SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE NAME : 23CST207
- DATABASE MANAGEMENT SYSTEMS

II YEAR / IV SEMESTER

Unit 5- Physical Storage and MongoDB
Topic 5 : B tree and B+ Tree

B tree

- B-Tree is a self-balancing search tree. In most of the other self-balancing search trees (like AVL and Red-Black Trees)

(OR)

- A B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children

Cont..

- To understand the use of B-Trees, we must think of the huge amount of data that cannot fit in main memory
- The main idea of using B-Trees is to reduce the number of disk accesses.

Cont ..

- Considerations for disk-based storage systems.
- Indexed Sequential Access Method (ISAM)
- *m*-way search trees
- B-trees

Properties of B-Tree

- **B-Tree of Order m** has the following properties...
- **Property #1** - All **leaf nodes** must be **at same level**.
- **Property #2** - All nodes except root must have at least **[m/2]-1** keys and maximum of **m-1** keys.
- **Property #3** - All non leaf nodes except root (i.e. all internal nodes) must have at least **m/2** children.
- **Property #4** - If the root node is a non leaf node, then it must have **atleast 2** children.
- **Property #5** - A non leaf node with **n-1** keys must have **n** number of children.
- **Property #6** - All the **key values in a node** must be in **Ascending Order**.

Insertion

- **Step 1 -** Check whether tree is Empty.
- **Step 2 -** If tree is **Empty**, then create a new node with new key value and insert it into the tree as a root node.
- **Step 3 -** If tree is **Not Empty**, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.

Cont..

- **Step 4 -** If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.
- **Step 5 -** If that leaf node is already full, **split** that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.
- **Step 6 -** If the spilting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

Insertions Algorithm

- **def** insert (entry) :
  - Find target leaf *L*
  - **if** *L* has less than *m* – 2 entries :
    - add the entry

    else :
    - Allocate new leaf L'
    - Pick the m/2 highest keys of *L* and move them to *L'*
    - Insert ***highest key*** of *L* and corresponding address leaf into the parent node
    - If the parent is full :
      - Split it and add the middle key to its parent node
    - Repeat until a parent is found that is not full

Deletion

- **def** delete (record) :
  - Locate target leaf and remove the entry
  - If leaf is less than half full:
    - Try to re-distribute, taking from sibling (adjacent node with same parent)
    - If re-distribution fails:
      - Merge leaf and sibling
      - Delete entry to one of the two merged leaves
      - Merge could propagate to root

Problem

- Construct a B-Tree of order 5 following numbers
- 3,14,7,1,8,5,11,17,13,6,23,12,20,26,4,16,18,

  24,25,19

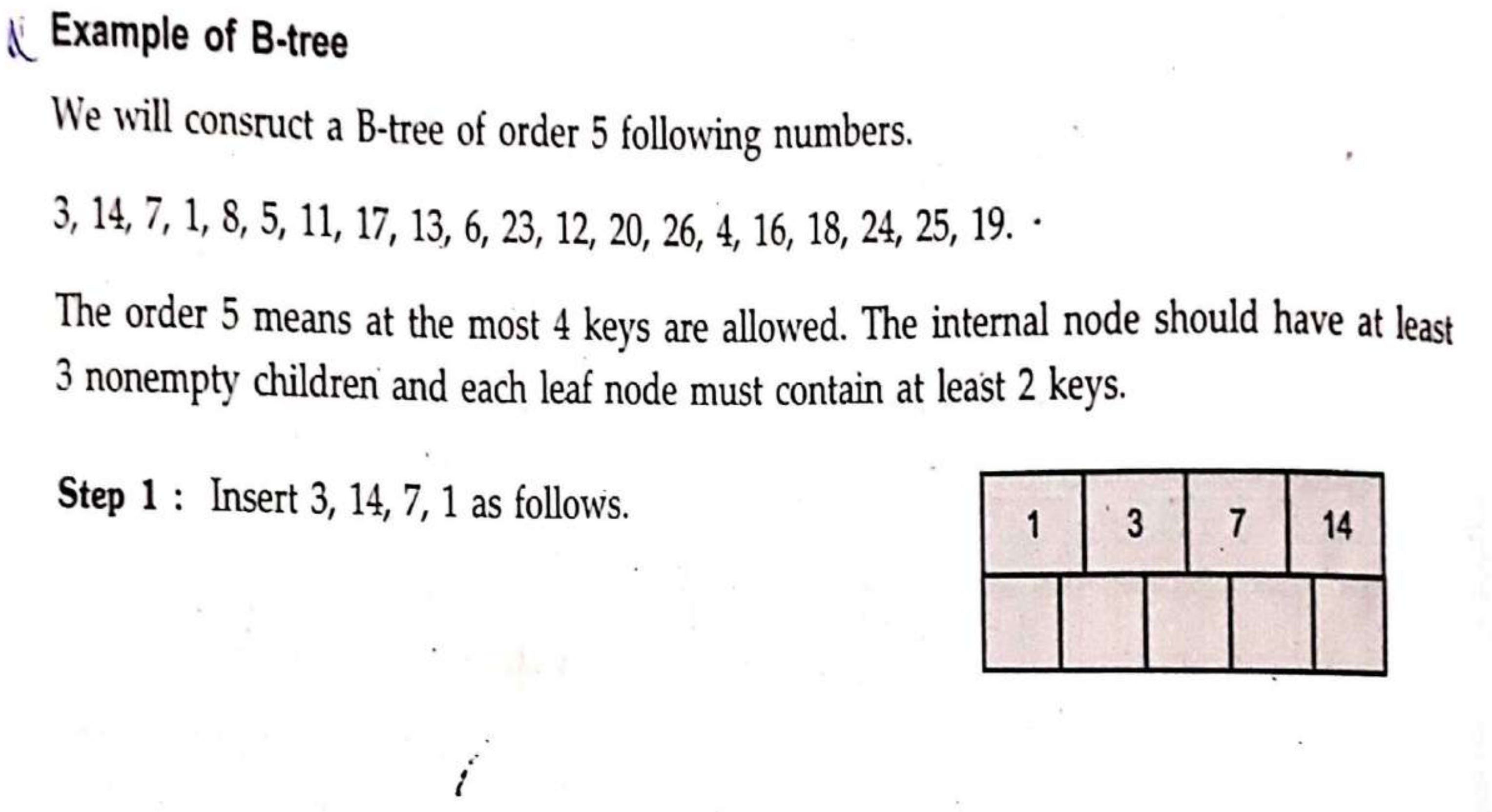
the order is 5

Mar child =5

Min child= 5 / 2 =2.5 and Max keys = m-1 ,

i.e 5-1=4 , Min keys =(5/2)-1=2.5-1=1.5

Cont..



Example of B-tree

We will consruct a B-tree of order 5 following numbers.

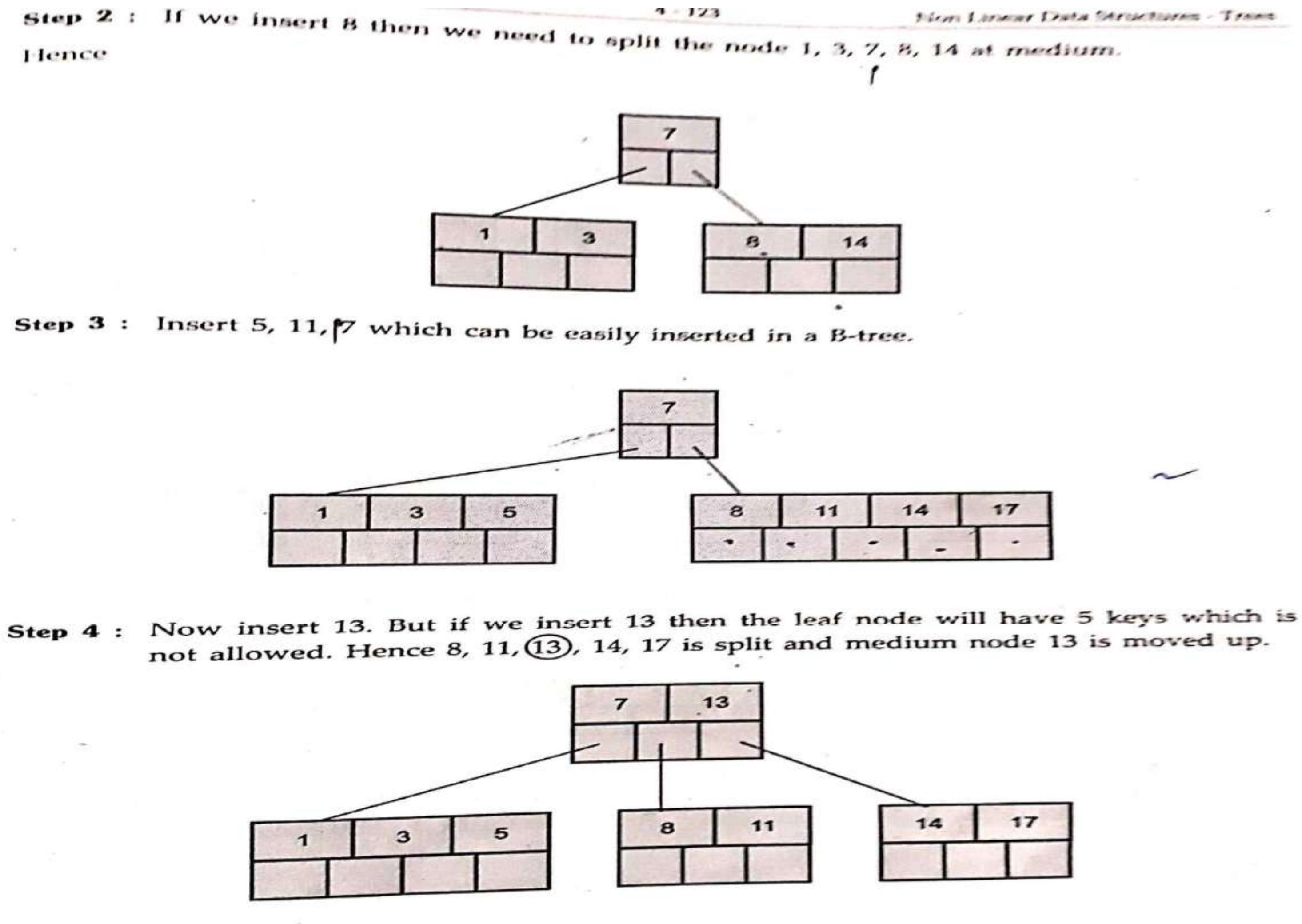
3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19.

The order 5 means at the most 4 keys are allowed. The internal node should have at least 3 nonempty children and each leaf node must contain at least 2 keys.

Step 1 : Insert 3, 14, 7, 1 as follows.

1	3	7	14

Cont..



Step 2 : If we insert 8 then we need to split the node 1, 3, 7, 8, 14 at medium. Hence

7
1
3
8
14

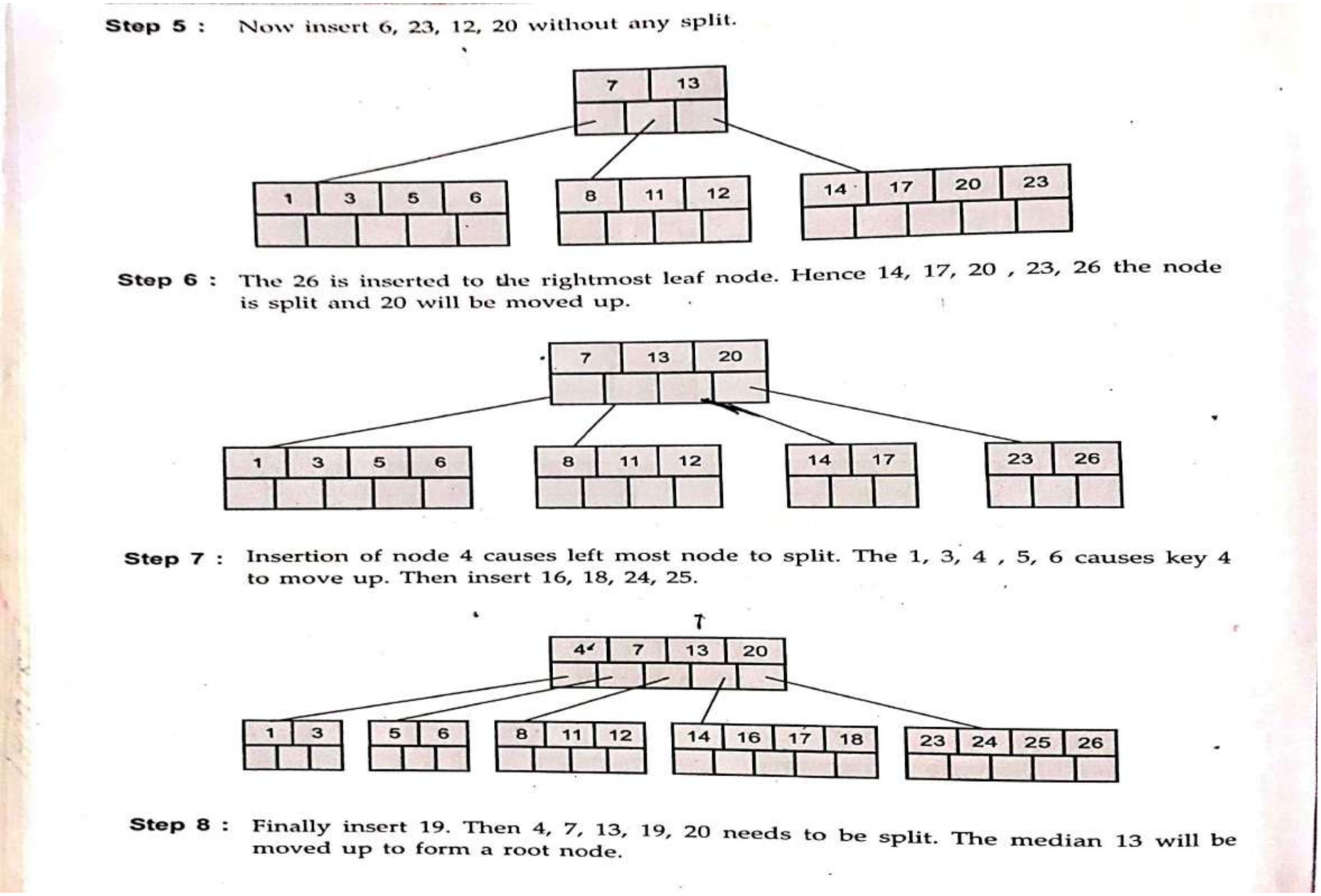
Step 3 : Insert 5, 11, 17 which can be easily inserted in a B-tree.

7
1
3
5
8
11
14
17

Step 4 : Now insert 13. But if we insert 13 then the leaf node will have 5 keys which is not allowed. Hence 8, 11, 13, 14, 17 is split and medium node 13 is moved up.

7
13
1
3
5
8
11
14
17

Cont..



Step 5 : Now insert 6, 23, 12, 20 without any split.

7 13
1 3 5 6
8 11 12
14 17 20 23

Step 6 : The 26 is inserted to the rightmost leaf node. Hence 14, 17, 20 , 23, 26 the node is split and 20 will be moved up.

7 13 20
1 3 5 6
8 11 12
14 17
23 26

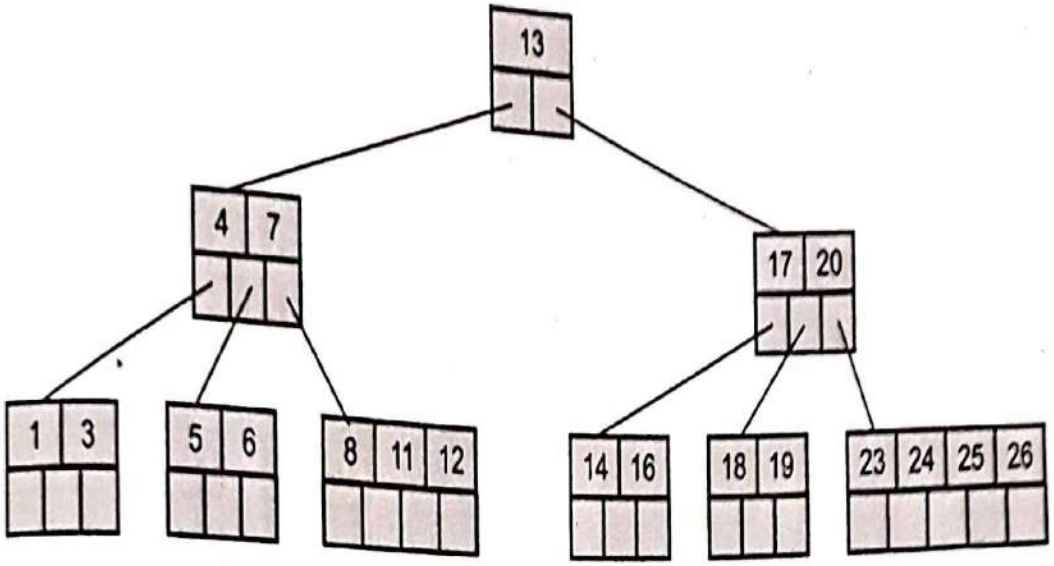
Step 7 : Insertion of node 4 causes left most node to split. The 1, 3, 4 , 5, 6 causes key 4 to move up. Then insert 16, 18, 24, 25.

4 7 13 20
1 3
5 6
8 11 12
14 16 17 18
23 24 25 26

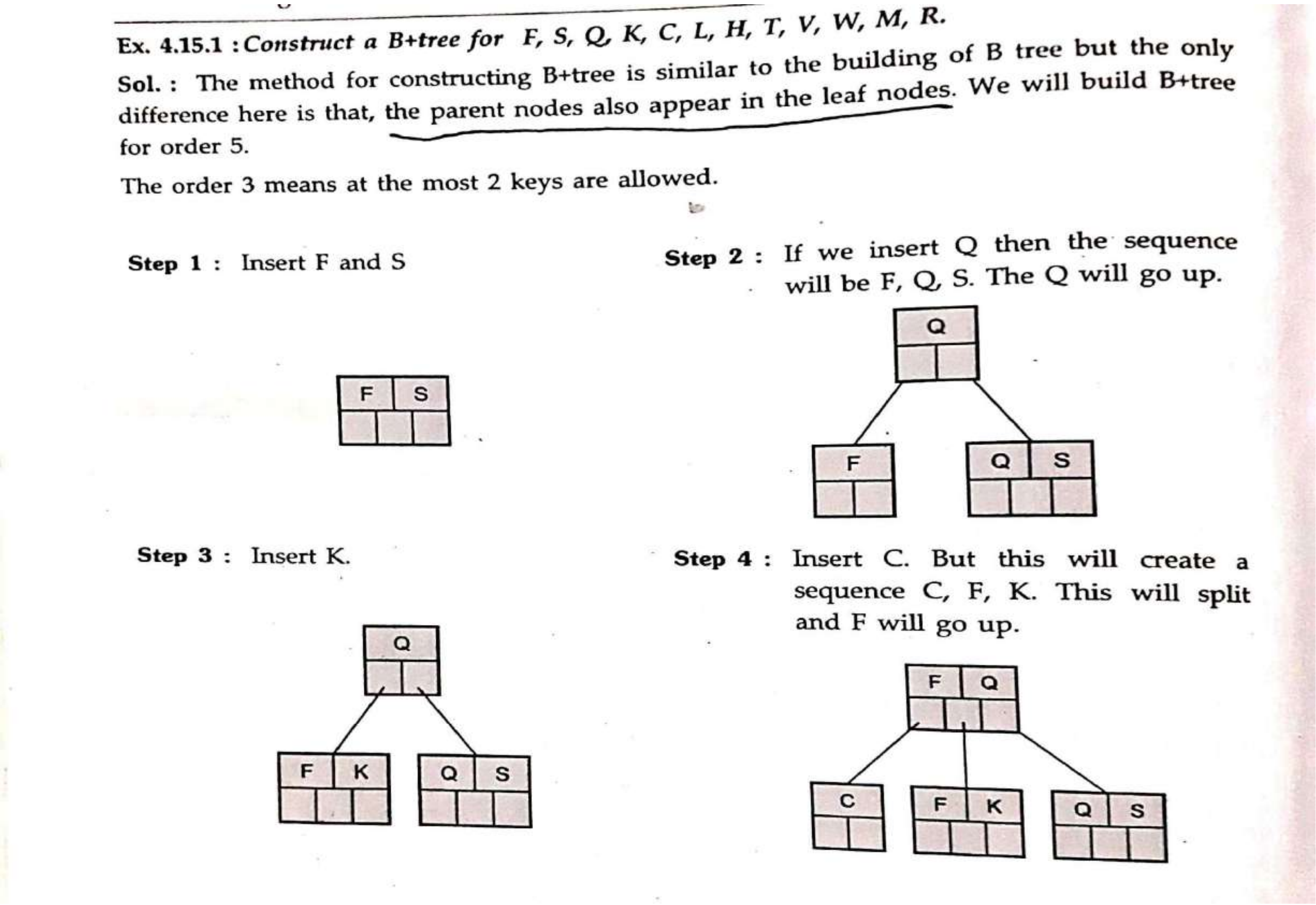
Step 8 : Finally insert 19. Then 4, 7, 13, 19, 20 needs to be split. The median 13 will be moved up to form a root node.

Cont..

The tree then will be -



B+ Tree



Ex. 4.15.1 : *Construct a B+tree for F, S, Q, K, C, L, H, T, V, W, M, R.*

Sol. : The method for constructing B+tree is similar to the building of B tree but the only difference here is that, the parent nodes also appear in the leaf nodes. We will build B+tree for order 5.

The order 3 means at the most 2 keys are allowed.

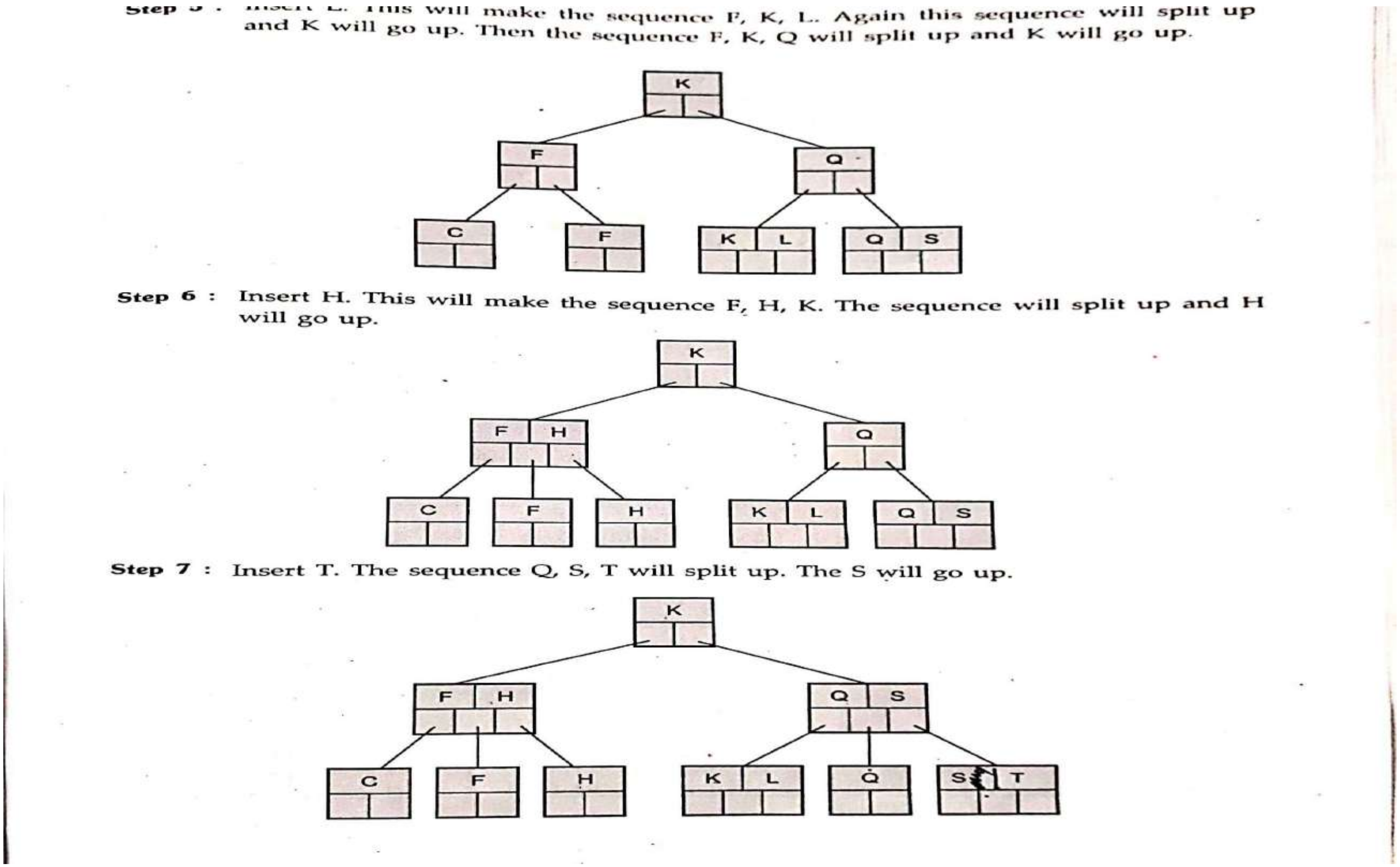
Step 1 : Insert F and S

Step 2 : If we insert Q then the sequence will be F, Q, S. The Q will go up.

Step 3 : Insert K.

Step 4 : Insert C. But this will create a sequence C, F, K. This will split and F will go up.

Cont..



Step 5 : Insert L. This will make the sequence F, K, L. Again this sequence will split up and K will go up. Then the sequence F, K, Q will split up and K will go up.

K
F
Q
C
F
K L
Q S

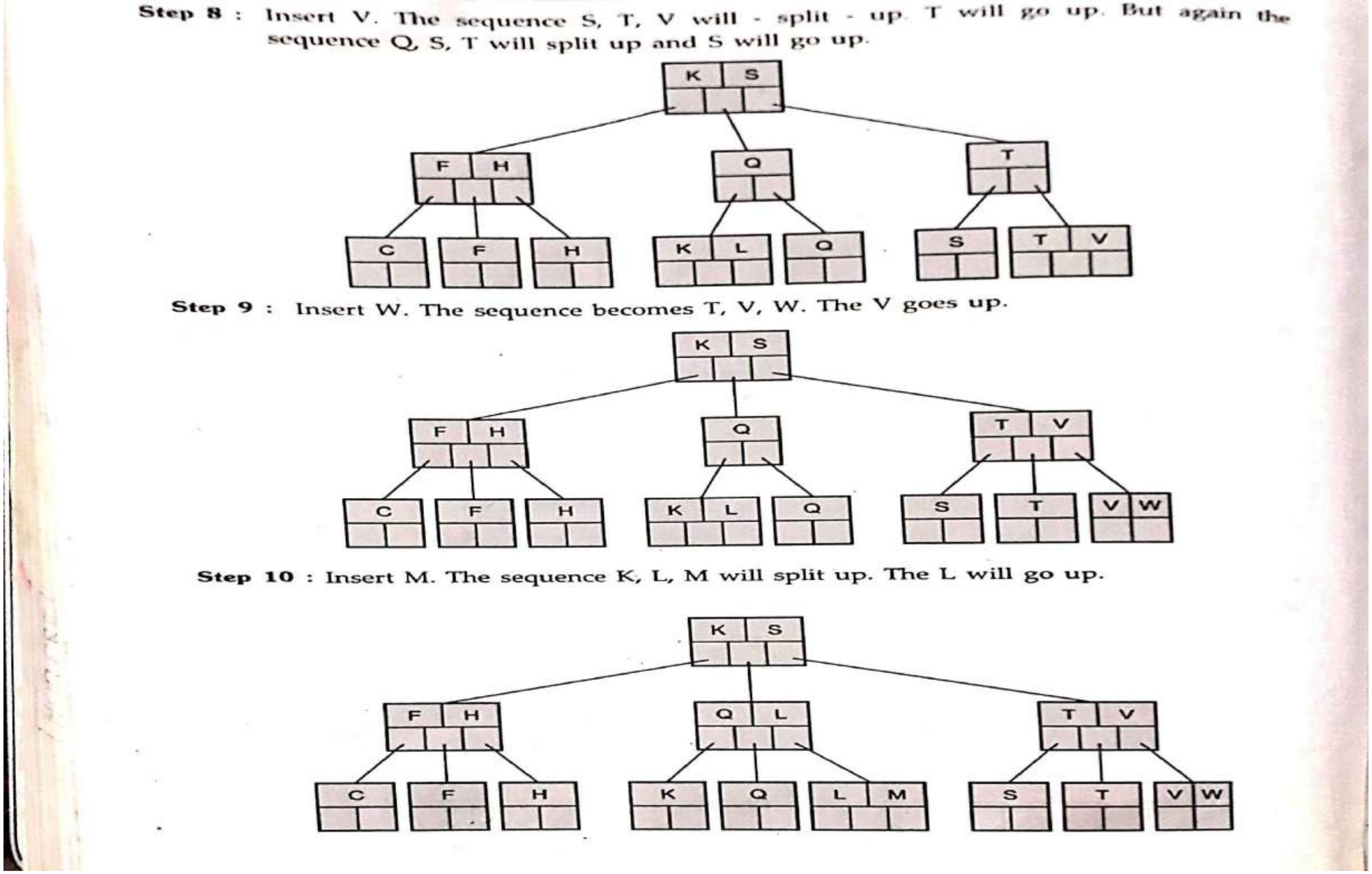
Step 6 : Insert H. This will make the sequence F, H, K. The sequence will split up and H will go up.

K
F H
Q
C
F
H
K L
Q S

Step 7 : Insert T. The sequence Q, S, T will split up. The S will go up.

K
F H
Q S
C
F
H
K L
Q
S T

Cont..



Step 8 : Insert V. The sequence S, T, V will - split - up. T will go up. But again the sequence Q, S, T will split up and S will go up.

Step 9 : Insert W. The sequence becomes T, V, W. The V goes up.

Step 10 : Insert M. The sequence K, L, M will split up. The L will go up.

Cont..

Step 11 : Insert R.

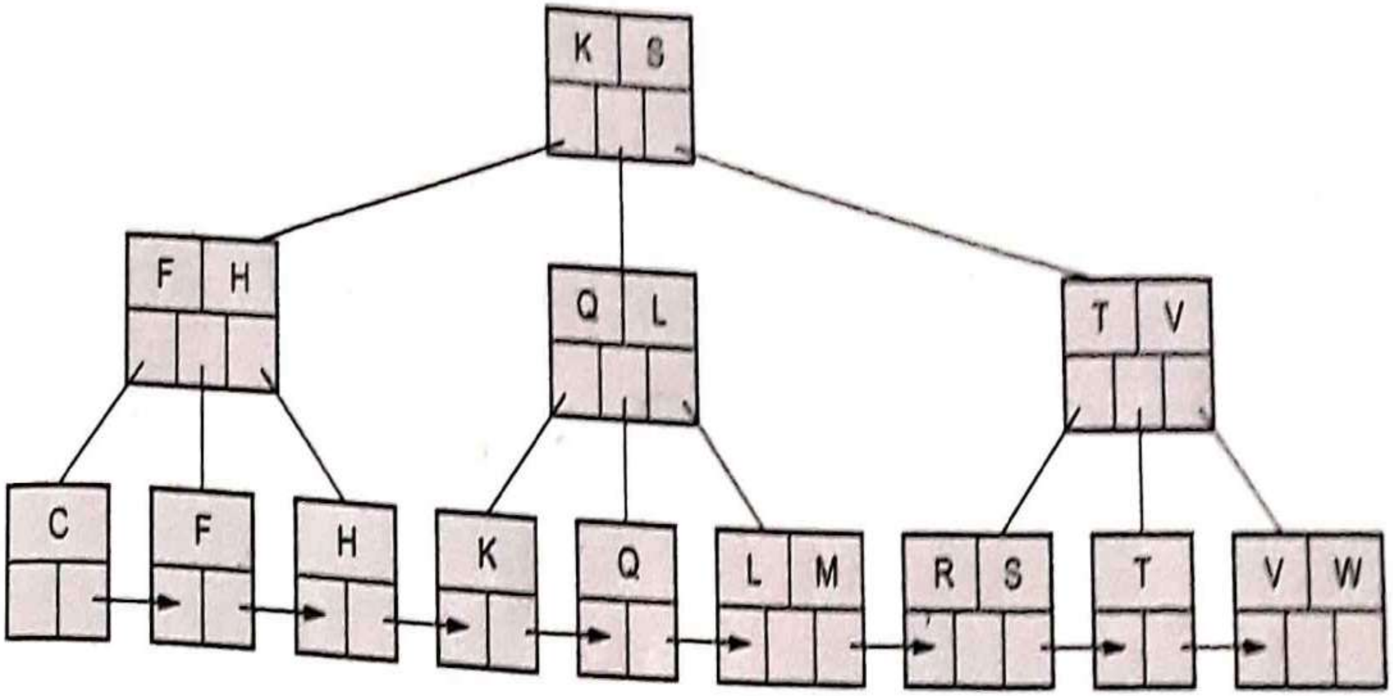
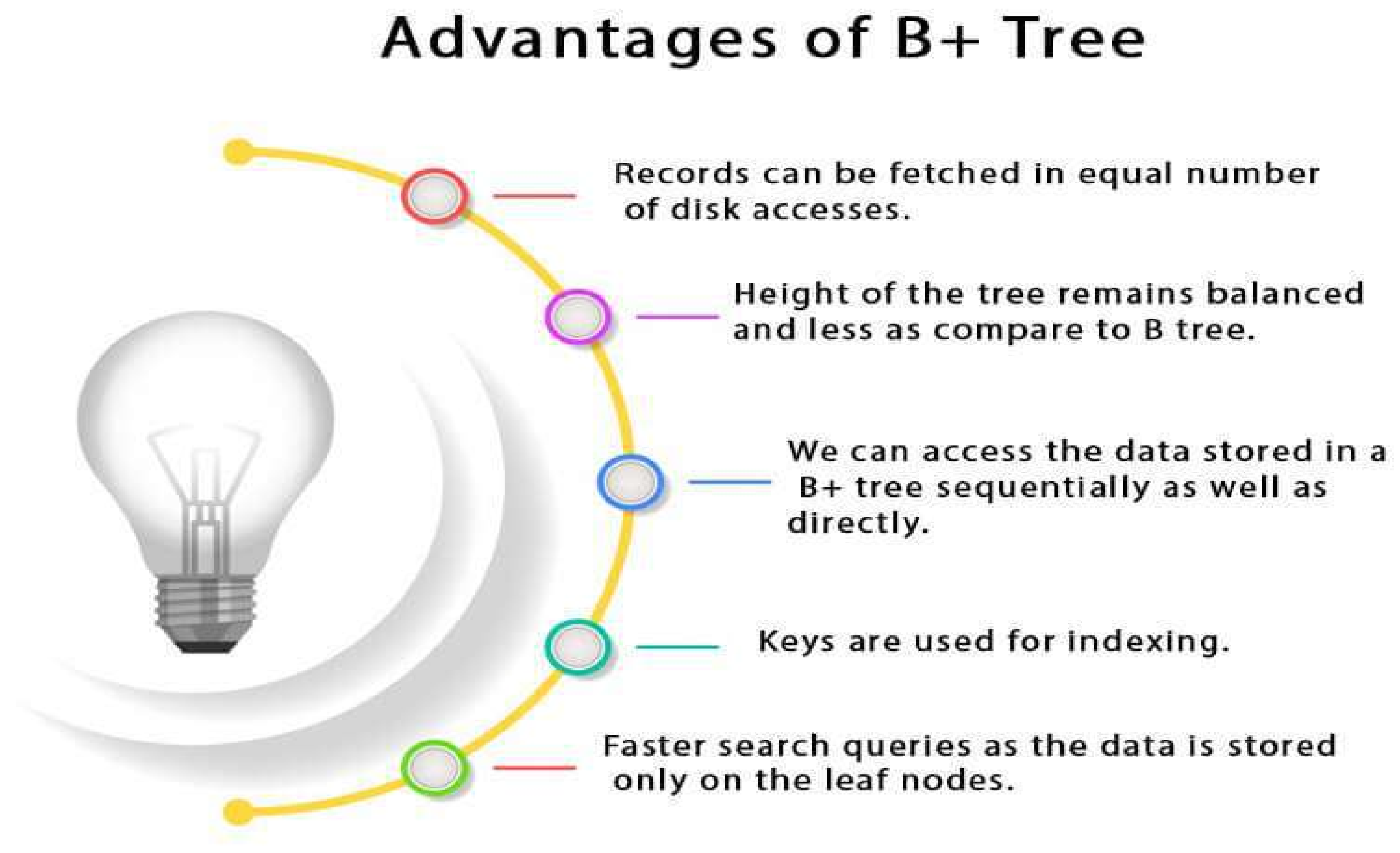


Fig. 4.15.2

Advantages of B-tree

- The B-tree uses all of the ideas described above. In particular, a B-tree:
- keeps keys in sorted order for sequential traversing
- uses a hierarchical index to minimize the number of disk reads
- uses partially full blocks to speed insertions and deletions
- keeps the index balanced with a recursive algorithm

Advantages of B+tree



Advantages of B+ Tree

- Records can be fetched in equal number of disk accesses.
- Height of the tree remains balanced and less as compare to B tree.
- We can access the data stored in a B+ tree sequentially as well as directly.
- Keys are used for indexing.
- Faster search queries as the data is stored only on the leaf nodes.

B Tree Vs B+ Tree

SN	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes are so complicated and time consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

Thank you