Advanced Algorithms & Data Structures

AIM OF THE SESSION

To familiarize students with the basic concept of B Tree

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Demonstrate The B Tree.
2. Describe the types of B Tree traversals.
3. Constructing B Tree and traversals.
4. Constructing an expression tree.
5. List out the advantages and applications of B Tree.

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Define B Tree
2. Describe the various traversals in B Tree Expression tree construction
3. Summarize definition, types and operations of B Tree and its applications

Deletion from a B-tree

- Deleting an element on a B-tree consists of three main events: **searching the node where the key to be deleted exists**, deleting the key and balancing the tree if required.
- While deleting a tree, a condition called **underflow** may occur. Underflow occurs when a node contains less than the minimum number of keys it should hold.
- The terms to be understood before studying deletion operation are:
    - **Inorder Predecessor**

      The largest key on the left child of a node is called its inorder predecessor.
    - **Inorder Successor**

      The smallest key on the right child of a node is called its inorder successor.

Deletion Operation

Before going through the steps below, one must know these facts about a B tree of degree **m.**

1. A node can have a maximum of m children. (i.e. 3)
2. A node can contain a maximum of `m - 1` keys. (i.e. 2)
3. A node should have a minimum of `⌈m/2⌉` children. (i.e. 2)
4. A node (except root node) should contain a minimum of `⌈m/2⌉ - 1` keys. (i.e. 1)

There are three main cases for deletion operation in a B tree.

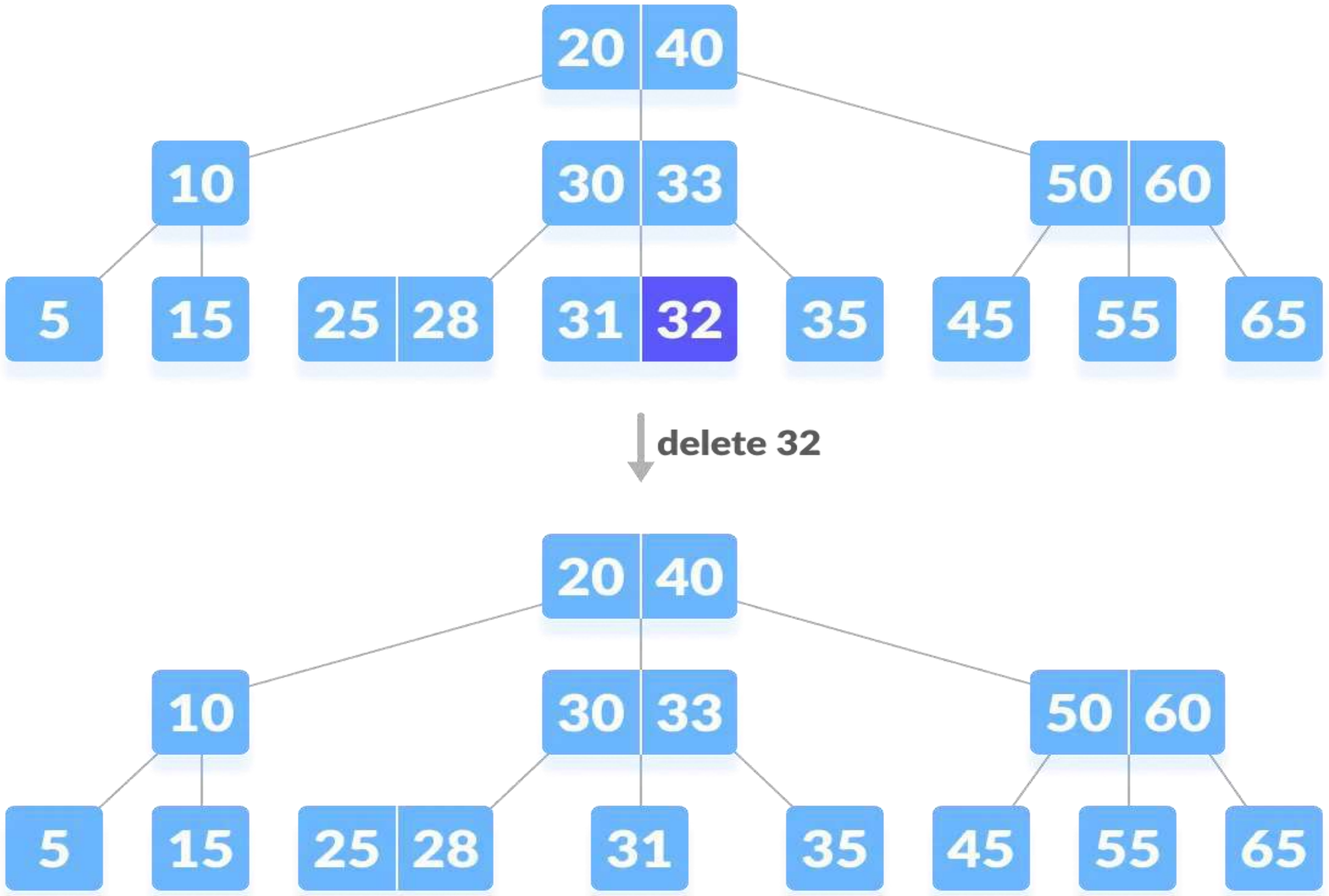
Case I

The key to be deleted lies in the leaf. There are two cases for it.

1. The deletion of the key does not violate the property of the minimum number of keys a node should hold.

   In the tree below, deleting 32 does not violate the above properties.

Deleting a leaf key (32) from B-tree



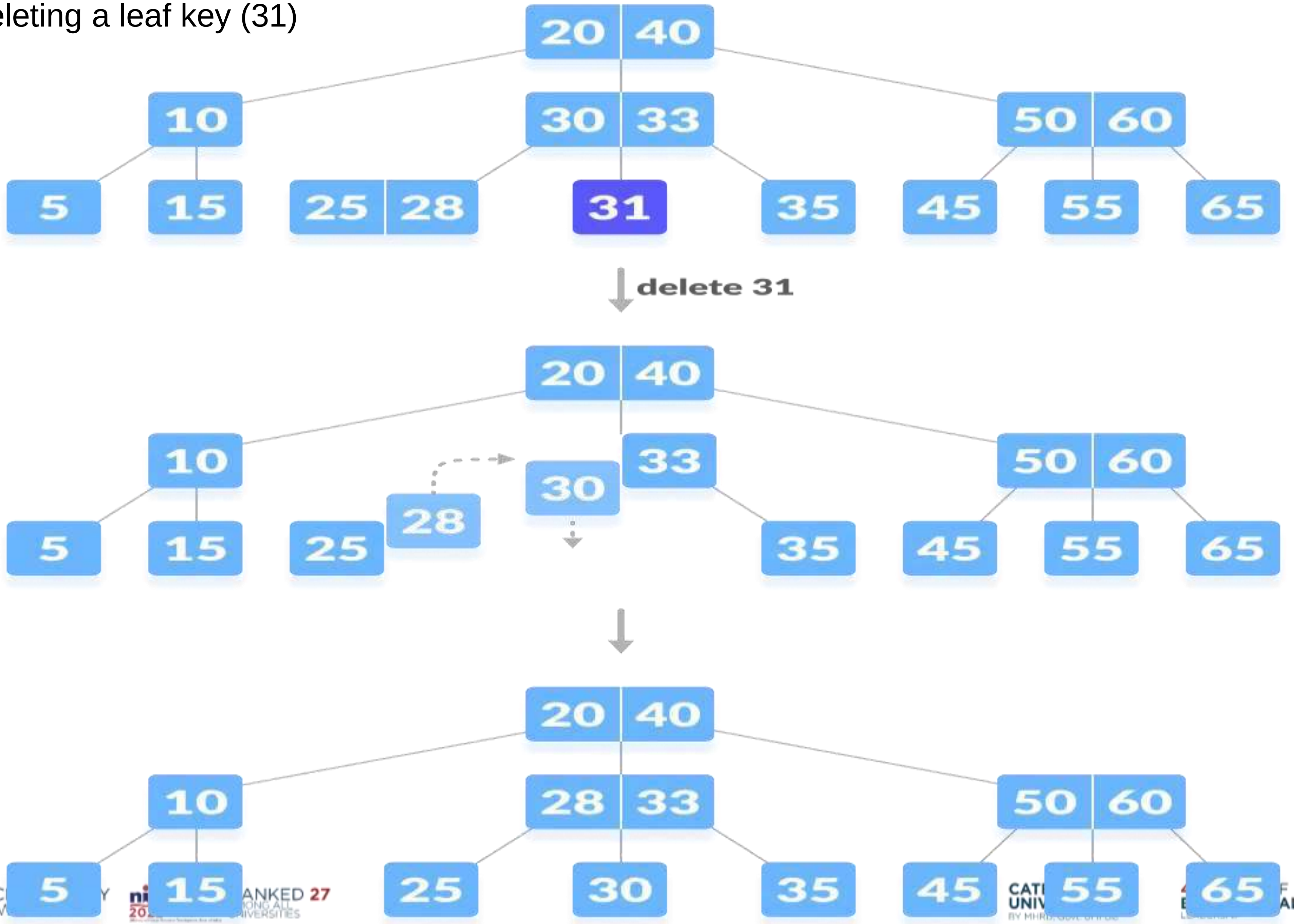
2. The deletion of the key violates the property of the minimum number of keys a node should hold. In this case, we borrow a key from its immediate neighboring sibling node in the order of left to right.

   First, visit the immediate left sibling. If the left sibling node has more than a minimum number of keys, then borrow a key from this node.

   Else, check to borrow from the immediate right sibling node.

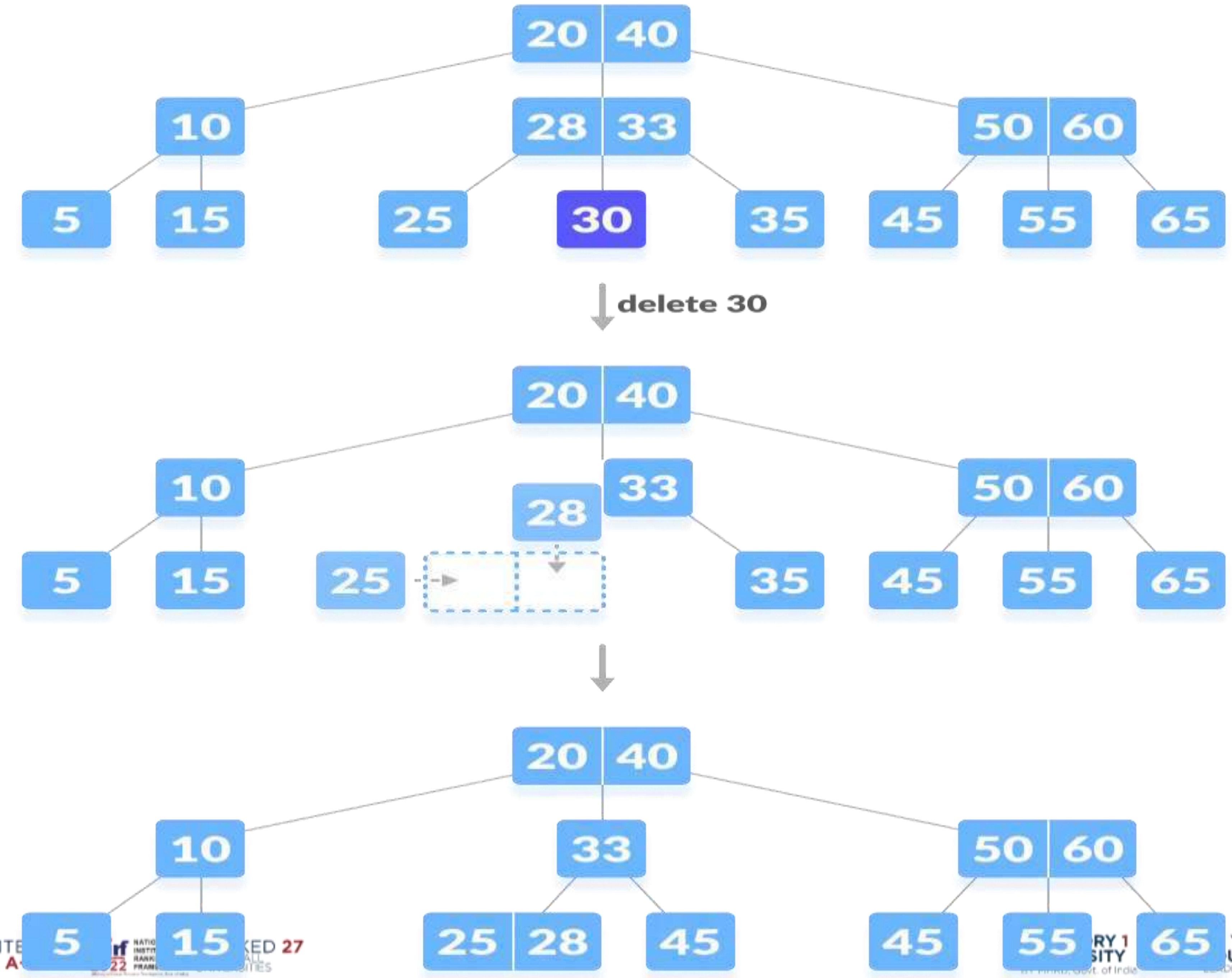
   In the tree below, deleting 31 results in the above condition. Let us borrow a key from the left sibling node.

Deleting a leaf key (31)



If both the immediate sibling nodes already have a minimum number of keys, then merge the node with either the left sibling node or the right sibling node. **This merging is done through the parent node.**

Deleting 30 results in the above case.

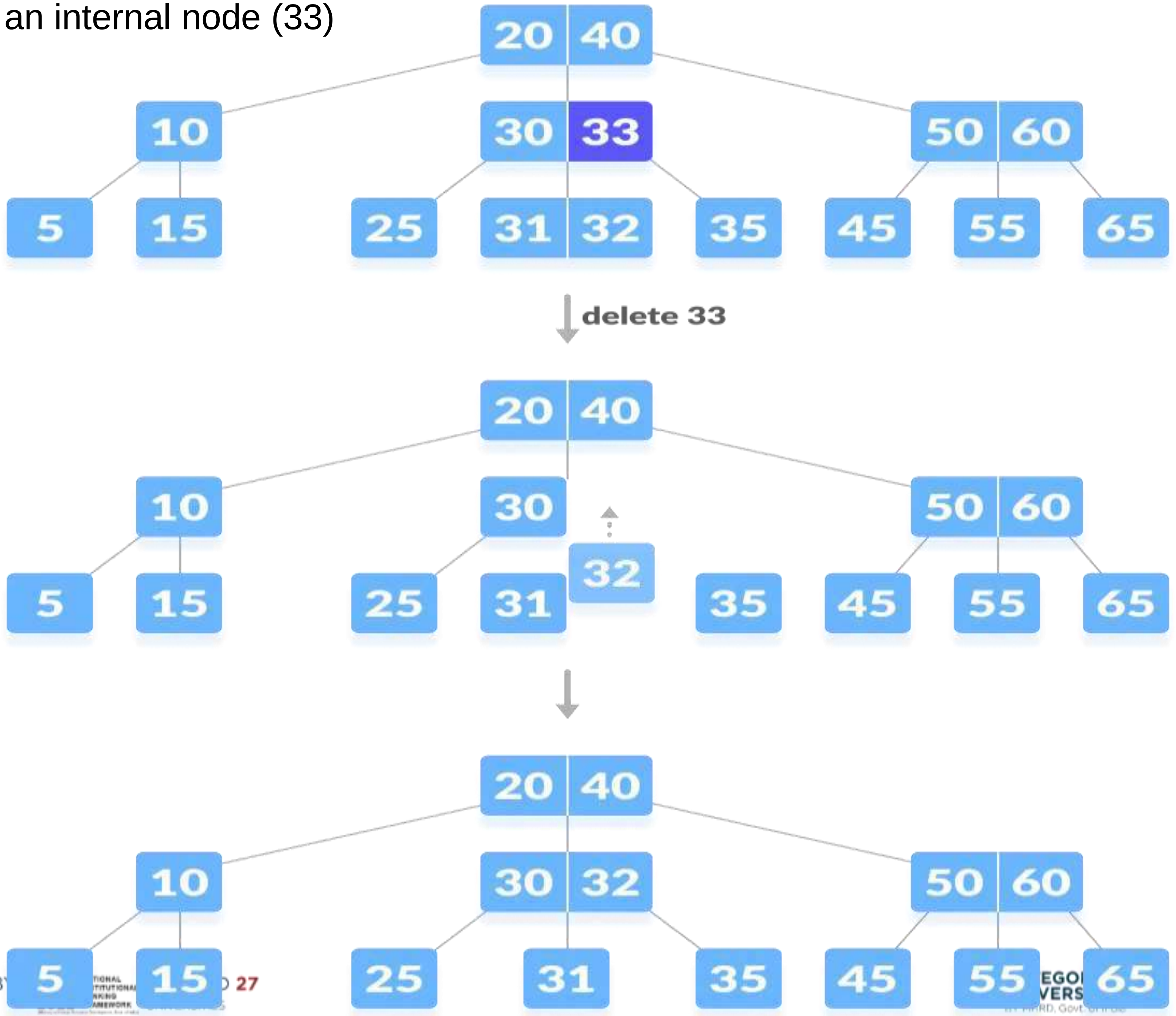

20
40
10
28
33
50
60
5
15
25
30
35
45
55
65
delete 30
20
40
10
33
28
50
60
5
15
25
35
45
55
65
20
40
10
33
50
60
5
15
25
28
45
45
55
65

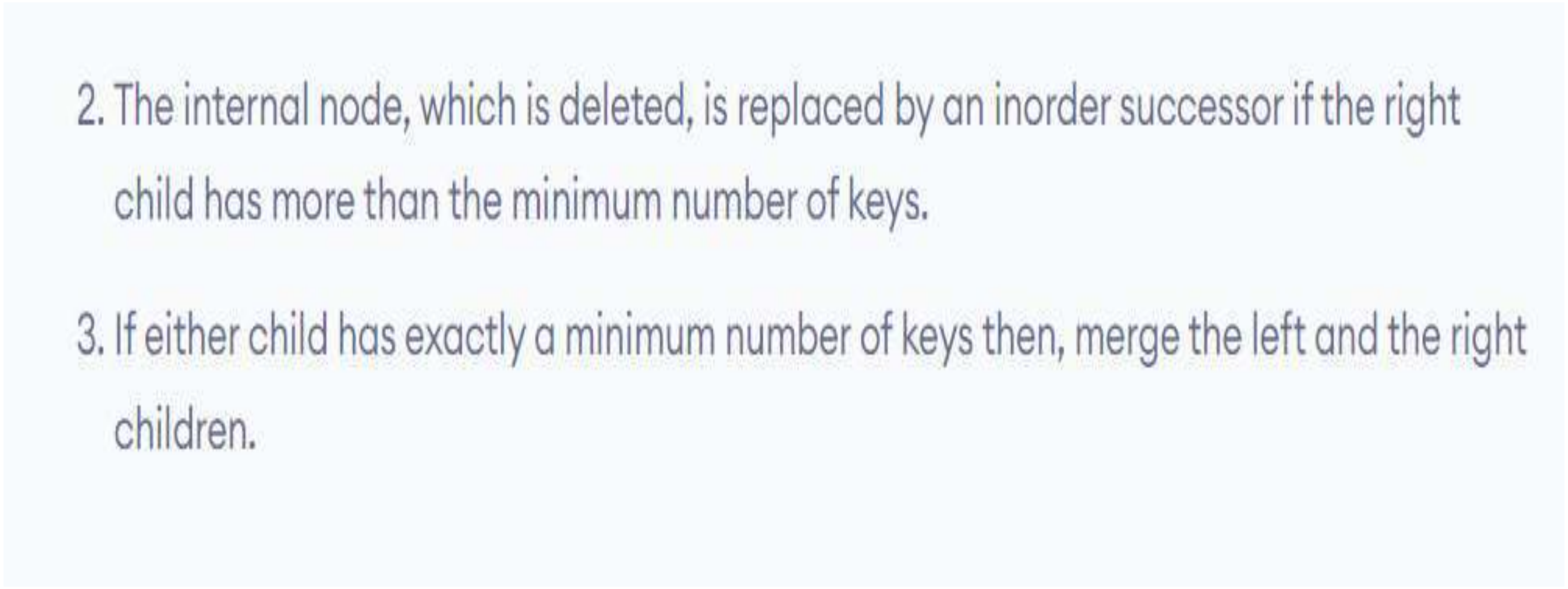
Case II

If the key to be deleted lies in the internal node, the following cases occur.

1. The internal node, which is deleted, is replaced by an inorder predecessor if the left child has more than the minimum number of keys.

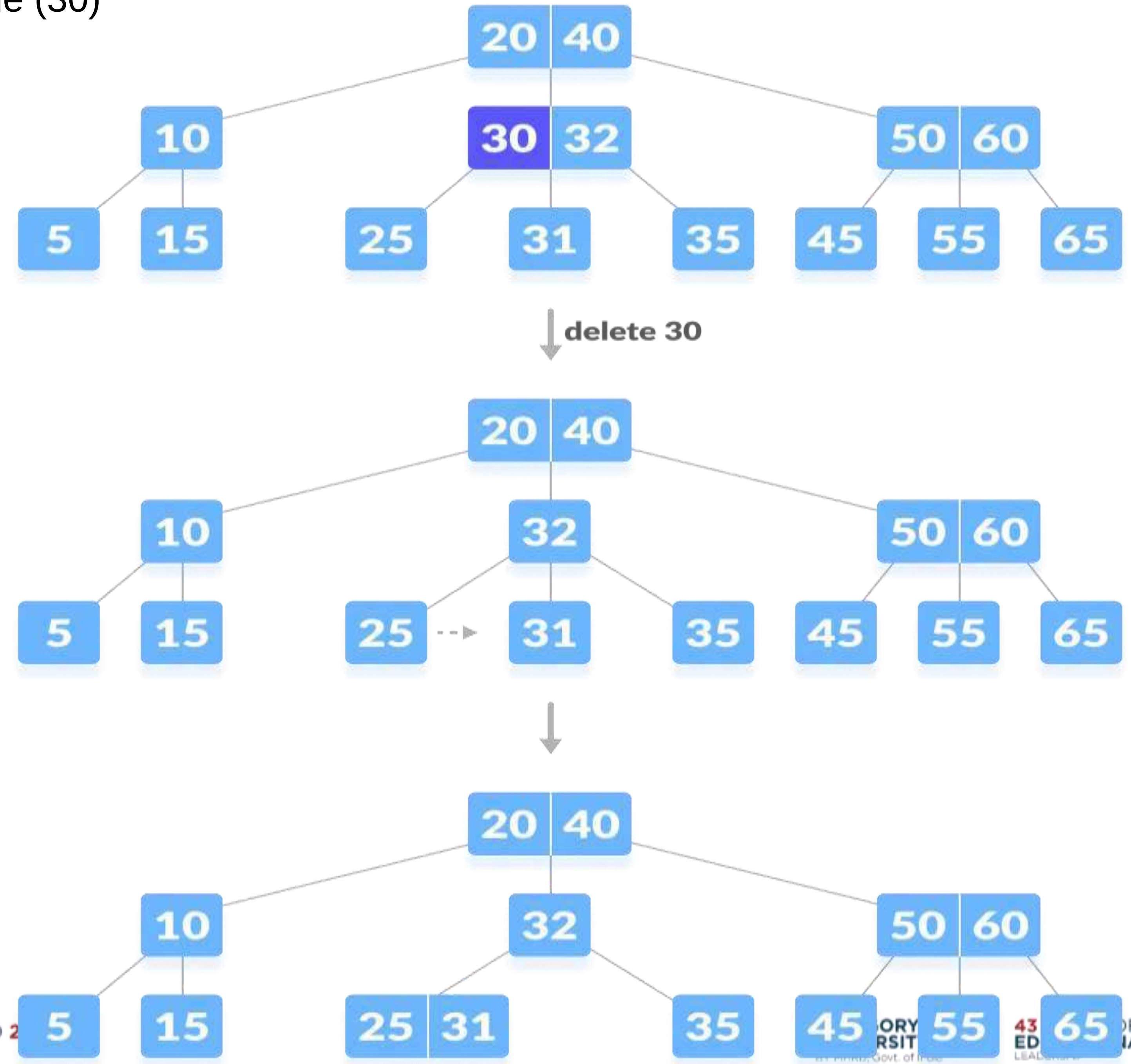
Deleting an internal node (33)





2. The internal node, which is deleted, is replaced by an inorder successor if the right child has more than the minimum number of keys.
3. If either child has exactly a minimum number of keys then, merge the left and the right children.

Deleting an internal node (30)

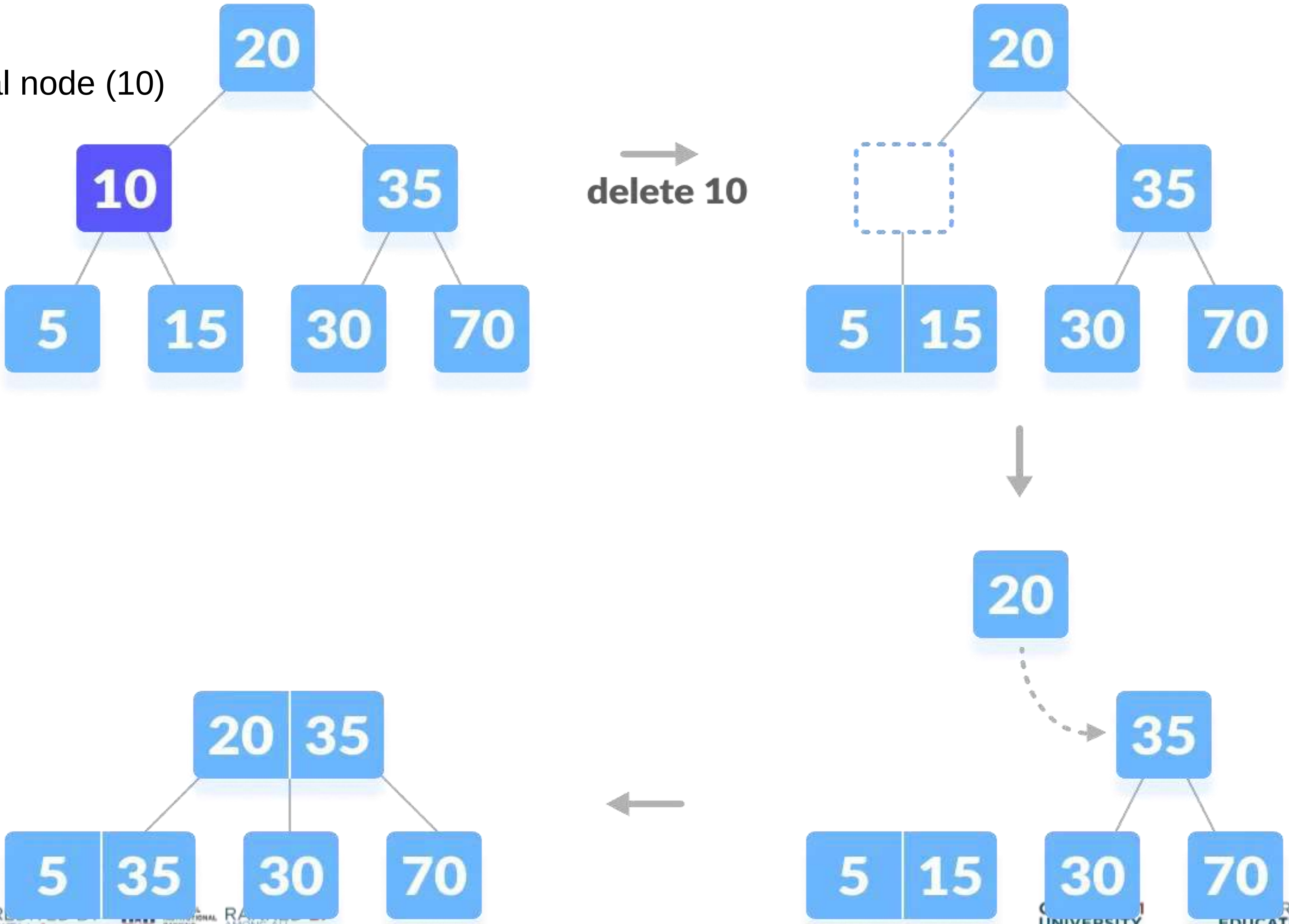


- After merging if the parent node has less than the minimum number of keys then, look for the siblings.

In this case, the height of the tree shrinks. If the target key lies in an internal node, and the deletion of the key leads to a fewer number of keys in the node (i.e. less than the minimum required), then look for the inorder predecessor and the inorder successor. If both the children contain a minimum number of keys then, borrowing cannot take place. This leads to Case II(3) i.e. merging the children.

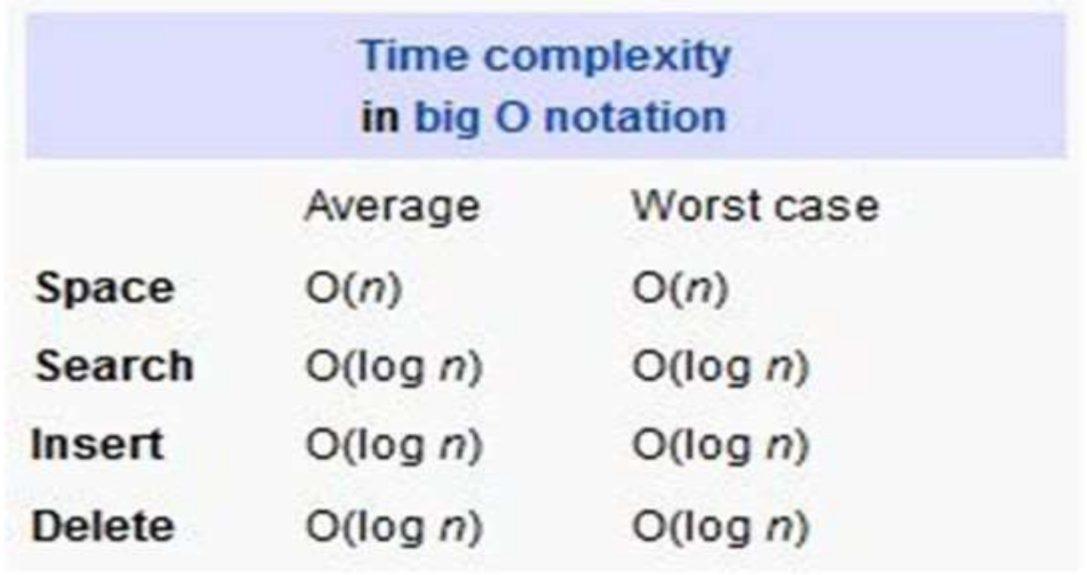
Again, look for the sibling to borrow a key. But, if the sibling also has only a minimum number of keys then, merge the node with the sibling along with the parent. Arrange the children accordingly (increasing order).

Deleting an internal node (10)



Analysis of B-Trees

- Time Complexity:



Time complexity in big O notation		
	Average	Worst case
Space	O(*n*)	O(*n*)
Search	O(log *n*)	O(log *n*)
Insert	O(log *n*)	O(log *n*)
Delete	O(log *n*)	O(log *n*)

IMPORTANT FACTS RELATED TO THE SESSION

Advantages of a B Trees

- B Trees are conceptually simple and easy to implement.
- B Trees provide efficient search and retrieval operations.
- Traversing a B Tree is a well-defined process, and various algorithms (in-order, pre-order, post-order) allow for easy navigation and processing of all elements in the tree.

Disadvantages of B Trees

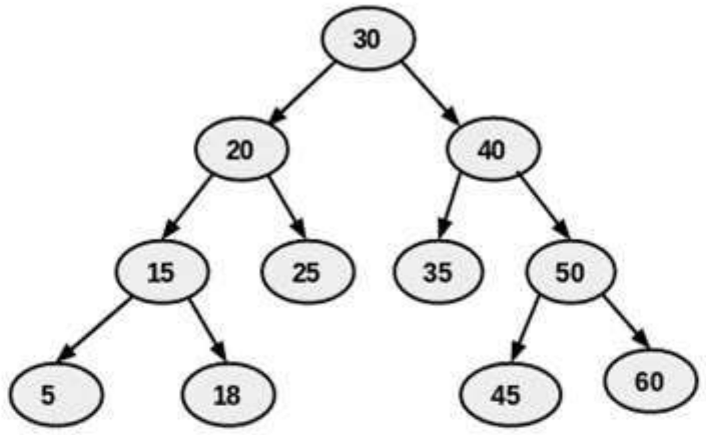
- B Trees can have higher memory overhead compared to other data structures.
- Although B Trees are conceptually simple, ensuring proper balance and implementing self-balancing mechanisms can add complexity to the code.
- The performance of a binary search tree can be sensitive to the order in which elements are inserted.

Applications/Uses of B Trees in real life

- File systems in computers often use B Trees to represent the hierarchical structure of directories and file.
- B Trees can represent organizational hierarchies in business structures
- B Trees are used to represent mathematical expressions in compilers and evaluators

EXAMPLES

1.Write Inorder, Preorder, Post Order to below tree



2. Draw an Expression Tree to A * B + C / D.

SUMMARY

- Tree data structure is a collection of data (Node) which is organized in hierarchical structure recursively
- A tree in which every node can have a maximum of two children is called B Tree.
- Several terminologies are used in tree like follows..
  root, parent, child, height, level, deapth and subtree etc….
- In a B Tree, we perform the following traversals
  i) Inorder ii) Preorder iii) post order

- An expression tree is a special type of B Tree that is used to store algebraic expressions. In an expression tree, each internal node corresponds to the operator and each leaf node corresponds to the operand.

- Advantages, Disadvantages and Applications/Uses of B Trees in real life.

SELF-ASSESSMENT QUESTIONS

What is a tree in data structures?

a. A hierarchical data structure

b. A linear data structure

c. A non-linear data structure

d. A sequential data structure

In a B Tree, each node has at most how many children?

a. 1
b. 2
c. 3
d. 4

TERMINAL QUESTIONS

1. Define B Tree?

2. Define the height of a B Tree?

3. What is an in-order traversal of a B Tree?

3. Define a leaf node in the context of a B Tree?

4. Define the term "B Tree traversal.“?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. "Mark Allen Weiss, Data Structures and Algorithm Analysis in C, 2010 , Second Edition, Pearson Education.
2. 2. Ellis Horowitz, Fundamentals of Data Structures in C: Second Edition, 2015.
3. A.V.Aho, J. E. Hopcroft, and J. D. Ullman, "Data Structures And Algorithms", Pearson Education, First Edition Reprint 2003.

4. Sites and Web links:

1. https://nptel.ac.in/courses/106102064
2. https://in.udacity.com/course/intro-to-algorithms--cs215
3. https://www.coursera.org/learn/data-structures?action=enroll

THANK YOU