AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-1

Today's Target

- Introduction to Red-Black Tree
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Computer Science & Engineering , Information Technology & Computer Science Allied

AKTU : Syllabus

Unit-I : Introduction

Introduction: Algorithms, Analyzing Algorithms, Complexity of Algorithms, Growth of Functions, Performance Measurements, Sorting and Order Statistics - Shell Sort, Quick Sort, Merge Sort, Heap Sort, Comparison of Sorting Algorithms, Sorting in Linear Time.

UNIT-II : Advanced Data Structures

Advanced Data Structures: Red-Black Trees, B – Trees, Binomial Heaps, Fibonacci Heaps, Tries, Skip List .

Red-Black Tree

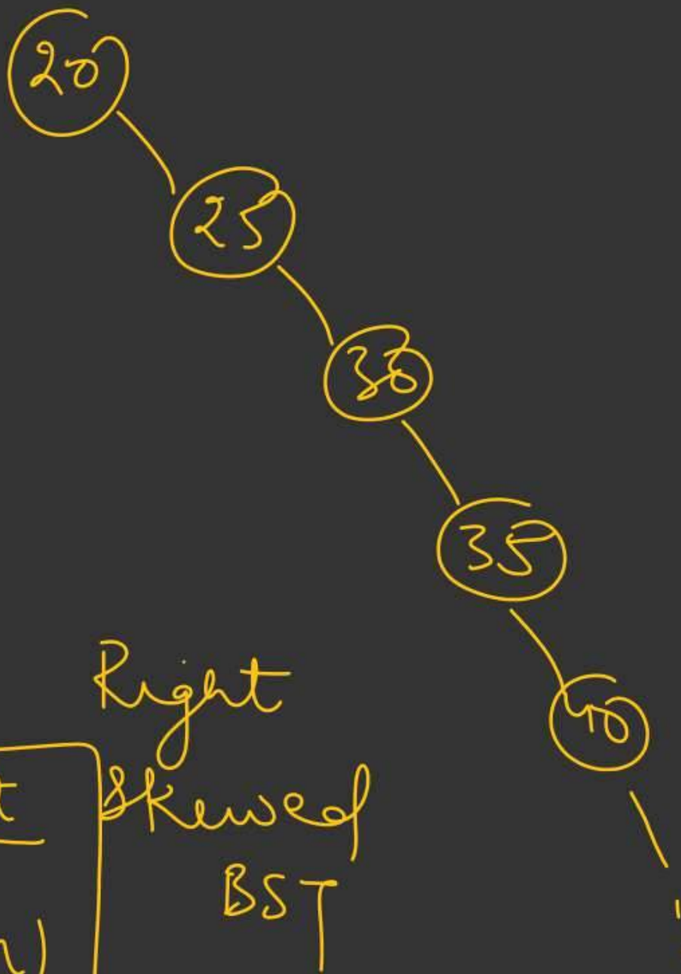
Binary Search Tree

(1) AVL-Tree

(2) R-B Tree

(1) Recolor

(2) Rotation



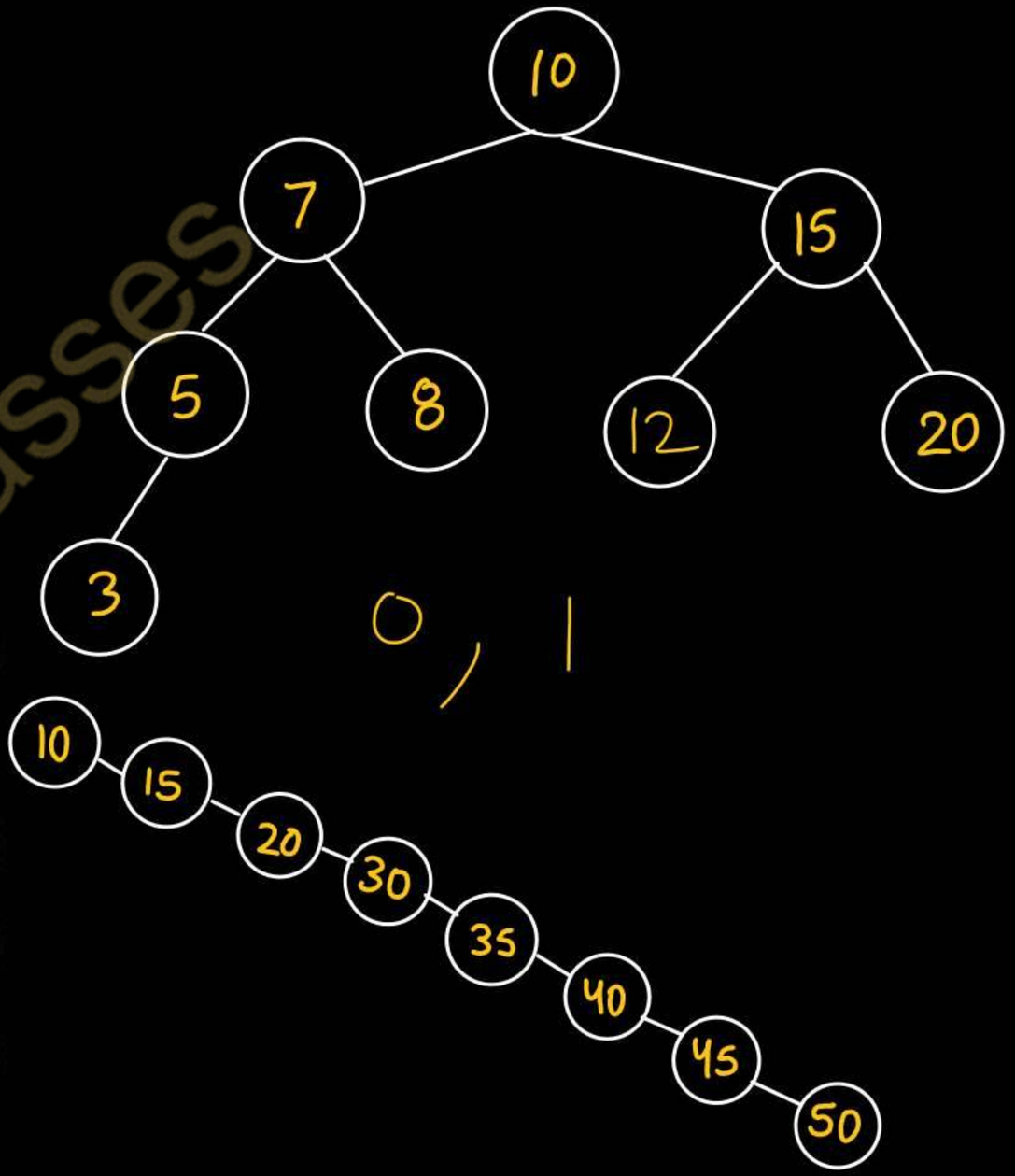
Right Skewed BST

n-node BST

	avg.	Best case	worst
Search Insert Delete	$O(\log n)$	$O(1)$	$O(n)$

Why Red Black Tree?

- RB tree is a self balanced binary search tree where each node contains an extra bit that represents a color to ensure that the tree is balanced during any operations performed on the tree like insertion, deletion, etc.
- Unlike AVL tree, RB tree requires a maximum of two rotations to balance the large tree.
- AVL tree is strictly balanced, while the RB tree is not completely height-balanced but RB tree guarantees $O(\log_2 n)$ time for all operations like insertion, deletion, and searching.

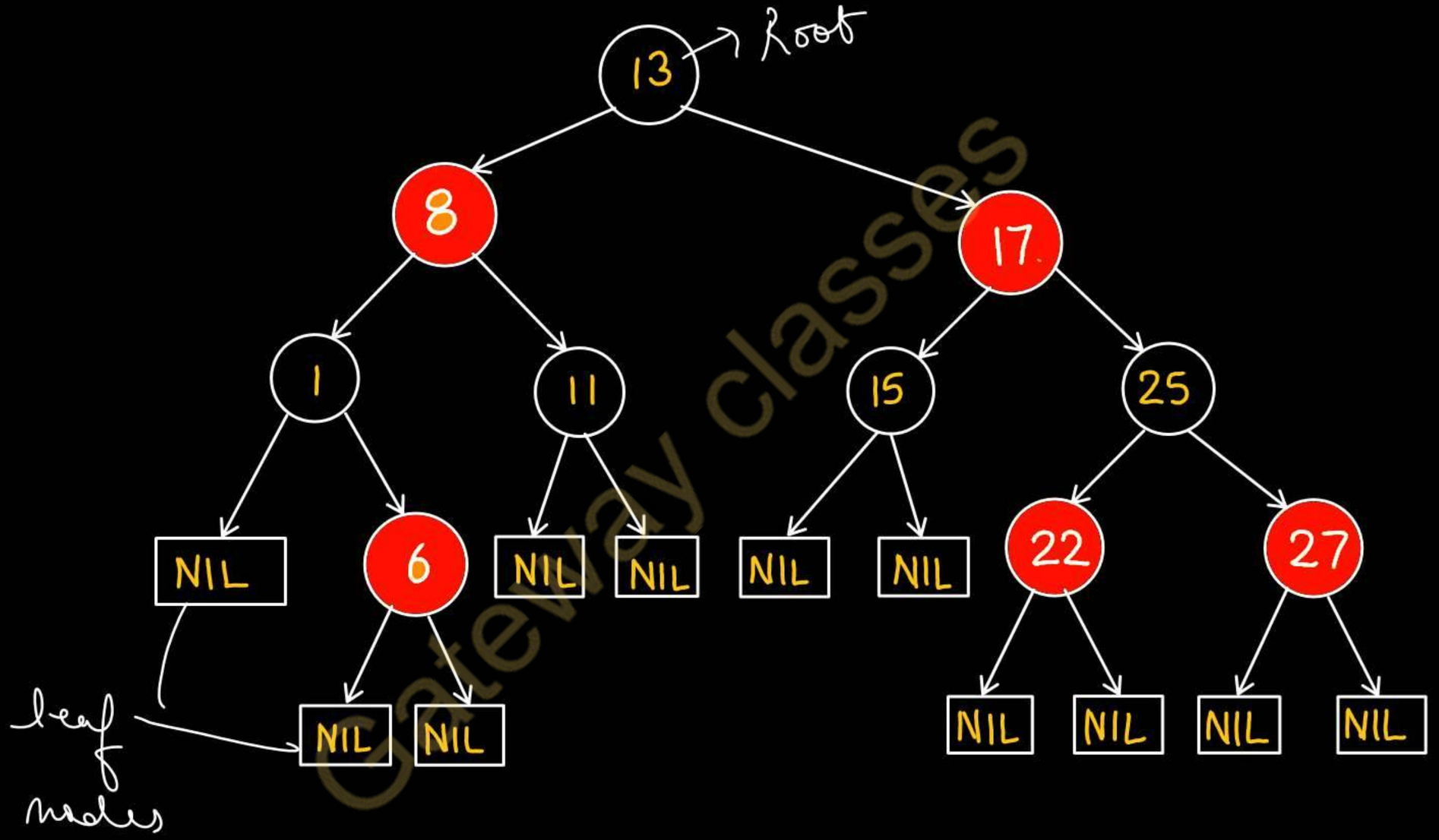


Properties of Red Black Tree

It is a self-balancing Binary Search tree. Each node is either Red or Black. Every node stores one extra bit that represents the color of the node.

- Root node is always black in color.
- Nodes having no child are considered internal nodes and these are connected to the NIL nodes (always black in color). The NIL nodes are the leaf nodes in RB tree.
- If the node is Red, then its children should be in Black color. Or we can say that there should be no red-red parent-child relationship.
- Every path from a node to any of its descendant's NIL node should have same number of black nodes (Same black height).

Red Black Tree

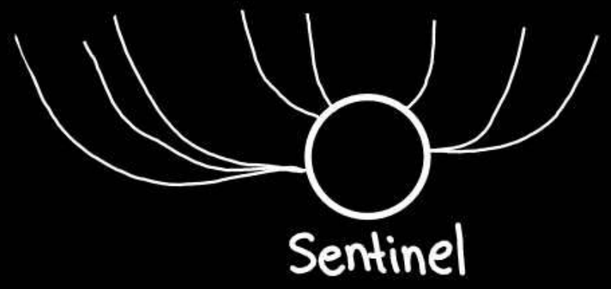


Theorem: A red-black tree with n internal nodes has height(h) at most 2log(n+1)

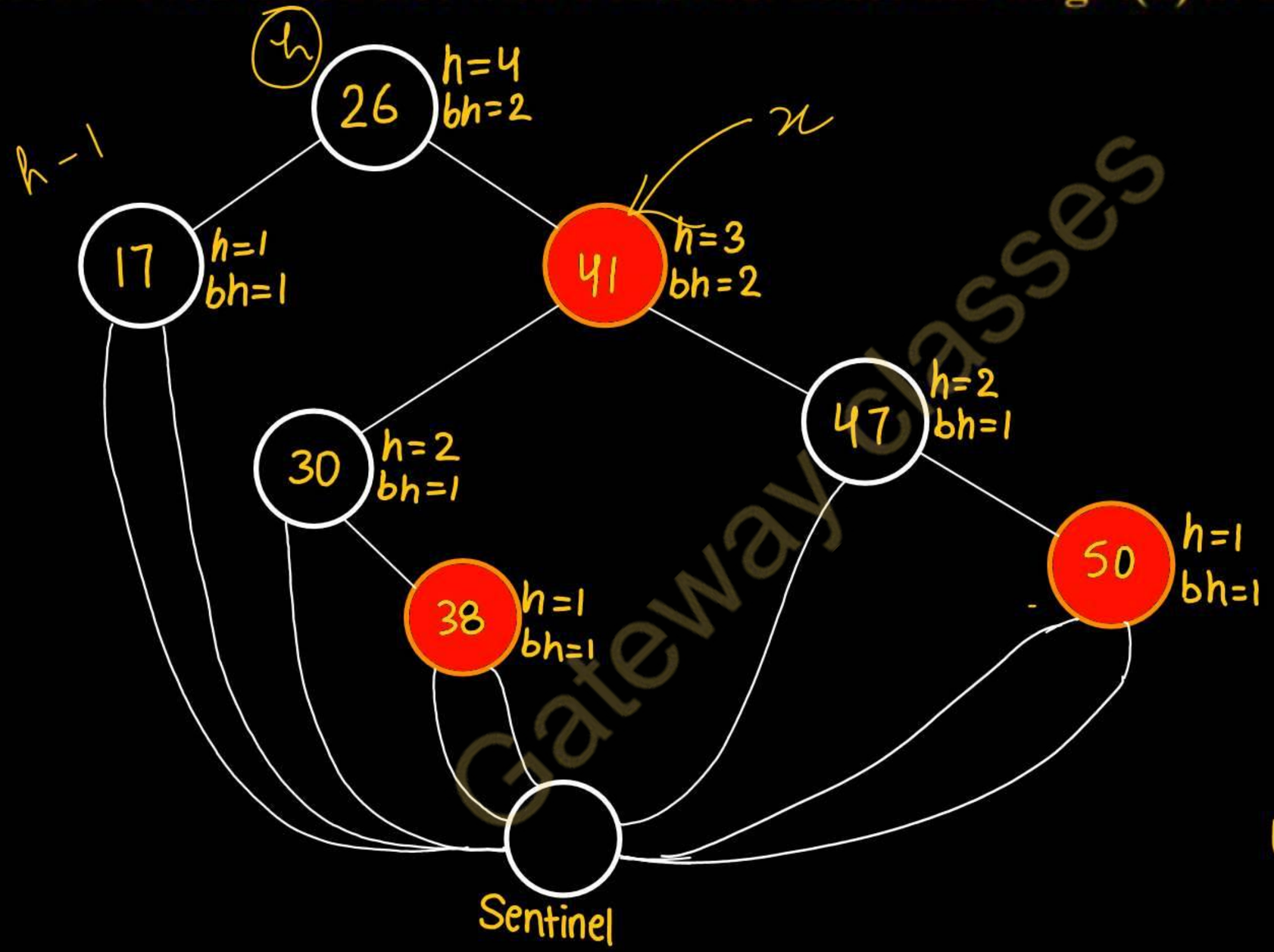
h=4

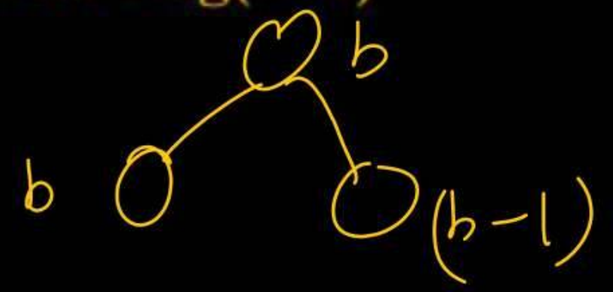
- Any node with height h has black-height at least $h/2$ and subtree rooted at any node x contains at least $2^{bh(x)} - 1$ internal nodes
- Any child of x has height $h - 1$ and black-height either b (if the child is red) or $b - 1$ (if the child is black).
- By the inductive hypothesis, each child has at least $2^{bh(x)-1} - 1$ internal nodes. Thus, the subtree rooted at x contains at least $2 \cdot (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$ internal nodes. (+ 1 is for x itself.)
- Let h and b be the height and black-height of the root.
- By the above two facts, $n \geq 2^b - 1 \geq 2^{h/2} - 1$. Adding 1 to both sides and then taking logs gives $\lg(n + 1) \geq h/2$,

$$h \leq 2 \lg(n + 1).$$



Theorem: A red-black tree with n internal nodes has height(h) at most 2log(n+1)





$$2\left(2^{b-1}-1\right)+1$$

$$= 2^b - 1$$

$$n \geq 2^b - 1$$

$$n \geq 2^{h/2} - 1$$

$$n+1 \geq 2^{h/2}$$

$$\log(n+1) \geq 2^{h/2}$$

$$h \leq 2\log(n+1)$$

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-2

Today's Target

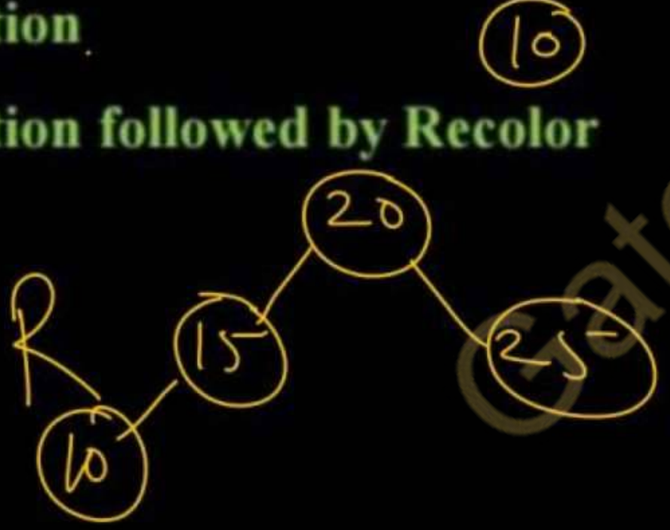
- Insertion operation in Red-Black Tree ✓
- AKTY PYQs

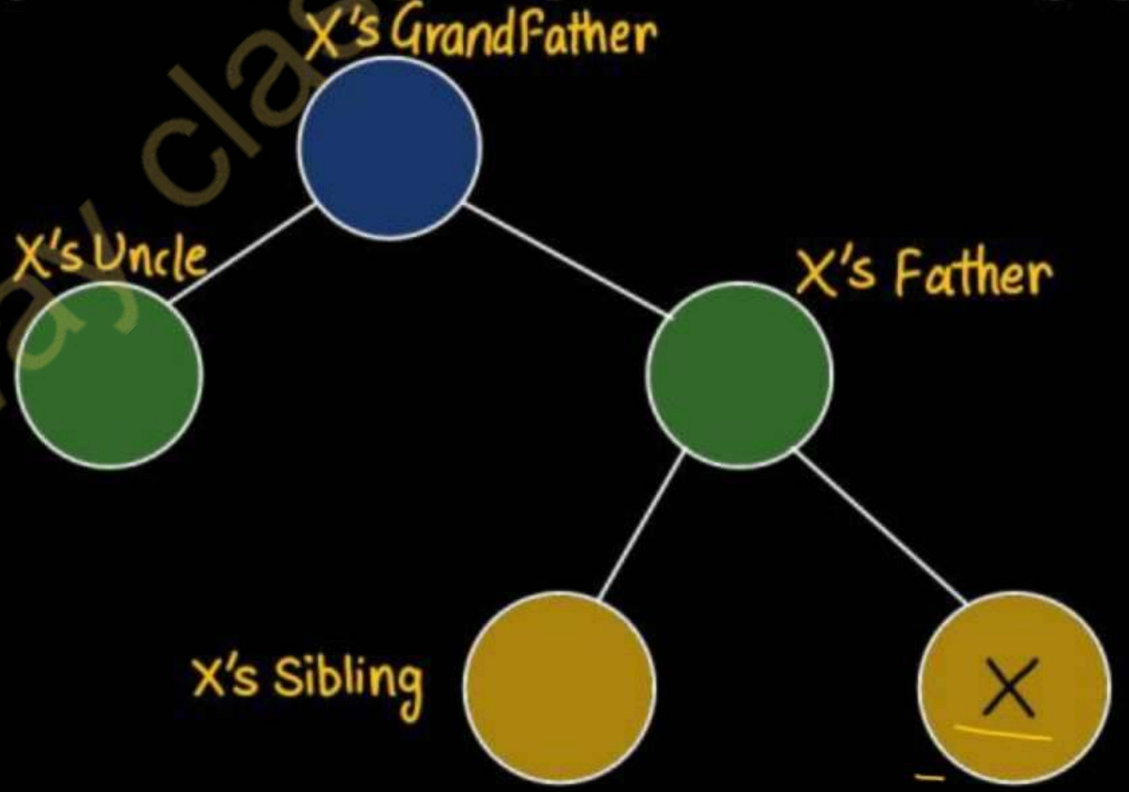
By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Insertion in Red Black Tree

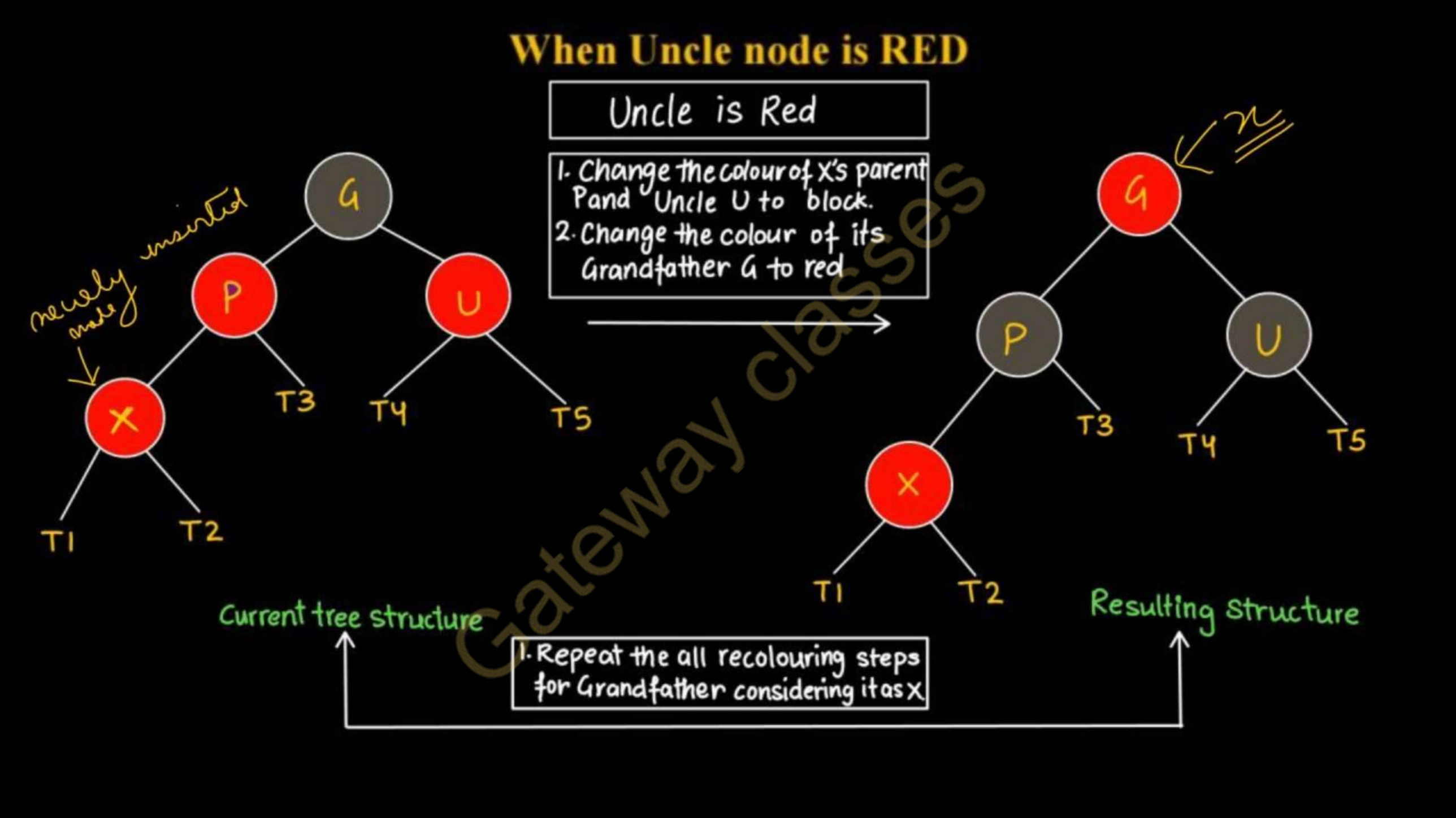
Similar to insertion operation in Binary Search Tree. But it is inserted with a color property. After every insertion operation, we need to check all the properties of Red-Black Tree. If all the properties are satisfied then we go to next operation otherwise we perform the following operation to make it Red Black Tree.

1. Recolor
2. Rotation
3. Rotation followed by Recolor





When Uncle node is RED

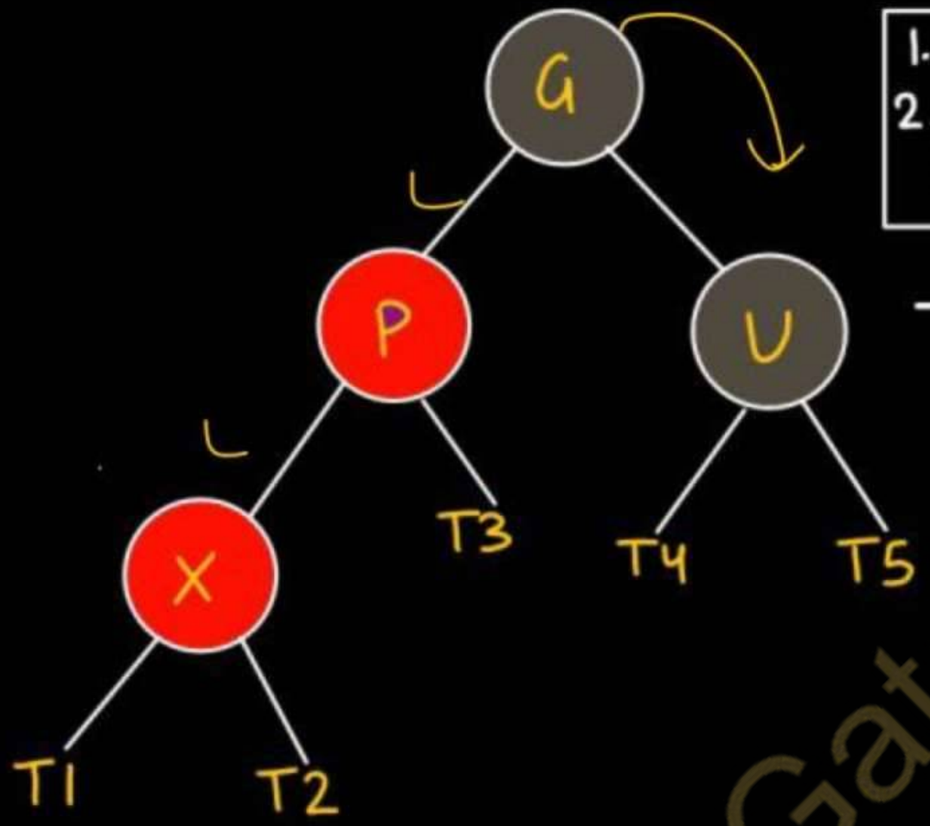


When Uncle node is BLACK

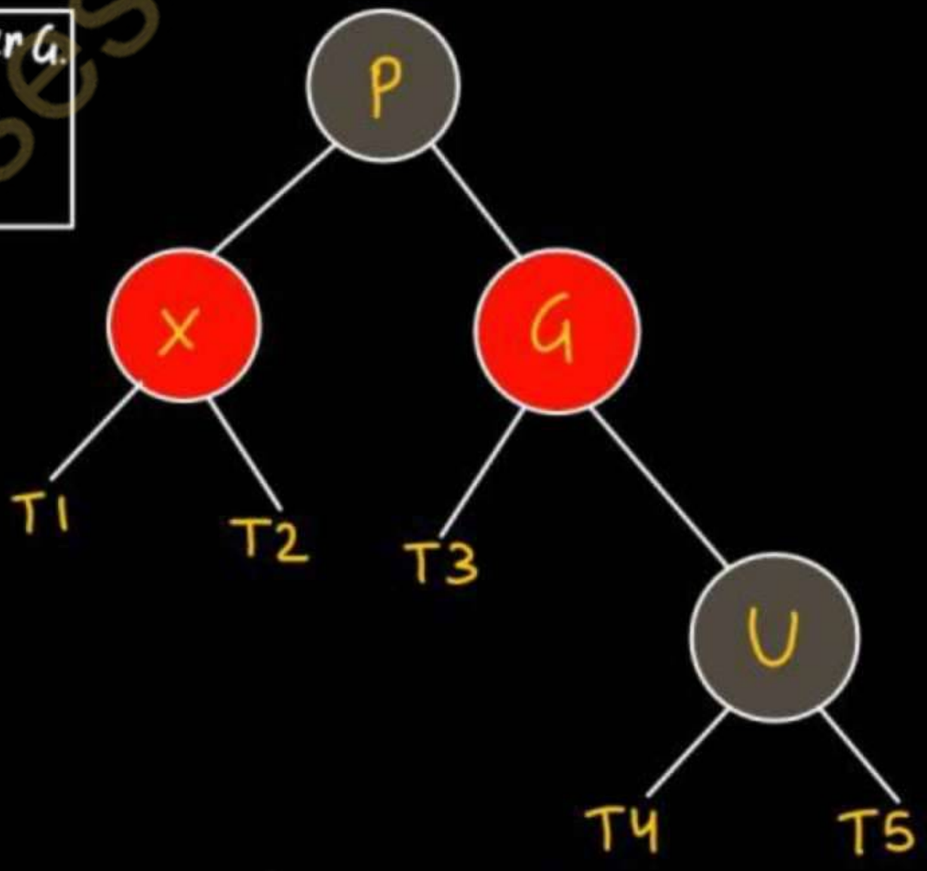
Case 1: LL rotation

Uncle is Black

1. Right rotation of grandfather G.
2. Then ~~swap~~ Recolour the colours of Grandfather G and Parent P.



Current Tree Structure



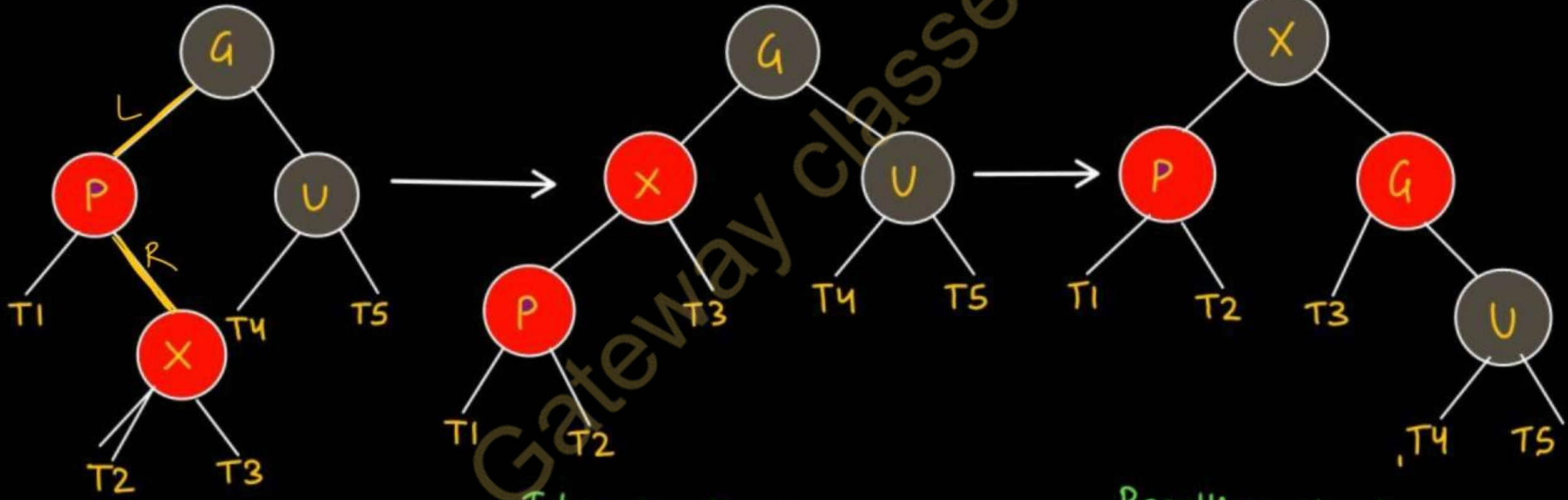
Resulting Structure

...When Uncle node is BLACK

Case 2: LR rotation

1. Left Rotation of Parent P.
2. Then apply LL rotation case.

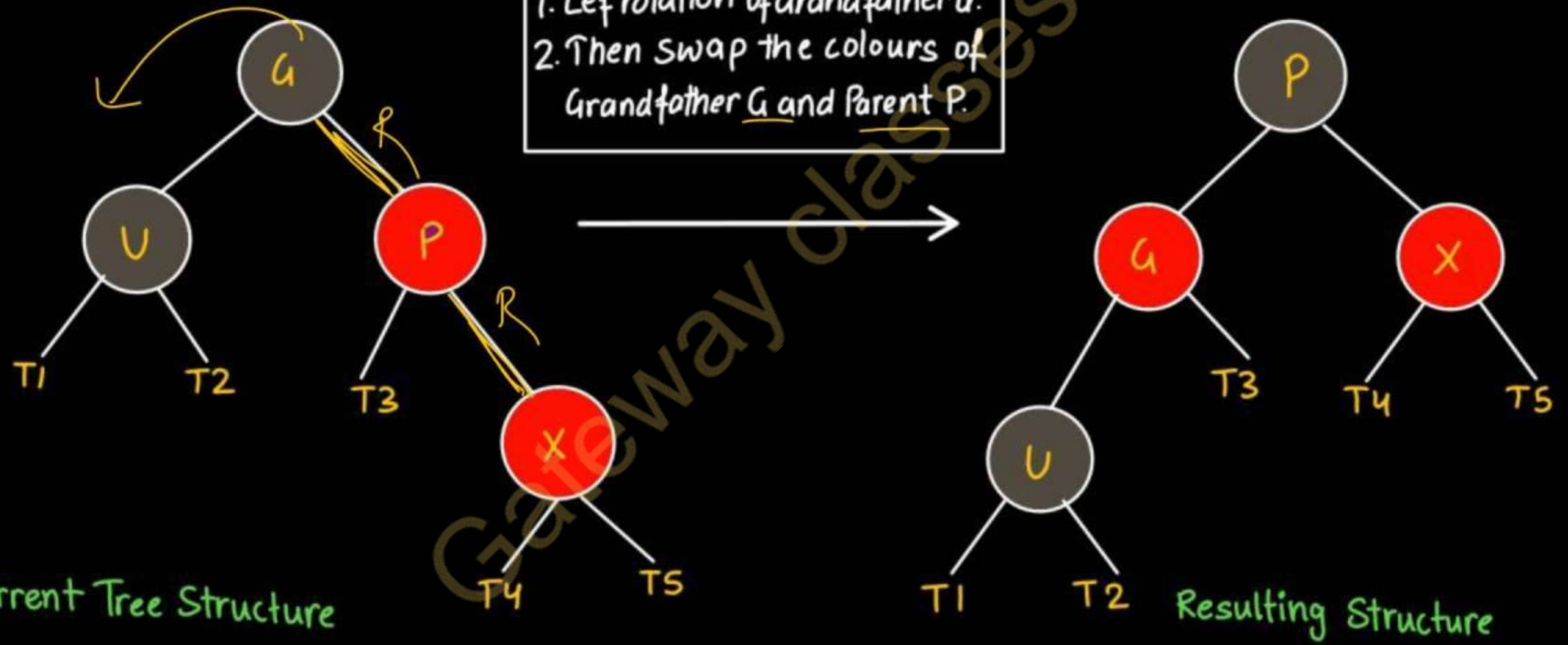
Recolor grandparent and new node.



...When Uncle node is BLACK

Case 3: RR rotation(What do you mean by left rotation in RB tree?)

1. Left rotation of Grandfather G.
2. Then swap the colours of Grandfather G and Parent P.

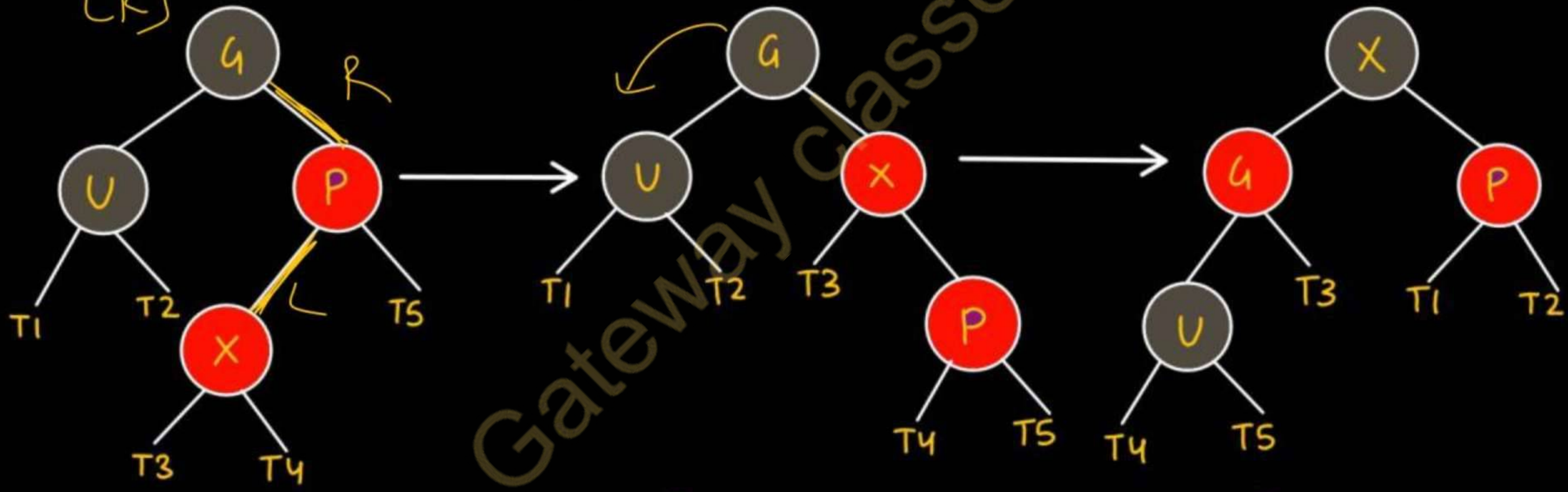


...When Uncle node is BLACK

Case 4: RL rotation

RL
LR
new, G

1. Right rotate Parent P.
2. Now apply RR rotation case.



Current Tree structure

Intermediate Structure

Resulting Structure

Steps for Insertion into RB tree

Let x be the newly inserted node.

1. Perform BST insertion and make newly inserted nodes as RED.
2. If x is the root, make it BLACK.
3. Do the following if the color of x's parent is not BLACK **and** x is not the root.
    a) If x's uncle is RED
        (i) Change the color of parent and uncle as BLACK.
        (ii) Color of a grandparent as RED.
        (iii) Change x = x's grandparent, repeat steps 2 and 3 for new x.

Steps for Insertion into RB tree Contd..

b) If x's uncle is BLACK, then there can be four configurations for x, x's parent and x's grandparent

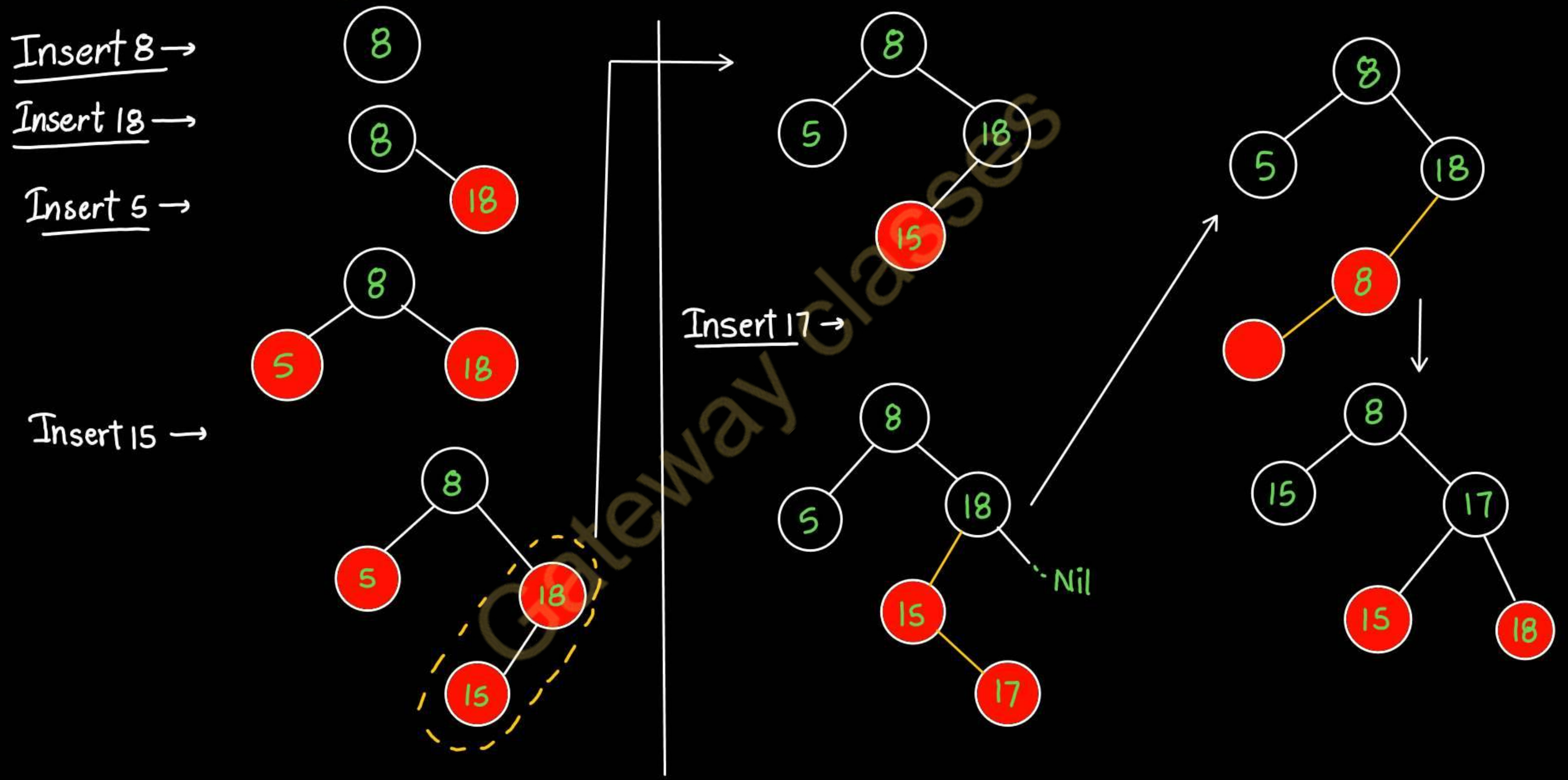
(i) **Left Left Case :** swap colors of grandparent and parent after rotations

(ii) **Left Right Case :** swap colors of grandparent and inserted node after rotations

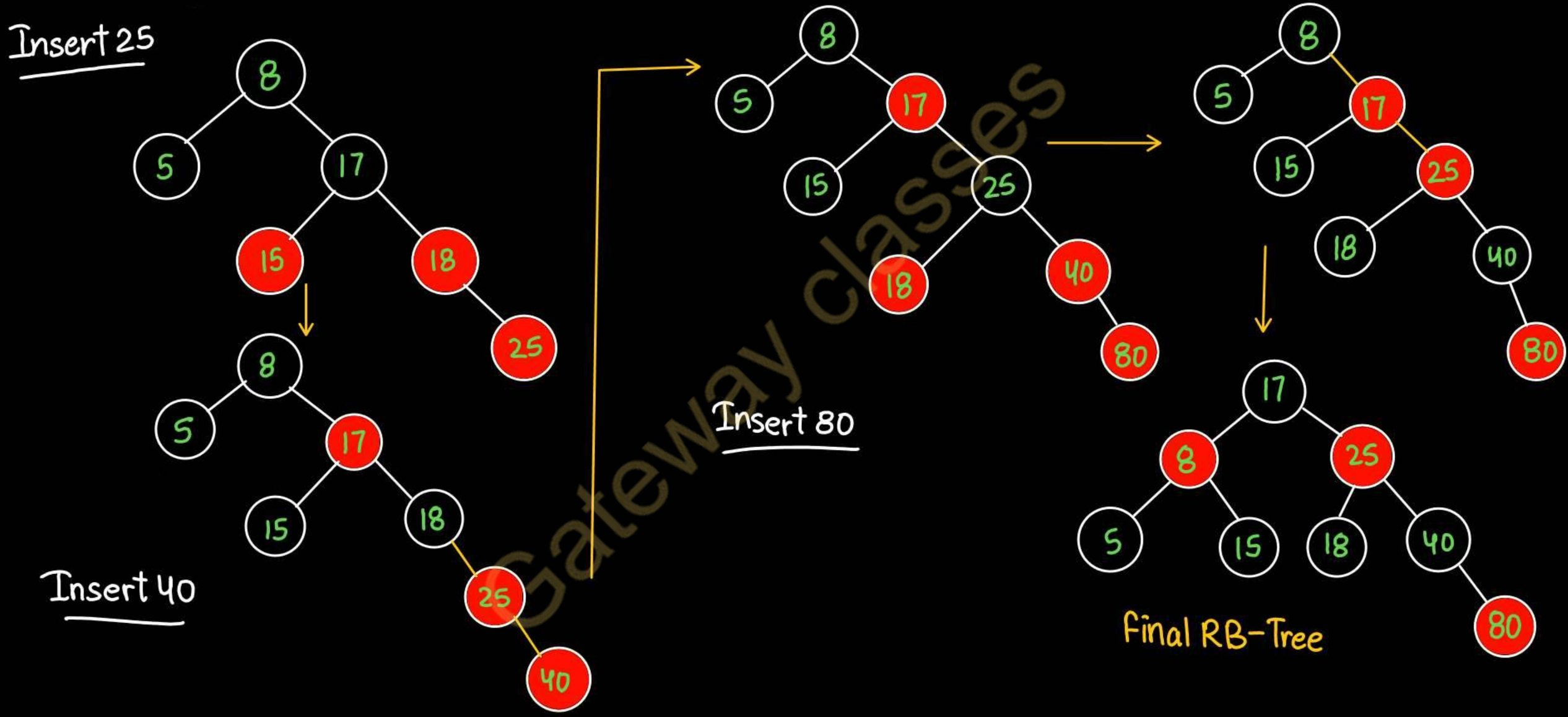
(iii) **Right Right Case :** swap colors of grandparent and parent after rotations

(iv) **Right Left Case:** swap colors of grandparent and inserted node after rotations

Insert the following elements into a RB tree: 8,18,5,15,17,25,40,80



Insert the following elements into a RB tree: 8,18,5,15,17,25,40,80



RB-Tree Insertion Pseudocode

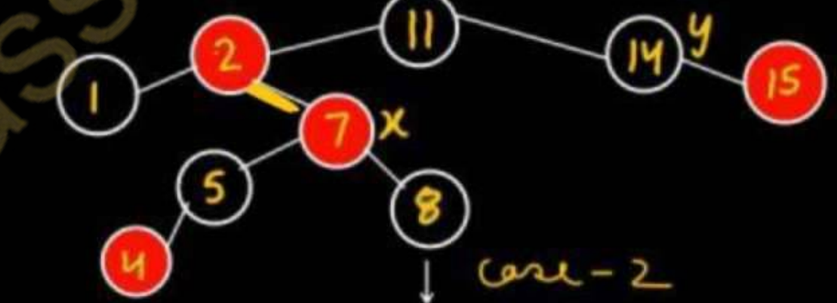
```
RB-INSERT(T,x)
1. TREE-INSERT(T,x)  // Insert x into BST(T)
2. color[x] ← RED
3 while x ≠ root[T] and color[p[x]] = RED
4.     do if p[x] = left[p[p[x]]]
5.         then y ← right[p[p[x]]]
6.              if color[y] = RED
7.                 then color[p[x]] ← BLACK          → Case 1
8.                      color[y] ← BLACK             → Case 1
9.                      color[p[p[x]]] ← RED         → Case 1
10.                     x ← p[p[x]]                  → Case 1
11.             else if x = right[p[x]]
12.                    then x ← p[x]                 → Case 2
13.                         LEFT-ROTATE(T,x)         → Case 2
14.                 color[p[x]] ← BLACK              → Case 3
15.                 color[p[p[x]]] ← RED             → Case 3
16                  RIGHT-ROTATE(T,p[p[x]])          → Case 3
17.    else (same as then clause
            with "right" and "left" exchanged)
18. color[root[T]] ← BLACK
```

p[x] ⇒ parent
p[p[x]] ⇒ grandparent

(a)

Case 1

(b)

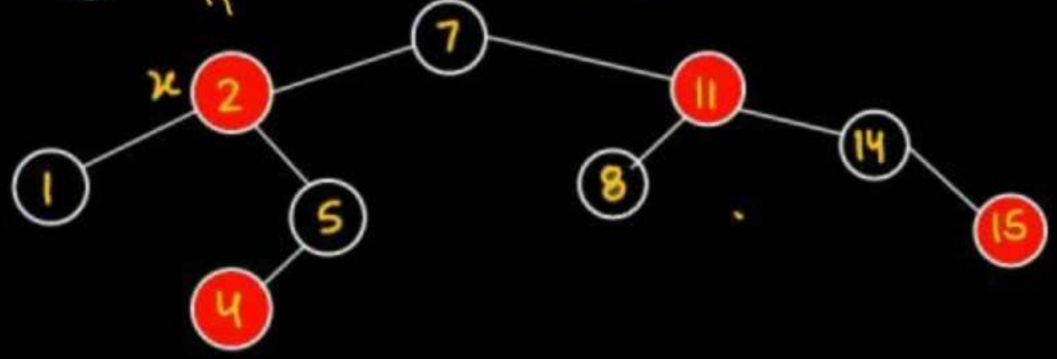


Case - 2

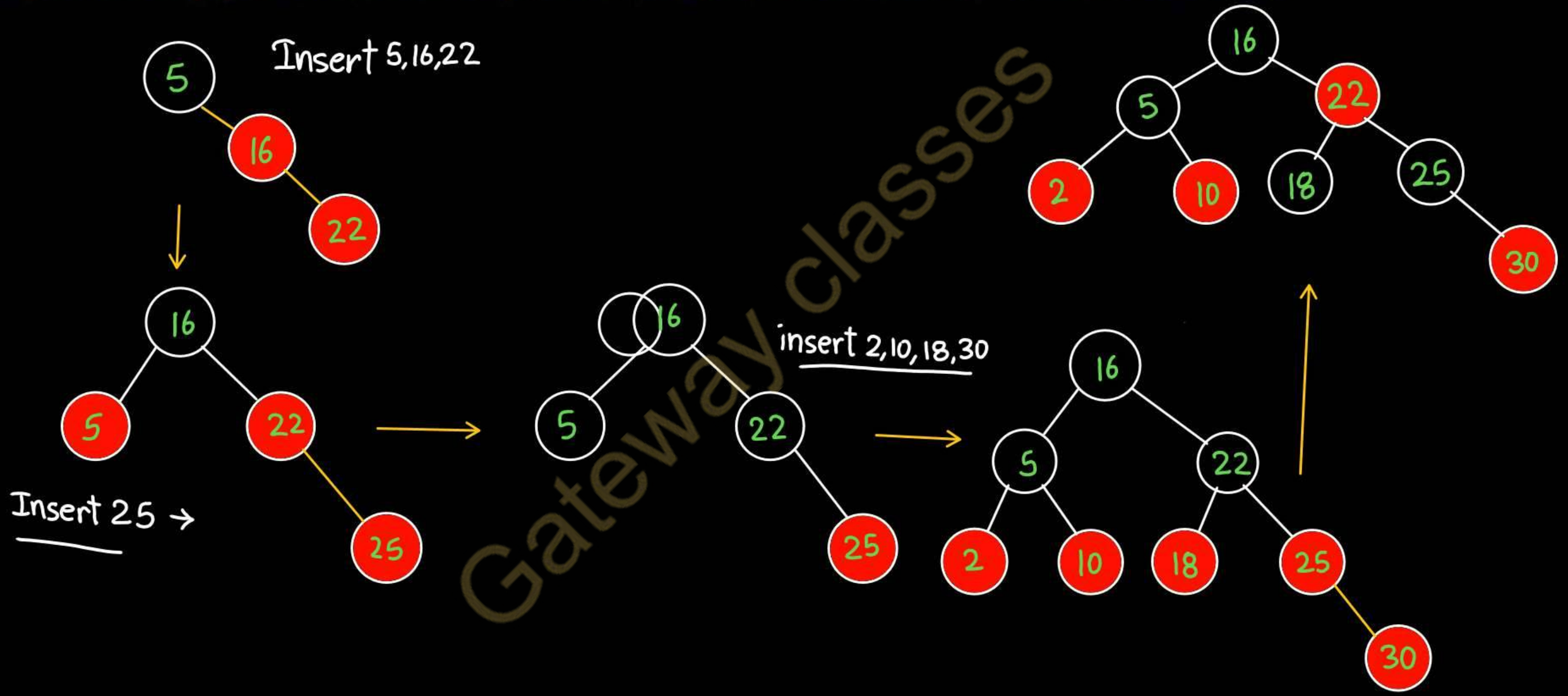
(c)

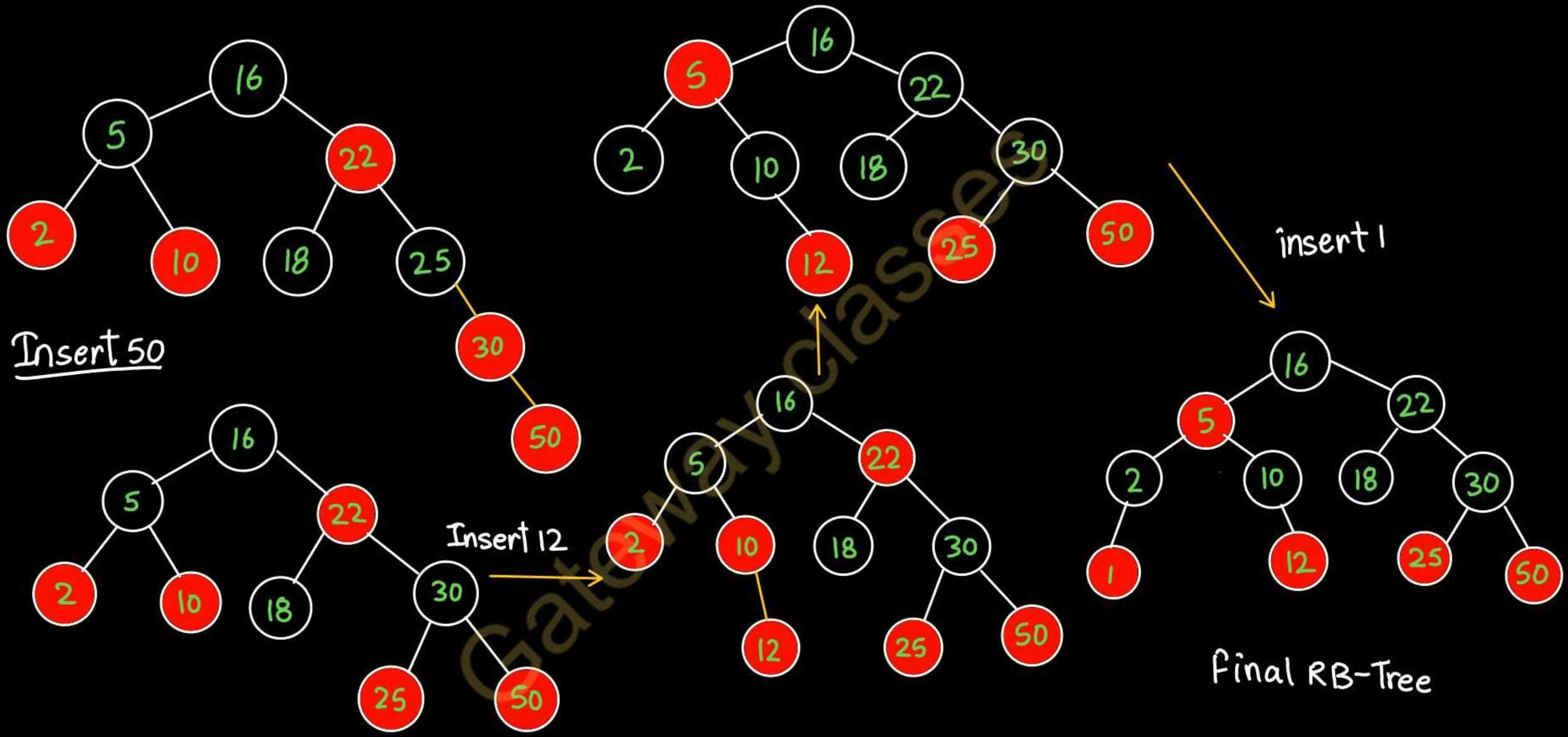
Case - 3

(d)



Question: Discuss the various cases for insertion of key in red-black tree for given sequence of key in an empty red-black tree- {5,16,22,25,2,10,18,30,50,12,1}. (AKTU-2020-21)





AKTU PYQs

1. Insert the following elements using the properties of RB tree: 61,58,51,32,39,29. (AKTU 2023-24)
2. Discuss the various cases for insertion of key in red-black tree for given sequence of key in an empty red-black tree- {15,13,12,16,19,23,5,8}. Also show that a red-black tree with **n** internal nodes has height at most **2log(n+1).** **(AKTU 2021-22)**(AKTU 2022-23)
3. Explain left rotation in RB tree. (AKTU-2021-22)
4. Write algorithm for insertion in RB tree. Discuss the various cases for insertion of key in red-black tree for given sequence of key in an empty red-black tree- {5,16,22,25,2,10,18,30,50,12,1} .(AKTU-2020-21)
5. Insert the following elements in the empty RB tree:{12,9,81,76,23,43,65,88,76,32,54}.Now delete 23 and 81 . (AKTU-2019-20)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-3

Today's Target

- Deletion operation in Red-Black Tree
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

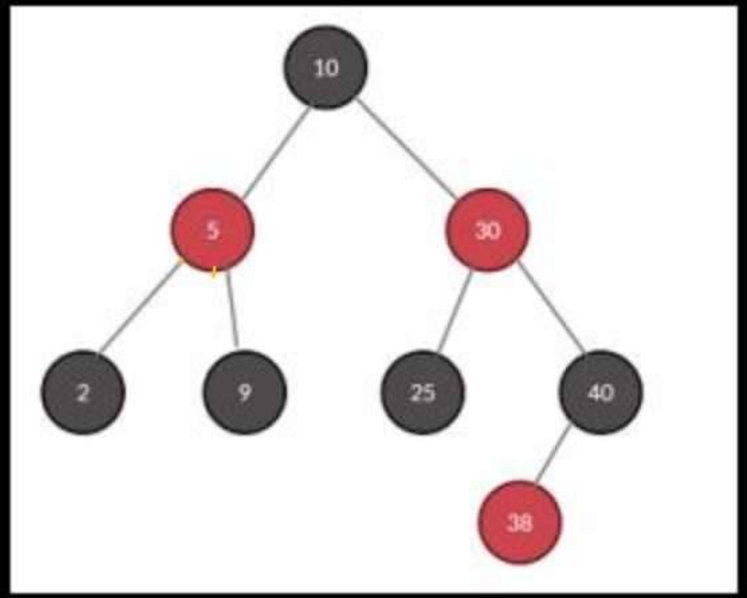
Deletion in Red Black Tree

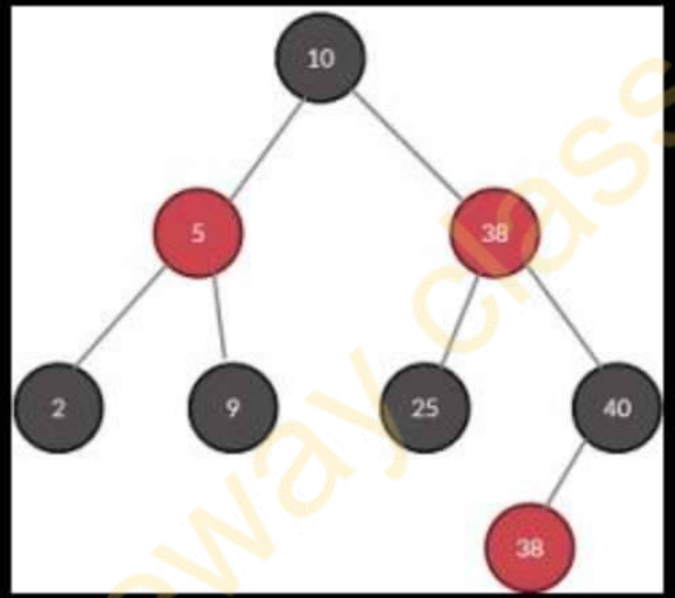
Case #	Condition	Action
1.	If node to be delete is a red leaf node	Just remove it from the tree
2.	If DB node is root	Remove the DB and root node becomes black.
3.	(a) If DB's sibling is black, and (b) DB's sibling's children are black	(a) Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) (b) Make DB's sibling red. (c) If DB's parent is black, make it DB, else make it black
4.	If DB's sibling is red	(a) Swap color DB's parent with DB's sibling (b) Perform rotation at parent node in direction of DB node. (c) Check which case can be applied to this new tree and perform that action

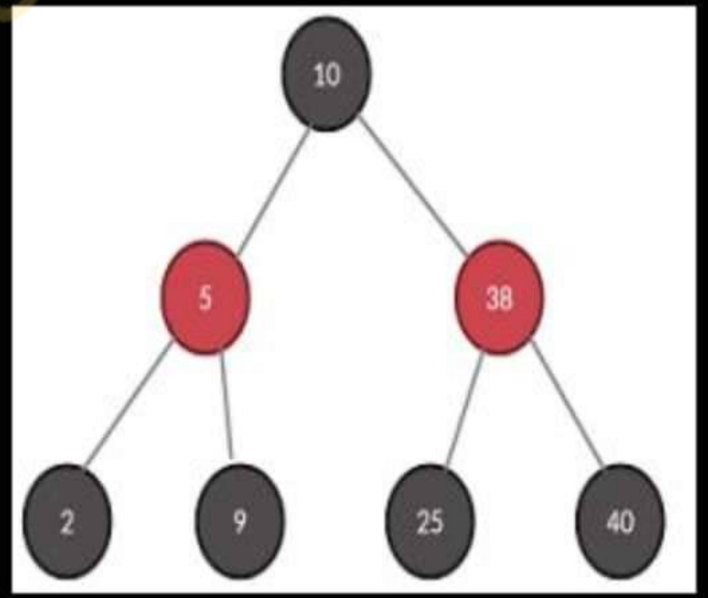
Case #	Condition	Action
5.	(a) DB's sibling is black (b) DB's sibling's child which is far from DB is black (c) DB's sibling's child which is near to DB is red	(a) Swap color of sibling with sibling's red child (b) Perform rotation at sibling node in direction opposite of DB node (c) Apply case 6
6.	(a) DB's sibling is black, and (b) DB's sibling's far child is red (remember this node)	a) Swap color of DB's parent with DB's sibling's color (b) Perform rotation at DB's parent in direction of DB (c) Remove DB sign and make the node normal black node (d) Change colour of DB's sibling's far red child to black.

Example 1: Delete 30 from the RB tree

Case 1 is applicable: If node to be deleted in red leaf node then just remove it.

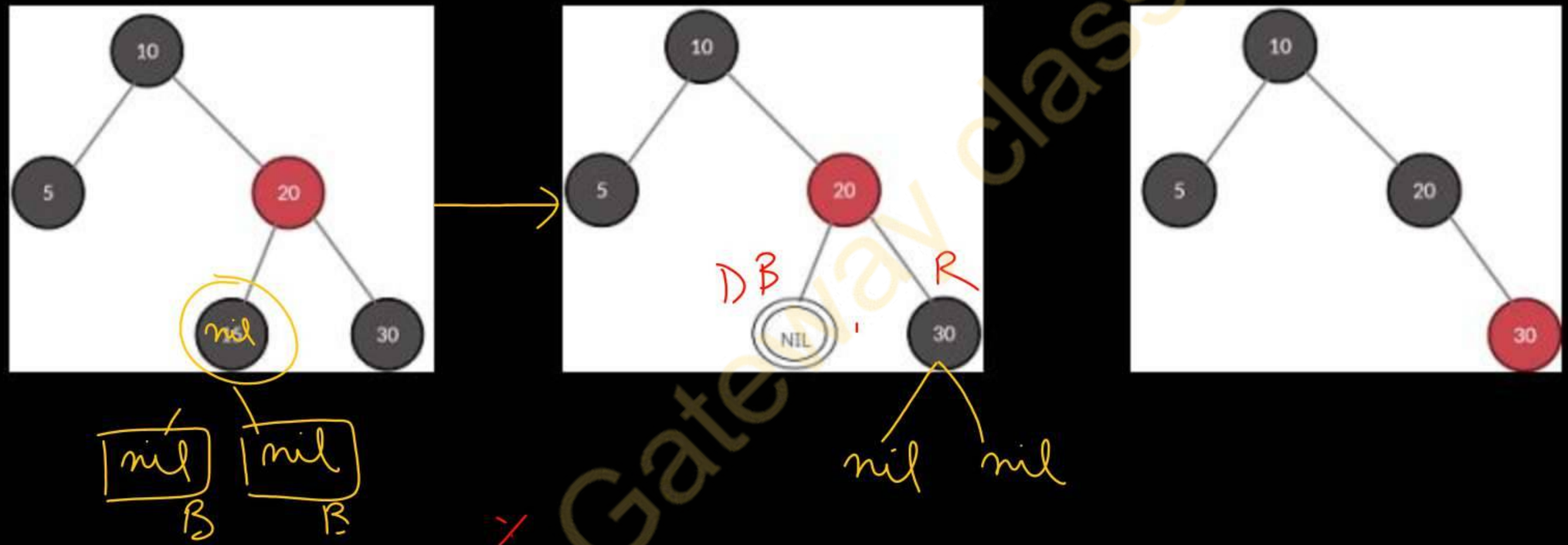






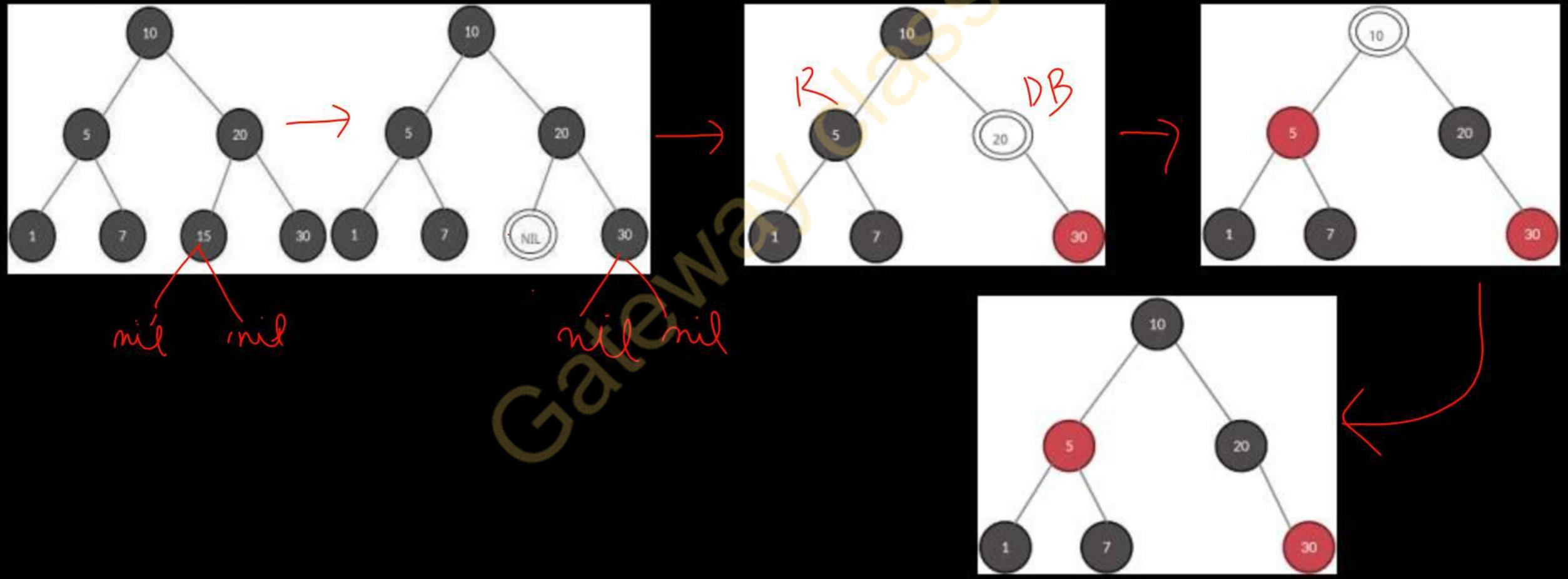
Example 2: Delete 15 from RB tree

Case 3 is applicable: If DB's sibling is black, and DB's sibling's children are black then…..Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) ,Make DB's sibling red, (If DB's parent is black, make it DB, else make it black)



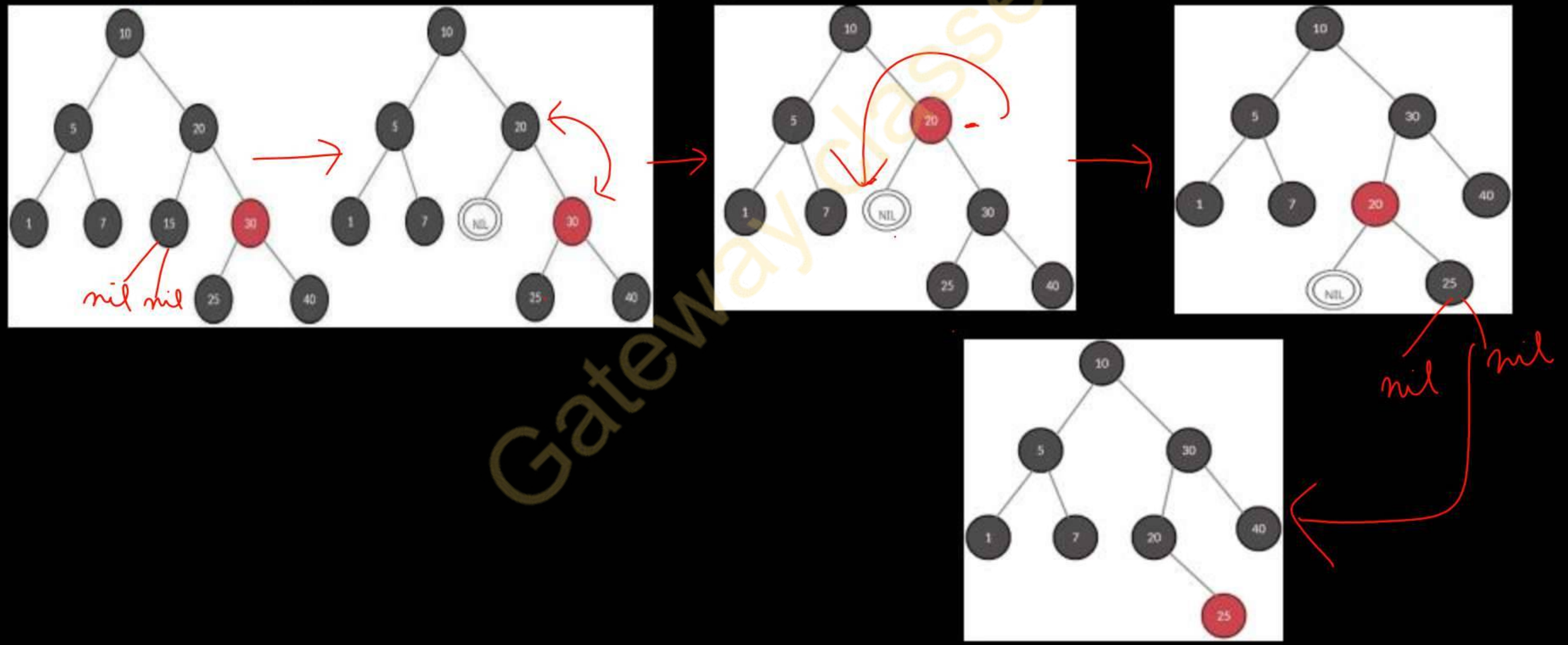
Example 3: Delete 15 from RB tree

Case 3 is applicable: If DB's sibling is black, and DB's sibling's children are black then…..Remove the DB (if null DB then delete the node and for other nodes remove the DB sign) ,Make DB's sibling red, (If DB's parent is black, make it DB, else make it black).Then



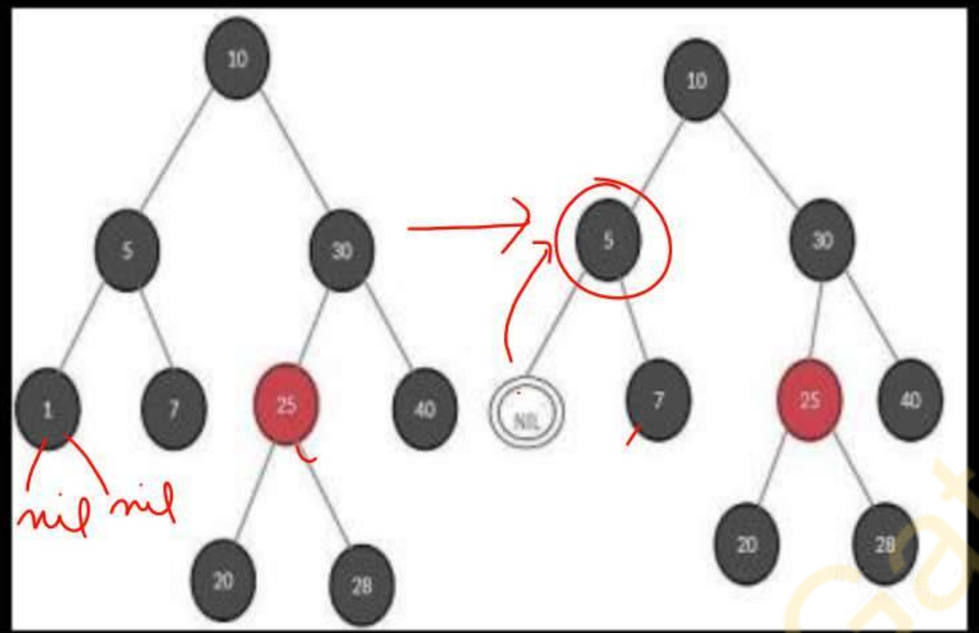
Example 4: Delete 15 from RB tree

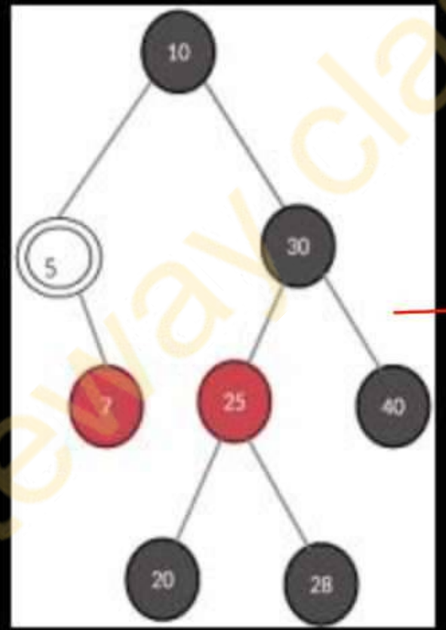
(a) **Case 4 is applicable:** If DB's sibling is red then Swap color of DB's parent with DB's sibling and Perform rotation at parent node in direction of DB node. Then Check which case **(case-3 here)** can be applied to this new tree and perform that action

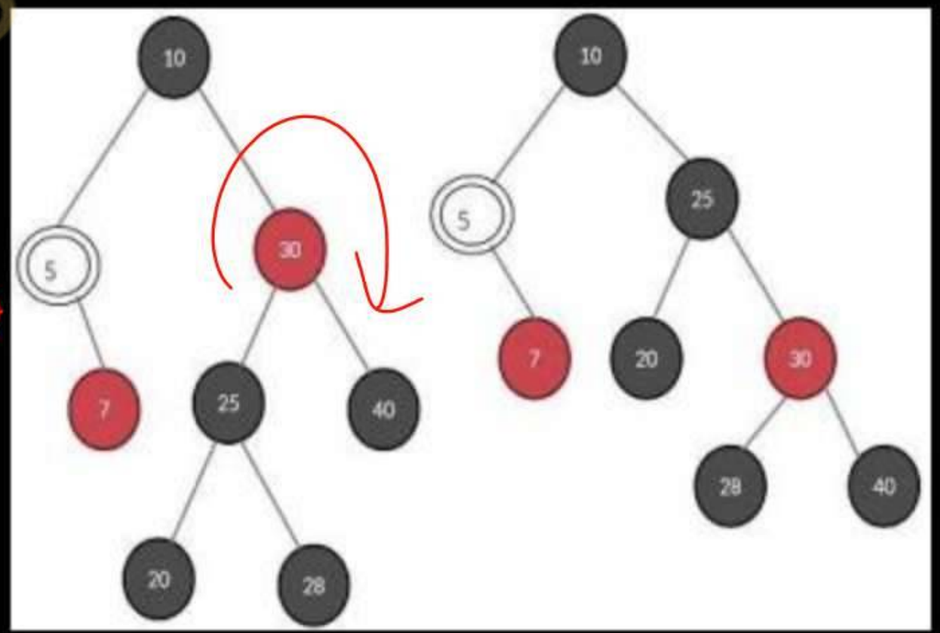


Example 5: Delete 1 from RB tree

Case 5 is applicable: DB's sibling is black, DB's sibling's child which is far from DB is black and DB's sibling's child which is near to DB is red…then Swap color of sibling with sibling's red child, Perform rotation at sibling node in direction opposite of DB node and Apply case 6

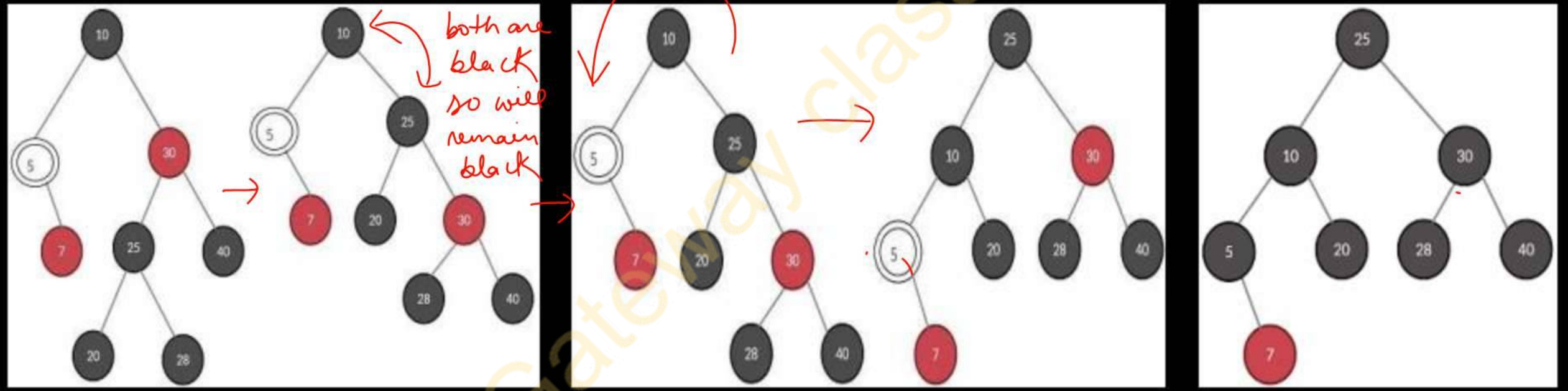




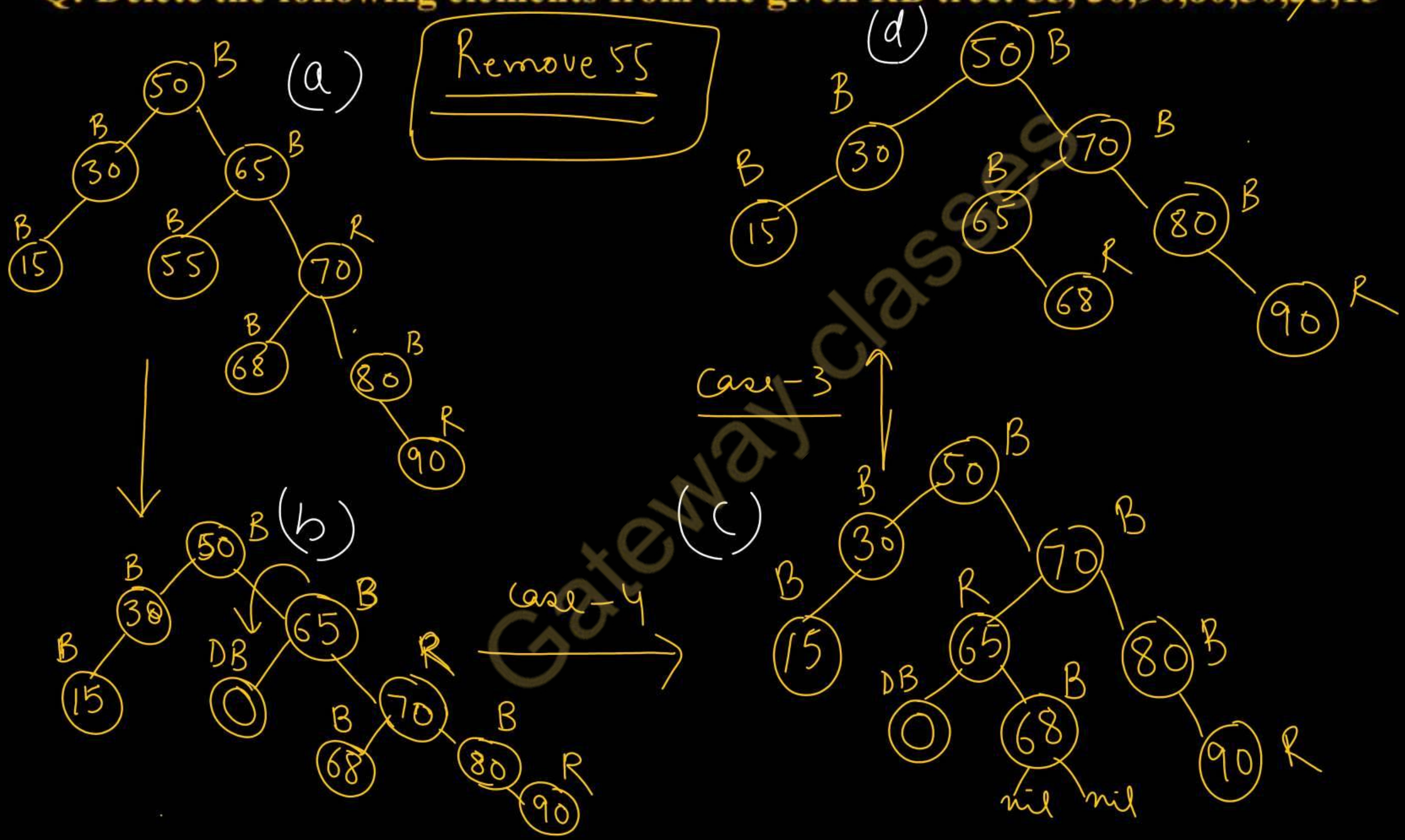


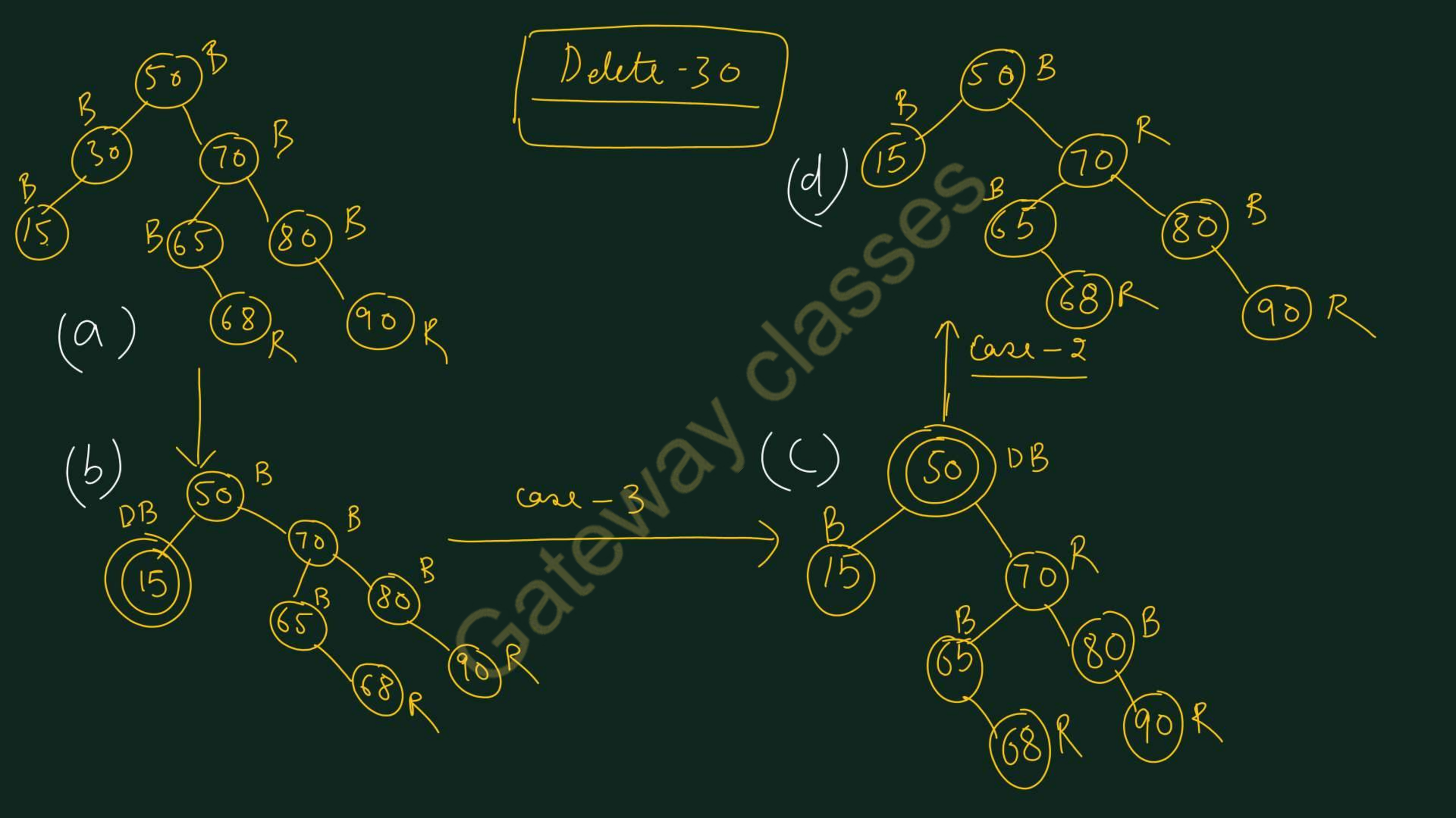
11-3

Case 6 is applicable: DB's sibling is black, and DB's sibling's far child is red…then Swap color of DB's parent with DB's sibling's color, Perform rotation at DB's parent in direction of DB , Remove DB sign and make the node normal black node, and Change colour of DB's sibling's far red child to black.

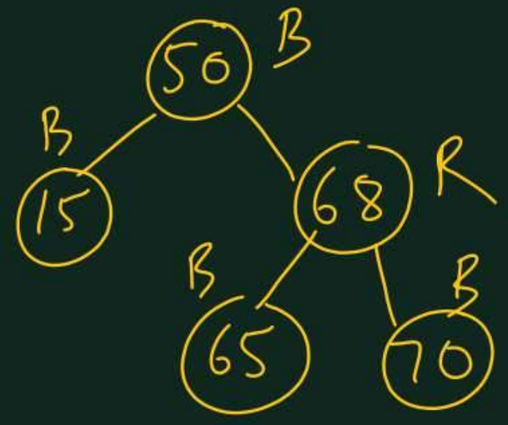


Q: Delete the following elements from the given RB tree: 55, 30,90,80,50,35,15

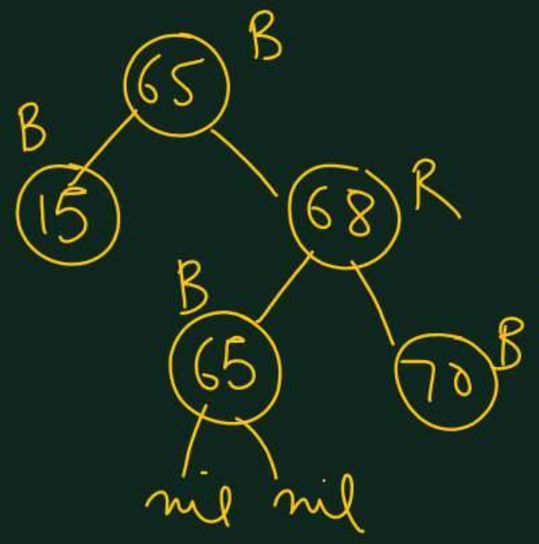


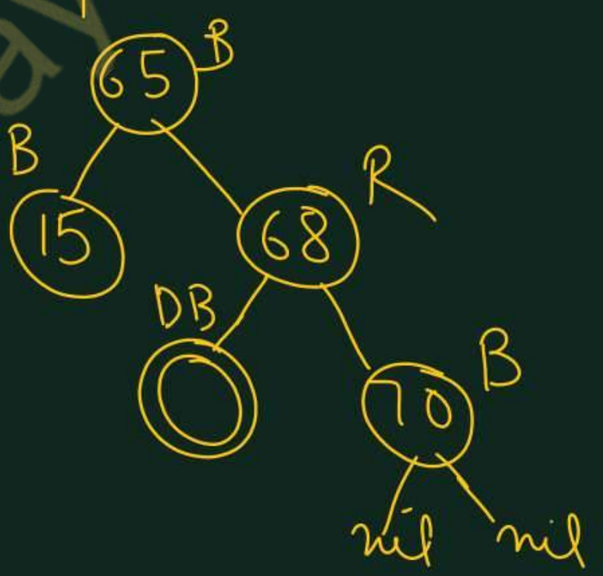

Delete -30
(a)
(b)
(c)
(d)
case - 3
case - 2
Gateway classes


50 B
B 15
70 R
B 65
80 B
R 68
90 R
Remove 90
Remove 80
case-5
case-6
DB
nil
Gateway classes

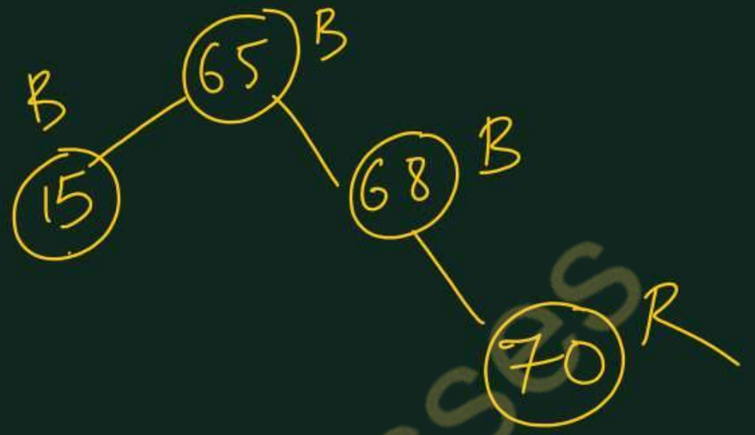


Remove 50





case-3



AKTU PYQs

1. Insert the following elements using the properties of RB tree: 61,58,51,32,39,29.(AKTU 2023-24)

2. Discuss the various cases for insertion of key in red-black tree for given sequence of key in an empty red-black tree- {15,13,12,16,19,23,5,8}. Also show that a red-black tree with **n** internal nodes has height at most **2log(n+1). (AKTU 2021-22)**(AKTU 2022-23)

3. Explain left rotation in RB tree.(AKTU-2021-22)

4.Write algorithm for insertion in RB tree. Discuss the various cases for insertion of key in red-black tree for given sequence of key in an empty red-black tree- {5,16,22,25,2,10,18,30,50,12,1} .(AKTU-2020-21)

5. Insert the following elements in the empty RB tree:{12,9,81,76,23,43,65,88,76,32,54}.Now delete 23 and 81 .(AKTU-2019-20)

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-4

Today's Target

B Tree

- Properties of B-Tree
- Insertion operation into B-Tree
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am

- M.Tech Gold Medalist
- Net Qualified

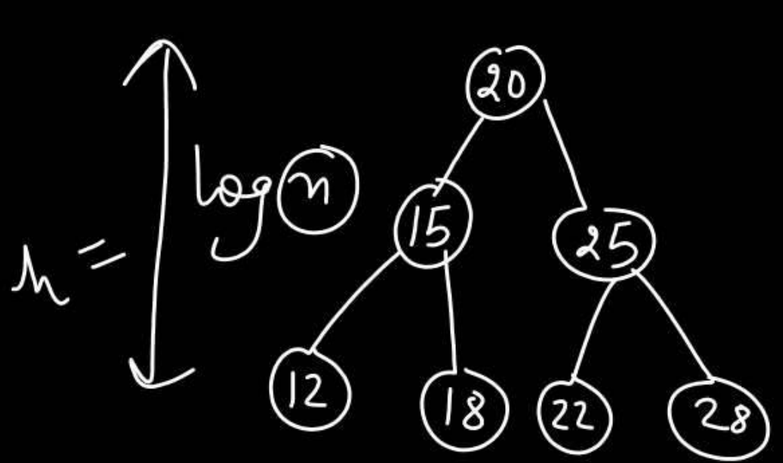
Why B Tree?

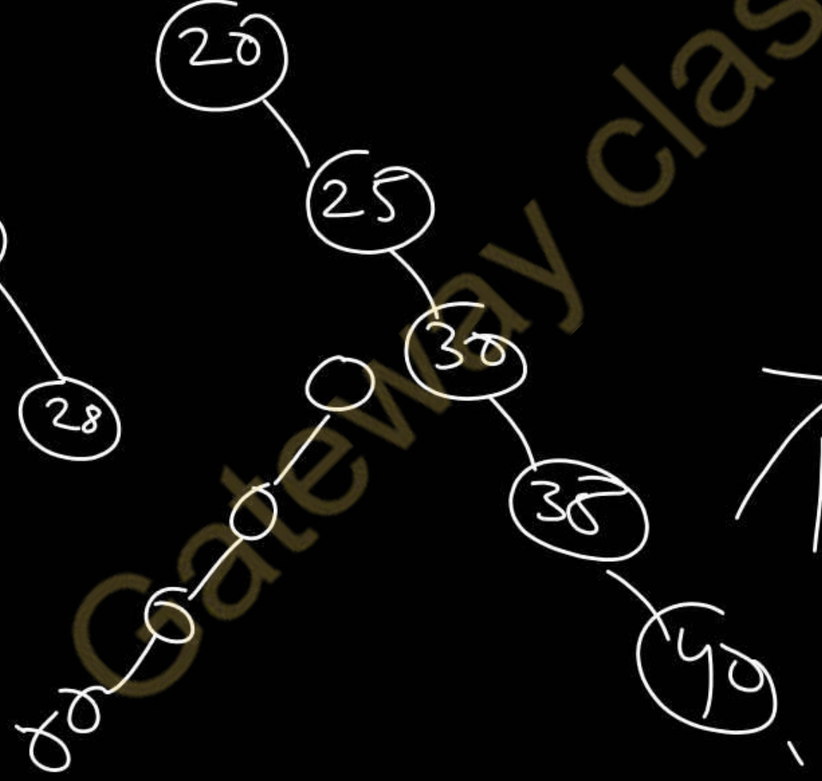
(Balanced Tree)

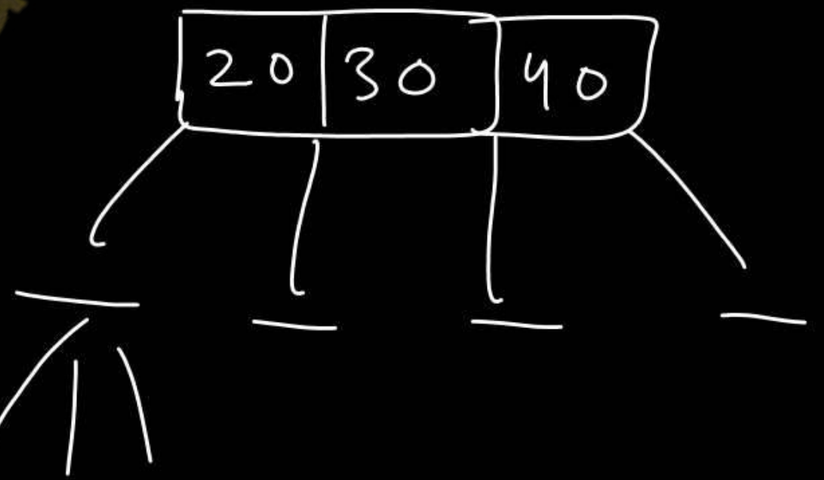
(Self balanced M-way search tree)

M → Order of B-Tree

Binary Search Tree







B Tree (Generalization of BST)

- B-Tree is a self-balanced M-way Search Tree in which every node contains multiple keys and has more than two children.

Properties of B-Tree

- All leaf nodes must be at same level.
- All nodes except root must have at least ceil(M/2)-1 keys and maximum of M-1 keys.
- All non leaf nodes except root (i.e. all internal nodes) must have at least ceil(M/2) children.
- If the root node is a non leaf node, then it must have at least 2 children.
- A non leaf node with n-1 keys must have n number of children.
- All the key values in a node must be in Ascending Order.
- B-Tree grows and shrinks from the root unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
- Time complexity to search, insert and delete is O(logn).

B tree Structure		In terms of order 'M'	In terms of minimum degree 't'
Maximum Keys	All nodes	M-1	2t-1
Maximum Children	All internal nodes (including root)	M	2t
Minimum Keys	All nodes except root	Ceil(M/2)-1	t-1
	Root	1	1
Minimum children	All internal nodes (except root)	Ceil(M/2)	t
	Root	2	2

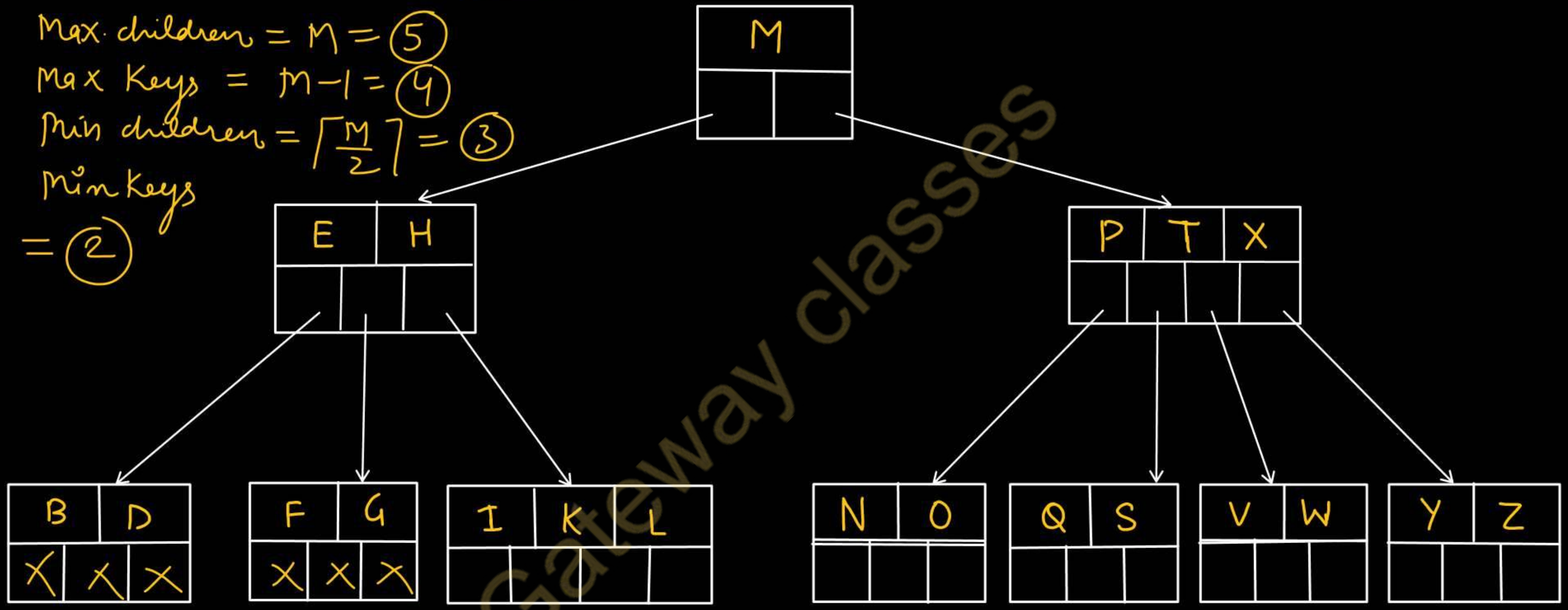
Min children

In a 2-3-4 tree ,any internal node can have either two, three, or four children. It is also called a 2-4 tree.

Max - children

Example: B-Tree of Order 5(M=5)

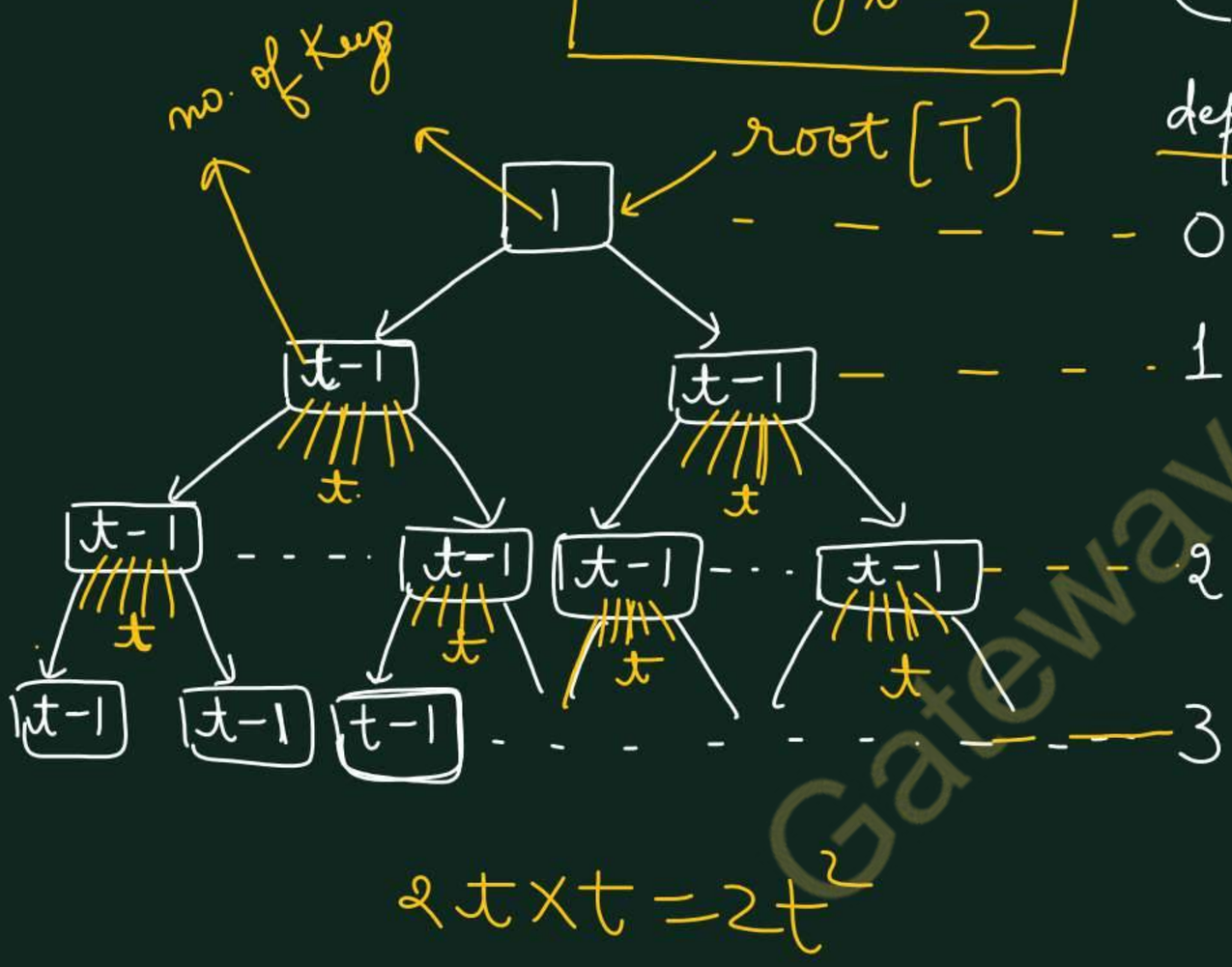
Max. children = M = (5)
Max Keys = M−1 = (4)
Min children = $\lceil \frac{M}{2} \rceil$ = (3)
Min keys = (2)



Proof: For $n \geq 1$, for any n-Key B-Tree T of height h and Minimum degree $t \geq 2$,

$$h \leq \log_t \frac{n+1}{2}$$

(AKTU - 2018 - 19)



no. of keys in one node

$$n \geq 1 + (t-1) \sum_{i=1}^{h} 2 \cdot t^{i-1}$$

$$n = 1 + 2 \cdot (t-1) \left(\frac{t^h - 1}{t-1} \right)$$

$$n = 2 \cdot t^h - 1$$

$$n \geq 1 + (t-1)\left[\sum_{i=1}^{h} 2 \cdot t^{i-1}\right]$$

$$\geq 1 + 2(t-1)\left[\sum_{i=1}^{h} t^{i-1}\right]$$

$$\geq 1 + 2(t-1)\left[t^0 + t^1 + \ldots\ldots t^{h-1}\right]$$

$$\geq 1 + 2(t-1)\left[\frac{t^h - 1}{(t-1)}\right]$$

$$\geq 1 + 2 \cdot t^h - 2$$

$$n \geq 2 \cdot t^h - 1 \Rightarrow (n+1) \geq 2 \cdot t^h$$

$$(n+1) \geq 2 \cdot t^h$$

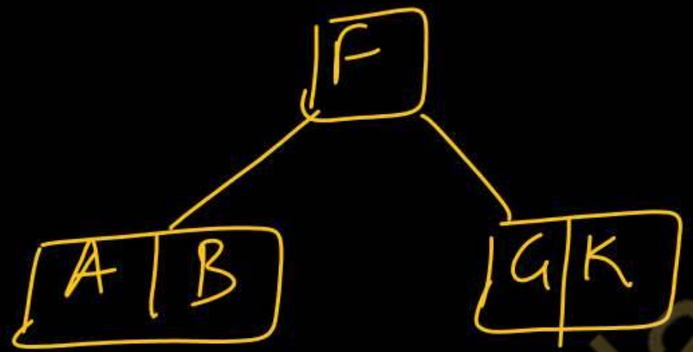
$$t^h \leq (n+1)/2$$

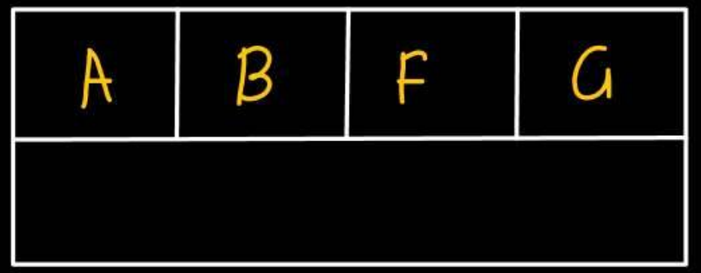
$$\boxed{h \leq \log_t\left(\frac{n+1}{2}\right)}$$

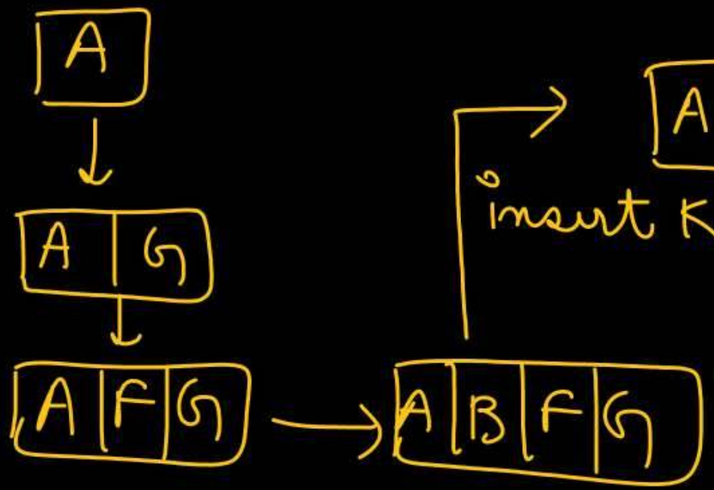
Insert the following letters into what is originally an empty B-tree of order 5:

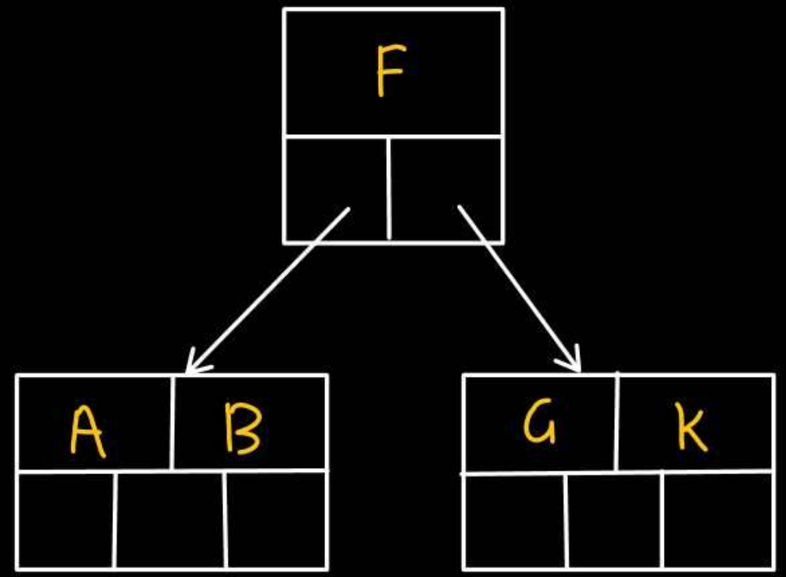
A G F B K D H M J E S I R X C L N T U P.

Max children = 5
max Keys = 4
Min children = 3
Min. Keys = 2

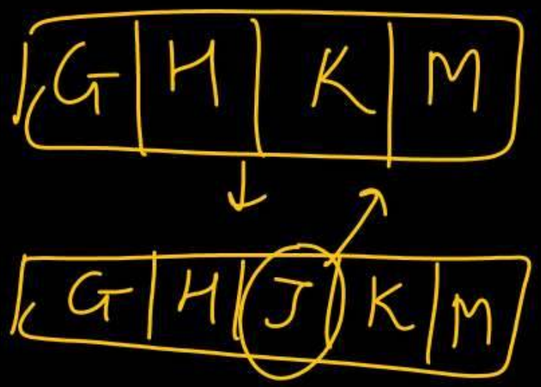


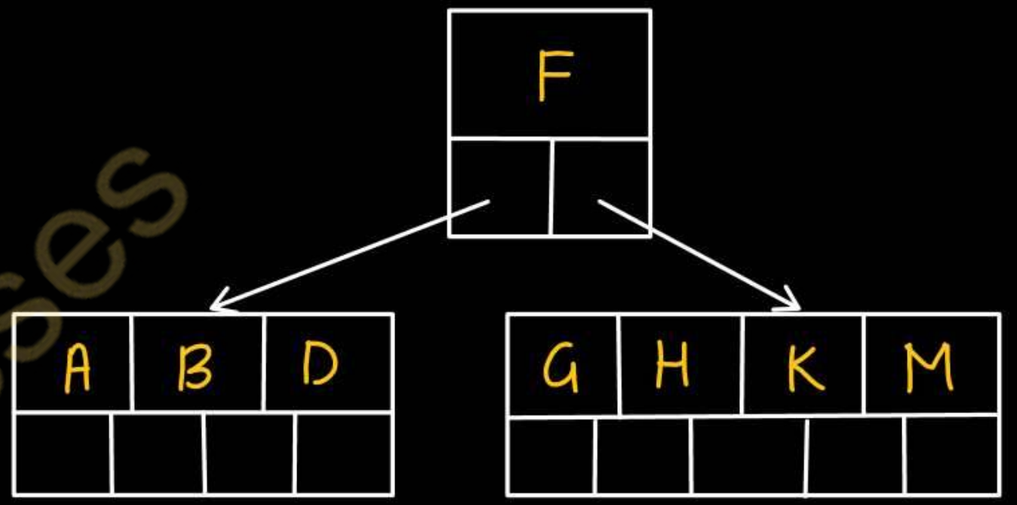


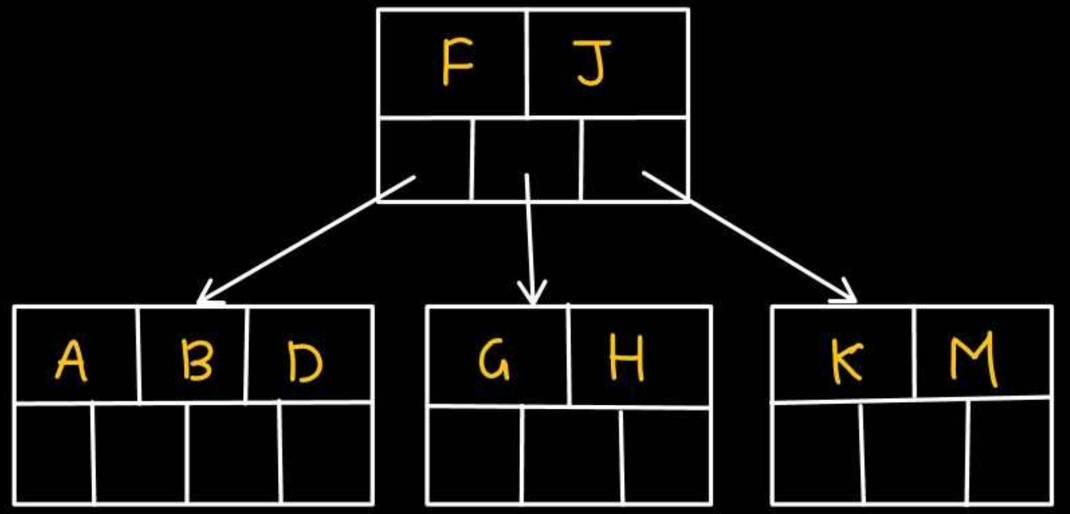




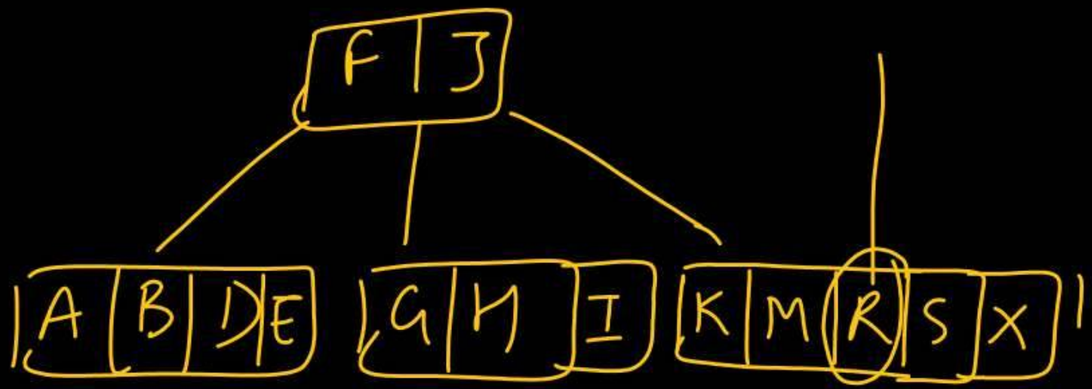
A G F B K D H M J E S I R X C L N T U P.

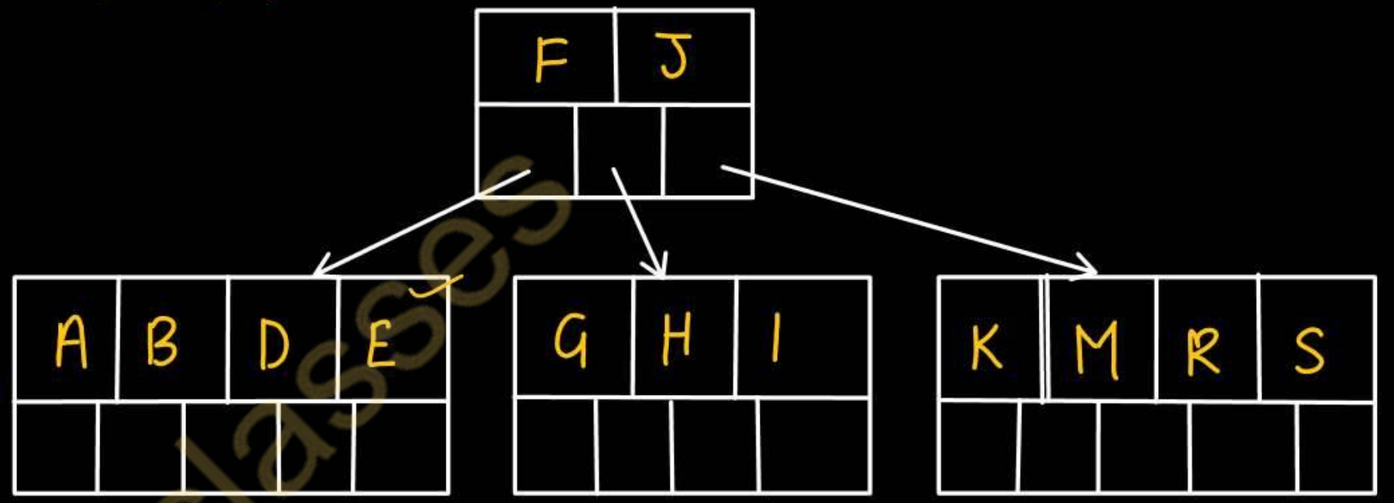


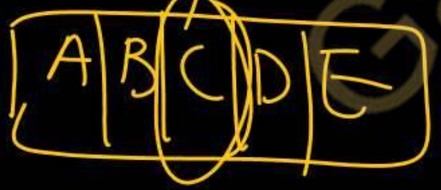


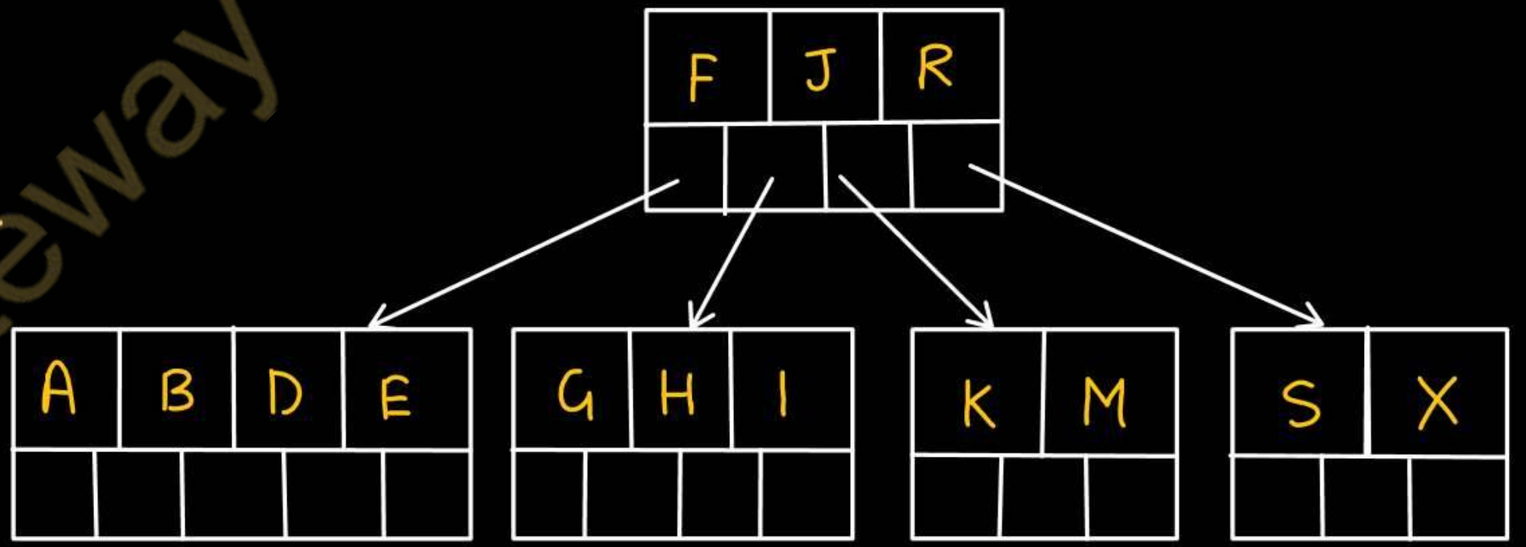


~~A G F B K D H M~~ J E S I R X C L N T U P.

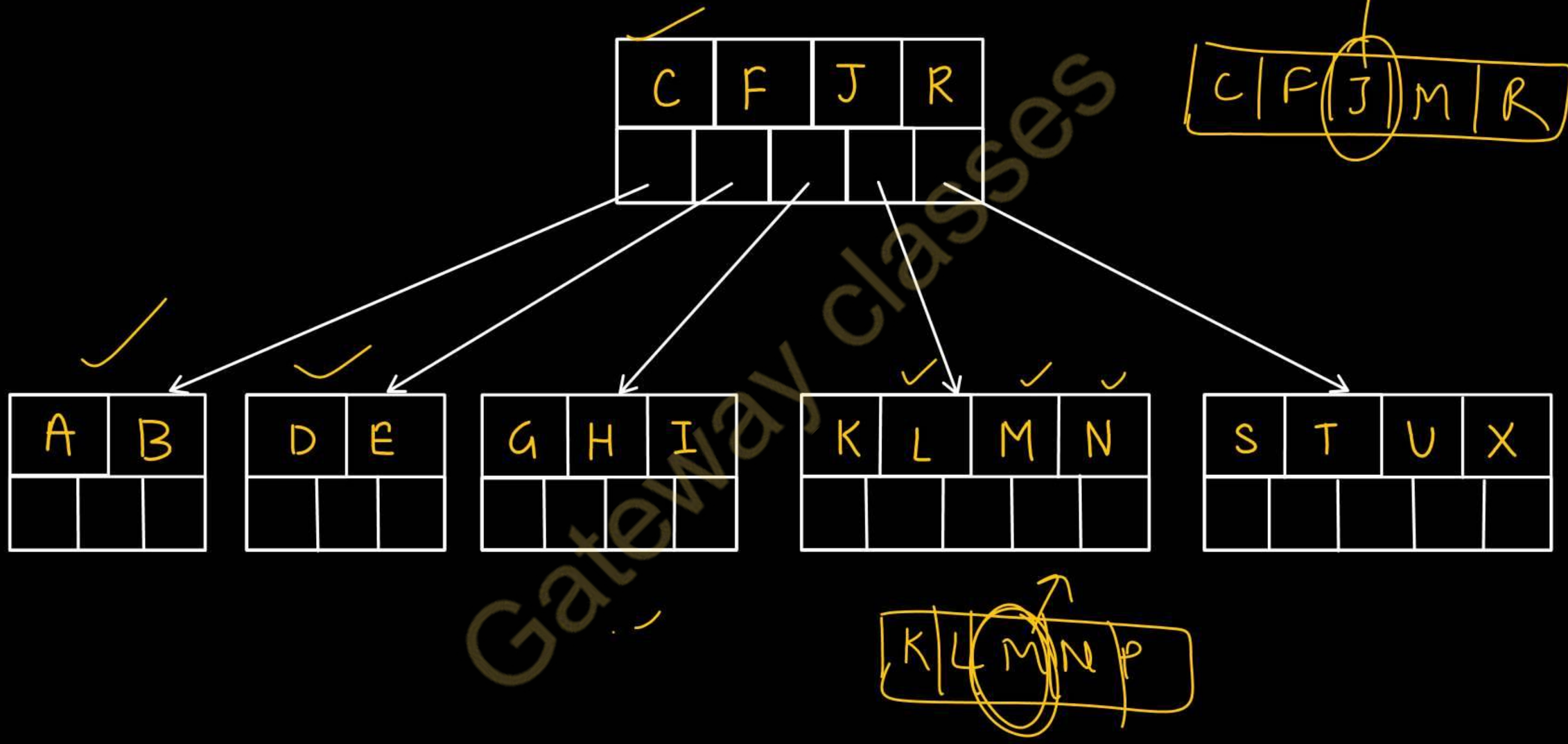




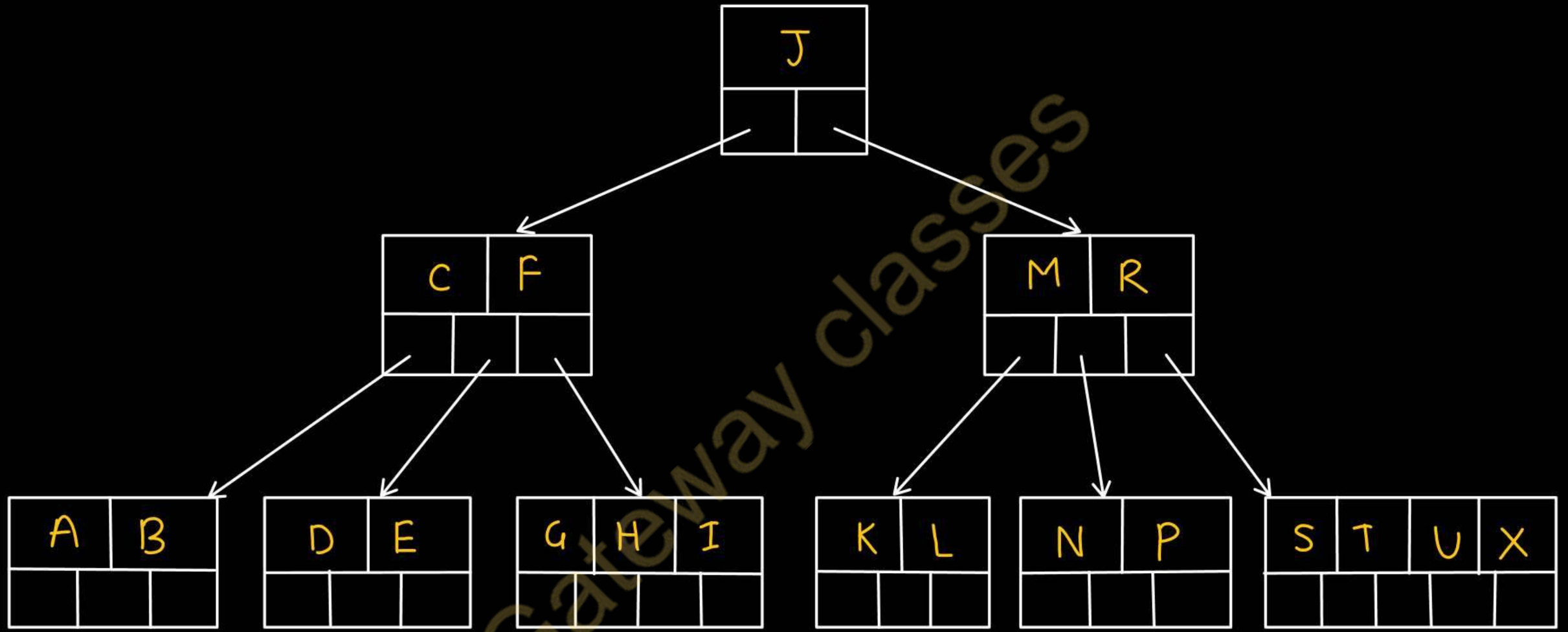




~~A G F B K D H M J E S I R X~~ C L N T U P.

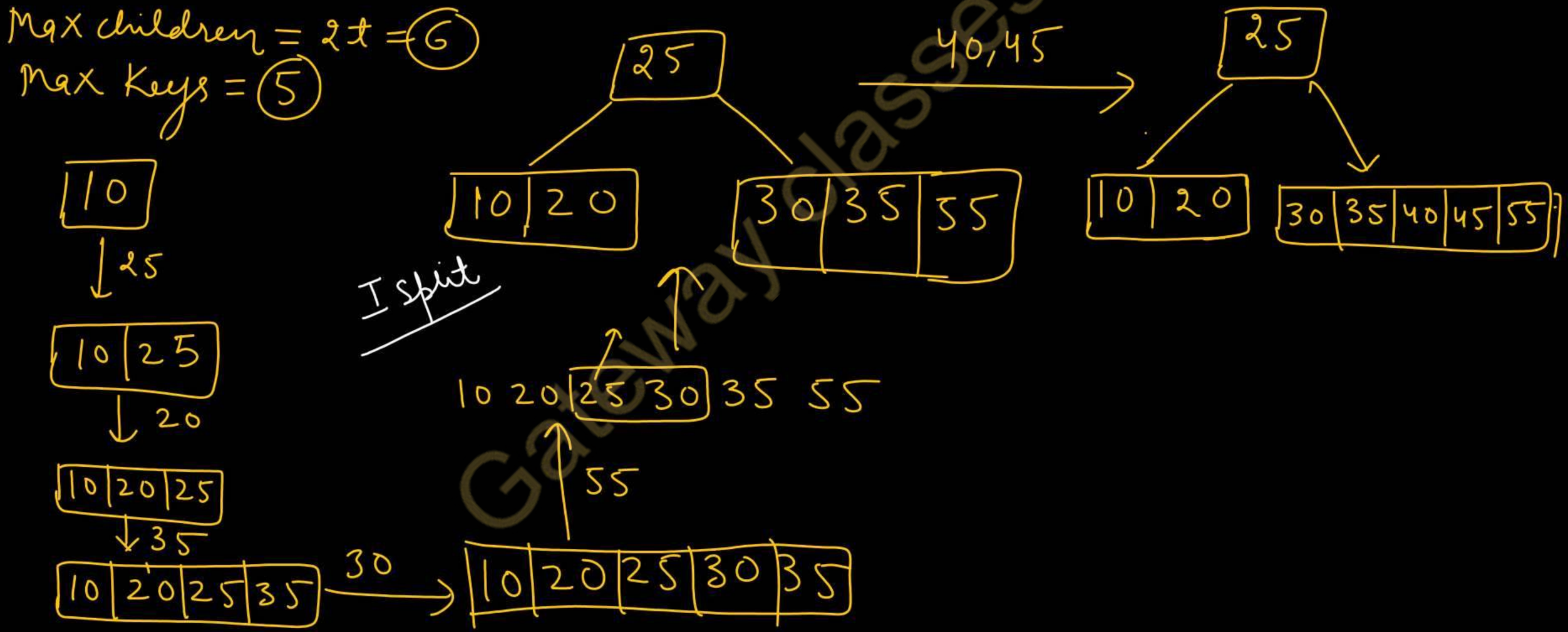


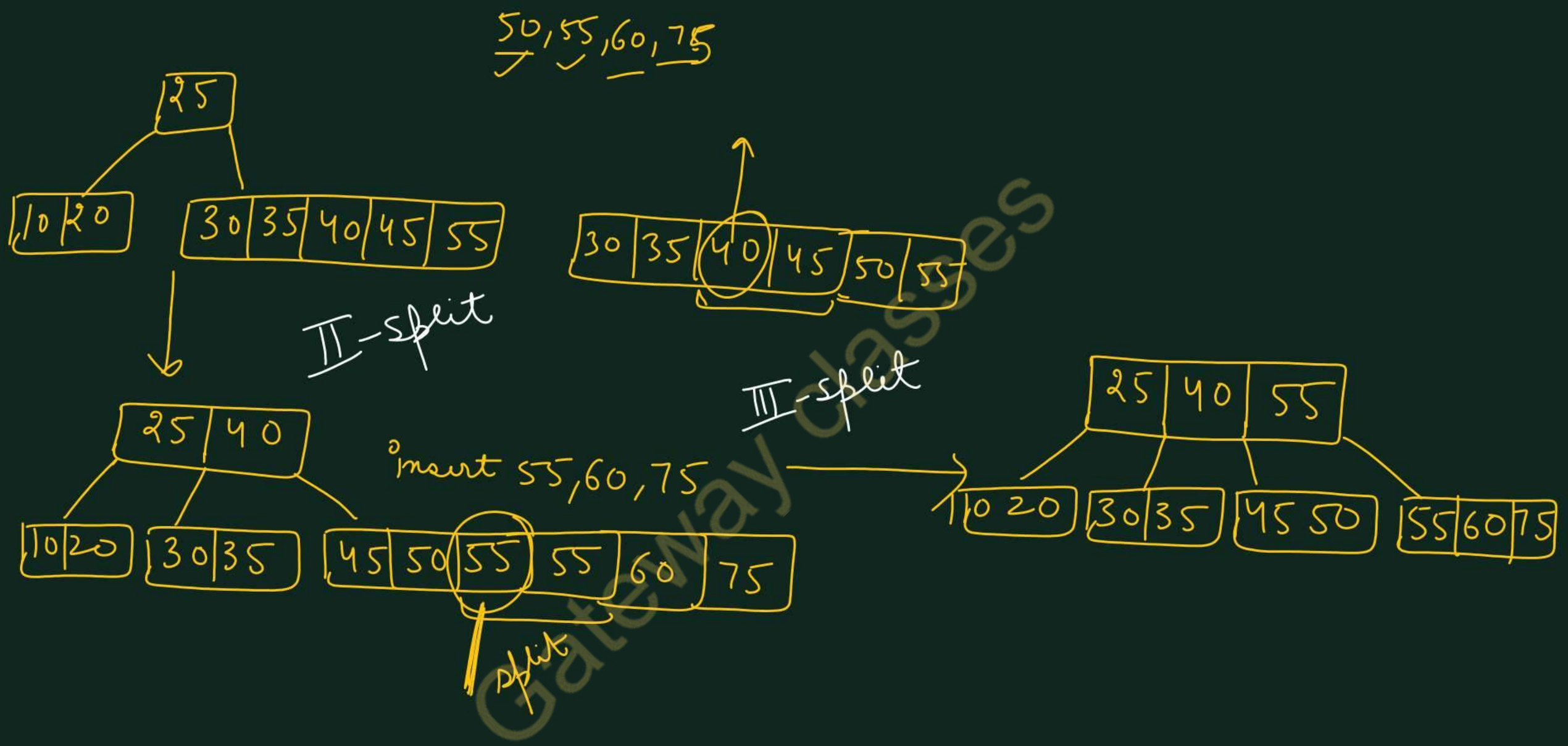
A G F B K D H M J E S I R X C L N T U P.



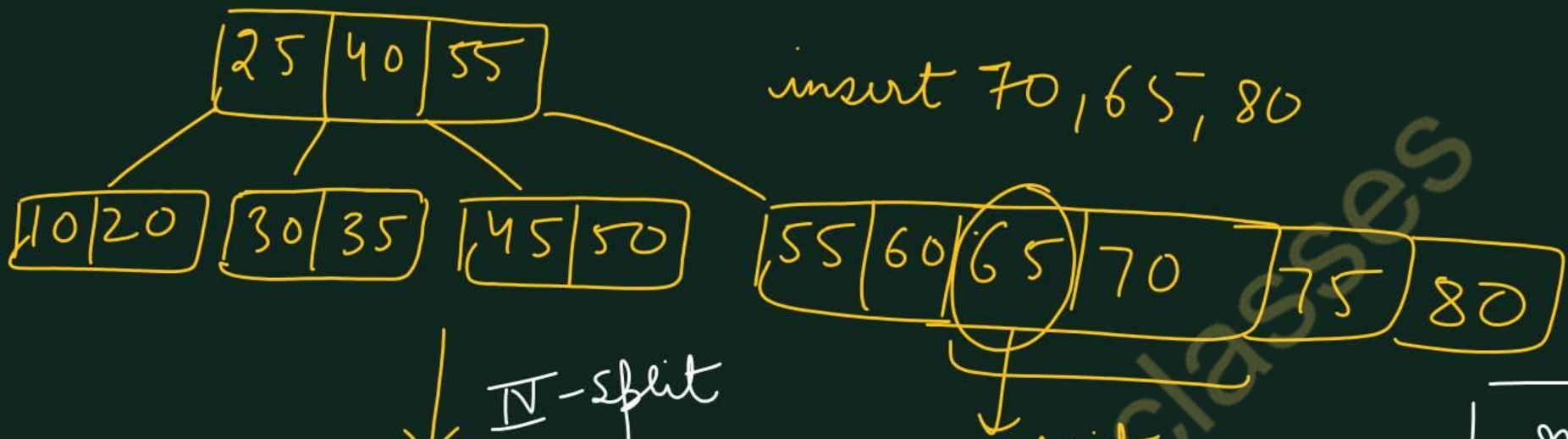
Q: Using minimum t as 3, insert following sequence of integers:

10,25,20,35,30,55,40,45,50,55,60,75,70,65,80,85,90 in initially empty B-Tree. Give number of node splitting operations that take place.(AKTU-2019-20)




50, 55, 60, 75
25
10 20
30 35 40 45 55
II-split
30 35 40 45 50 55
25 40
insert 55, 60, 75
10 20
30 35
45 50 55 55 60 75
split
III-split
25 40 55
10 20
30 35
45 50
55 60 75
Gateway classes

70, 65, 80, 85, 90



insert 70, 65, 80

IV-split

split

no. of splits = 4

insert 85, 90.

AKTU PYQs

1. Using minimum t as 3, insert following sequence of integers: 10,25,20,35,30,55,40,45,50,55,60,75,70,65,80,85,90 in initially empty B-Tree. Give number of node splitting operations that take place. (AKTU-2019-20)
2. Explain B-tree and its properties. Also write B-tree deletion cases with example. (AKTU 2023-24)
3. Show the results of inserting the keys F,S,Q,K,C,L,H,T,V,W,M,R,N,P,A,B,X,Y,D,Z,E in an empty b-tree with t=3. (AKTU 2021-22)
4. Insert the following keys in a *2-3-4 B Tree*: 40, 35, 22, 90, 12, 45, 58, 78, 67, 60 and then delete key 35 and 22 one after other. (AKTU 2018-19)

M=4

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-5

Today's Target

- Deletion operation in B-Tree
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

B-Tree Deletion Cases

Case 1: x is a leaf node and contains the target key to be deleted.

1.1. Leaf node contain more than min no. key then simply delete that key form node.

1.2. Leaf node contain min number of keys then 3 subcases:

1.2.1. Borrow from left sibling iff that sibling contain more than min number of keys. Move max key of left sibling to parent node and parent key to target node and then delete target key.

1.2.2. Borrow from right sibling iff that sibling contain more than min number of keys. Move minimum key of right sibling to parent node and parent key to target node and then delete target key.

1.2.3. Neither left nor right sibling contain more than min keys then merge both the siblings with parent and then delete target key.

Case 2: x is an internal node and contains the target key. There are 3 sub-cases:

(left child)

2.1. predecessor child node has more than min no. of keys then replace the key with its in-order predecessor

(Right child)

2.2. successor child node has more than min no. of keys then replace the key with its in-order successor

2.3. Neither predecessor nor successor child has more than min no. of keys then merge the both the predecessor and successor node with parent and then delete the key. After merging if the parent node has less than the minimum number of keys then, look for the siblings as in Case 1.

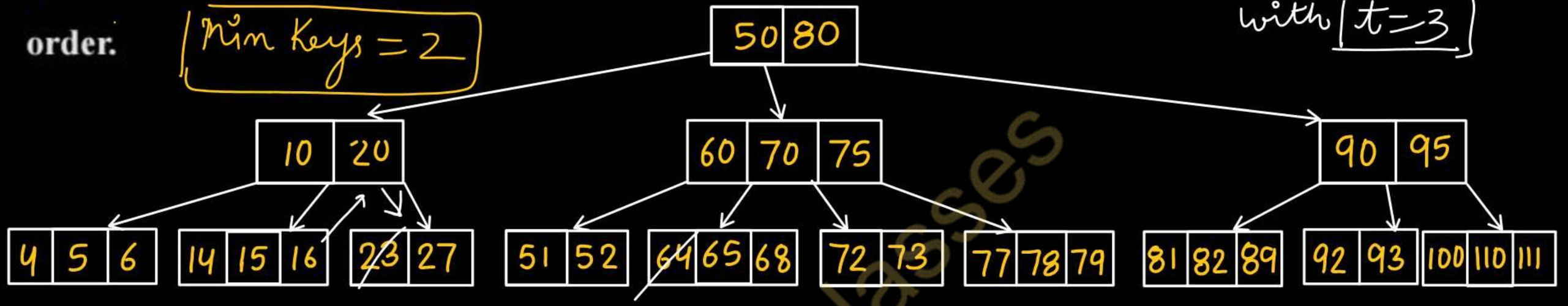
$t=3$

2

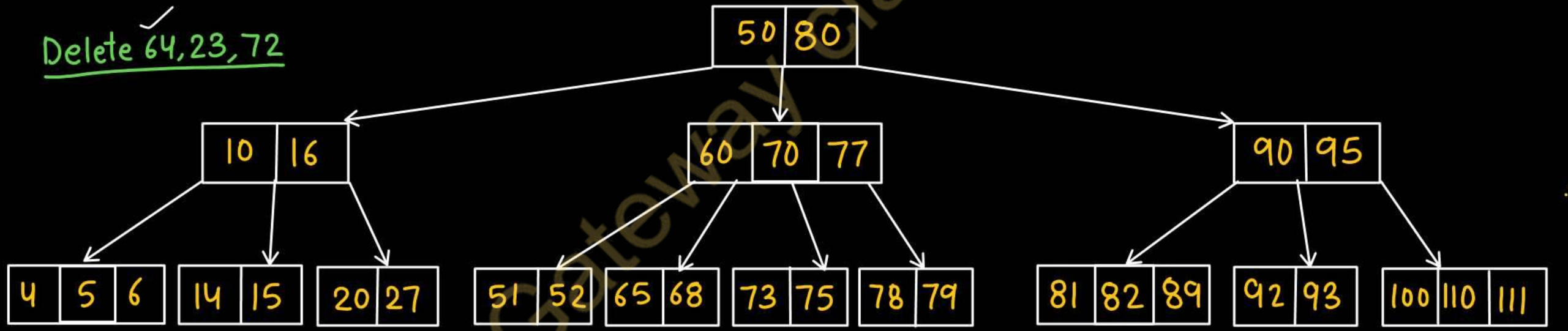
Q: Delete 64, 23, 72, 65, 20, 70, 95, 77, 80, 100, 6, 27, 60, 16, and 50 from the B tree in the given order.

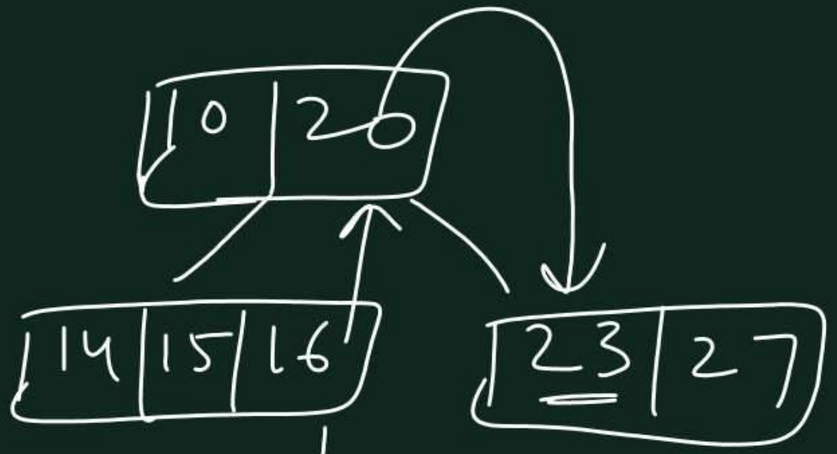
Min Keys = 2

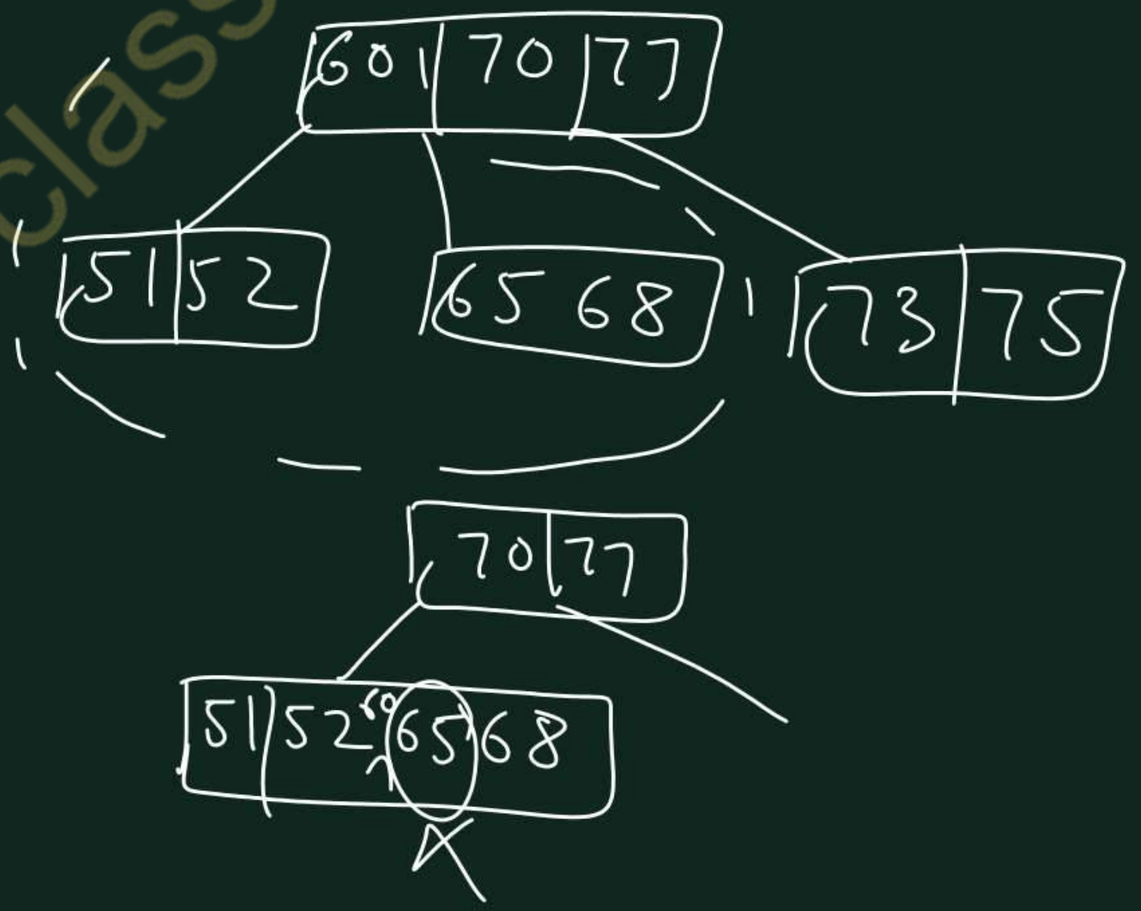
with $t = 3$



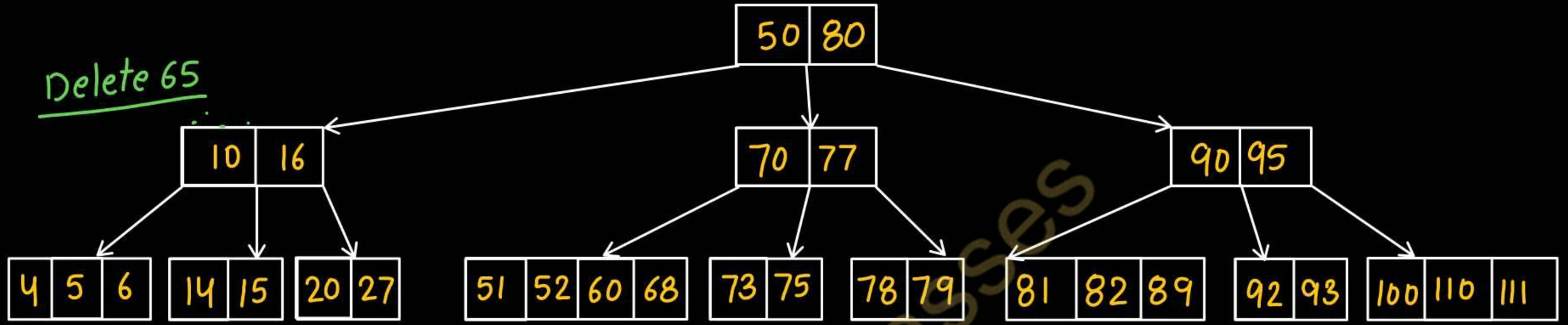
Delete 64, 23, 72



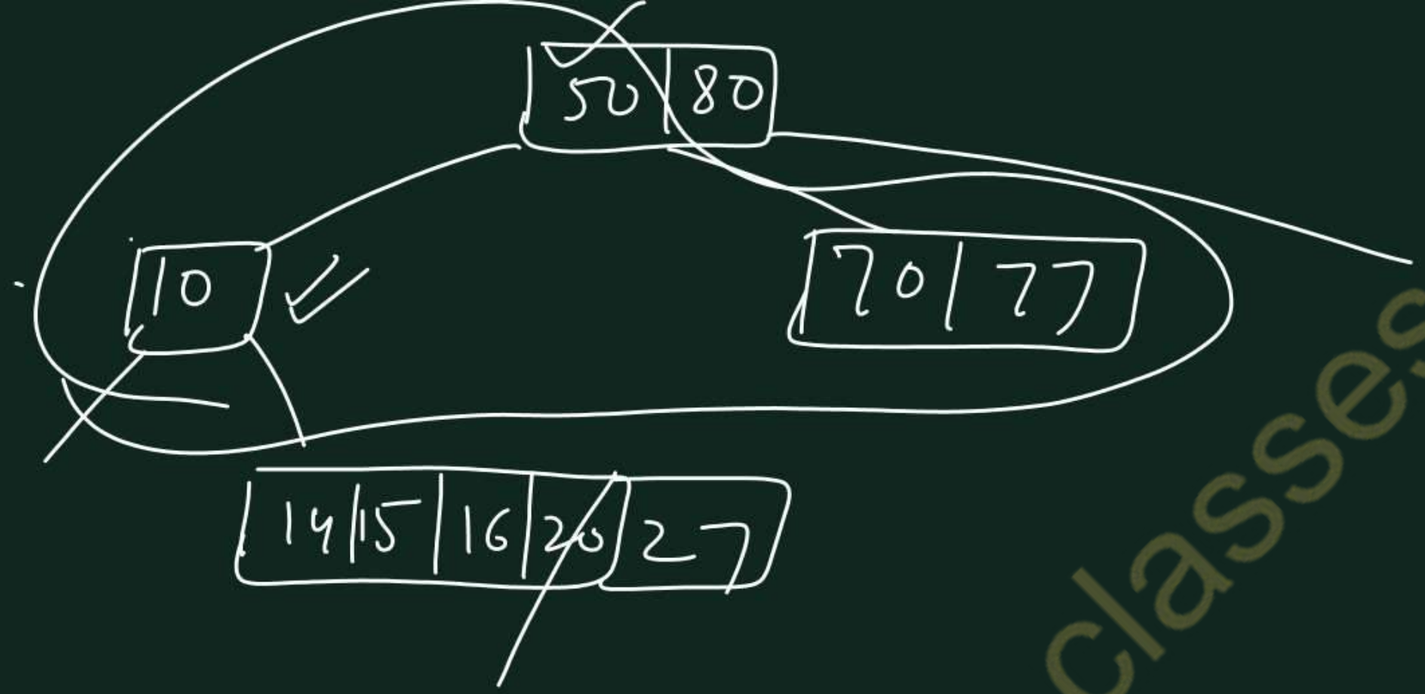




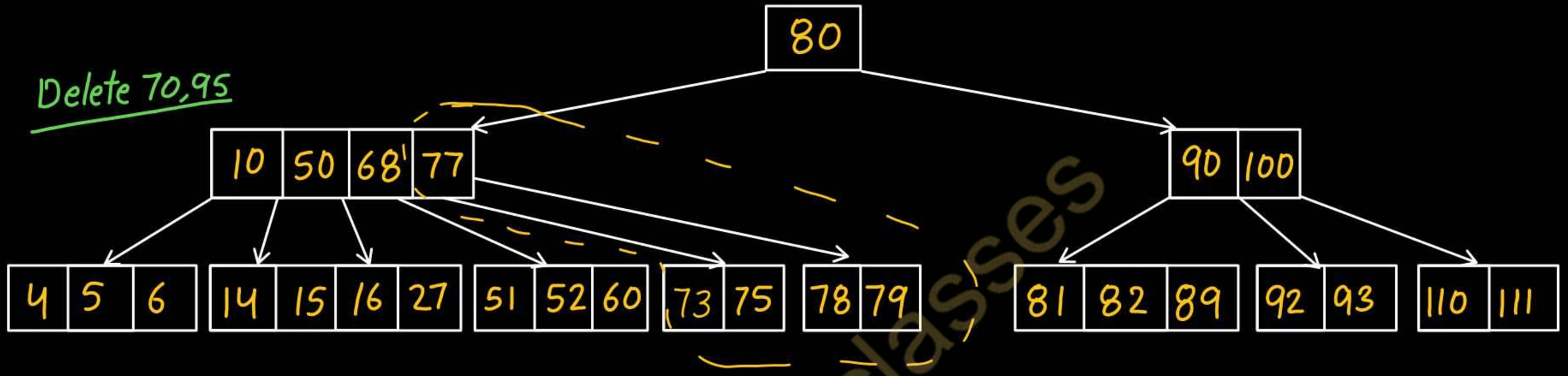
Delete 65



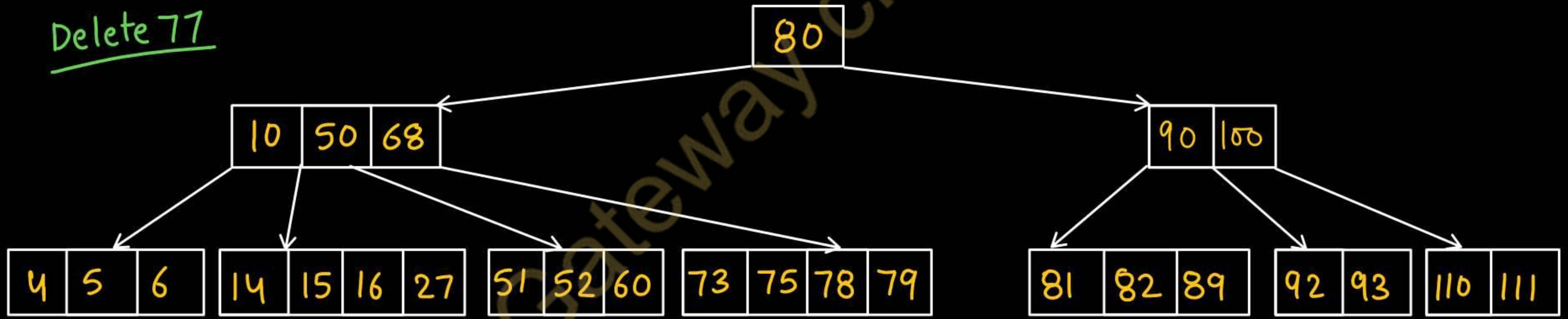
Delete 20



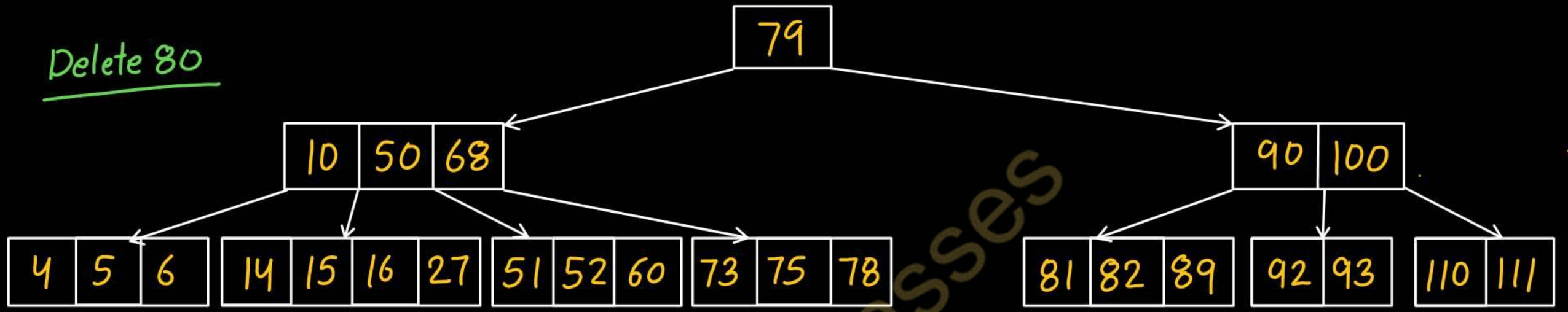
Delete 70,95

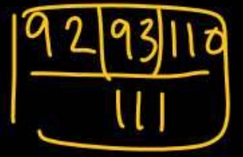


Delete 77



Delete 80





Delete 100

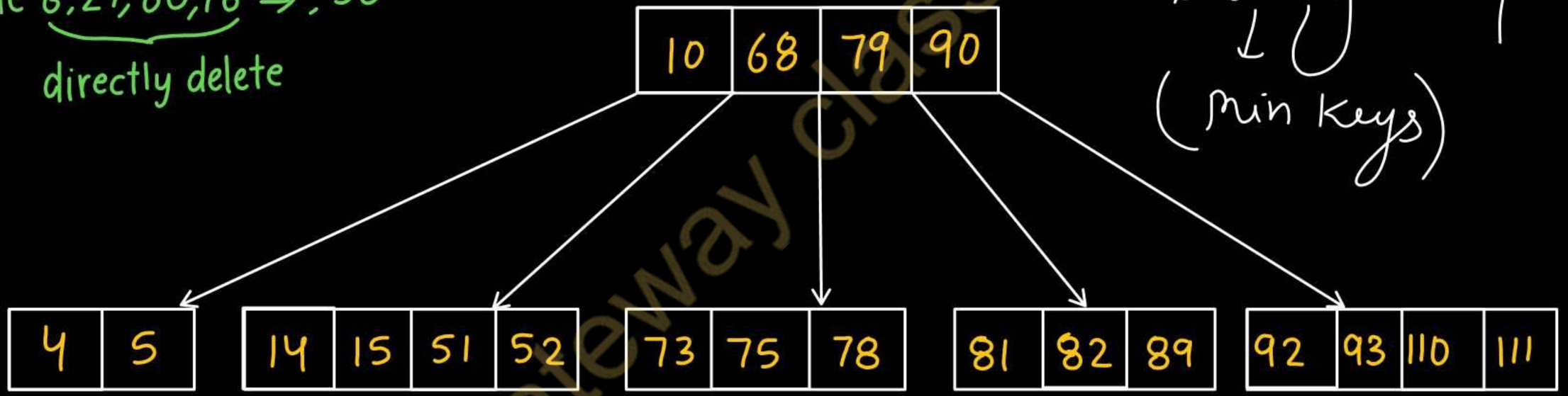
Now delete 6,27,60,15

Delete 6, 27, 60, 16 ⇒, 50

directly delete

it will create underflow condition on internal node [10] so it will be merged with sibling and parent

(Min keys)



AKTU PYQs

1. Using minimum t as 3, insert following sequence of integers:
10,25,20,35,30,55,40,45,50,55,60,75,70,65,80,85,90 in initially empty B-Tree. Give number of node splitting operations that take place. (AKTU-2019-20)

2. Explain B-tree and its properties. Also write B-tree deletion cases with example.(AKTU 2023-24)

3. Show the results of inserting the keys F,S,Q,K,C,L,H,T,V,W,M,R,N,P,A,B,X,Y,D,Z,E in an empty b-tree with t=3. (AKTU 2021-22)

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-6

Today's Target

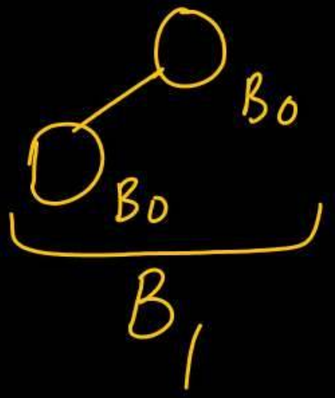
Binomial Heap

- Properties of Binomial Tree
- Properties of Binomial Heap
- Binomial Heap Operations
  - Creating a new binomial heap
  - Finding the minimum key
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

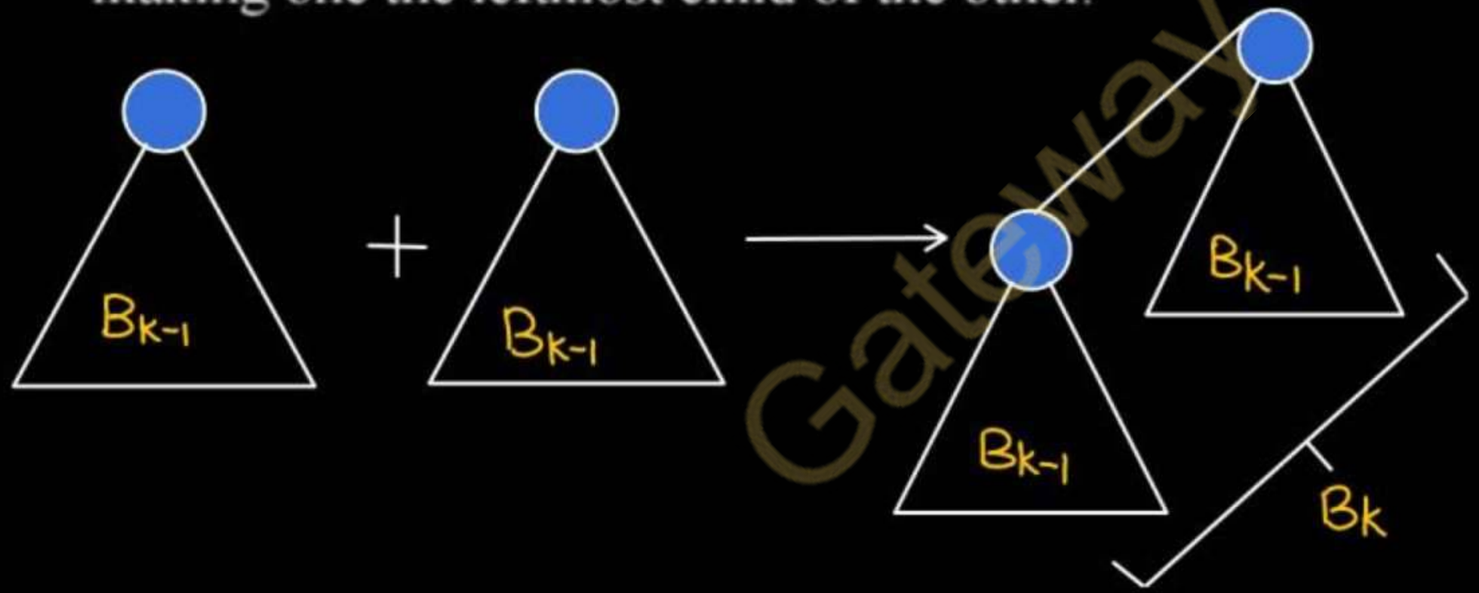
Binomial Heap

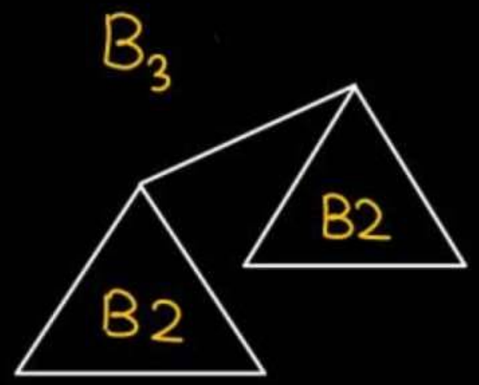
A Binomial Heap is a collection of Binomial Trees.

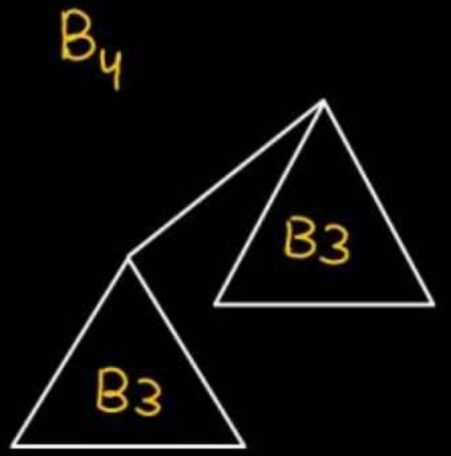


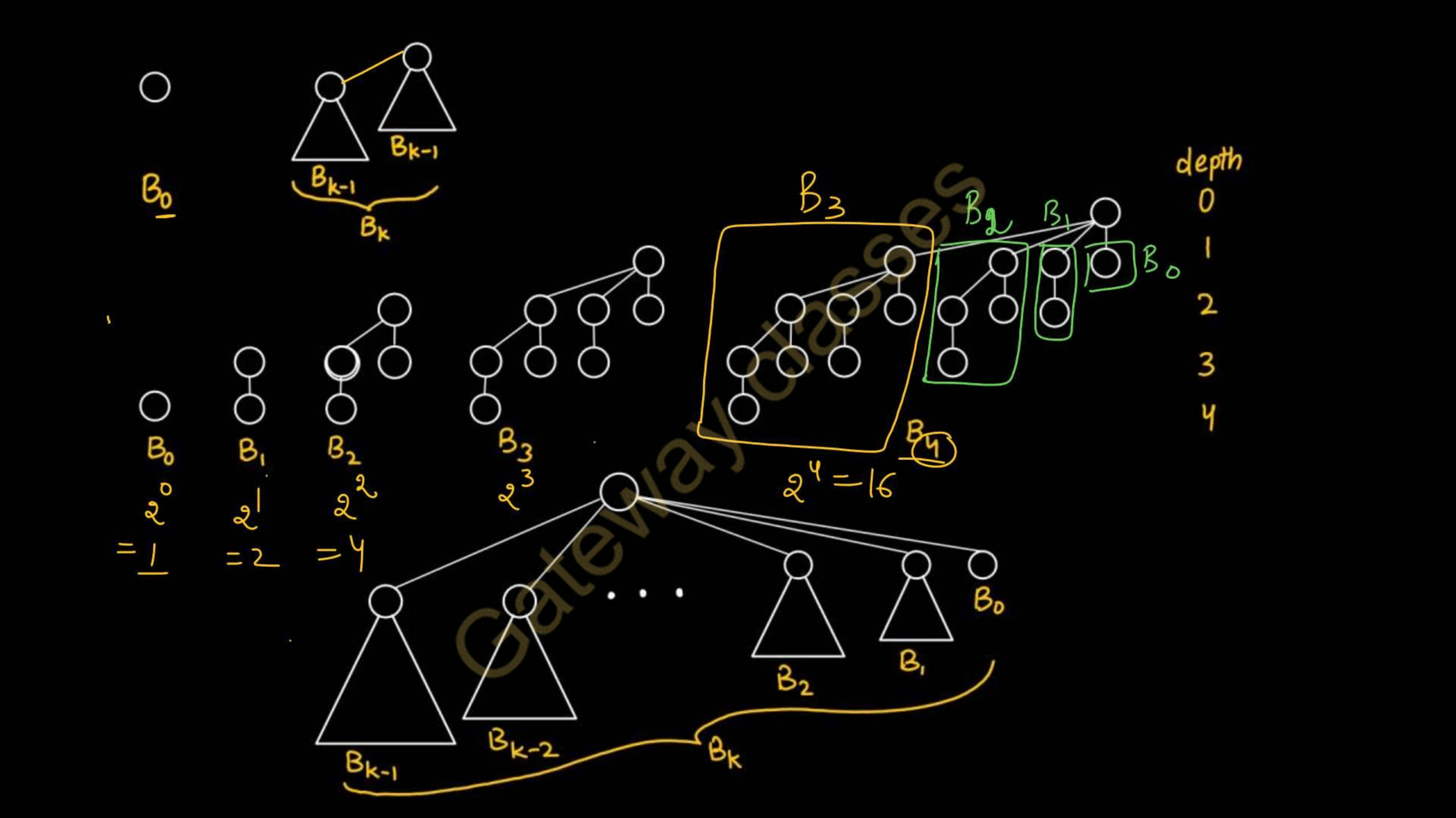
What is a Binomial Tree?

- A Binomial Tree is an ordered tree.
- A Binomial Tree of order 0 has 1 node.
- A Binomial Tree of order k can be constructed by taking two binomial trees of order k-1 and making one the leftmost child of the other.







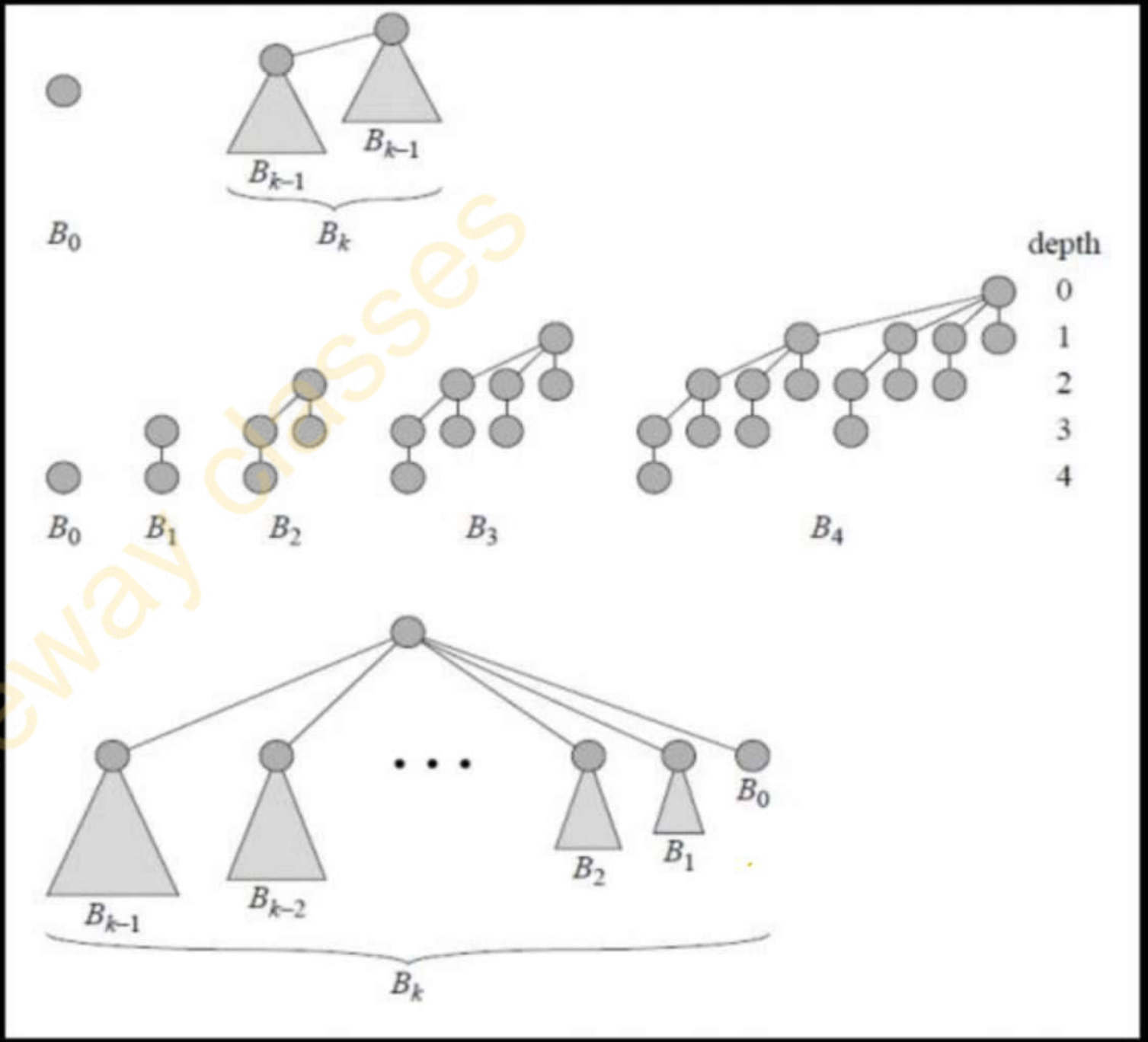

B_0
B_{k-1}
B_{k-1}
B_k
B_0 B_1 B_2 B_3
2^0 2^1 2^2 2^3
$= 1$ $= 2$ $= 4$
B_3
B_2 B_1 B_0
B_4
$2^4 = 16$
depth
0
1
2
3
4
B_{k-1} B_{k-2} B_k
B_2 B_1 B_0
Gatewayclasses

Properties of Binomial Tree: (Imp)

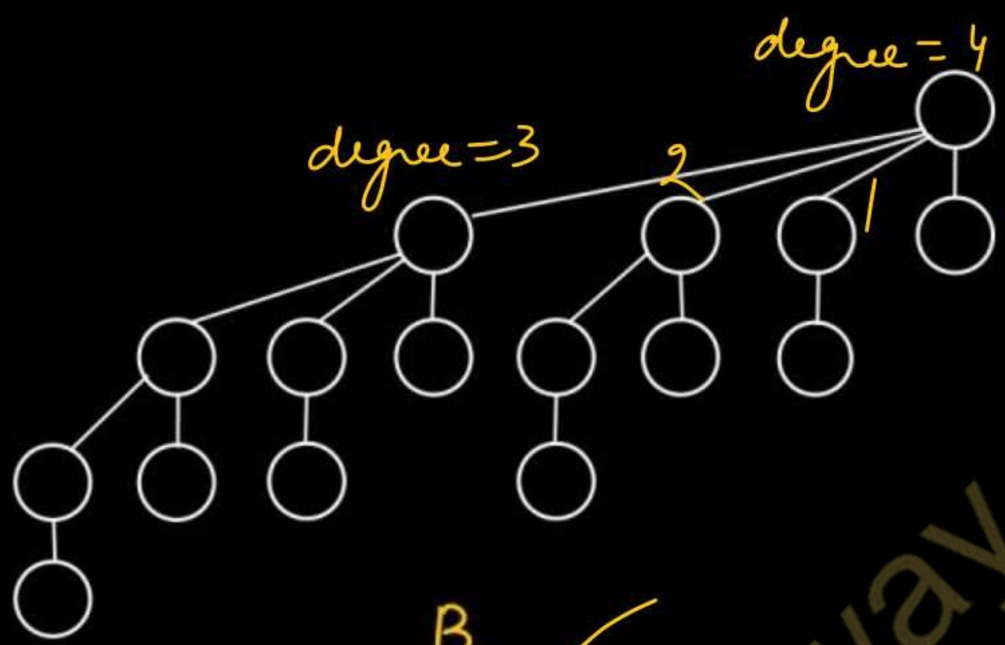
A Binomial Tree of order k the has following properties... K ⇒ order

1. It has exactly 2^k nodes.
2. It has depth as k.
3. There are exactly ${}^kC_i$ nodes at depth i for i = 0, 1, . . . , k.
4. The root has degree k and children of the root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right.

- The maximum degree of any node in an n-node binomial tree is lg n.



K_{C_i} ✓



depth	
0	$4C_0 = 1$
1	$4C_1 = 4$
2	$4C_2 = 6$
3	$4C_3 = 4$
4	$4C_4 = 1$

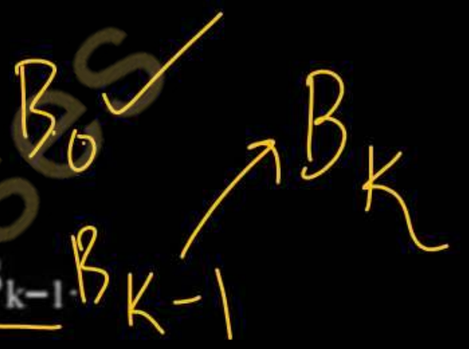
B_4 ✓

$K = 4$

$$K_{C_i} = \frac{\lfloor k \rfloor}{\lfloor i \rfloor \times \lfloor k-i \rfloor}$$

Proof of Binomial Tree properties (AKTU 2023-24)

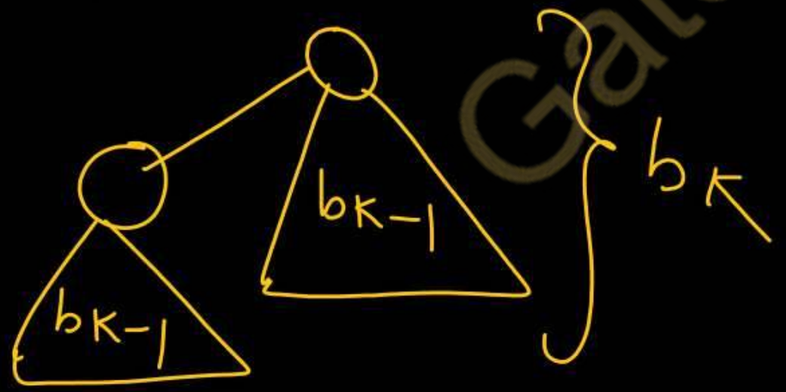
- The proof is by induction on k.
- For each property, the basis is the binomial tree B0.
- Verifying that each property holds for B0 is trivial.
- For the inductive step, we assume that the lemma holds for B_{k-1}.



Proof of PROPERTY 1: B_k has exactly 2^k nodes.

B_0 has 1 nodes.

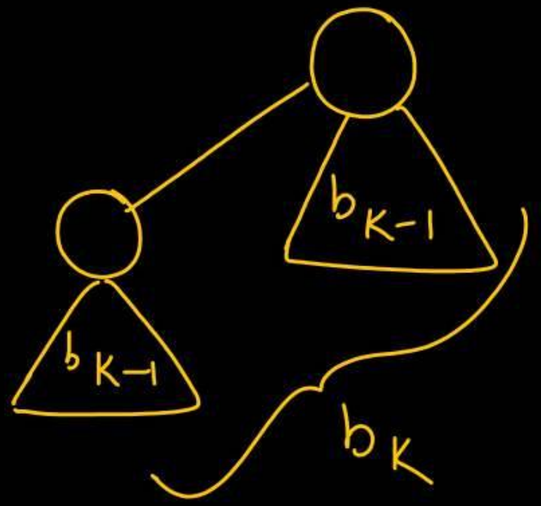
Binomial tree B_k consists of two copies of B_{k-1}, and so B_k has $2^{k-1} + 2^{k-1} = 2^k$ nodes.



Proof of PROPERTY 2: B_k has depth as k

Because of the way in which the two copies of B_{k-1} are linked to form B_k, the maximum depth of a node in B_k is one greater than the maximum depth in B_{k-1}. By the inductive hypothesis, this maximum depth is $(k - 1) + 1 = k$.

$B_0 = 0$



depth of $b_{K-1} = K-1$

$b_K = K$

Proof of PROPERTY 3: There are exactly ${}^{k}C_{i}$ nodes at depth i for i = 0, 1, . . . , k.

Let D(k, i) be the number of nodes at depth i of binomial tree B_k.

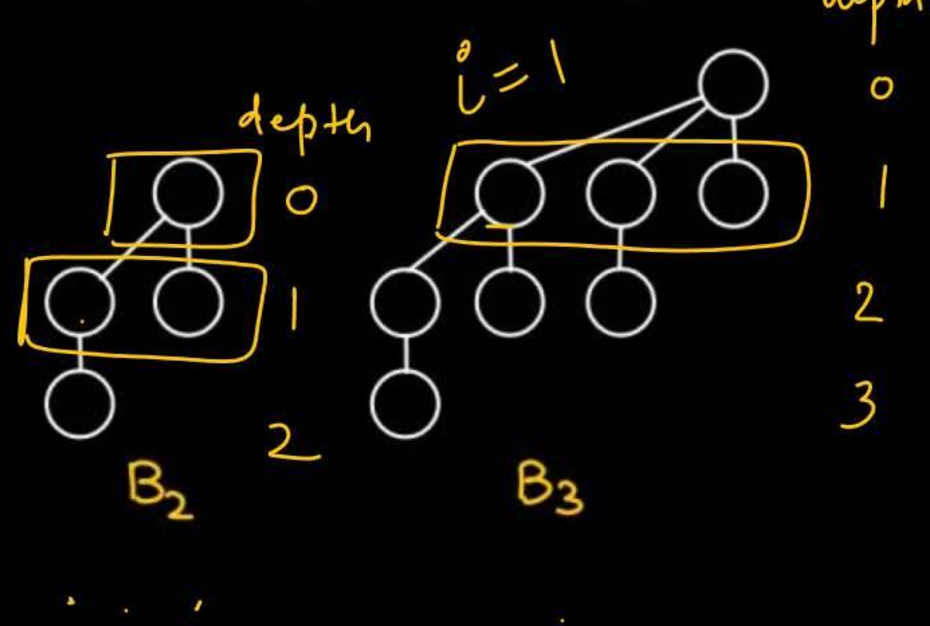
Since B_k is composed of two copies of B_{k-1} linked together, the number of nodes at depth i in B_k is the number of nodes at depth i in B_{k-1} plus the number of nodes at depth i − 1 in B_{k-1}.

Thus

$$D(k,i) = D(k-1,i) + D(k-1,i-1)$$

$$= {}^{k-1}C_{i} + {}^{k-1}C_{i-1} \text{ (by the inductive hypothesis)}$$

$$= {}^{k}C_{i}$$



Proof of PROPERTY 4: The root has degree k and children of the root are themselves Binomial Trees with order k-1, k-2,.. 0 from left to right.

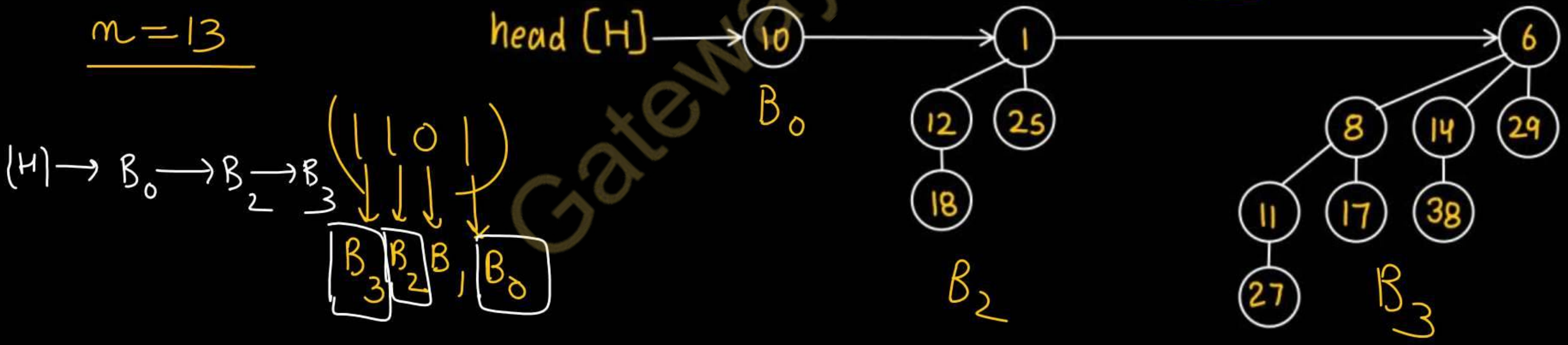
The only node with greater degree in B_k than in B_{k-1} is the root, which has one more child than in B_{k-1}. Since the root of B_{k-1} has degree $k - 1$, the root of B_k has degree k.

From left to right, the children of the root of B_{k-1} are roots of $B_{k-2}, B_{k-3}, \ldots, B_0$. When B_{k-1} is linked to B_{k-1}, therefore, the children of the resulting root are roots of $B_{k-1}, B_{k-2}, \ldots, B_0$.

What is a Binomial Heap?

A binomial heap H is a set of binomial trees that satisfies the following binomial heap properties.

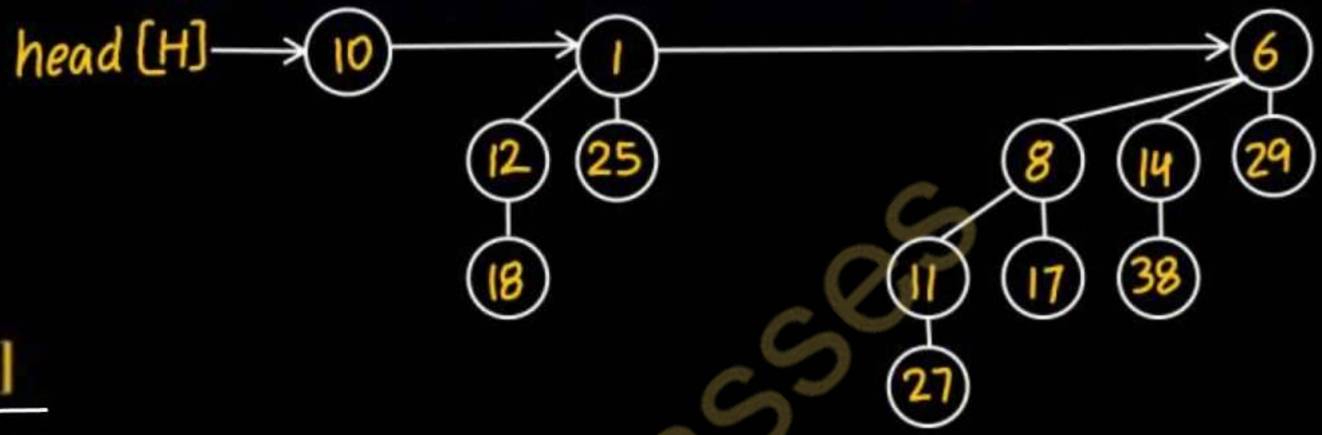
- Each binomial tree in H obeys the min-heap property: the key of a node is greater than or equal to the key of its parent.
- For any nonnegative integer k, there is at most one binomial tree in H whose root has degree k. It implies that an n-node binomial heap H consists of at most [lg n] + 1 binomial trees.

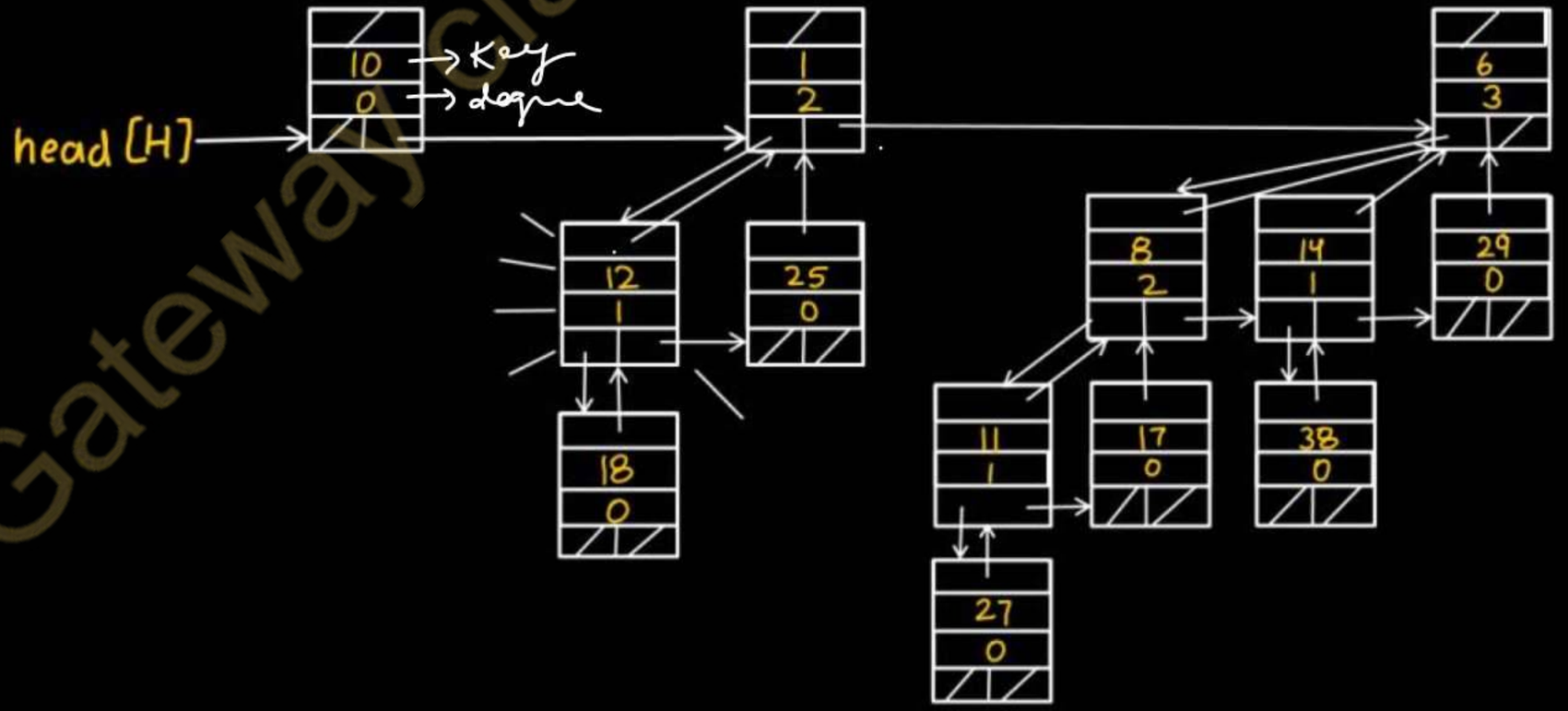


Representation of binomial heap

Each node contains ~~has~~ the following information…

1. Key field
2. pointer to its parent: p[x]
3. Pointer to its leftmost child: child[x]
4. Pointer to the sibling of x immediately to its right: sibling[x]
5. Number of children of *x* :*degree*[*x*]





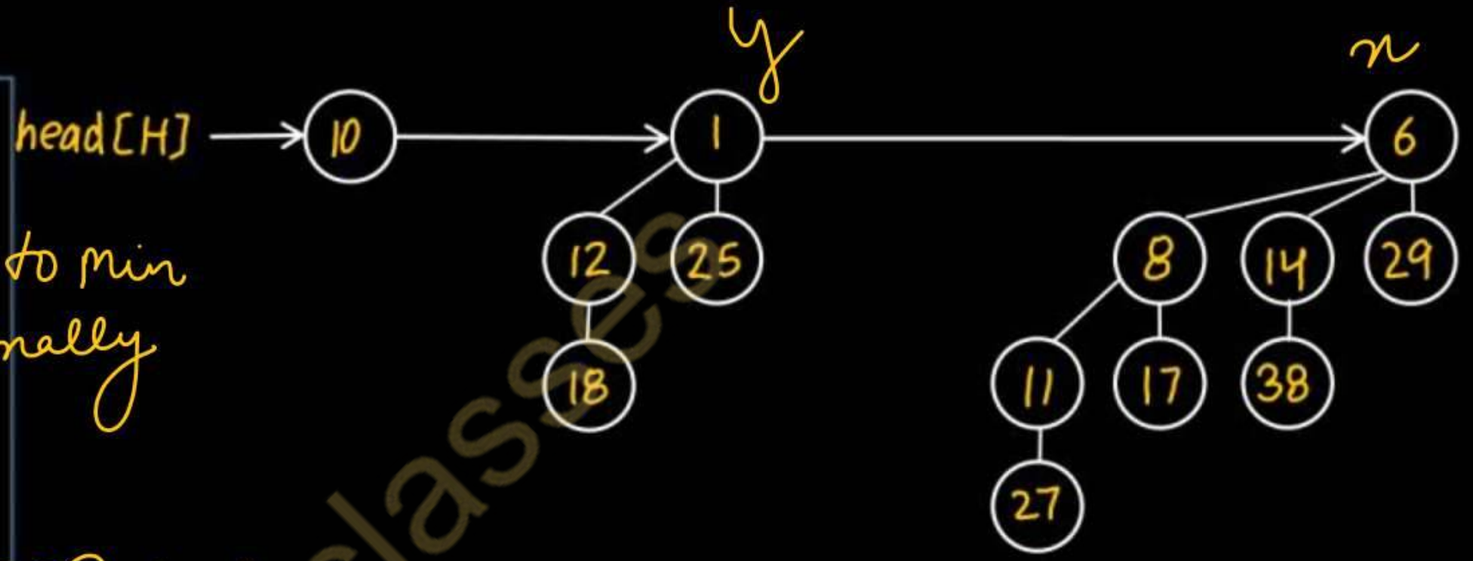
Binomial Heap operations

- Creating a new binomial heap
- Finding the minimum key
- Union of two binomial heaps (Most Important)
- Inserting a node into binomial heap
- Extracting the node with minimum key (Imp)
- Decreasing a key (Imp)
- Deleting a key

1. Finding the minimum key

```
BINOMIAL-HEAP-MINIMUM(H)
1  y ← NIL
2  x ← head[H]
3  min ← ∞
4  while x ≠ NIL
5        do if key[x] < min
6              then min ← key[x]
7                   y ← x
8        x ← sibling[x]
9  return y
```

y will point to min element finally



$min = \infty$

(I) $10 < \infty$ (T) $\Rightarrow min = 10$

(II) $1 < 10$ (T) $\Rightarrow min = 1$

(III) $6 < 1$ (F) $\Rightarrow min = 1$

At most $[\lg n] + 1$ roots to check, So running time: $O(\lg n)$.

2. Creating a new binomial heap

MAKE-BINOMIAL-HEAP
allocate and return an object H
where $head[H] = NIL$.
End.

The running time: $\Theta(1)$.

Head[H] → NIL

AKTU PYQs

1. Prove all four properties of binomial tree.(AKTU 2023-24)
2. Discuss properties of binomial tree.(AKTU 2022-23)(AKTU 2020-21)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-7

Today's Target

- Union operation on two Binomial Heaps
- Inserting a node in Binomial Heap
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

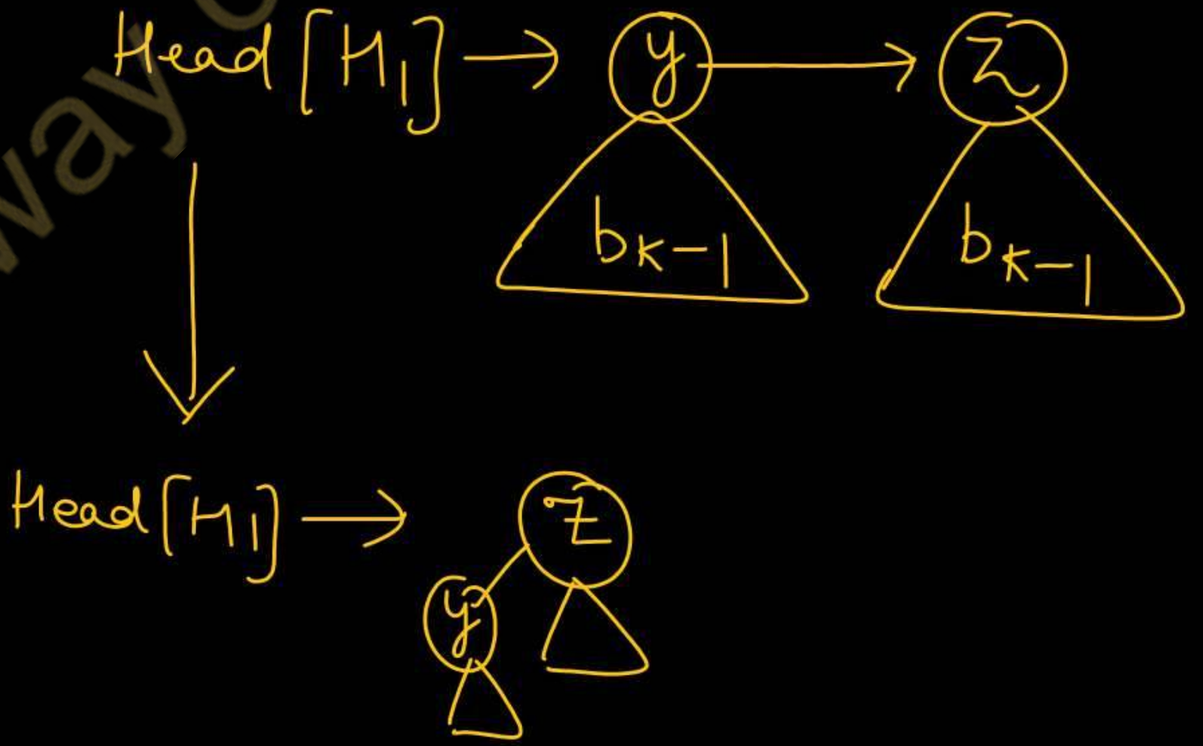
Union of two binomial heaps

The BINOMIAL-HEAP-UNION procedure repeatedly links binomial trees whose roots have the same degree.

The following procedure links the B_{k-1} tree rooted at node y to the B_{k-1} tree rooted at node z; that is, it makes z the parent of y. Node z thus becomes the root of a B_k tree.

```
BINOMIAL-LINK(y, z)
1  p[y] ← z
2  sibling[y] ← child[z]
3  child[z] ← y
4  degree[z] ← degree[z] + 1
```

Running Time: O(1)



BINOMIAL-HEAP-UNION($H1$, $H2$)

1 $H \leftarrow$ MAKE-BINOMIAL-HEAP()

2 $head[H] \leftarrow$ BINOMIAL-HEAP-MERGE($H1$, $H2$)

3 free objects $H1$ and $H2$ but not the lists they point to

4 **if** $head[H]$ = NIL

5 **then return** H

6 $prev\text{-}x \leftarrow$ NIL

7 $x \leftarrow head[H]$

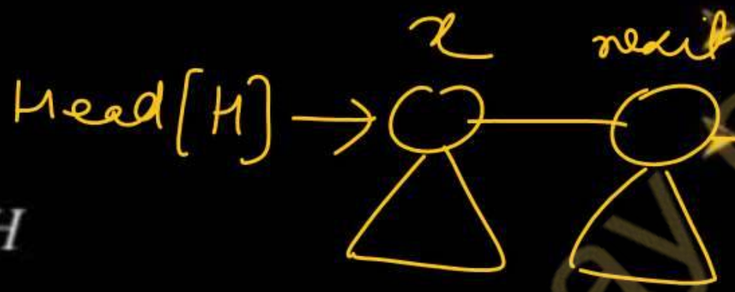
8 $next\text{-}x \leftarrow sibling[x]$

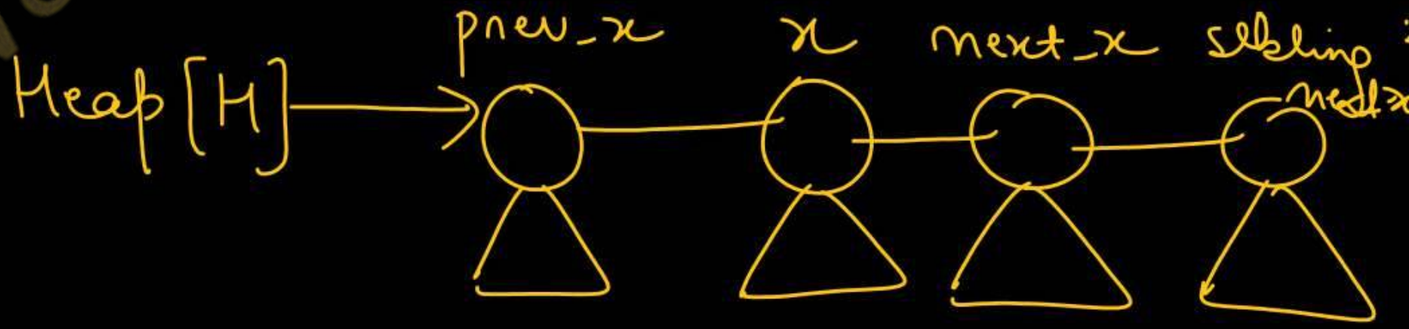
- x points to the root currently being examined
- $prev\text{-}x$ points to root preceding x on the root list:

  $sibling[prev\text{-}x] = x$

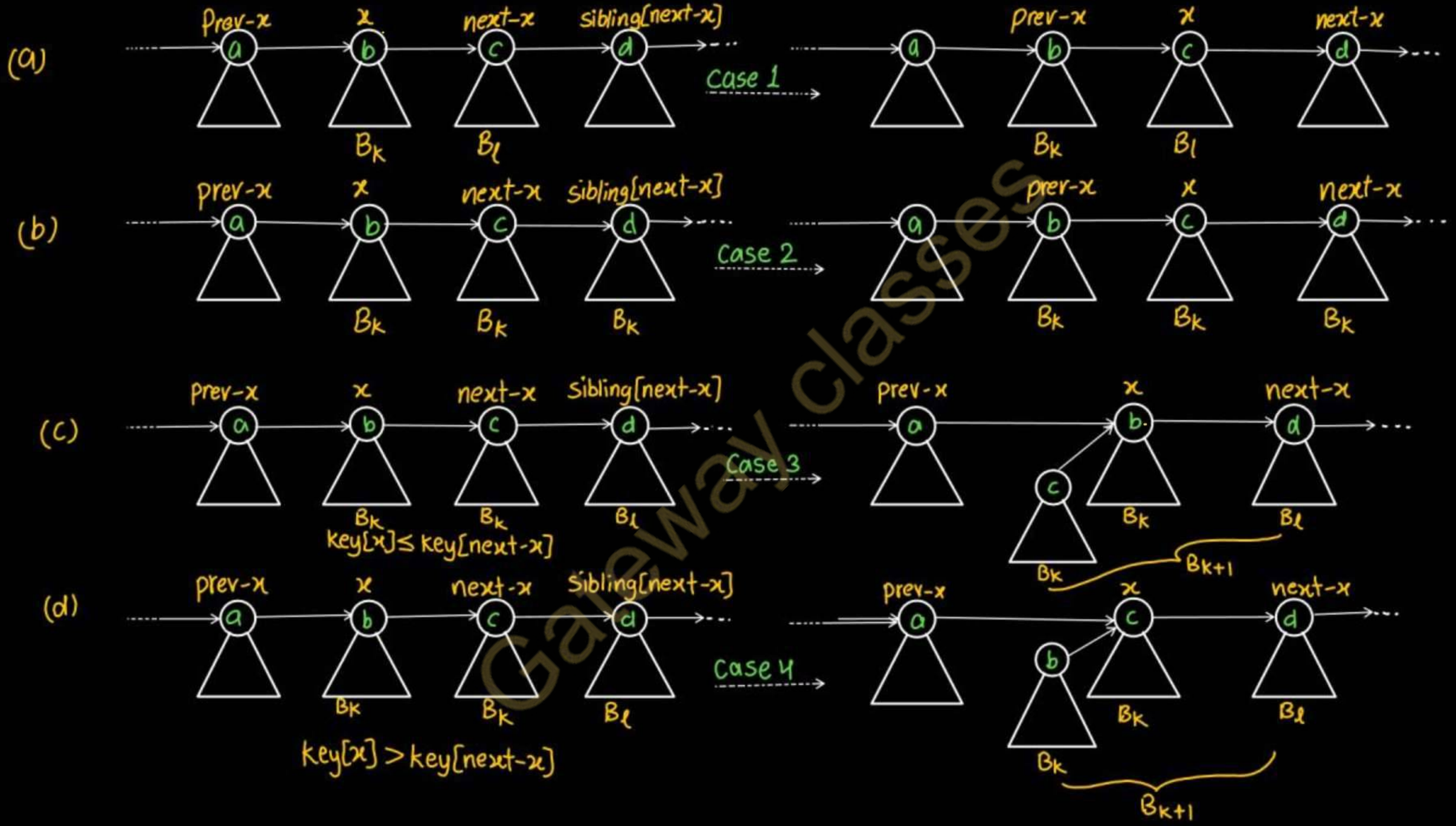
  initially x has no predecessor, so $prev\text{-}x$ set to NIL
- $next\text{-}x$ points to root following x on the root list:

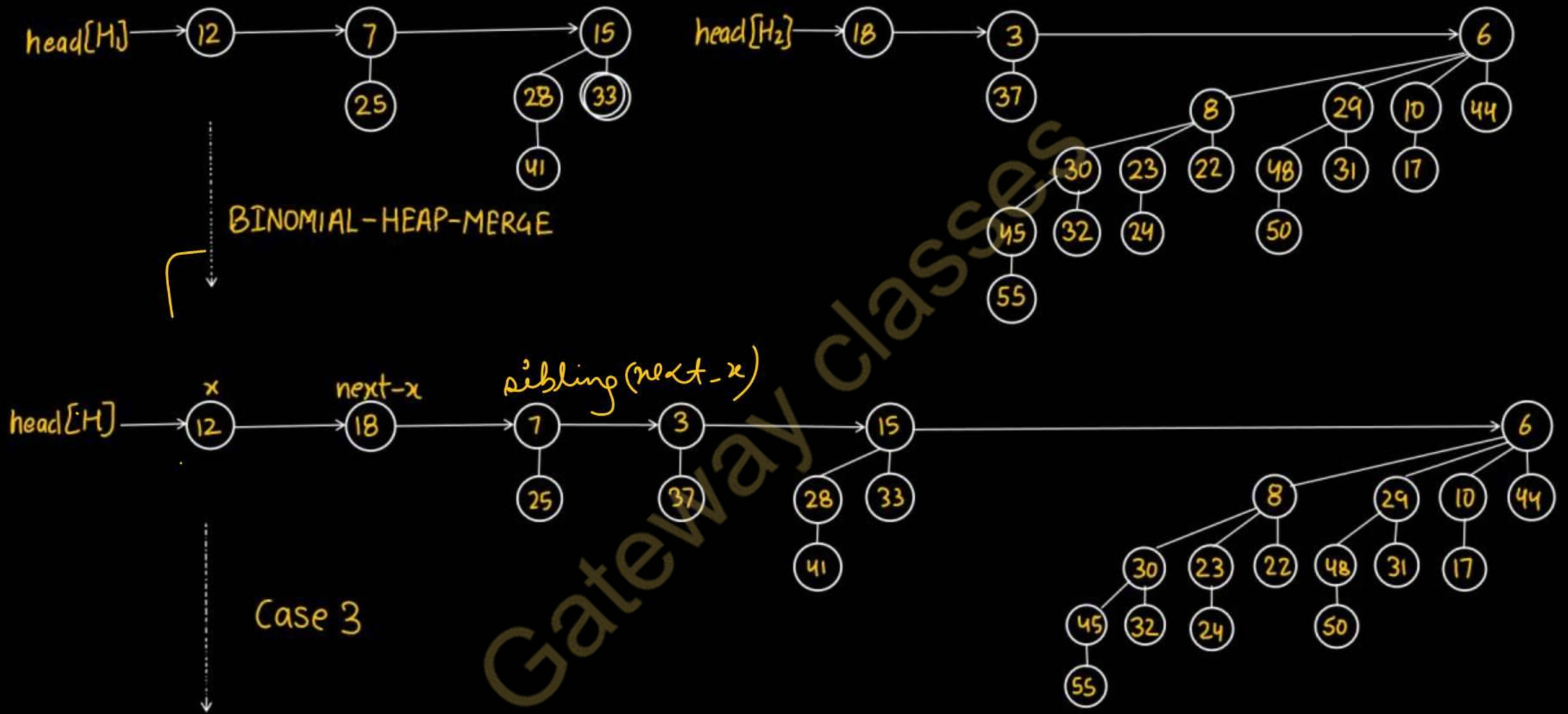
  $sibling[x] = next\text{-}x$

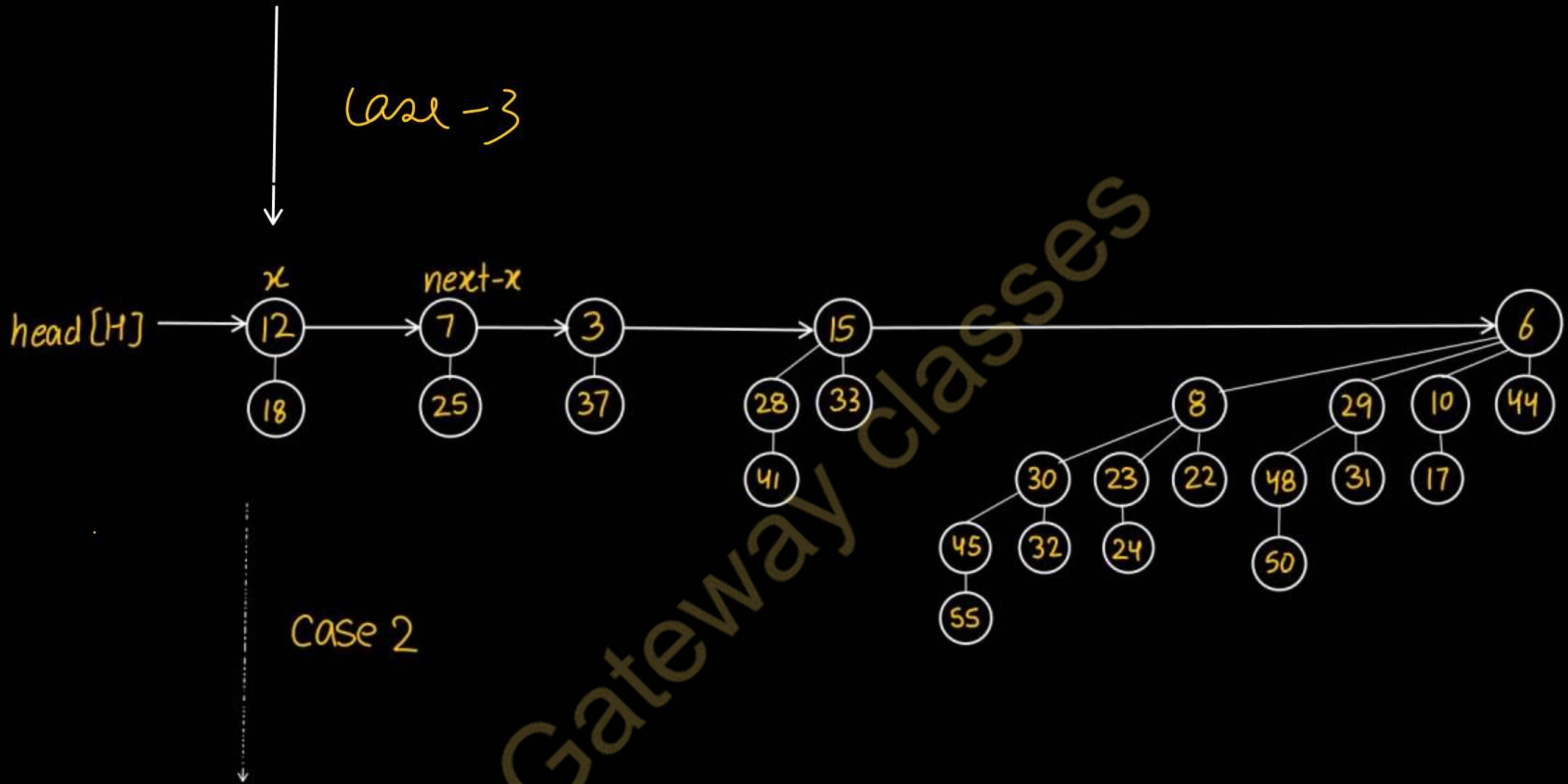


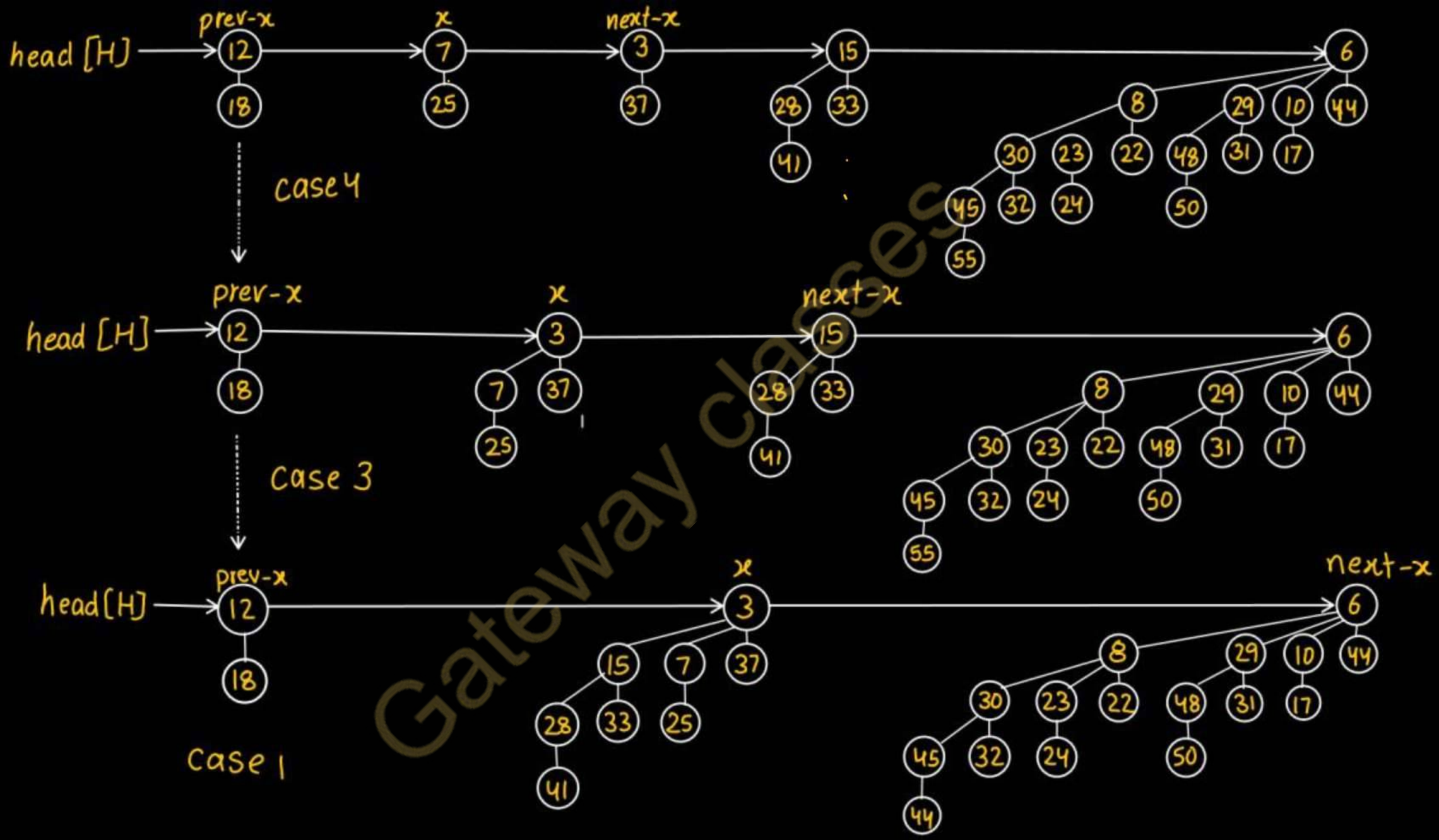


```
9  while next-x ≠ NIL
10      do if (degree[x] ≠ degree[next-x]) or (sibling[next-x] ≠ NIL and degree[sibling[next-x]] = degree[x])
11            then prev-x ← x                                  Cases 1 and 2
12                 x ← next-x                                  Cases 1 and 2
13            else if key[x] ≤ key[next-x]
14                      then sibling[x] ← sibling[next-x]      Case 3
15                           BINOMIAL-LINK(next-x, x)          Case 3
16            else if prev-x = NIL                             Case 4
17                      then head[H] ← next-x                  Case 4
18            else sibling[prev-x]← next-x                     Case 4
19            BINOMIAL-LINK(x, next-x)                         Case 4
20            x ← next-x                                       Case 4
21      next-x ← sibling[x]
22 return H
```


(a)
Prev-x
x
next-x
sibling[next-x]
a
b
c
d
B_k
B_l
Case 1
(b)
prev-x
x
next-x
sibling[next-x]
Case 2
(c)
Case 3
key[x] ≤ key[next-x]
B_{k+1}
(d)
Case 4
key[x] > key[next-x]
Gateway classes







Time complexity of BINOMIAL-HEAP-UNION

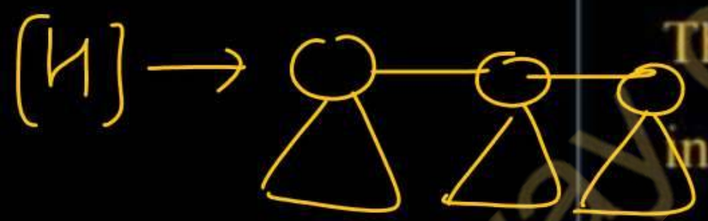
- H1 contain n1 nodes and H2 contain n2 nodes, so that n = n1+n2.
- H1 contains at most ⌊lg n1⌋+1 roots and H2 contains at most ⌊lg n2⌋+1 roots, and so H contains atmost ⌊lg n1⌋+⌊lg n2⌋+2 ≤ 2 ⌊lg n⌋+2 = O(lg n) roots immediately after the call of BINOMIAL-HEAP-MERGE. So, BINOMIAL-HEAP-MERGE running time: O(lg n).
- Each iteration of the **while** loop takes O(1) time, and there are atmost ⌊lg n1⌋ + ⌊lg n2⌋ + 2 iterations because each iteration either advances the pointers one position down the root list of *H* or removes a root from the root list.

"The total running time of BINOMIAL-HEAP-UNION : $O(\lg n)$"

Inserting a node in Binomial Heap

The following procedure inserts node x into binomial heap H, assuming that x has already been allocated and $key[x]$ has already been filled in.

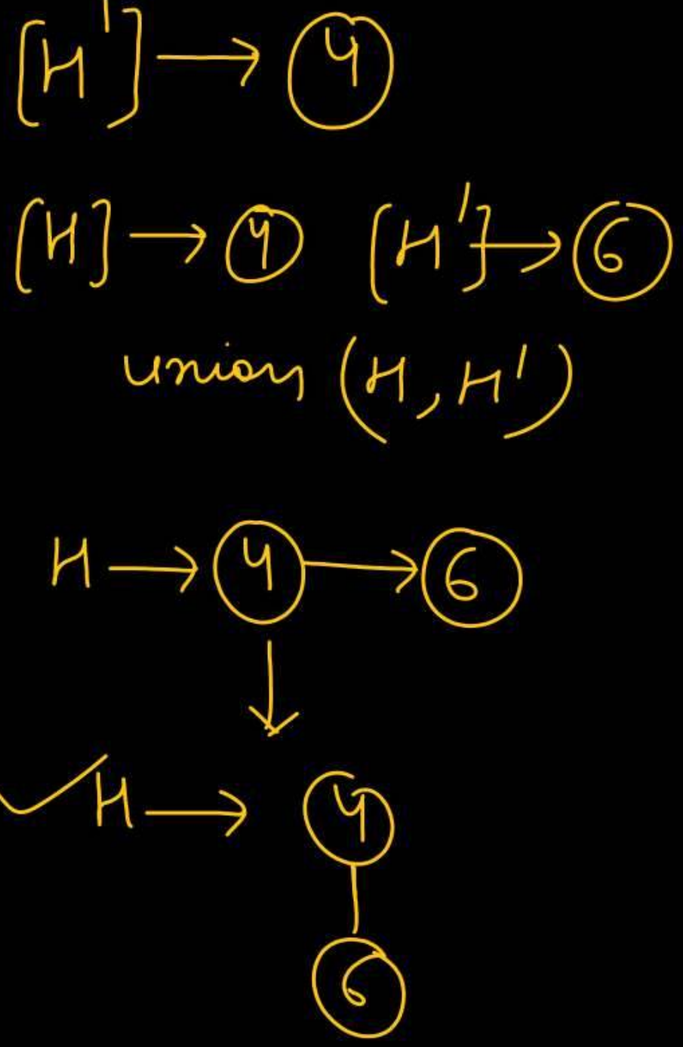
```
BINOMIAL-HEAP-INSERT(H, x)
1 H' ← MAKE-BINOMIAL-HEAP()
2 p[x] ← NIL
3 child[x] ← NIL
4 sibling[x] ← NIL
5 degree[x] ← 0
6 head[H'] ← x
7 H ← BINOMIAL-HEAP-UNION(H, H')
```

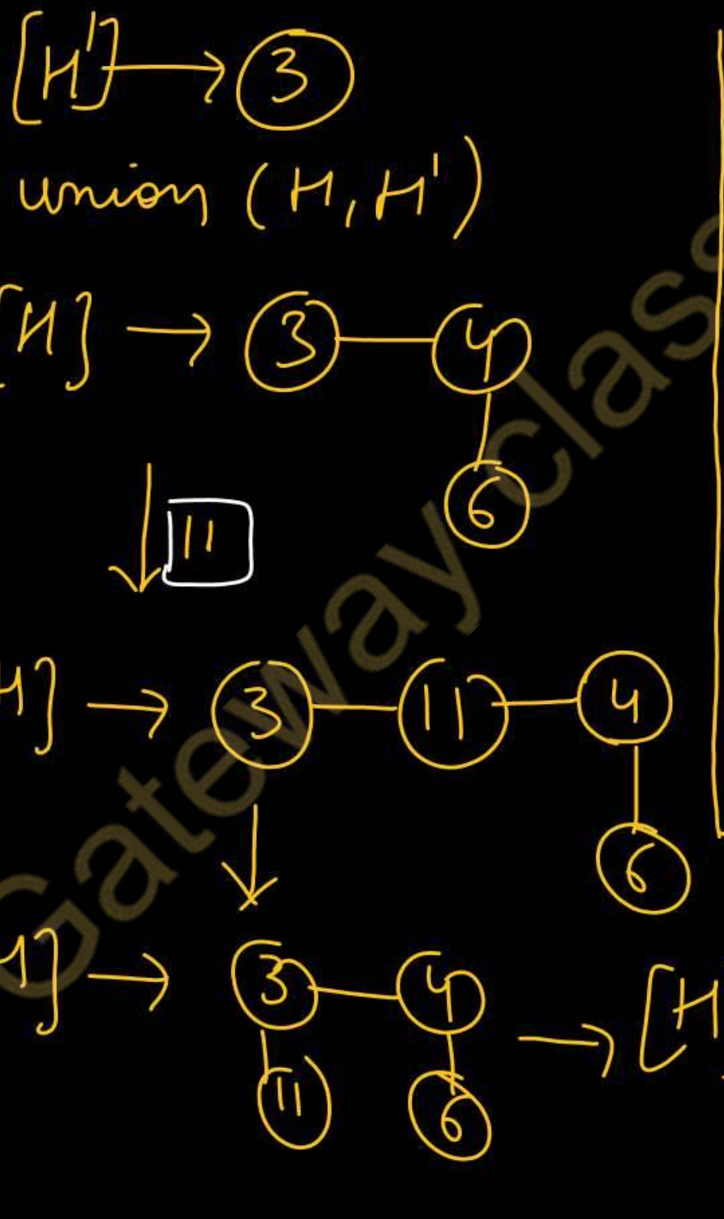


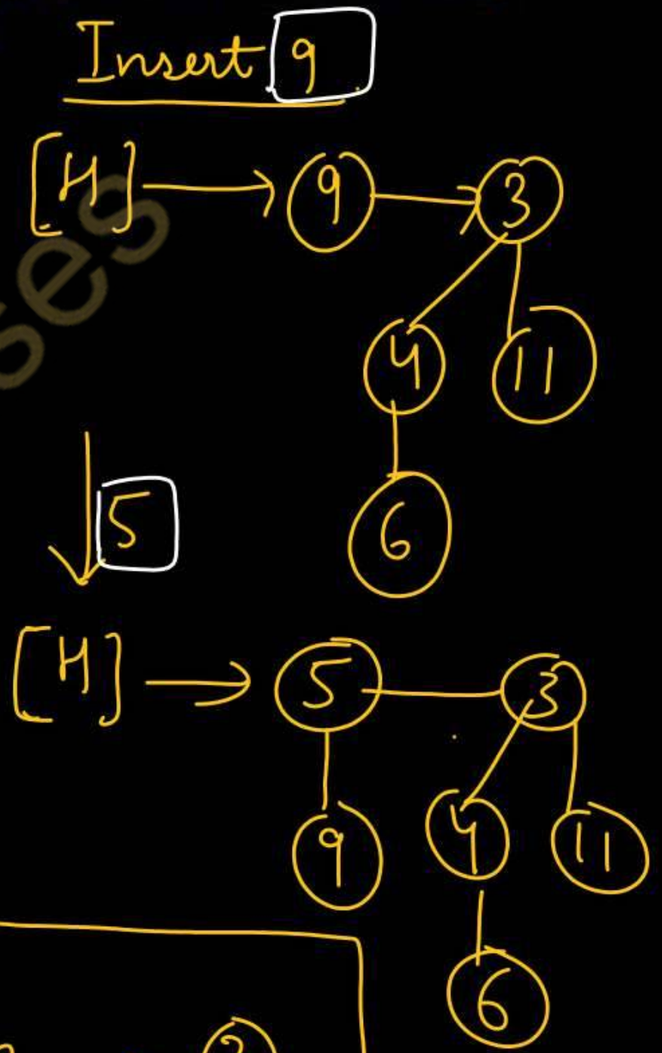
The procedure makes a one-node binomial heap H' in $O(1)$ time and unites it with the n-node binomial heap H in $O(\lg n)$ time.

Question: Insert following elements in initially empty Binomial Heap

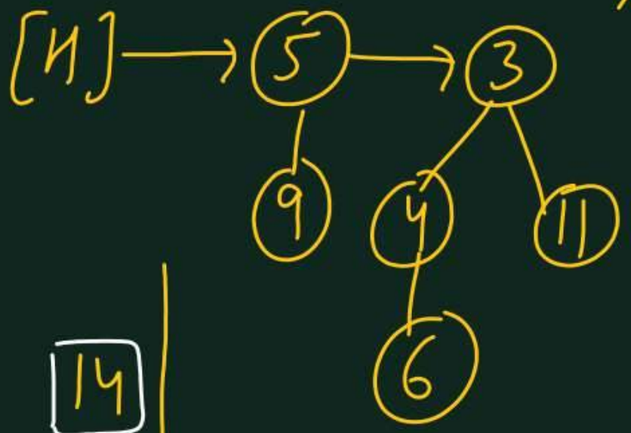
4,6,3,11,9,5,14,10,21,7,13,20,2



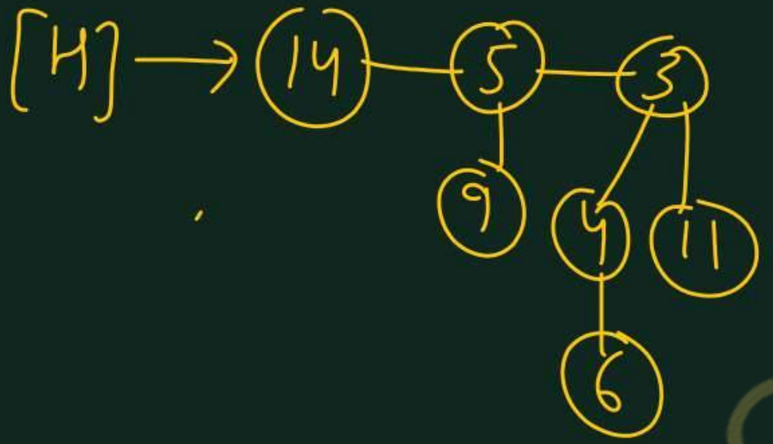




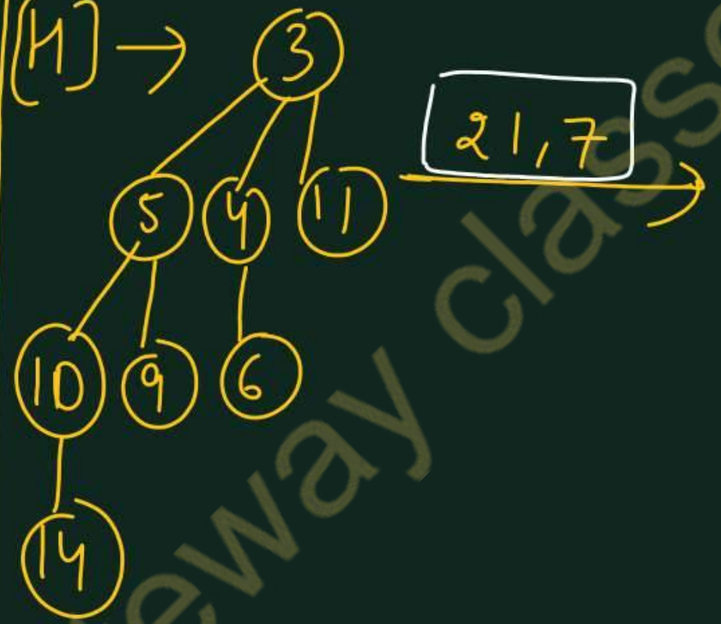
14, 10, 12, 17, 13, 20, 2



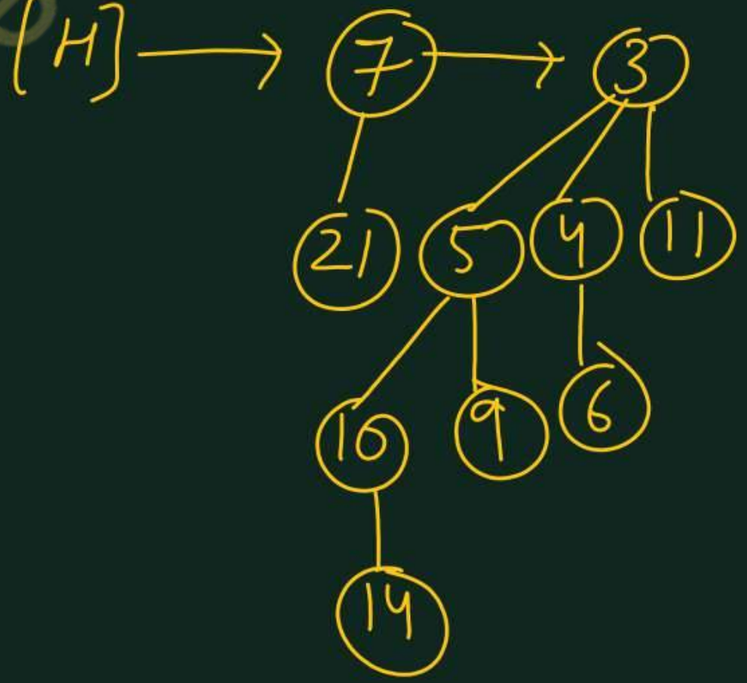
14



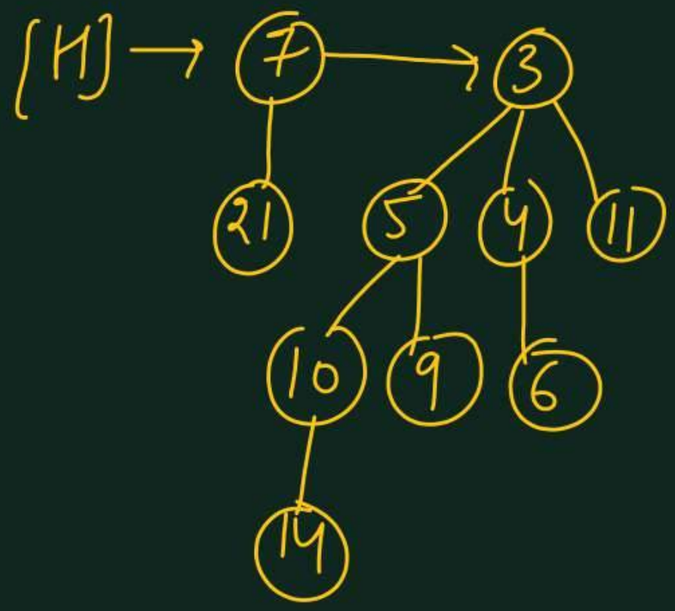
10

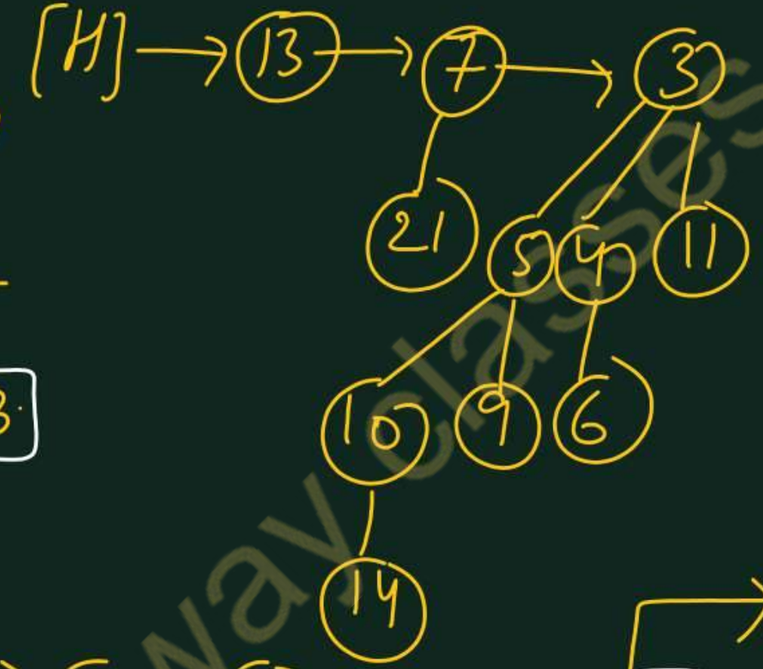


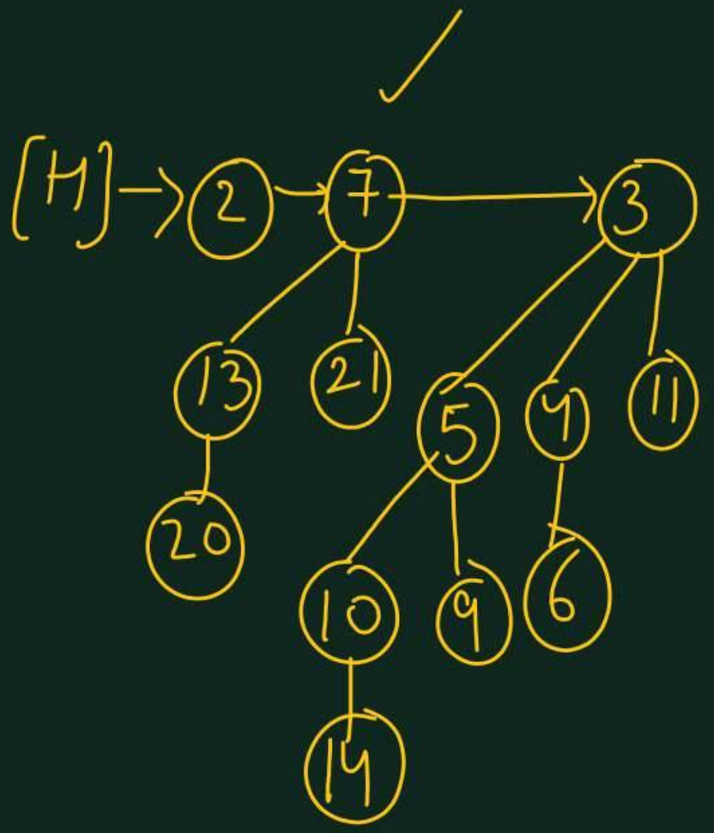
21, 7



13, 20, 21







AKTU PYQs

1. Explain and write an algorithm for union of two binomial heaps and write its time complexity.

(AKTU 2022-23)(AKTU 2020-21)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-8

Today's Target

Binomial Heap Operations

- Extracting the node with minimum key
- Decreasing a key
- Deleting a key
- AKTY PYQs

By Dr. Nidhi Parashar Ma'am

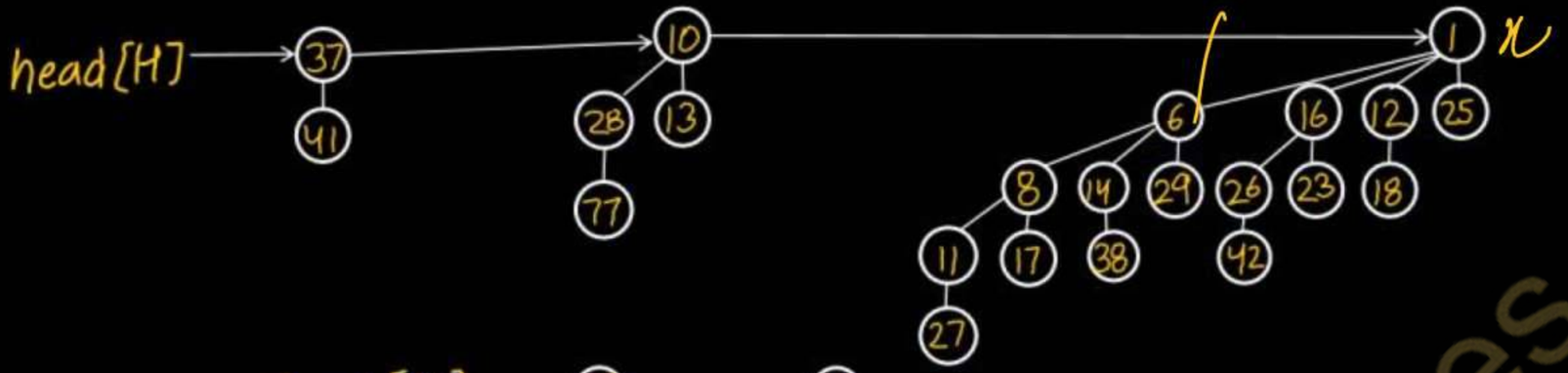
- M.Tech Gold Medalist
- Net Qualified

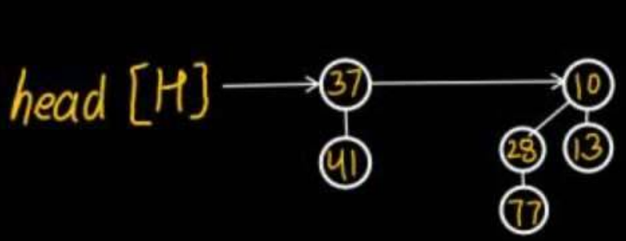
Extracting the node with minimum key

The following procedure extracts the node with the minimum key from binomial heap H and returns a pointer to the extracted node.

```
BINOMIAL-HEAP-EXTRACT-MIN(H)
1  find the root x with the minimum key in the root list of H, and remove x from the root list of H
2  H' ← MAKE-BINOMIAL-HEAP()
3  reverse the order of the linked list of x's children, setting the p field of each child to NIL, and
   set head[H'] to point to the head of the resulting list
4  H ← BINOMIAL-HEAP-UNION(H, H')
5  return x
```

Since each of lines 1–4 takes $O(\lg n)$ time if H has n nodes, BINOMIAL-HEAPEXTRACT-MIN runs in $O(\lg n)$ time.



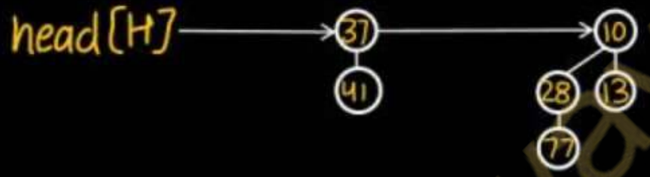


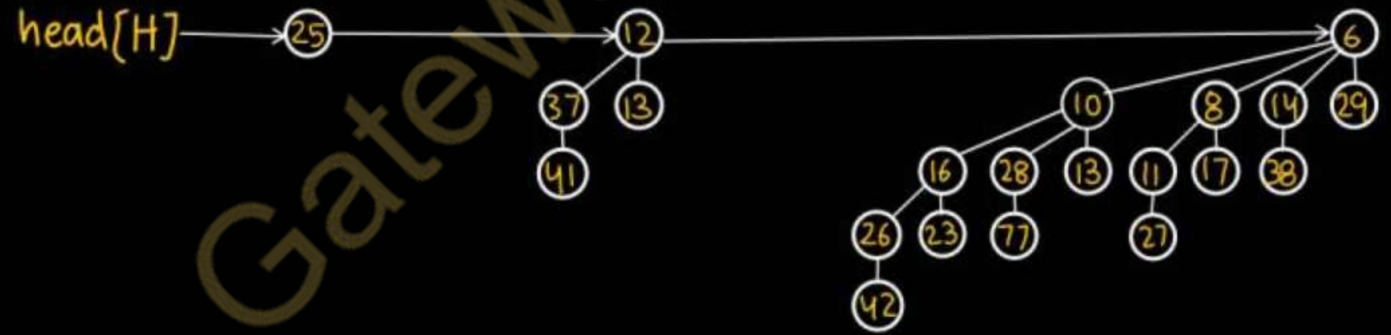
$min = \infty$

$min = 37$

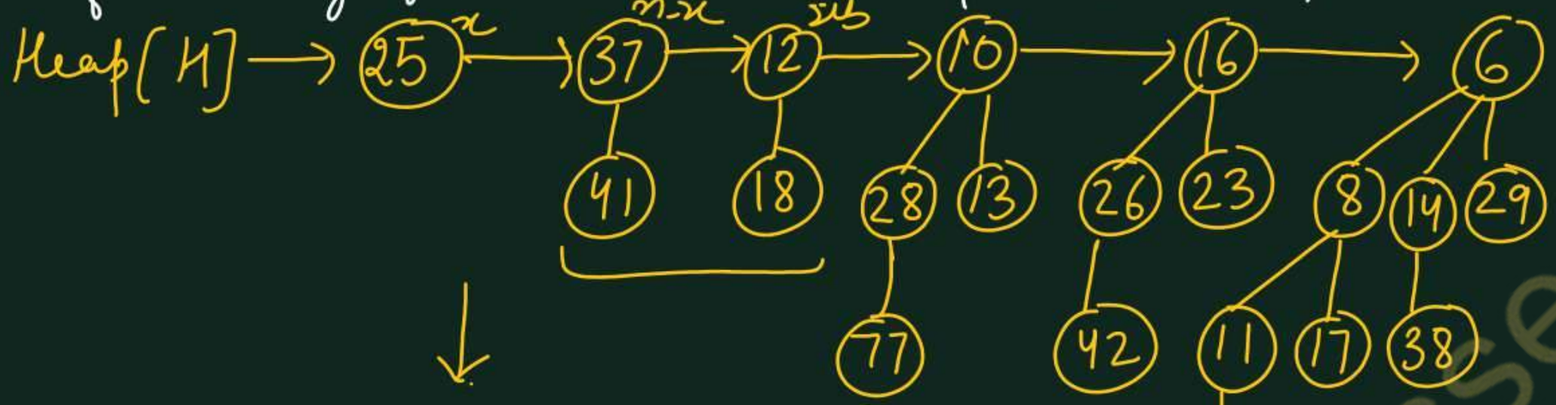
~~16~~

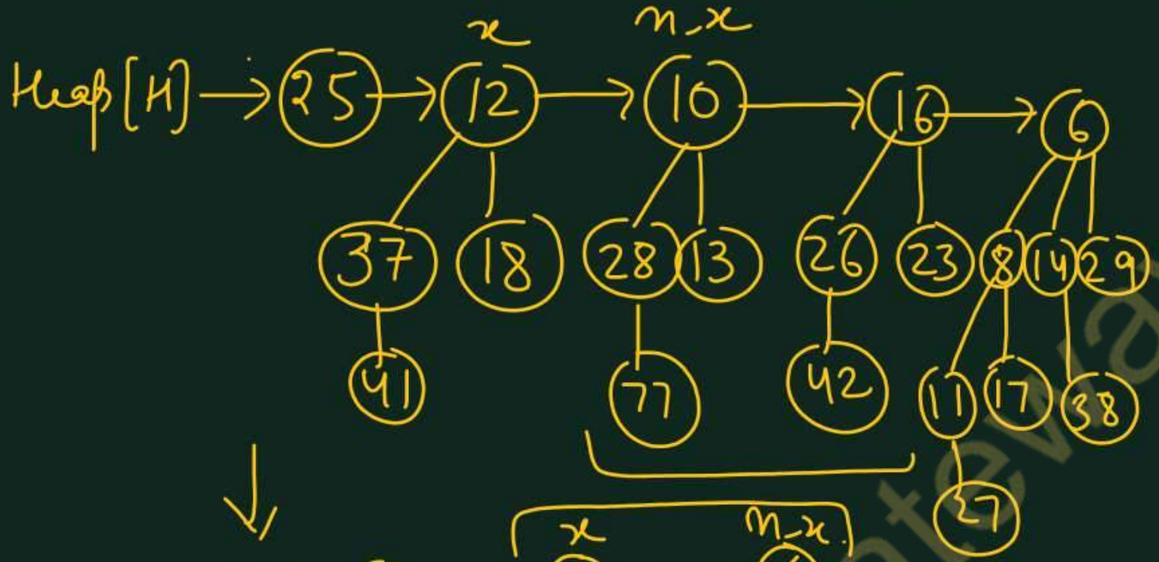
1

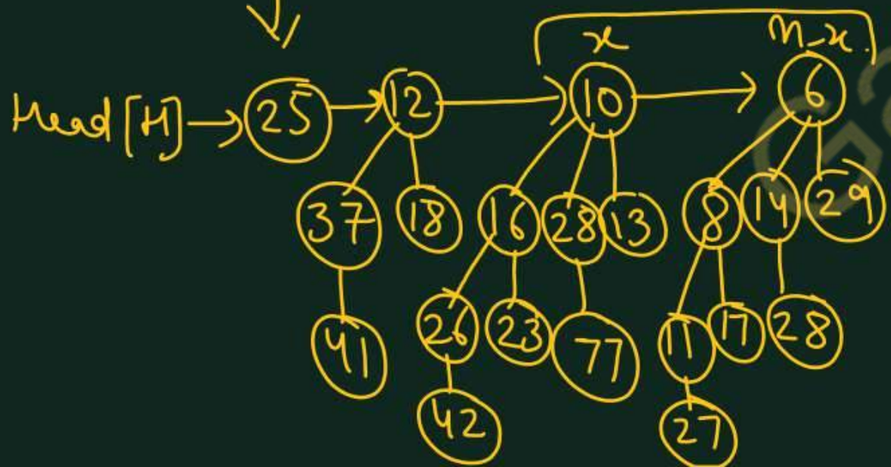




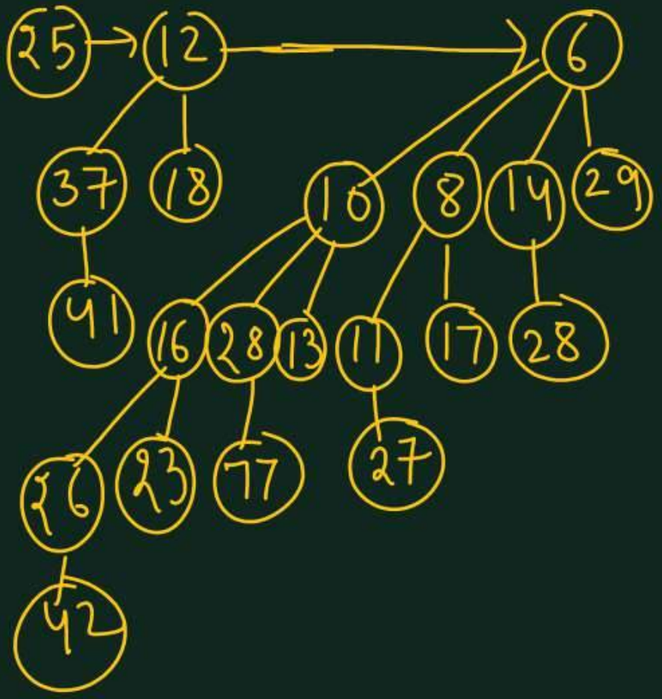
After Merging two binomial Heaps H and H' →







Final Binomial Heap

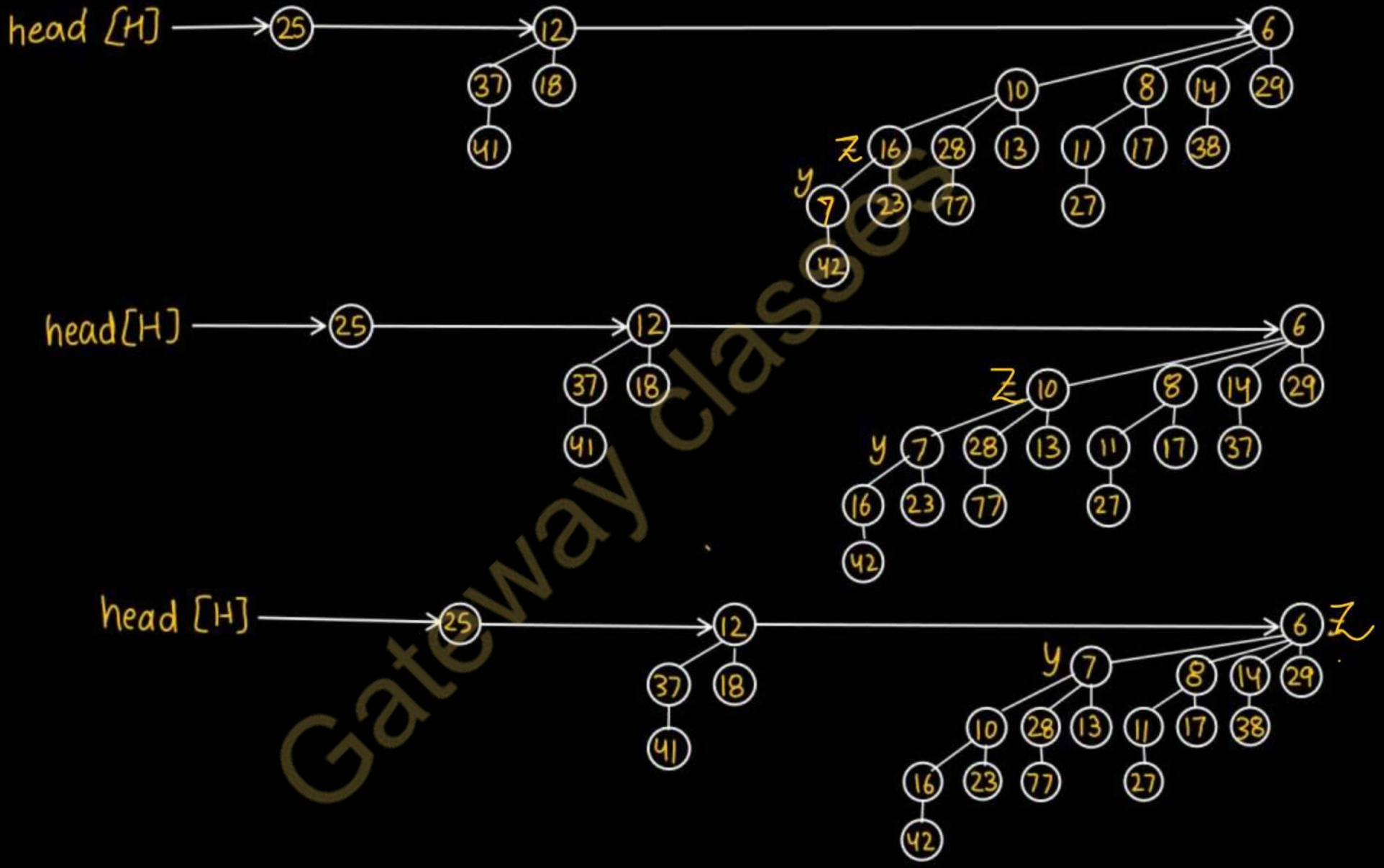


Decreasing a key

- The procedure decreases the key of node x in a binomial heap H to a new value k.
- It signals an error if k is greater than x's current key.

```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
1       if k > key[x]
2           then error "new key is greater than current key"
3       key[x] ← k
4       y ← x
5       z ← p[y]
6       while z ≠ NIL and key[y] < key[z]
7               do exchange key[y] ↔ key[z]
8       y ← z
9       z ← p[y]
```

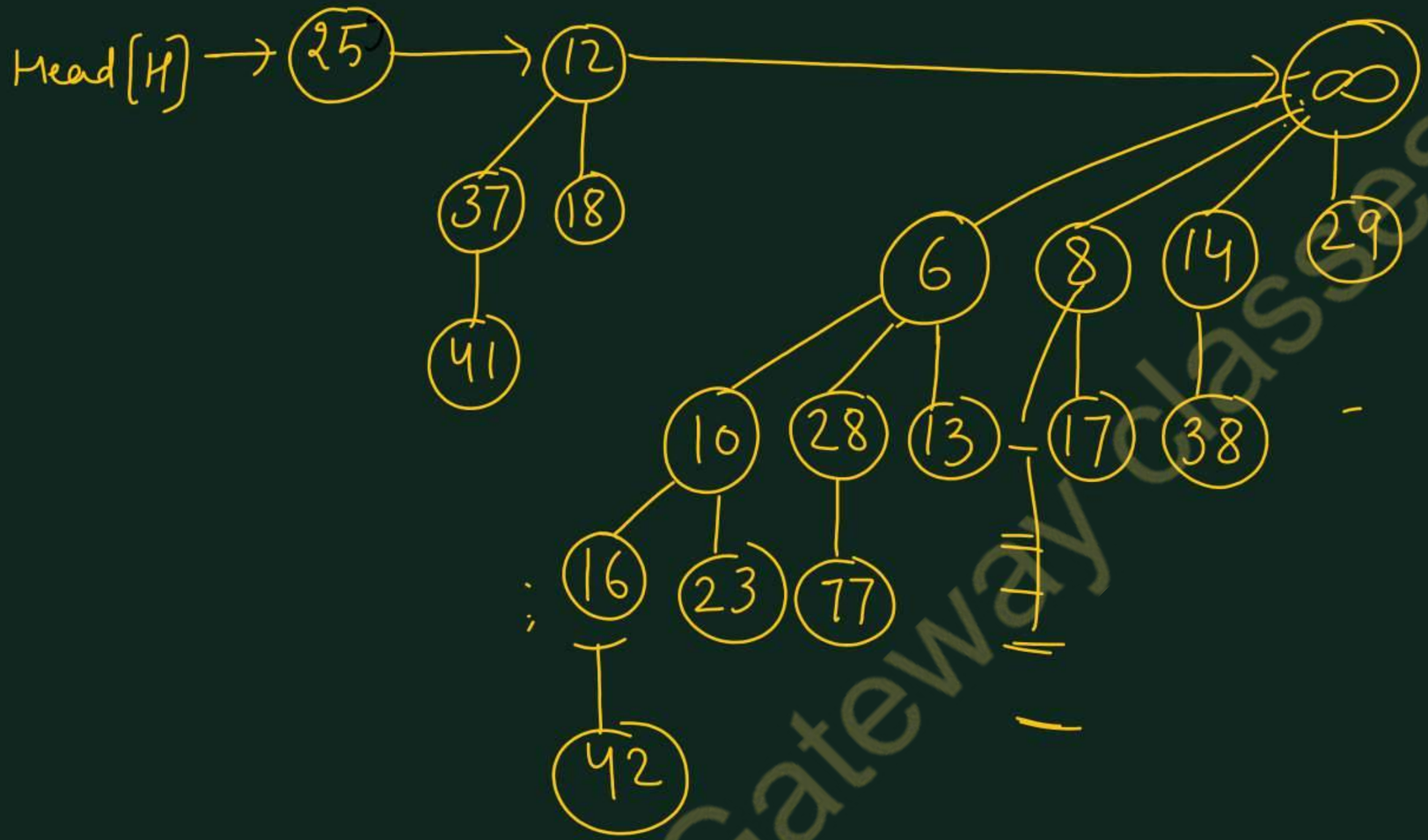
Procedure takes $O(\lg n)$ time



Deleting a key

- It deletes a node x's key from binomial heap H in $O(\lg n)$ time.
- The following implementation assumes that no node currently in the binomial heap has a key of $-\infty$.

```
BINOMIAL-HEAP-DELETE(H, x)
1       BINOMIAL-HEAP-DECREASE-KEY(H, x, −∞)
2       BINOMIAL-HEAP-EXTRACT-MIN(H)
```



Operations	Binary Heap	Binomial Heap	Fibonacci Heap
Procedure	Worst-case	Worst-case	Amortized
Making Heap	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Inserting a node	$\Theta(\log(n))$	$O(\log(n))$	$\Theta(1)$
Finding Minimum key	$\Theta(1)$	$O(\log(n))$	$O(1)$
Extract-Minimum key	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$
Union or merging	$\Theta(n)$	$O(\log(n))$	$\Theta(1)$
Decreasing a Key	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(1)$
Deleting a node	$\Theta(\log(n))$	$\Theta(\log(n))$	$O(\log(n))$

AKTU PYQs

1. What is binomial heap? Write the algorithm for decrease key operation in binomial heap and also write its complexity.(AKTU 2021-22)
2. Explain the algorithm for deleting a given element from binomial heap. Give an example for the same. (AKTU 2019-20)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

UNIT-2: Advanced Data Structures

Lecture-9

Today's Target

Fibonacci Heap

- Properties of Fibonacci Heap
- Fibonacci Heap Operations
- AKTY PYQs

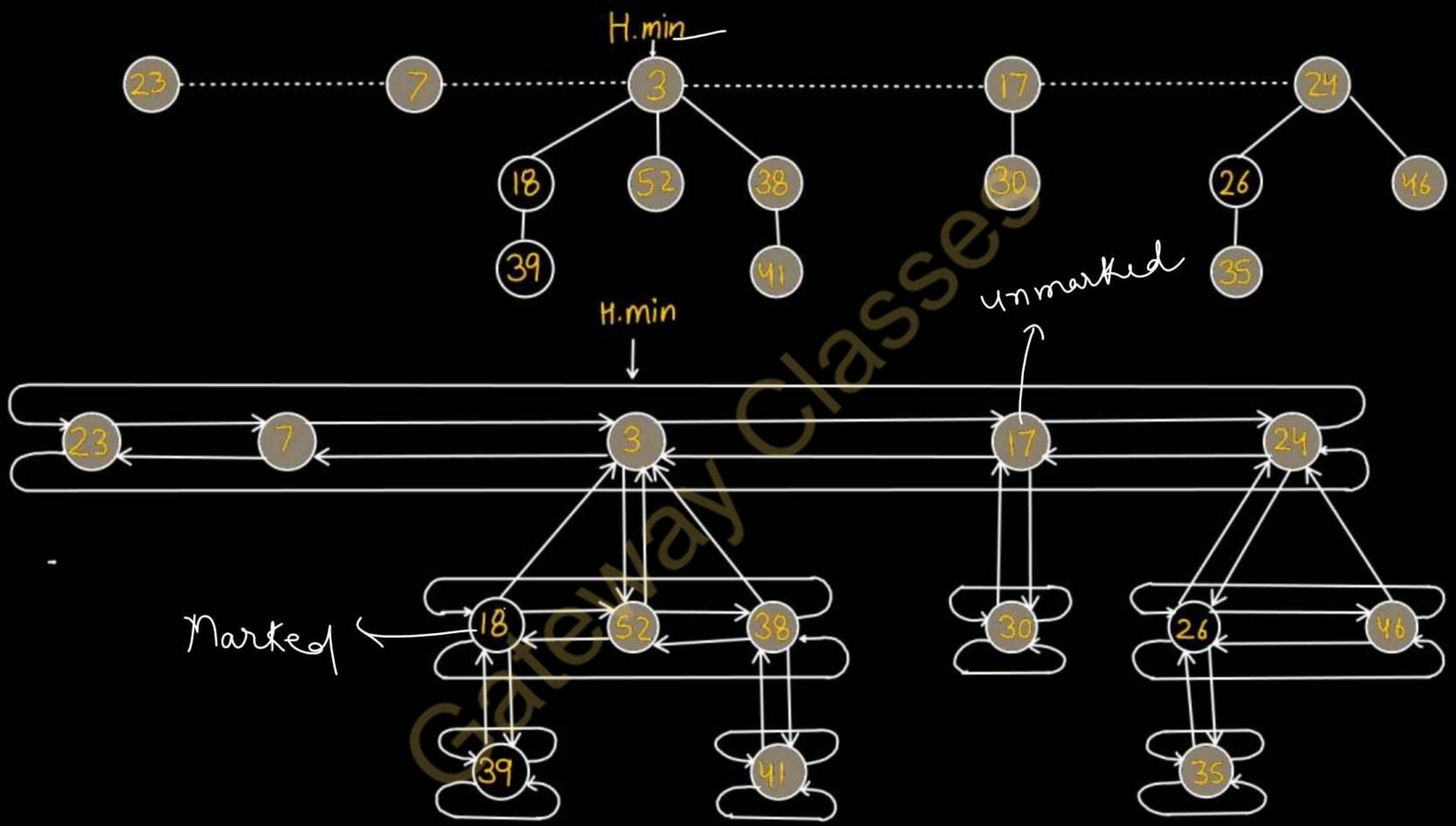
By Dr. Nidhi Parashar Ma'am

- M.Tech Gold Medalist
- Net Qualified

Fibonacci Heap

- It is a set of min heap-ordered trees. (i.e. The parent is always smaller than the children.)
- It can have multiple trees of equal degrees, and each tree doesn't need to have 2^k nodes.
- Fibonacci heap H is accessed by a pointer H:*min* to the root of a tree containing the minimum key. If more than one root has a key with the minimum value, then any such root may serve as the minimum node. When a Fibonacci heap H is empty, H:*min* is NIL.
- All the roots and siblings are stored in a separated circular-doubly-linked list. The main advantages of using a circular doubly linked list is that deleting a node from the tree takes O(1) time and concatenation of two such lists takes O(1) time
- Each node is marked or unmarked. Unmarked represents that the node has lost one child. The newly created node is unmarked.

Potential function of Fibonacci heap = No. of rooted tree + 2* number of marked nodes = 5 + 2 * 4 = 13.



Creating a Fibonacci Heap

- The procedure allocates and returns the Fibonacci heap object H
- Set H.*n* = 0 and
- H.*min* = NIL;
- There are no trees in H.

The amortized cost of MAKE-FIB-HEAP is thus equal to its O(1) actual cost.

Finding the minimum node

The minimum node of a Fibonacci heap H is given by the pointer H.*min*, so we can find the minimum node in O(1) actual time.

Inserting a node

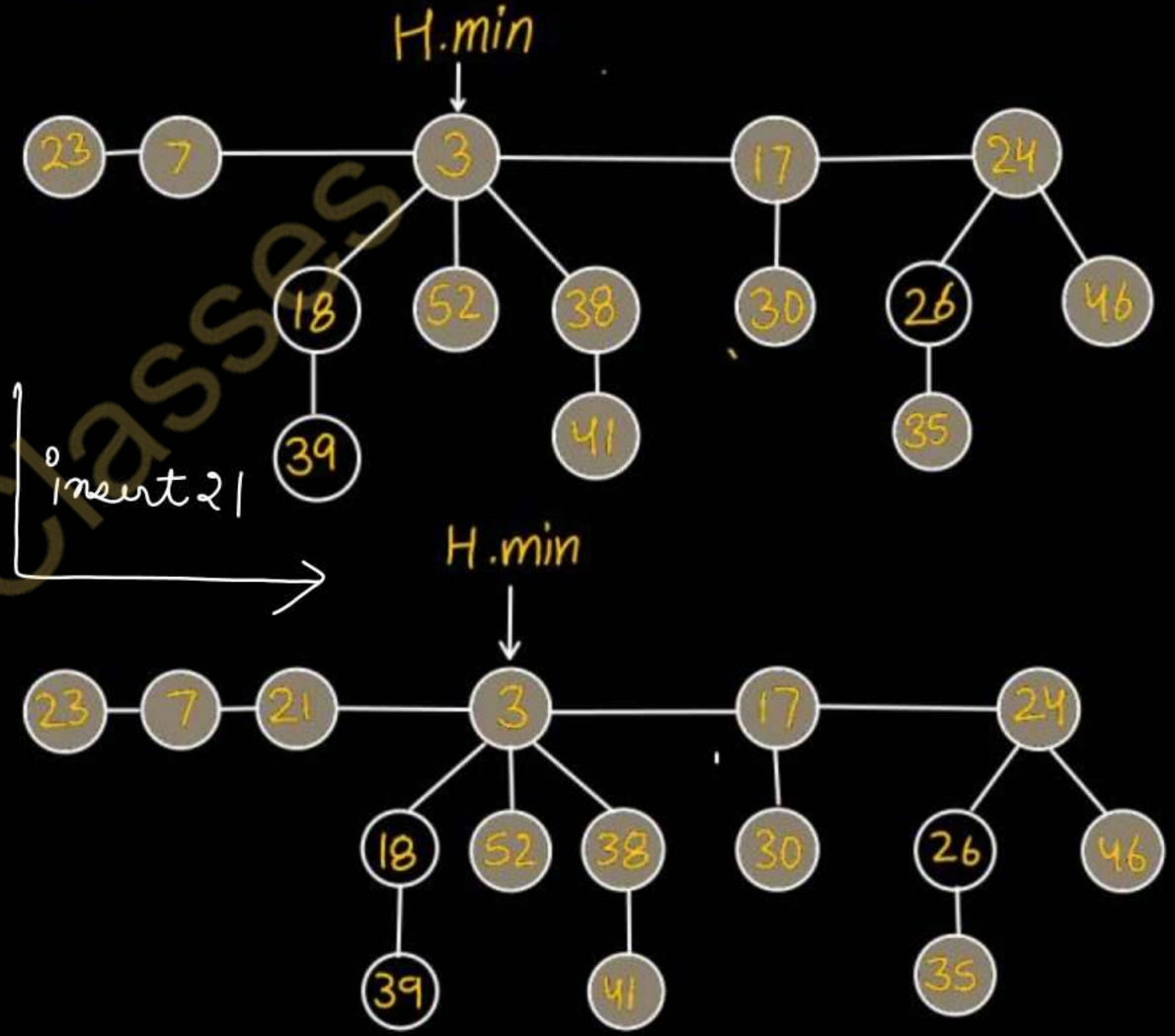
```
FIB-HEAP-INSERT(H , x)
1       x. degree =0
2       x. p =NIL
3       x. child =NIL
4       x. mark = FALSE
5       if H. min == NIL
6       create a root list for H containing just x
7       H. min = x
8       else insert x into H's root list
9       if x. key < H. min. key
10      H. min = x
11      H. n = H. n + 1
```

setting up the new node pointers

// Empty heap

H.min

21



Time Complexity: O(1)

Uniting two Fibonacci heaps

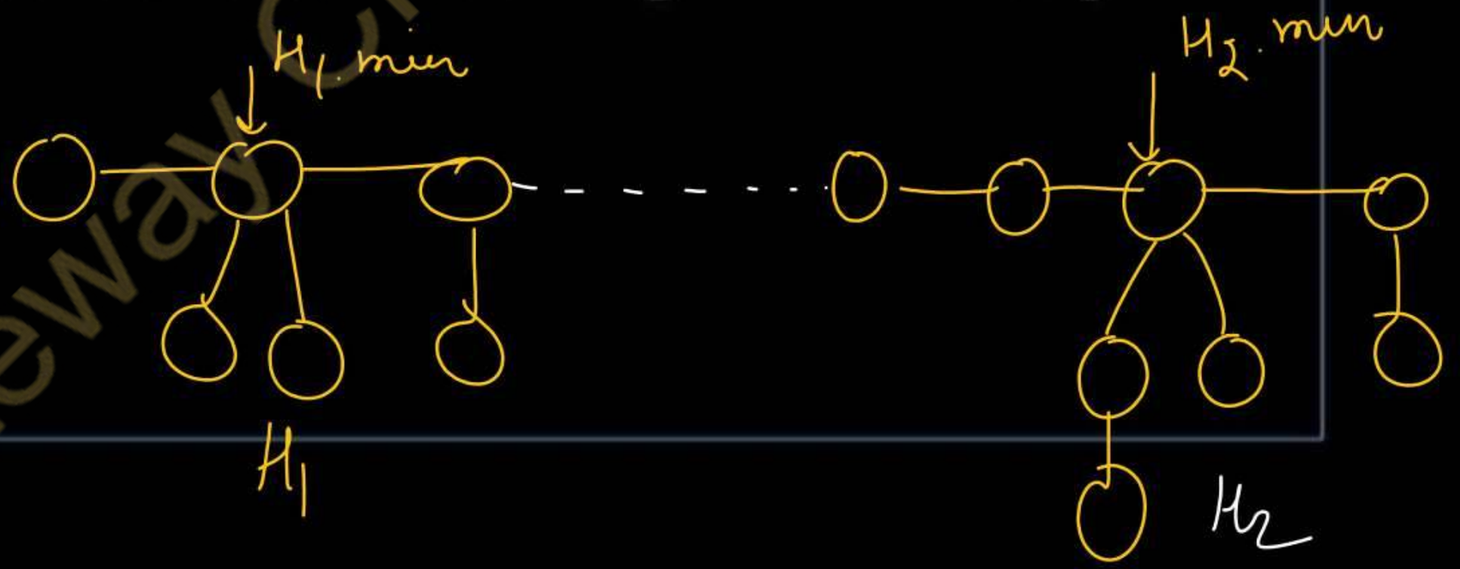
The following procedure unites Fibonacci heaps H1 and H2, destroying H1 and H2 in the process. It simply concatenates the root lists of H1 and H2 and then determines the new minimum node.

FIB-HEAP-UNION(H1,H2)

1. Join root lists of Fibonacci heaps H1 and H2 and make a single Fibonacci heap H.
2. If H1(min) < H2(min) then:

   H(min) = H1(min).
3. Else:

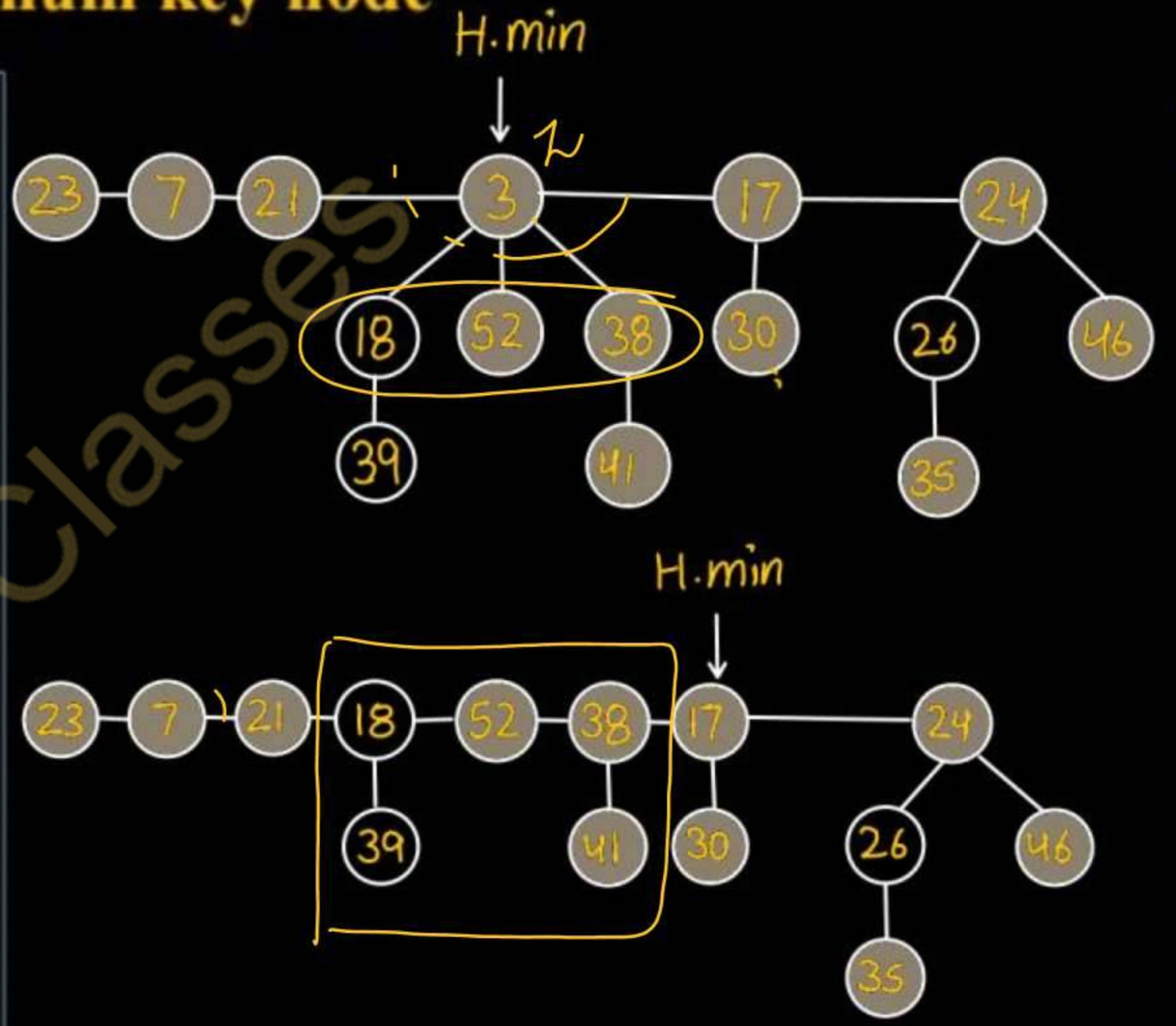
   H(min) = H2(min).

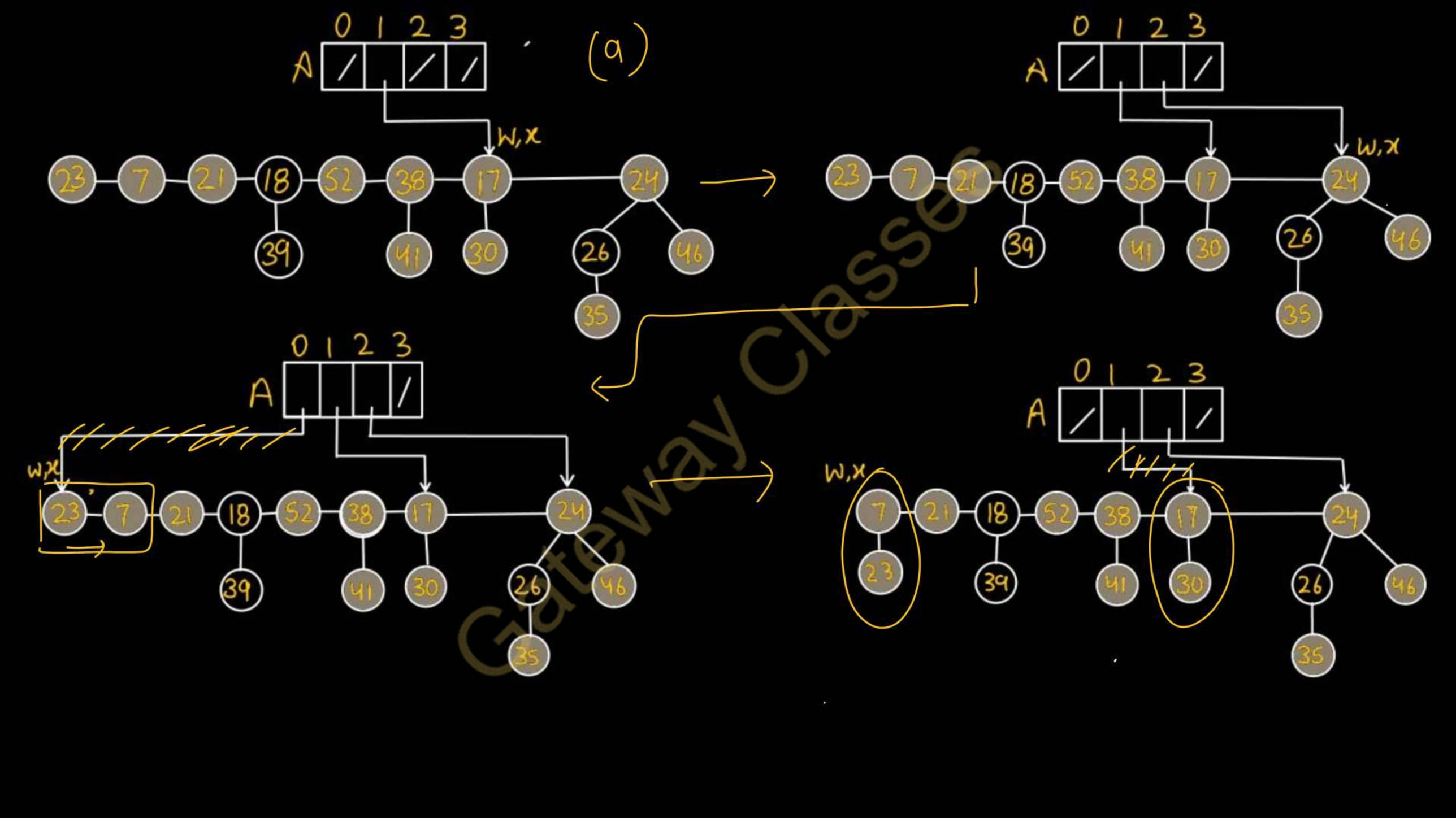


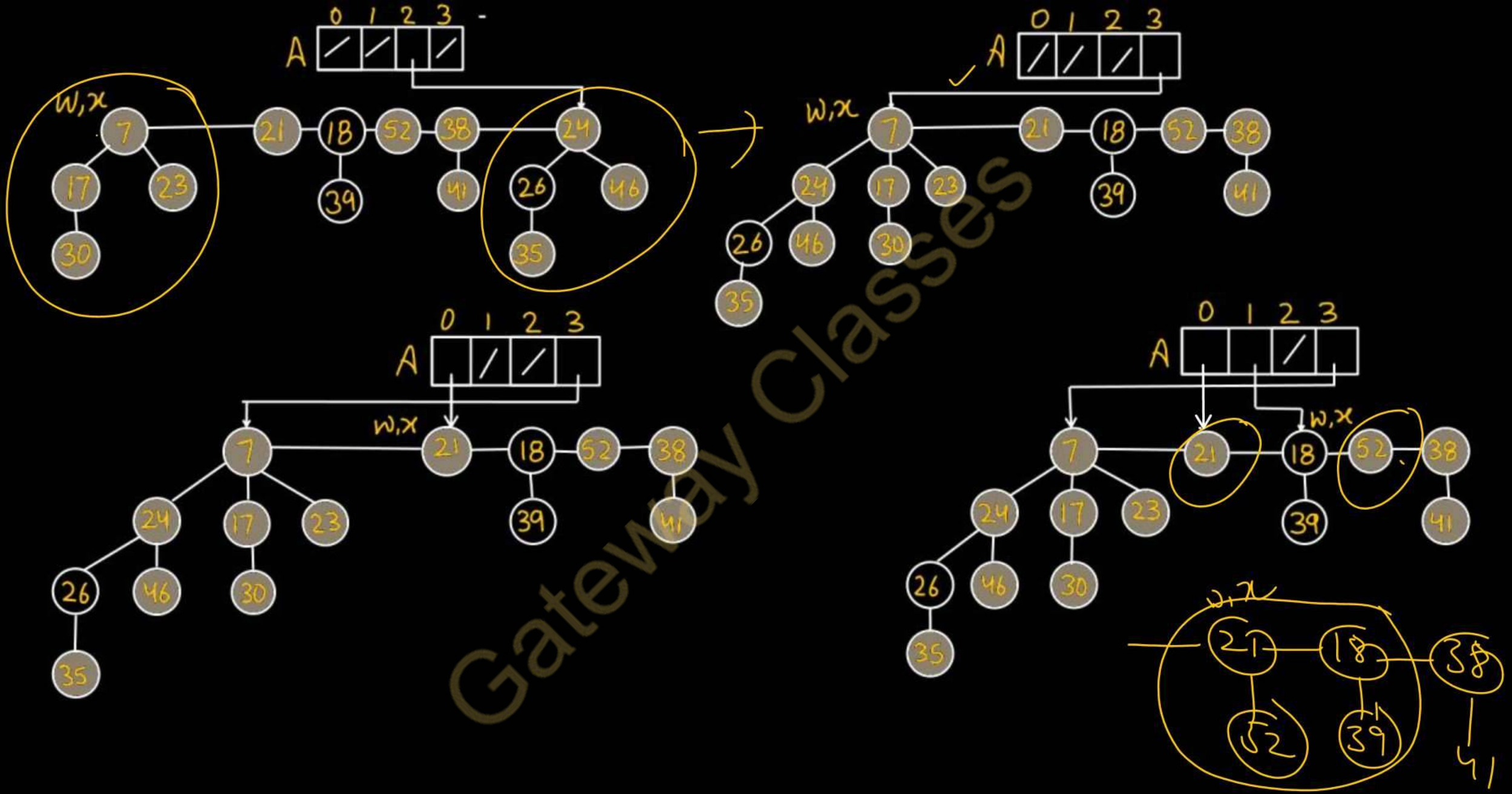
The amortized cost of FIB-HEAP-UNION is therefore equal to its O(1) actual cost.

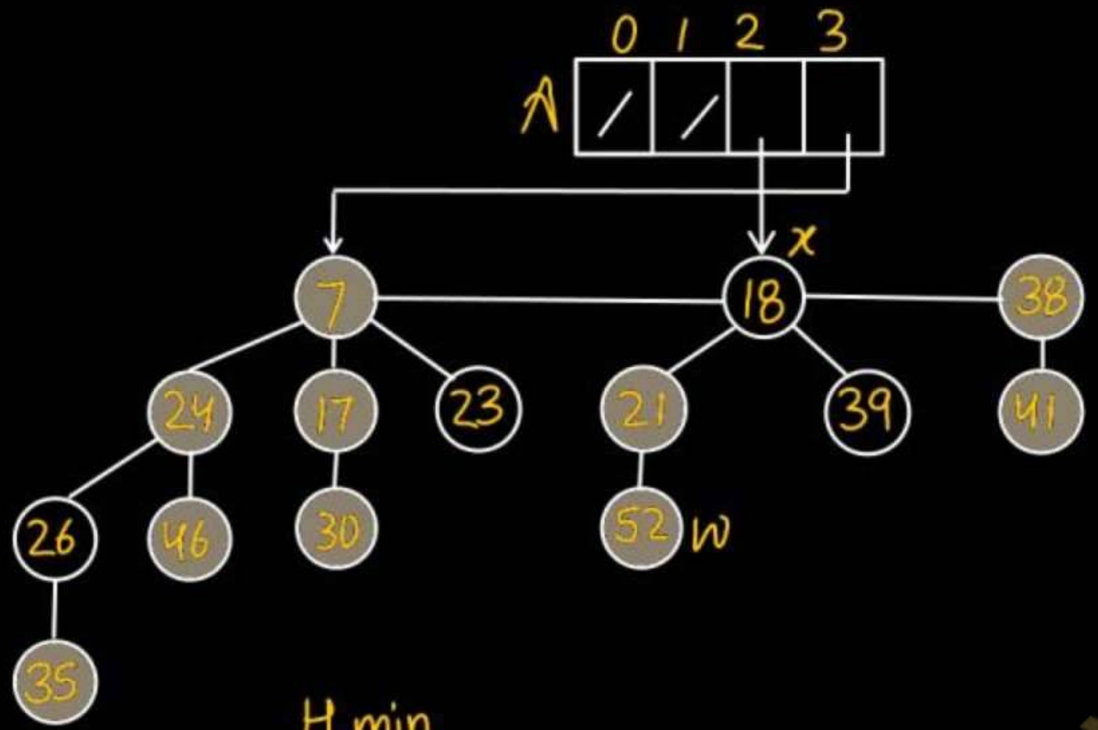
Extracting the minimum key node

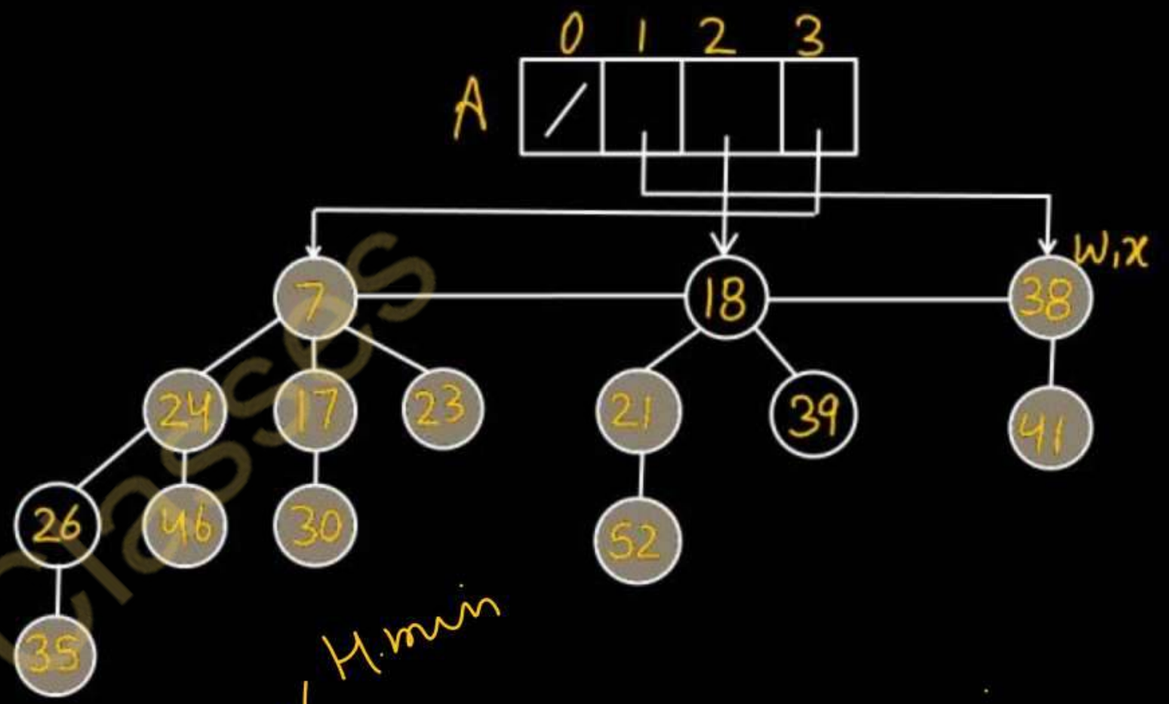
```
FIB-HEAP-EXTRACT-MIN(H)
1  z = H. min
2  if  z ≠ NIL
3        for each child x of z
4              add x to the root list of H
5              x. p = NIL
6        remove z from the root list of H
7        if z == z.right
8              H.min = NIL
9        else H. min = z.right
10             CONSOLIDATE(H)
11       H.n = H.n-1
12 return z
```

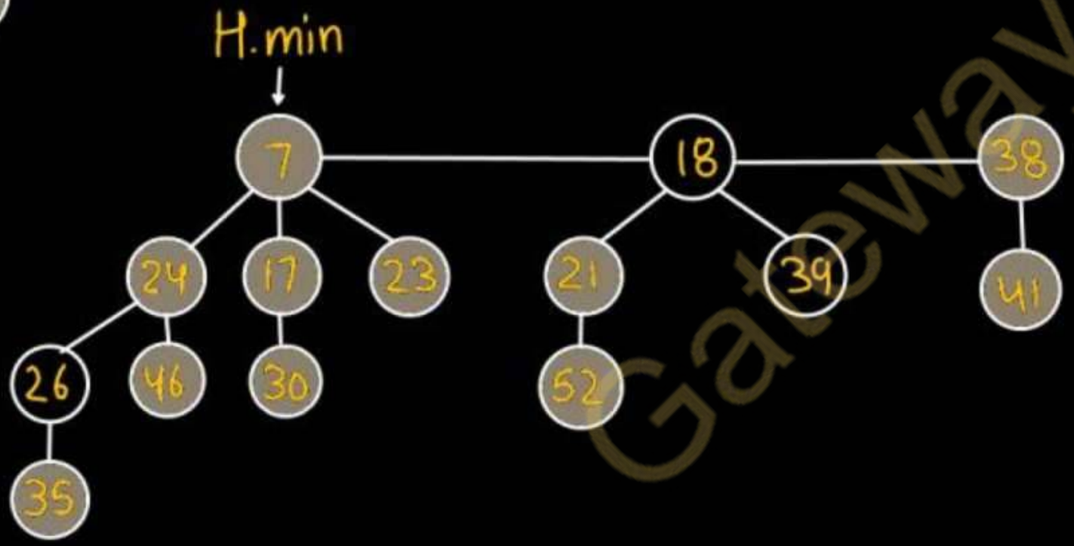


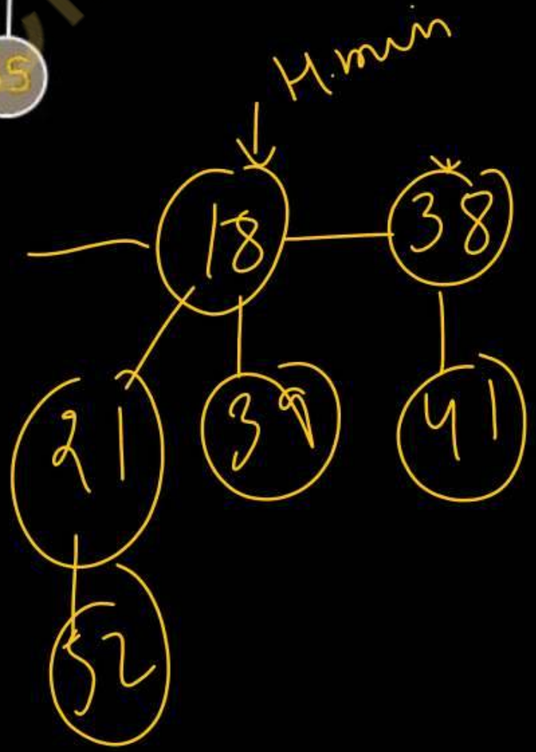






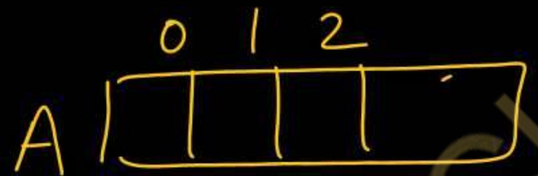


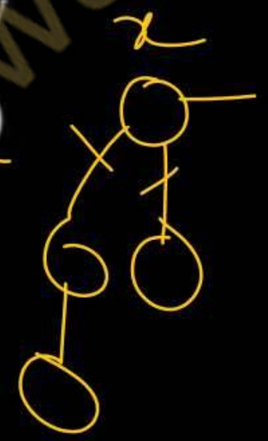




```
CONSOLIDATE(H)
1 let A[0....D(H. n)] be a new array
2 for i = 0 to D(H.n)
3       A[i] = NIL
4 for each node w in the root list of H
5       x = w
6       d = x.degree
7       while A[d] ≠ NIL
8             y = A[d] // another node with the same degree as x
9             if x.key > y.key
10                  exchange x with y
11            FIB-HEAP-LINK(H,y, x)
12            A[d] = NIL
13            d = d + 1
14      A[d] = x
15 H.min = NIL
```

initialization of pointer array





This code ensures that no two trees in the fibonacci heap have same degree

```
16 for i = 0 to D(H.n)
17        if A[i] ≠ NIL
18                  if H.min == NIL
19                            create a root list for H containing just A[i]
20                            H.min = A[i]
21                  else insert A[i] into H's root list
22                  if A[i].key < H.min:key
23                            H.min = A[i]
```


```
FIB-HEAP-LINK(H, y, x)
1         remove y from the root list of H
2         make y a child of x, incrementing x. degree
3         y. mark = FALSE
```

Amortized cost of extracting the minimum node is O(lg n)

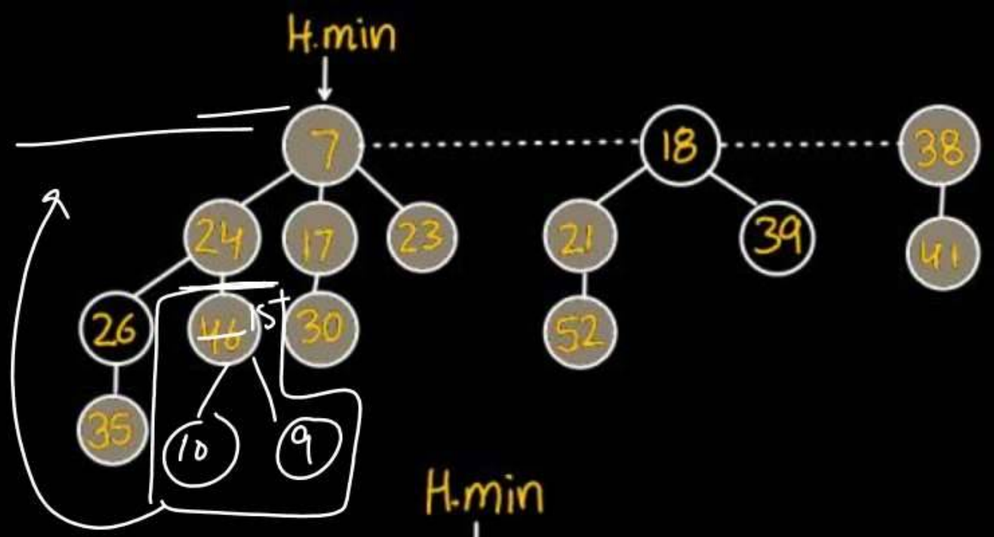
Decreasing a key

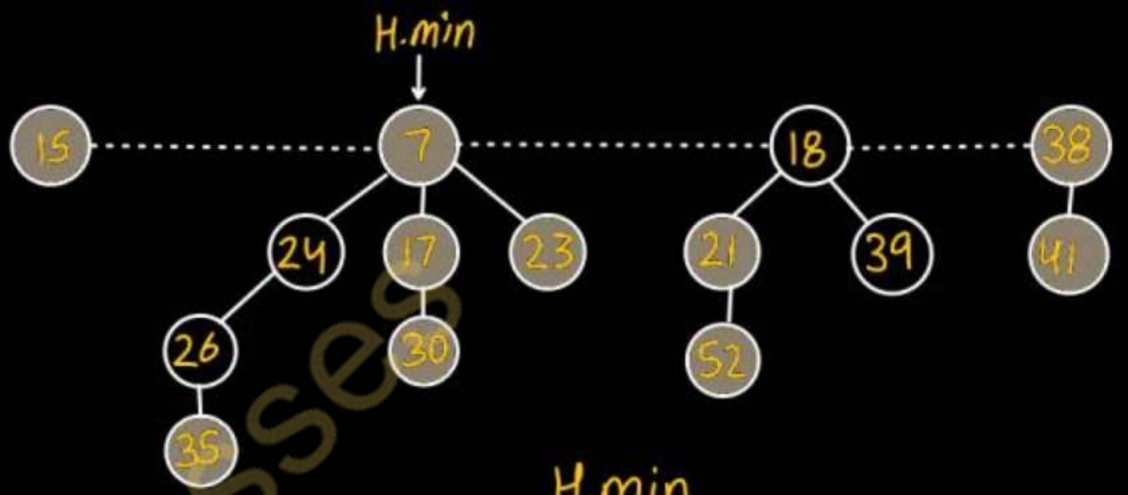
```
FIB-HEAP-DECREASE-KEY(H, x, k)
1 if k > x. key
2         error "new key is greater than current key"
3 x. key = k
4 y = x. p
5 if y ≠ NIL and x. key < y. key
6         CUT(H, x, y)
7         CASCADING-CUT(H, y)
8 if x. key < H .min .key
9         H.min = x
```

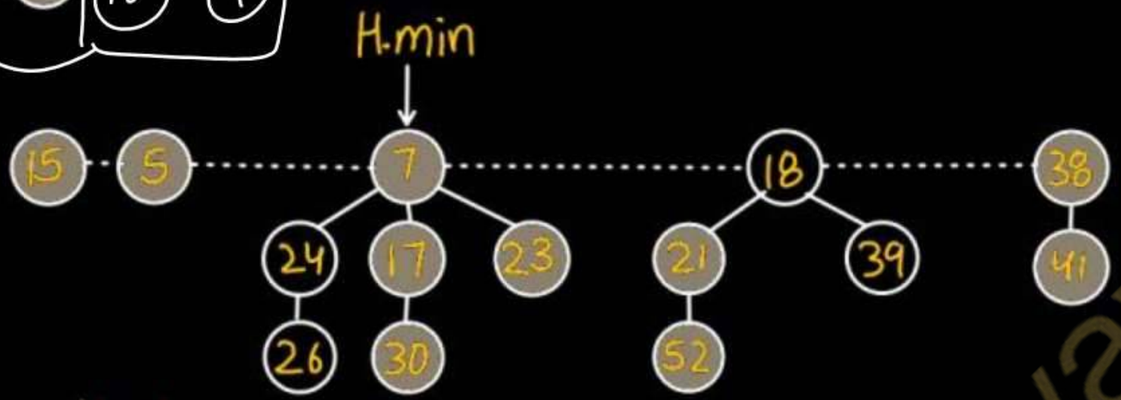
```
CUT(H, x, y)
1.remove x from the child list of y, decrementing y. degree
2. add x to the root list of H
3. x.p = NIL
4. x.mark = FALSE
```

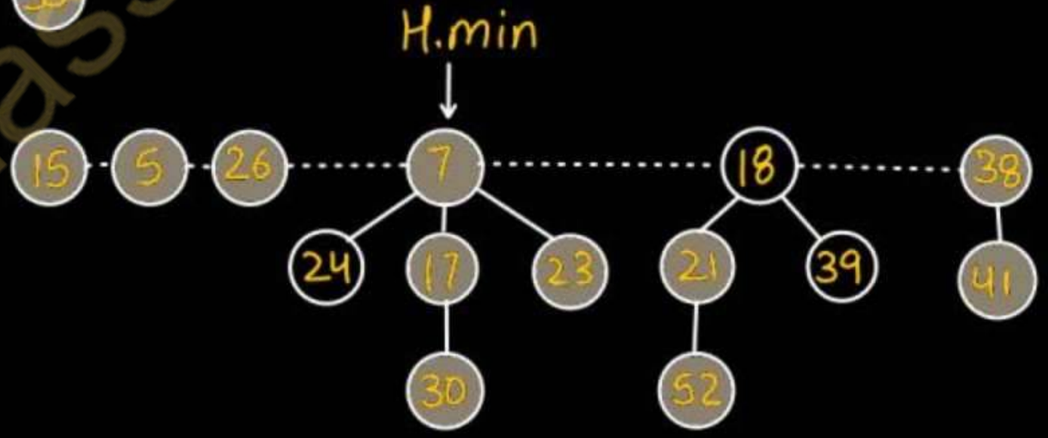
```
CASCADING-CUT(H, y)
1. z = y. p
2. if z ≠ NIL
3.         if y. mark== FALSE
4.                   y. mark= TRUE
5.         else CUT(H, y, z)
6.                   CASCADING-CUT(H, z)
```

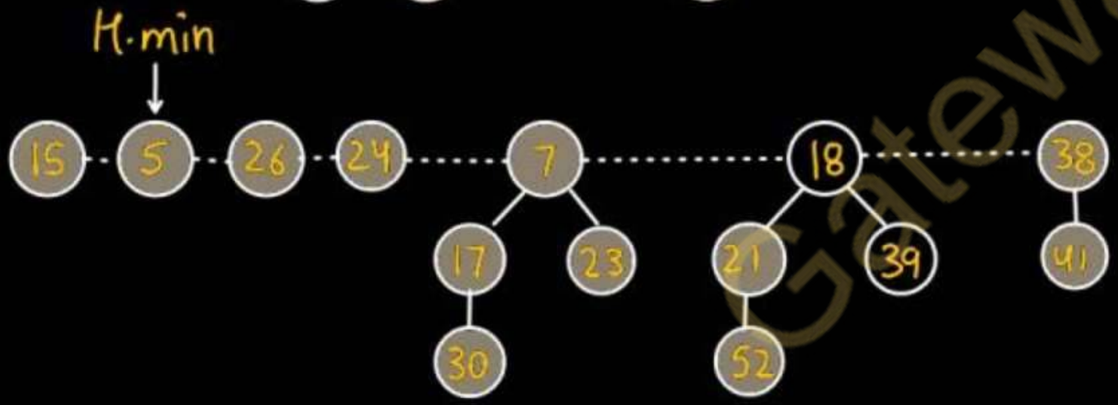
amortized cost of FIB-HEAP-DECREASE-KEY is at most O(1)











Deleting a node

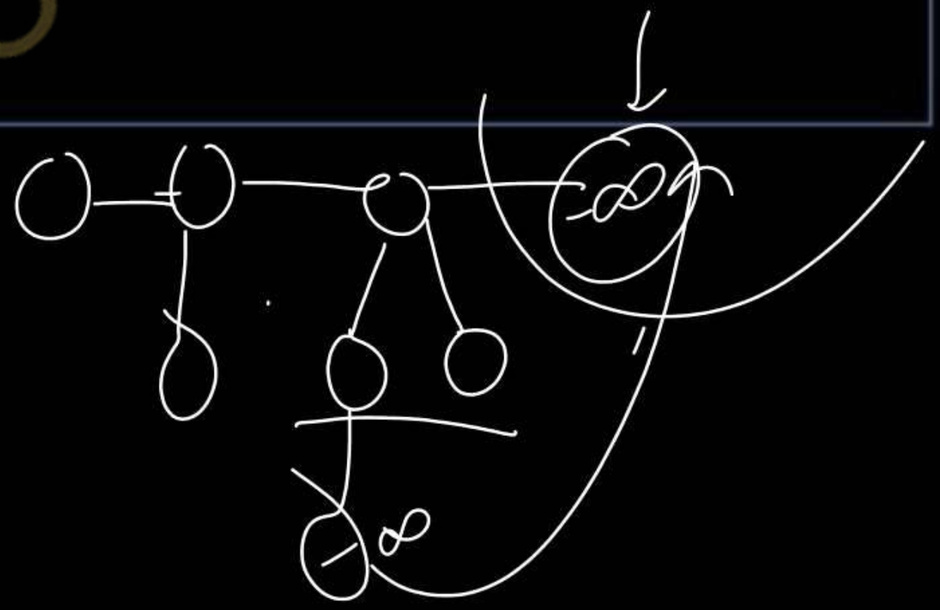
We assume that there is no key value of $-\infty$ currently in the Fibonacci heap.

FIB-HEAP-DELETE(H, x)

1 FIB-HE AP-DECREASE-KEY(H, x, $-\infty$)

2 FIB-HEAP-EXTRACT-MIN.H/

The amortized cost is: O(log n).



Feature	Fibonacci Heap	Binomial Heap
Structure	Collection of min-heap-ordered trees	Collection of binomial trees
– Insertion	O(1) amortized	O(log n)
– Deletion	O(log n)	O(log n)
– Decrease-key	O(1) amortized	O(log n)
– Merging	O(1) amortized	O(log n)
Space Efficiency	Requires more space due to additional properties	Requires less space due to simpler structure
Key Advantage	Extremely fast insertion and merging	Efficient decrease-key operations
Applications	Advanced algorithms, specifically decrease-key operations	Priority queues, graph algorithms
Potential Trade-offs	Increased space overhead, larger constant factors	Slightly slower operations compared to Fibonacci heaps

AKTU PYQs

1. Write short note on Fibonacci heap. (AKTU 2023-24)
2. Write the properties of Fibonacci heap. (AKTU 2021-22)
3. What are the various differences in Binomial and Fibonacci Heap?Explain. (AKTU 2022-23)

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

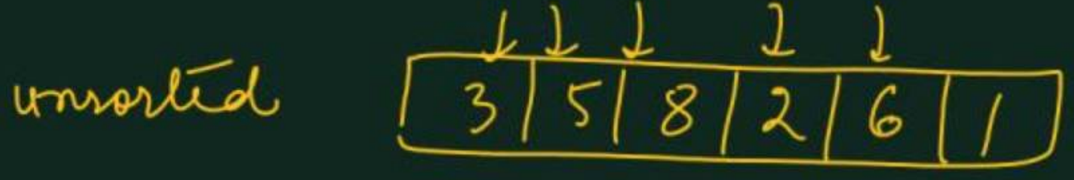
UNIT-2: Advanced Data Structures

Lecture-10

Today's Target

- Skip Lists
- Trie data structure
- AKTY PYQs

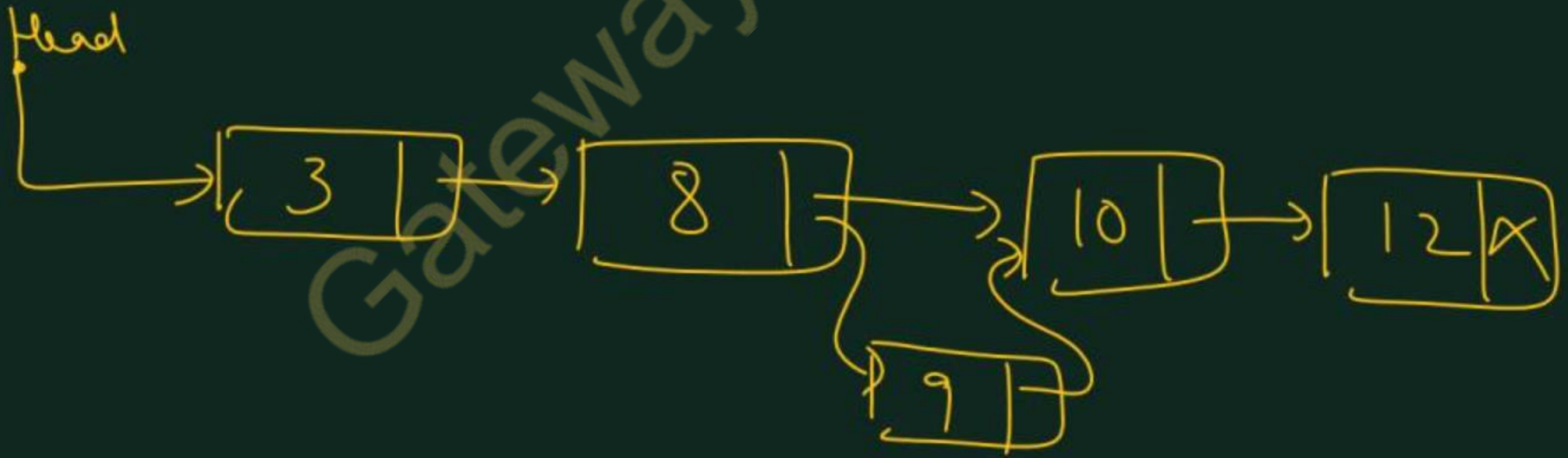
By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified



$O(n)$

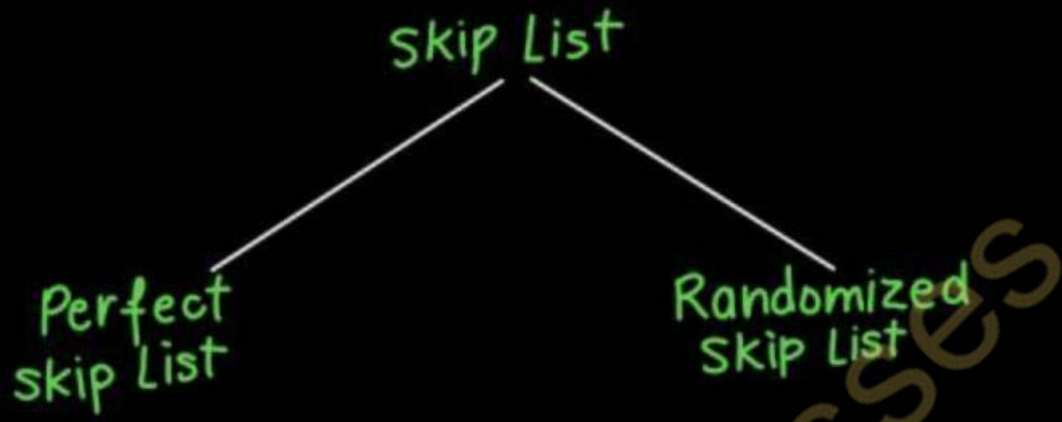
$O(\log n)$

insert/delete $= O(n)$



Skip List

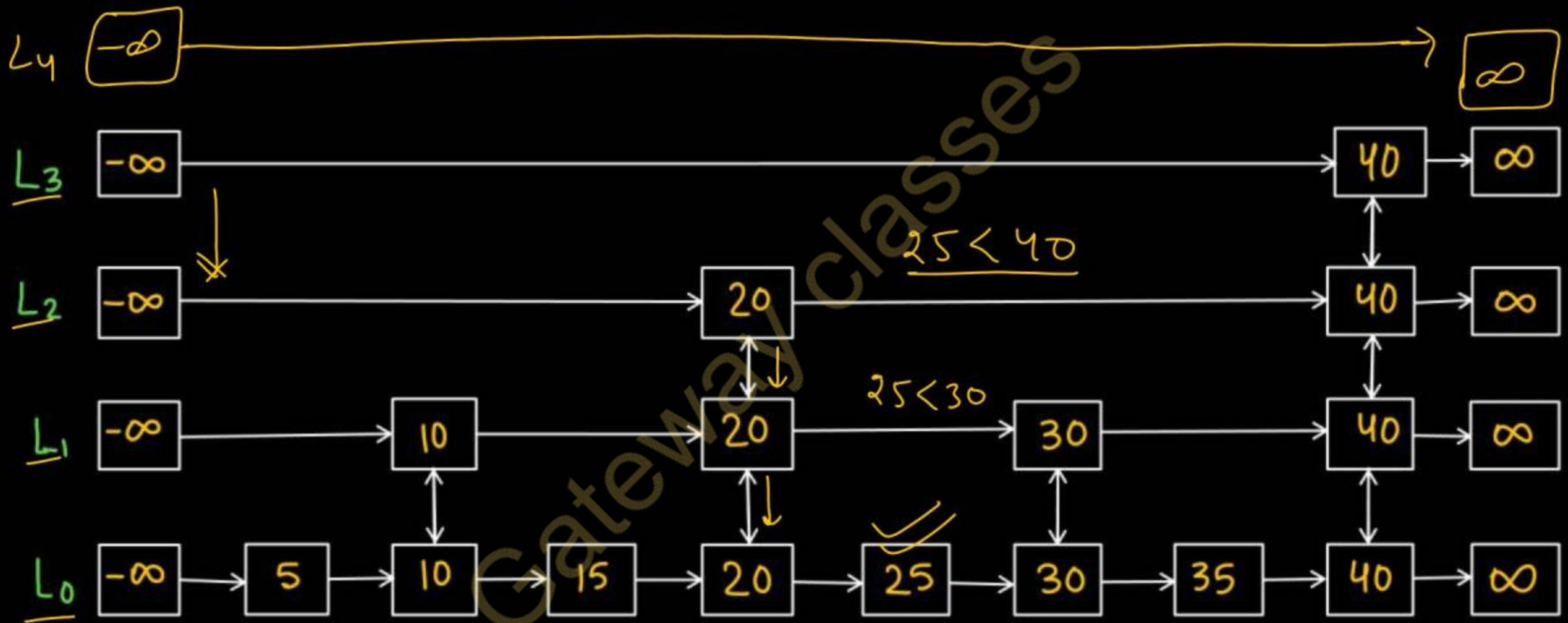
- A **probabilistic data structure** that allows efficient searching, insertion, and deletion operations in a sorted list.
- It **consists of multiple levels/layers**, with the bottom level representing the original sorted list.
- It **skips many elements from the whole list**, in one upper layer.
- The topmost layer acts as an **"express lane"** because it contains fewer elements than the layers below it.
- Each layer below has progressively more elements than the layer above it.
- By utilizing these additional layers, skip lists can effectively "**skip**" over a large number of elements in a single step during searches, reducing the average time complexity.
- Each element in a lower level has a pointer to the next occurrence of the same element in the level above allowing for efficient traversal and search operations using skip pointers.



Perfect Skip List

- Guarantees a balanced and predictable structure .
- Each level has exactly half the number of nodes as the level below it.
- The topmost level contains just two nodes (One is starting node and second is last node), and each subsequent level doubles the number of nodes compared to the level above it.

Search 25



- We started with a normal linked list (level 0)
- Every other node in level 0 (2nd node from original list) raised to level 1
- Every other node in level 1 (4th node from the original list) raised to level 2
- Every other node in level 2 (8th node from the original list) and raised it to level 3
- There will be $O(\log_2 n)$ levels

How long would it take us to find an item or determine it is not present in the list – O(log(n))

Proof –

- At each level we visit at most 2 nodes
- At any node, x, in level i , you sit between two nodes (p, q) at level i+1 and you will need to visit at most one other node in level i before descending
- There are O(log(n)) levels
- **So we visit at most O(2*log(n)) levels = O(log(n))**

Advantage of a perfect skip list

- It provides optimal search performance.
- The time complexity for search operations is O(log n), just like in a balanced binary search tree.

Disadvantage of perfect skip list

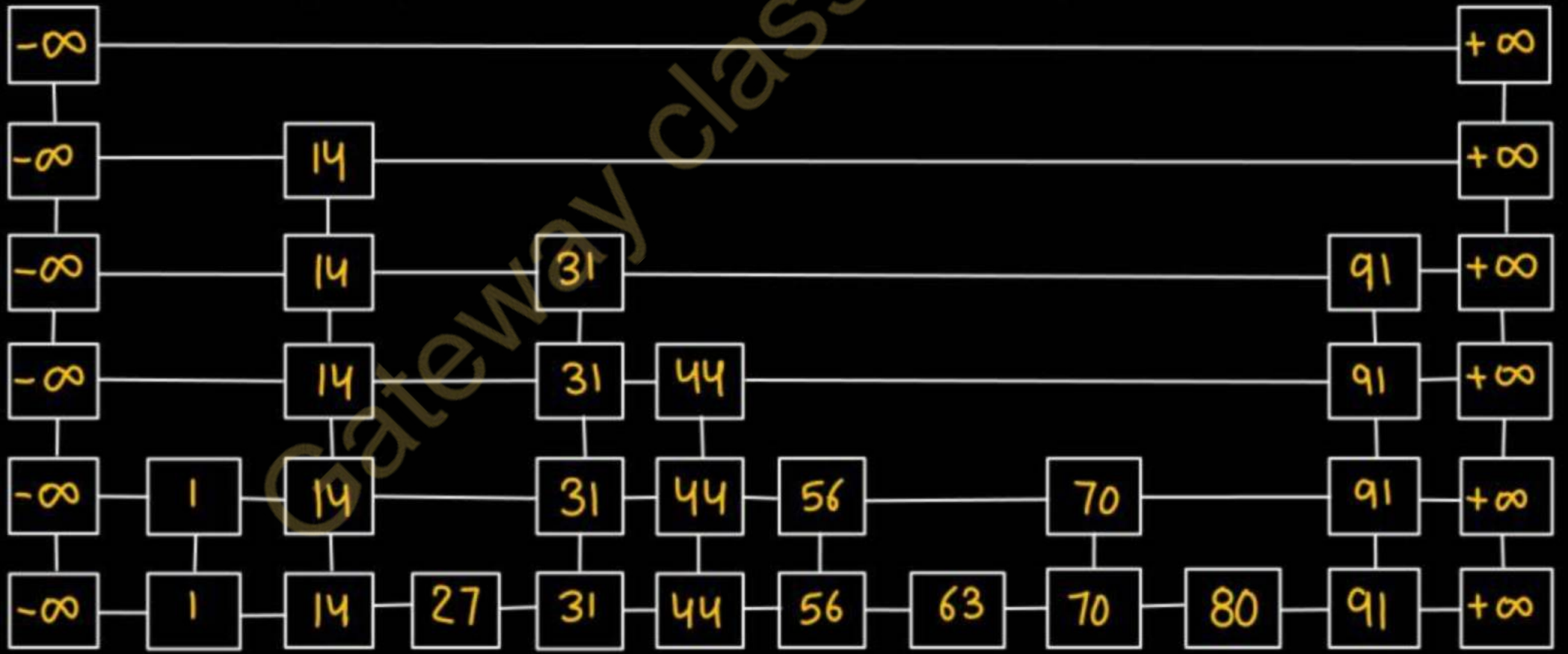
e delete 50 from the perfect skip list, the diagram is co...

ANE 3 has 7 and LANE 2 has only 3 elements.

- To insert a new node or remove a node. In the worst case, all n-1 remaining nodes would need their levels adjusted to maintain the pattern described above.
- Maintaining the perfect structure of a skip list during insertions and deletions can be more complex and may require additional operations to rebalance the levels.

Randomized Skip List

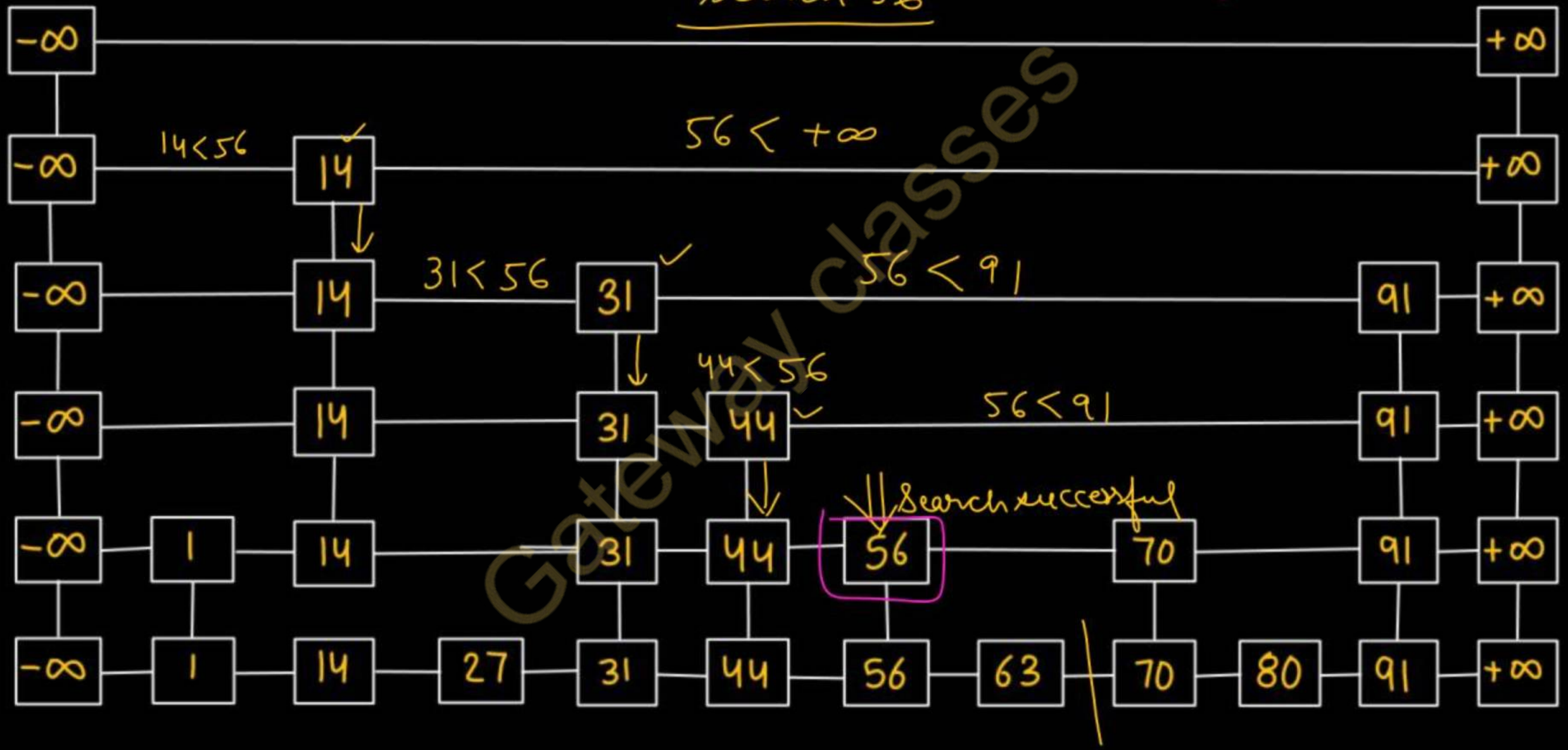
- Introduces randomness into the structure.
- The level of each node is determined randomly during the insertion process, following a certain probability distribution.
- Unpredictable structure that may vary between different instances of the skip list.



Searching in Skip List (Imp)

67

Search 56



Searching Algorithm (Imp)

SEARCH_SkipList(x) ← element to be searched

Start at the first position of the top list

At the current position p, compare x with y key(after(p))

```
if (x=y):
        return element(after(p))
if (x > y): we
        scan forward
if(x<y):
        drop down
```

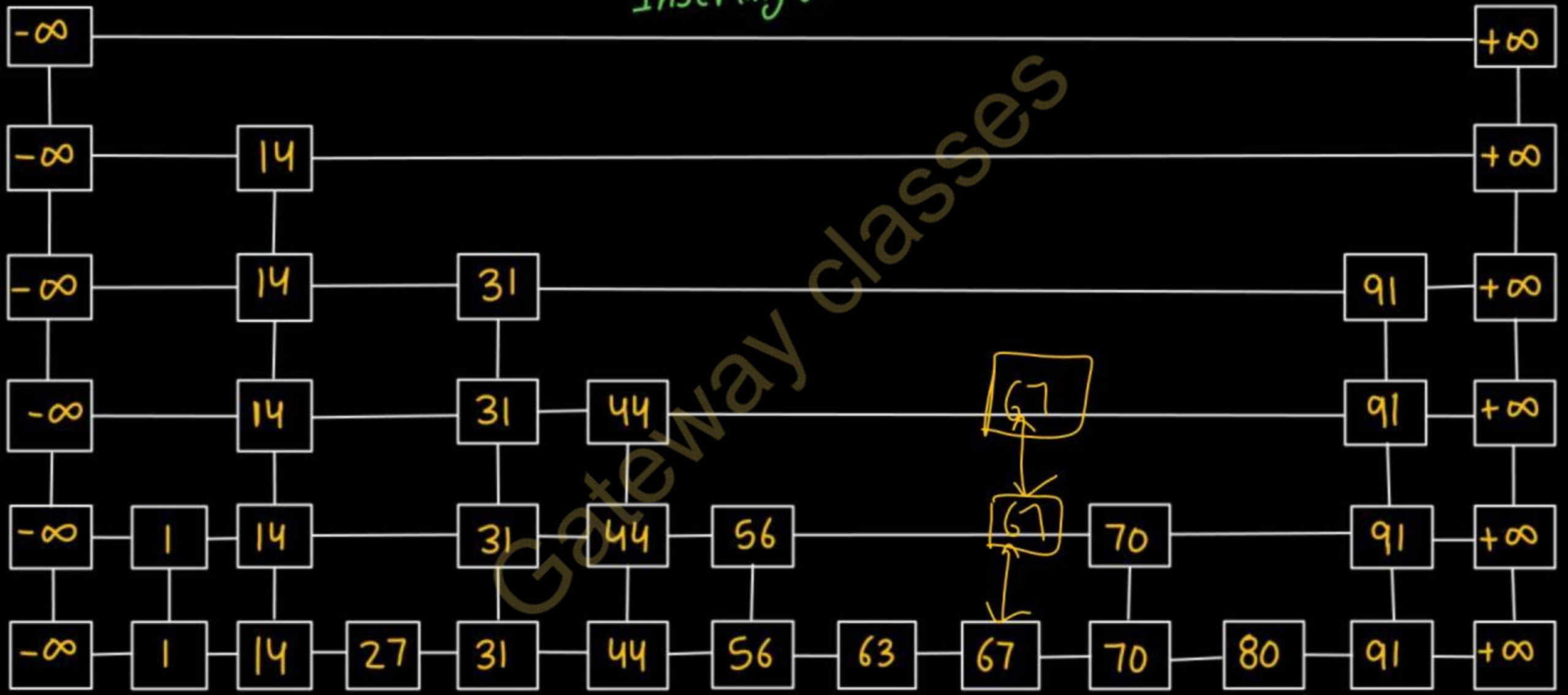
If drop down past the bottom list

```
return NO_SUCH_KEY
```

Insertion in Skip List

Skip list after inserting 67:

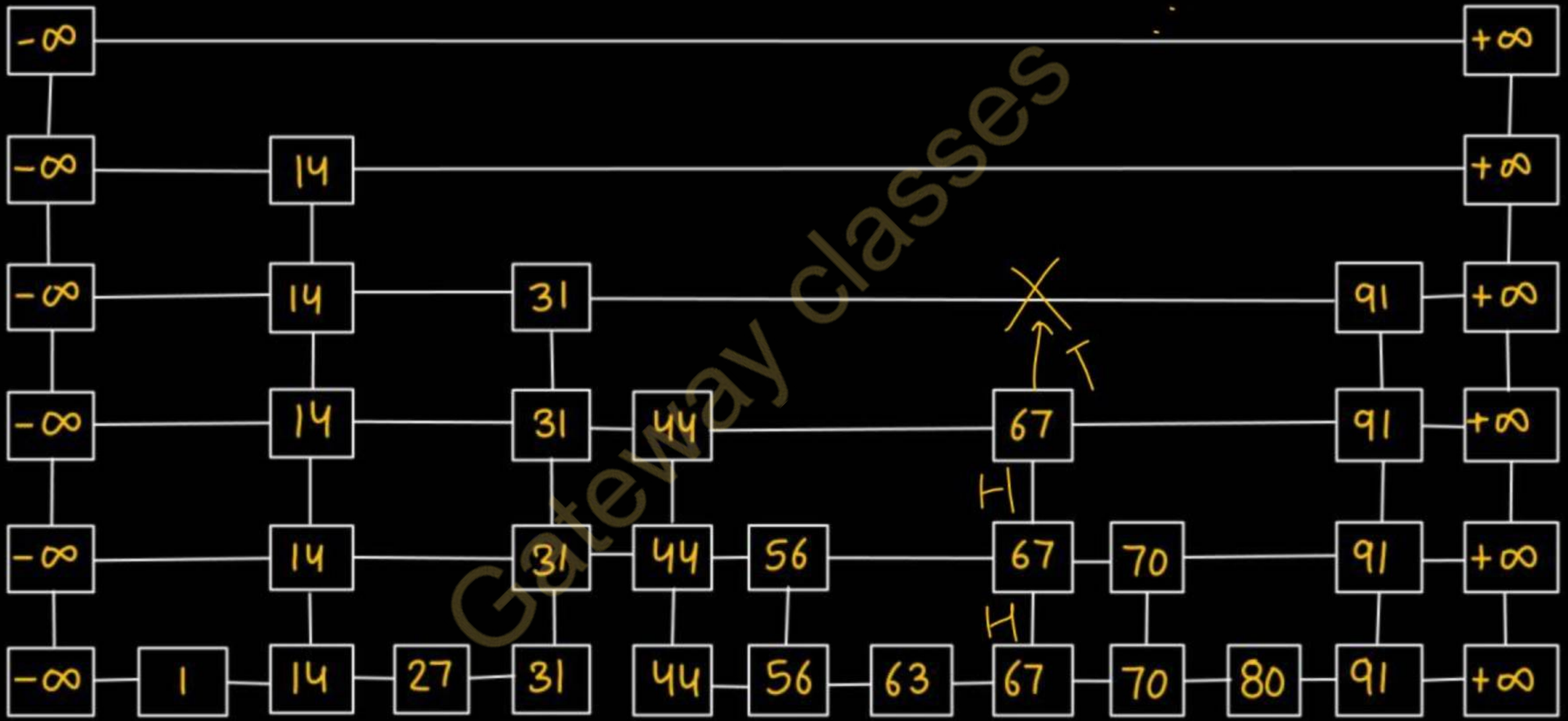
Inserting 67



Coin Randomization in Randomized Skip List

The coin flip is based on a probability distribution that determines the node level.

1. Begin with a new node that must be added to the skip list.
2. Using a coin flip, determine the node's level:
   - 2.1 New node is placed in the original lane.
   - 2.2 Toss a coin.
3. If head comes, then place new node in upper layer.
4. If tail comes, then stop the promotion in upper layers.
   - 4.1 You have to toss the coin until you reach the maximum lane, if head comes again and again.


-∞
+∞
-∞
14
+∞
-∞
14
31
91
+∞
-∞
14
31
44
67
91
+∞
-∞
14
31
44
56
67
70
91
+∞
-∞
1
14
27
31
44
56
63
67
70
80
91
+∞
H
H
T
Gateway classes

Skip List Insertion Algorithm

Randomized SkipList_Insert(x)

1. Repeatedly toss a coin until get tails
2. i the number of times the coin came up heads
3. if (i >= h)

   add to the skip list new lists $S_{h+1},\ldots\ldots,S_{i+1}$ each containing only the two

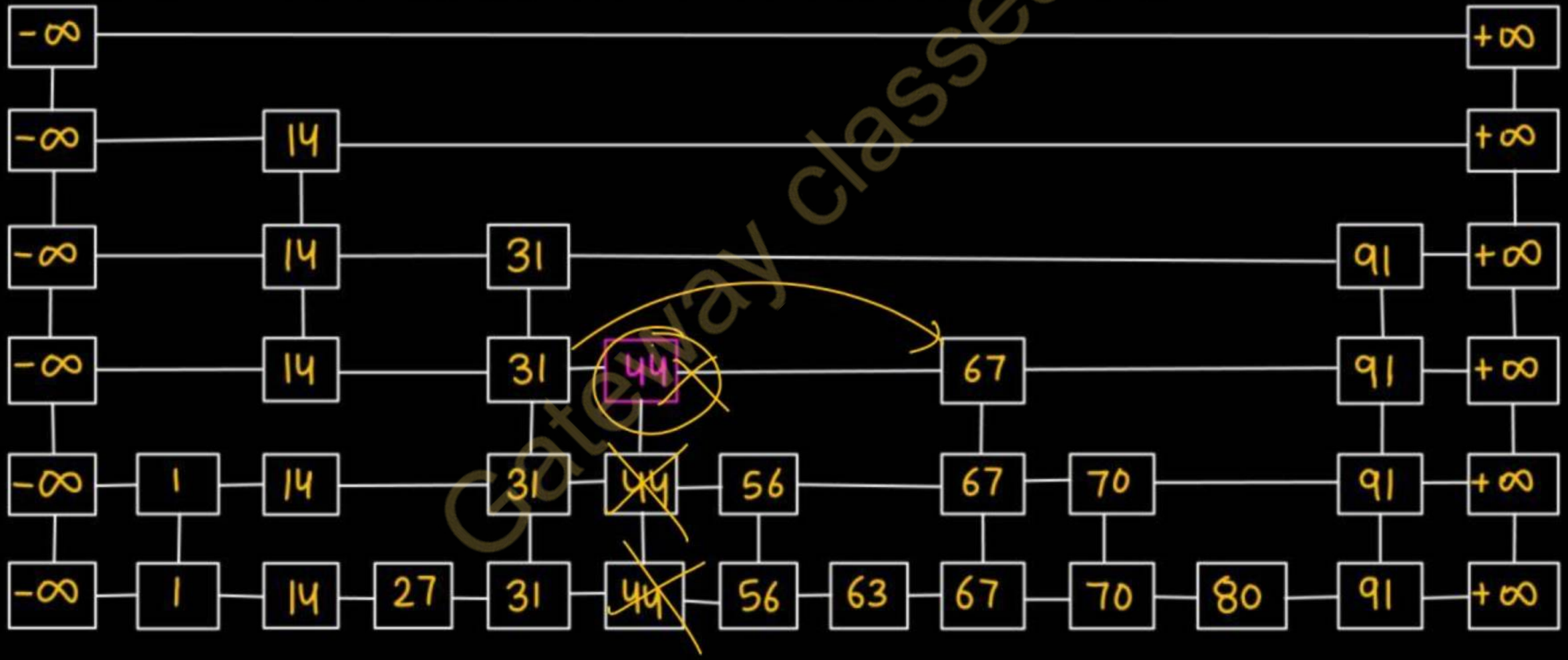
   - - special keys

4. Search for x in the skip list and find the positions $P_0, P_1,\ldots\ldots P_i$ of the items with largest key less than x in each list $S_0, S_1,\ldots S_i$.
5. For j =0,……i,

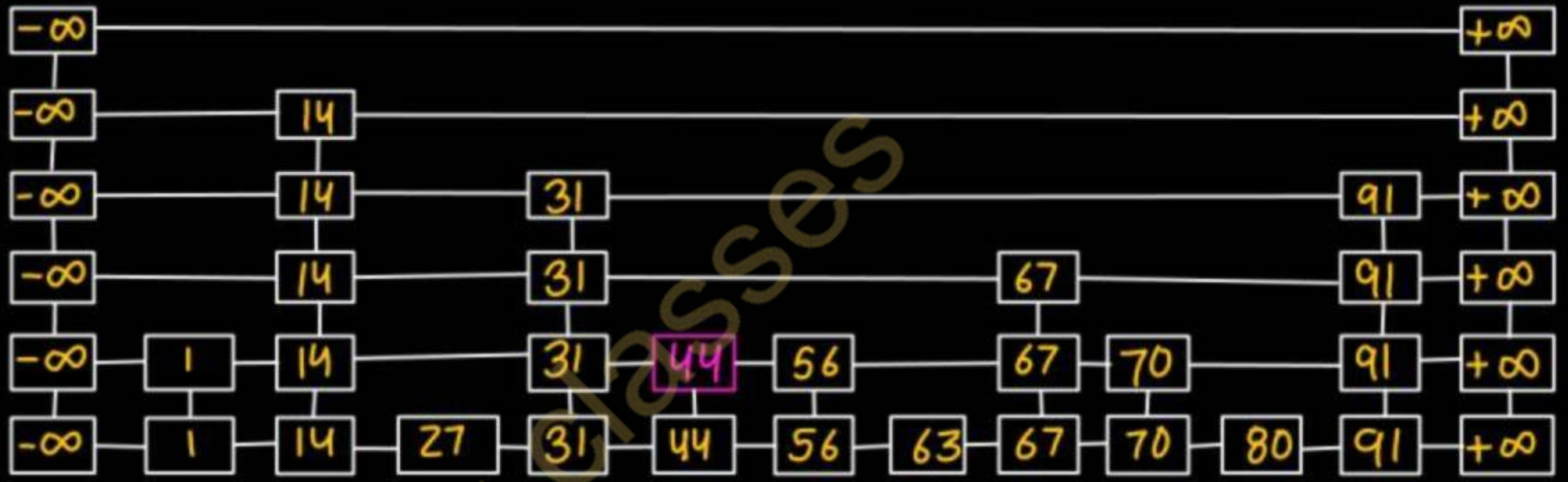
   insert item (x) into list S_j, after position P_j

Deletion in Skip List

- First search the node, and then simply delete it at all levels.
- Before deleting a node, simply ensure that no other node is pointing to it.



Remove all references to this node and release it. Move to the node below in the bottom list.



Release this node as well.

Deleting 44

Conclusion:

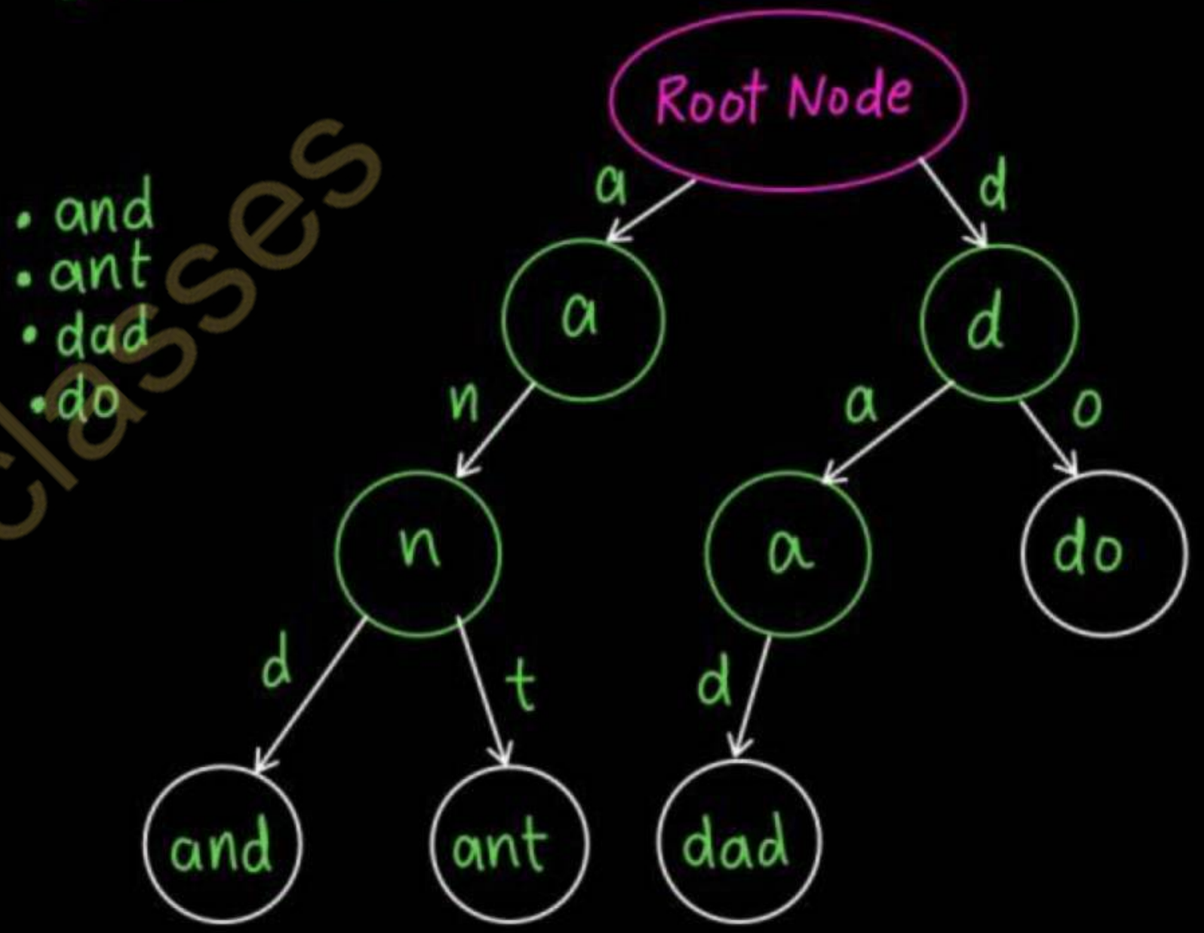
It is also easy to implement and very fast as the search, insert, delete operations have an expected time complexity of O(log(n)), along with O(n) expected space complexity.

What is Trie?

- Trie is a Tree based data structure used for storing a collection of strings and performing efficient search, insert, delete, prefix search and sorted-traversal-of-all operations on them.
- Trie follows a property that If two strings have a common prefix then they will have the same ancestor in the trie. It allows to find all words with a given prefix.
- Tries are generally used in string search applications like auto-search.

Retrieve

Tree Data Structure



- and
- ant
- dad
- do

Properties of a Trie

- Each Trie has an empty root node, with links to other nodes
- Each node represents a string and each edge represents a character.
- Trie can contain any number of characters including alphabets, numbers, and special characters.
- Each path from the root to any node represents a word or string.

Trie has three following types:

- Standard trie
- Compressed trie
- Suffix tree

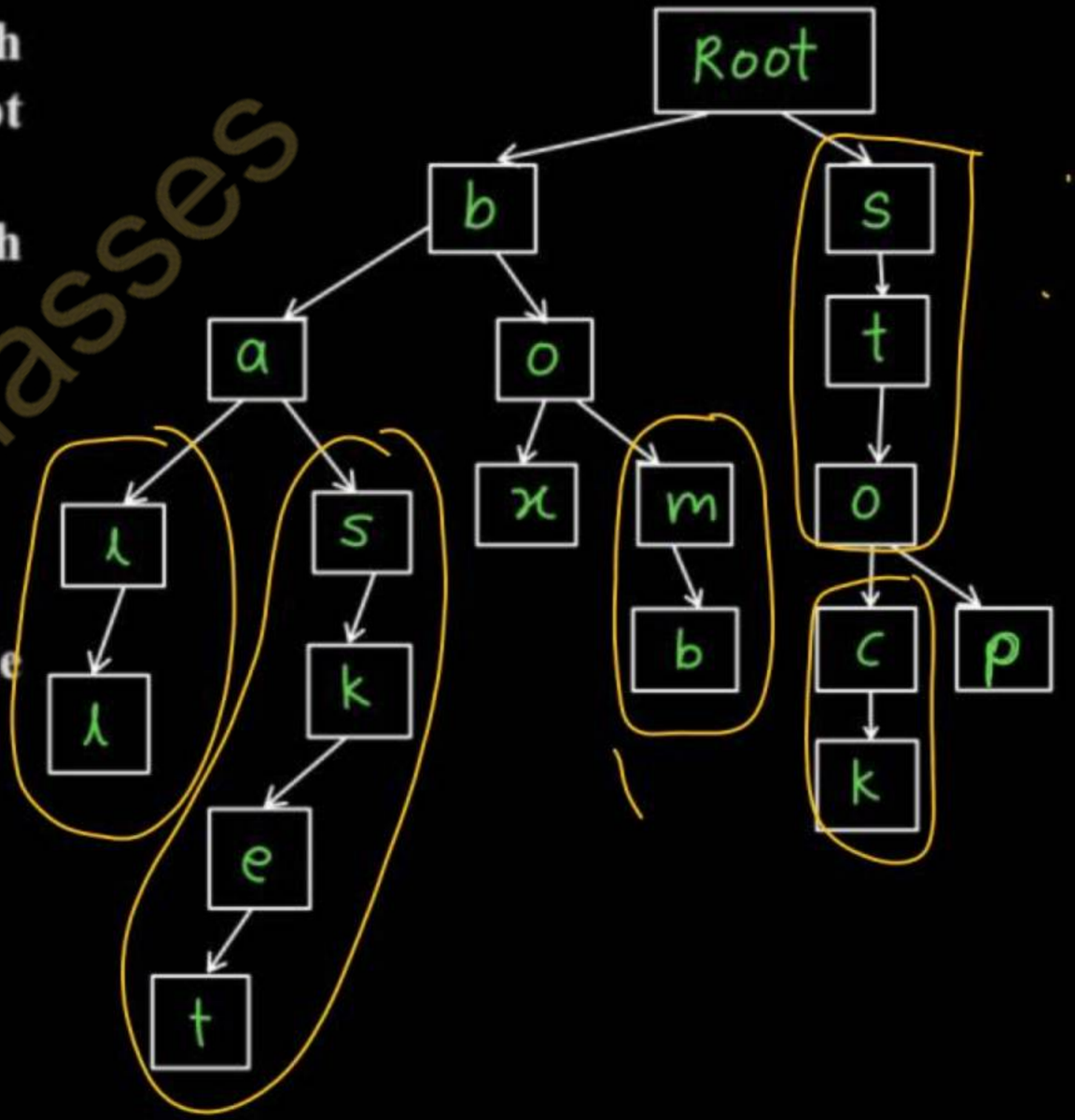
Standard trie

- A Standard trie is an ordered tree in which each node is labeled with a character except for the root node.
- If we want to generate a string, we traverse through the tree's nodes.

Example

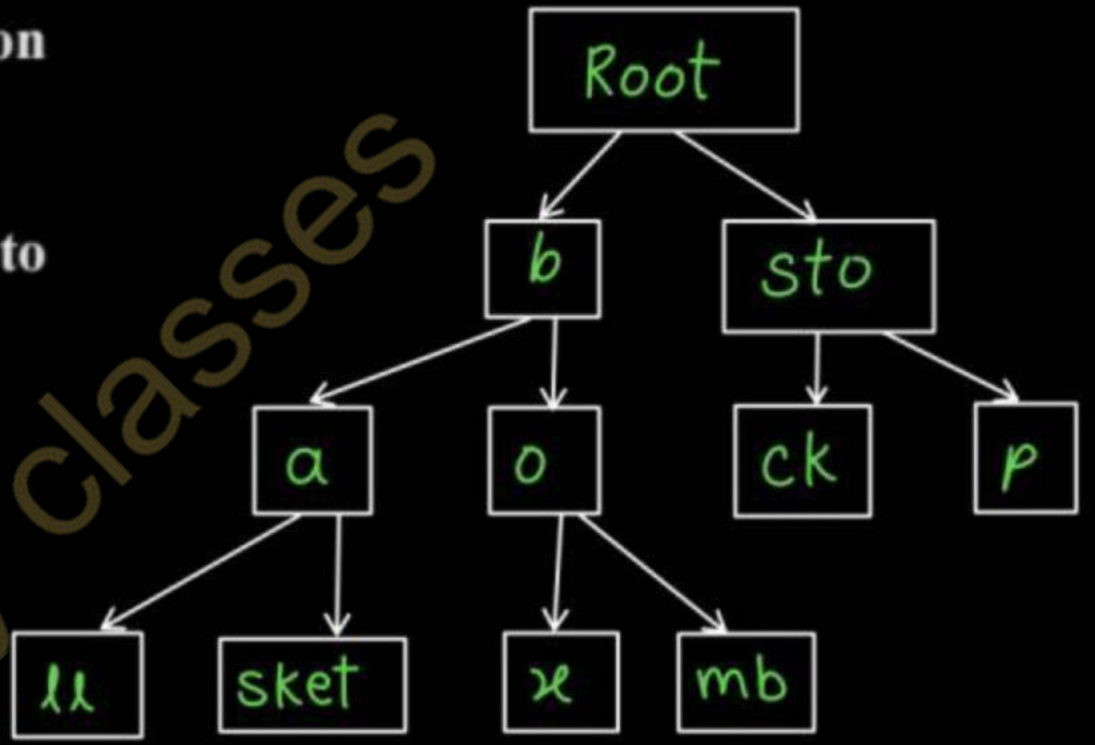
String S = {ball, box, bomb, basket, stock, stop}

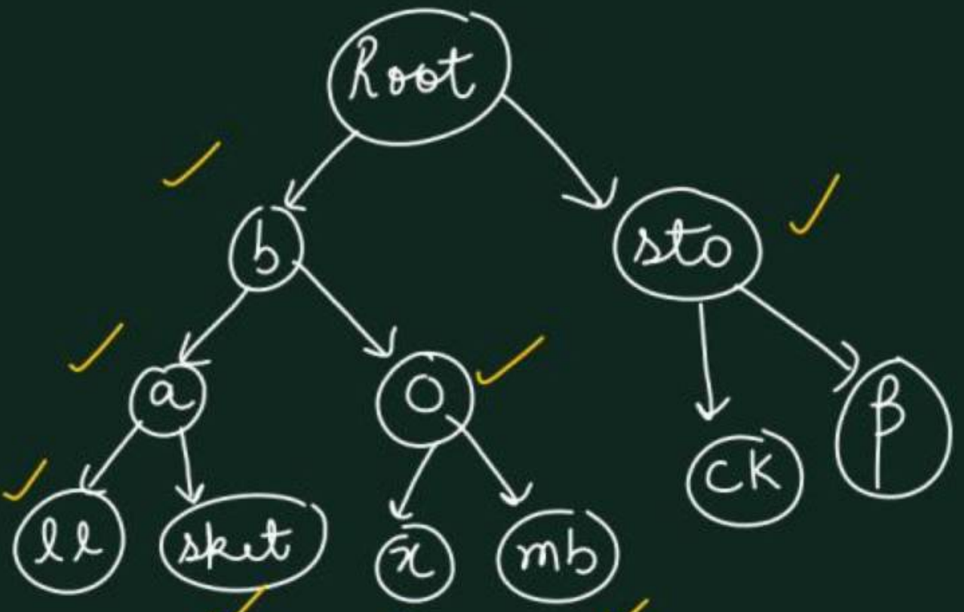
In standard trie, every traversal from the root node represents one of the words in the string S.



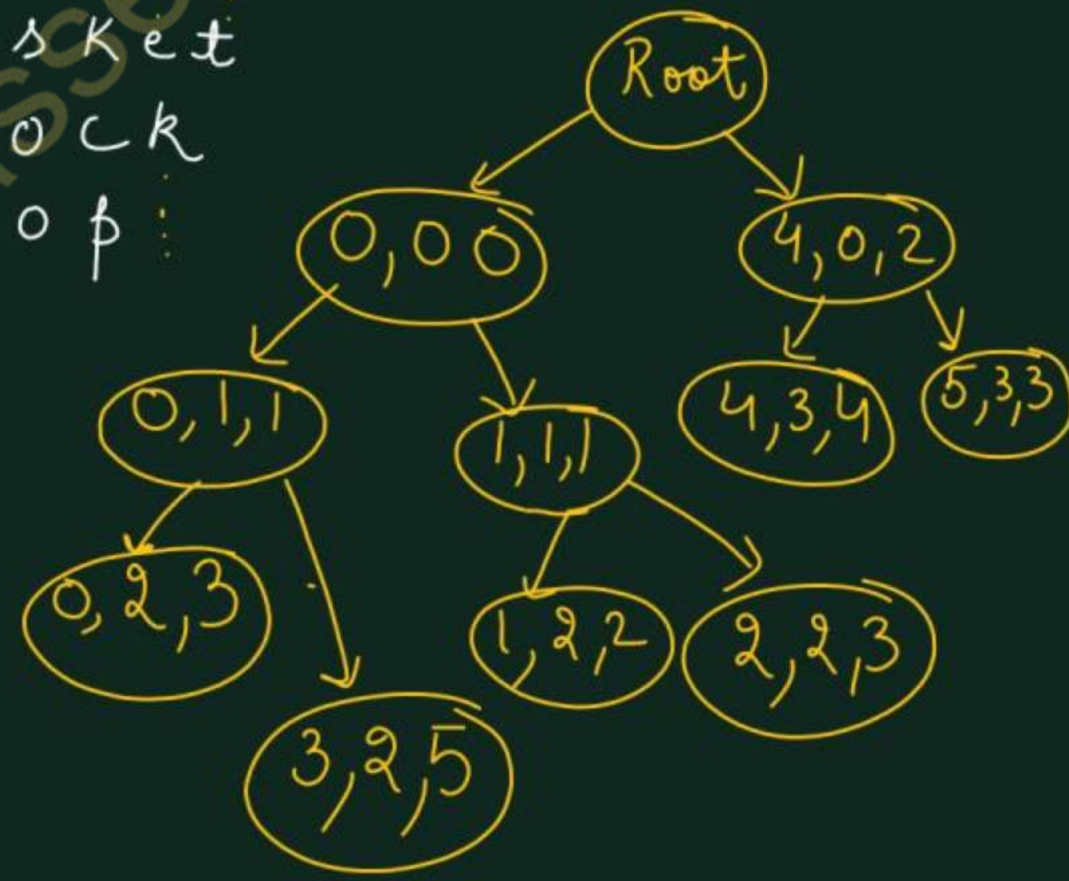
Compressed trie

- A compressed trie is a compact representation of a standard trie.
- If we have a node with one child, we try to merge them to get the easy form of the trie.





	0	1	2	3	4	5
S[0] =	b	a	l	l		
S[1] =	b	o	x			
S[2] =	b	o	m	b		
S[3] =	b	a	s	k	e	t
S[4] =	s	t	o	c	k	
S[5] =	s	t	o	p		



Node(i, j, k)

i → string index in which prefix is present.

j → starting index of the prefix

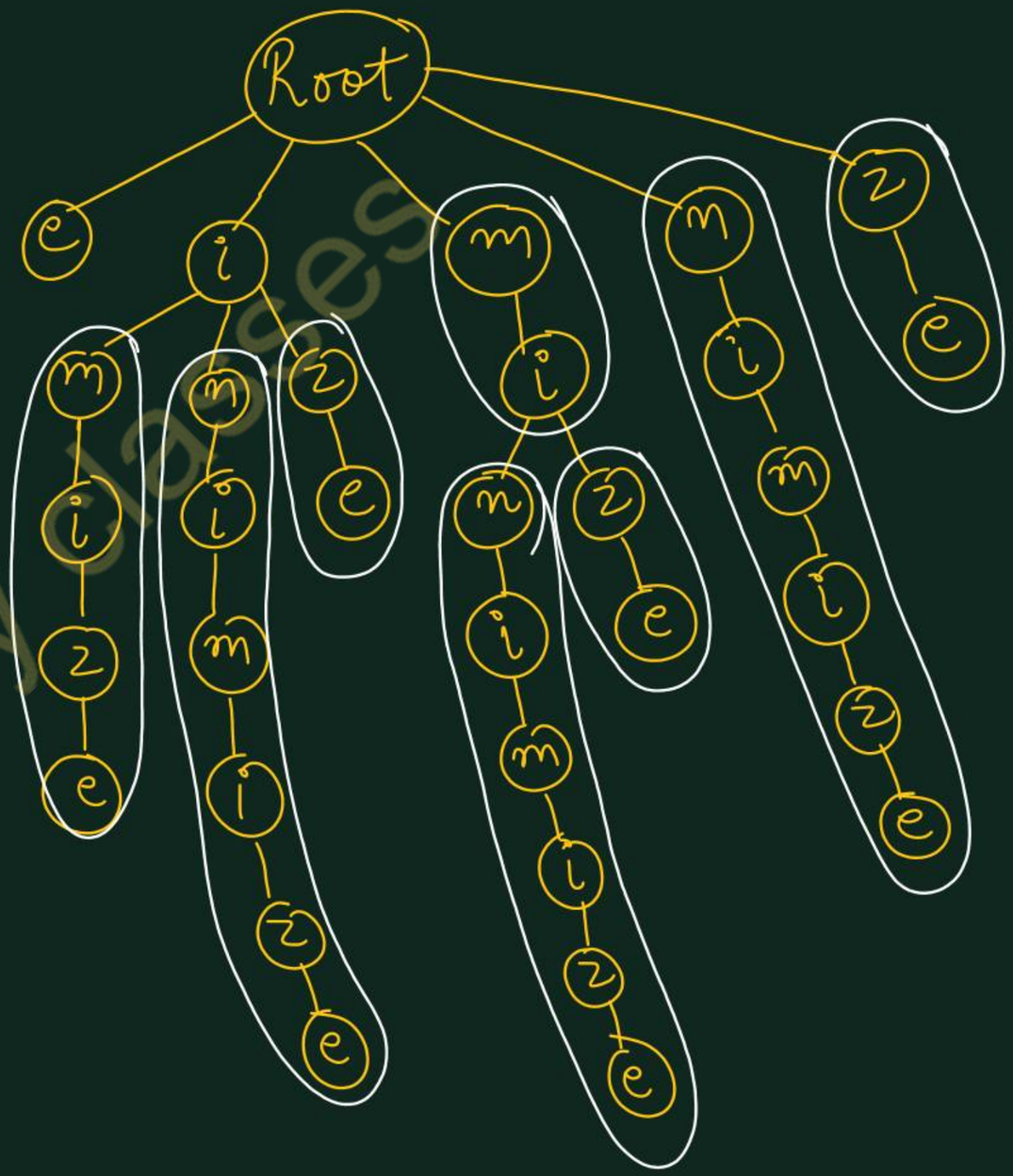
k → ending index of prefix

Suffix Tree

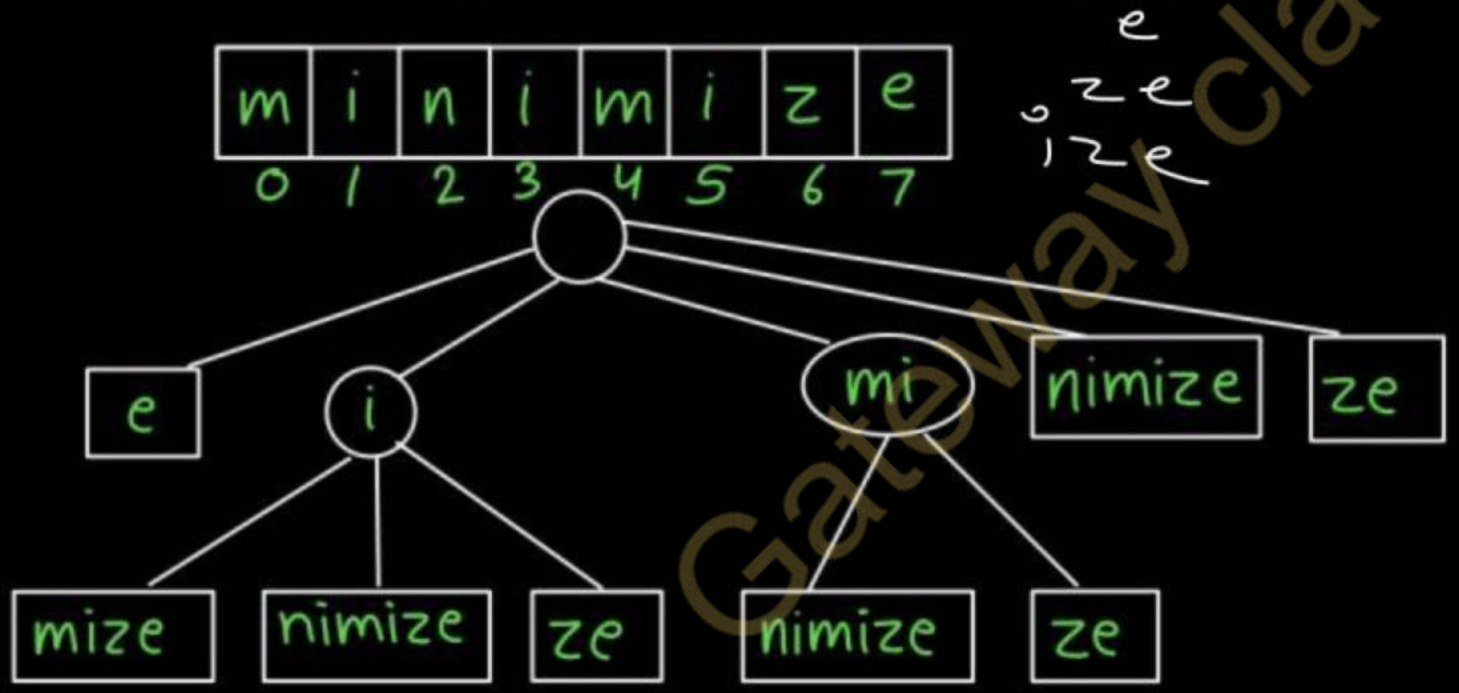
string :- minimize

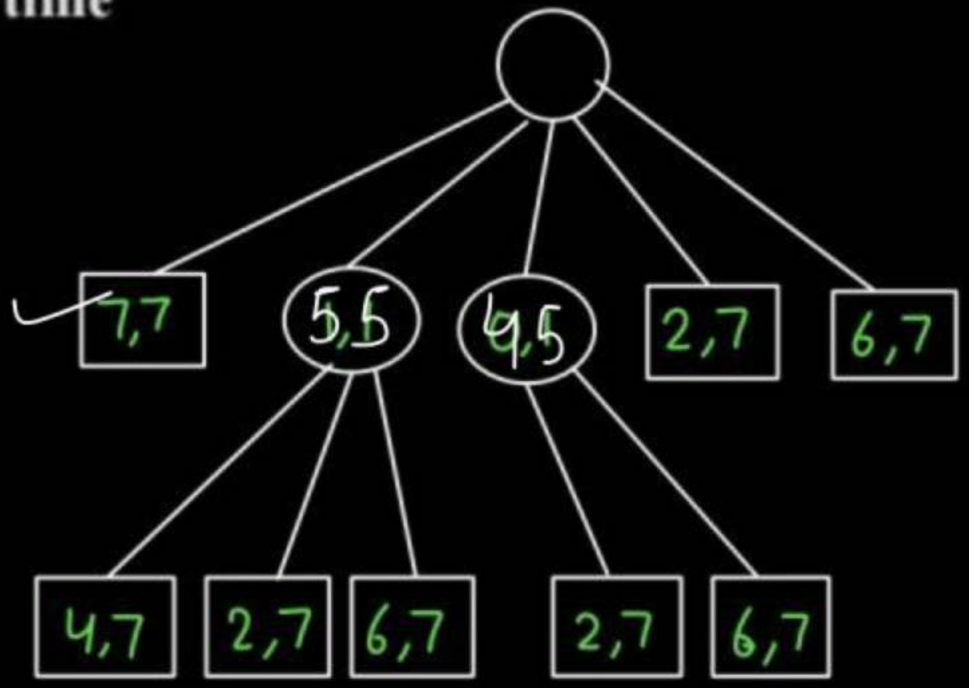
possible suffix

e
ze
ize
mize
imize
nimize
inimize
minimize



- **A suffix trie** is a compressed trie for all the suffixes of a text.
- The suffix trie for a text X of size n from an alphabet of size d
- -stores all the n(n-1)/2 suffixes of X in O(n) space
- Supports arbitrary pattern matching and prefix matching queries in O(dm) time, where m is the length of the pattern.It can be constructed in O(dn) time





AKTU PYQs

1. Discuss Skip list and its operations. (AKTU 2022-23)
2. Explain Skip list in brief. (AKTU 2021-22)
3. What is skip list? Explain the search operation in skip list with example and also write its algorithm.(AKTU 2021-22)