11/6/2023

3.1 Multiway Search Tree

BTree

Multiway Search Tree

- Generalised versions of binary trees where each node contains multiple elements

Multiway Search Tree

- A Multiway Search Tree of **order n** is a general tree in which
  - each node has **n or fewer subtrees** and
  - contains no of keys as **one less than no of subtrees**

- i.e. In an **m-Way tree of order n**,
  - each node contains a **maximum of n – 1 elements and n children.**

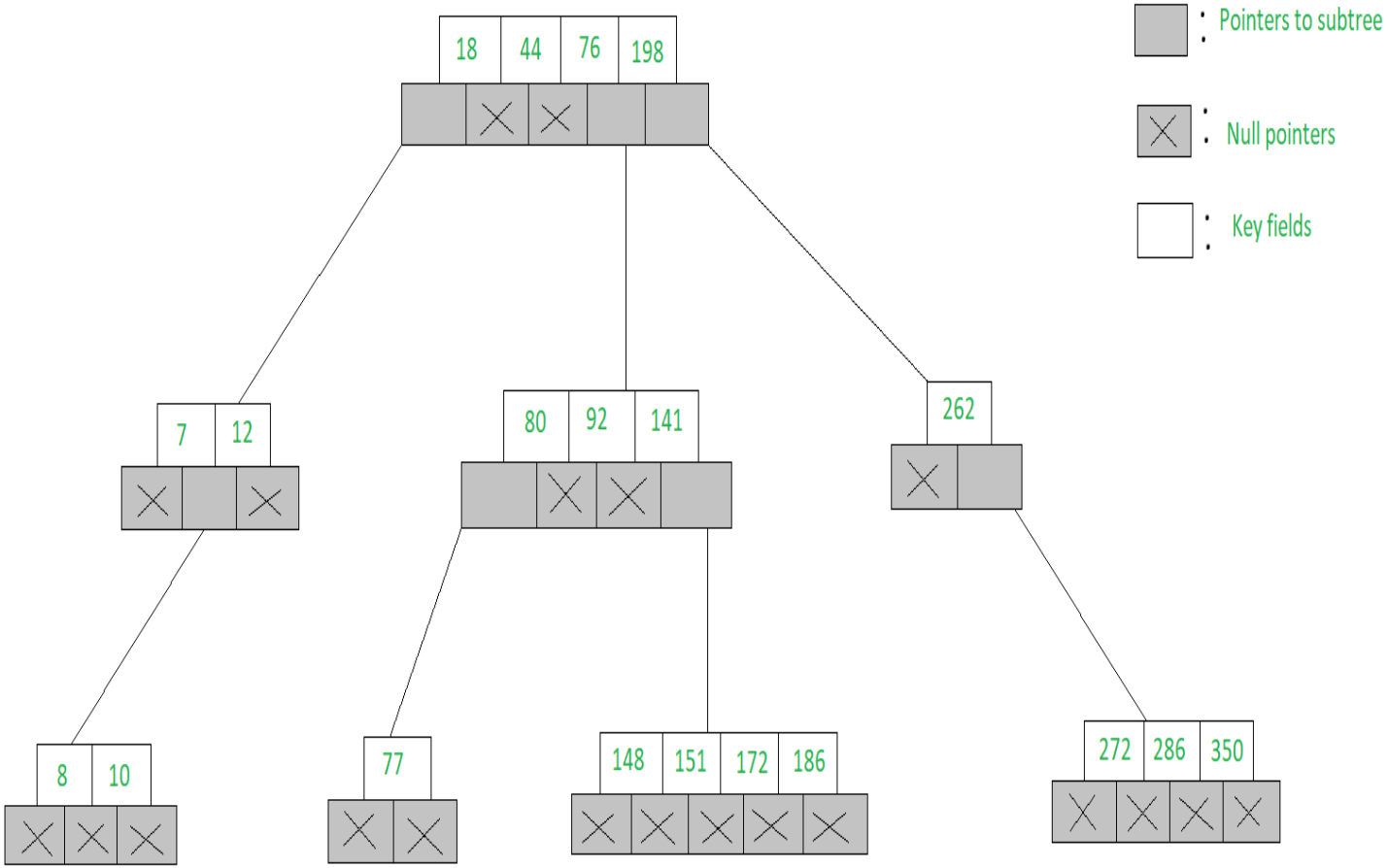
- If the Node has **4 subtrees, it contains 3 keys**

Multiway Search Tree

- The goal of m-Way search tree of **height h** calls for **O(h) no. of accesses for an insert/delete/retrieval operation.**

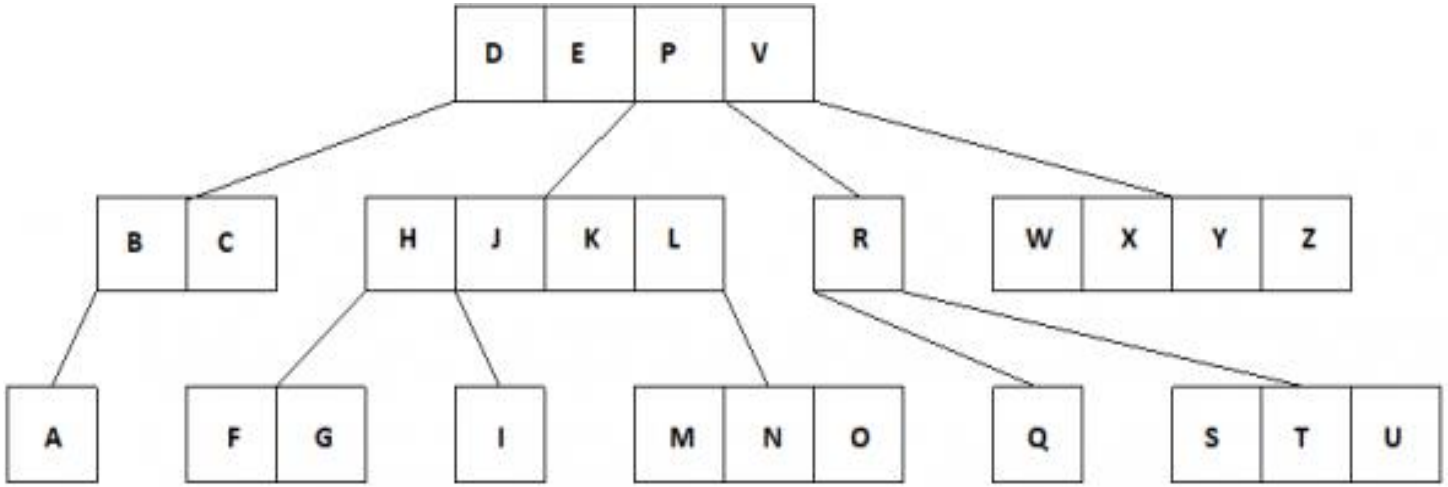
Multiway Search Tree

- 5-Way search tree
- Each node has at most **5 child nodes and at most 4 keys**

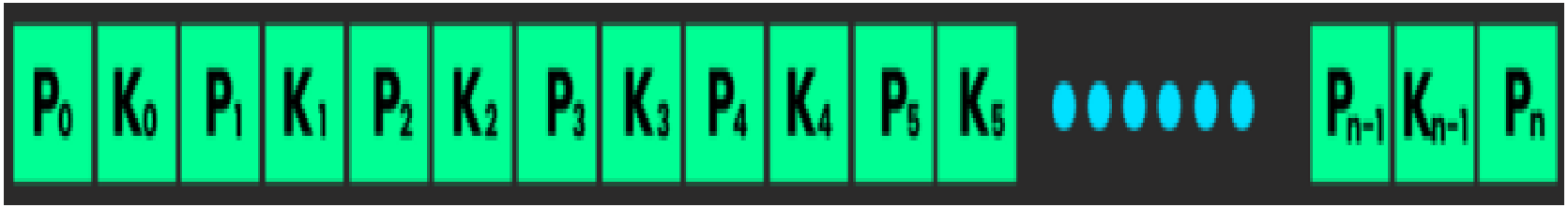


Multiway Search Tree

- 5-Way search tree
- Multiway tree of **order 5**
- Each node has at most **5 child nodes and at most 4 keys**

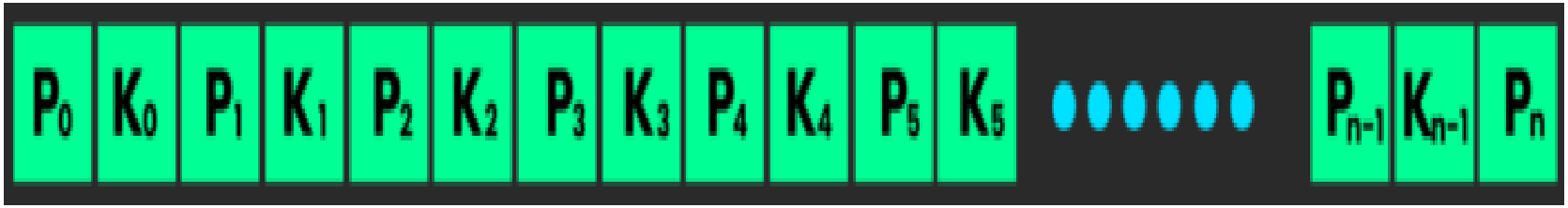


Structure of M-Way Search Tree Node



- P0, P1 …, Pn are the **pointers to the node's sub-trees**
- K0, K1, …, Kn-1 are the **key values of the node.**

Structure of M-Way Search Tree Node



- All the key values are stored in **ascending order.**

- The basic properties of M-way search trees:

1) **Key vales** in the sub tree pointed by **P0 < key value K0**.
2) **Key values** in the sub-tree pointed by **P1 > key value K0**

Similar pattern for the rest of the P's and K's.

Multiway Search Tree

- Structure of a node of an m-Way tree

```
struct node {
        int count;
        int value[MAX];
        struct node* child[MAX + 1];
        };
```

Multiway Search Tree

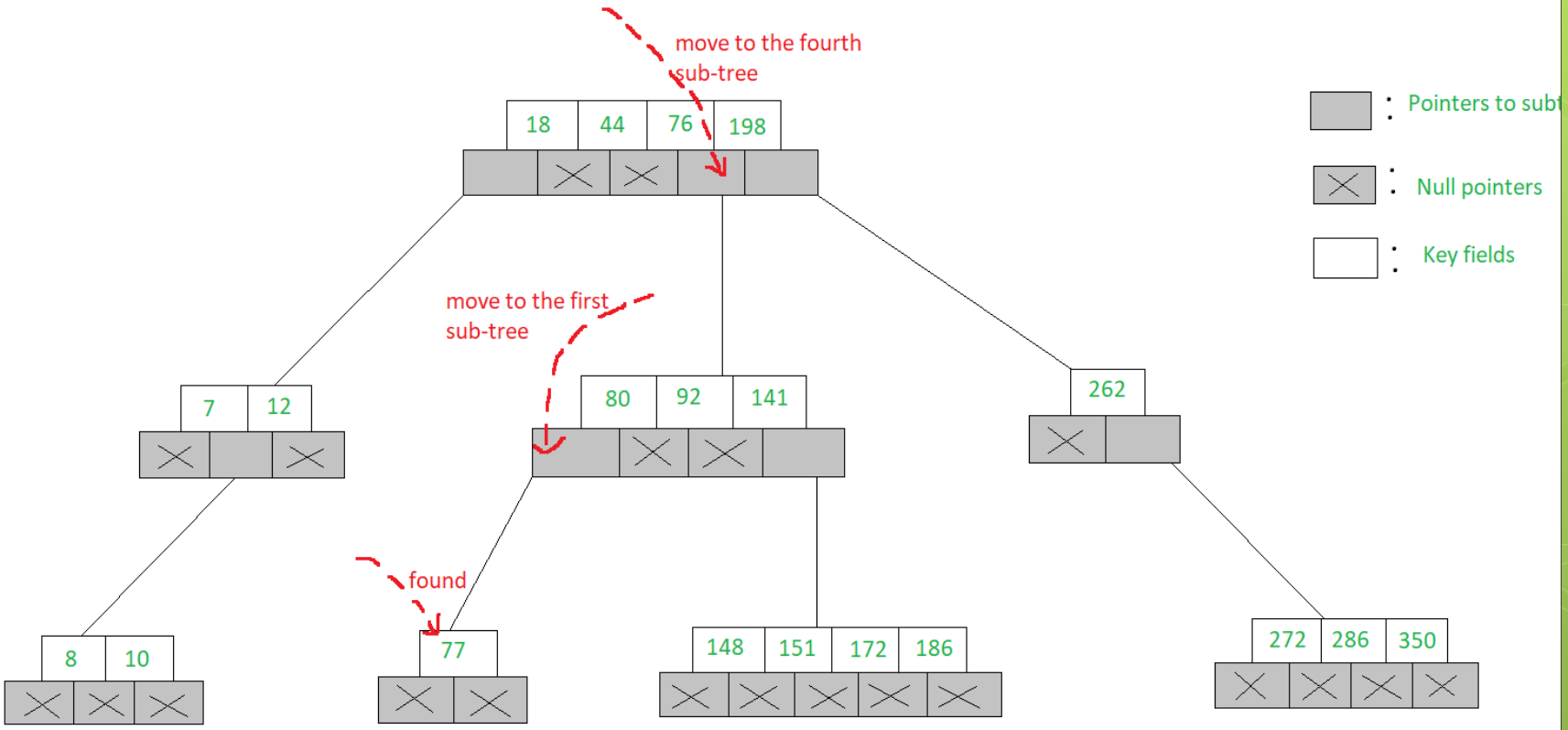
- Structure of a node of an m-Way tree

```
struct node {
        int count;
        int value[MAX];
        struct node* child[MAX + 1];
        };
```

- **count** represents the **number of children that a particular node has**
- The **values of a node** stored in the array **value**
- The **addresses of child** nodes are stored in the **child** array
- The **MAX** macro signifies the **maximum number of values that a particular node can contain**

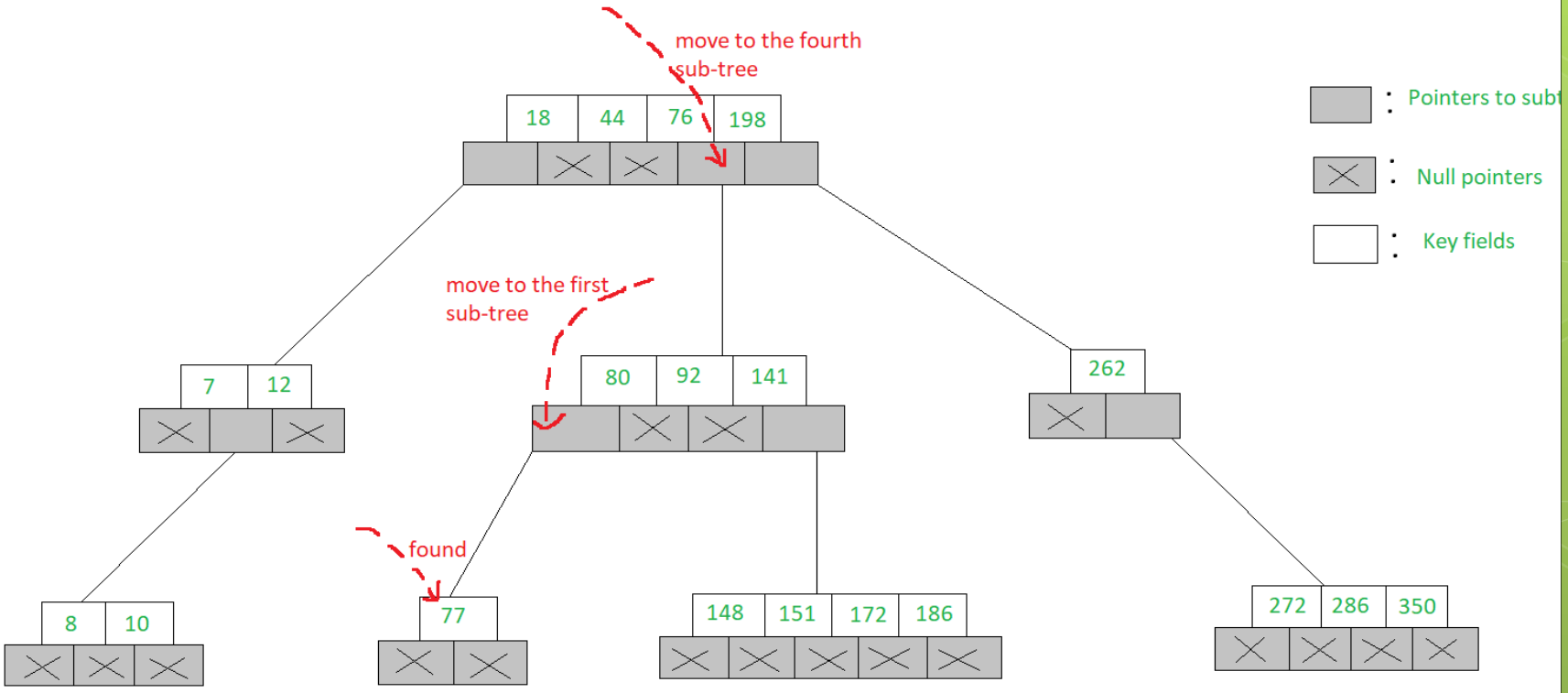
Searching in an m-Way search tree

- **Similar to that of binary search tree**
- To search for **77 in the 5-Way search tree,**



To search for 77

- **Begin at the root**
- As 77> 76> 44> 18, move to the **fourth sub-tree**
- In the root node of the fourth sub-tree, **77< 80** & therefore we move to the **first sub-tree of the node.**
- Since 77 is available in the only node of this sub-tree



11/6/2023

B Tree

B Tree

- A **balanced order n multiway search tree** in which each **non-root node contains atleast (n-1)/2 keys** is called B Tree of Order n

- **B Tree of O(n)**

- **Max no of keys in each node (root/non-root)=n-1**
- **Min** no of keys in **each non-root node=(n-1)/2**

B Tree

- **B Tree of O(5)**

- **Max no of keys in each node (root/non-root)=n-1=4**
- **Min** no of keys in **each non-root node=(n-1)/2=2**

Properties of B Tree

1) All leaf nodes will be at same level
2) Every node except root must contain at least ([n-1]/2) keys. The root may contain minimum 1 key.
3) All nodes (including root) may contain at most n – 1 keys.
4) Number of children of a node is equal to the number of keys in it plus 1.

Properties of B Tree

5) All keys of a node are sorted in increasing order.
6) The child between two keys k1 and k2 contains all keys in the range from k1 and k2.
7) B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.

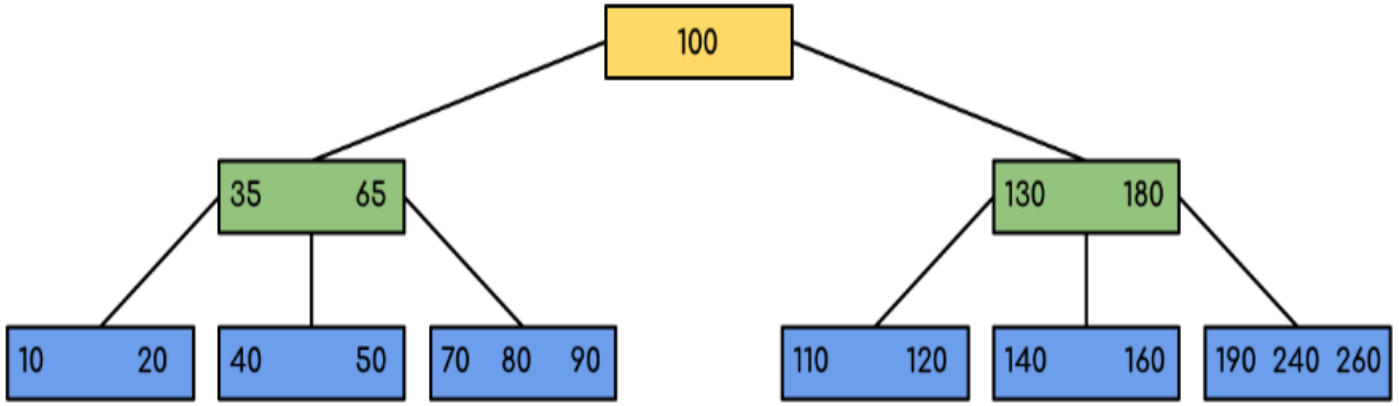
Properties of B Tree

1) The B stands for balanced,

2) In a B-tree the left and right side of each node is roughly kept to the same size (number of subnodes)

Example

1) B-Tree of order 5.
2) All the leaf nodes are at the same level



Animation for Searching in B Tree

- https://condor.depaul.edu/ichu/csc383/notes/notes7/B-Trees_files/tree-search.gif

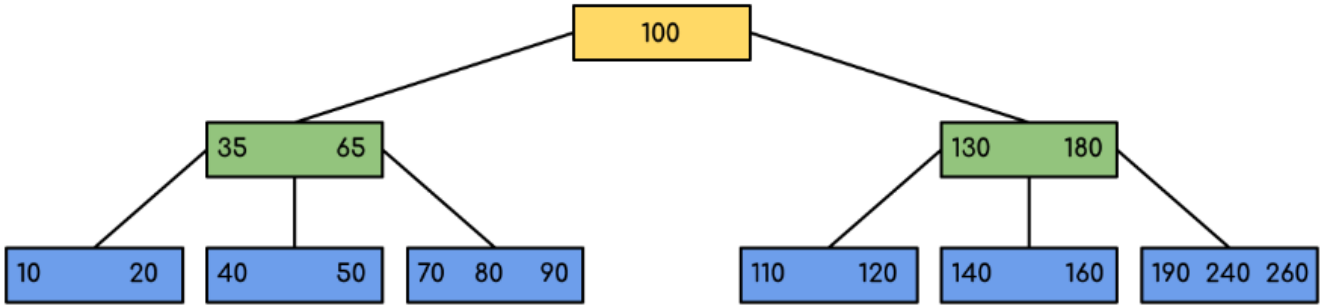
Courtesy: https://condor.depaul.edu/ichu/csc383/notes/notes7/btree.html

Searching in B Tree

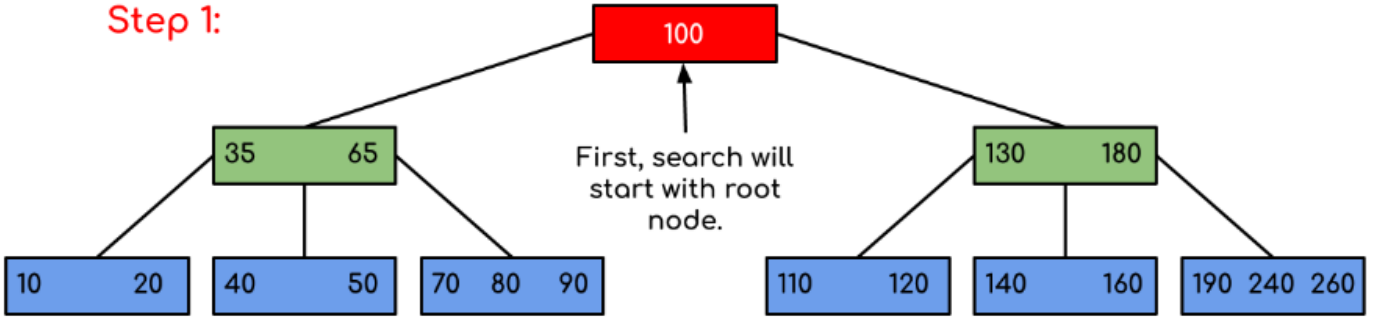
- Searching in B Trees is similar to that in Binary search tree.
  - Let the key to be searched be k.
  - Start from the root
  - Recursively traverse down.
  - For every visited non-leaf node, if the node has the key, we simply return the node.
  - Otherwise, we recur down to the appropriate child (The child which is just before the first greater key) of the node.
  - If we reach a leaf node and don't find k in the leaf node, we return NULL.

Searching in B Tree

Search for 120

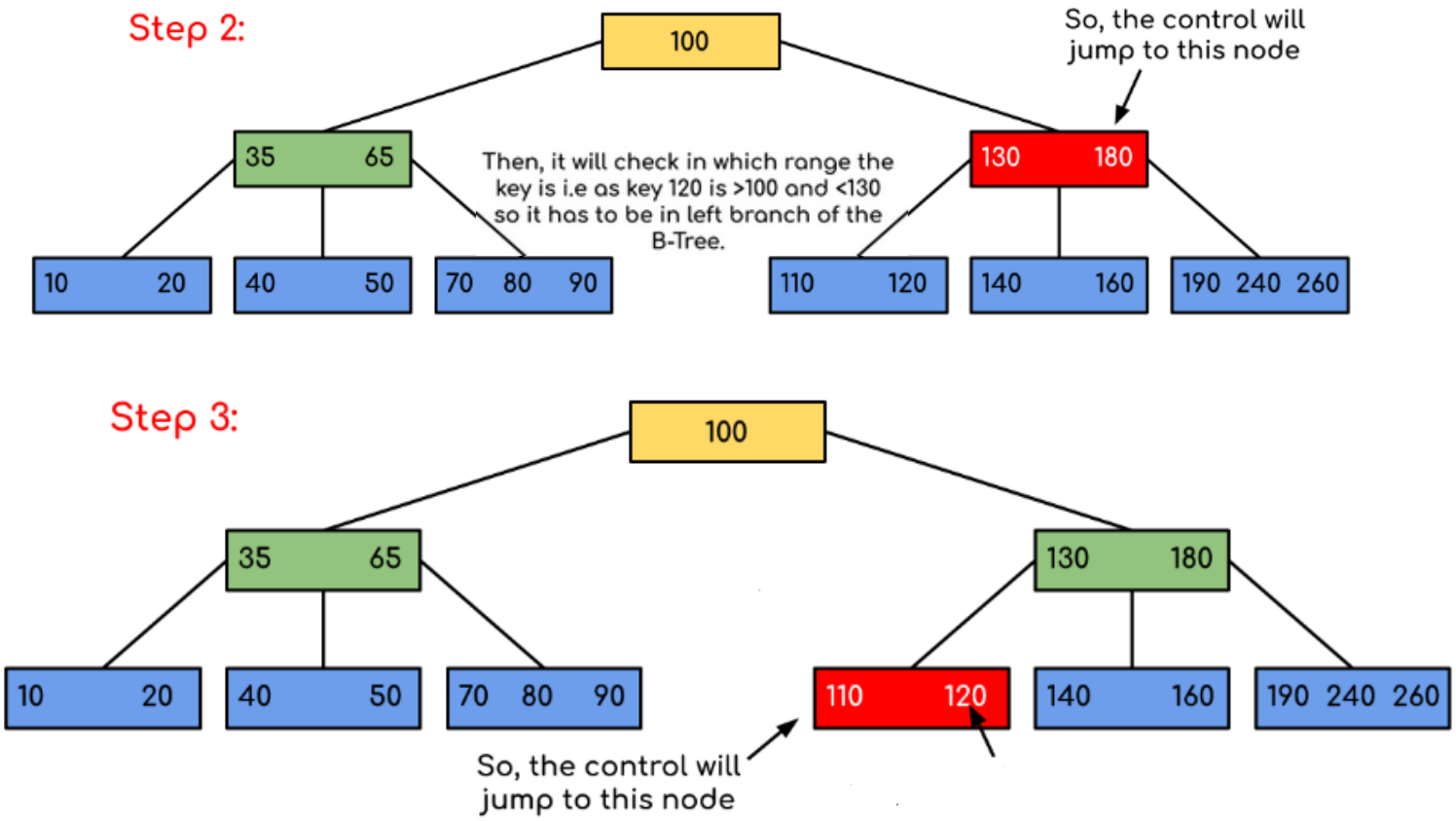


Step 1:



Searching in B Tree

Search for 120



11/6/2023

Insertion in B Tree

Insertion in B Tree

- Insertion requires first traversal in B Tree
- **Check key to be inserted is already existing or not**, through traversal
- Suppose the key does not exist in tree then through traversal , it will reach the leaf node
- **We will have 2 cases for inserting the keys**

Insertion in B Tree

The 2 cases are-

- **Case 1: Node is Not Full**
- **Case 2: Node is already full**

Insertion in B Tree

- **Case 1: Node is Not Full-**
  - We simply add the key in the Node
- **Case 2: Node is already full-**
  - **Split the Node in 2 nodes**
  - **Median key goes to the parent of that node**
  - **If parent is also full then same process will be repeated until it will get non-full parent node**

Algorithm for Insertion in B Tree

- The following algorithm applies:

1) Run the search operation and find the appropriate place of insertion.
2) **Insert the new key at the proper location, but if the node has a maximum number of keys** already:
3) **The node, along with a newly inserted key, will split from the middle element.**
4) The **middle element will become the parent** for the other two child nodes.
5) **The nodes must re-arrange keys in ascending order.**

Courtesy-https://www.guru99.com/b-tree-example.html

Algorithm for Insertion in B Tree

TIP

The following is not true about the insertion algorithm:

- Since the node is full, therefore it will split, and then a new value will be inserted

CORRECT METHOD-

- **The node, along with a newly inserted key, will split from the middle element.**

Courtesy-https://www.guru99.com/b-tree-example.html

11/6/2023

Insertion with Odd Order

Insertion in B Tree

- Create a B Tree of Order 5
- n=5
- List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160

- **Max no of keys in each node (root/non-root)=n-1=4**
- **Min no of keys in each non-root node=(n-1)/2=2**

Insertion in B Tree

- Create a B Tree of Order 5
- List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120, 140,160

Insertion in B Tree

- Create a B Tree of Order 5
- List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120, 140,160

Insertion in B Tree

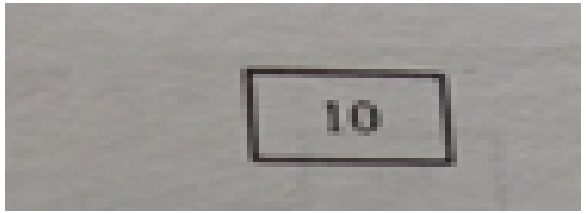
- Create a B Tree of Order 5
- List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120, 140,160

Insertion in B Tree

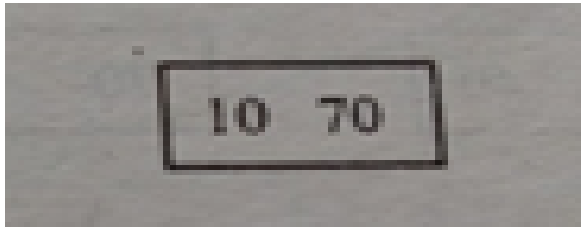
- Create a B Tree of Order 5
- List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120, 140,160

Insertion in B Tree

List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160

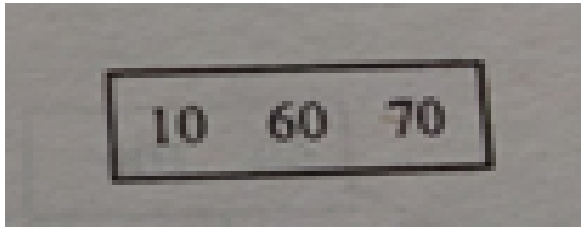


Insert 10



Insert 70

After Inserting 70, the keys in the node will be sorted

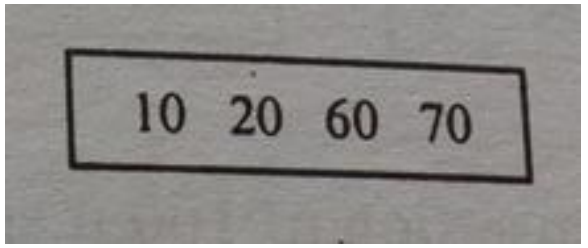


Insert 60

After Inserting 60, the keys in the node will be sorted

Insertion in B Tree

List of Keys=10,70,60,20,110,40,80,130, 100,50,190,90,180,240,30,120,140 ,160



Insert 20

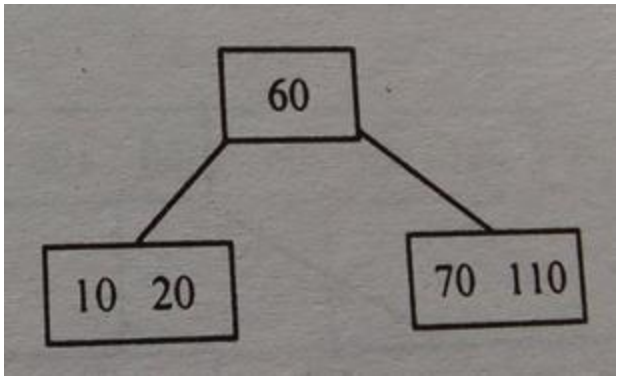
After Inserting 20, the keys in the node will be sorted

Insert 110

Node was already full,

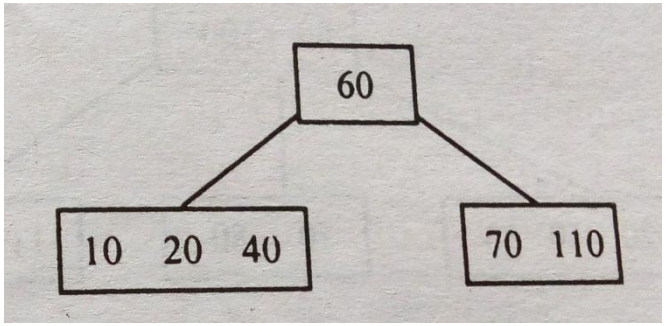
After insertion of 110 , It splits into 2 nodes

60 is the median key, so it goes to parent or becomes root



Insertion in B Tree

List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160

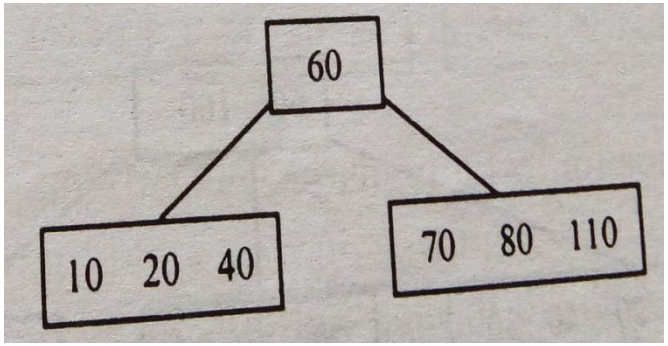


Insert 40

After Inserting 40, the keys in the node will be sorted

Insert 80

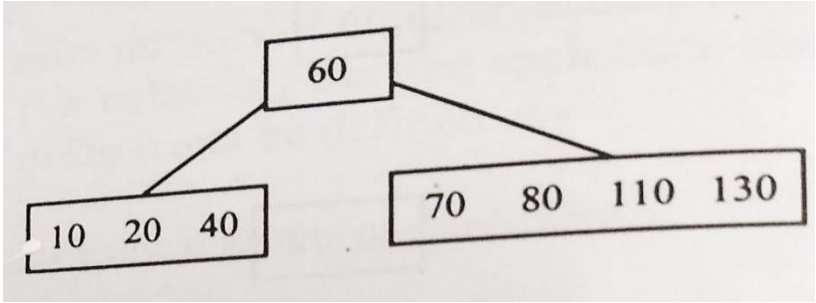
After Inserting 80, the keys in the node will be sorted



Insertion in B Tree

List of Keys=10,70,60,20,110,40,80,130, 100,50,190,90,180,240,30,120,140 ,160

Insert 130

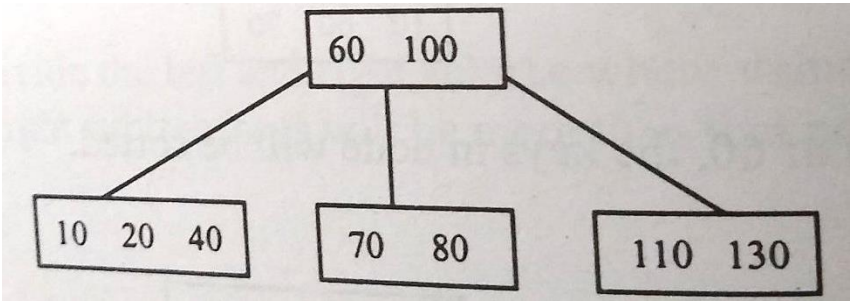


Insert 100

Node was already full

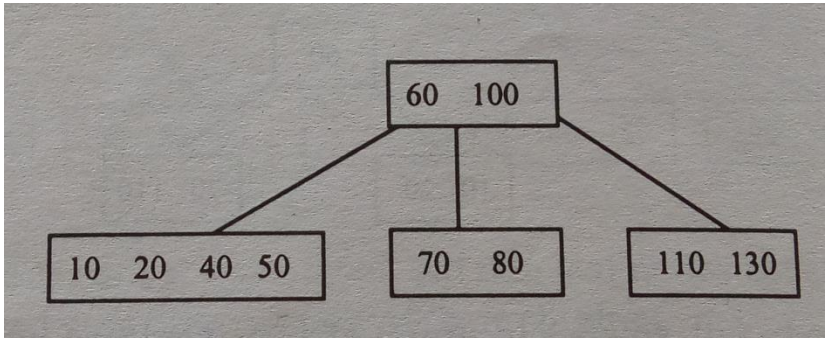
After insertion of 100, it splits in 2 nodes, 100 is the median key ,

100 goes up to the parent node



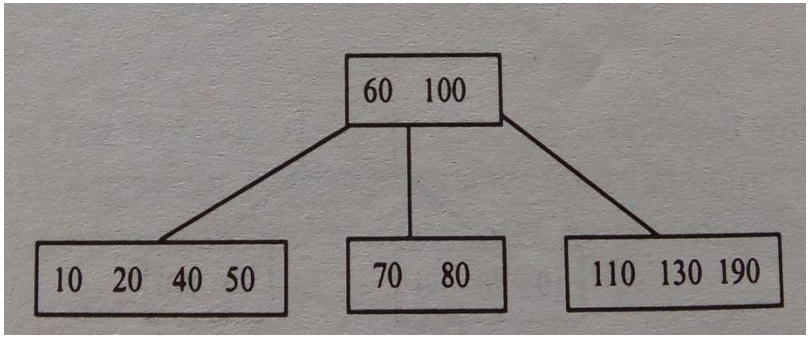
Insertion in B Tree

List of Keys=10,70,60,20,110,40,80, 130,100,50,190,90,180,240,30,120,140,160



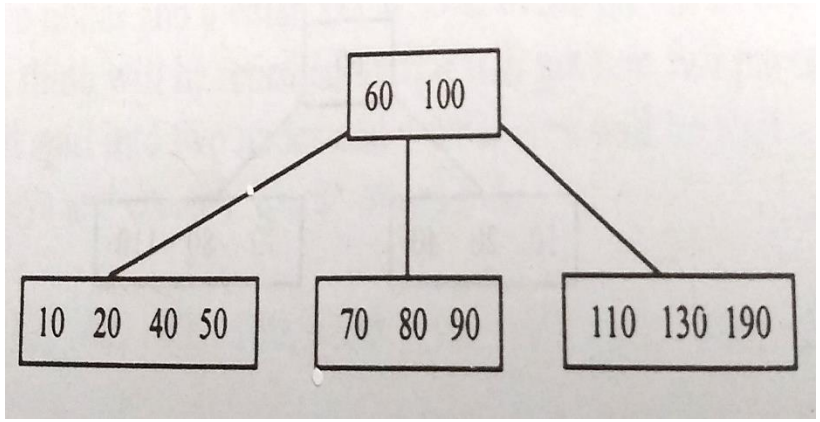
Insert 50

Insert 190



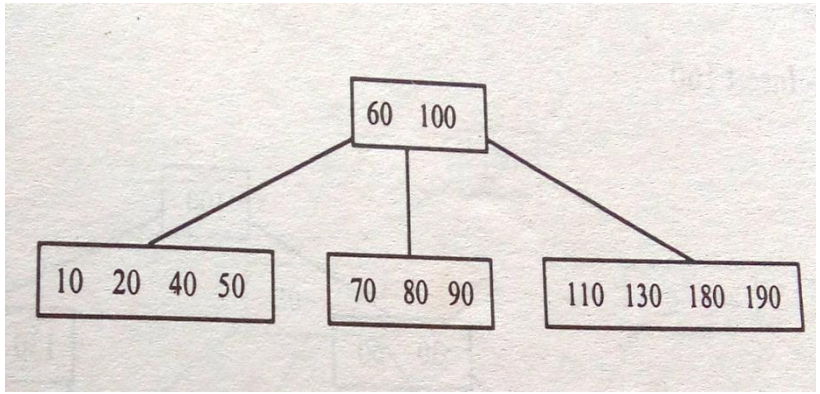
Insertion in B Tree

List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160



Insert 90

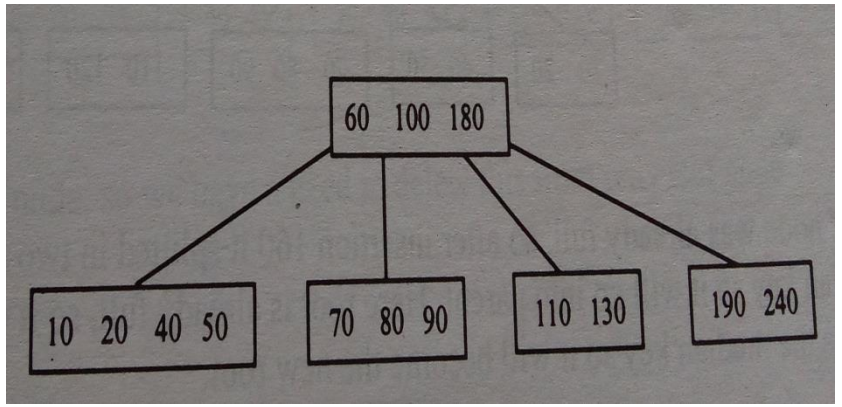
Insert 180



Insertion in B Tree

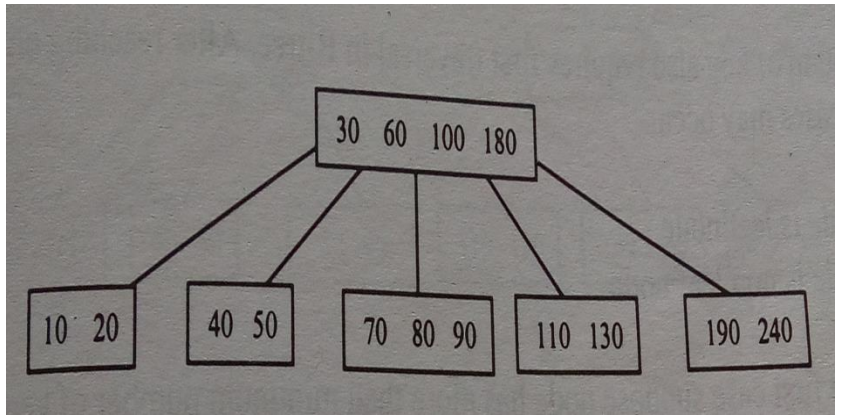
List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160

Insert 240



Insert 30

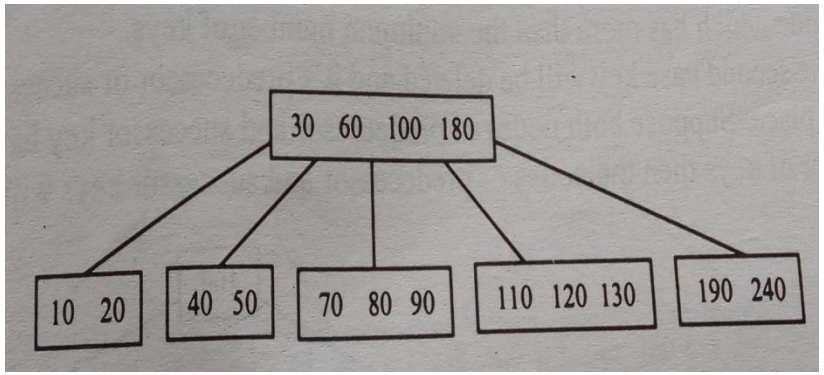
Node was already full, so after insertion of 30, splits in 2 nodes, 30 is the median key so it will go to the parent



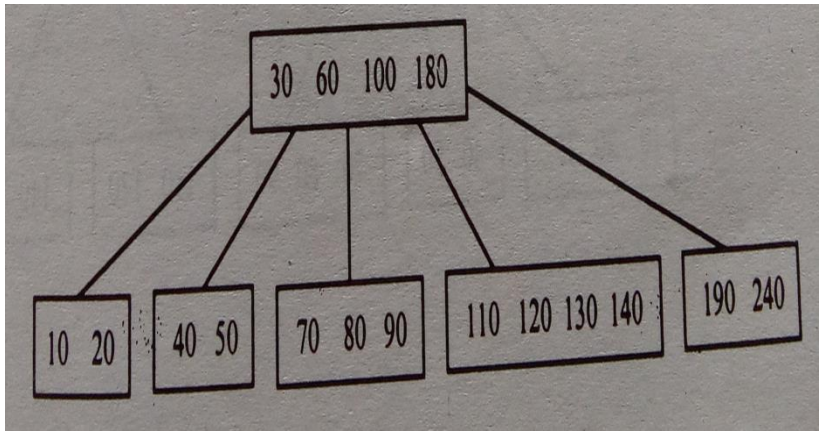
Insertion in B Tree

List of Keys=10,70,60,20,110,40,80 ,130,100,50,190,90,180,240, 30,120,140,160

Insert 120



Insert 140



Insertion in B Tree

List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160

Insert 160

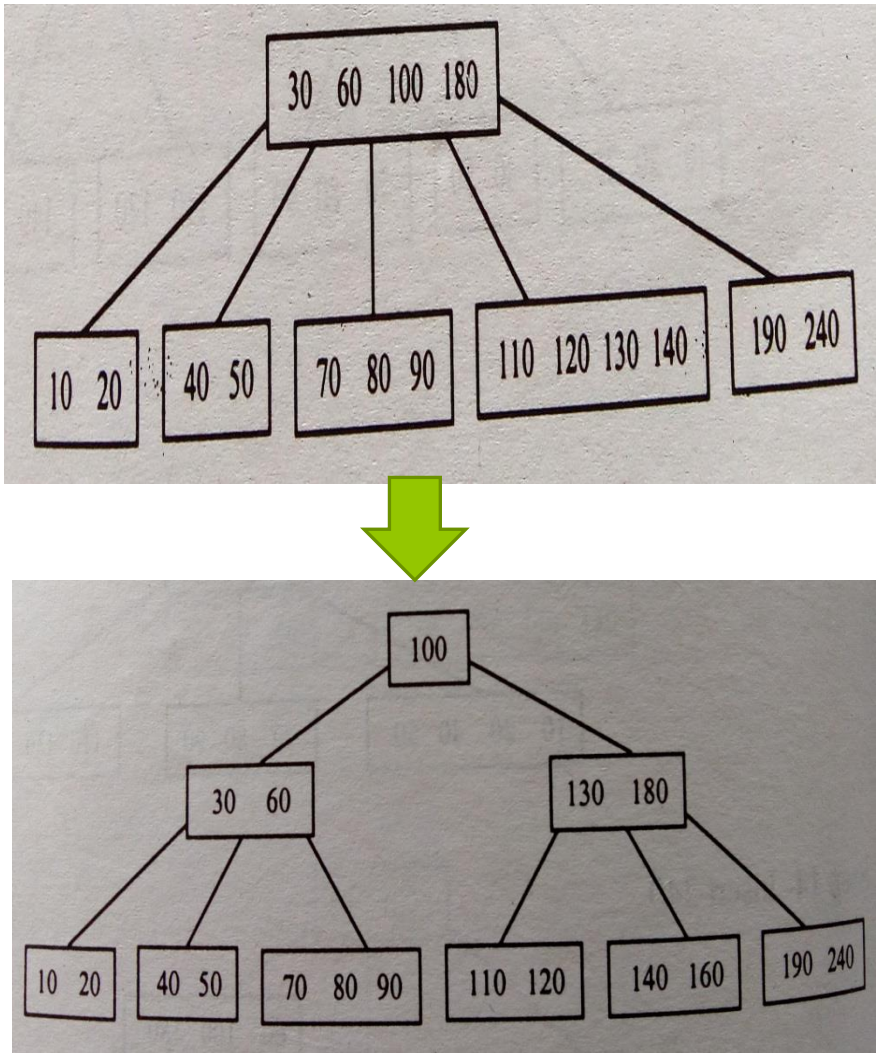
Node was already full,

After insertion of 160

Splits into 2 nodes

130 is the median so it goes up

Root is already full, so it splits in 2 nodes , 100 is the median so it becomes new root



Insertion in B Tree

- Create a B Tree of Order 5
- n=5
- List of Keys=10,70,60,20,110,40,80,130,100,50,190,90,180,240,30,120,140,160

- **Max no of keys in each node (root/non-root)=n-1=4**
- **Min no of keys in each non-root node=(n-1)/2=2**

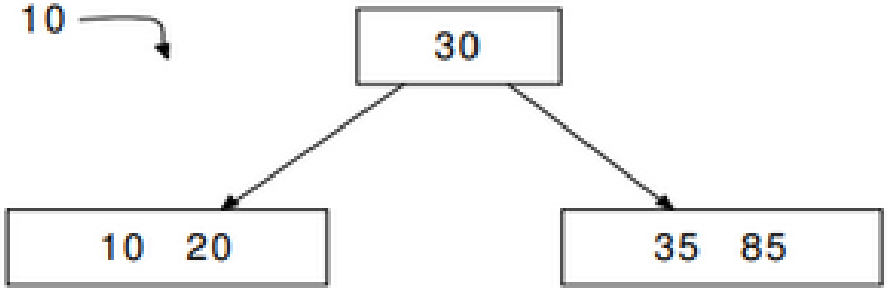
- Create B-Tree of order 5 from the following list of data items: 20, 30, 35, 85, 10, 55, 60, 25,81,18,22

- **Create B-Tree of order 5 from the following list of data items: 20, 30, 35, 85, 10, 55, 60, 25,81,18,22**

Step 1:

Insert 20, 30, 35 and 85

Step 2:

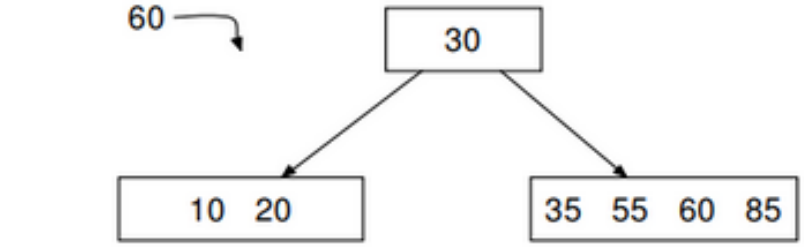


- Create B-Tree of order 5 from the following list of data items: 20, 30, 35, 85, 10, 55, 60, 25,81,18,22

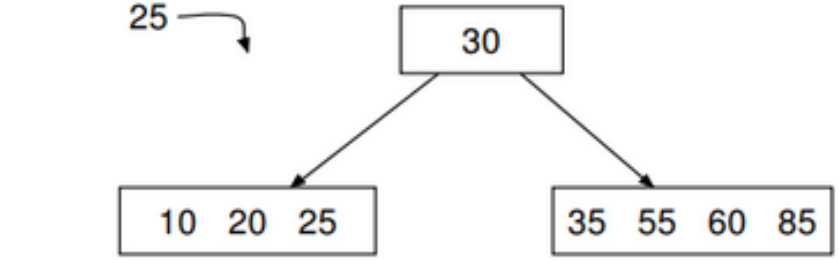
Step 3:

Insert 55

Step 4:



Step 5:



11/6/2023

Insertion with Even Order

Insertion in B Tree

- In case of **even number of keys,**
- The middle node will be selected by
  - **Left bias or**
  - **Right bias**

Insertion in B Tree

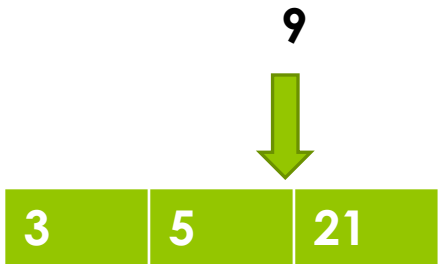
- Order =4
- List=5,3,21,9,1,13,2,7

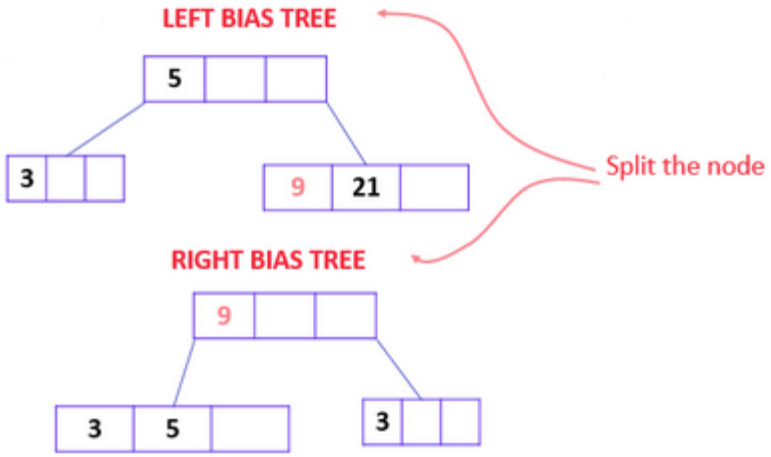
Insertion in B Tree

- Order =4
- List=5,3,21,9,1,13,2,7

- Insert 9
- Even number of keys,

- The middle node will be selected by
  - **Left bias or**
  - **Right bias**

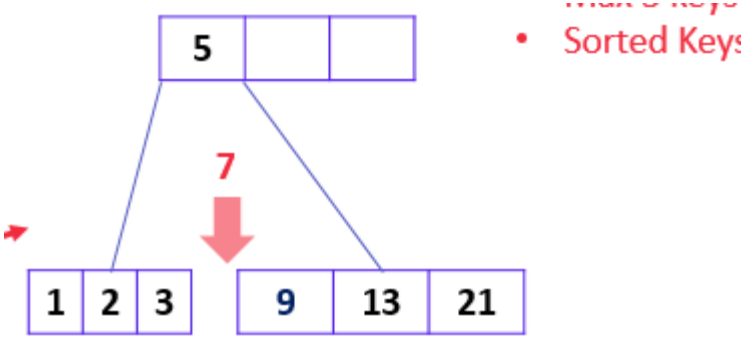


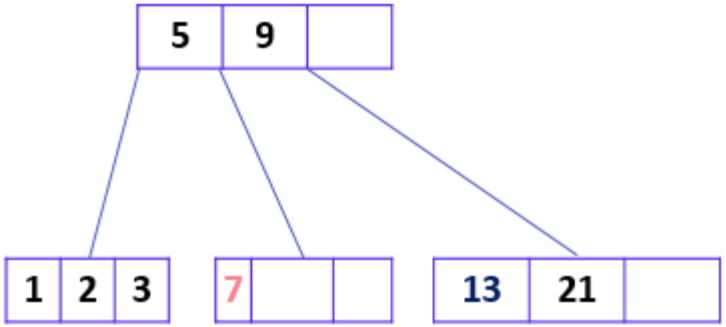


Insertion in B Tree

- Order =4
- List=5,3,21,9,1,13,2,7

- Middle key by **Left bias**





Splitting in B Tree

List=10, 20, 30, 40, 50, 60, 70, 80 and 90

Insert in an initially empty B-Tree of **Order 6**

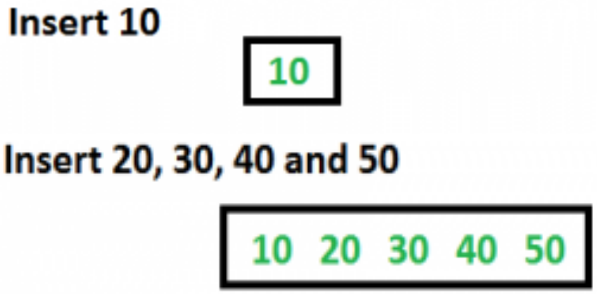
n=6

- **Max no of keys in each node (root/non-root)=n-1=5**
- **Min no of keys in each non-root node=(n-1)/2**
  =(6-1)/2
  =5/2
  =2

Splitting in B Tree

List=10, 20, 30, 40, 50, 60, 70, 80 and 90

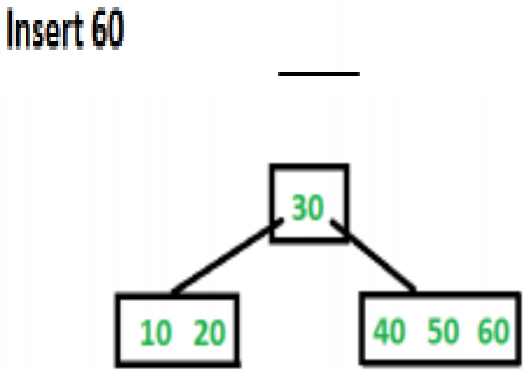
Insert 10



Insert 20,30,40

Insert 60

1) Since root node is full,
2) **Node will split into two nodes**
3) **Median=30, using Left Bias so 30 goes to parent or becomes root,**



11/6/2023

Uses/Application of B Tree

Use of B Tree

1) B-trees are balanced search trees designed to work well **on magnetic disks or other direct-access secondary storage devices**
2) Better at **minimizing disk I/O operations**
3) There is huge amount of data that cannot fit in main memory.
   - When the number of keys is high, **the data is read from disk in the form of blocks.**
   - **Disk access time is very high compared to the main memory access time.**