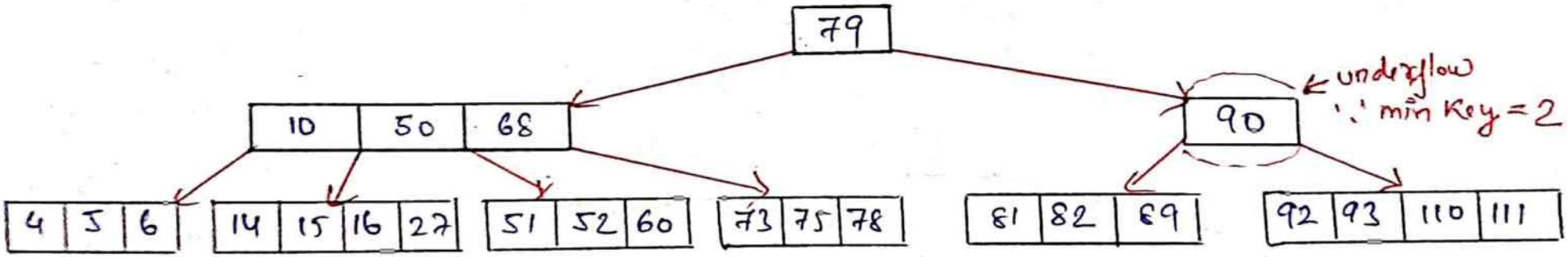
Delete 80

Here, Key 80 is in internal node. Replace by its inorder predecessor key 79.



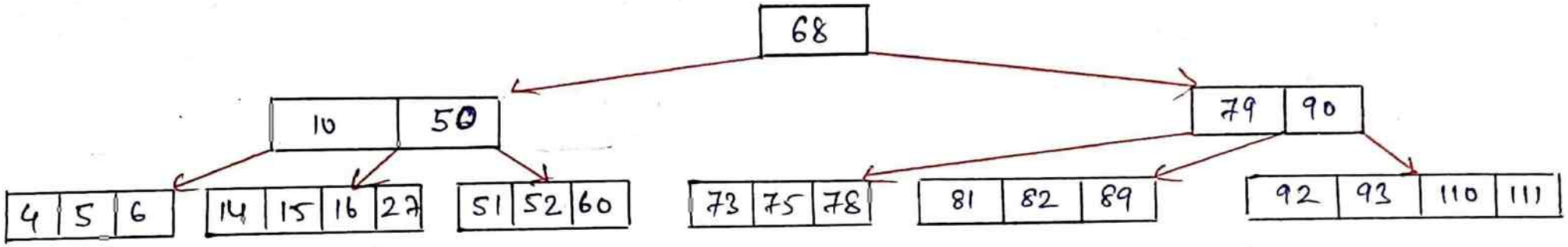
Delete 100

Here, Key 100 is in internal node. For Key 100, both its inorder predecessor and inorder successor is having minimum number of keys. So delete Key 100 and combine its children.



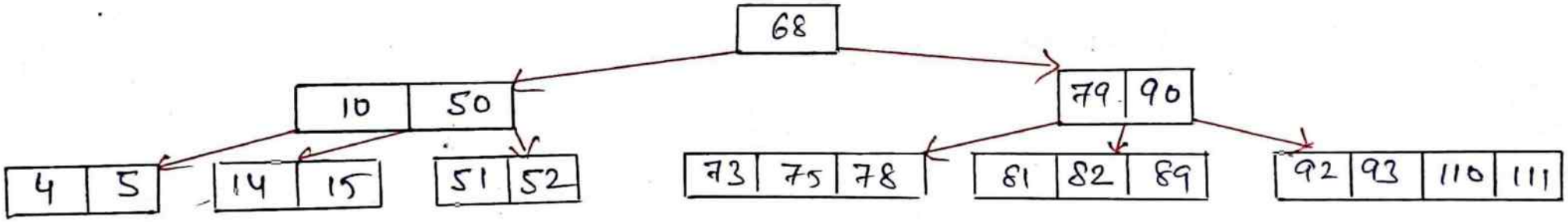
After deleting the Key 100, the internal node having Key 90 suffers from underflow because minimum Key should be 2.

So, Borrow Key from its immediate left sibling.



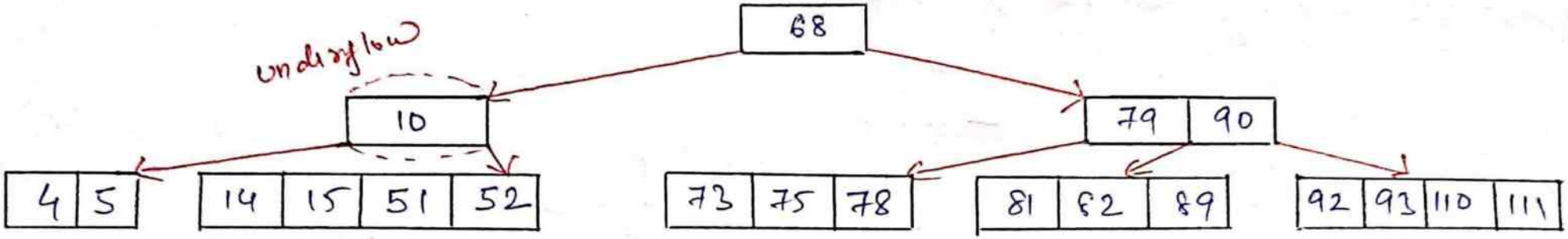
Delete 6, 27, 60, 16

Here these keys are in leaf node and deleting these keys doesnot affect B-tree as the corresponding leaf nodes contains more than minimum number of keys.



Delete 50

Here, the key 50 is in internal node. Both its Left subtree and right subtree are having minimum number of keys. So we can't replace by its inorder predecessor and inorder successor key. So solution is combine its children.



Here, underflow occurs in internal node. Borrow key from its sibling. But key borrowing is not possible (∵ immediate right sibling has minimum number of keys). So bring parent key 68 down and combine.

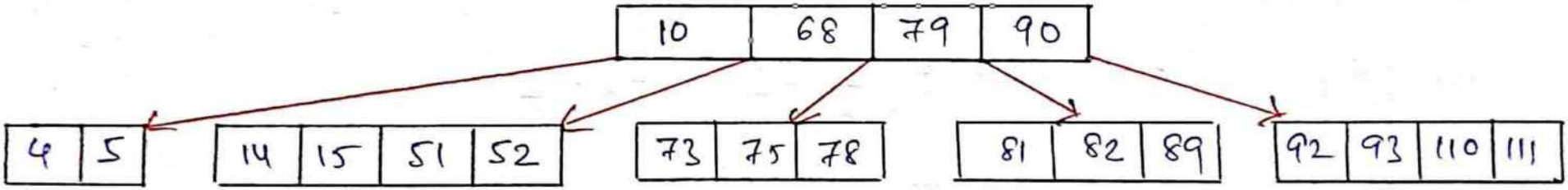
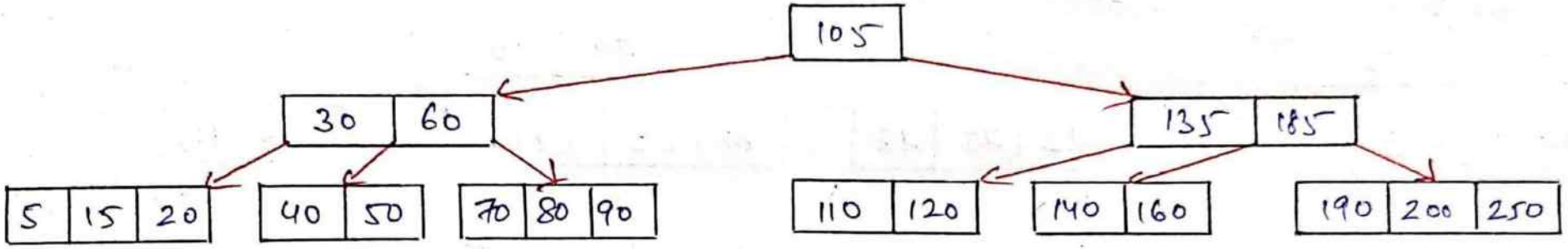


fig B-tree of order 5.

Q Consider the following B-tree of order 5.



Delete 190, 60, 40, 140.

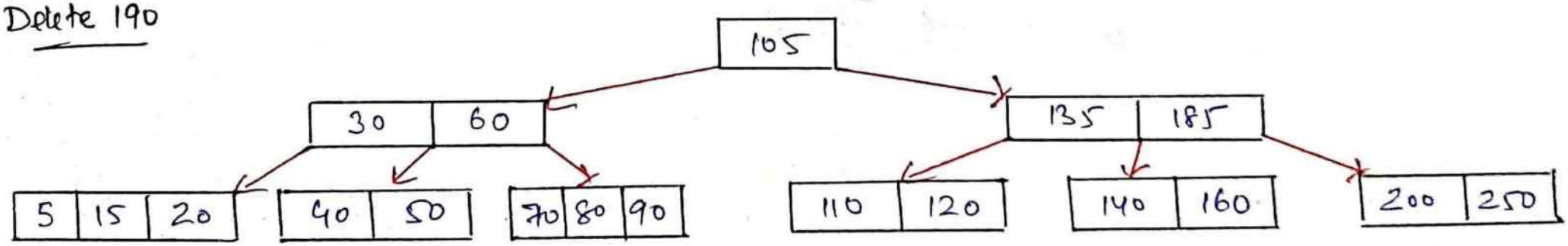
Soln Order = $m = 5$

$\therefore$ Minimum children in each node $= \lceil \frac{m}{2} \rceil = \lceil \frac{5}{2} \rceil = \lceil 2.5 \rceil = 3$

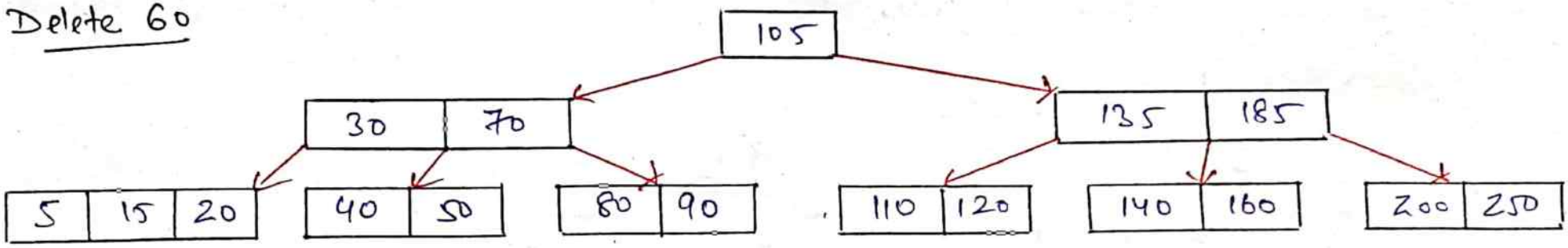
$\therefore$ Minimum Keys in each node $= \lceil \frac{m}{2} \rceil - 1 = 3 - 1 = 2$

Note: Root node can have minimum 1 key.

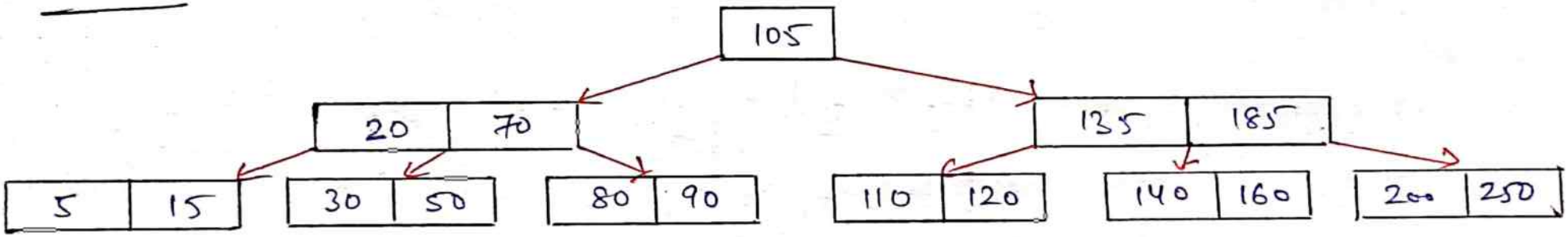
Delete 190



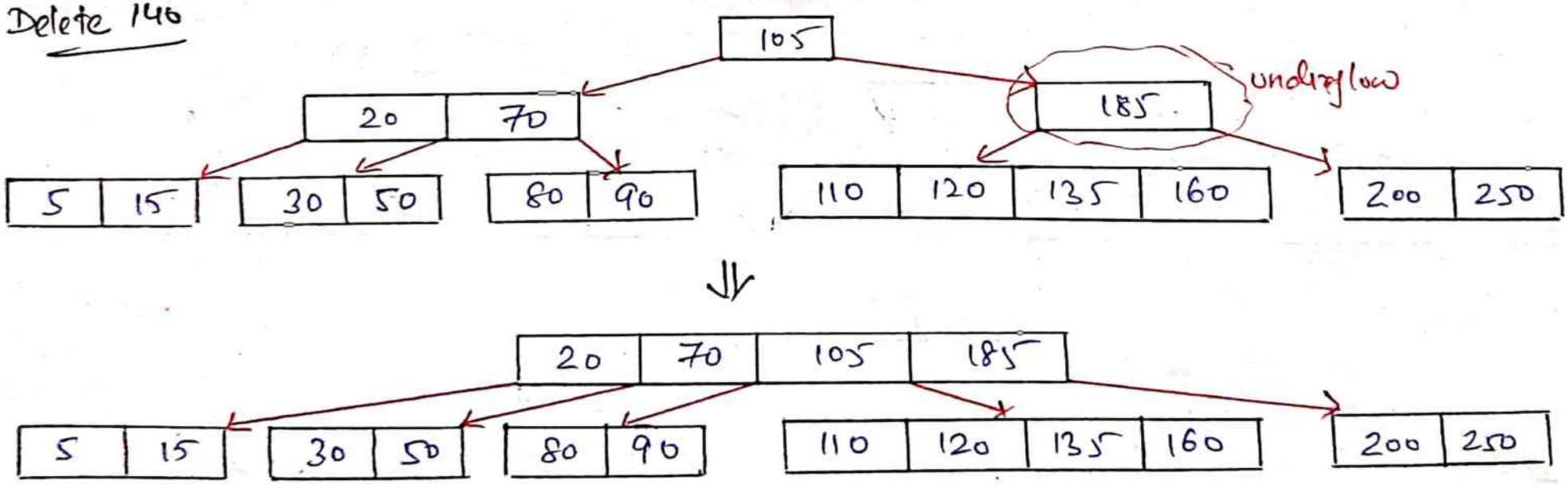
Delete 60



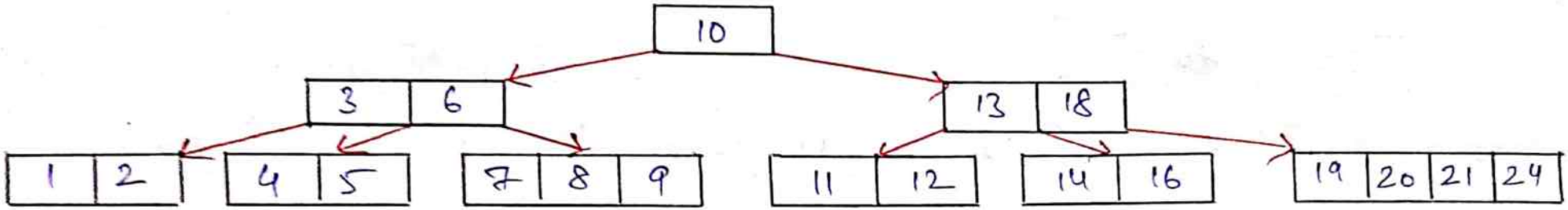
Delete 40



Delete 140



Q Consider the following B-tree of order 5.



Delete 8, 18, 16, 4

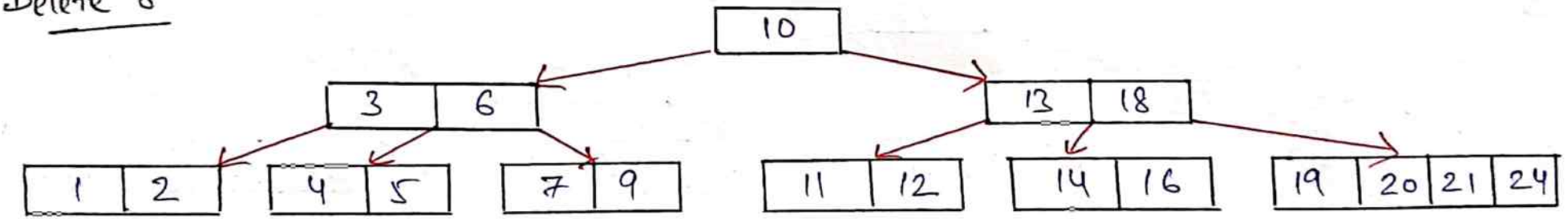
Soln Order = m = 5

∴ Minimum children in each node $= \lceil \frac{m}{2} \rceil = \lceil \frac{5}{2} \rceil = \lceil 2.5 \rceil = 3$

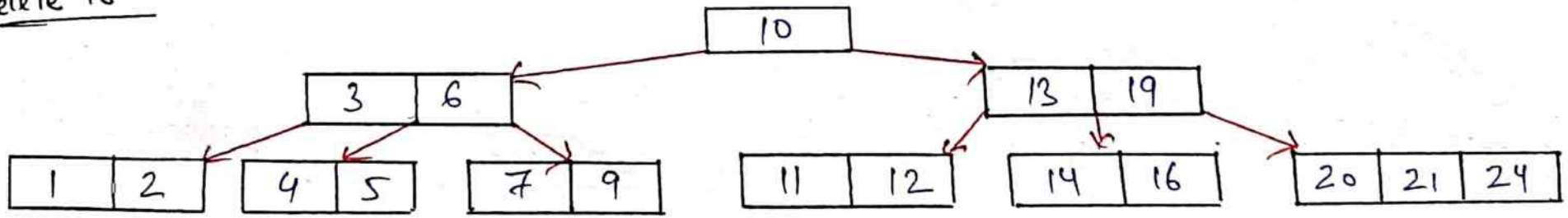
Minimum Key in each node $= \lceil \frac{m}{2} \rceil - 1 = 3 - 1 = 2$

Note: Root node can have minimum 1 key.

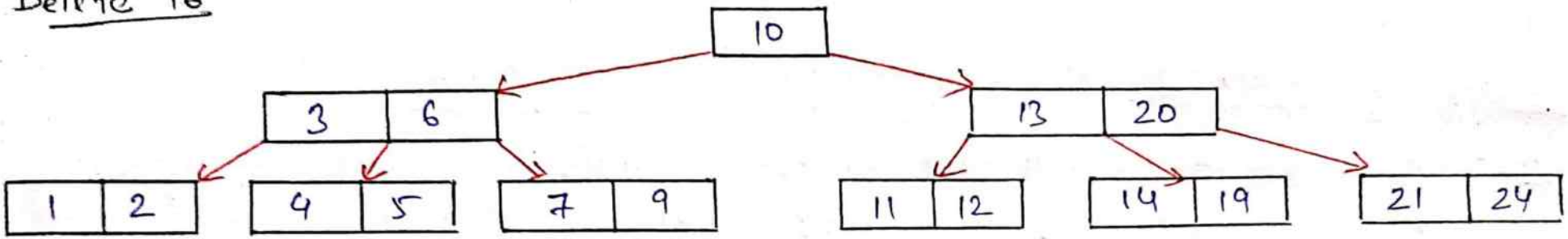
Delete 8



Delete 18



Delete 16



Delete 4

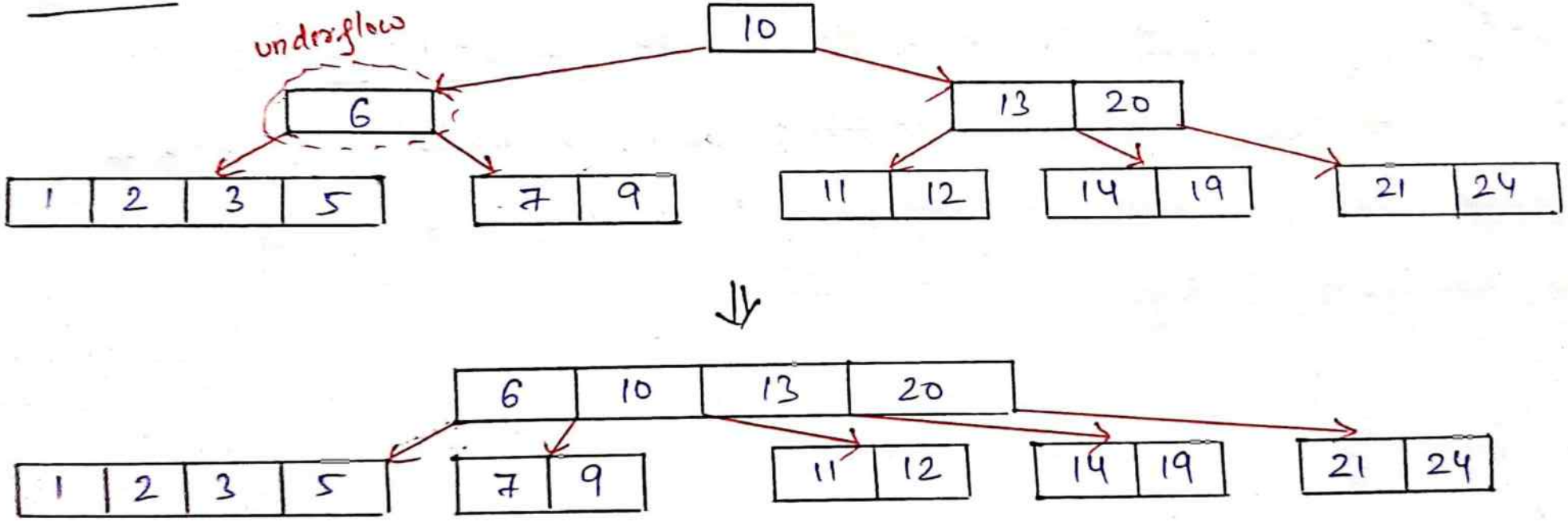


fig B-tree of order 5

NOTE

① B-tree of order 3 is also known as 2-3 tree.

A 2-3 tree is a search tree where a node can have 2 children or maximum 3 children i,e 1 key or maximum 2 keys.

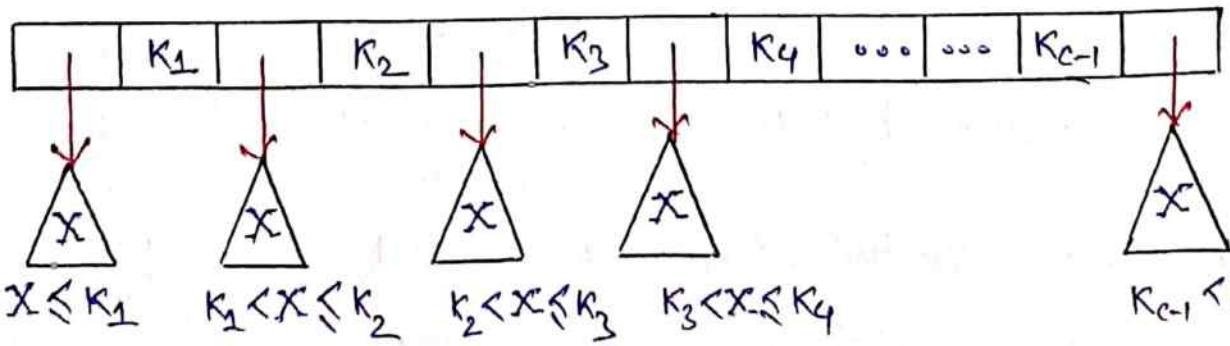
② B-tree of order 4 is also known as 2-3-4 tree.

A 2-3-4 tree is a search tree where a node can have 2 children or 3 children or maximum 4 children i,e 1 key or 2 keys or maximum 3 keys.

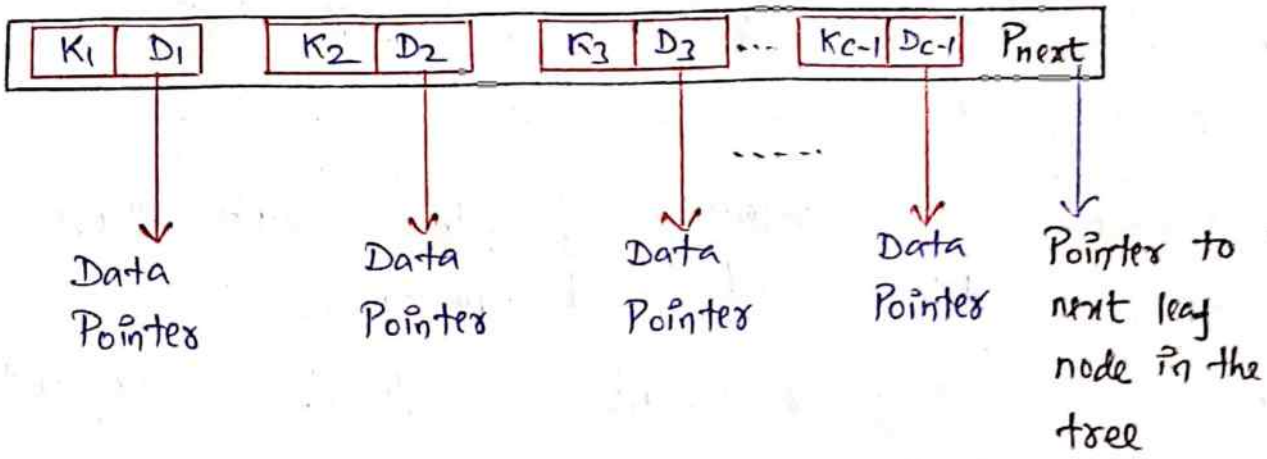
B+ tree

- One of the drawback of B-tree is the difficulty of traversing the keys (or) record sequentially.
- B+ tree is an extension of B tree which allows efficient insertion, deletion and search operations.
- In B tree, keys or record are stored in the internal as well as leaf nodes.
  Whereas in B+ tree, keys or record are stored only in the leaf nodes while the internal nodes only stores the index to these records.
- B+ tree are used to store the large amount of data which cannot be stored in the main memory.
  Due to the fact that, size of main memory is always limited, the internal nodes (index to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.
- The internal nodes of B+ tree are also called index nodes.

Structure of internal nodes of B+ tree



Structure of the leaf nodes of a B+ tree



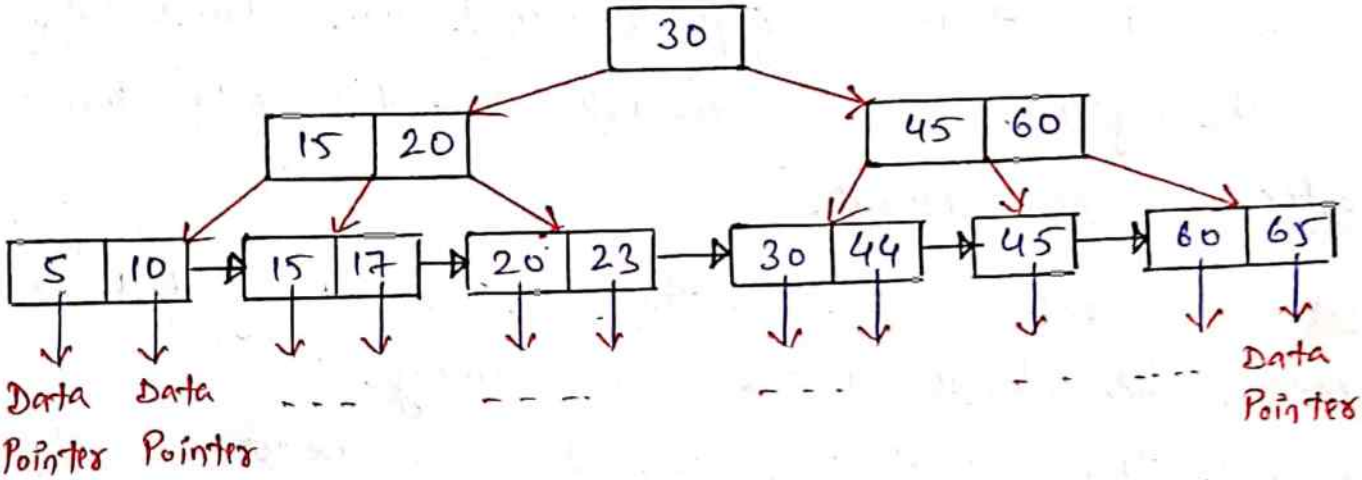


fig B+ tree of order 3

Note: All Keys or record are present in leaf nodes.

B Tree	B+ Tree
1. Key or record can be stored in leaf nodes as well as internal node.	1. Key or record can only be stored in leaf nodes
2. Search can end at any node.	2. Search always ends at leaf node.
3. No redundant Keys.	3. Redundant Keys may exist.
4. Slow Sequential access.	4. Efficient Sequential access.
5. Higher trees.	5. Flatter trees.