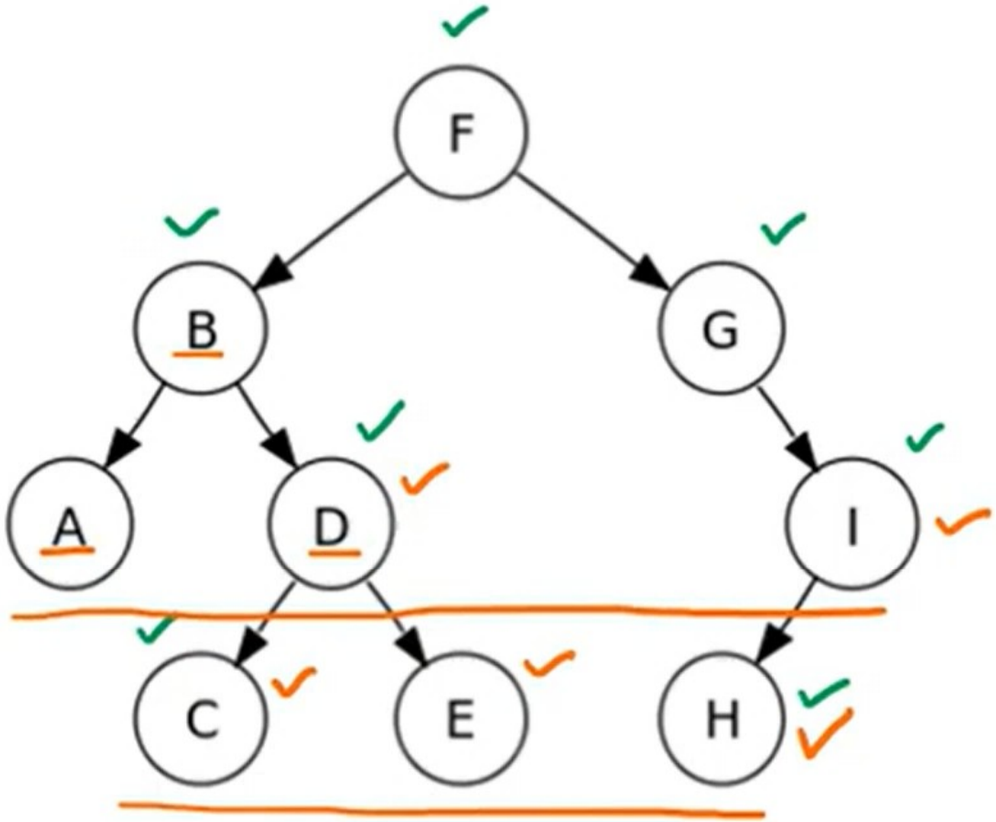
Tree

- Tree is a non linear data structure
- Represent data containing hierarchical relationship between elements



Types of Tree

Binary Tree

- Complete Binary Tree
- Extended Binary Tree (2-Tree)

Binary Search Tree (BST)

AVL Tree

m-Way Tree

B-Tree

B^+-Tree

Binary Tree

- Binary Tree is non linear data structure with maximum two children for each parent

Root

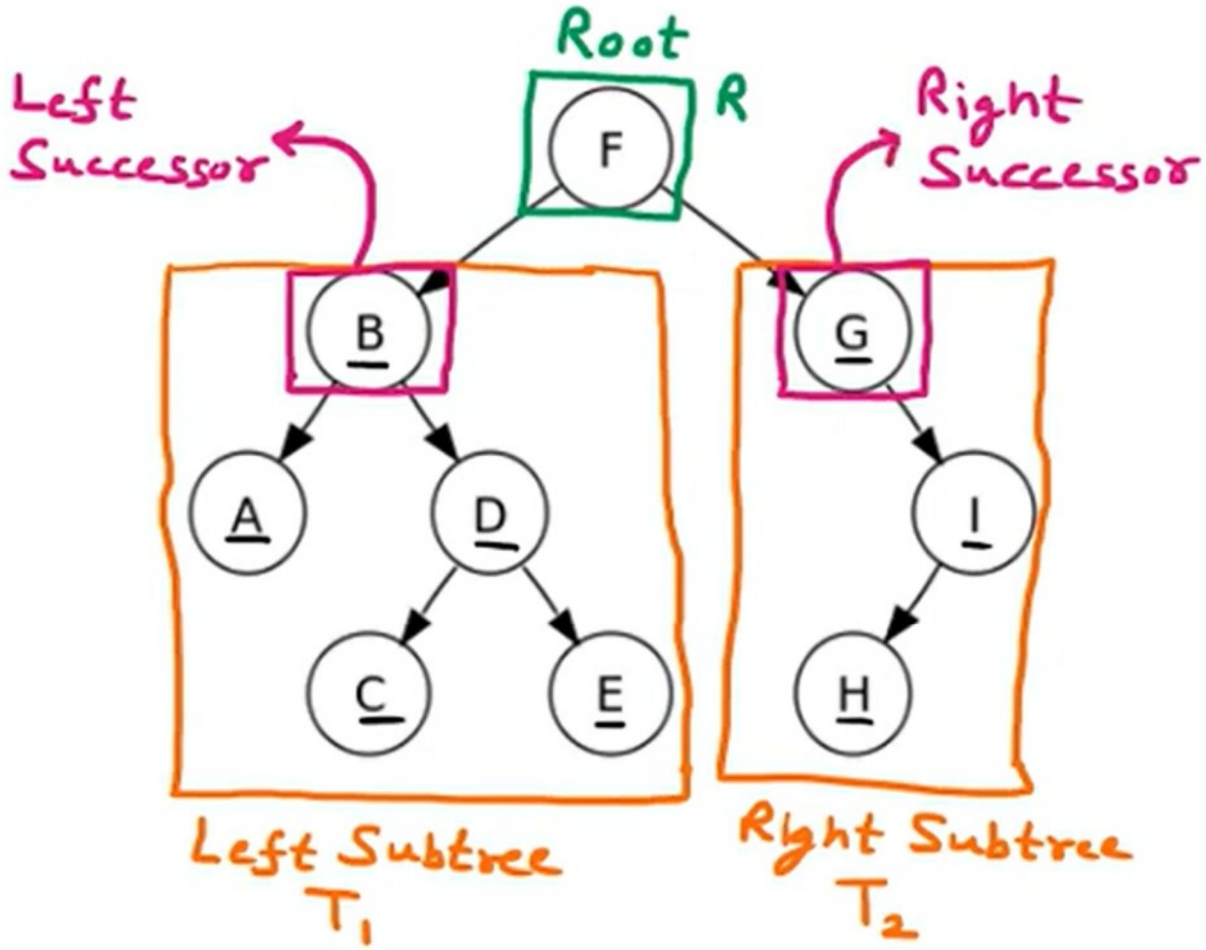
- Node at the top of hierarchy is called Root Node

Left & Right Subtree

- If Tree contain Root, then its left is Left Subtree and righ is Right Subtree

Left Subtree Nodes: B, A, D, C, E

Right Subtree Nodes: G, I, H



Left & Right Successor

- Root of Left Subtree is called Left Successor and Root of Right Subtree is called Right Successor

Binary Tree

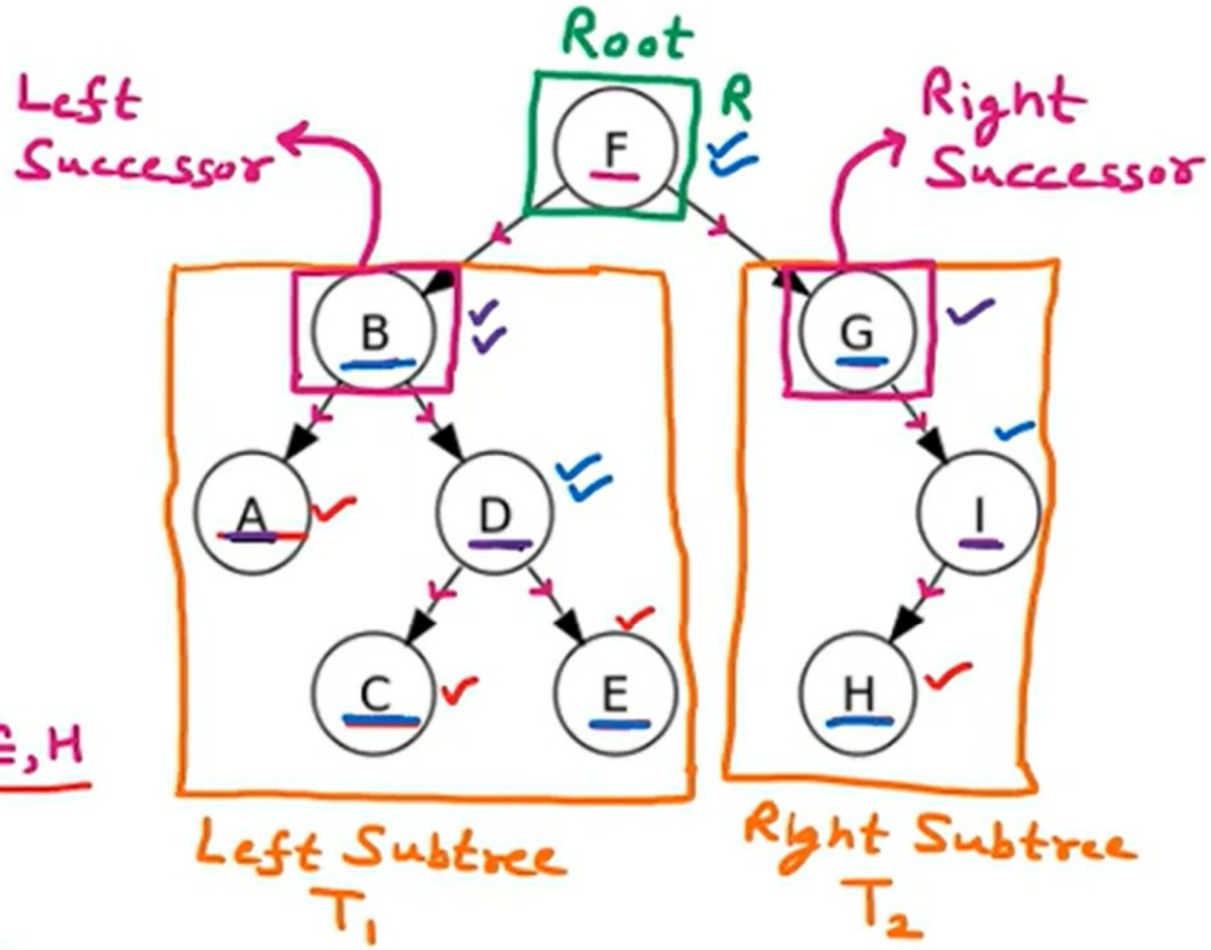
Successor

- Any Node in Binary Tree has either 0, 1 or 2 Successors

Nodes has 2 Successors: F, B, D

Nodes has 1 Successor: G, I

Nodes has 0 (No) Successor: A, C, E, H



Terminal Nodes

- Nodes with 0 (No) Successor called Terminal Nodes

A, C, E, H

Predecessor (Ancestor)

- Every Node in Binary Tree, except Root has unique parent called Predecessor or Ancestor

Binary Tree

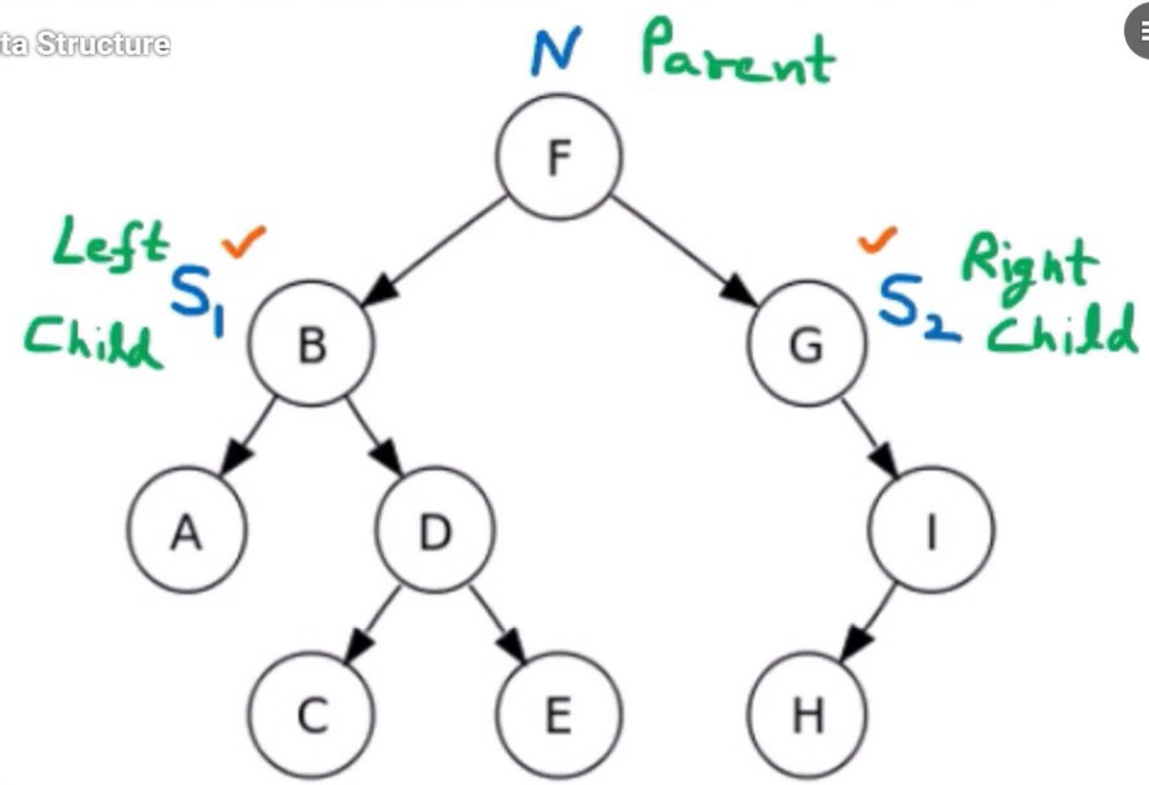
Terminology

- Terminology describes relationships between Nodes of Tree

Parent, Child & Siblings

Suppose Node N in Tree has Left Successor S_1 and Right Successor S_2 then

- N is called Parent of S_1 & S_2
- S_1 is called Left Child & S_2 is called Right Child of N
- S_1 & S_2 are Siblings



Binary Tree

Terminology

- Terminology describes relationships between Nodes of Tree

Edge

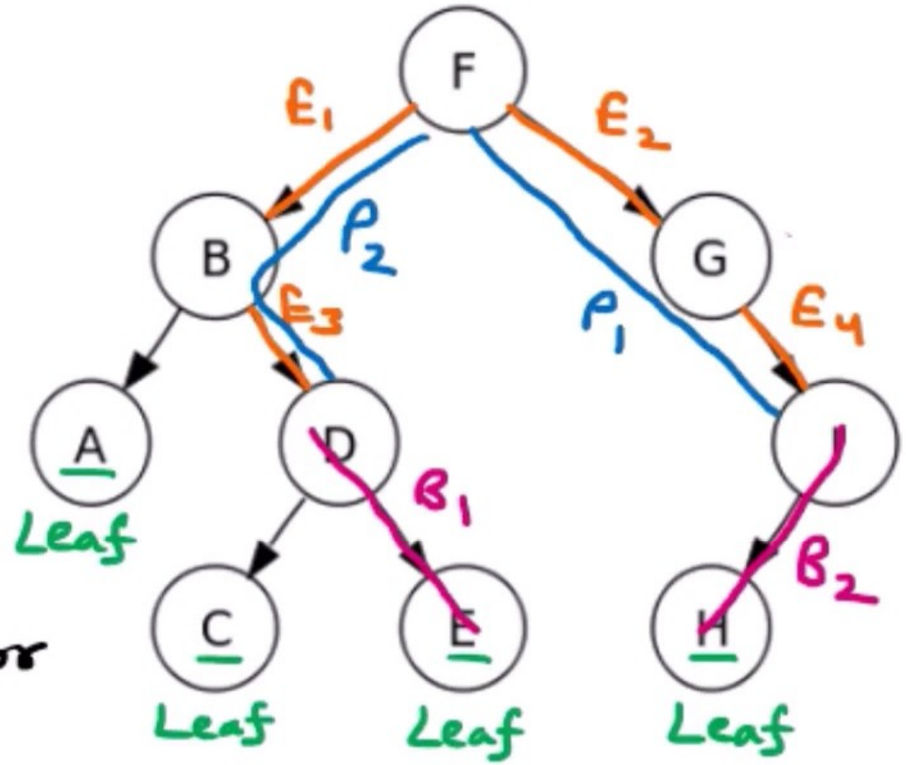
- Line drawn from Node to Successor is called Edge

Path

- Sequence of consecutive Edges is called Path

Leaf

- Terminal Node is called Leaf



Branch

- Path ending in Leaf is called Branch

Binary Tree

Terminology

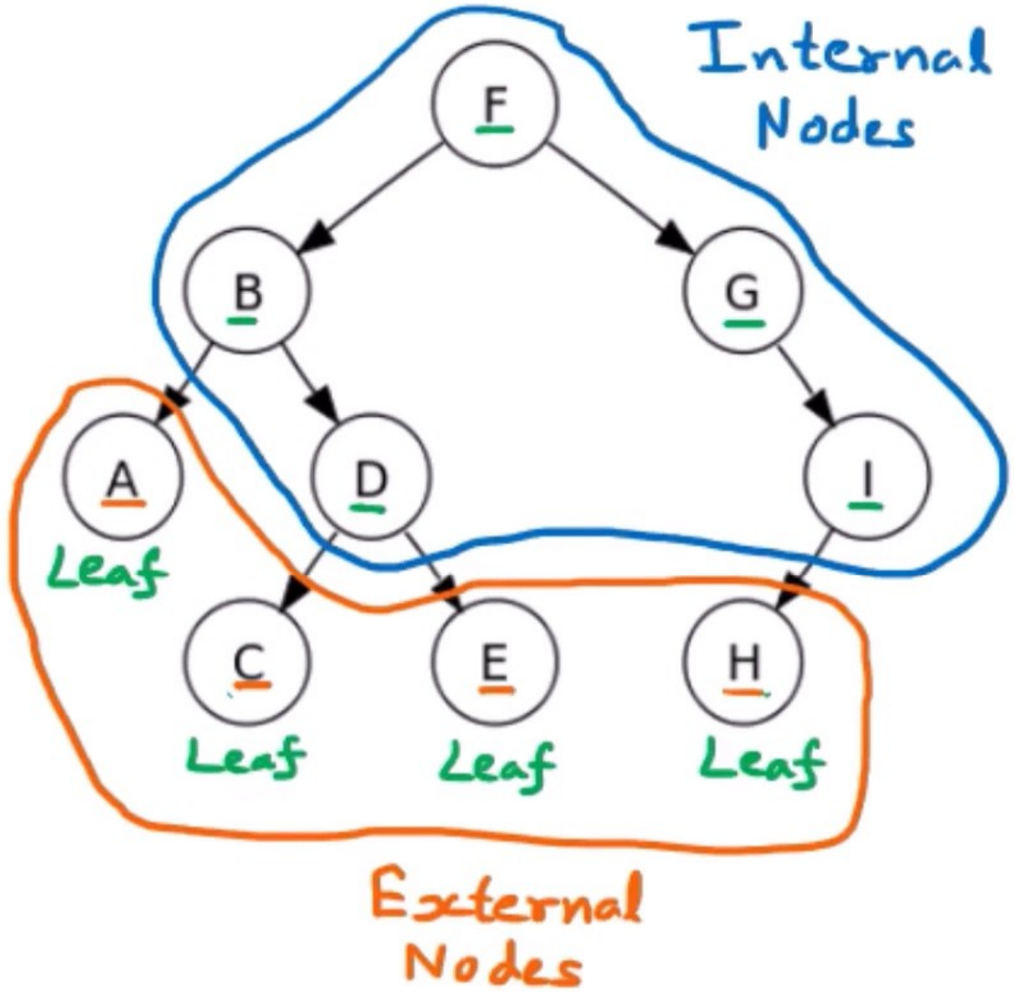
- Terminology describes relationships between Nodes of Tree

External Nodes (Leaf Nodes)

- Nodes with zero children

Internal Nodes

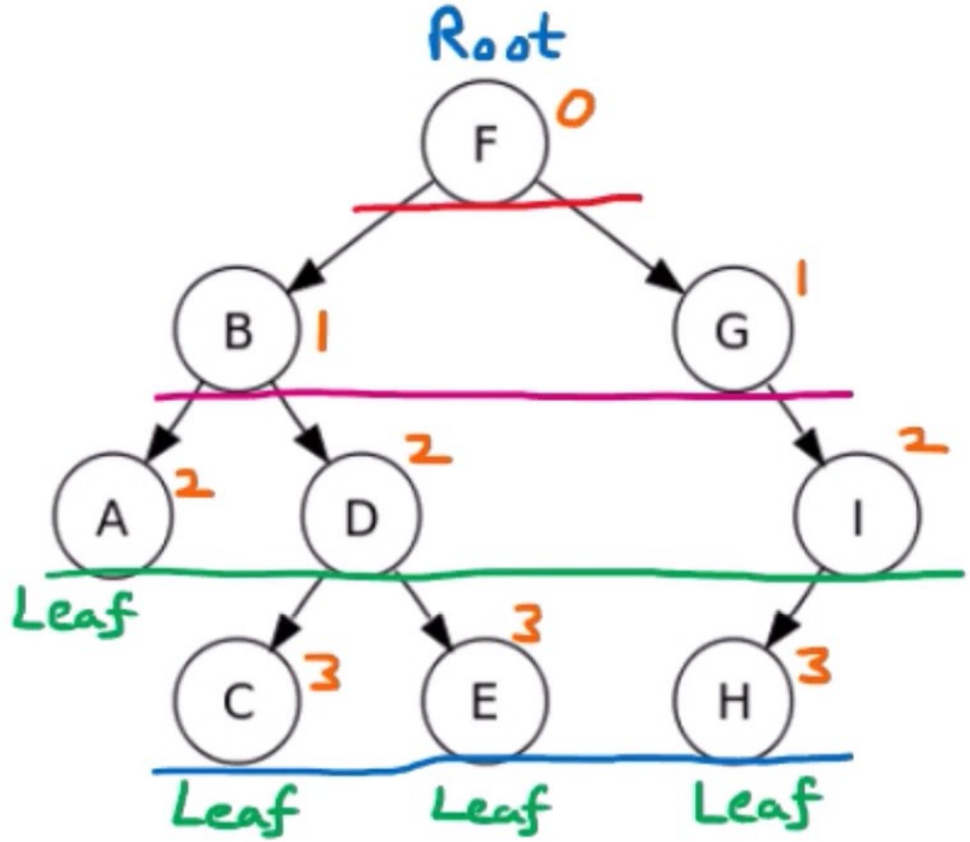
- Nodes except External Nodes



Binary Tree

Terminology

- Terminology describes relationships between Nodes of Tree



Level Number

- Each Node in Tree is assigne a Level Number
- Root is assigned Level Number 0
- Every other Node is assigned a Level Number which is 1 more than Level Number of its Parent
- Same Level Number belong to Same Generation

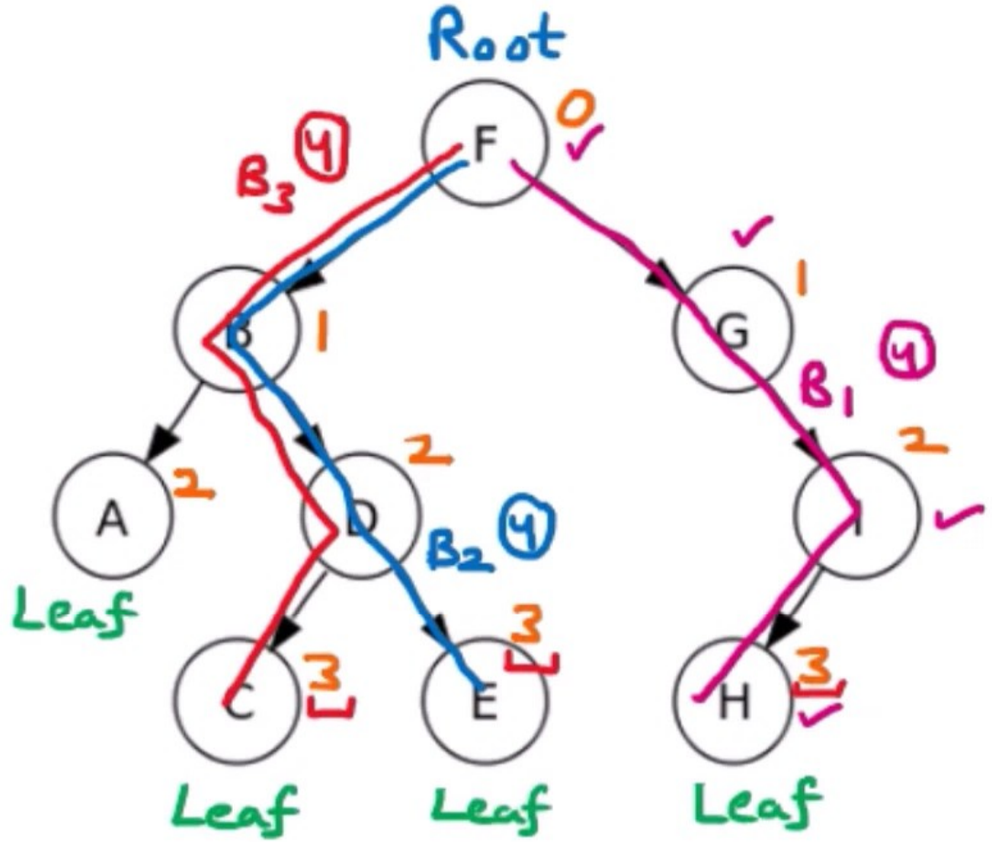
Binary Tree

Terminology

- Terminology describes relationships between Nodes of Tree

Depth (Height)

- Depth of a Tree is maximum number of Nodes in a Branch
- Depth is 1 more than Largest Level Number



Largest Level Number = 3

Depth = Largest Level Number + 1

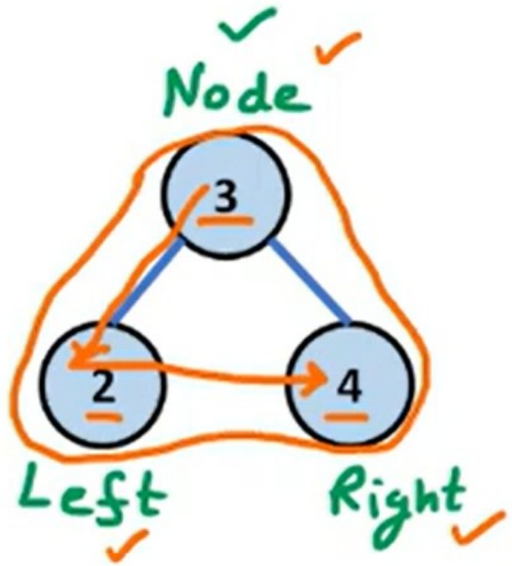
= 3 + 1 = 4

Traversing Binary Tree

Pre Order

Node - Left - Right

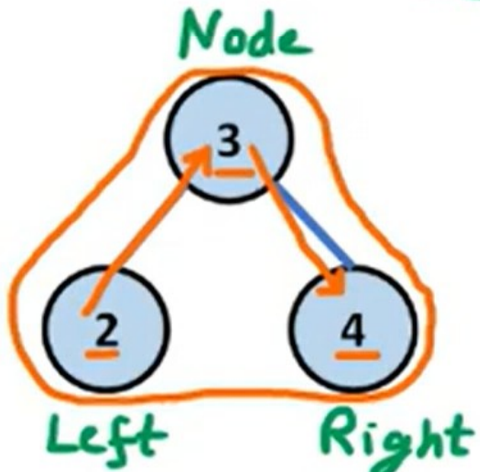
3 2 4



In Order

Left - Node - Right

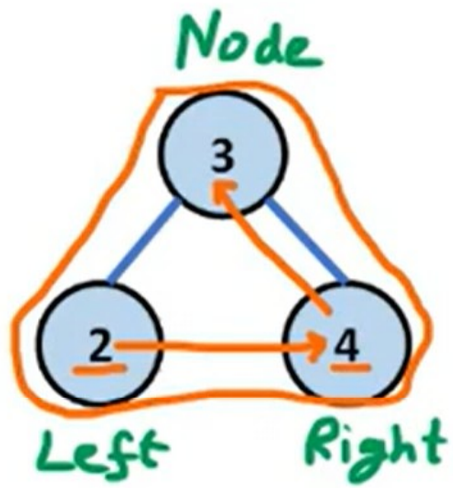
2 3 4



Post Order

Left - Right - Node

2 4 3



Traversing Binary Tree

Pre Order ✓

Node - Left - Right

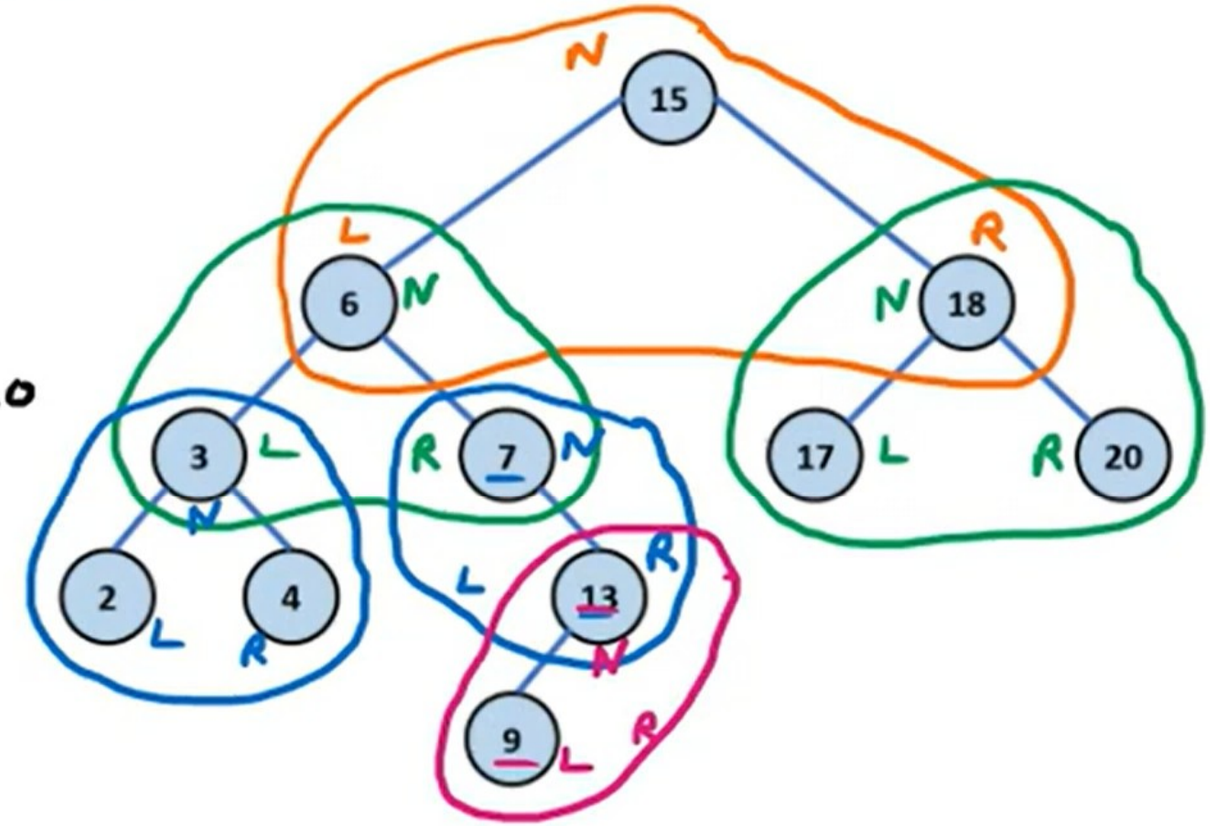
15 6 3 2 4 7 13 9 18 17 20

In Order

Left - Node - Right

Post Order

Left - Right - Node



15

6 3 2 4 7 9 13 18 17 20

Traversing Binary Tree

Pre Order

Node - Left - Right

15 6 3 2 4 7 13 9 18 17 20

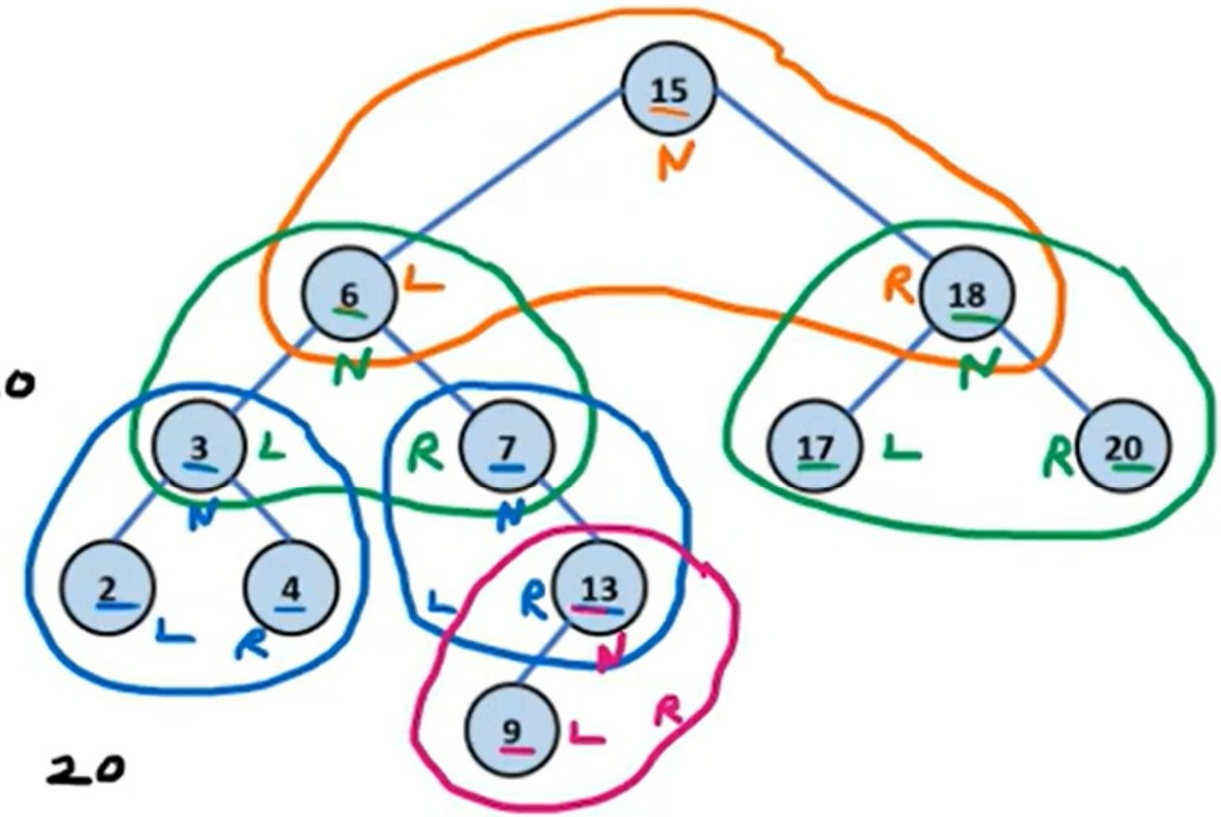
In Order ✓

Left - Node - Right

2 3 4 6 7 9 13 15 17 18 20

Post Order

Left - Right - Node 2 3 4 6 7 9 13 15 17 18 20



Traversing Binary Tree

Pre Order

Node - Left - Right

15 6 3 2 4 7 13 9 18 17 20

In Order

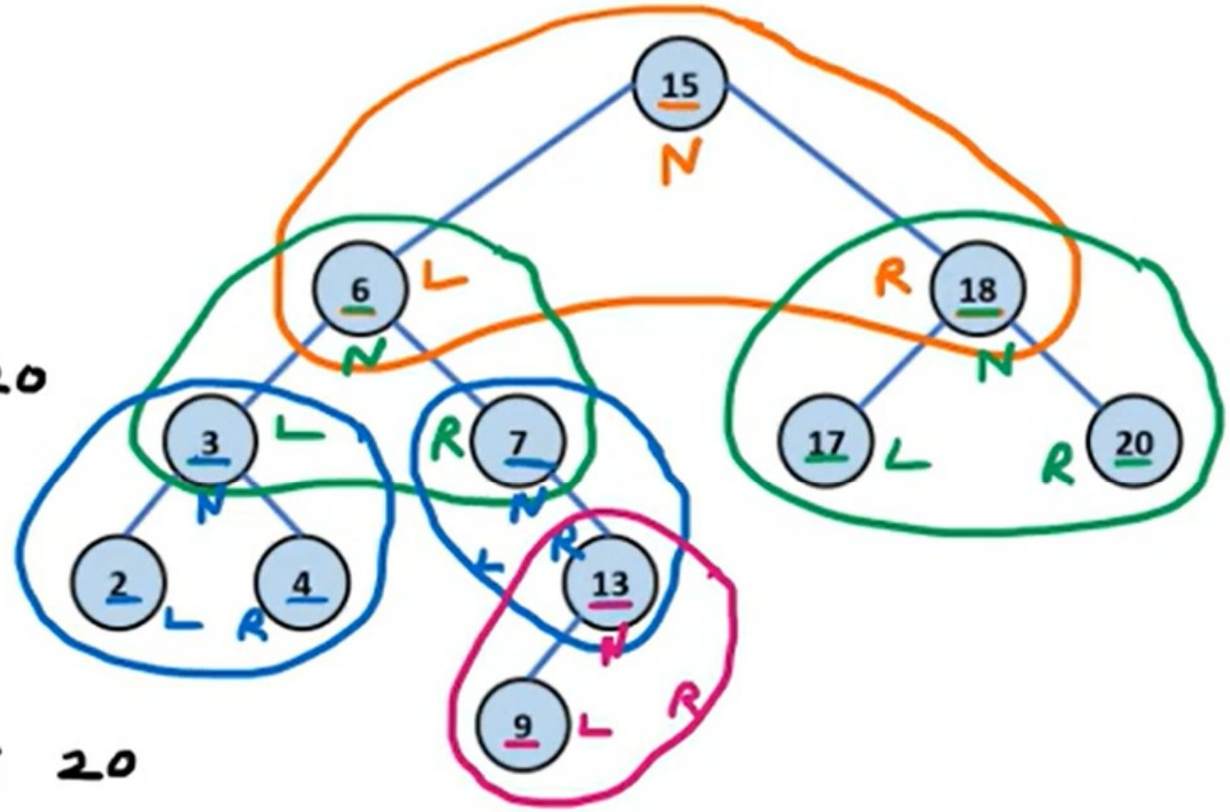
Left - Node - Right

2 3 4 6 7 9 13 15 17 18 20

Post Order ✓

Left - Right - Node 2 4 3 9 13 7 6 17 20 18 15

2 4 3 9 13 7 6 17 20 18 15



Construct Binary Tree from Arithmetic Expression

E denote following Arithmetic Expression:

E = (A – B) / ((C * D) + E)

- Construct the Binary Tree T of E

(A – B) / ((C * D) + E)

L N R

A – B

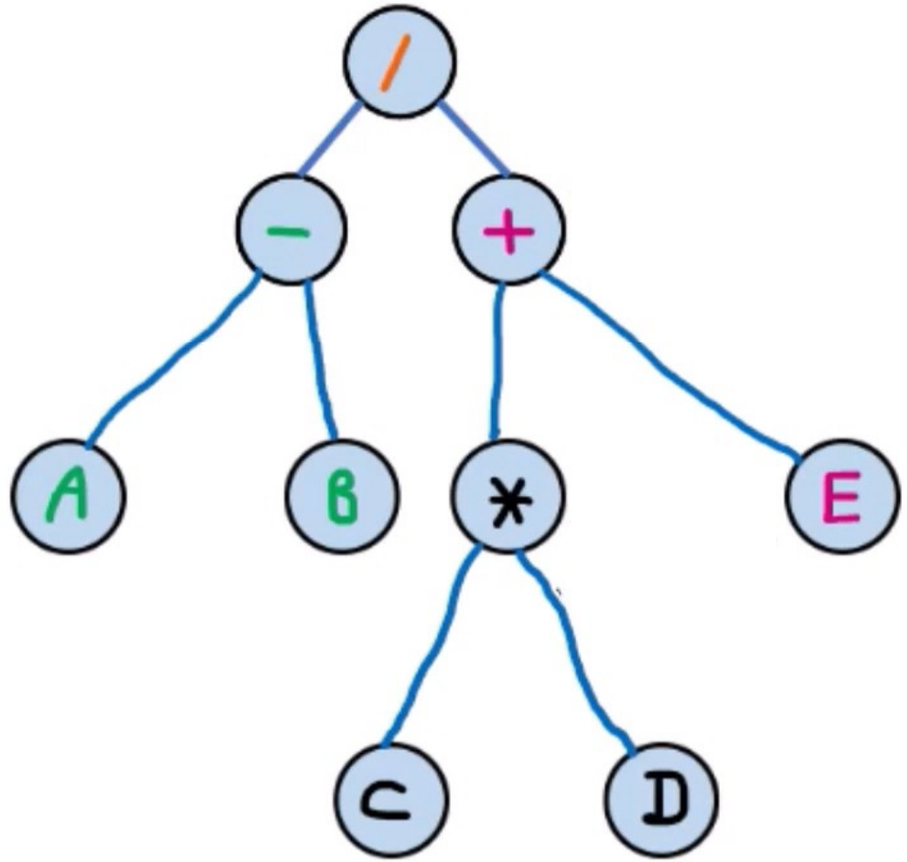
L N R

(C * D) + E

L N R

C * D

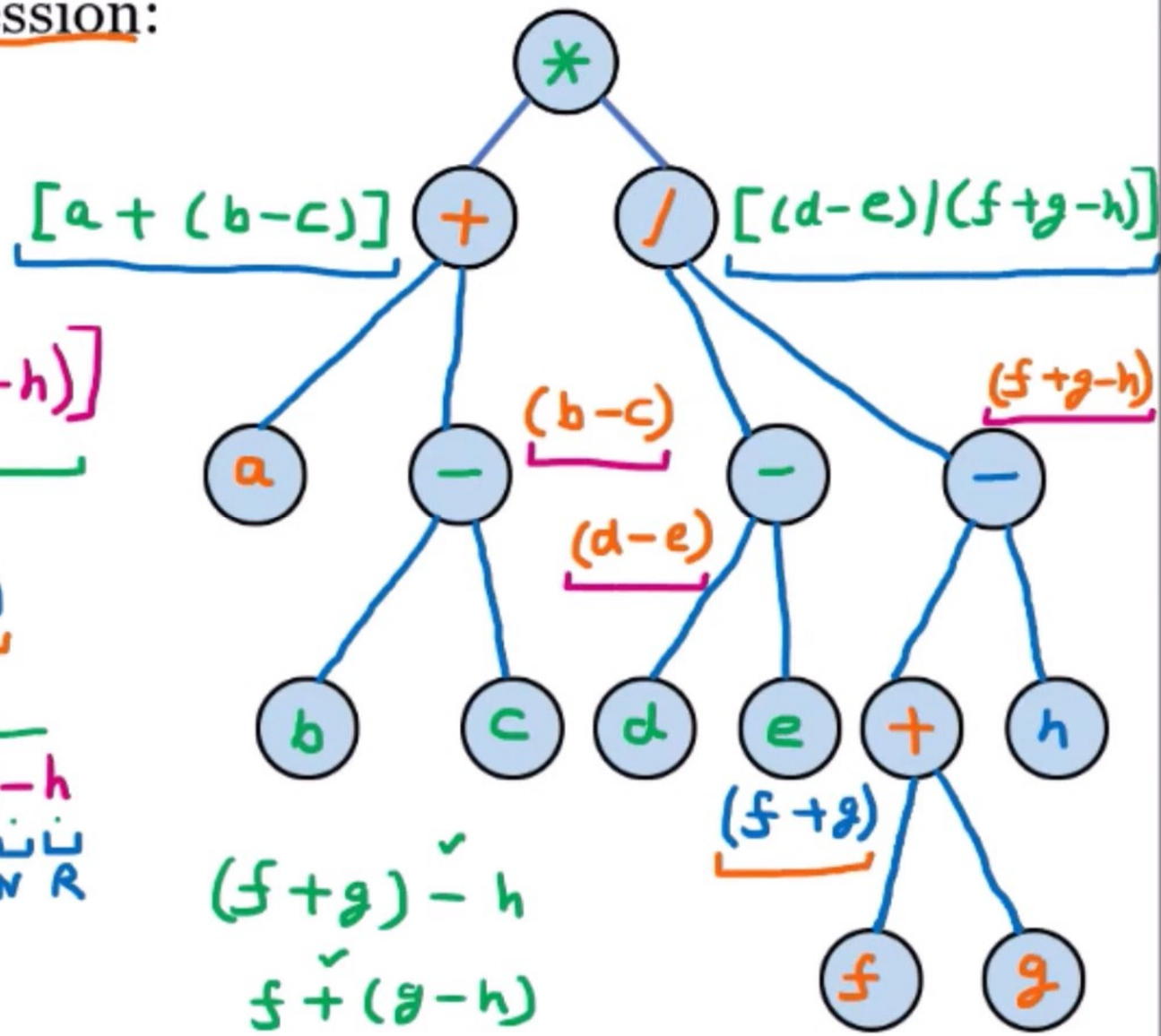
L N R



E denote following Arithmetic Expression:

E = [a + (b – c)] * [(d – e) / (f + g –h)]

- Construct the Binary Tree T of E.
- Calculate Preorder Traversal of T.
- Calculate Postorder Traversal of T.



[a + (b-c)] * [(d-e)/(f+g-h)]

L N R

a + (b-c) | (d-e) / (f+g-h)

L N R | L N R

b-c | d-e | (f+g)-h

L N R | L N R | L N R

f+g

L N R

(f+g) - h

f + (g-h)

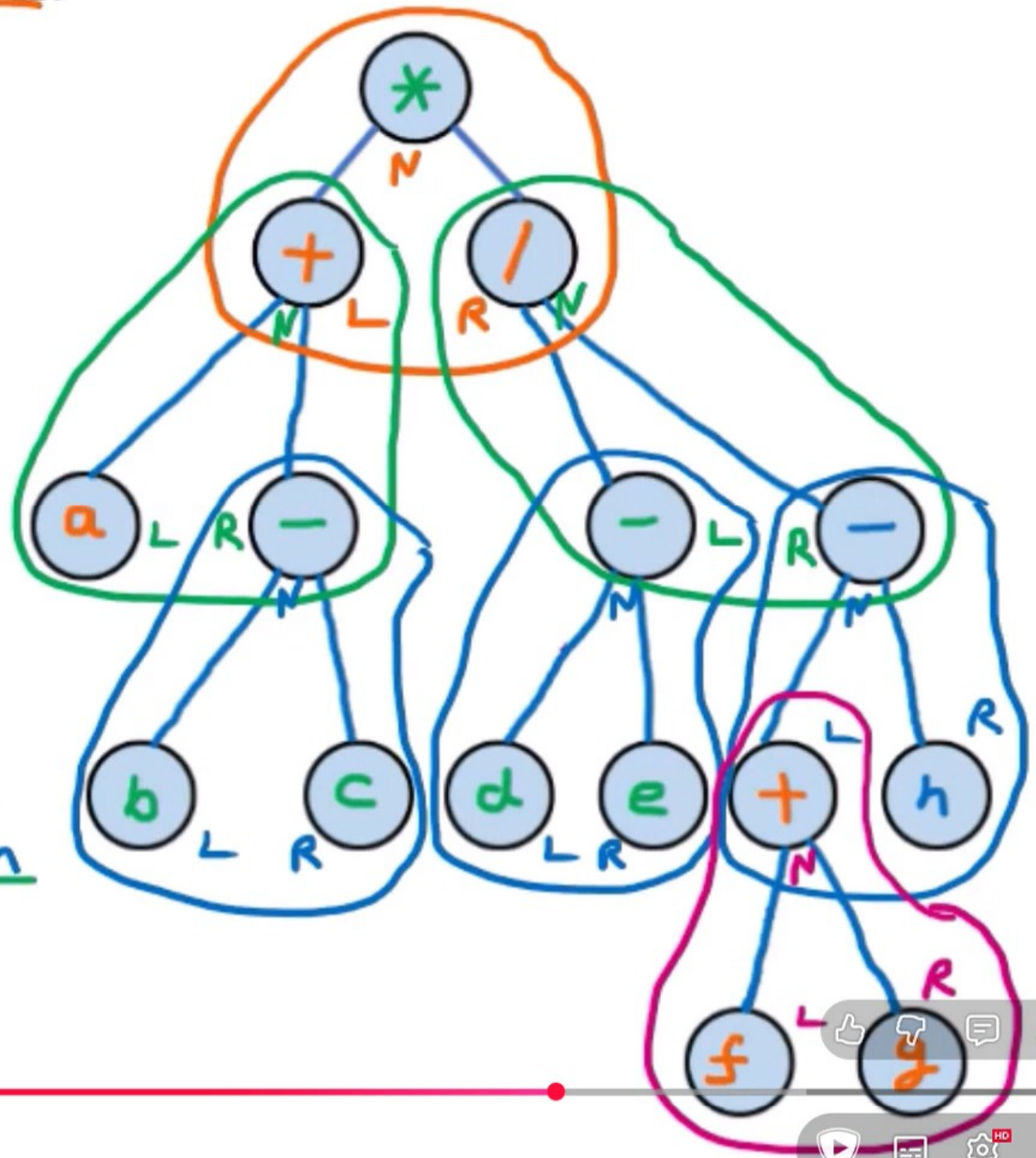
E denote following Arithmetic Expression:

E = [a + (b – c)] * [(d – e) / (f + g –h)]

- Construct the Binary Tree T of E.
- Calculate Preorder Traversal of T.
- Calculate Postorder Traversal of T.

Preorder (N L R):

* + a - b c / - d e - + f g h

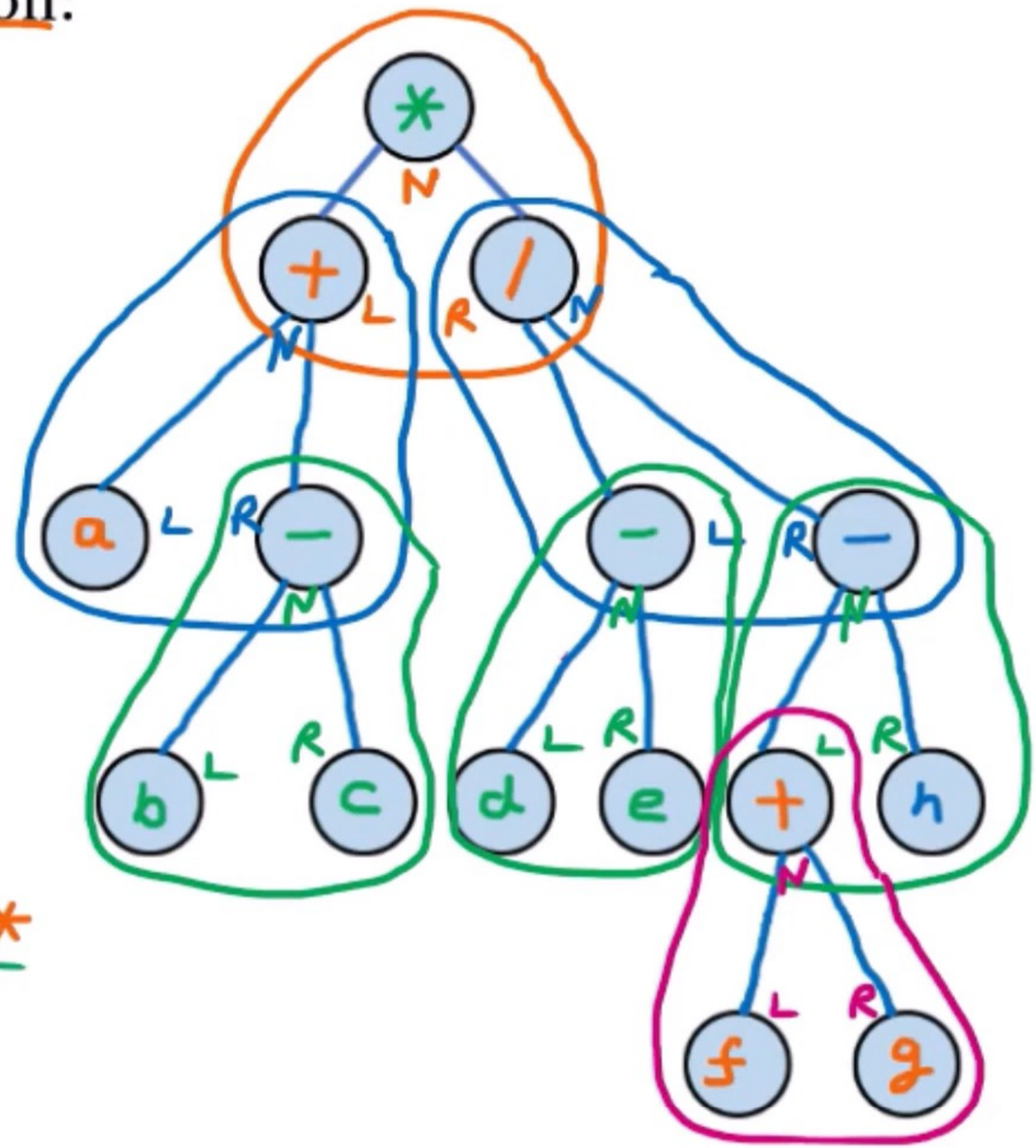


* + a - b c / - d e - + f g h

E denote following Arithmetic Expression:

E = [a + (b – c)] * [(d – e) / (f + g –h)]

- ✓ Construct the Binary Tree T of E.
- ✓ Calculate Preorder Traversal of T.
- Calculate Postorder Traversal of T.



Preorder (N L R):

* + a - b c / - d e - + f g h

Postorder (L R N):

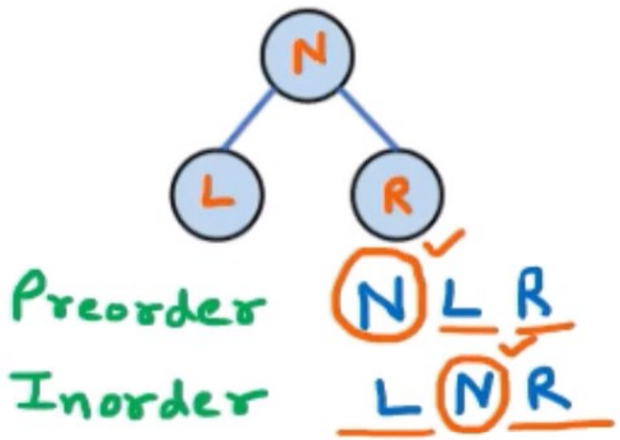
a b c – + d e – f g + h – / *

Construct Binary Tree with Preorder & Inorder

Construct Tree with following Sequences of Nodes:

Preorder: F A E K C D H G B

Inorder: E A C K F H D B G



Preorder: F A E K C D H G B

L L R L R R

Inorder: E A C K F H D B G

L R

E A C K H D B G

L R L R

E C K H B G

L R L

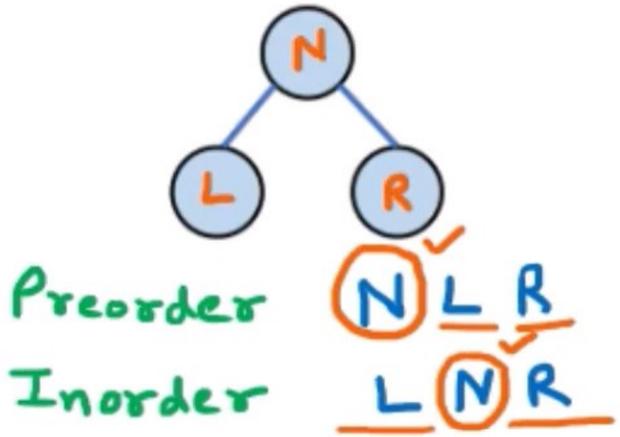
C B

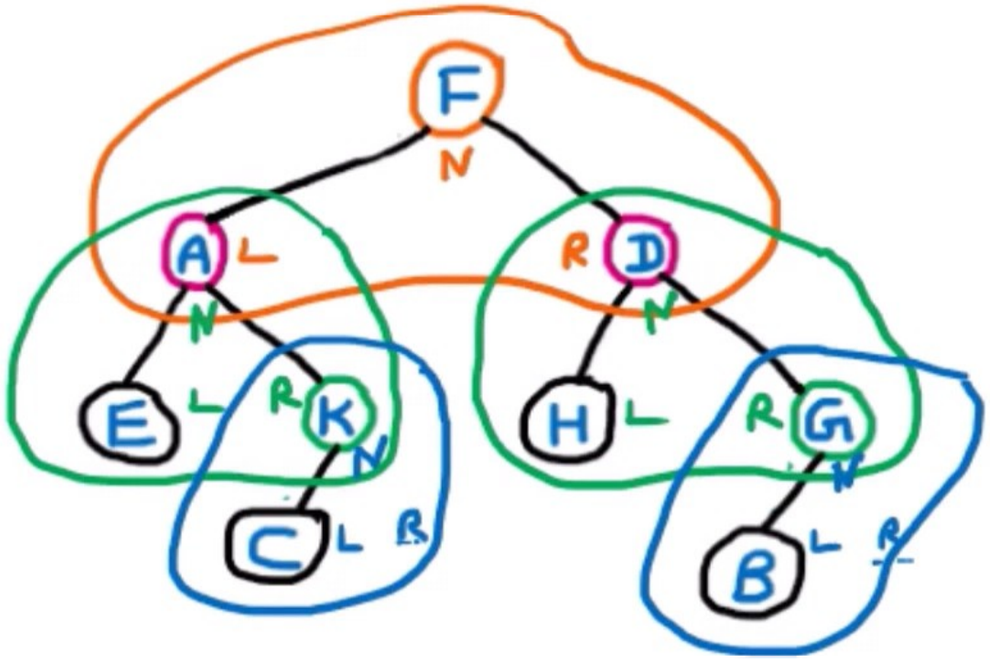
Construct Binary Tree with Preorder & Inorder

Construct Tree with following Sequences of Nodes:

Preorder: F A E K C D H G B

Inorder: E A C K F H D B G ✓



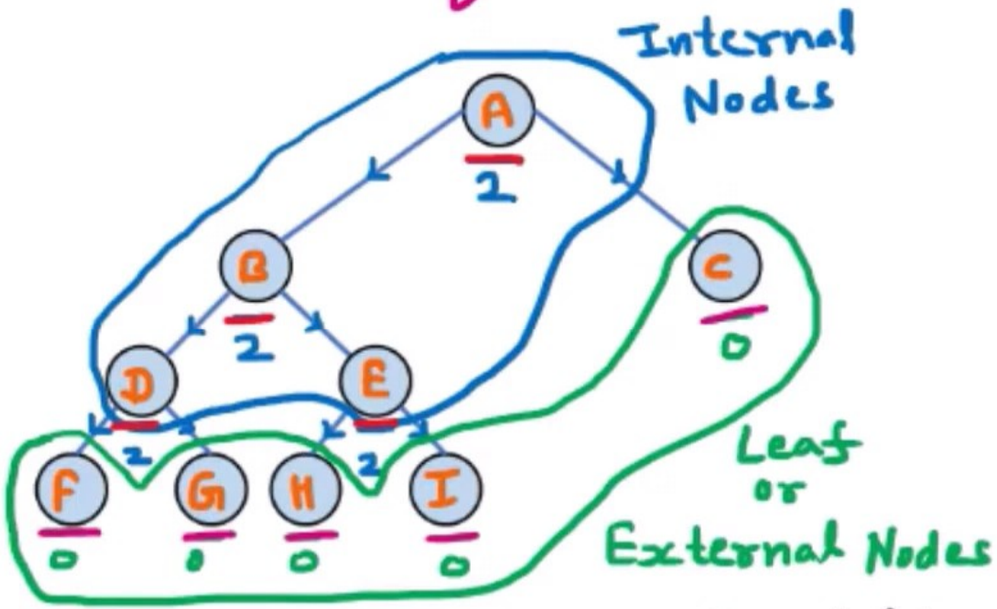


Inorder:

E A C K F H D B G

Types of Binary Tree

Full Binary Tree

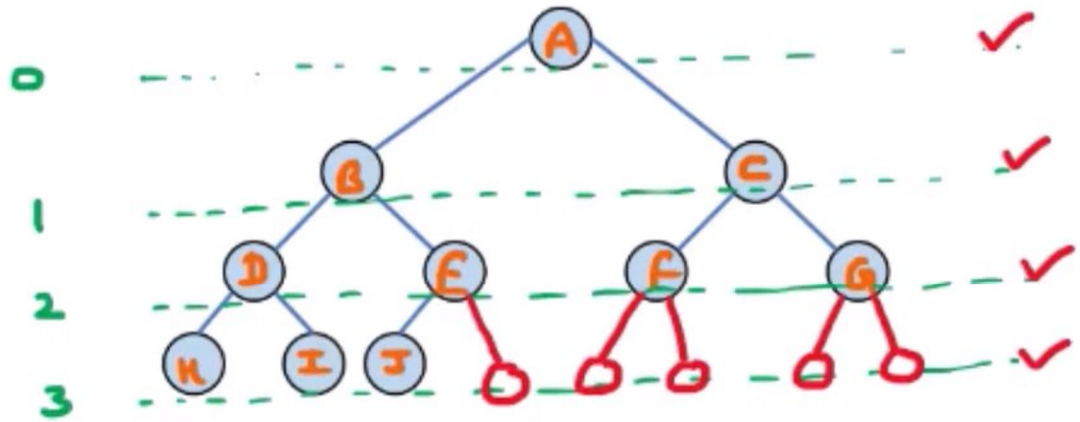


- Each Node has 0 or 2 children
- Each Node except External Nodes has 2 Children

Leaf Nodes = Internal Nodes + 1

5 4

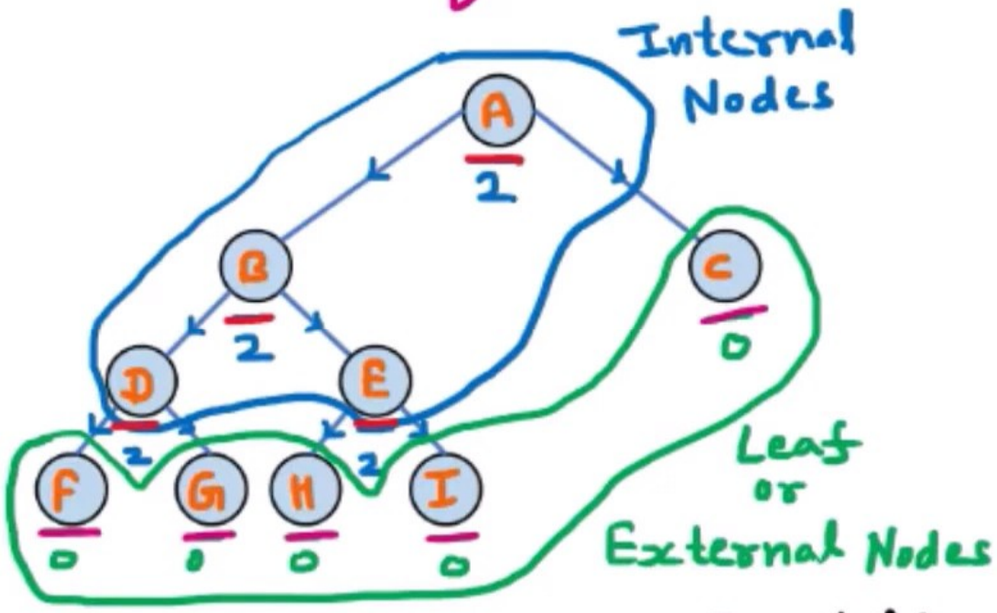
Complete Binary Tree



- All levels filled with Nodes except lowest level & lowest level nodes reside on left side

Types of Binary Tree

Full Binary Tree

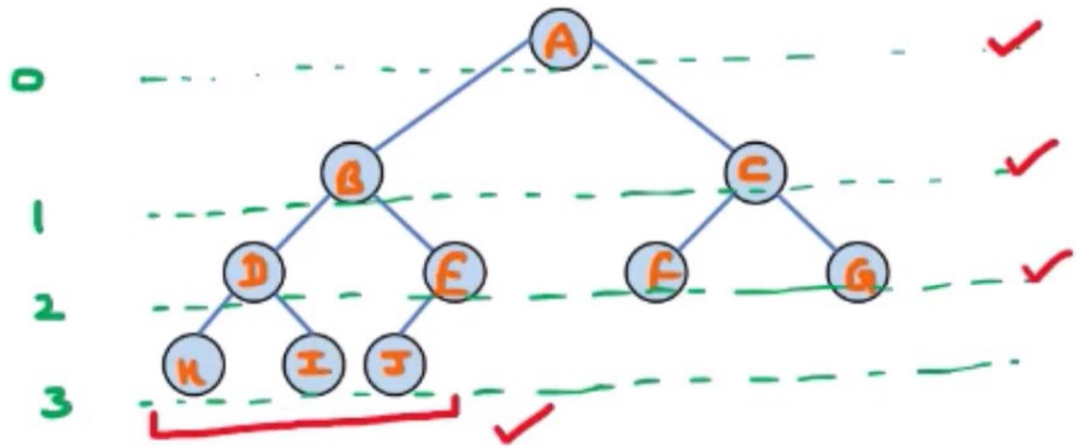


- Each Node has 0 or 2 children
- Each Node except External Nodes has 2 Children

Leaf Nodes = Internal Nodes + 1

4

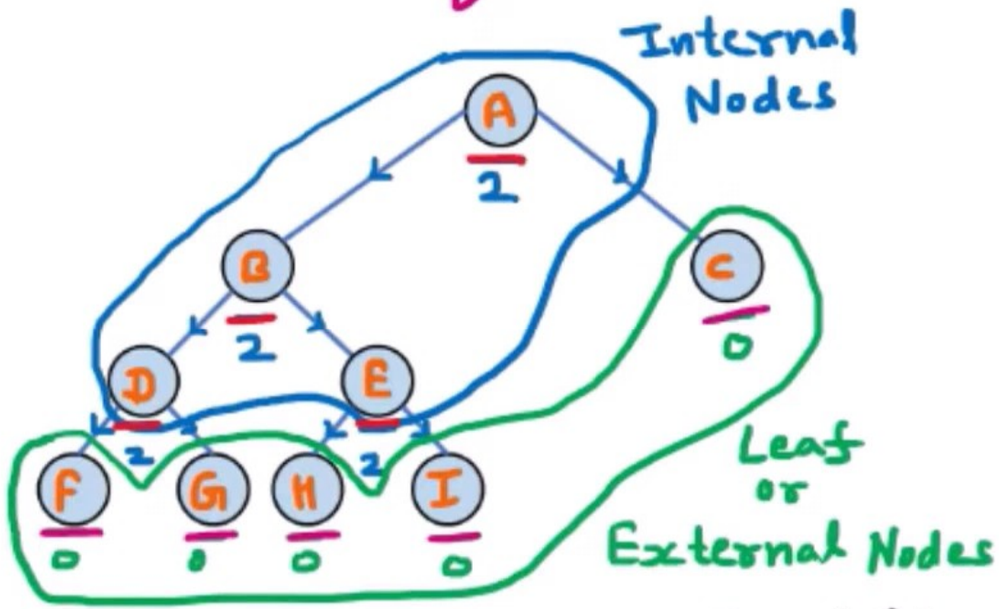
Complete Binary Tree



- All levels filled with Nodes except lowest level & lowest level nodes reside on left side

Types of Binary Tree

Full Binary Tree

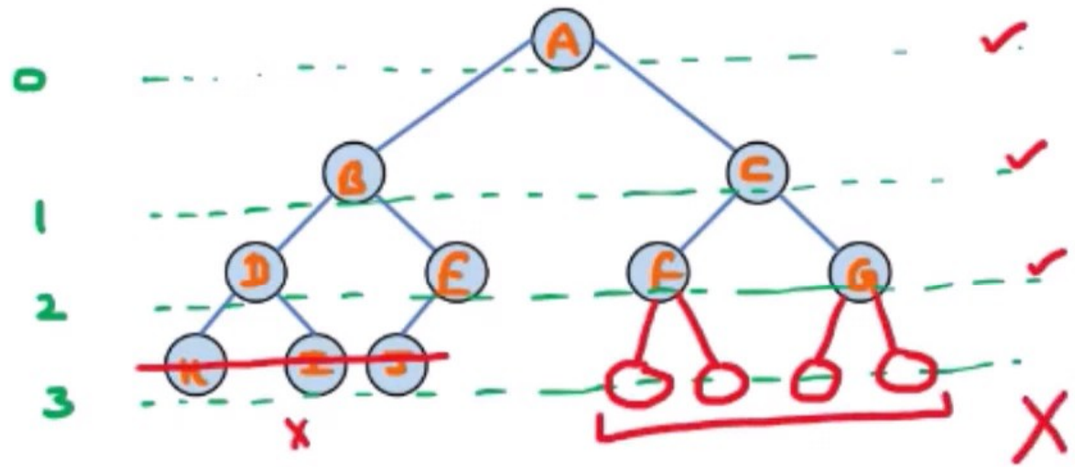


- Each Node has 0 or 2 children
- Each Node except External Nodes has 2 Children

Leaf Nodes = Internal Nodes + 1

5 4

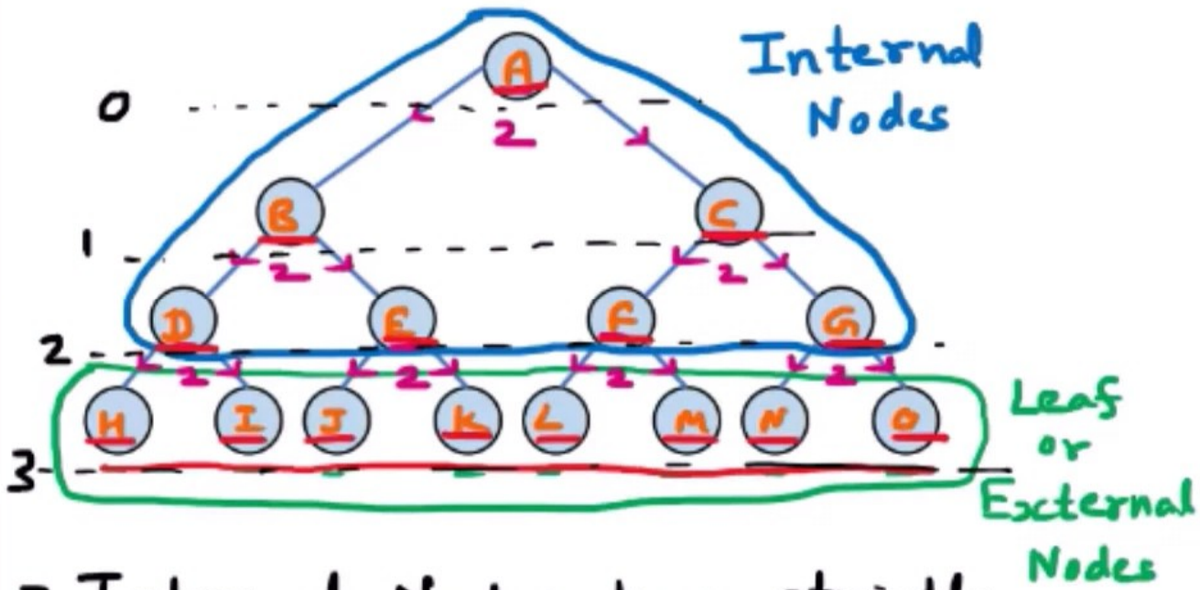
Complete Binary Tree



- All levels filled with Nodes except lowest level & lowest level nodes reside on left side

Types of Binary Tree

Perfect Binary Tree



Largest Level = 3

Height (Depth) = Largest Level + 1

= 3 + 1

= 4

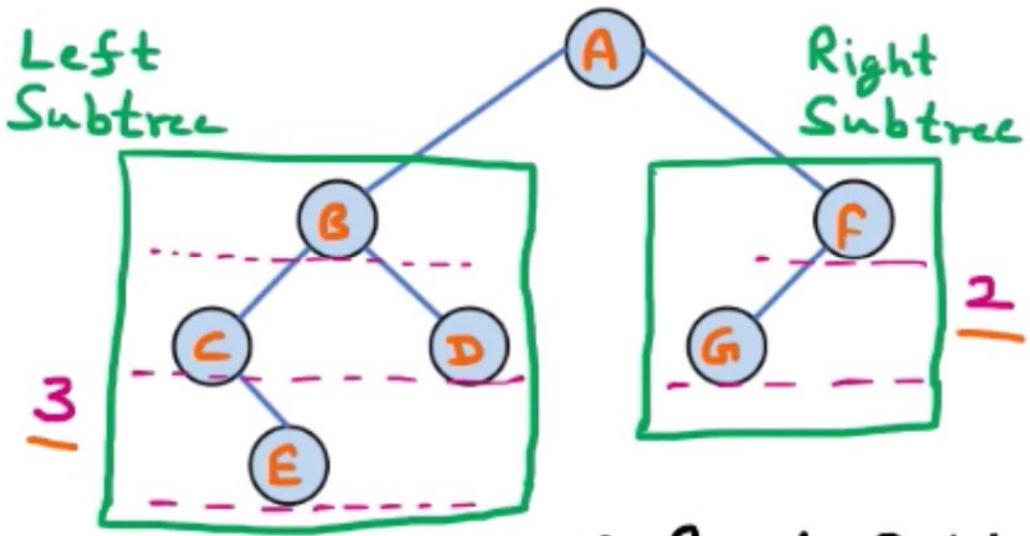
- Internal Nodes has strictly 2 children &
  External Nodes are at same Level

Perfect Binary Tree of Height h has $2^h - 1$ Nodes

$2^4 - 1 = 16 - 1 = 15$ ✓

Types of Binary Tree

Balanced Binary Tree



- Height of Left & Right Subtree vary by at most one $-1 \leq (h_L - h_R) \leq 1$

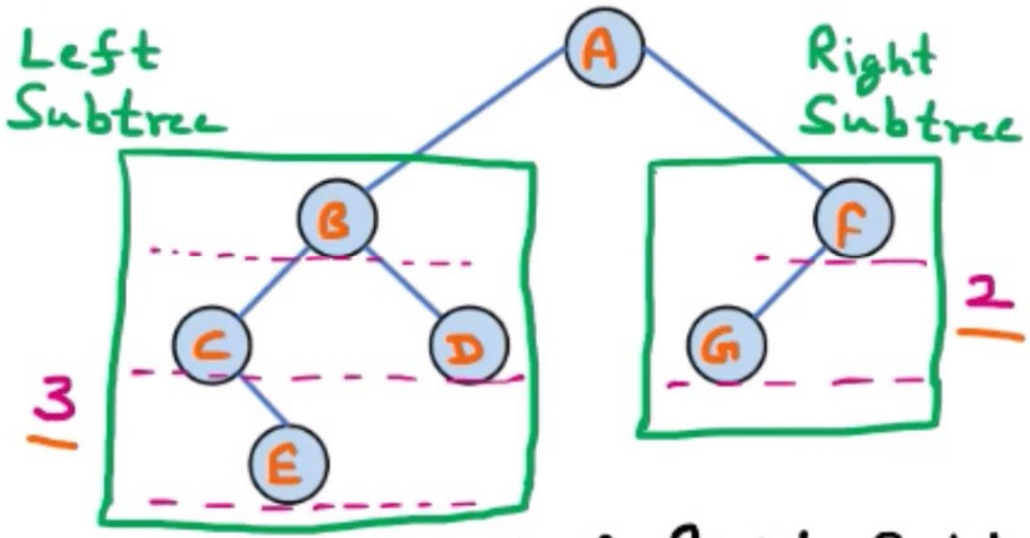
$h_L - h_R$	
0	✓
1	✓
-1	✓

$h_L - h_R = 3 - 2 = 1$

$h_R - h_L = 2 - 3 = -1$

Types of Binary Tree

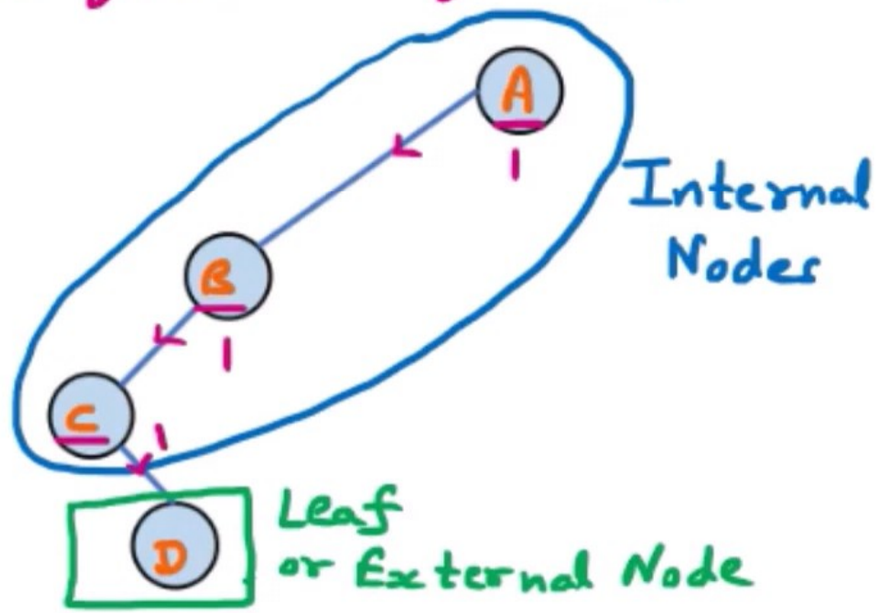
Balanced Binary Tree



- Height of Left & Right Subtree vary by at most one
- Tree height is O(log N), where N is number of Nodes

Example: AVL Tree, Red Black Tree

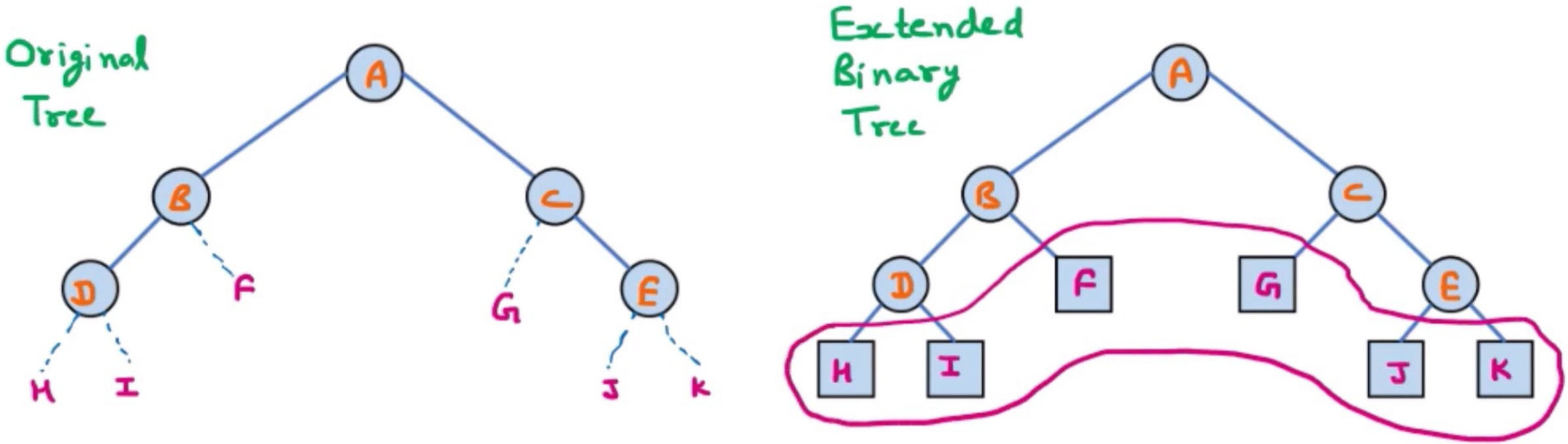
Degenerate Binary Tree (Pathological Binary Tree)



- Every Internal Node has Single Child

Types of Binary Tree

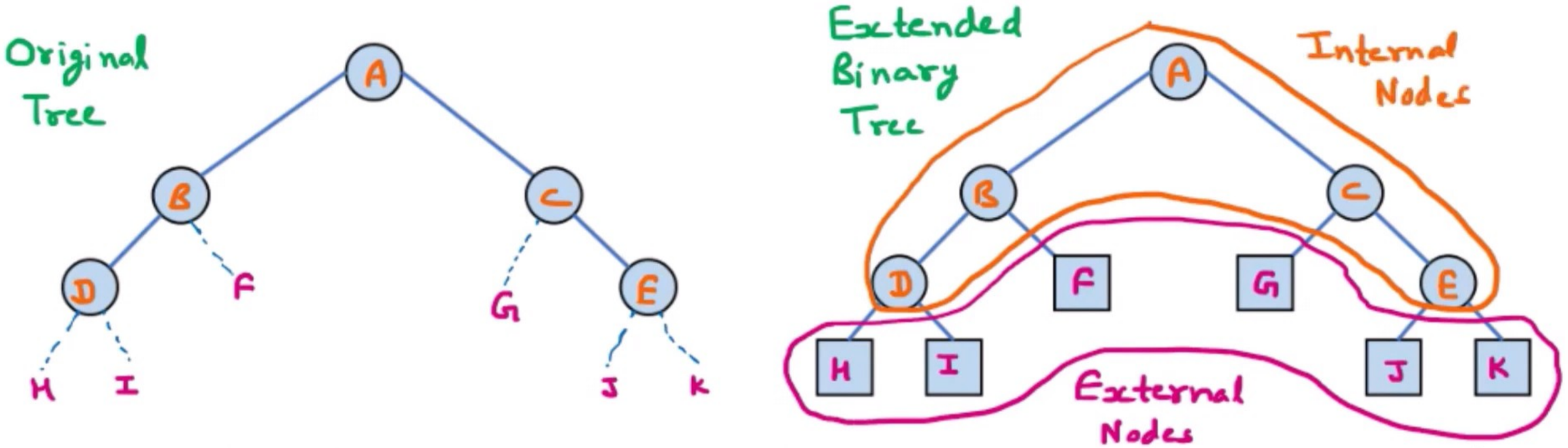
Extended Binary Tree



- All null subtree of original tree are replaced with special nodes called External Nodes,
  Whereas other nodes are called Internal Nodes

Types of Binary Tree

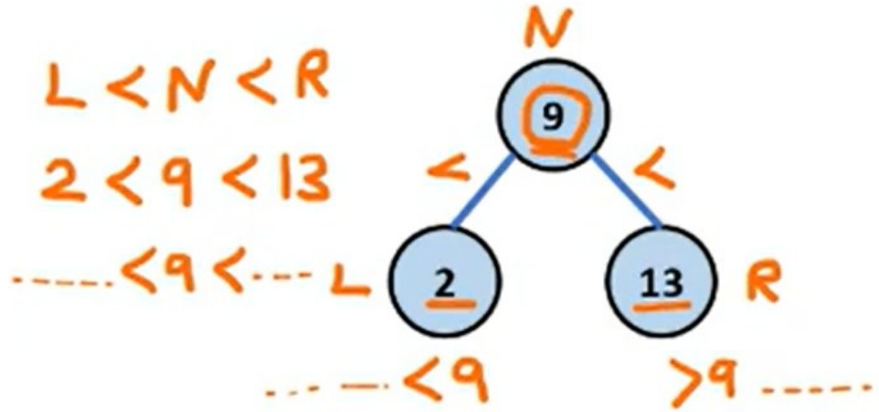
Extended Binary Tree



- All null subtree of original tree are replaced with special nodes called External Nodes,
  Whereas other nodes are called Internal Nodes

Binary Search Tree (BST)

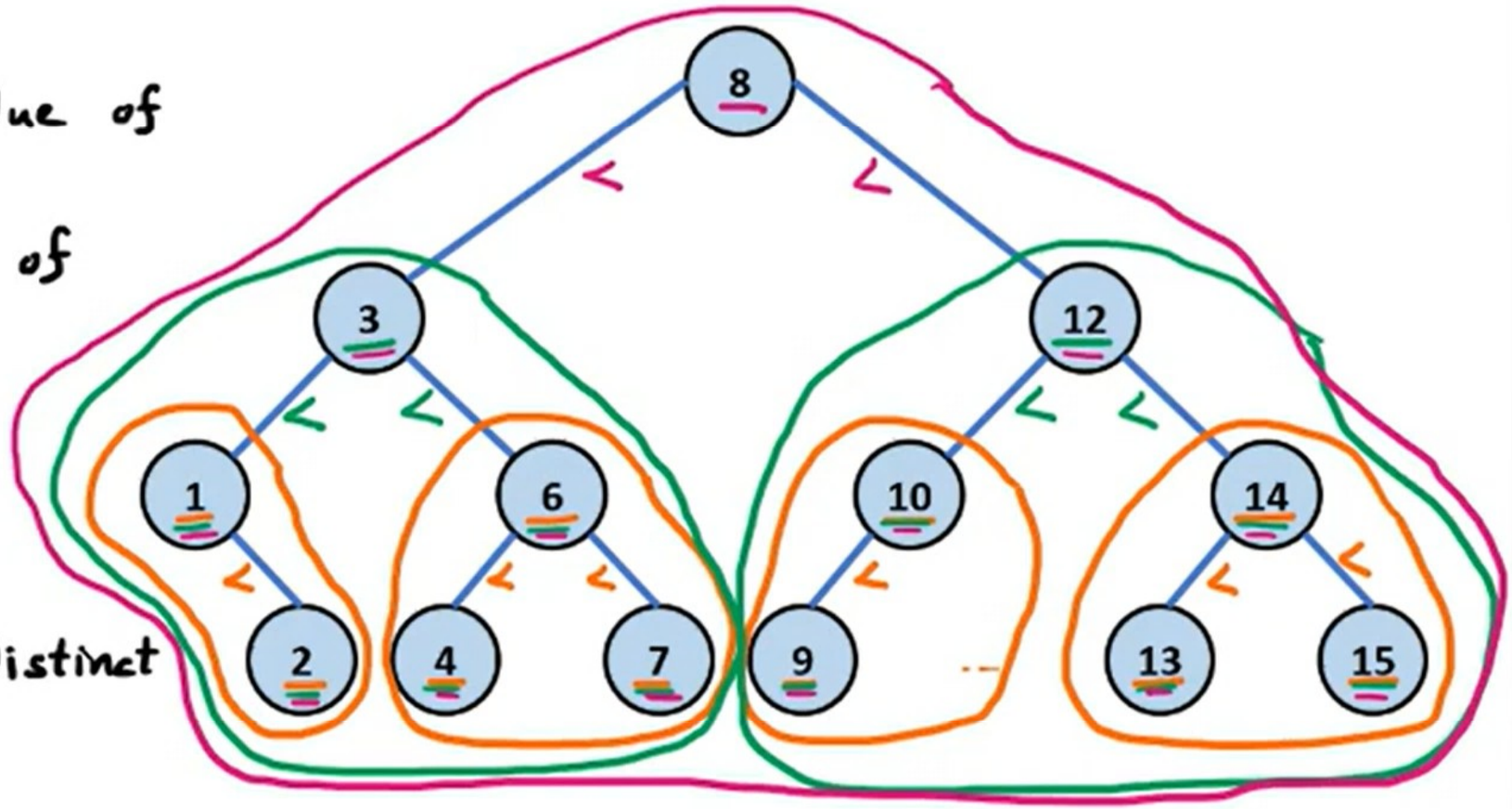
(Binary Sorted Tree)



Binary Tree is Binary Search Tree

if each Node N is

Greater than every value of
Left Subtree &
Less than every value of
Right Subtree of N

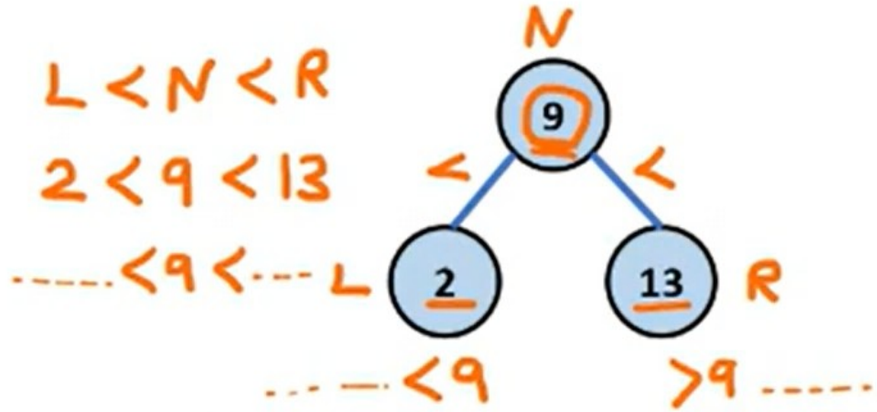


Assumption

All Nodes Values are Distinct

Binary Search Tree (BST)

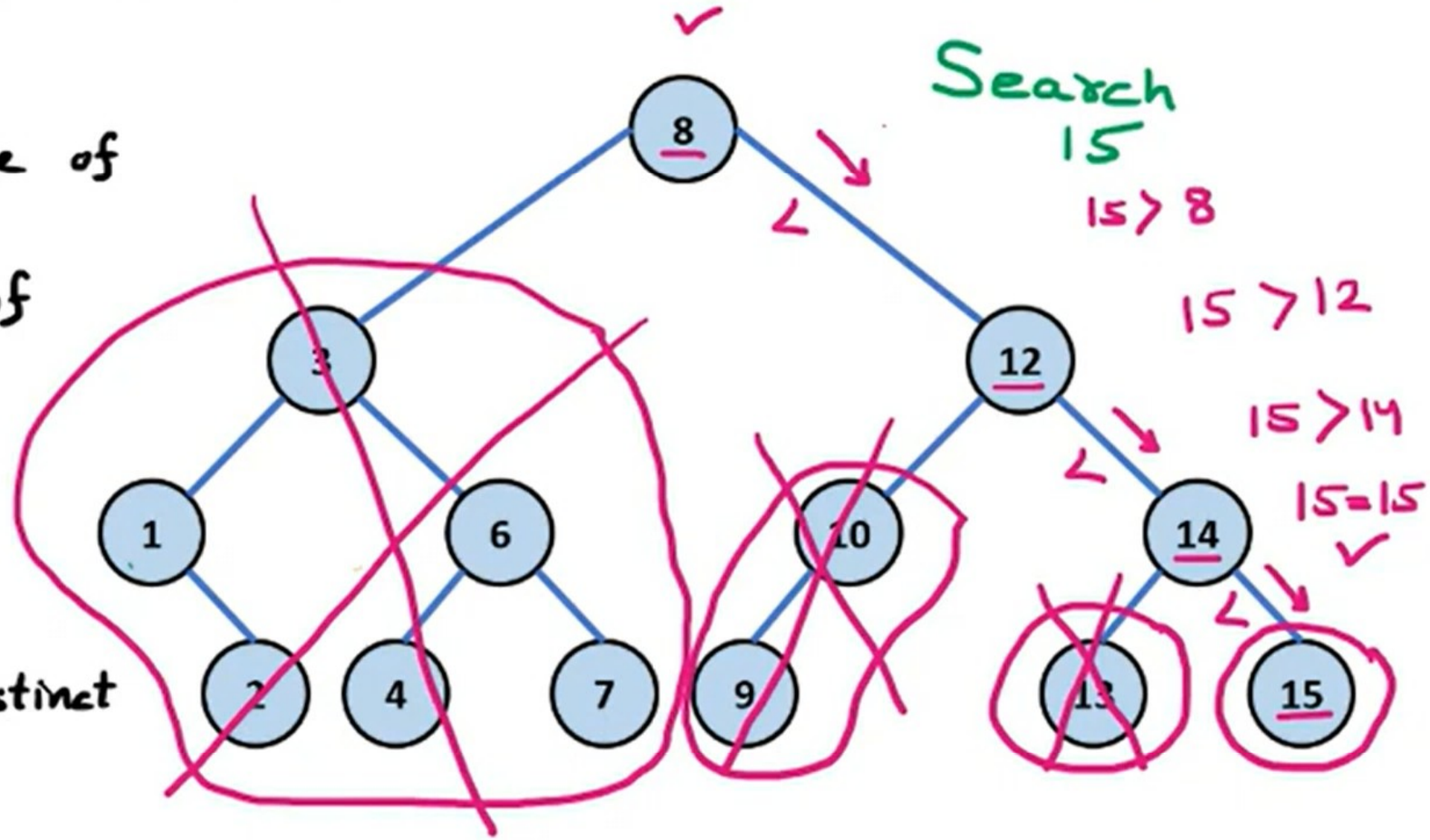
(Binary Sorted Tree)



Binary Tree is Binary Search Tree if each Node N is

Greater than every value of Left Subtree &

Less than every value of Right Subtree of N



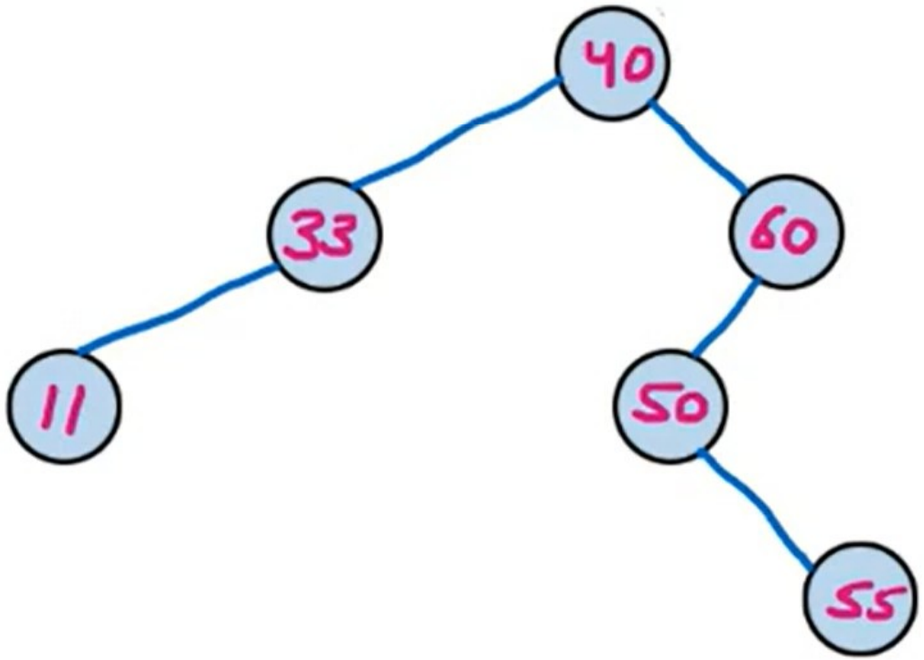
Assumption

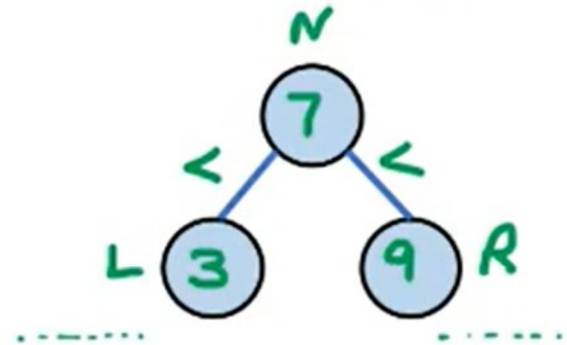
All Nodes Values are Distinct

Construction of Binary Search Tree (BST)

Suppose six Numbers are Inserted in order into empty Binary Search Tree:

40, 60, 50, 33, 55, 11



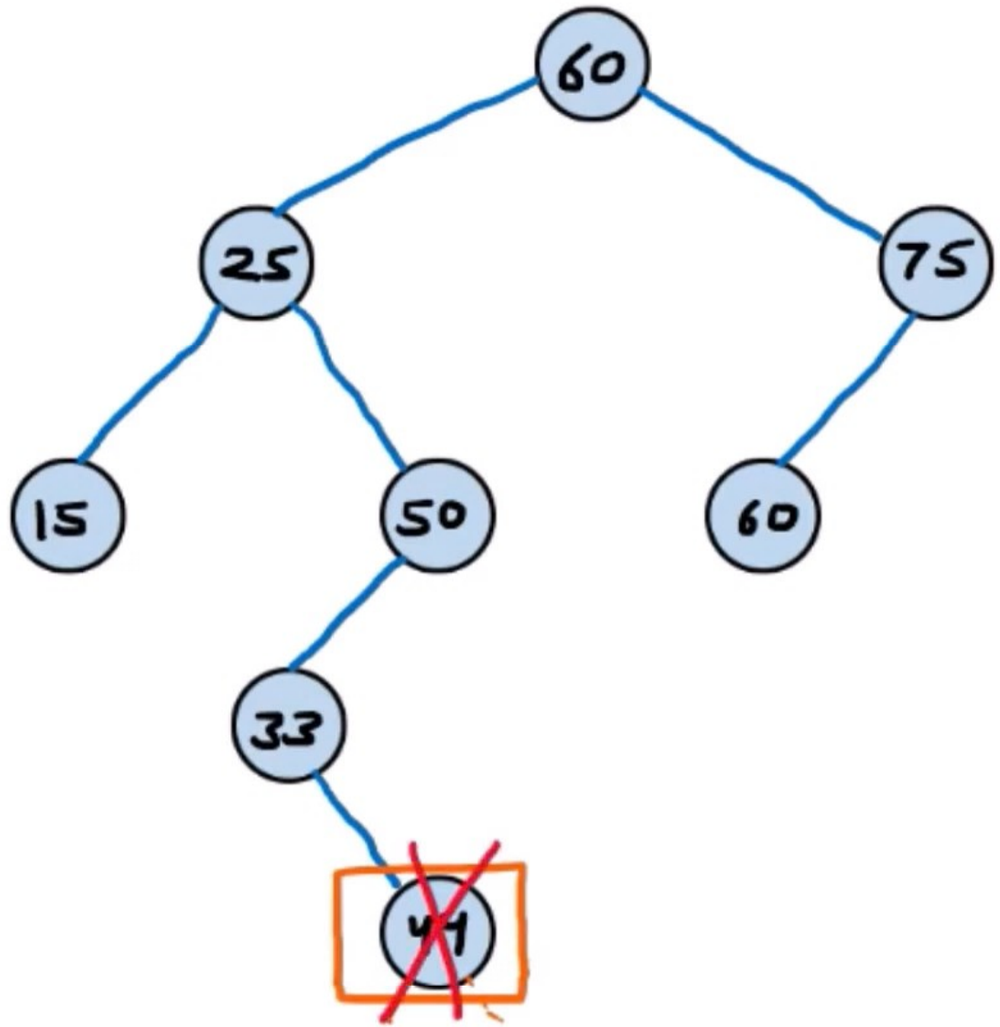


Deletion in Binary Search Tree (BST)

Three Cases:

Node has No Children

- Delete Node 44

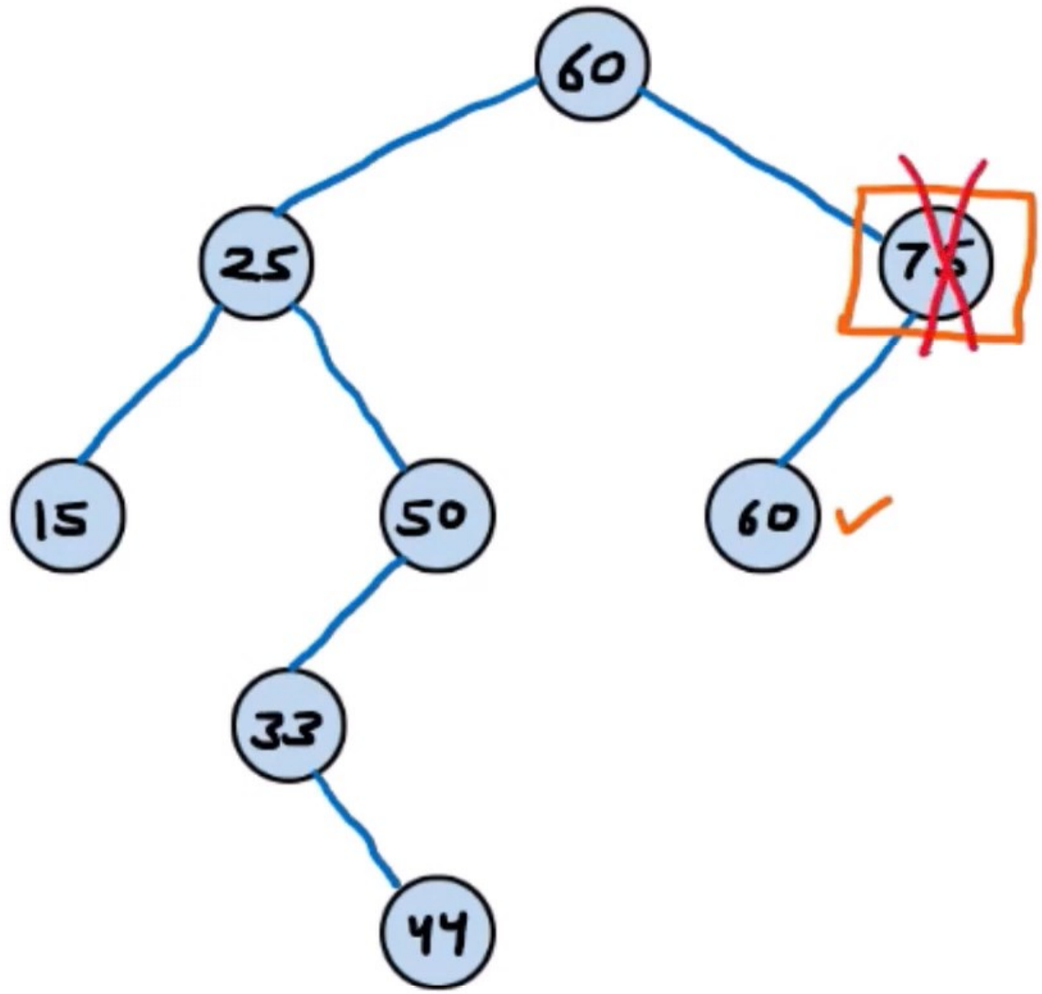


Deletion in Binary Search Tree (BST)

Three Cases:

Node has No Children
- Delete Node 44

Node has exactly One Child
- Delete Node 75

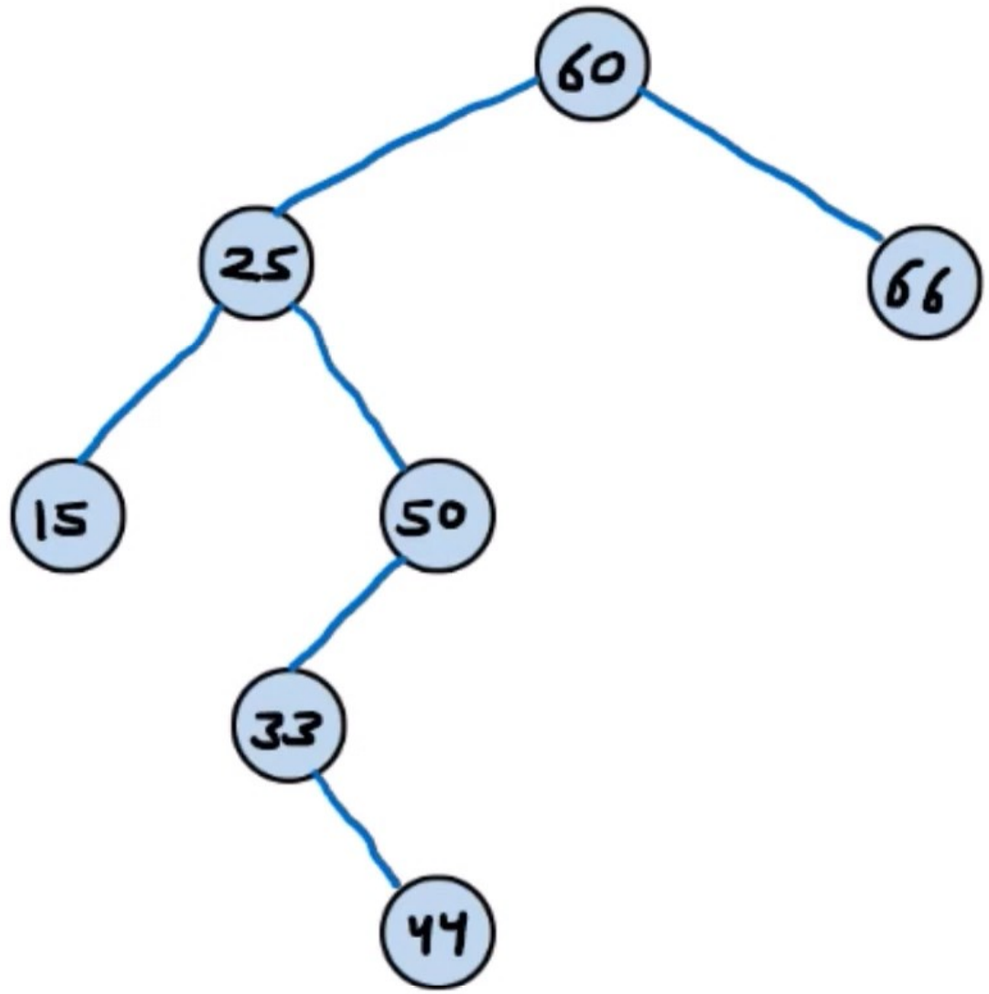


Deletion in Binary Search Tree (BST)

Three Cases:

Node has No Children
- Delete Node 44

Node has exactly One Child
- Delete Node 75



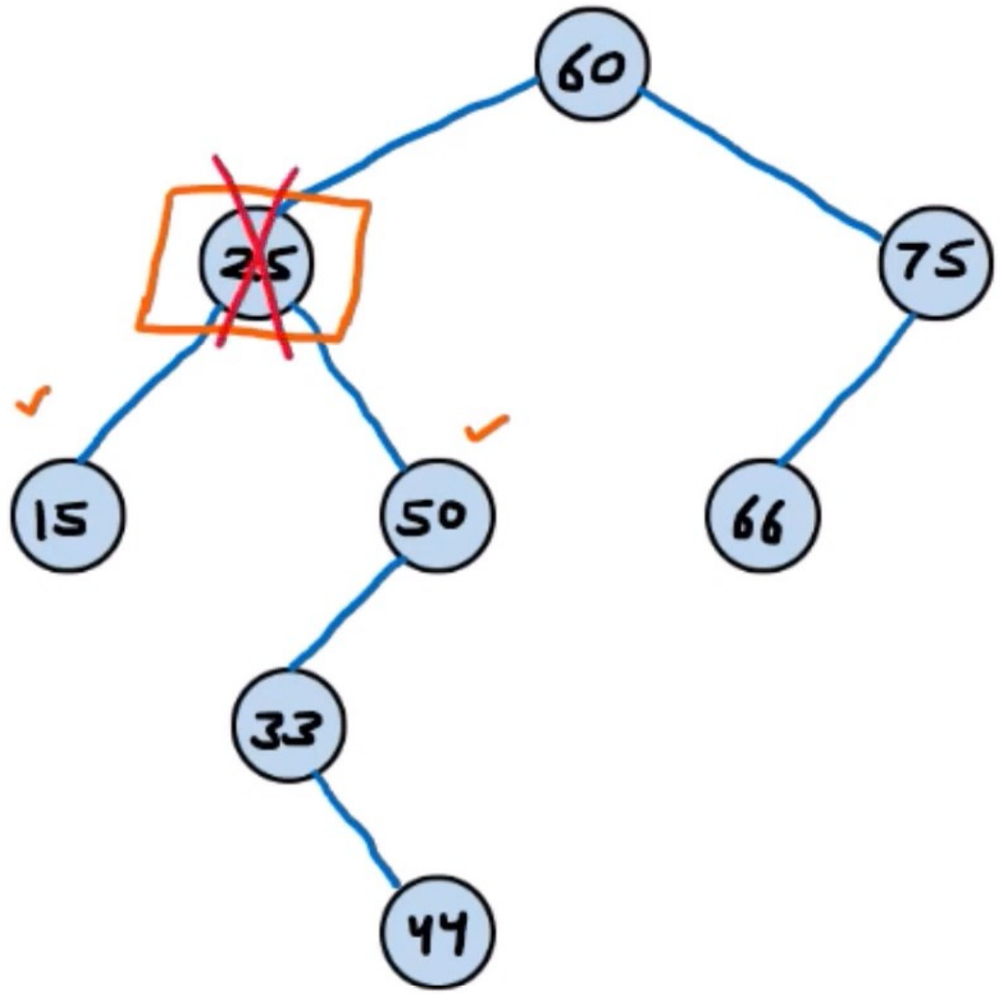
Deletion in Binary Search Tree (BST)

Three Cases:

Node has No Children
- Delete Node 44

Node has exactly One Child
- Delete Node 75

Node has Two Children
- Delet Node 25



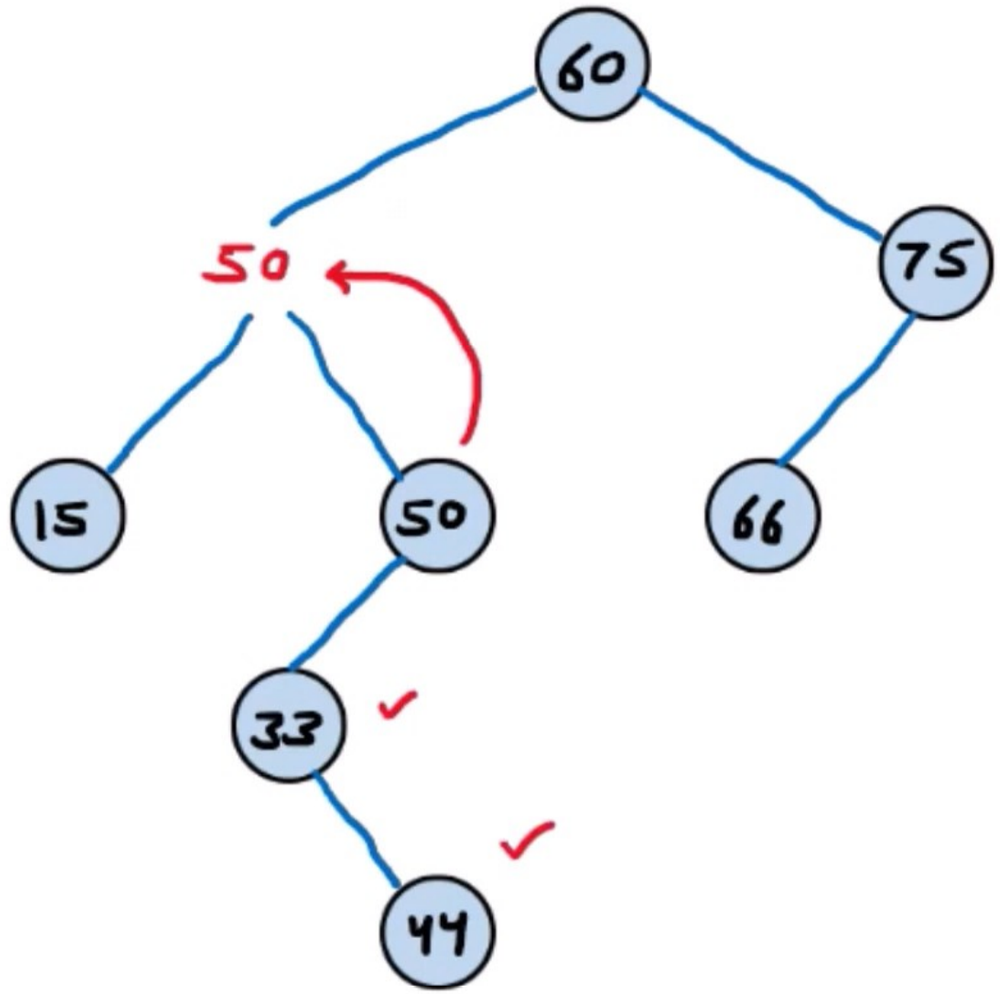
Deletion in Binary Search Tree (BST)

Three Cases:

Node has No Children
- Delete Node 44

Node has exactly One Child
- Delete Node 75

Node has Two Children
- Delet Node 25



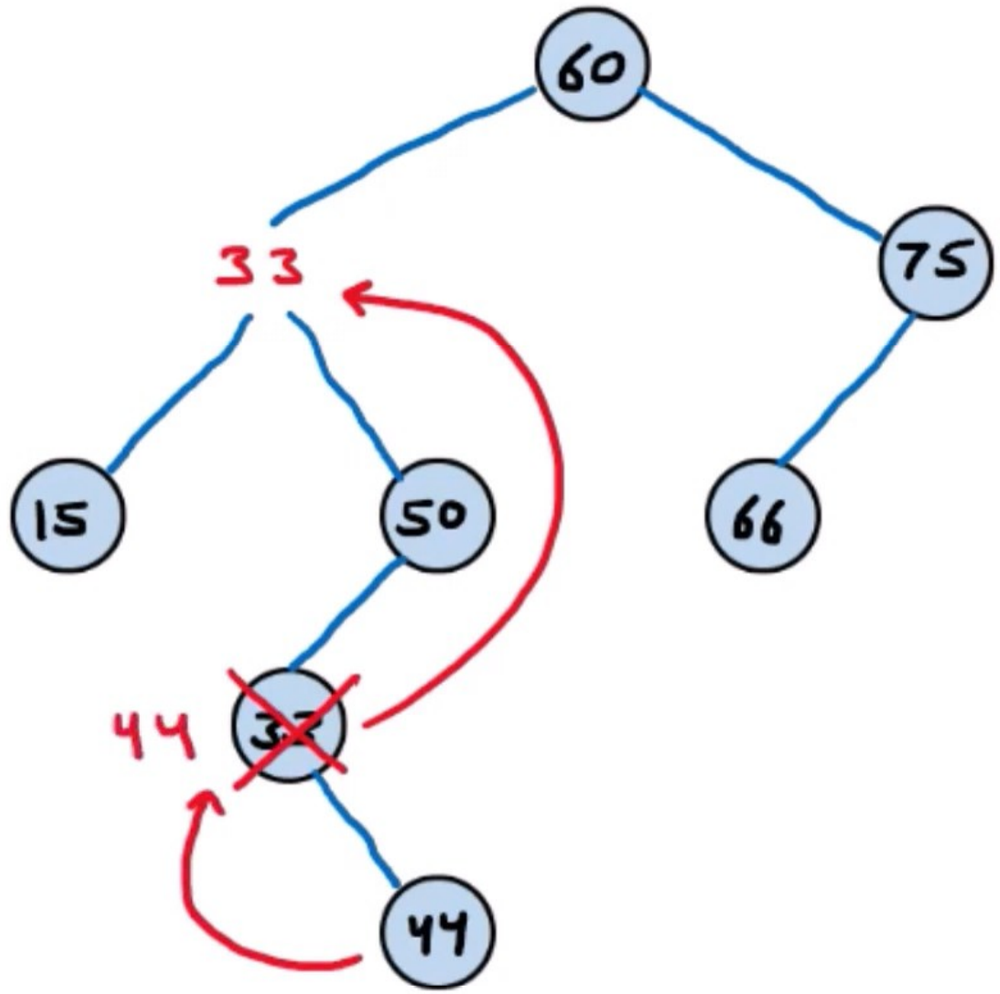
Deletion in Binary Search Tree (BST)

Three Cases:

Node has No Children
- Delete Node 44

Node has exactly One Child
- Delete Node 75

Node has Two Children
- Delet Node 25

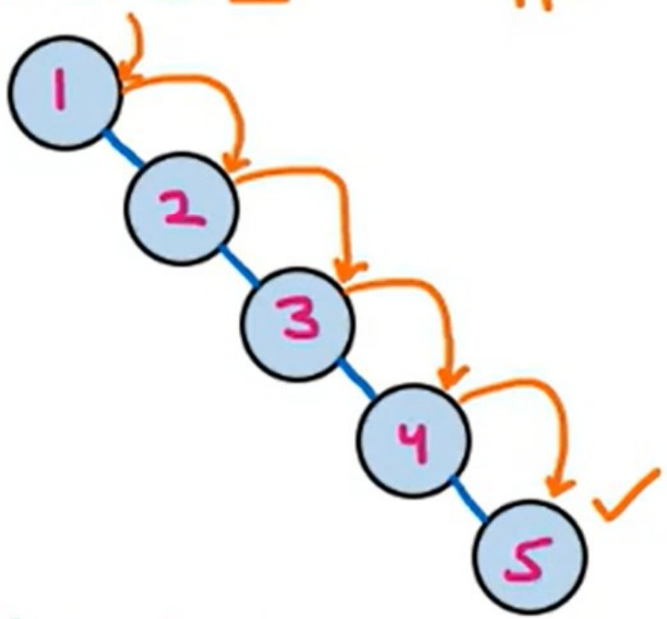


Skewed Binary Search Tree (BST)

Right Skewed ✓

1, 2, 3, 4, 5

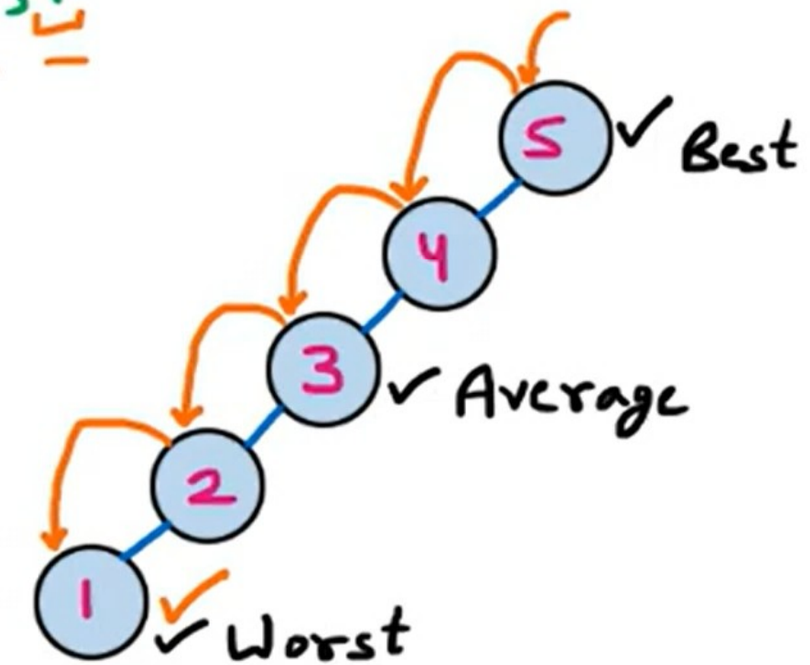
n ✓



Left Skewed ✓

5, 4, 3, 2, 1

n ✓



Disadvantages

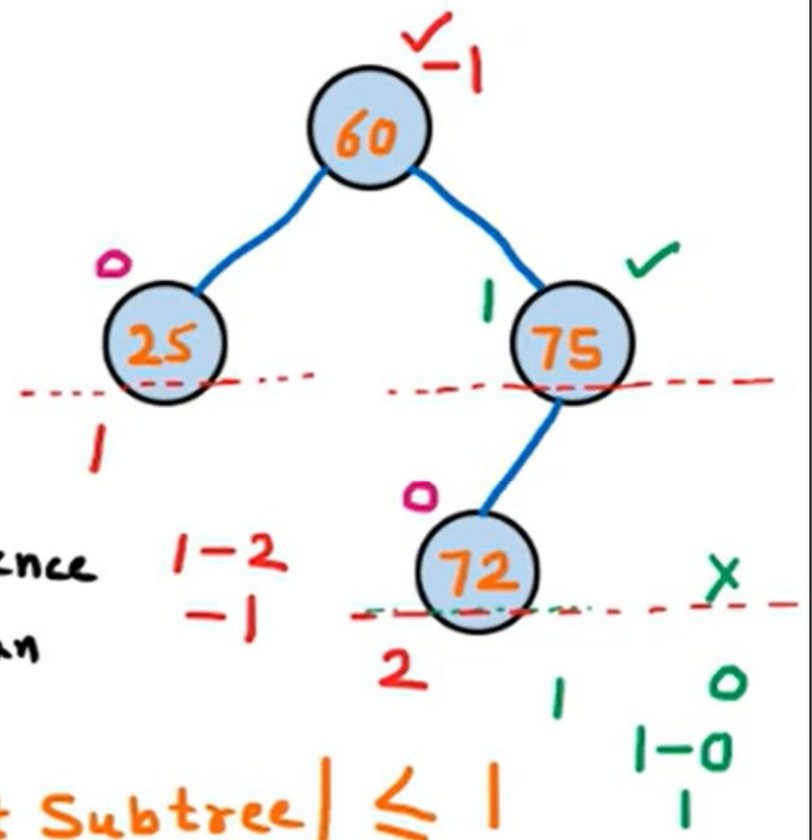
- Take more time for searching in Worst Case
- Worst case Time Complexity is O(n)

Solution

AVL Tree

Adelson-Velsky and Landis (AVL)

- Most popular Balanced Tree
- Takes less time for searching in Worst case
- Worst case Time Complexity of Search is $O(\log n)$
- AVL Tree is Binary Search Tree where difference of Left Subtree & Right Subtree of any Node can not be more than 1



$$|\text{Height of Left Subtree} - \text{Height of Right Subtree}| \leq 1$$

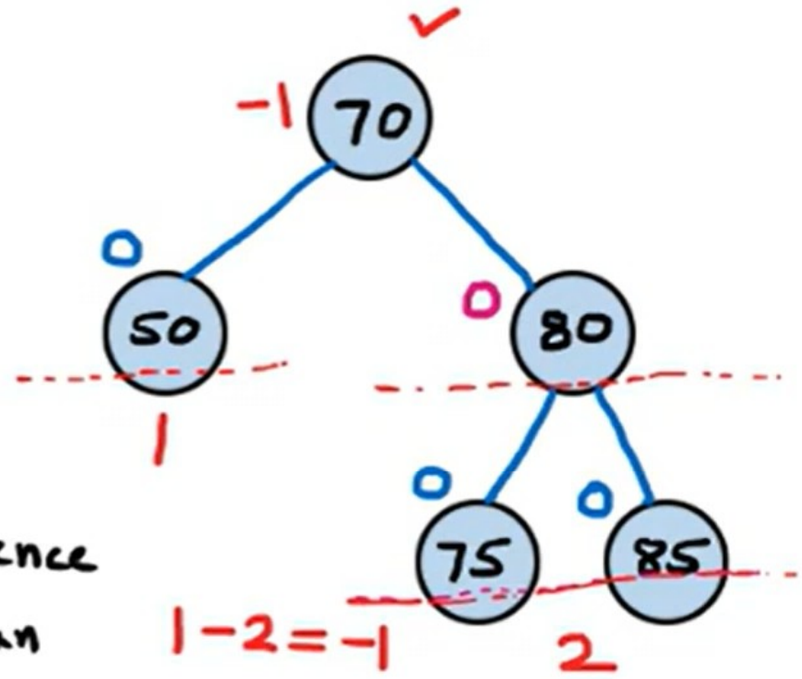
$$|h(T^L) - h(T^R)| \leq 1 \qquad h(T^L) - h(T^R) = 0, 1, -1$$

Balancing Factor

- Balancing factor of AVL Tree can be either 0, 1 or -1

Adelson-Velsky and Landis (AVL)

- Most popular Balanced Tree
- Takes less time for searching in Worst case
- Worst case Time Complexity of Search is O(log n)
- AVL Tree is Binary Search Tree where difference of Left Subtree & Right Subtree of any Node can not be more than 1



$$|\text{Height of Left Subtree} - \text{Height of Right Subtree}| \leq 1$$

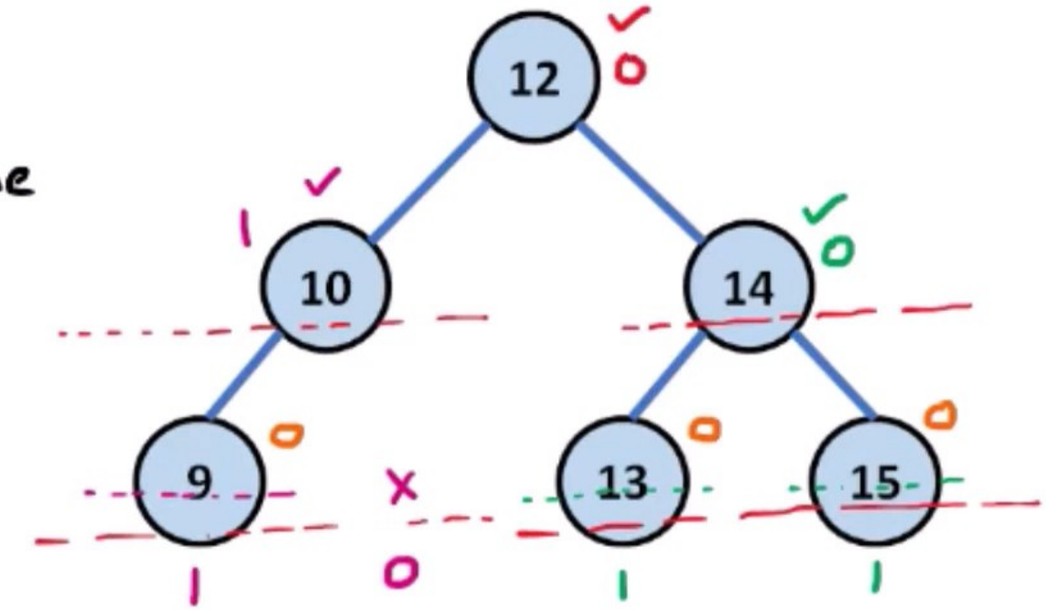
$$|h(T^L) - h(T^R)| \leq 1 \qquad h(T^L) - h(T^R) = 0, 1, -1$$

Balancing Factor

- Balancing factor of AVL Tree can be either 0, 1 or -1

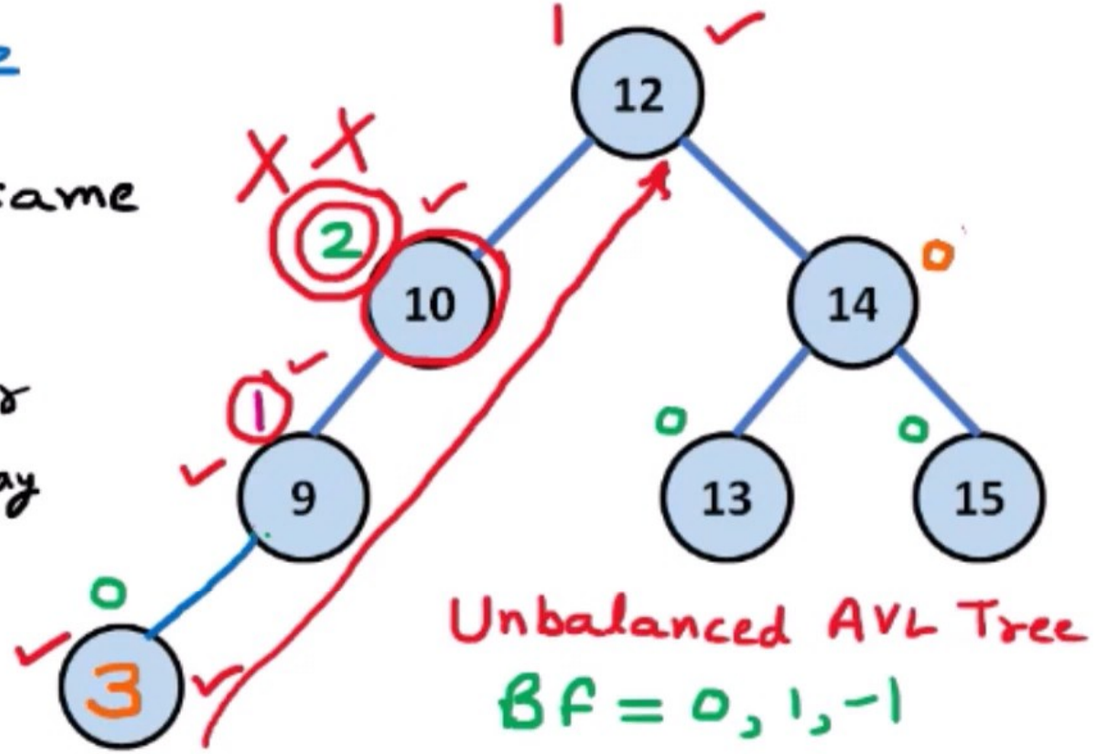
Insertion in AVL Tree

- Inserting element in AVL is same as BST
- After Insertion, Balancing factor of any Node may affect & Tree may become Unbalanced



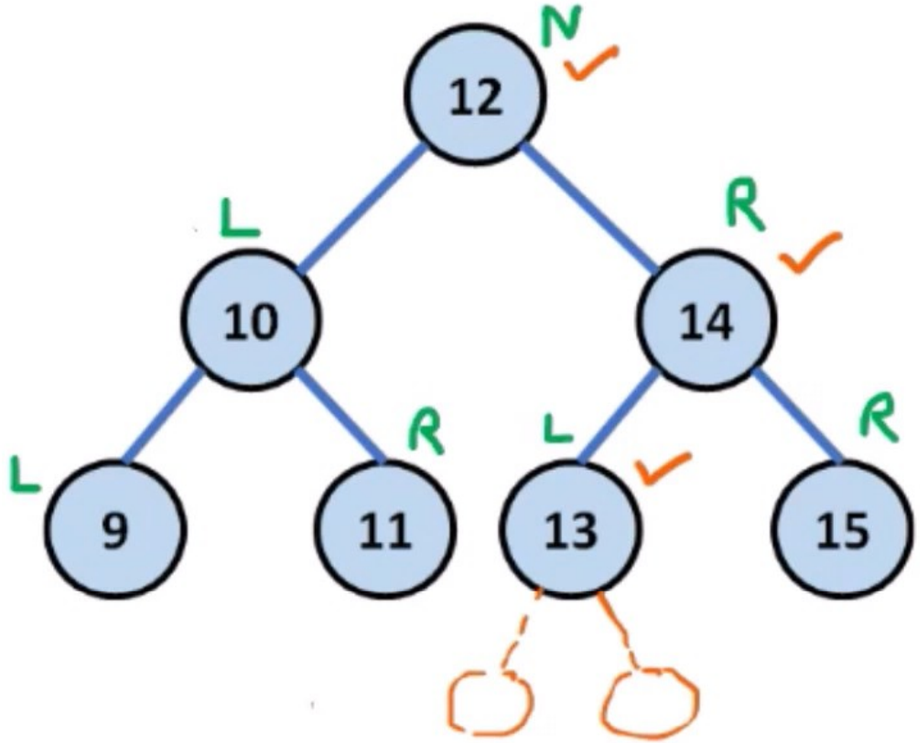
Insertion in AVL Tree

- Inserting element in AVL is same as BST
- After Insertion, Balancing factor of any Node may affect & Tree may become Unbalanced
- Rotations are made on Unbalanced Tree to restore Balanced Tree
- To perform Rotation, Node is identified whose Balancing factor (BF) is neither 0, 1 or -1 and is nearest ansestor to Inserted Node on path from Inserted Node to Root



Insertion in AVL Tree

AVL Rotations while Insertion



LL Rotation

- Inserted Node is in Left Subtree of Left Subtree of Node

RR Rotation

- Inserted Node is in Right Subtree of Right Subtree of Node

LR Rotation

- Inserted Node is in Right Subtree of Left Subtree of Node

RL Rotation

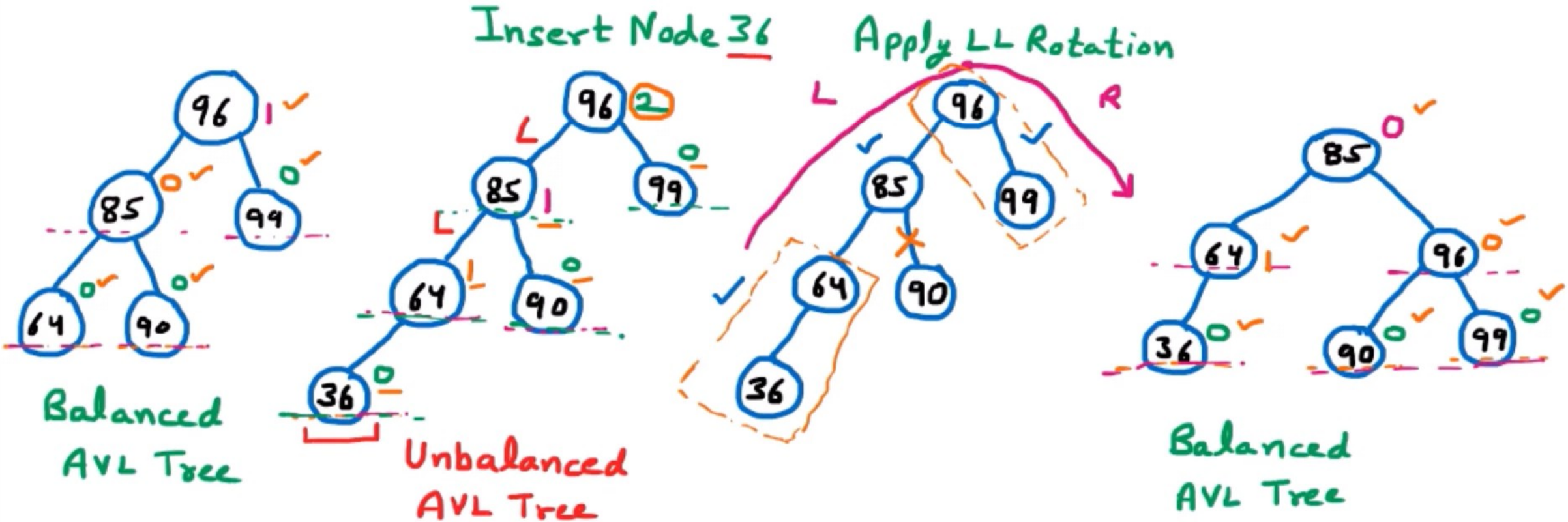
- Inserted Node is in Left Subtree of Right Subtree of Node

Rotations while Insertion in AVL Tree

LL Rotation

BF = 0, 1, -1

Inserted Node is in Left Subtree of Left Subtree

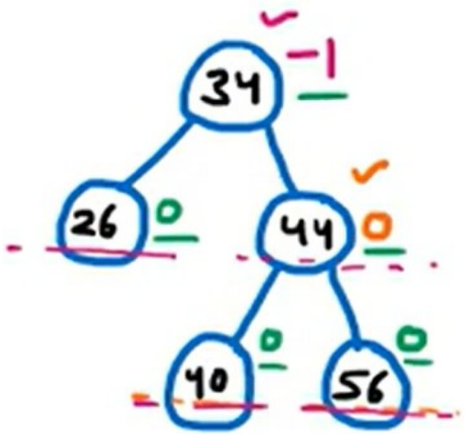


Rotations while Insertion in AVL Tree

RR Rotation

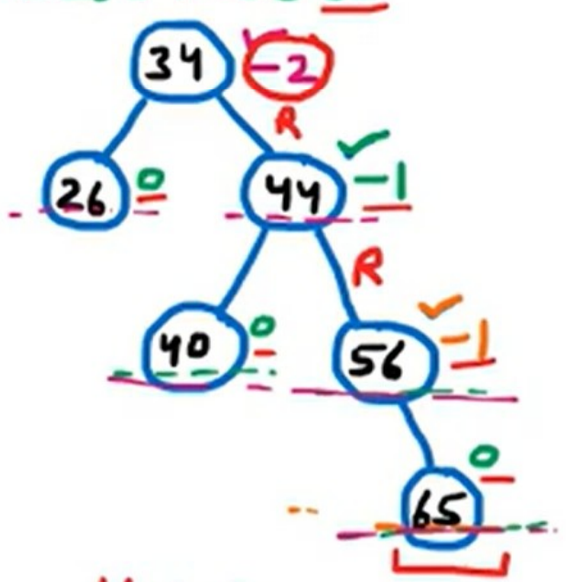
BF = 0, 1, -1

Inserted Node is in Right Subtree of Right Subtree



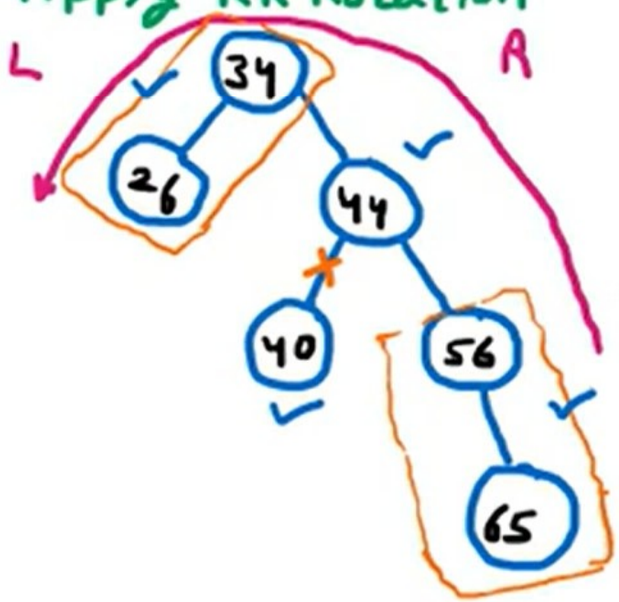
Balanced AVL Tree

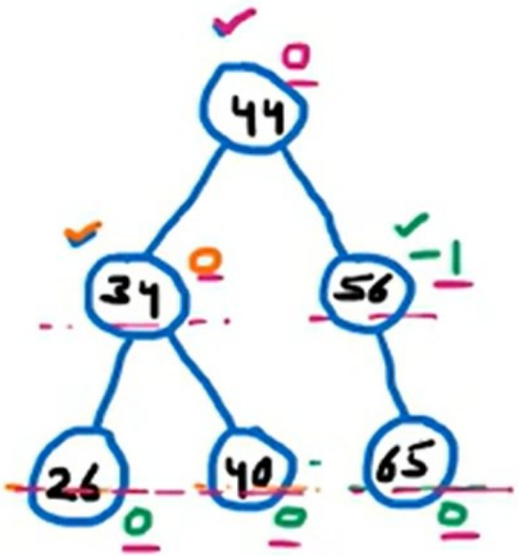
Insert Node 65



Unbalanced AVL Tree

Apply RR Rotation





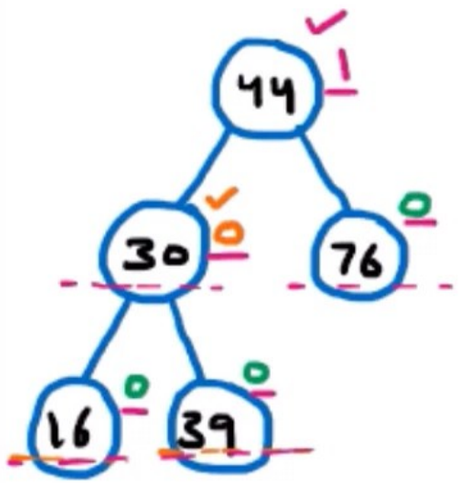
Balanced AVL Tree

Rotations while Insertion in AVL Tree

LR Rotation

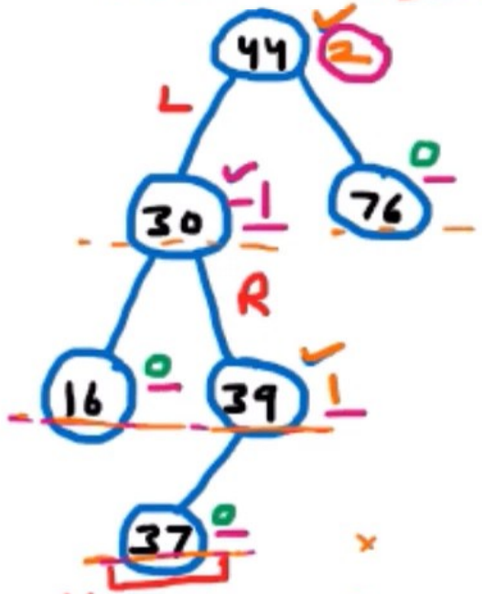
BF = 0, 1, -1

- Inserted Node is in Right Subtree of Left Subtree
- Double Rotation (RR Rotation followed by LL Rotation)



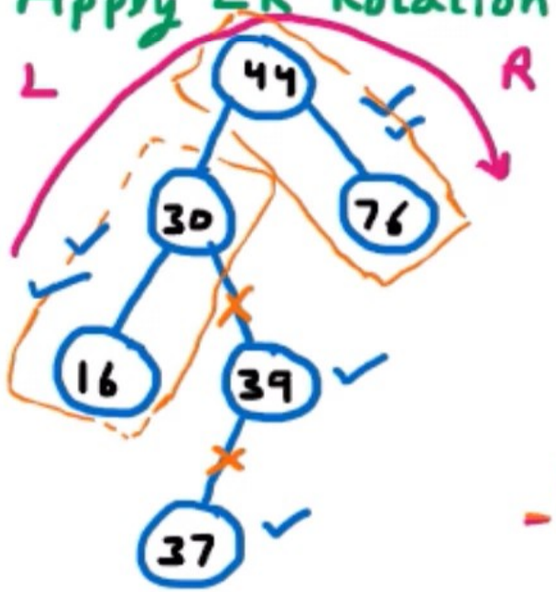
Balanced AVL Tree

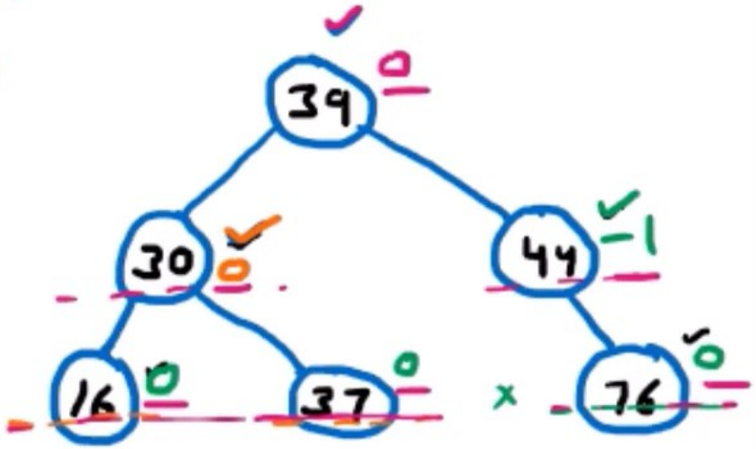
Insert Node 37



Unbalanced AVL Tree

Apply LR Rotation



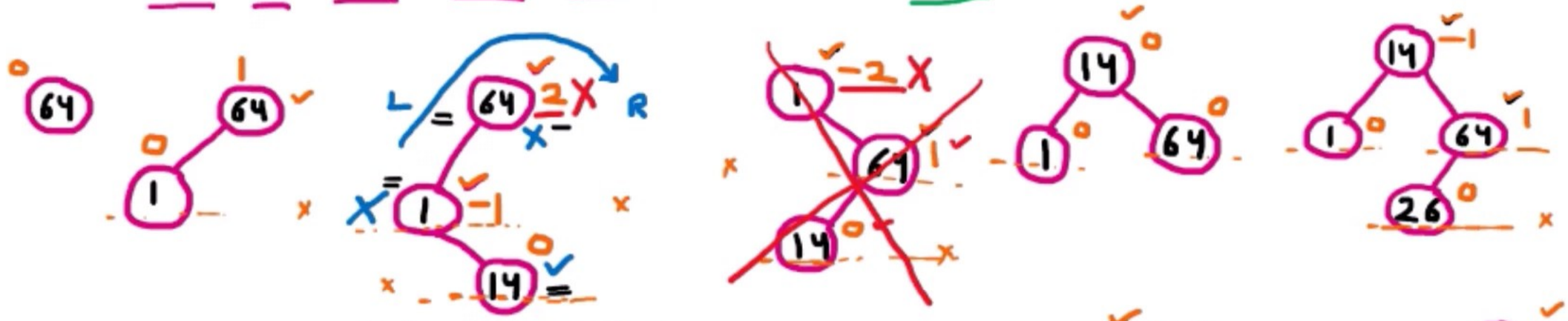


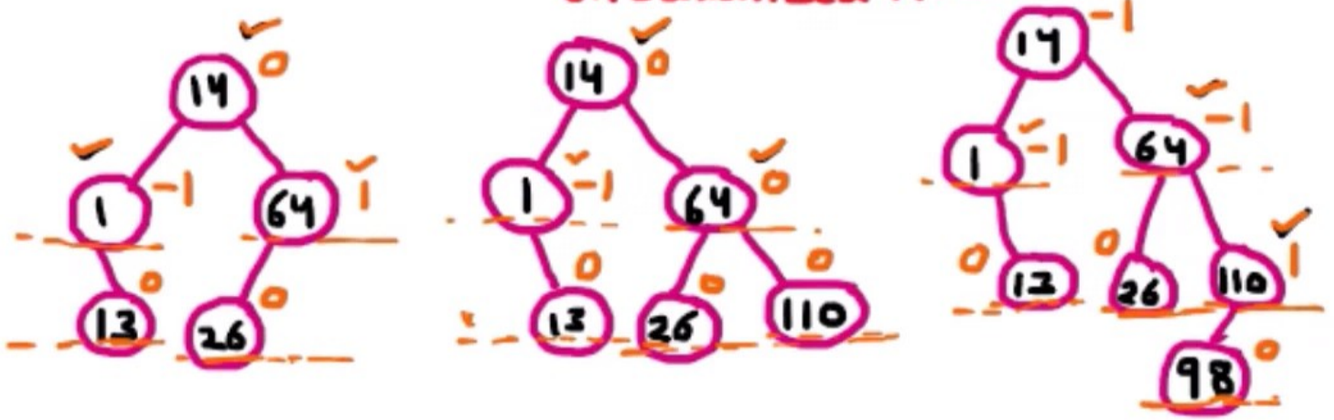
Balanced AVL Tree

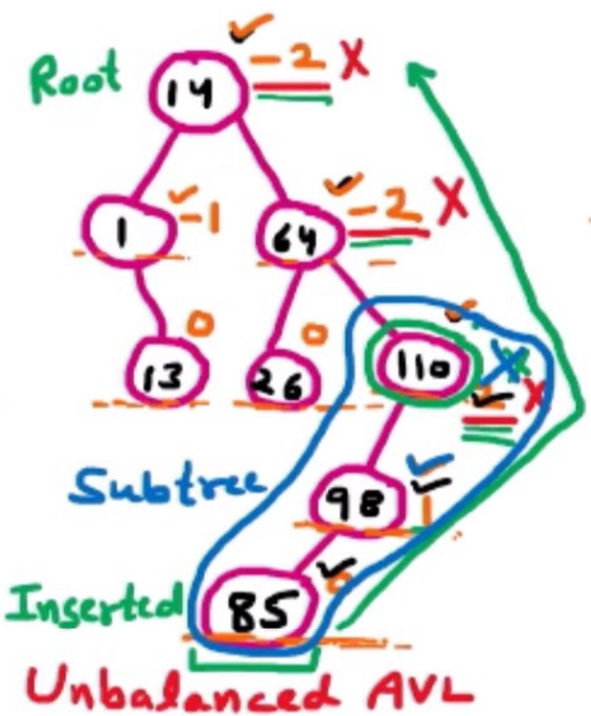
Construct AVL Tree from following Elements

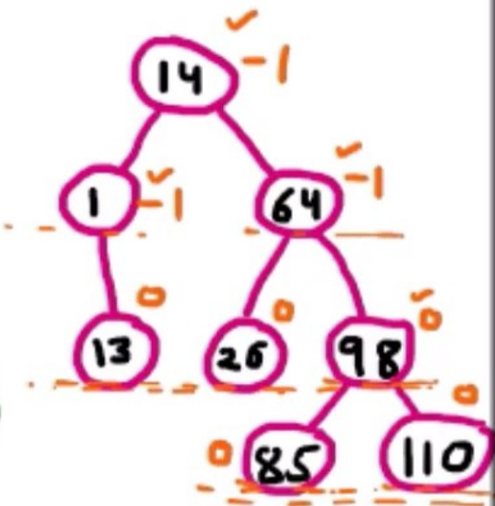
64, 1, 14, 26, 13, 110, 98, 85

BF = 0, 1, -1



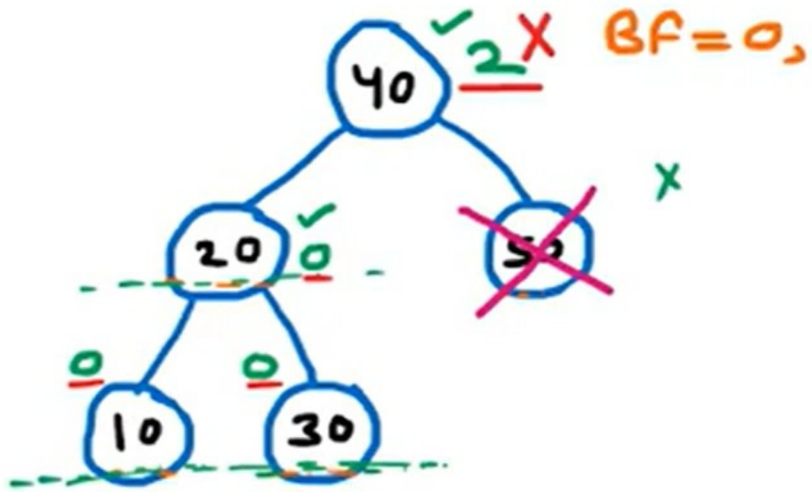






Deletion in AVL Tree

- After Deletion, Balancing factor of any Node may affect & Tree may become Unbalanced

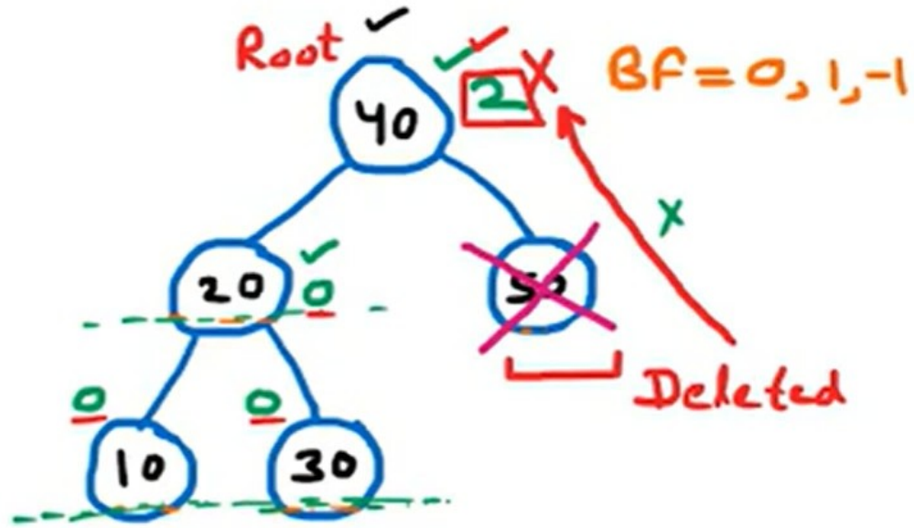


~~Balanced AVL Tree~~

Delete Node 50

Deletion in AVL Tree

- After Deletion, Balancing factor of any Node may affect & Tree may become Unbalanced
- Rotations are made on Unbalanced Tree to restore Balanced Tree
- To perform Rotation, Node is identified whose BF is neither 0, 1 or -1 and which is nearest ancestor to Deleted Node on Path from Deleted Node to Root



Unbalanced AVL Tree

Delete Node 50

Deletion in AVL Tree

AVL Rotations while Deletion

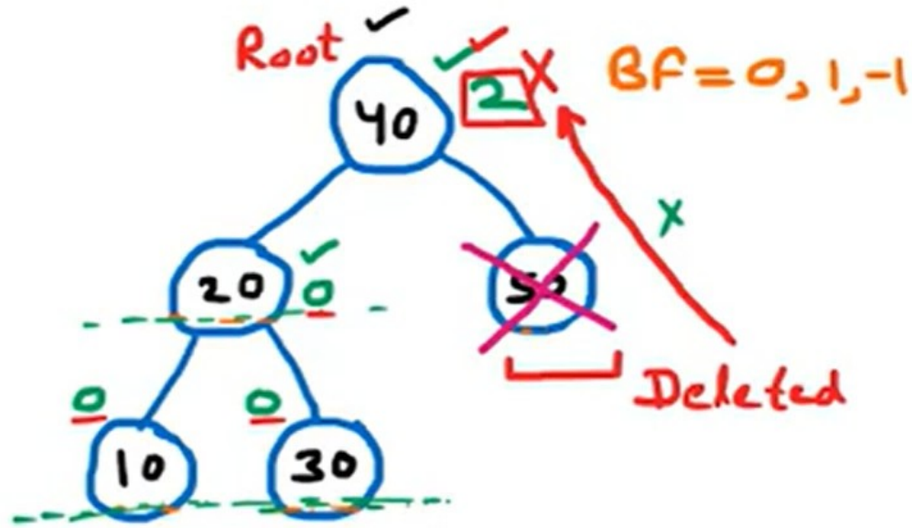
- Rotations are classified L or R depend on whether Deletion occur at Left or Right Subtree
- Depending on Value of BF(N) where N is Root of Left or Right Subtree

L Rotations are further Classified

L_0, L_1, L_{-1}

R Rotations are further Classified

R_0, R_1, R_{-1}



Unbalanced AVL Tree

Delete Node 50

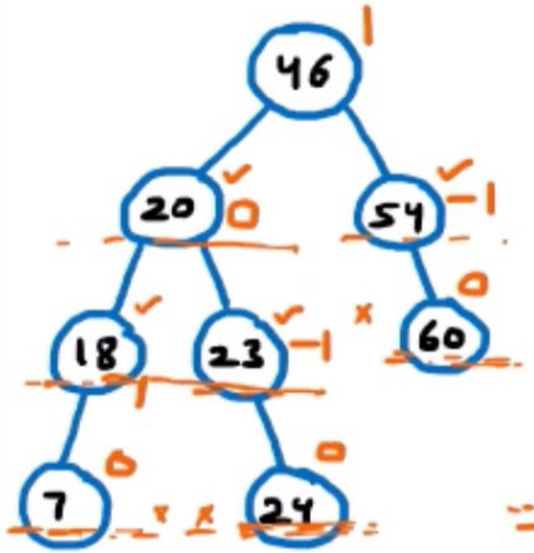
Deletion in AVL Tree

AVL Rotations while Deletion

BF = 0, 1, -1

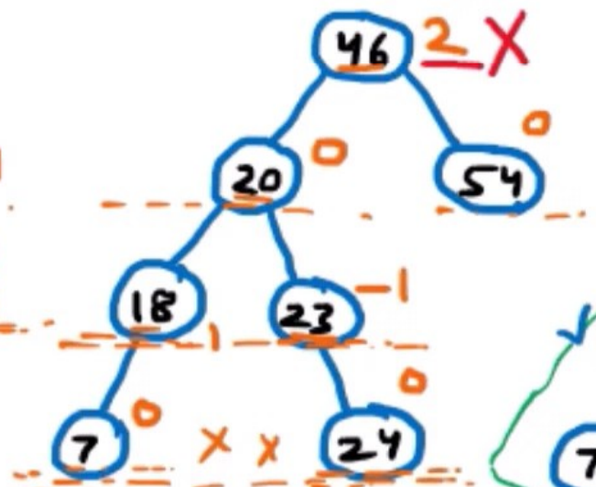
R_0 Rotation

If BF(N) = 0, then R_0 Rotation is applied

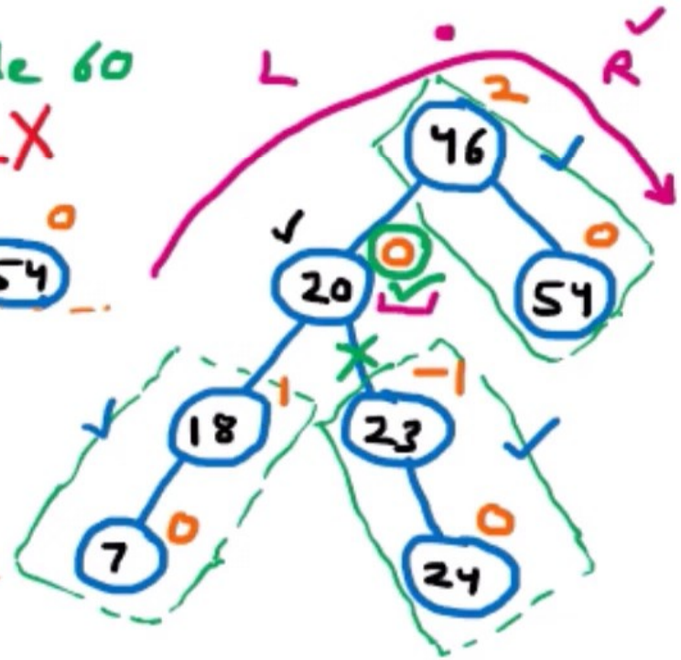


Balanced AVL Tree

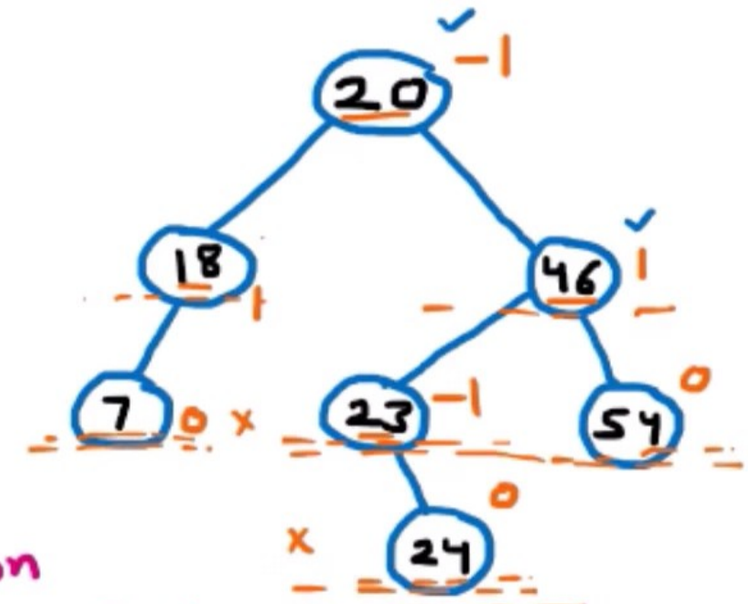
Delete Node 60



Unbalanced AVL Tree



Apply R_0 Rotation



Balanced AVL Tree

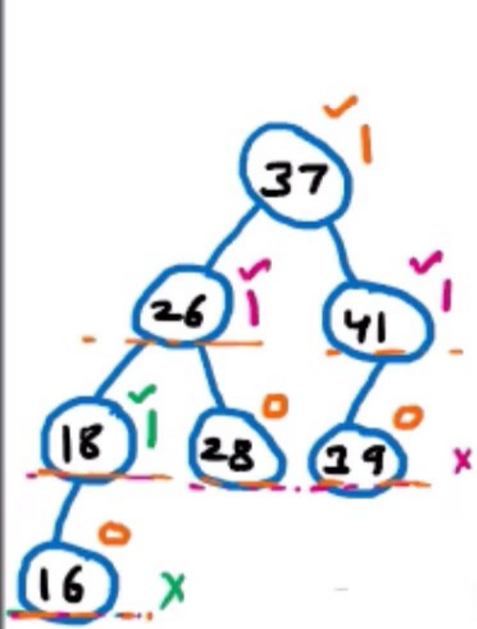
Deletion in AVL Tree

AVL Rotations while Deletion

RI Rotation

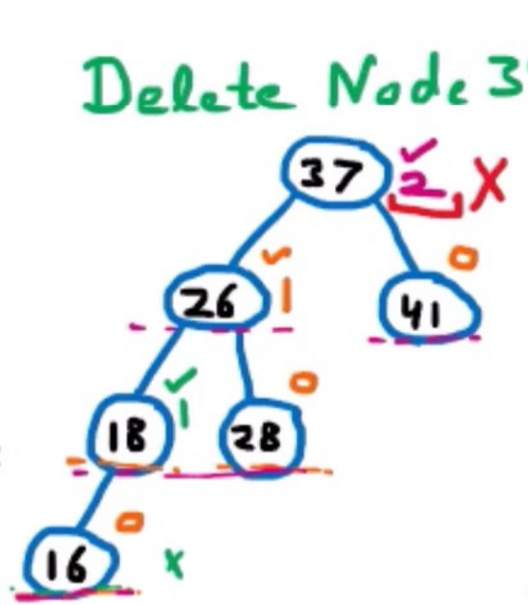
If Bf(N)=1, then RI Rotation is applied

Bf=0, 1, -1

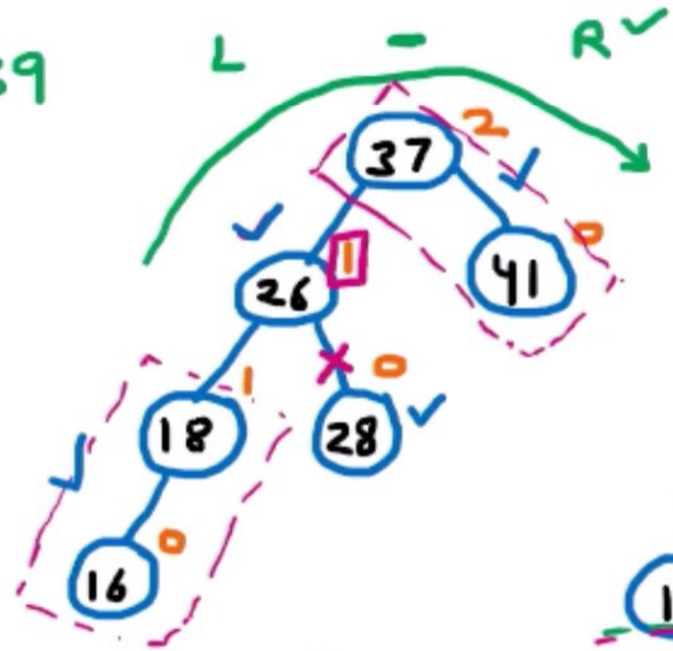


Balanced AVL Tree

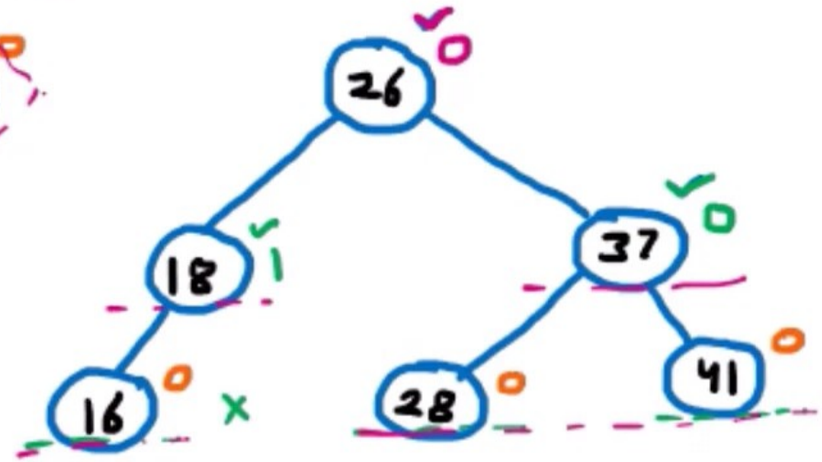
Delete Node 39



Unbalanced AVL Tree



Apply RI Rotation



Balanced AVL Tree

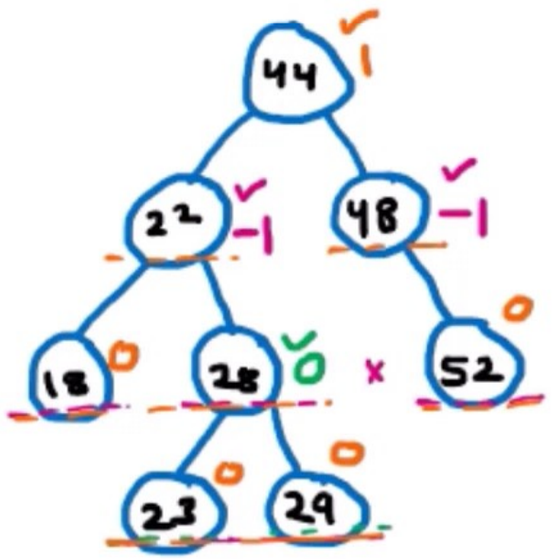
Deletion in AVL Tree

AVL Rotations while Deletion

BF = 0, 1, -1 ✓

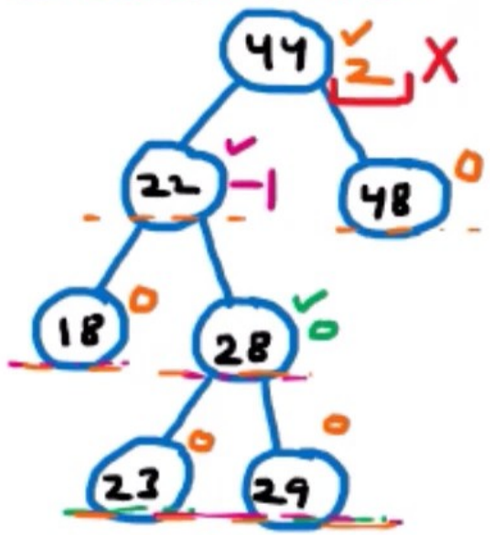
R-1 Rotation

If BF(N) = -1 then R-1 Rotation is applied

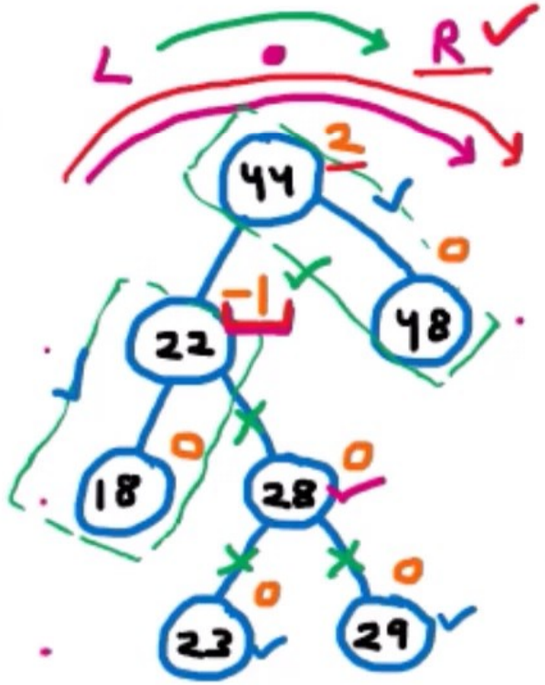


Balanced AVL Tree

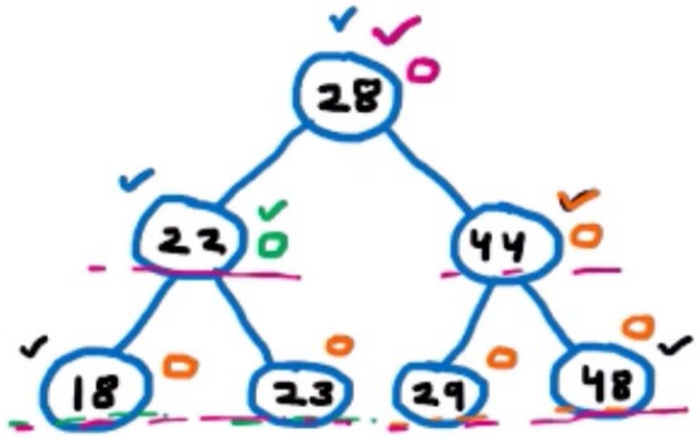
Delete Node 52



Unbalanced AVL Tree



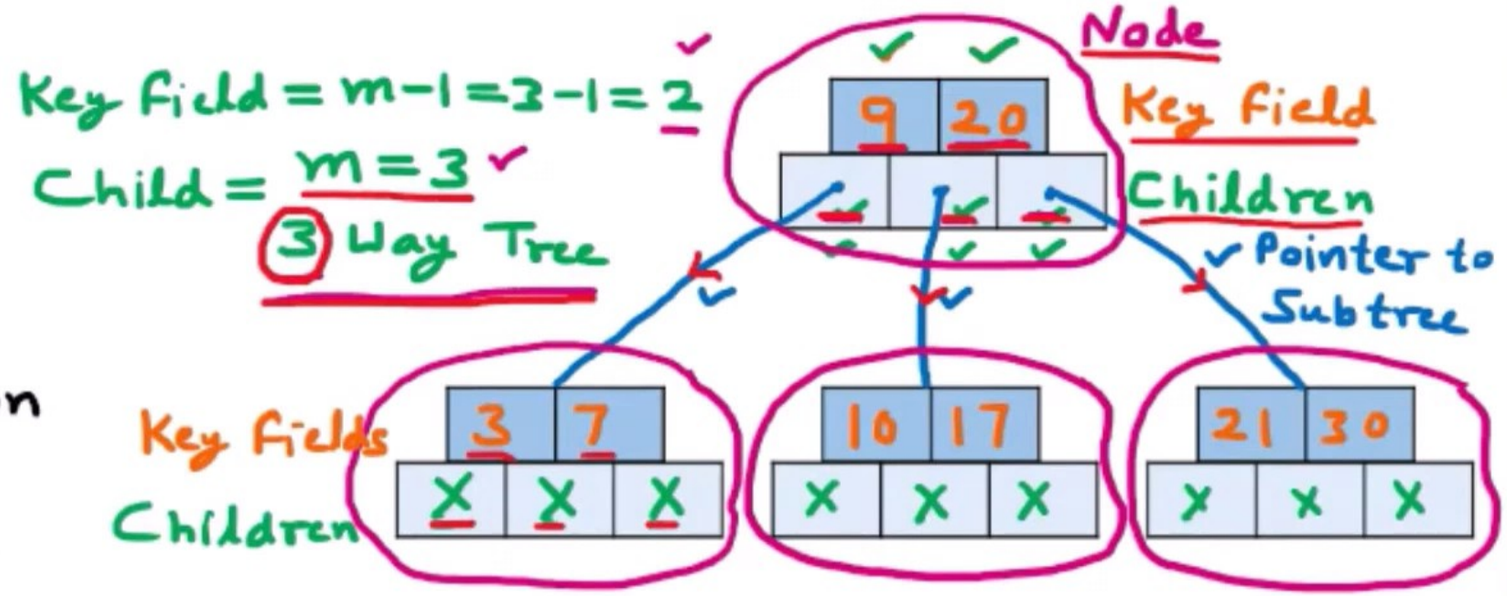
Apply R-1 Rotation



Balanced AVL Tree

m Way Tree

(Multi Way Tree)



- It is generalized version of Binary Tree
- It can have more than 2 Children
- It is of order m which means each node has maximum m child

  3 Way Tree → Maximum 3 Child m=3 m-1=3-1=2 2 Key fields

  5 Way Tree → Maximum 5 Child m=5 m-1=5-1=4 4 Key fields

- Each node has maximum m-1 key fields

m Way Tree

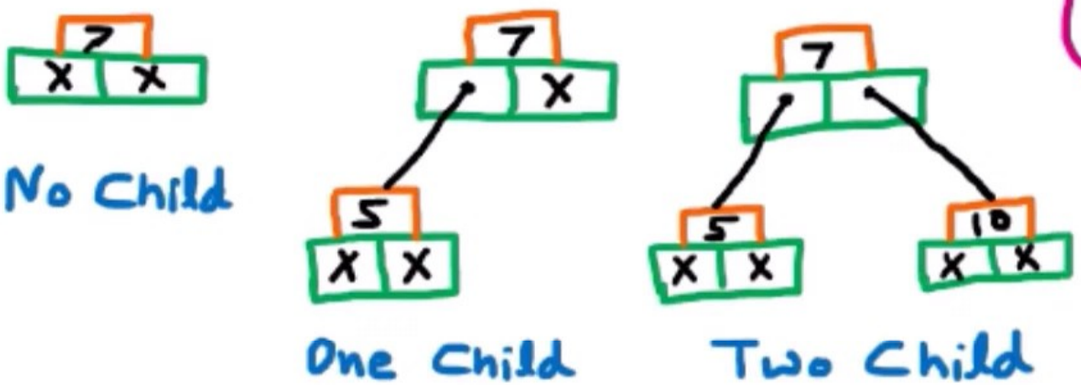
(Multi Way Tree)

2 Way Tree

$m = 2$

Maximum Child $= m = 2$

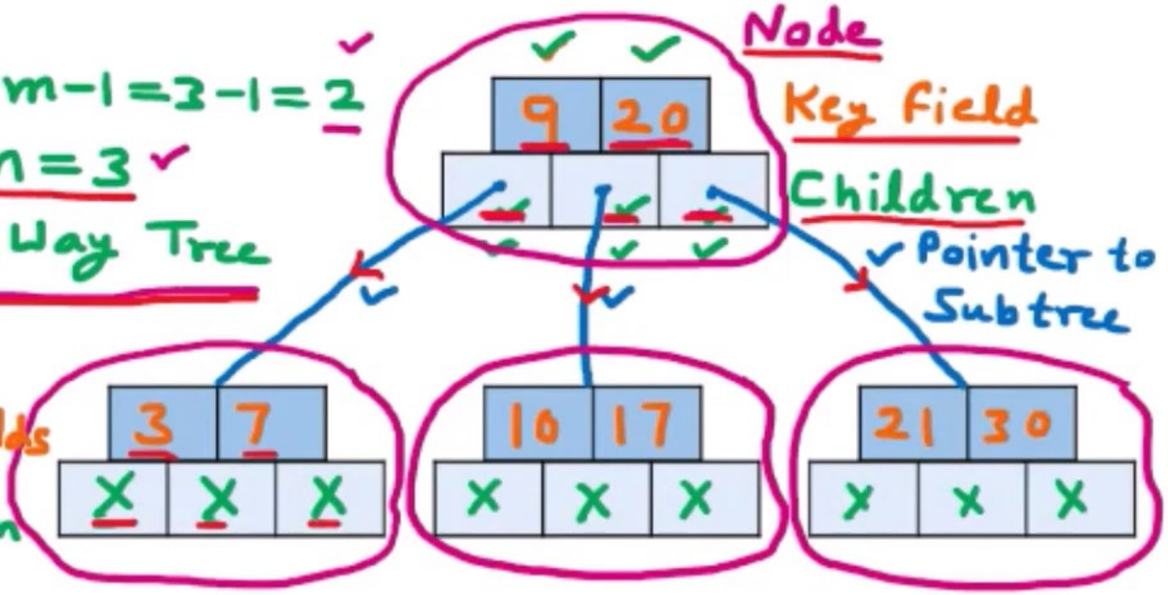
Key Field $= m - 1 = 2 - 1 = 1$



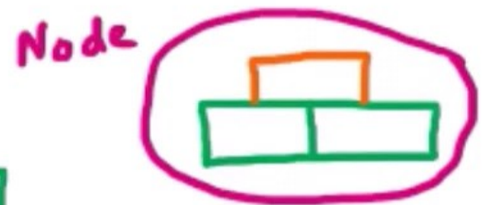
Key Field $= m - 1 = 3 - 1 = 2$

Child $= m = 3$

3 Way Tree



X Null Pointer



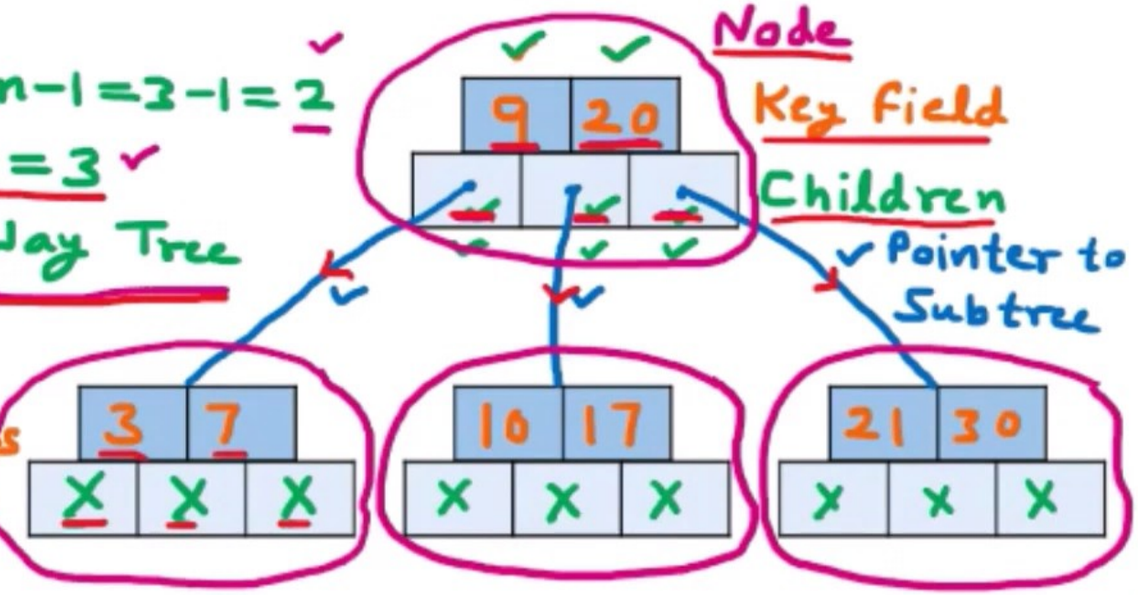
m Way Tree

(Multi Way Tree)

Key Field = m-1 = 3-1 = 2

Child = m = 3

3 Way Tree

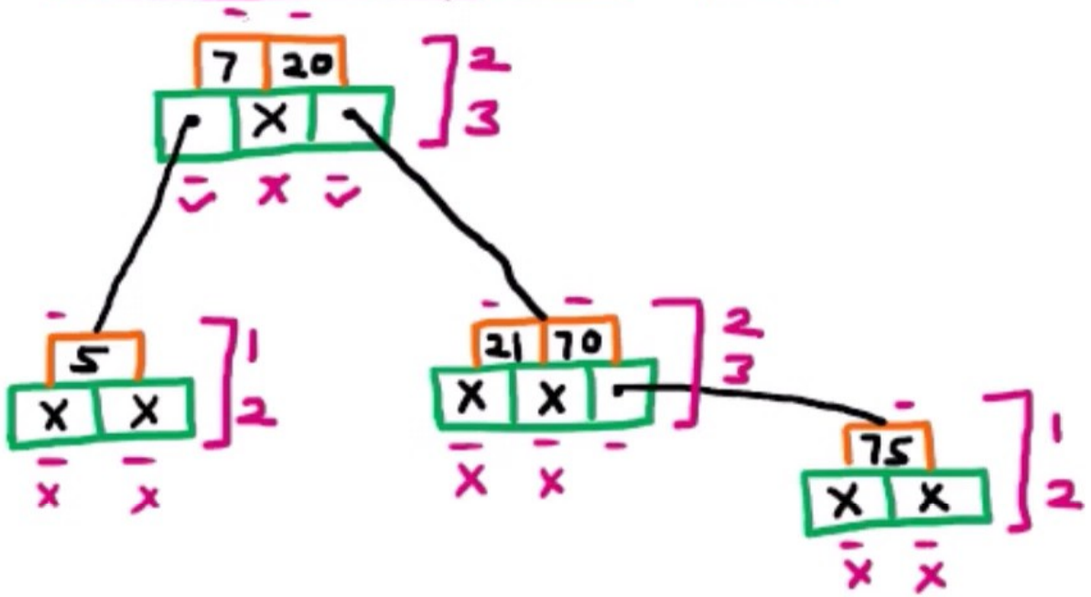


3 Way Tree

m = 3

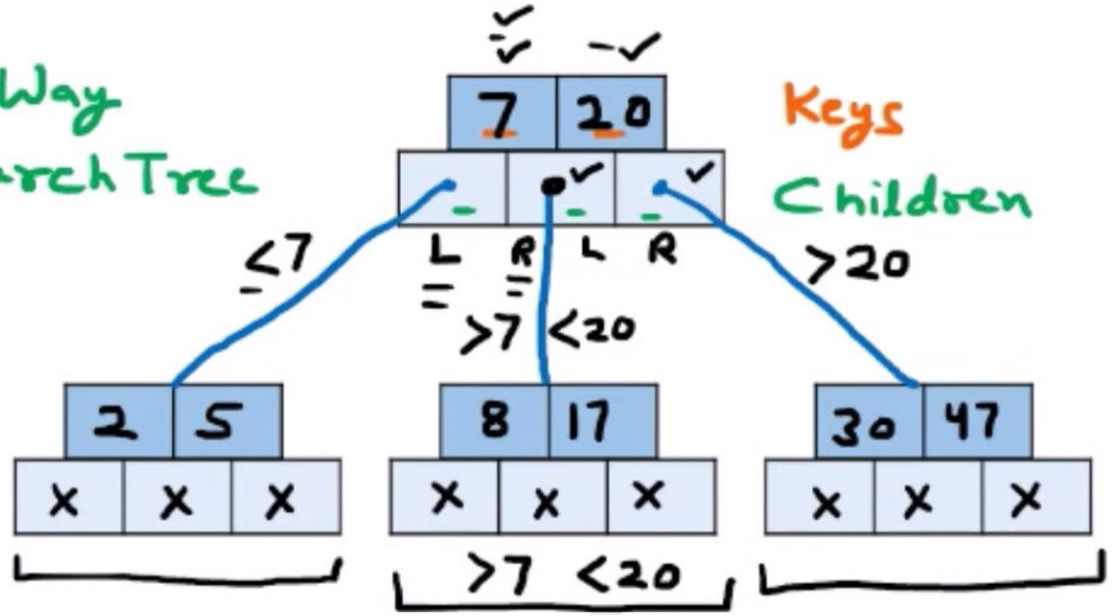
Maximum Child = m = 3 3, 2, 1, 0

Maximum Key Fields = m-1 = 3-1 = 2



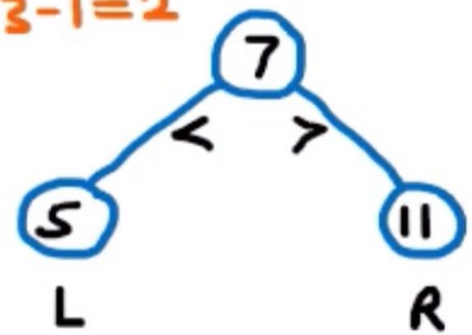
m Way Search Tree

- Generalized version of Binary Search Tree (BST)
- Keys in each node are arranged in Ascending Order
- Keys in the Left Subtree having Less Value
- Keys in the Right Subtree having Grerater Value



Children = m = 3

Key = m-1 = 3-1 = 2



m Way search tree of n elements has Maximum Height n & Minimum Height $\log_m(n+1)$

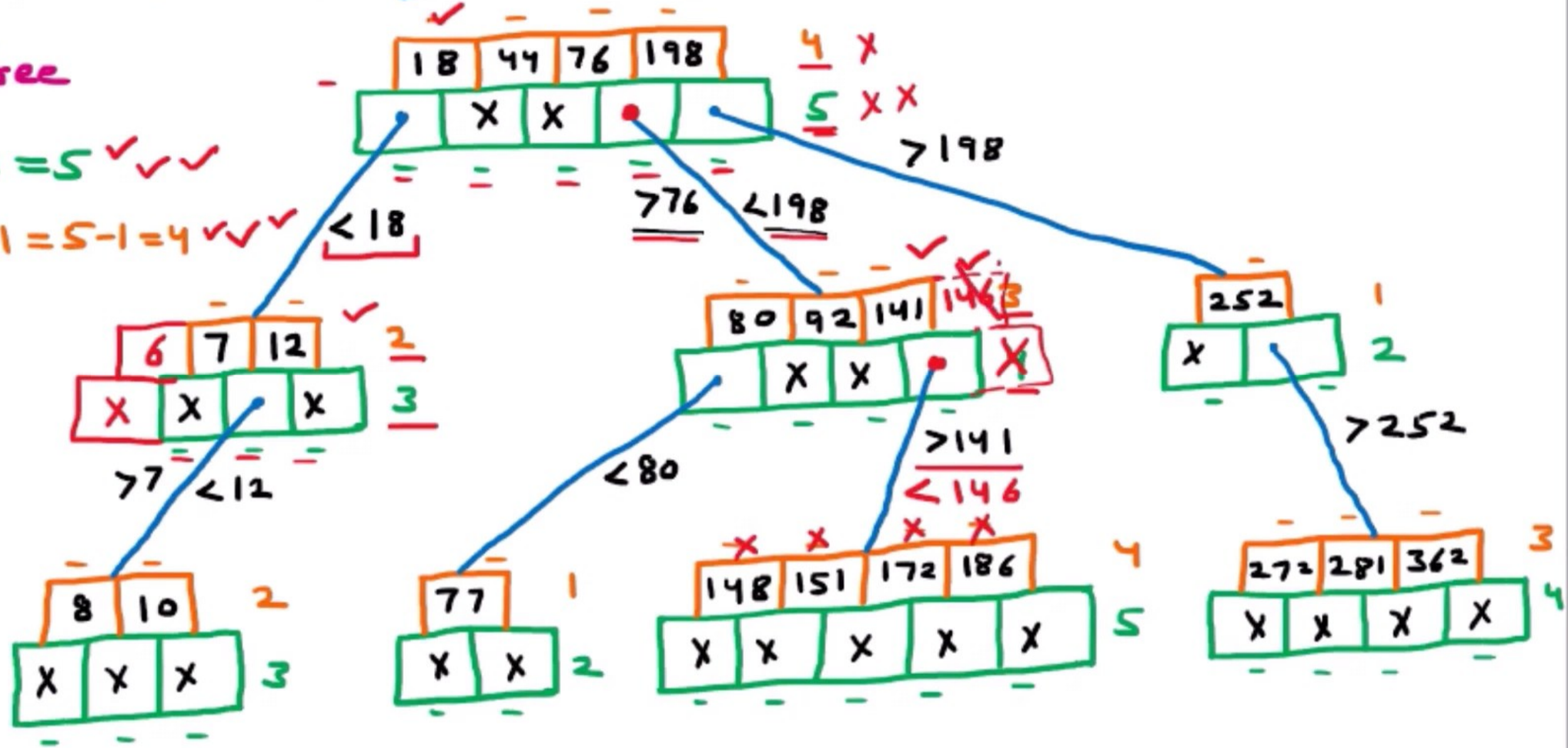
m Way Search tree of height h has Maximum Elements $m^h - 1$ & Minimum Elements h

Insertion in m Way Tree

5 Way Tree

Child = m = 5

Keys = m - 1 = 5 - 1 = 4



Insert 6

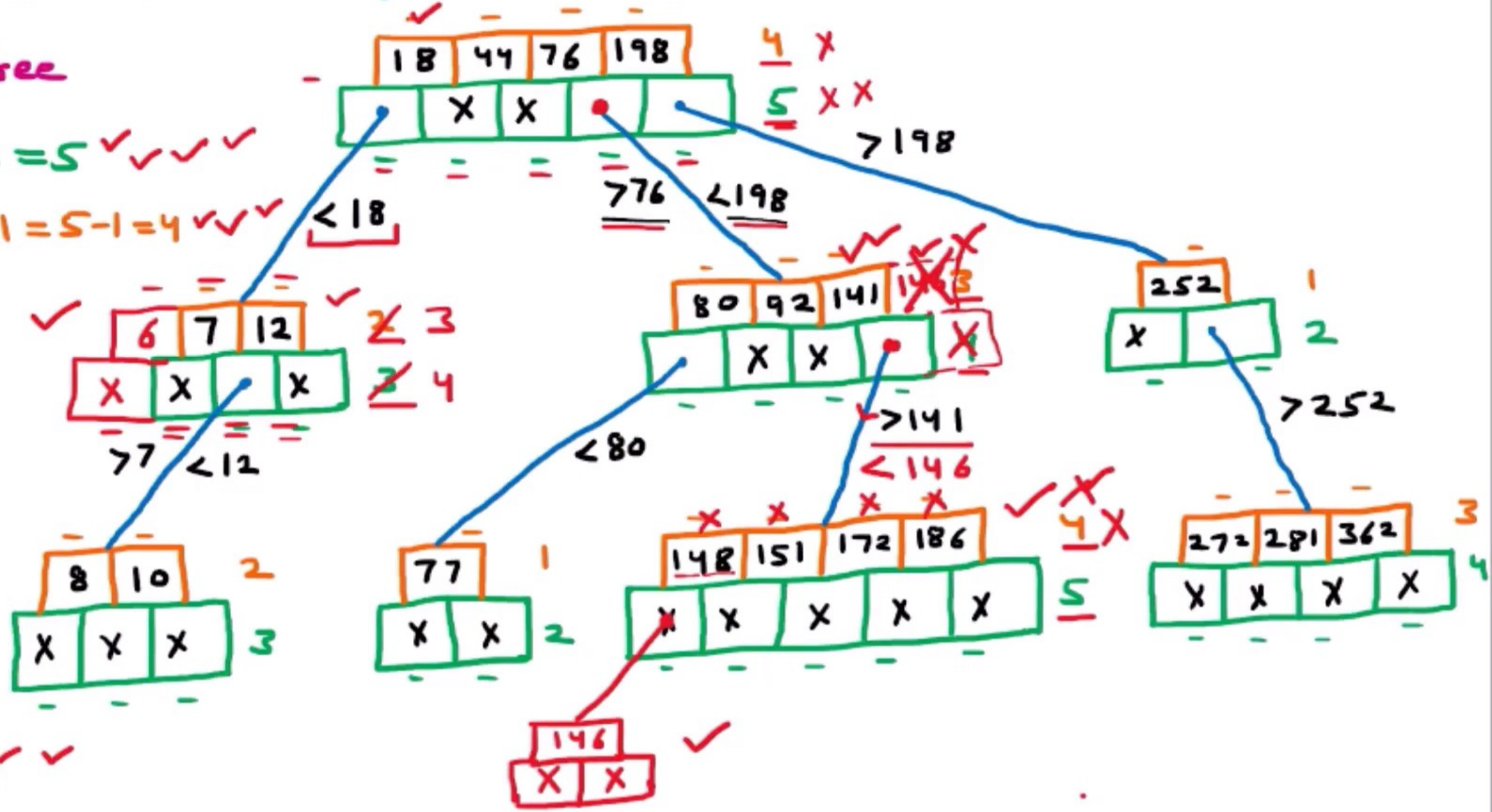
Insert 146

Insertion in m Way Tree

5 Way Tree

Child = m = 5

Keys = m - 1 = 5 - 1 = 4



Insert 6

Insert 146

m Way Tree Construction

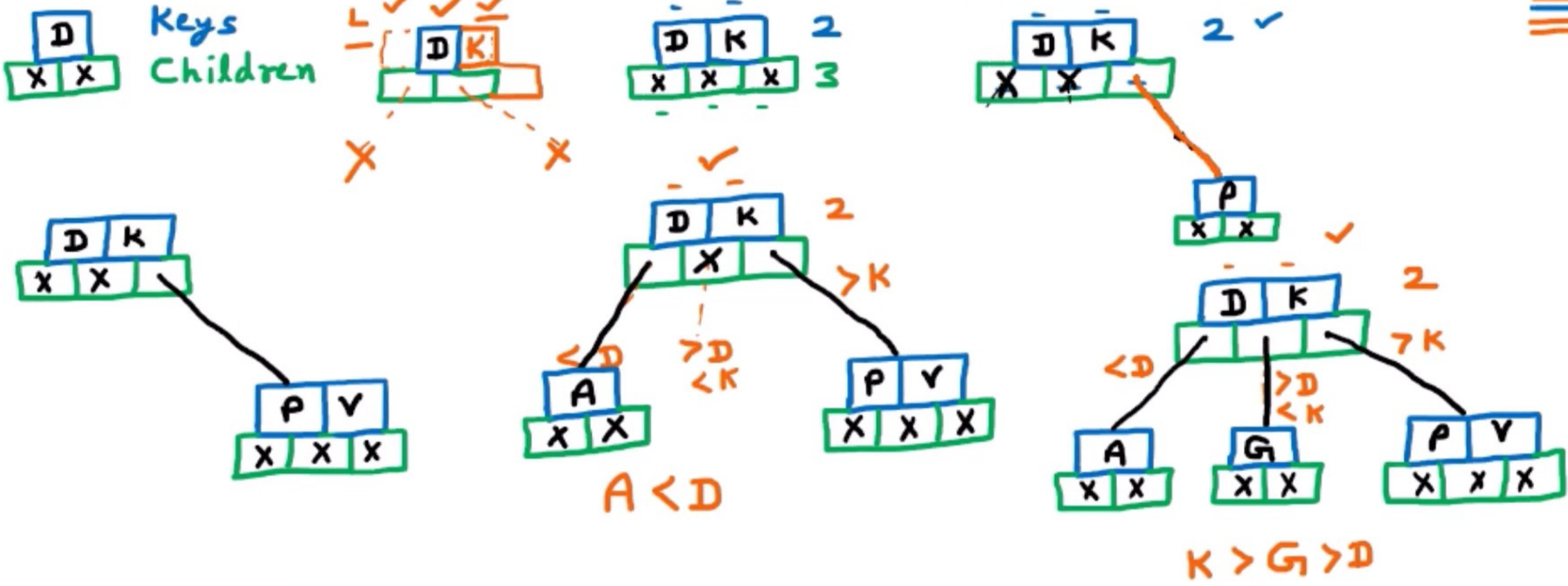
3 Way tree constructed out of empty search tree with following keys in order

D, K, P, V, A, G

m = 3

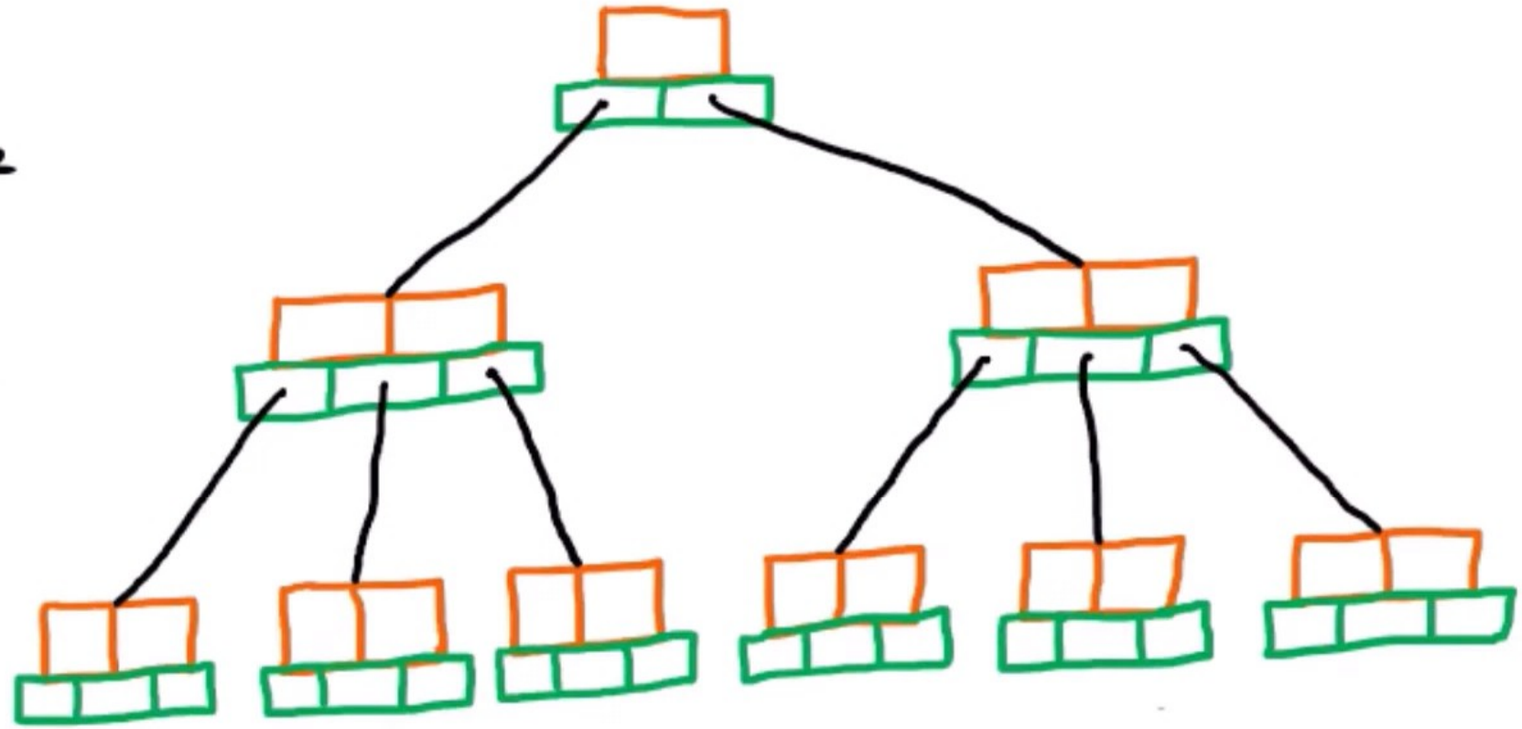
Maximum Children = m = 3

Maximum Keys = m - 1 = 3 - 1 = 2



B Tree

- Balanced m-way Tree
- Height of Tree should be as low as possible

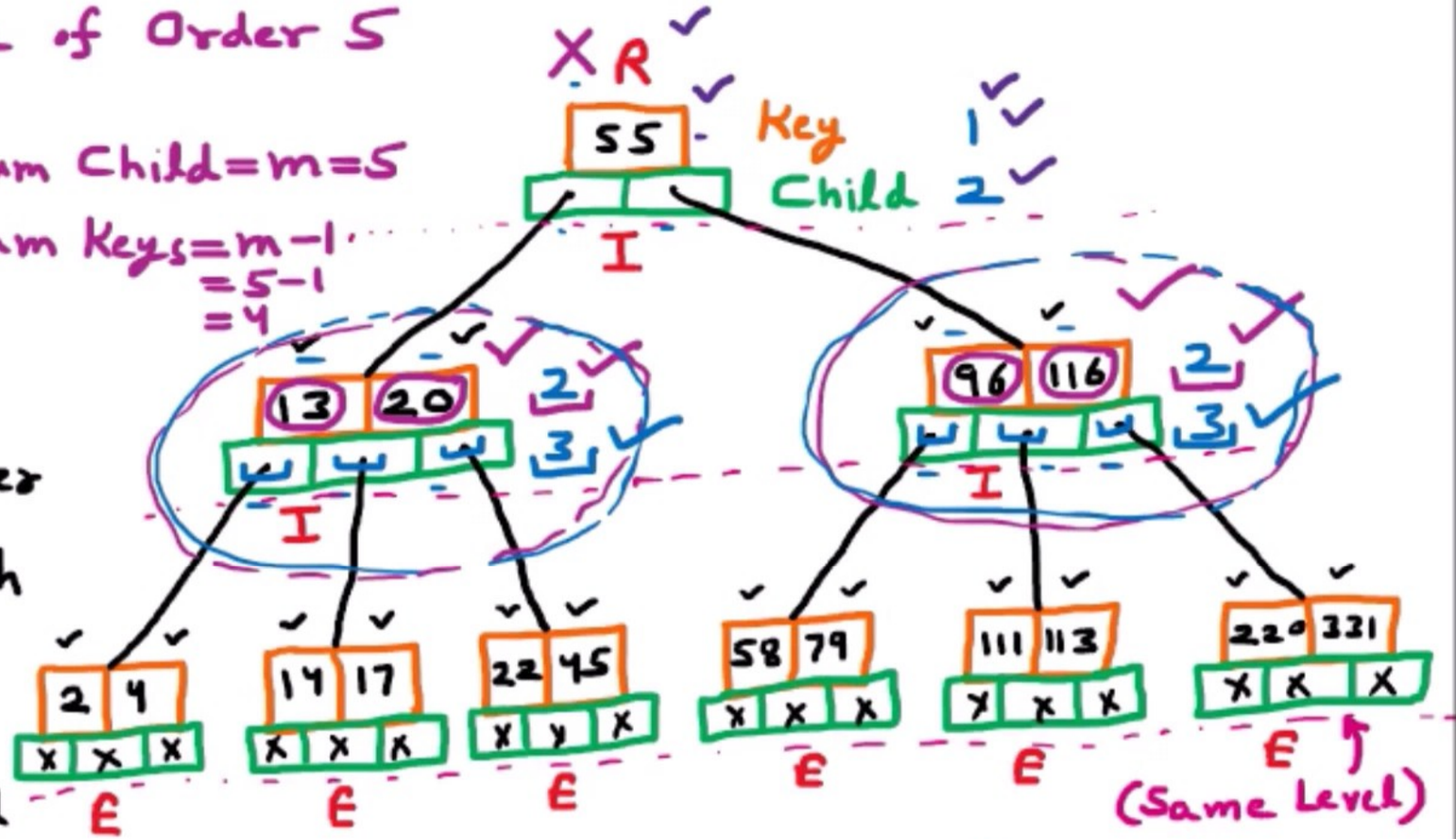


Advantage

- Minimize File Access due to their Restricted Height

B Tree

B Tree of Order 5
m=5
Maximum Child=m=5
Maximum Keys=m-1
=5-1
=4



Properties

- All keys of Node are sorted in Increasing Order
- Number of keys in each Internal Node is one less than number of Child
- All leaves at same level
- Root has at least 2 child and at most m child (Root has minimum 1 key)

$\left\lceil \frac{5}{2} \right\rceil = \lceil 2.5 \rceil = 3$

- Internal Node except Root have at least $\left\lceil \frac{m}{2} \right\rceil$ child and at most m child (Internal Node except Root have minimum $\left\lceil \frac{m-1}{2} \right\rceil$ keys)

$\left\lceil \frac{5-1}{2} \right\rceil = \left\lceil \frac{4}{2} \right\rceil = \lceil 2 \rceil = 2$

Insertion in B Tree

Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$

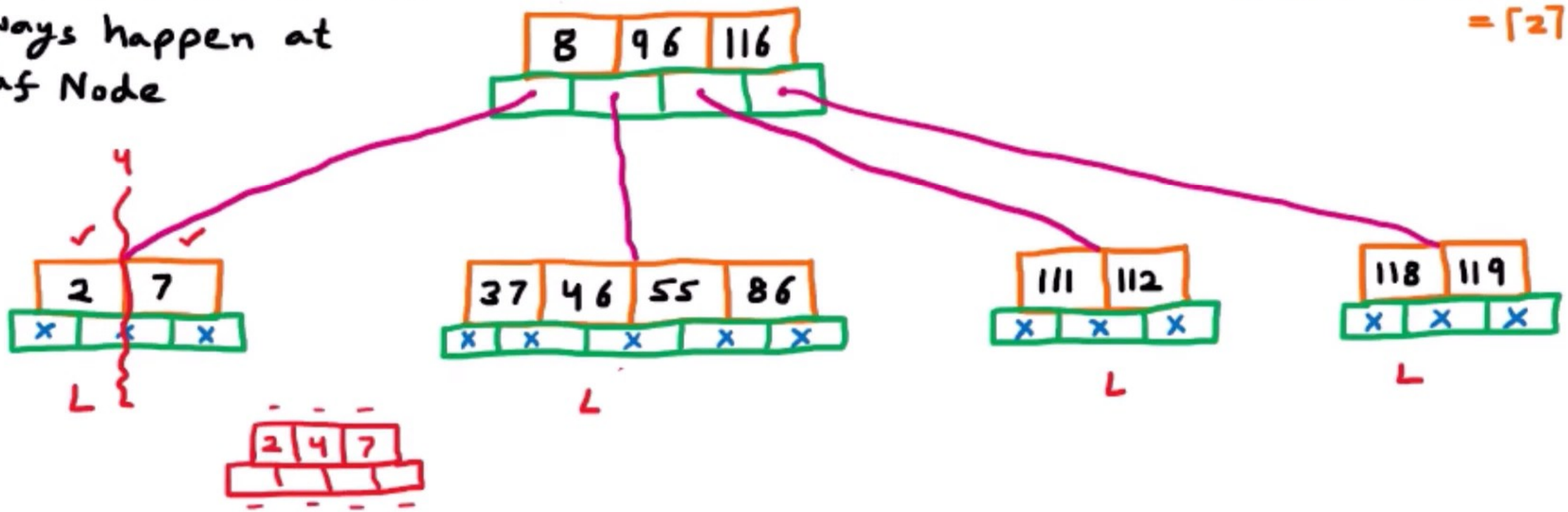
Internal Nodes except Root have

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$

BTree of Order 5

Insertion in B Tree always happen at Leaf Node



Insert 4

Insertion in B Tree

Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$ ✓

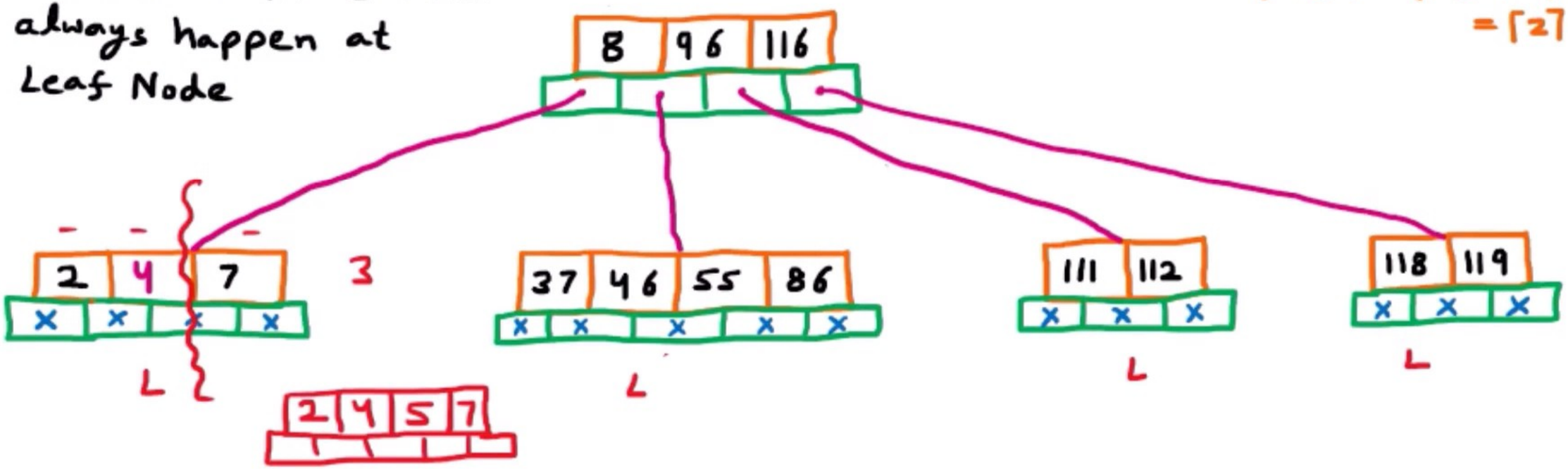
Internal Nodes except Root have

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$

BTree of Order 5

Insertion in B Tree always happen at Leaf Node



Insert 4, 5

Insertion in B Tree

Max Child = $m = 5$

Max Keys = $m-1 = 5-1 = 4$ ✓

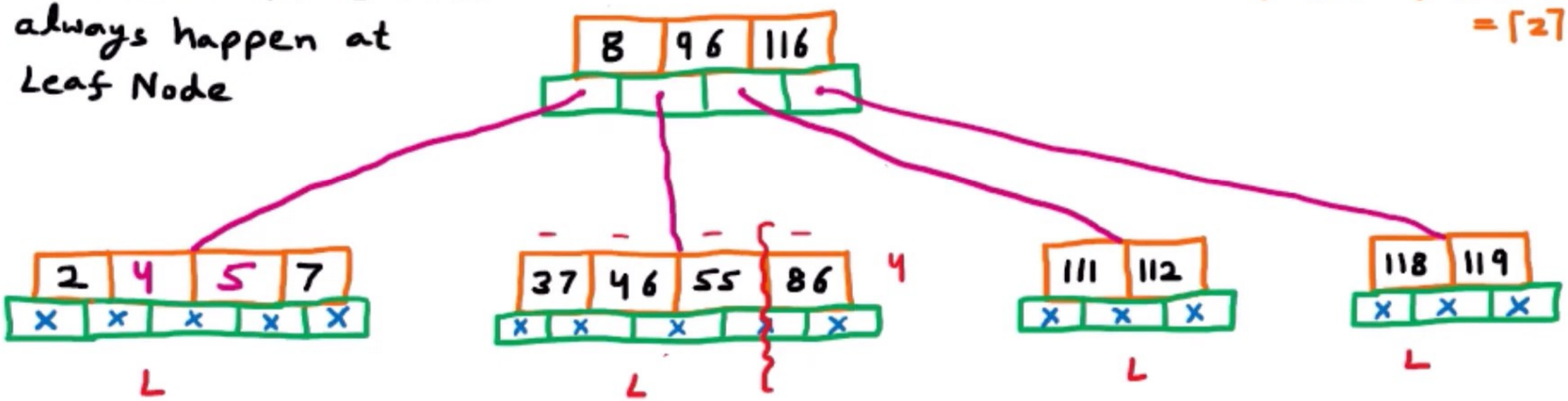
Internal Nodes except Root have

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

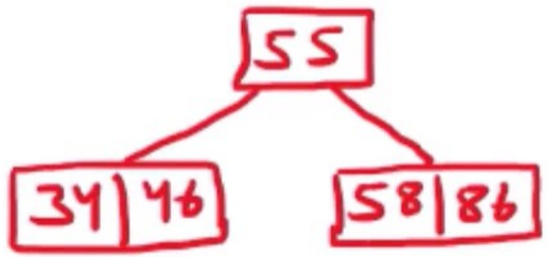
$= \lceil 2 \rceil = 2$

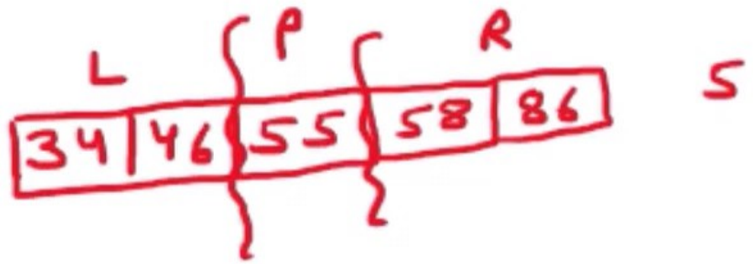
BTree of Order 5

Insertion in B Tree always happen at Leaf Node



Insert 4, 5, 58





Insertion in B Tree

Max Child = m = 5

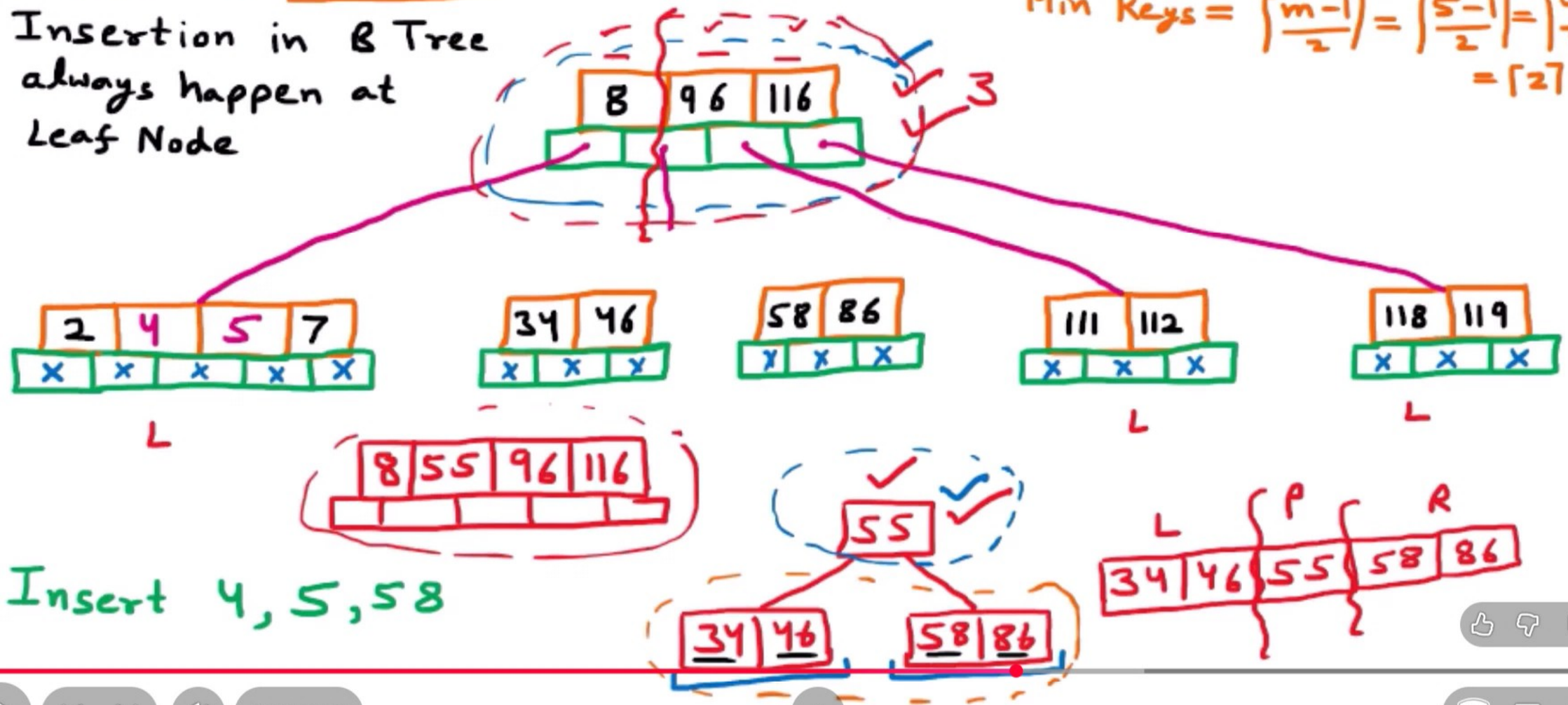
Max Keys = m-1 = 5-1 = 4

Internal Nodes except Root have

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil = \lceil 2 \rceil = 2$

B Tree of Order 5

Insertion in B Tree always happen at Leaf Node



Insert 4, 5, 58

Insertion in B Tree

Max Child = $m = 5$

Max Keys = $m-1 = 5-1 = 4$ ✓✓

Internal Nodes except Root have

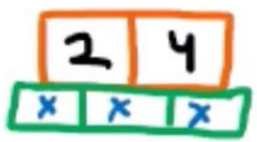
Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

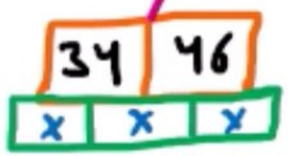
$= \lceil 2 \rceil = 2$

BTree of Order 5

Insertion in B Tree always happen at Leaf Node

4 ✓



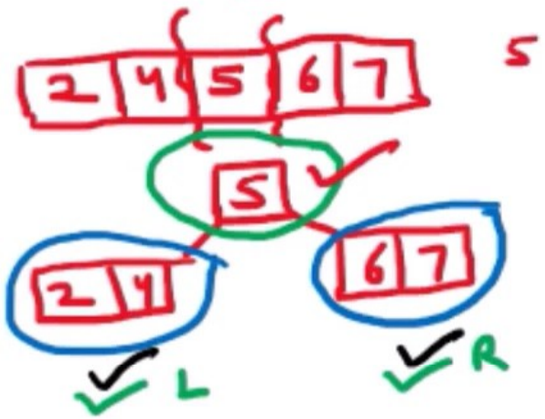


L

L

L

L



Insert 4, 5, 58, 6

Insertion in B Tree

Max Child = $m = 5$

Max Keys = $m-1 = 5-1 = 4$ ✓✓

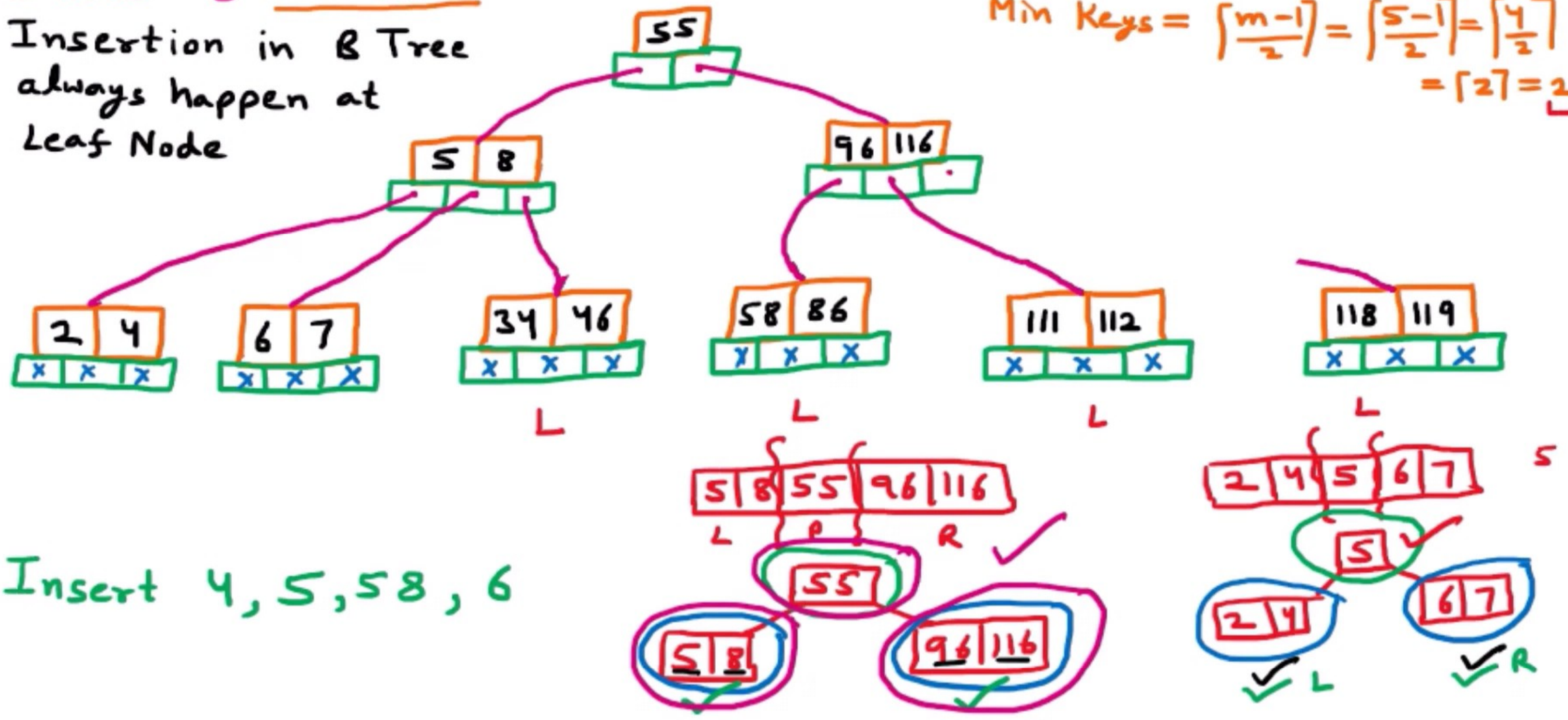
Internal Nodes except Root have

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$

BTree of Order 5

Insertion in B Tree always happen at Leaf Node



Insert 4, 5, 58, 6

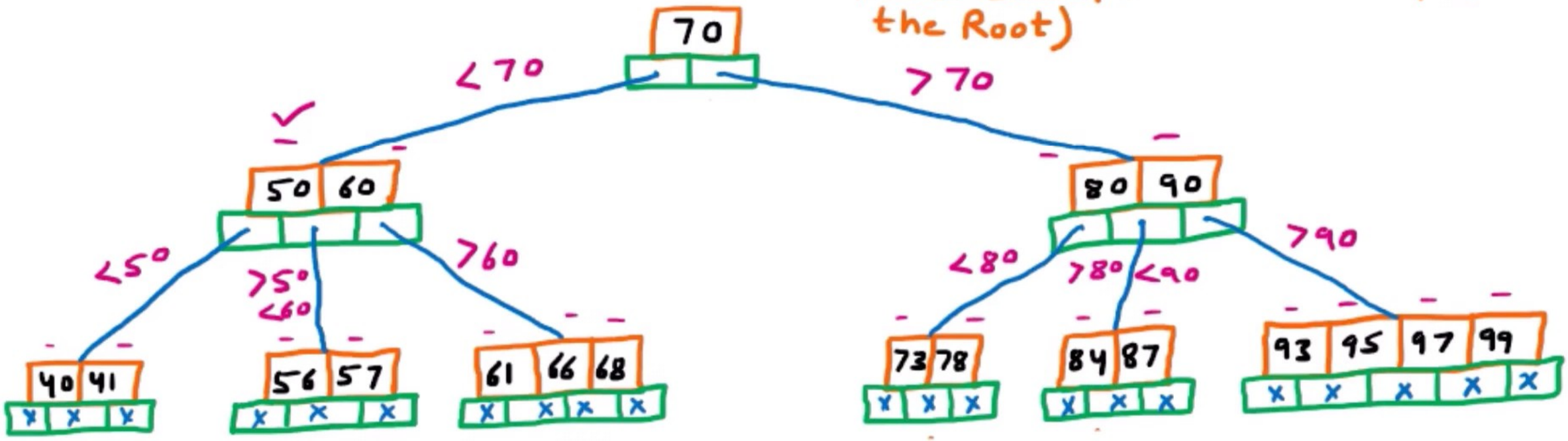
Deletion in B Tree

B Tree of Order 5

Max Child = $m = 5$

Max Keys $= m - 1 = 5 - 1 = 4$

Min Keys (at Internal Node except the Root) $= \lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil = \lceil 2 \rceil = 2$



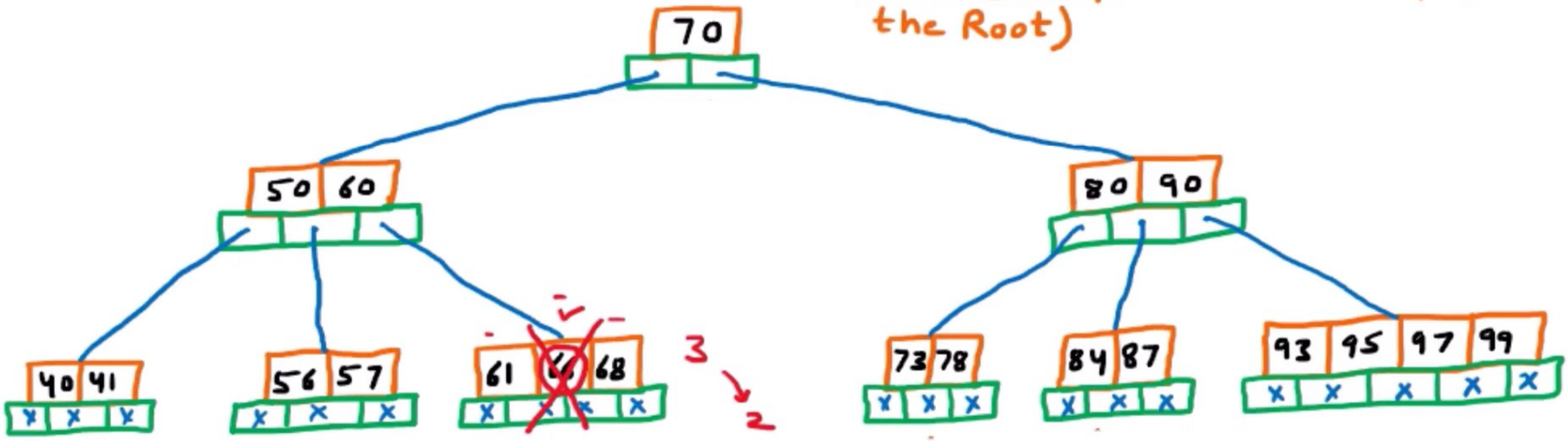
Deletion in B Tree

B Tree of Order 5

Max Child = $m = 5$

Max Keys $= m - 1 = 5 - 1 = 4$

Min Keys (at Internal Node except the Root) $= \lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil = \lceil 2 \rceil = 2$



Delete 66

Deletion in B Tree

B Tree of Order 5

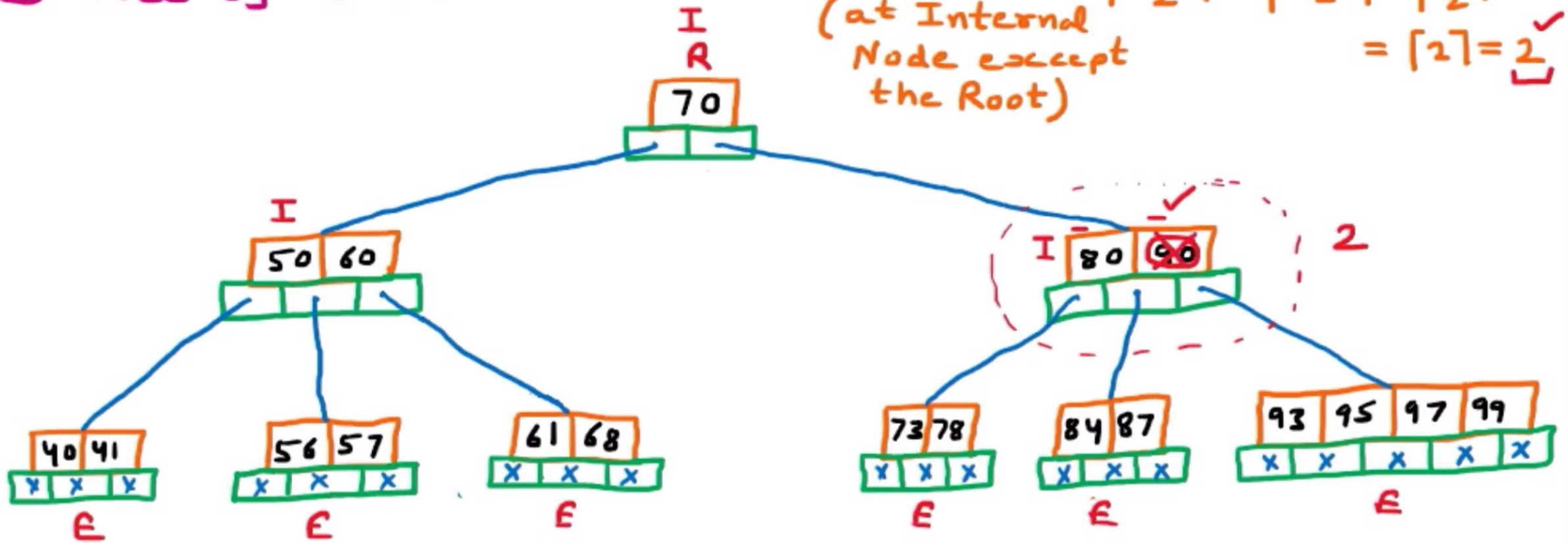
Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$ ✓

(at Internal Node except the Root)



Delete 66, 90

Deletion in B Tree

B Tree of Order 5

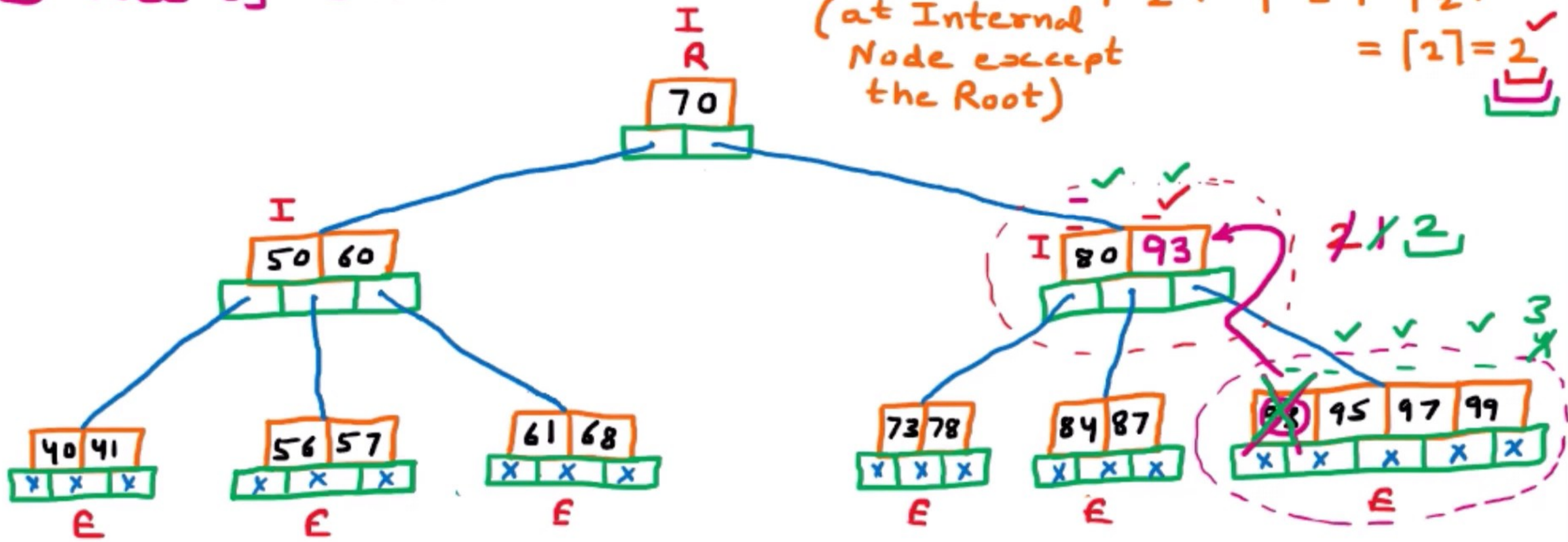
Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$

(at Internal Node except the Root)



Delete 66, 90

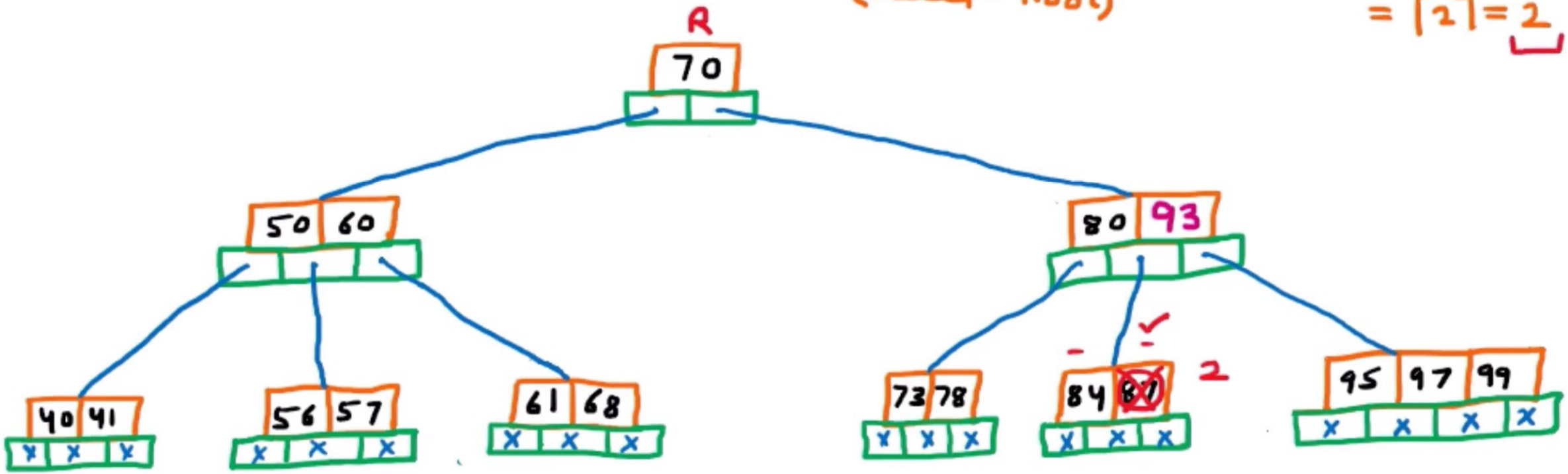
Deletion in B Tree

B Tree of Order 5

Max Child = $m = 5$

Max Keys $= m-1 = 5-1 = 4$

Min Keys (Except Root) $= \lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil = \lceil 2 \rceil = 2$



Delete 66, 90, 87

Deletion in B Tree

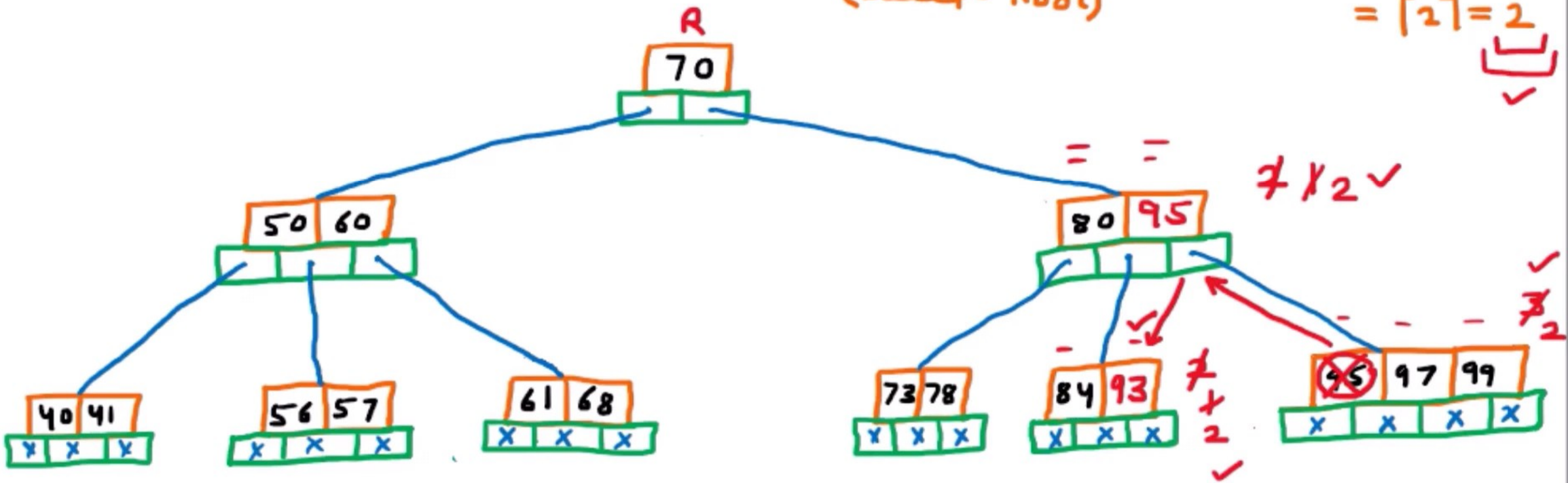
B Tree of Order 5

Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$

Min Keys (Except Root) = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$



Delete 66, 90, 87

Deletion in B Tree

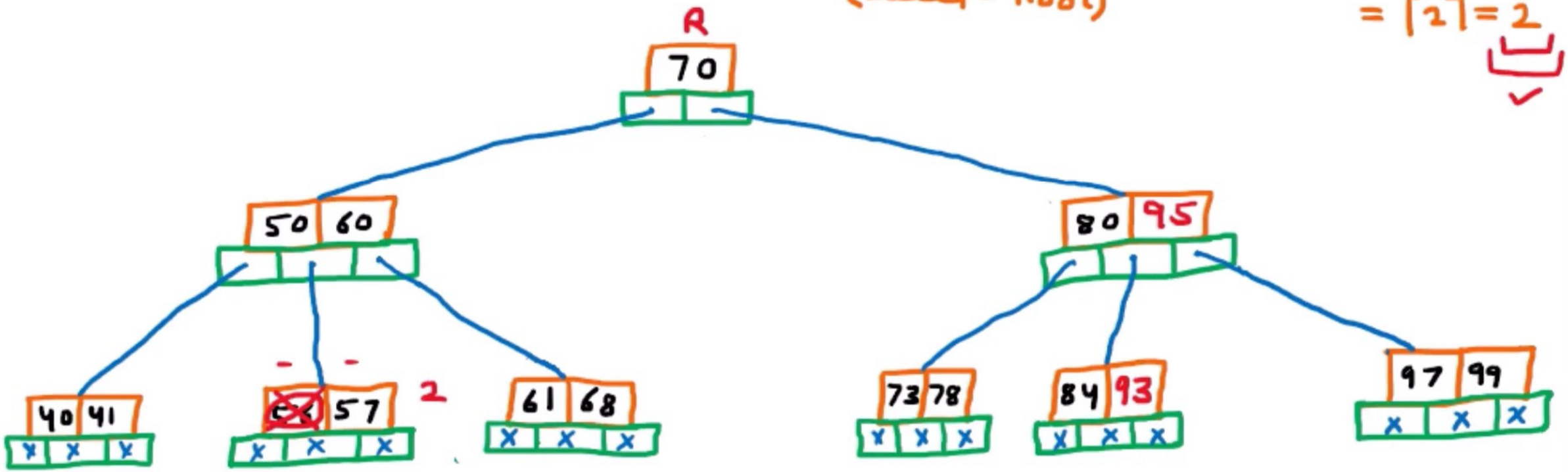
B Tree of Order 5

Max Child = $m = 5$

Max Keys $= m - 1 = 5 - 1 = 4$

Min Keys (Except Root) $= \lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

$= \lceil 2 \rceil = 2$



Delete 66, 90, 87, 56

Deletion in B Tree

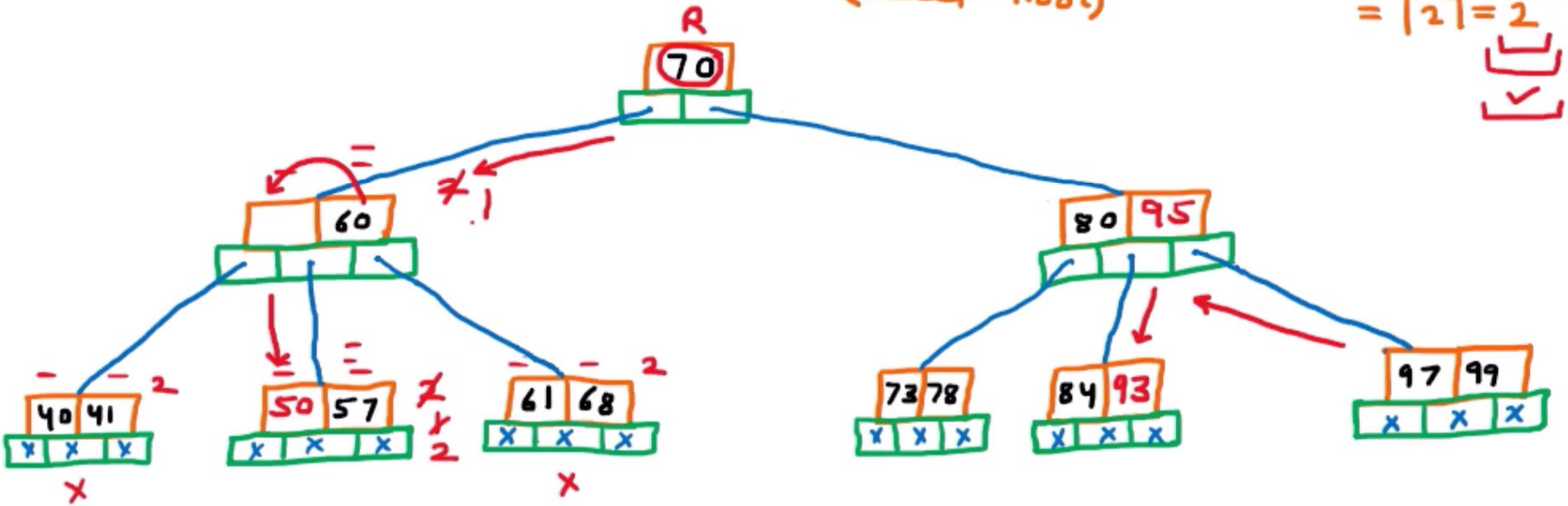
B Tree of Order 5

Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

(Except Root) $= \lceil 2 \rceil = 2$ ✓



Delete 66, 90, 87, 56

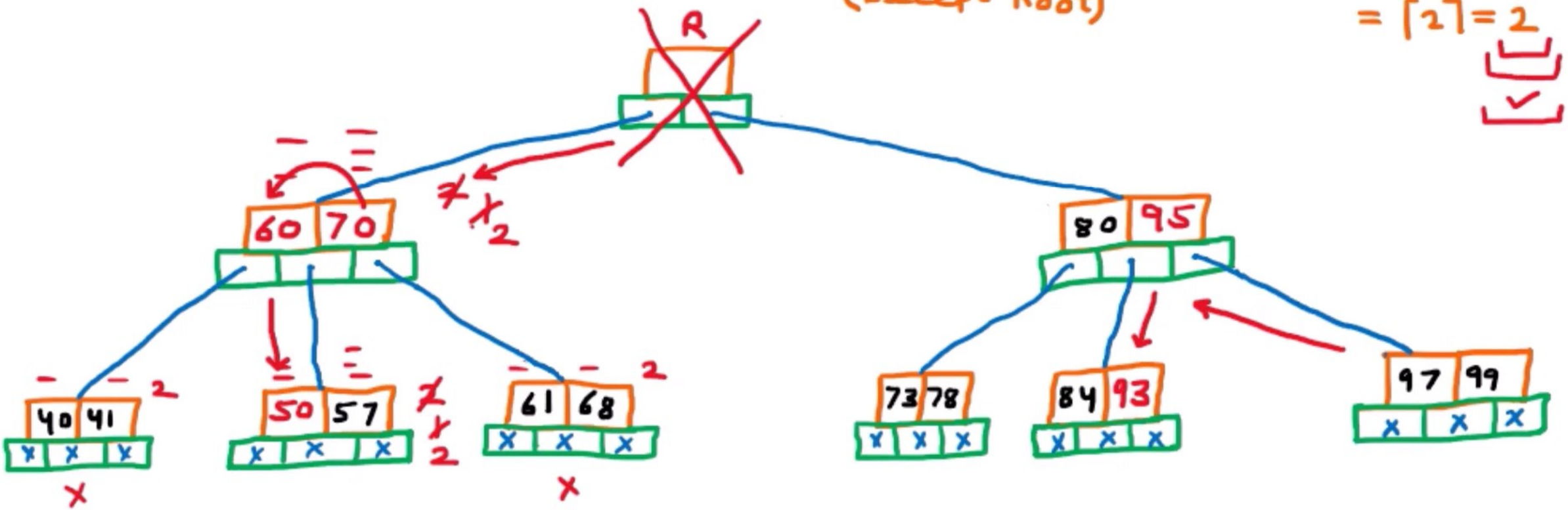
Deletion in B Tree

B Tree of Order 5

Max Child = $m = 5$

Max Keys $= m-1 = 5-1 = 4$

Min Keys (Except Root) $= \lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil = \lceil 2 \rceil = 2$ ✓



Delete 66, 90, 87, 56

Deletion in B Tree

B Tree of Order 5

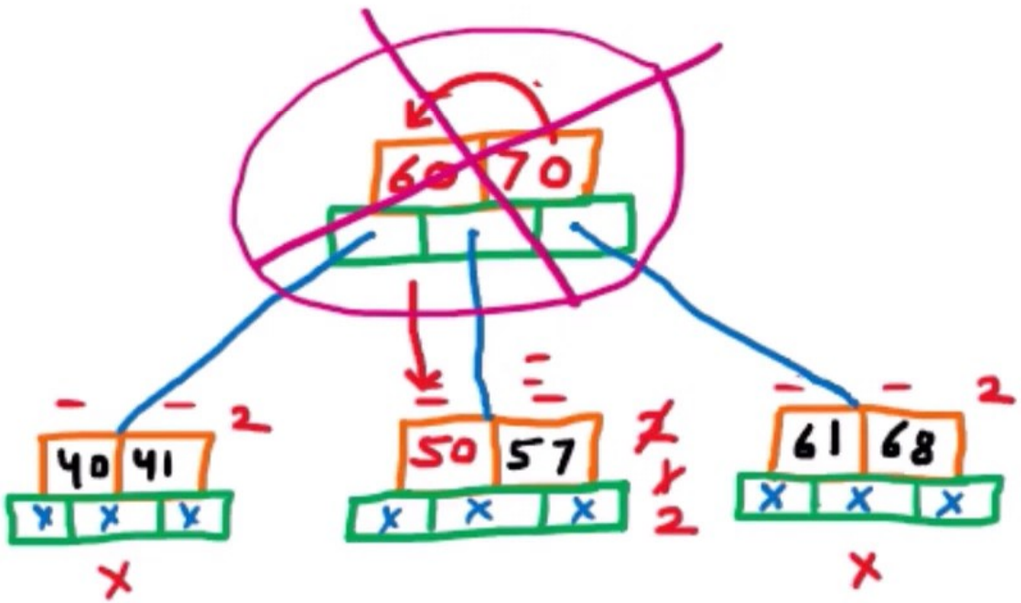
Max Child = $m = 5$ ✓

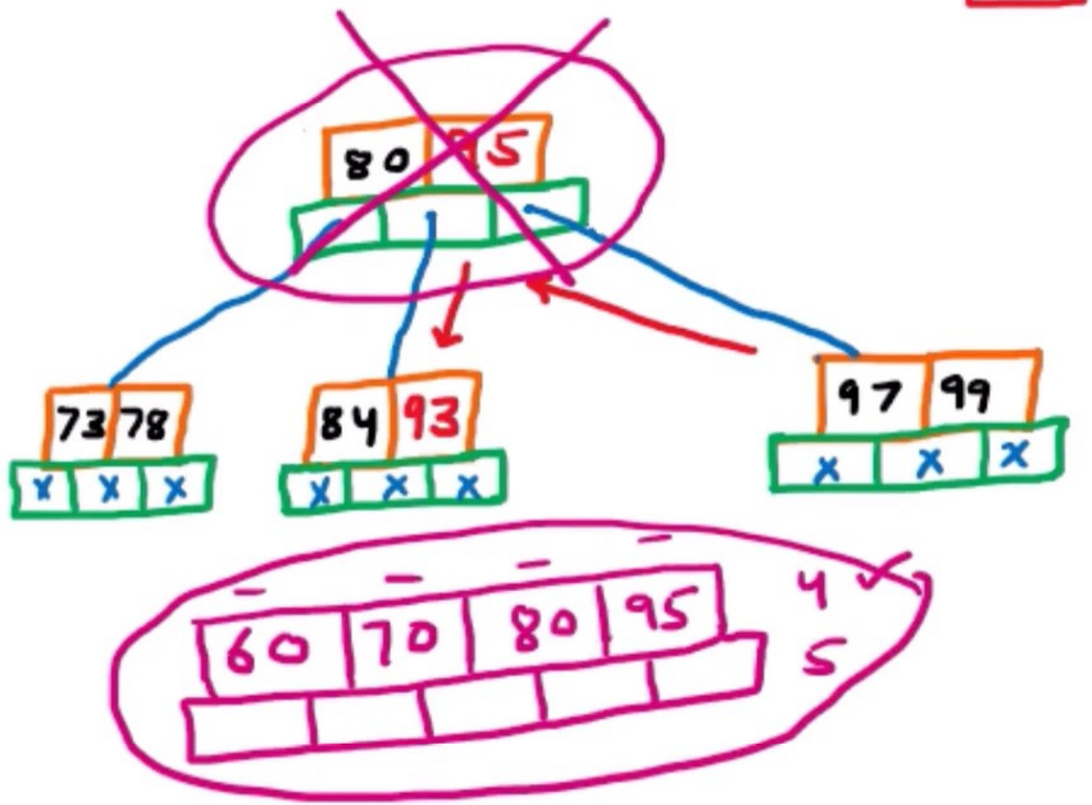
Max Keys = $m - 1 = 5 - 1 = 4$ ✓

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

(Except Root)

$= \lceil 2 \rceil = 2$ ✓





Delete 66, 90, 87, 56

Deletion in B Tree

B Tree of Order 5

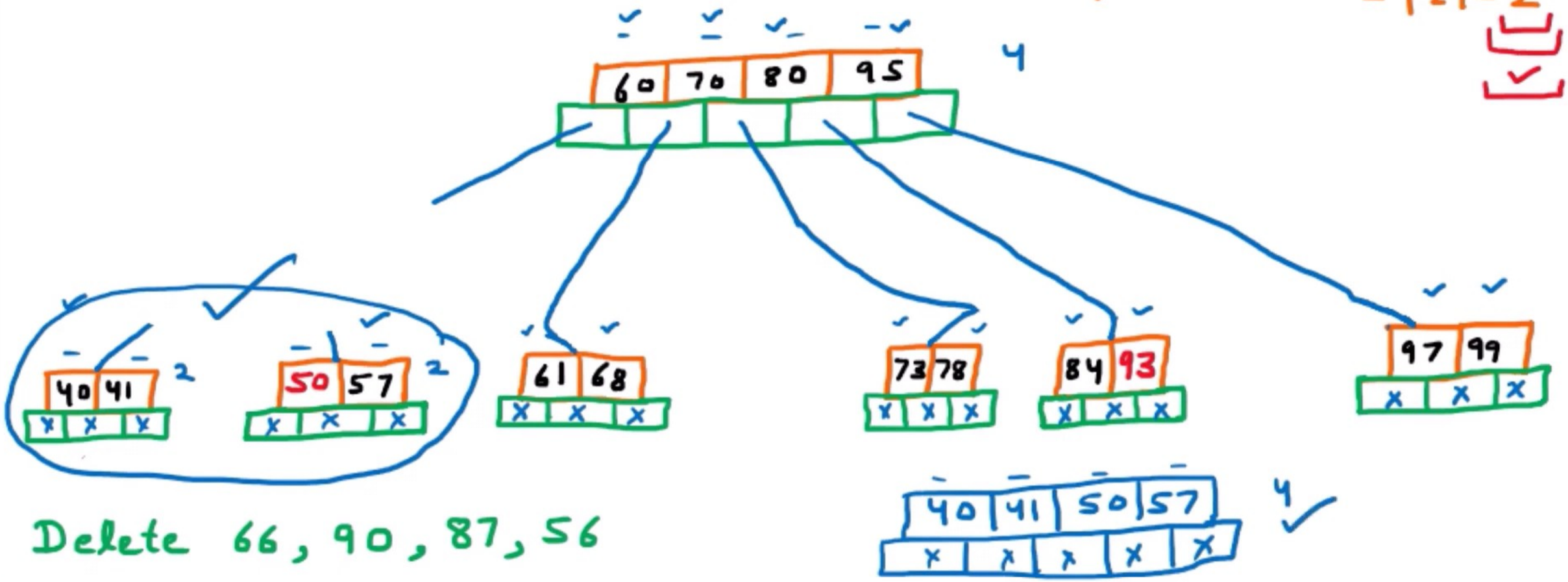
Max Child = $m = 5$

Max Keys = $m - 1 = 5 - 1 = 4$

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

(Except Root)

$= \lceil 2 \rceil = 2$



Delete 66, 90, 87, 56

Deletion in B Tree

B Tree of Order 5

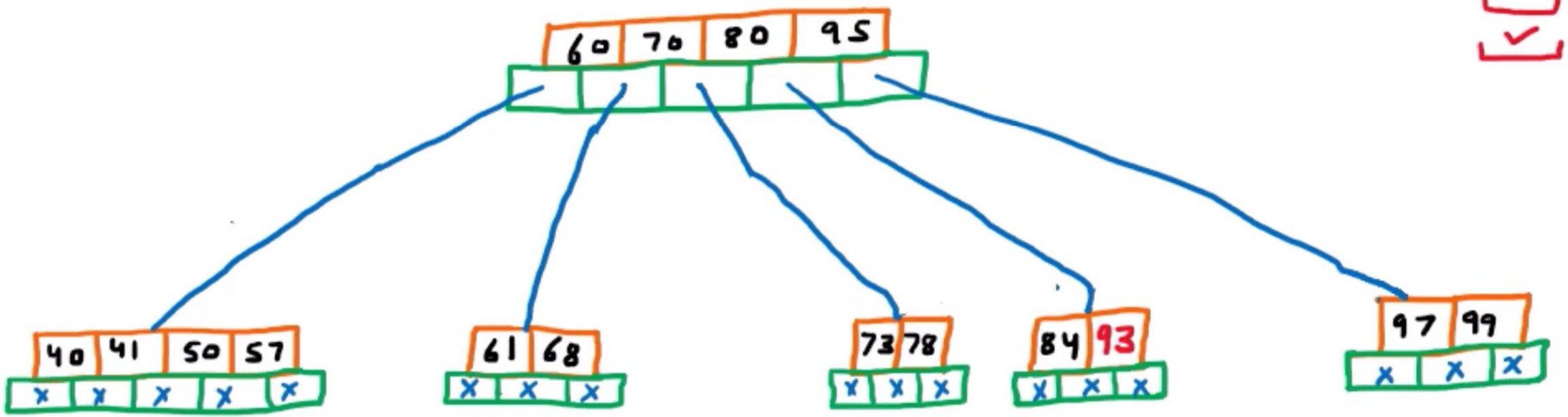
Max Child = $m = 5$ ✓

Max Keys = $m - 1 = 5 - 1 = 4$ ✓✓

Min Keys = $\lceil \frac{m-1}{2} \rceil = \lceil \frac{5-1}{2} \rceil = \lceil \frac{4}{2} \rceil$

(Except Root)

$= \lceil 2 \rceil = 2$ ✓



Delete 66, 90, 87, 56