



Tree 2

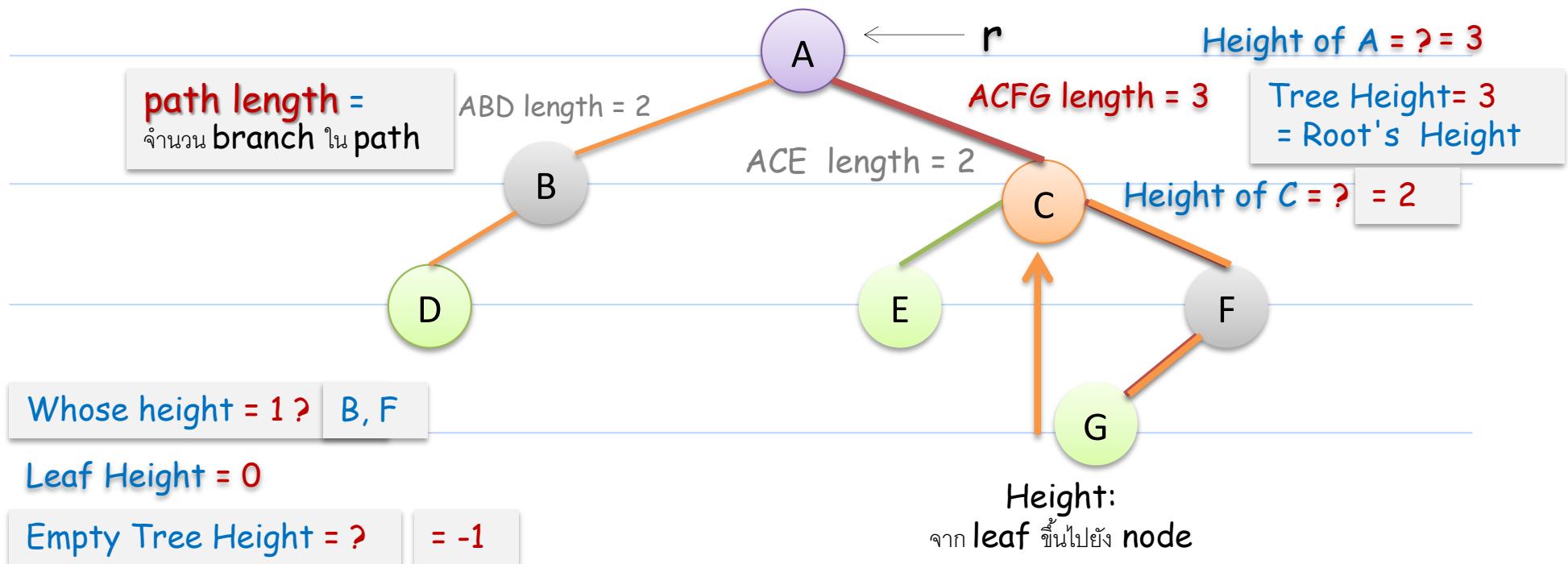
Kiatnarong Tongprasert

1. Tree Definitions
2. Binary Tree
 - Traversals
 - Binary Search Tree
 - Representations
 - Application : Expression Tree
3. AVL Tree
4. Which Representations ?
5. n-ary Tree
6. Generic Tree
7. Multiway Search Tree
8. B-Trees



Height

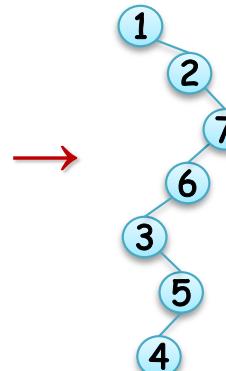
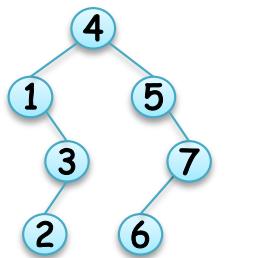
Height of node = longest path length from node to leaf
path ที่ยาวที่สุดจาก node นั้นถึง leaf



Why Balanced Tree ?

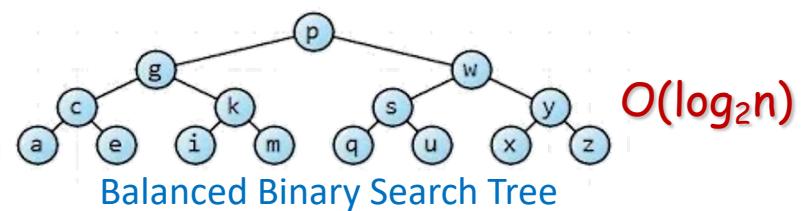
Tree Efficiency (searching, inserting, deleting,...) → $O(\text{Height})$

Binary Search Tree :
บางครั้งมีการเก็บค่าเป็นเล่น



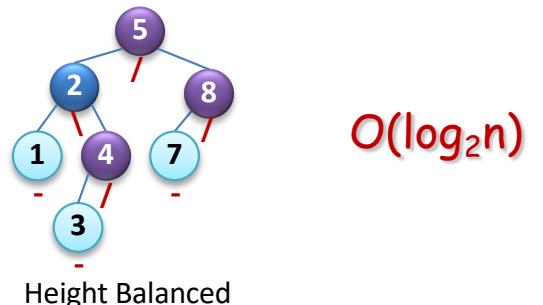
Linear Linked list or near
Linear Search $O(n)$

Binary Search Tree :
(Perfect) Balanced Tree
ทุกใบอยู่ที่ระดับเดียวกัน



Binary Search Tree :
Height Balanced Tree → AVL Maximum height $\leq 1.44 \log_2 n$

ในแต่ละโหนดต้นไม้ย่อยต้าน衡 และขวา มีความสูงต่างกัน ≤ 1



Height Balanced

Balance Factor

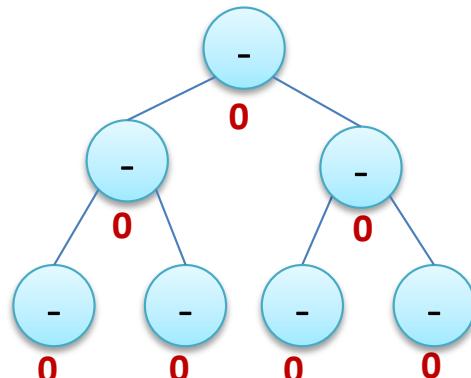
Height Balanced Tree :
(Nearly Balanced)

ในแต่ละโหนดต้นไม้มีอยู่ด้านซ้าย และขวา มีความสูงต่างกัน ≤ 1

Balance factor ทุกโหนดมีค่า 0 หรือ 1 หรือ -1

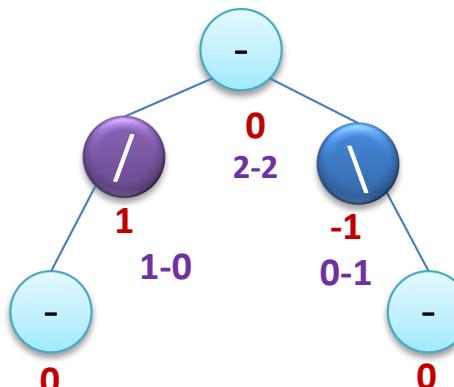
Balance factor ของแต่ละโหนด คือ ค่าความต่างระหว่างต้นไม้มีอยู่ด้านซ้าย และ ต้นไม้มีอยู่ด้านขวา

$$= \text{Height(Left Subtree)} - \text{Height(Right Subtree)}$$

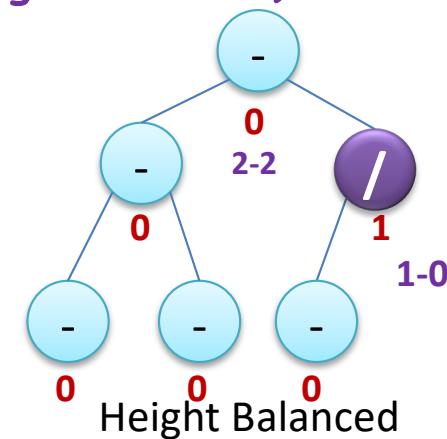


Height Balanced

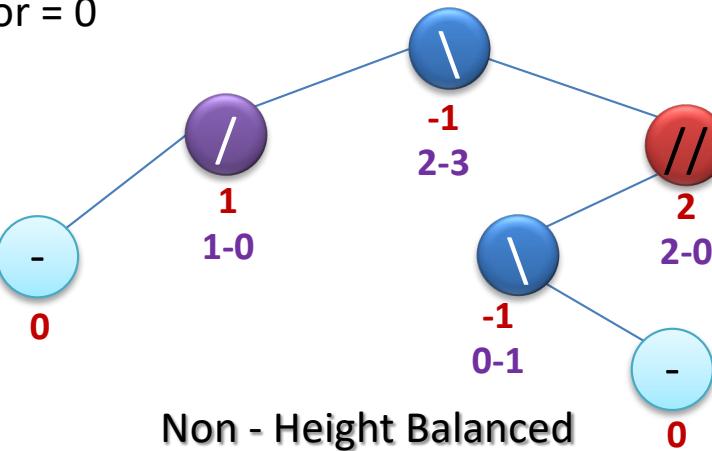
Every node's balance factor = 0



Height Balanced



Height Balanced



Non - Height Balanced

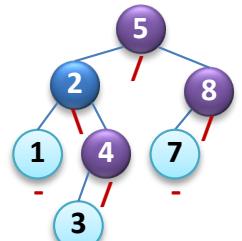
AVL (Height Balanced) Tree

AVLTree Self-rebalancing binary search tree to keep height balance.

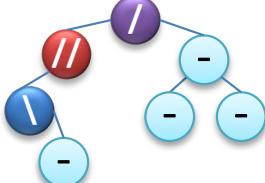
- Maximum height $\leq 1.44 \log_2 n$: ใกล้กับ complete binary tree.
- Search, insertion, deletion (both average & worst cases) $O(\log_2 n)$
- การเพิ่ม และ การลบ อาจต้องทำการหมุน
- Worst case จะไม่มากกว่า binary search tree.



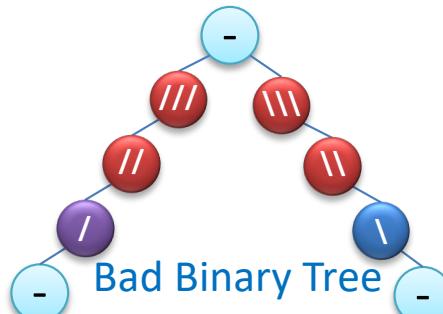
Adelson-Velskii Landis
(1962 Soviet inventors)



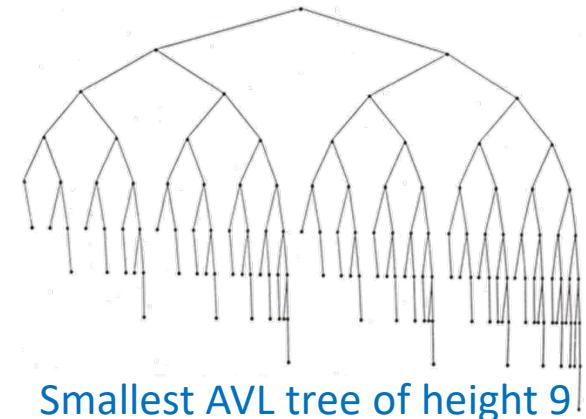
Height Balanced



Non-Height Balanced



Bad Binary Tree
Balance at root
is not enough.



Smallest AVL tree of height 9

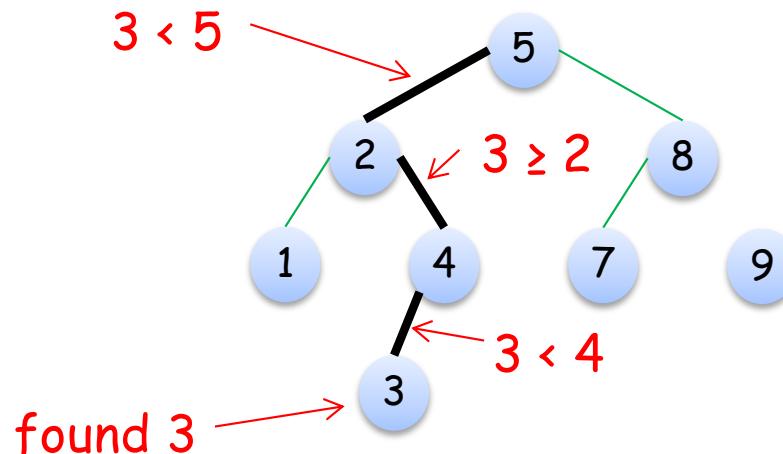
Searching in AVL = Searching in BST

AVL is a binary search tree.

Review Searching a BST (Binary Search Tree).

- เปรียบเทียบ key กับค่าในโหนดโดยเริ่มจาก root
- Follow associated link (recursively).
 - Left link if search key $<$ key in node
 - Right link if search key \geq key in node

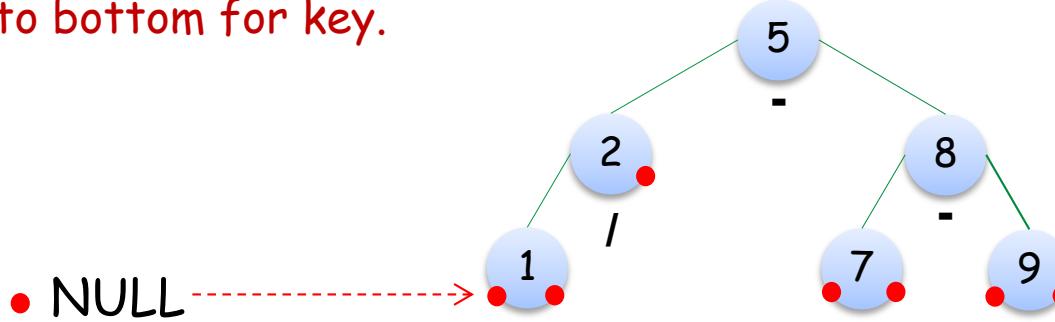
Ex. Search for 3



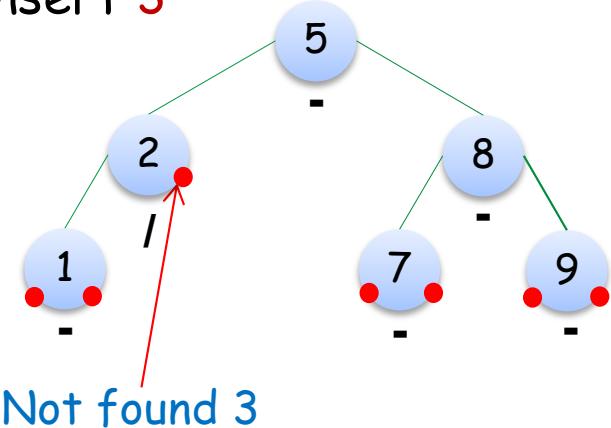
Review Inserting in BST

Review Inserting in BST

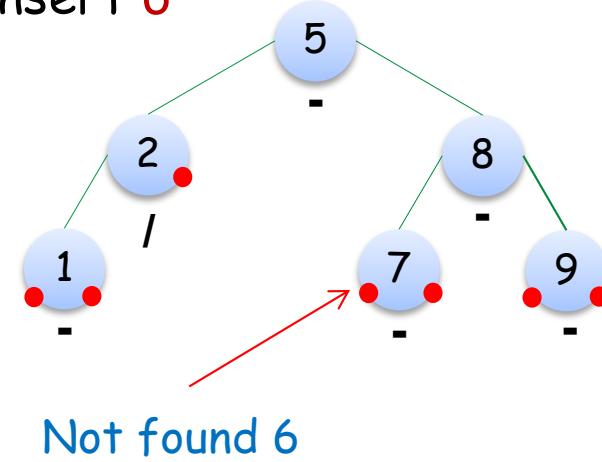
- Search to bottom for key.



Ex. Insert 3



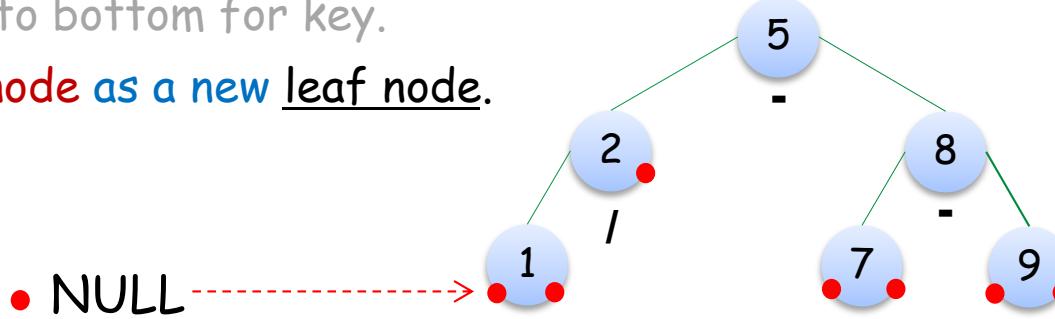
Ex. Insert 6



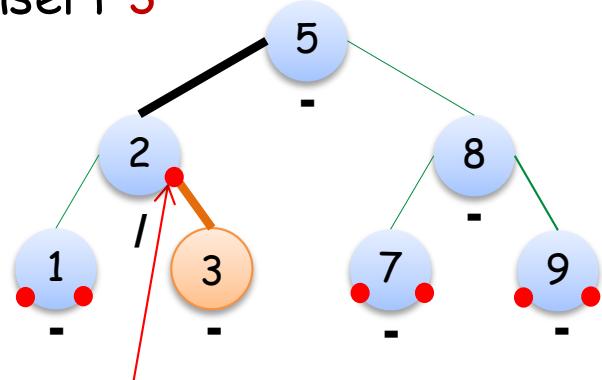
Review Inserting in BST

Review Inserting in BST

- Search to bottom for key.
- Insert node as a new leaf node.

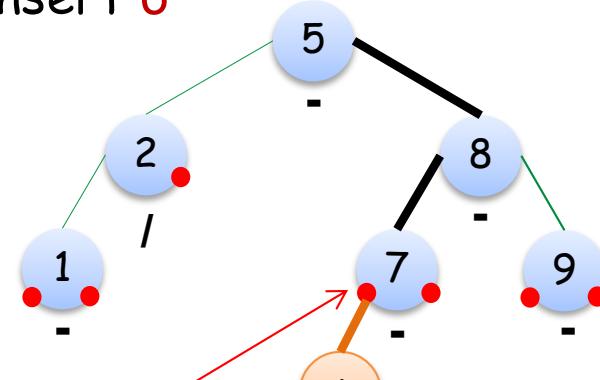


Ex. Insert 3



Not found 3
Insert 3 as a new leaf

Ex. Insert 6



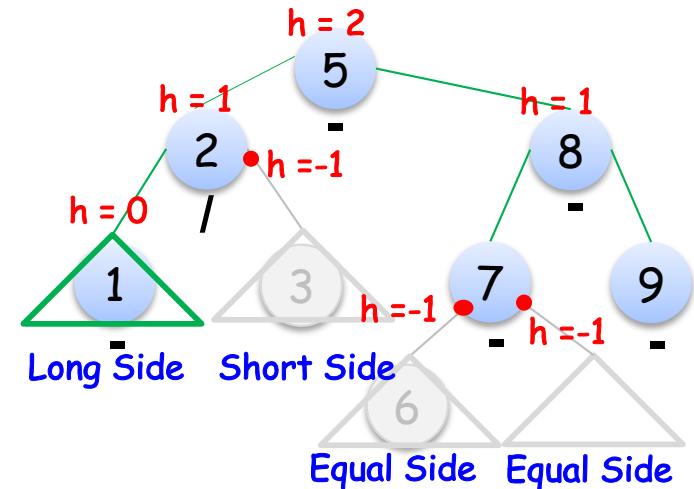
Not found 6
Insert 6 as a new leaf

Inserting in AVL without Rebalancing

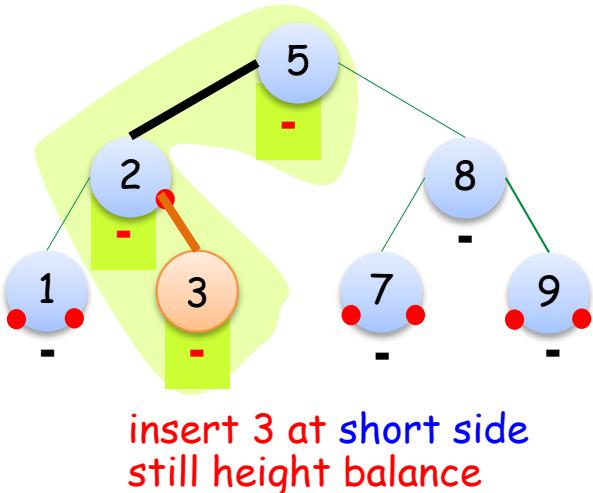
Inserting at short side or equal side

→ AVL's still height balanced

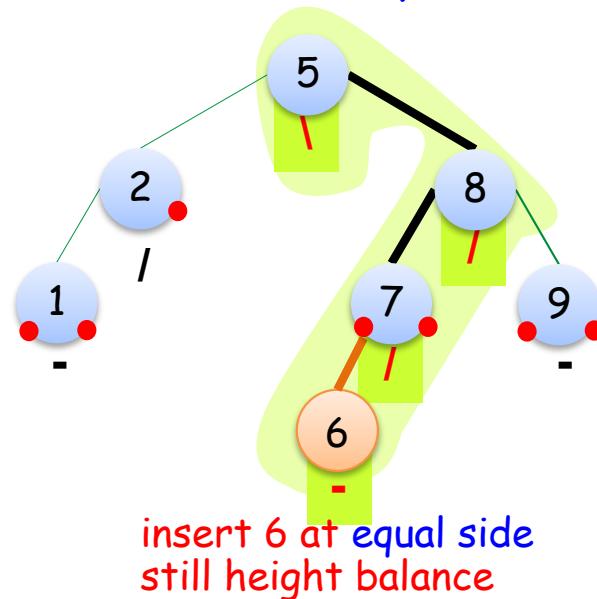
- Insert BST new leaf node.
- การเพิ่มโหนดจะเปลี่ยนความสูงของบางโหนด (เพิ่มขึ้น 1)
จาก root มา�ังโหนดใหม่ และเปลี่ยนค่า balance factors



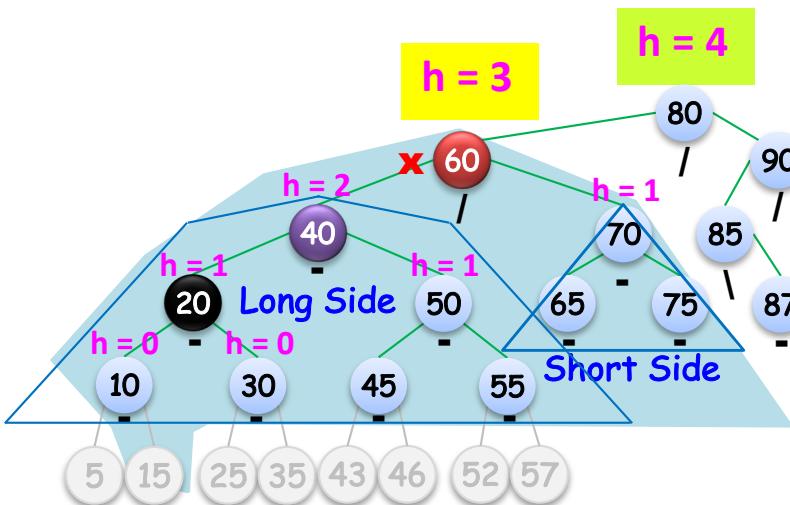
Ex. Insert 3 at short side.



Ex. Insert 6 at equal side.

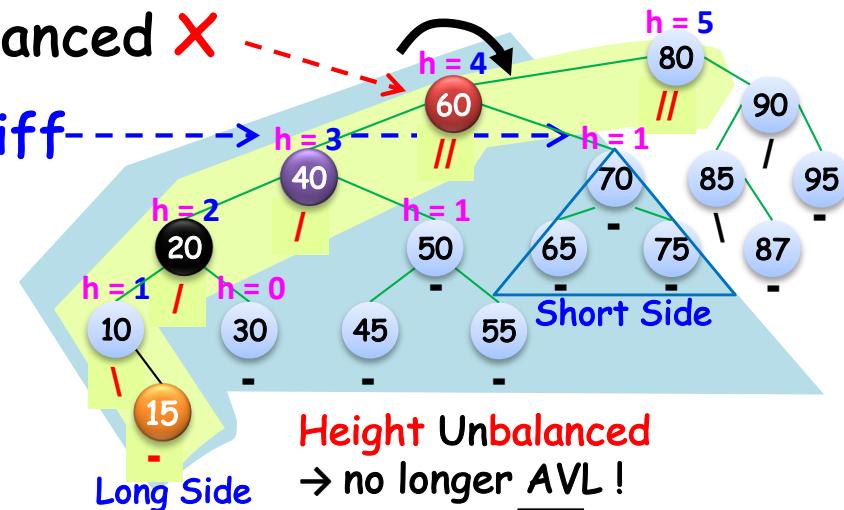


Insertion Re-balancing in AVL



1st unbalanced X
Height diff = 2

Insert 15



Inserting AVL in a long side.

1. ทำให้เกิด Unbalance (height diff = 2)

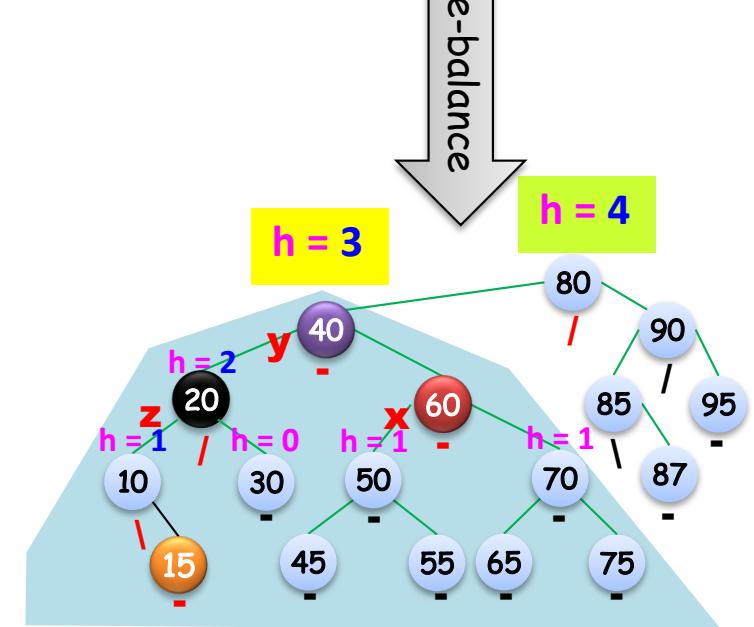
ด้านบนซึ่งไปหา root

- x = 1st unbalanced node

2. ต้องทำ rebalancing

การ rebalancing จะไม่เปลี่ยน:

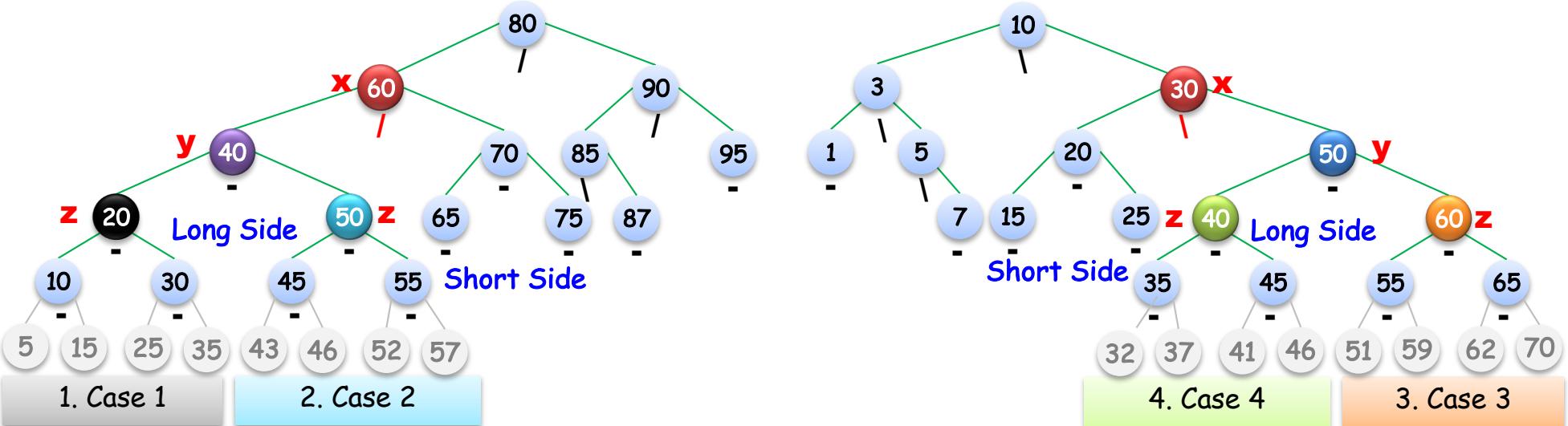
- ความสูงของต้นไม้ยังอยู่
- ความสูงของ root



Height balanced again → AVL

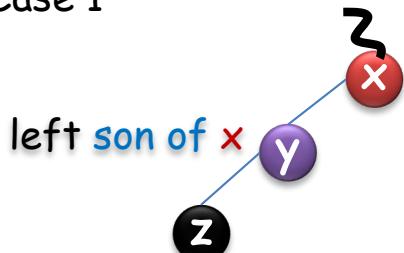
4 Insertions Re-balancing in AVL

Inserting AVL in a long side : ให้ $x = \text{node}$ และที่ไม่ balance , ตาม path ที่ insert ให้ y เป็นลูกของ x และ z เป็นลูกของ y

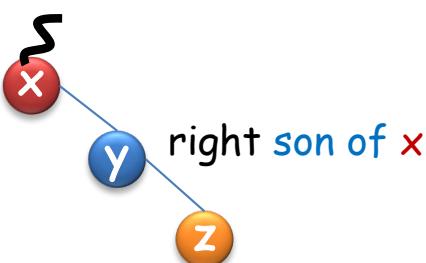


มี 4 กรณี เมื่อ insert ที่

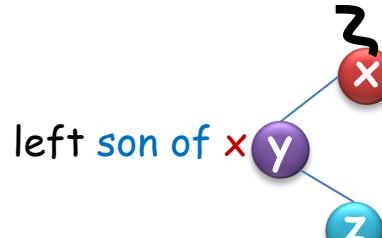
1. Case 1



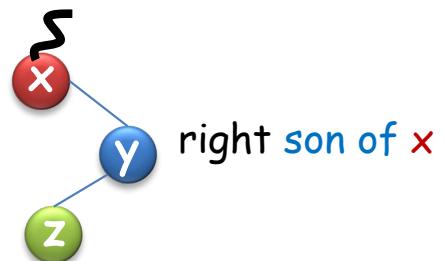
3. Case 3



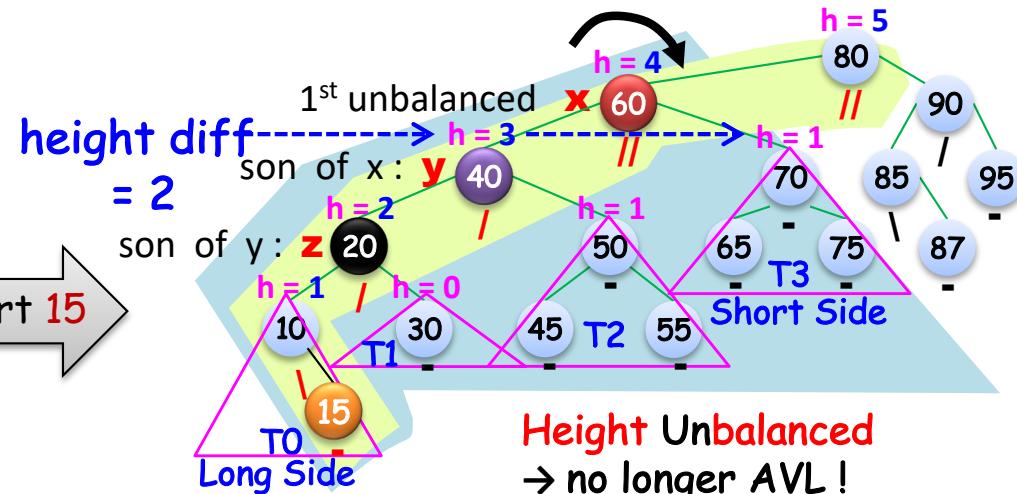
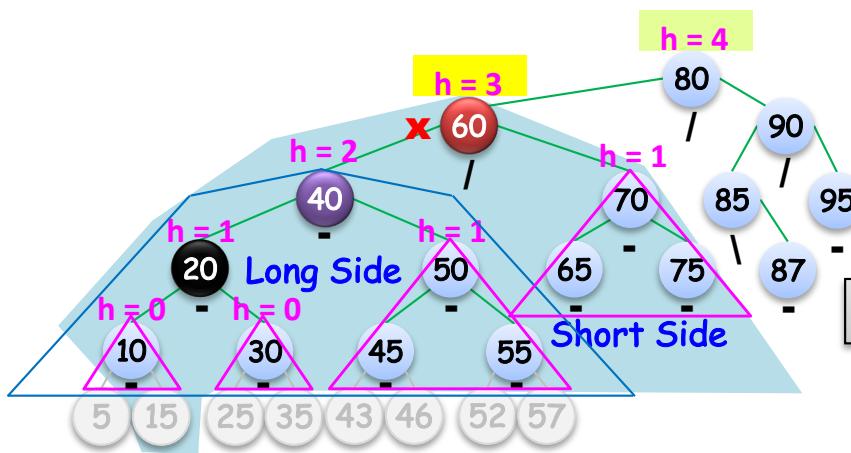
2. Case 2



4. Case 4



Rebalancing Left Subtree of Left Son of \times (1st Unbalanced Node)



Operation

1. Left rotate at x

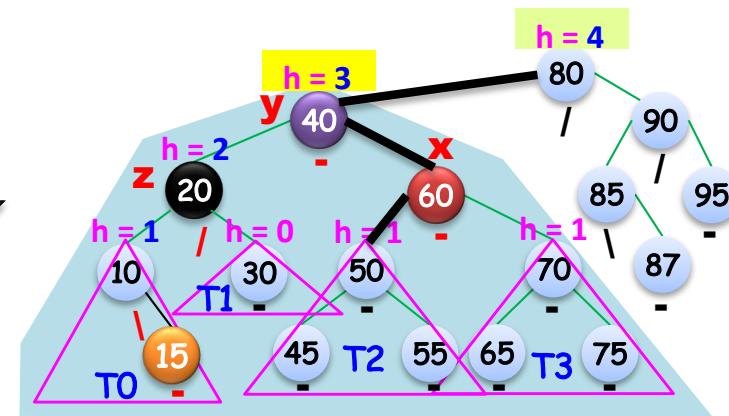
- y ขึ้นไปแทนที่ x
- x ลงมาเป็นลูกขวาของ y

2. Re-arrange T0, T1, T2, T3 to preserve the properties of BST.

- $T_0 \leq z \leq T_1 \leq y \leq T_2 \leq x \leq T_3$

3. Fixing some part of the shaded subtree's balance factors.

4. Height of the blue shaded subtree doesn't change -> also the whole AVL's.

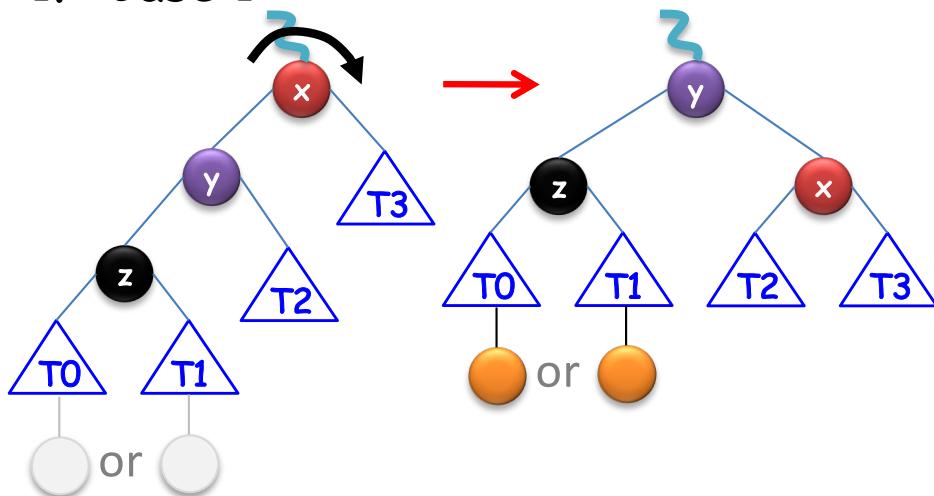


Height balanced again → AVL

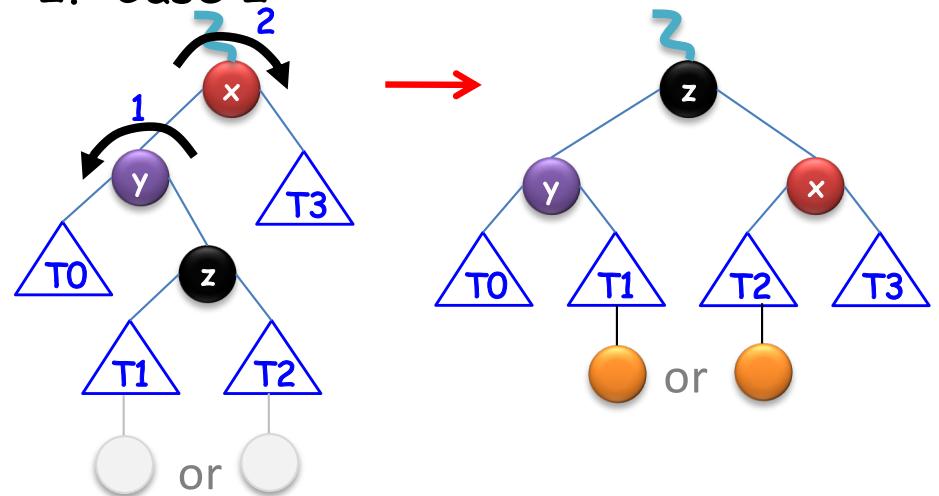
4 Kinds of Rebalancing in AVL

x = 1st unbalanced node, y = son of x , z = son of y

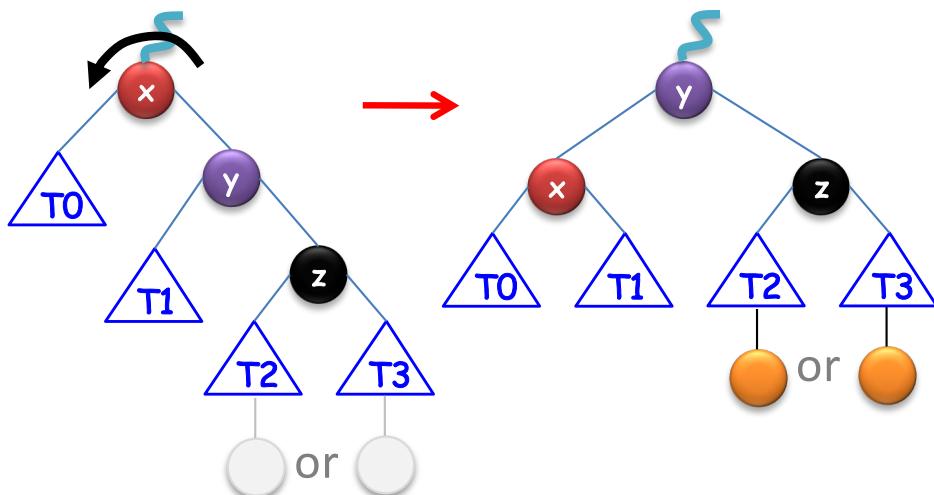
1. Case 1



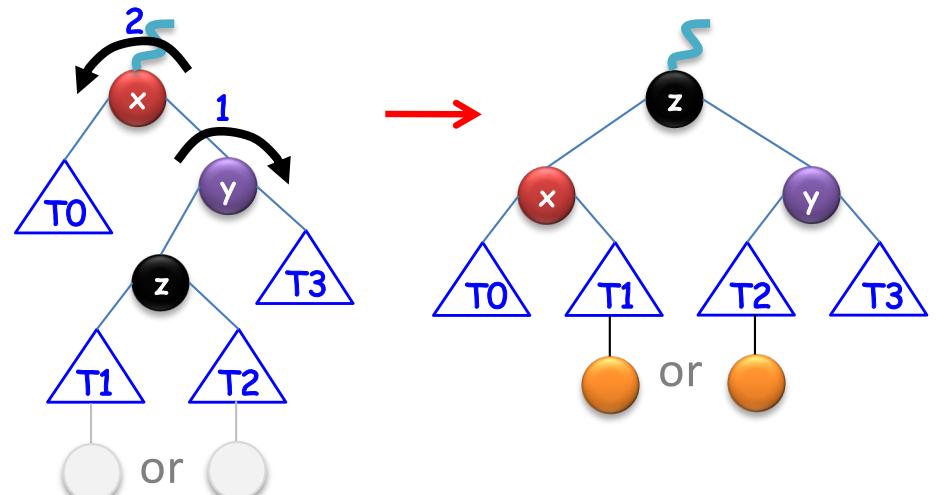
2. Case 2



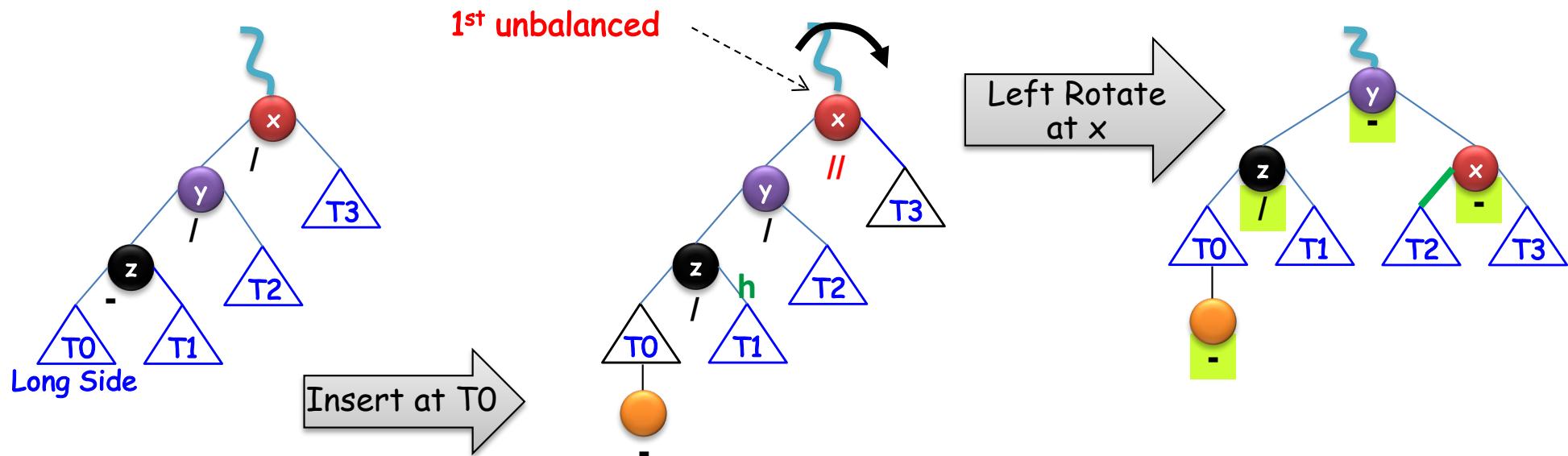
3. Case 3



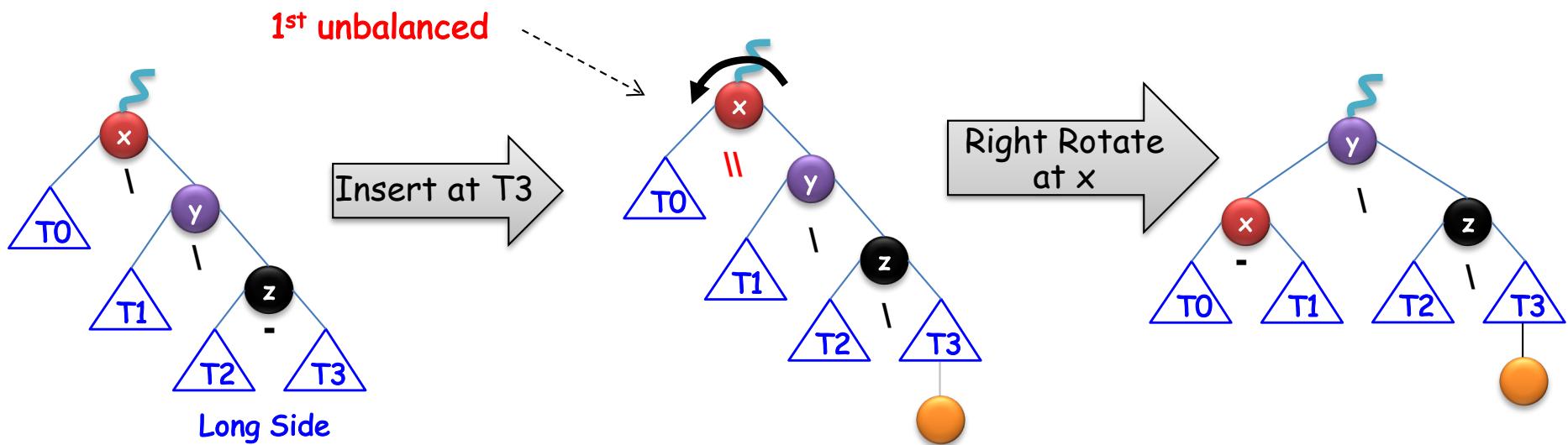
4. Case 4



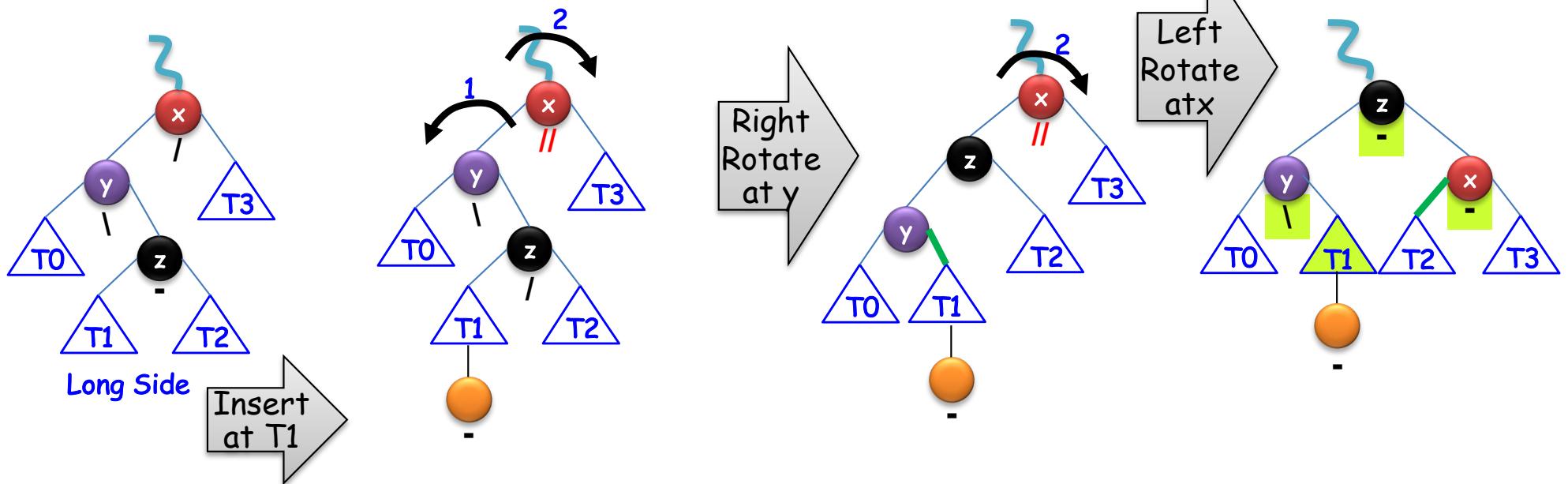
Case 1



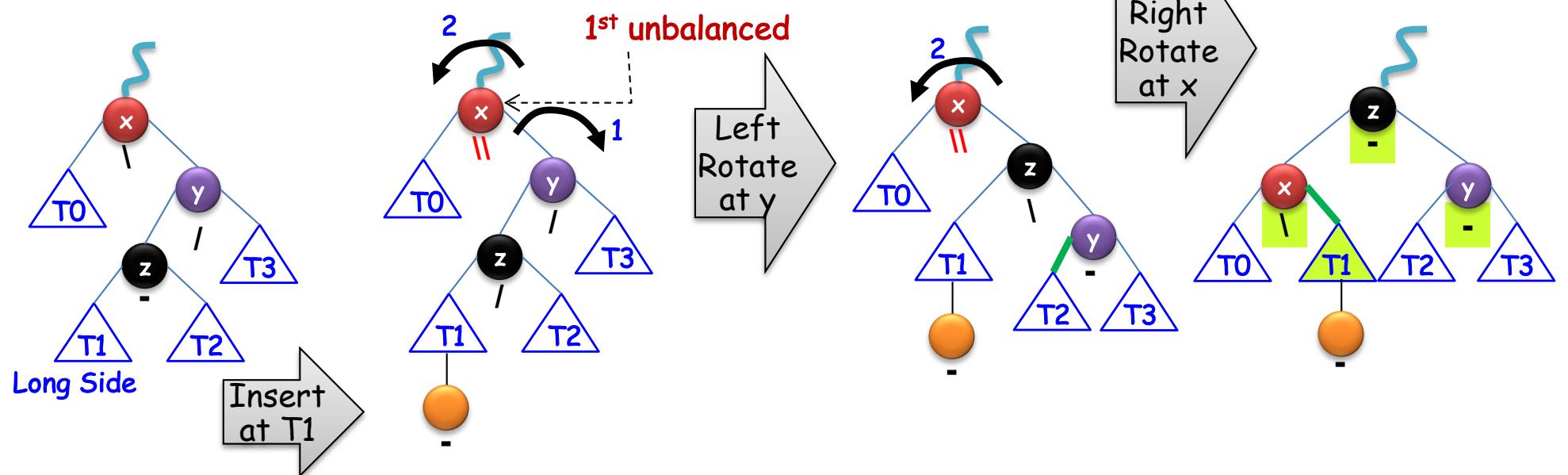
Case 3



Case 2



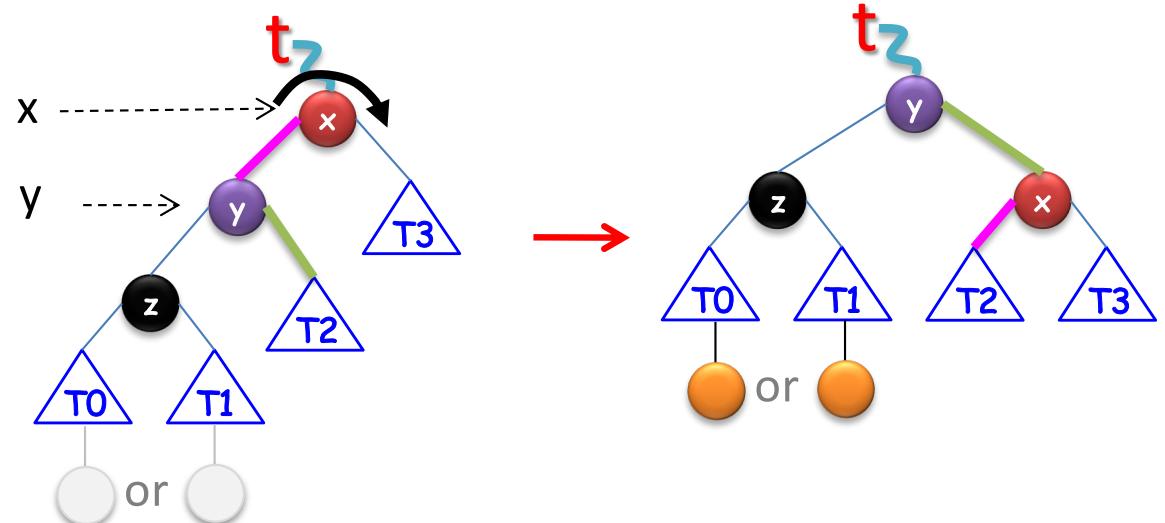
Case 4



Single Right Rotation & Single Left Rotation Algorithms

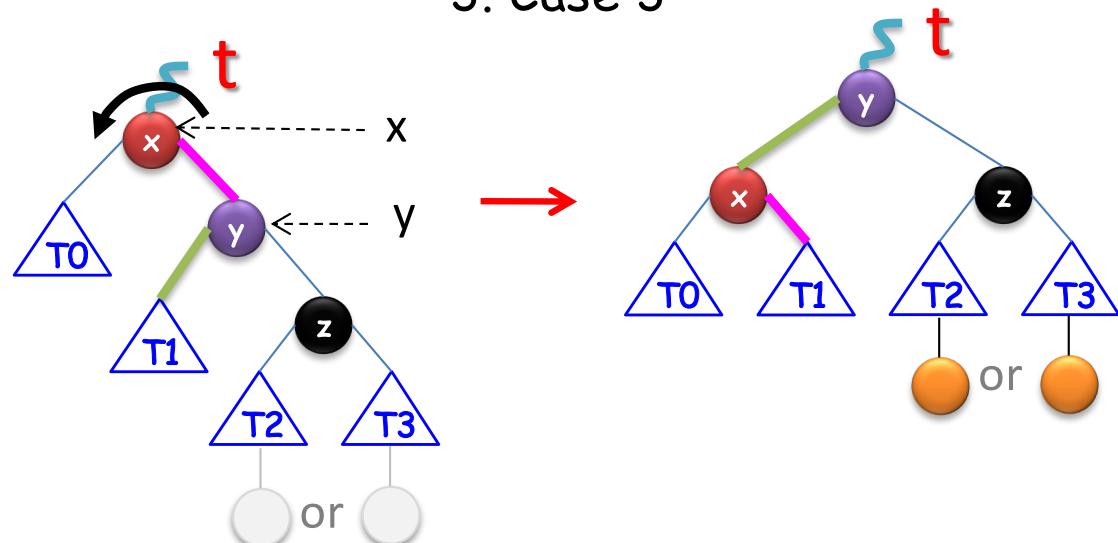
```
// Single Left Rotation  
leftRotate(x)  
y = x.left  
x.left = y.right  
y.right = x  
return y
```

1. Case 1



```
// Single Right Rotation  
rightRotate(x)  
y = x.right  
x.right= y.left  
y.left = px  
return y
```

3. Case 3



Double Rotations Algorithms

```
// Double Rotation :
```

```
// Case 2 :
```

```
doubleRotation(x)
```

```
    x.left = rightRotate(x.left)
```

```
    x = leftRotate(x)
```

```
    return x
```

```
// Case 4 :
```

```
doubleRotation (x)
```

```
    x.right = leftRotate(x.right)
```

```
    x = rightRotate(x)
```

```
    return x
```

```
//Rebalance()
```

```
balance = x.balanceValue
```

```
if balance == -2 :
```

```
    if x.right.balanceValue == 1 :
```

```
        x.right = leftRotage(x.right)
```

```
        x = rightRotage(x)
```

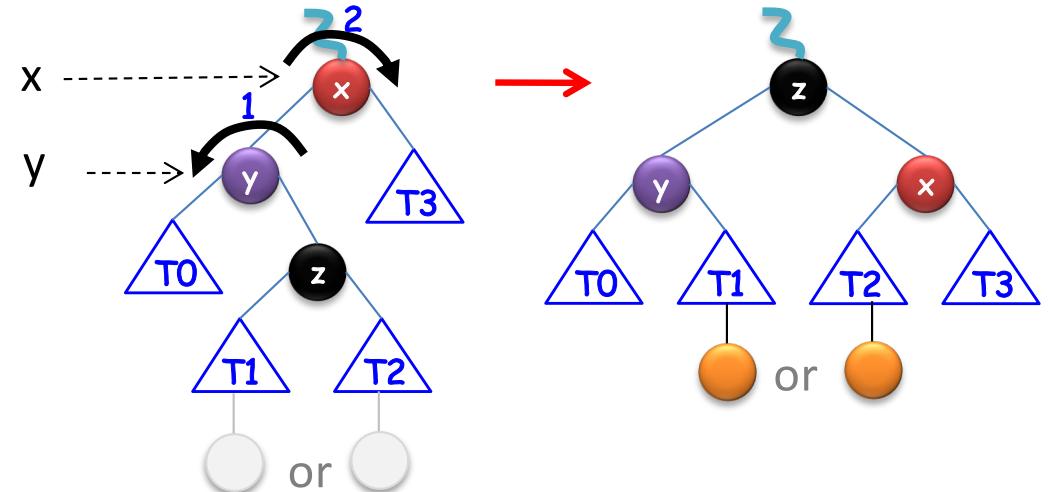
```
elif balance == 2 :
```

```
    if x.left.balanceValue == -1:
```

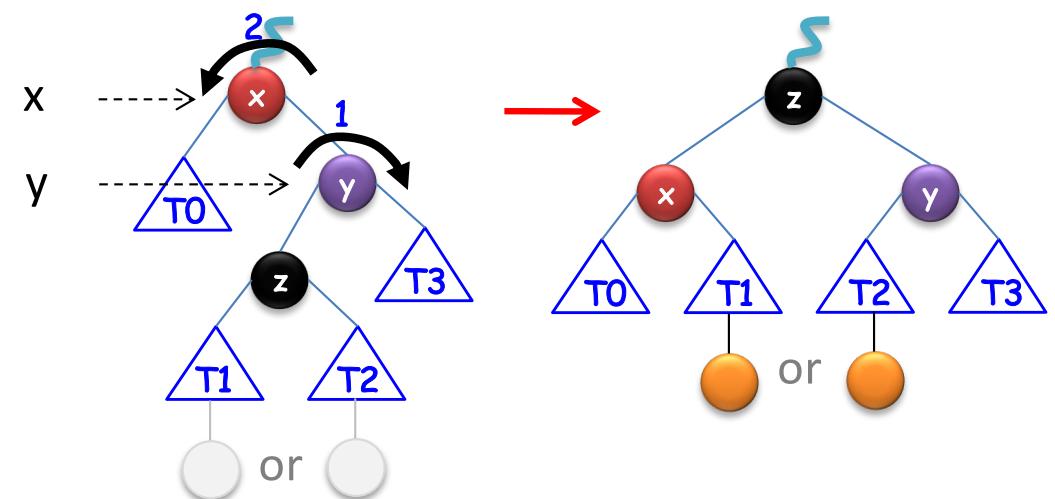
```
        x.left = rightRotage(x.left)
```

```
        x = leftRotage(x)
```

2. Case 2



4. Case 4



Tree 2

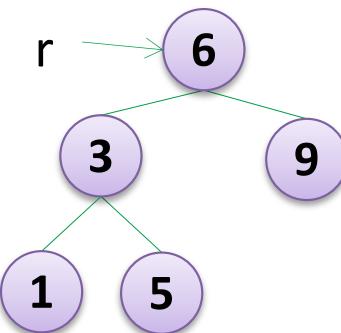
- 3. AVL Tree
Height Balanced Tree
- 4. Which Representations ?
- 5. n-ary Tree
- 6. Generic Tree
- 7. Multiway Search Tree
- 8. Top Down Tree
- 9. B-Tree



- 1. Tree Definitions
- 2. Binary Tree
 - Traversals
 - Binary Search Tree
 - Representations
 - Application : Expression Tree

Binary Tree Representations

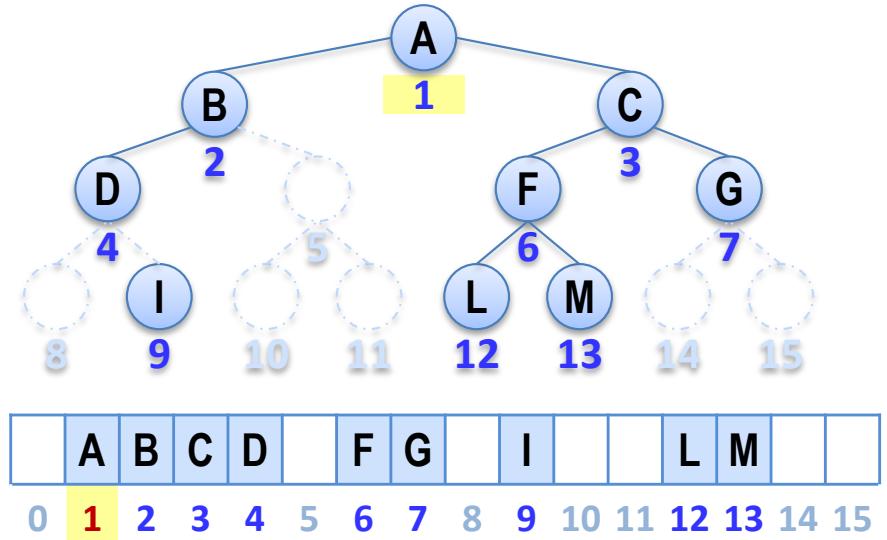
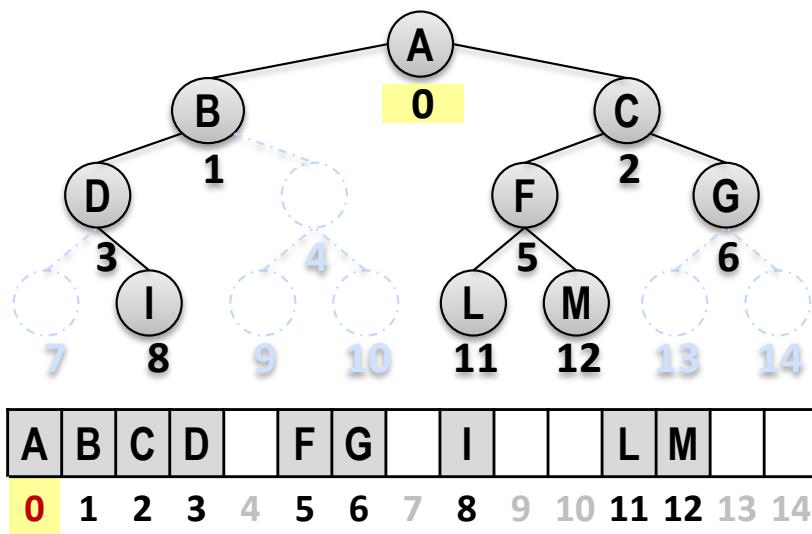
1. Dynamic



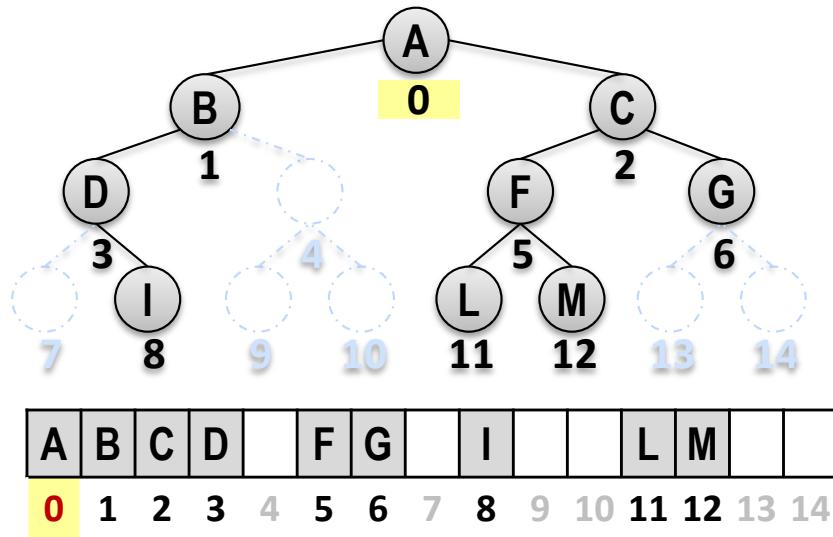
class node:

```
def __init__(self, data, left = None, right = None):  
    self.data = data  
    self.left = left if left is not None else None  
    self.right = right if right is not None else None
```

2. Sequential (Implicit) Array

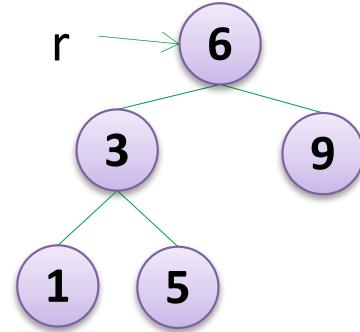


Sequential Array (Implicit Array)



- ง่าย
- ต้องกะขนาด array
- เพิ่มขึ้นอีก 1 level => array มีขนาด เพิ่มมากกว่าที่มืออยู่เดิม 1 เท่า (คิดเต็มที่ full binary tree)
- ถ้ากะขนาด array ถูก และ tree ใช้พื้นที่ส่วนใหญ่ของ array เช่น almost complete binary tree จะ save space เพราะไม่ต้องมีฟิลด์ left, right, father

Dynamic



- จำนวน node ถูกจำกัดโดย memory

Tree 2

- 3. AVL Tree
Height Balanced Tree
- 4. Which Representations ?
- 5. n-ary Tree
- 6. Generic Tree
- 7. Multiway Search Tree
- 8. Top Down Tree
- 9. B-Tree

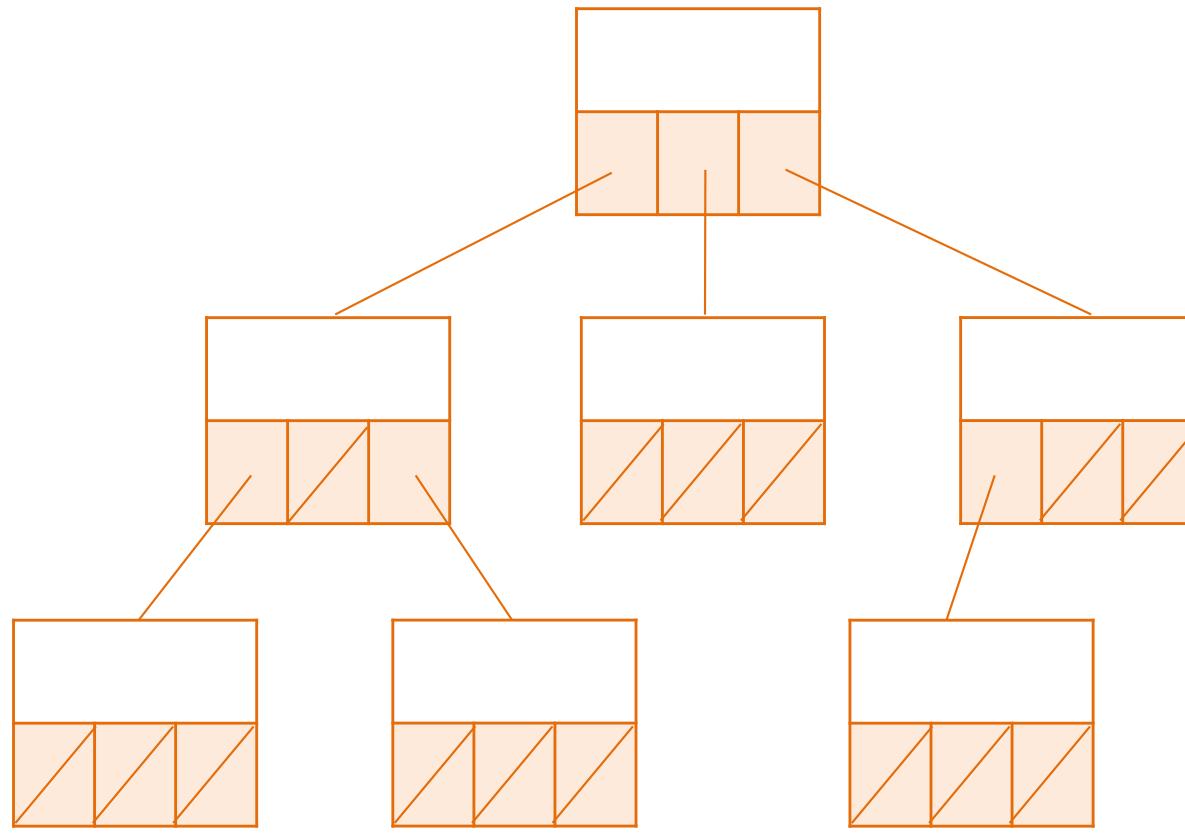


- 1. Tree Definitions
- 2. Binary Tree
 - Traversals
 - Binary Search Tree
 - Representations
 - Application : Expression Tree

n -ary (multiway order n) Trees

bi-nary Tree แต่ละ node มีลูกได้อย่างมากที่สุด 2 ตัว

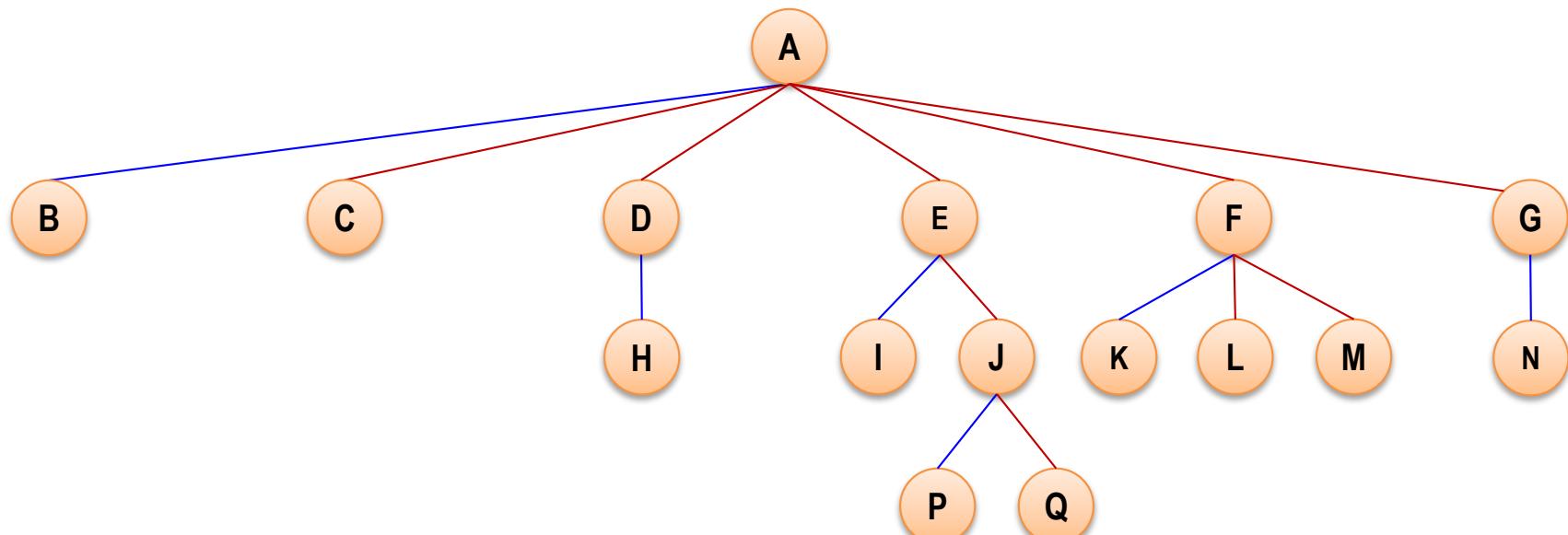
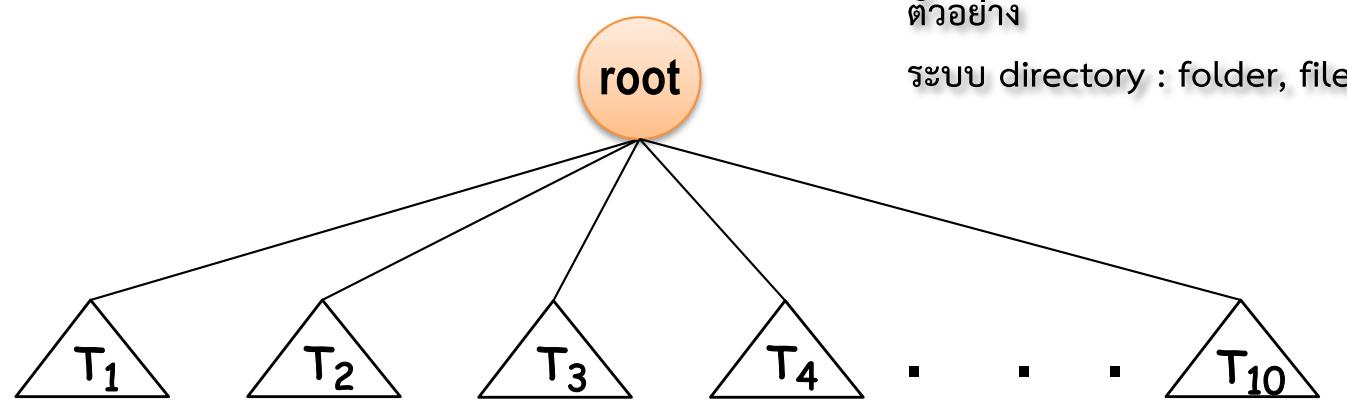
n -ary Tree แต่ละ node มีลูกได้อย่างมากที่สุด n ตัว



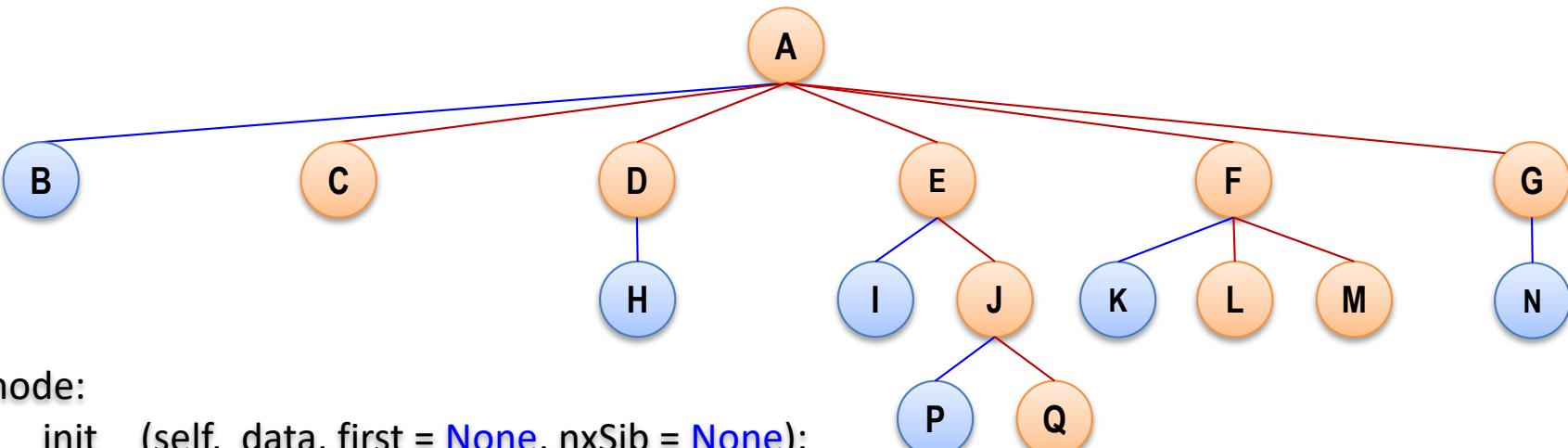
Ternary Tree $n = 3$

Generic Tree

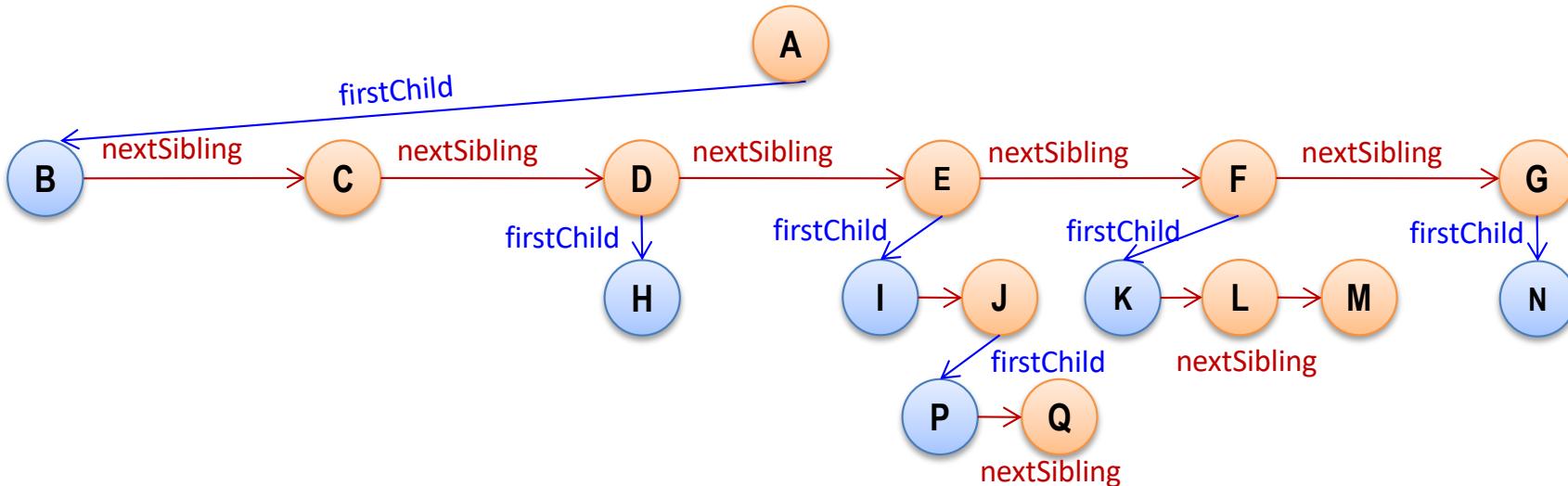
Generic Tree : แต่ละ node มีลูกได้ไม่จำกัดจำนวน



Implementation of Generic Tree



```
class node:  
    def __init__(self, data, first = None, nxSib = None):  
        self.data = data  
        self.firstChild = None if first is None else first  
        self.nextSibling = None if nxSib is None else nxSib
```



Tree 2

- 3. AVL Tree
Height Balanced Tree
- 4. Which Representations ?
- 5. n-ary Tree
- 6. Generic Tree
- 7. Multiway Search Tree
- 8. Top Down Tree
- 9. BalancedTree
B-Tree



- 1. Tree Definitions
- 2. Binary Tree
 - Traversals
 - Binary Search Tree
 - Representations
 - Application : Expression Tree

Multiway Search Tree

Multiway Tree order n : มีลูกได้มากที่สุด n ตัว (0, 1,... or n subtrees) (#max. sons = n)

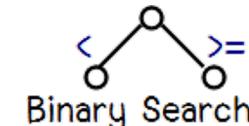
For order n :

- #max. sons = n
- #max. keys = n-1

Multiway Search Tree : for every key K

left descendants $\leq K$

right descendants $> K$



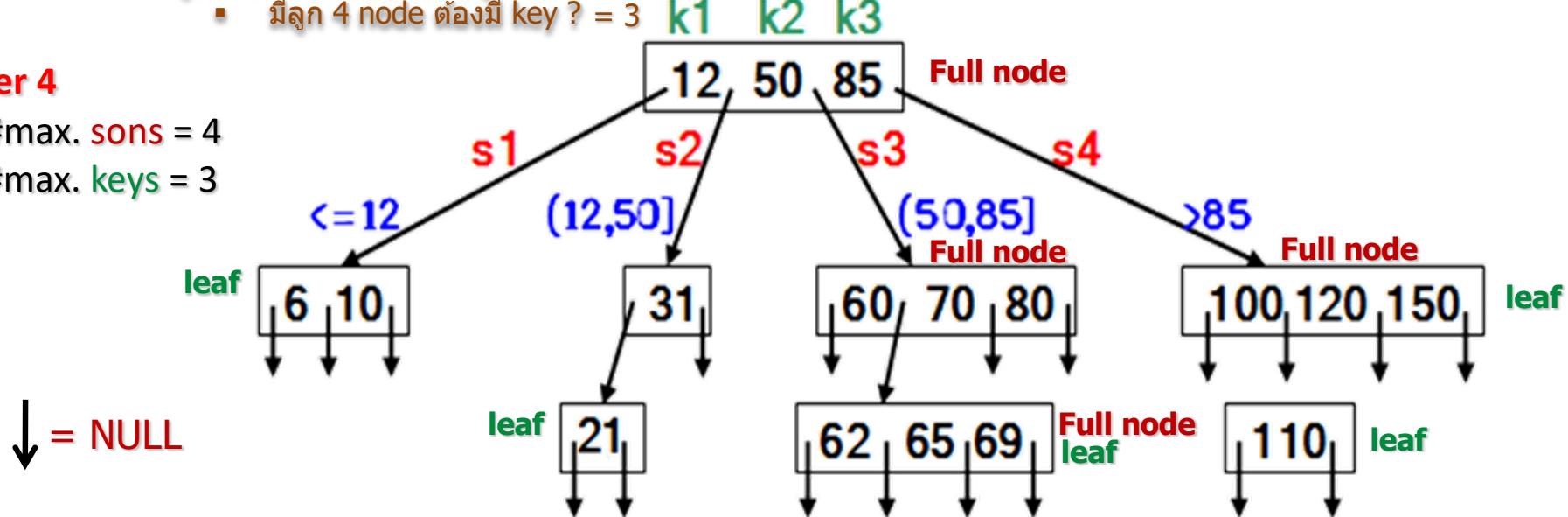
leaf : node ที่ไม่มีลูกเลย

Full node : node ที่มีจำนวน key เดิมที่

รูปตัวอย่าง root node มีลูก 4 ตัว
▪ มีลูก 4 node ต้องมี key ? = 3

Order 4

- #max. sons = 4
- #max. keys = 3



Tree 2

- 3. AVL Tree
Height Balanced Tree
- 4. Which Representations ?
- 5. n-ary Tree
- 6. Generic Tree
- 7. Multiway Search Tree
- 8. Top Down Tree
- 9. BalancedTree
B-Tree



- 1. Tree Definitions
- 2. Binary Tree
 - Traversals
 - Binary Search Tree
 - Representations
 - Application : Expression Tree

Top Down Tree

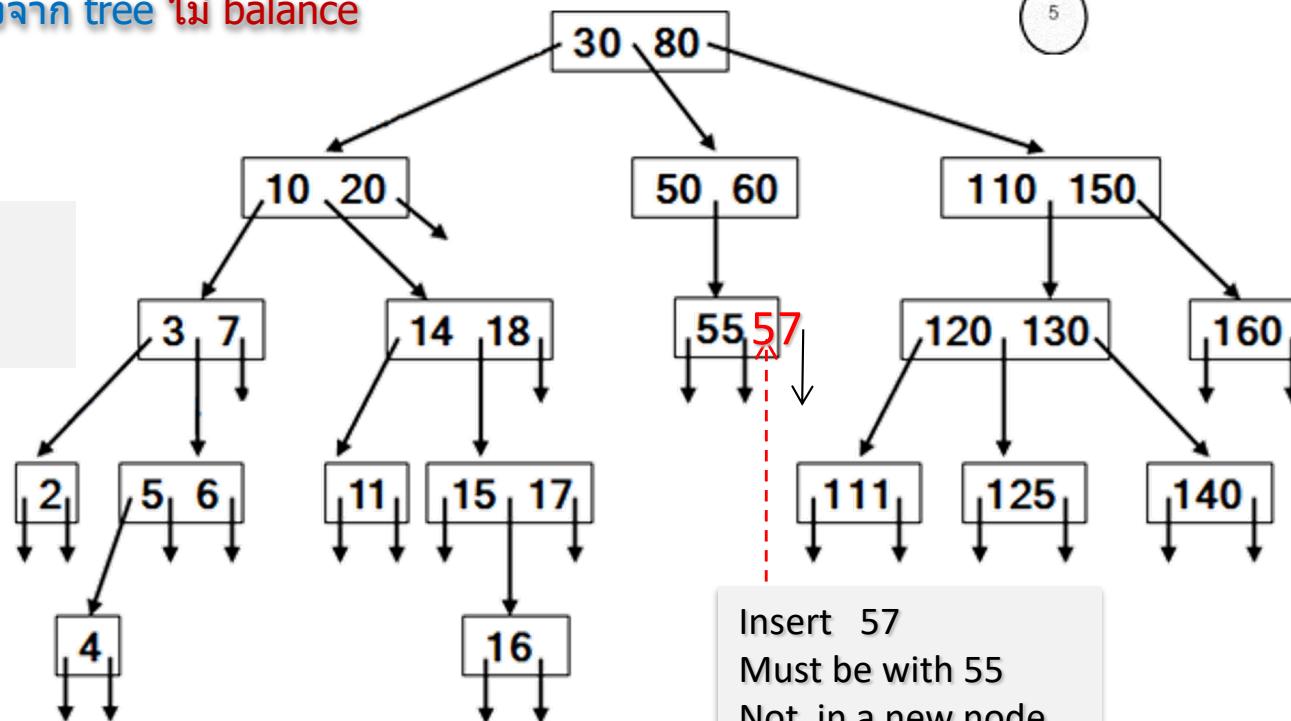
Top Down Tree : Non full node must be leaf.

Node ที่ไม่เต็มจะเป็น leaf ได้เท่านั้น เป็น internal node ไม่ได้

- Fill up the existing nodes before creating a new node !
เติม node ให้เต็มก่อนค่อยแทก node ใหม่

แนวคิด : มี nodes ให้น้อยที่สุด เพราะ อยากให้ height สั้น

แม้ว่าพยายามทำให้ node น้อย โดยเติมให้เต็มก่อนสร้าง node ใหม่ ก็ตาม
height อาจยังไฉ้เนื่องจาก tree ไม่ balance



Tree 2

- 3. AVL Tree
Height Balanced Tree
- 4. Which Representations ?
- 5. n-ary Tree
- 6. Generic Tree
- 7. Multiway Search Tree
- 8. Top Down Tree
- 9. Balanced Tree
B-Tree



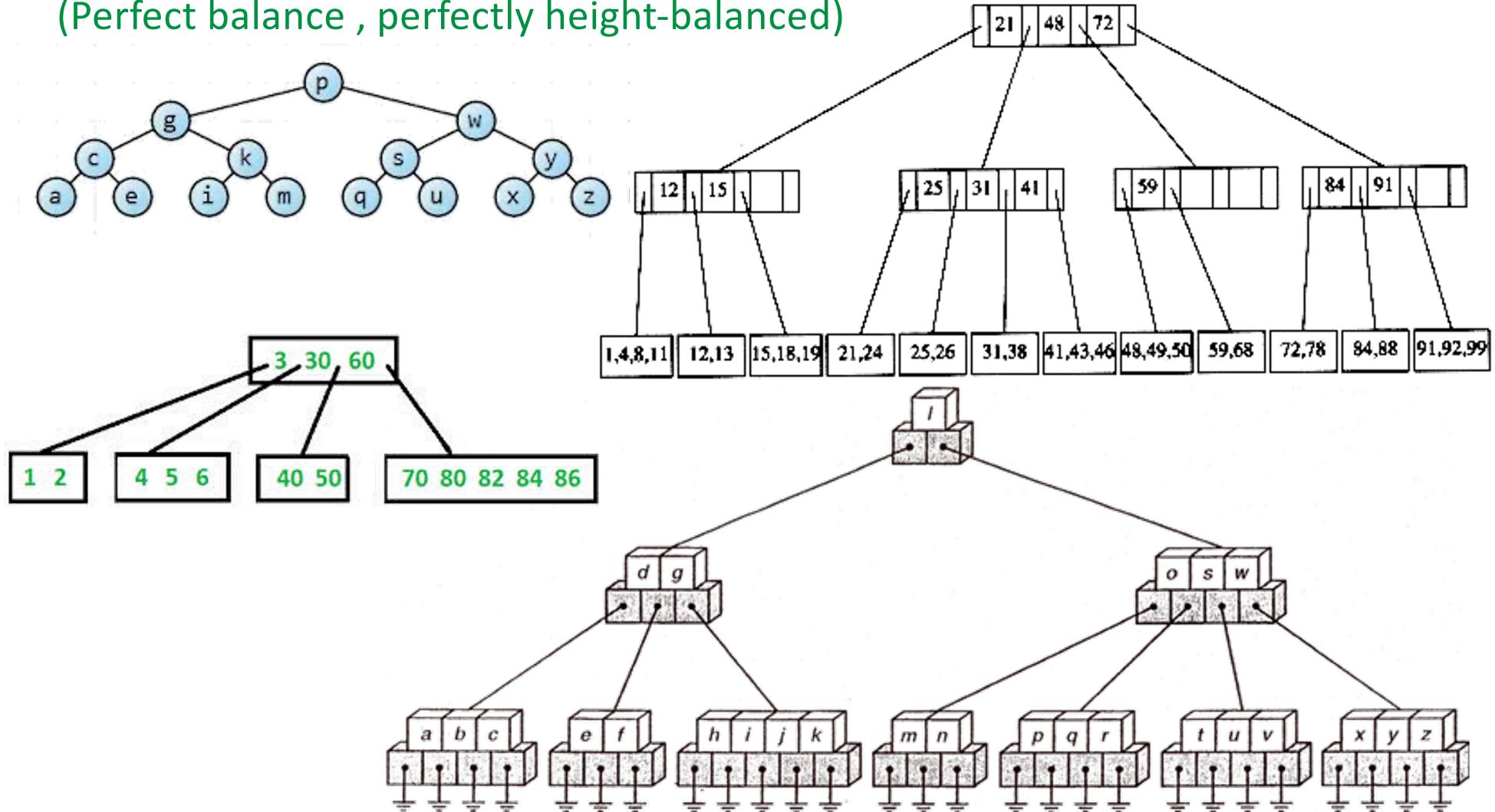
- 1. Tree Definitions
- 2. Binary Tree
 - Traversals
 - Binary Search Tree
 - Representations
 - Application : Expression Tree

Balanced Tree

Balanced Tree : every leaf node is at the same level

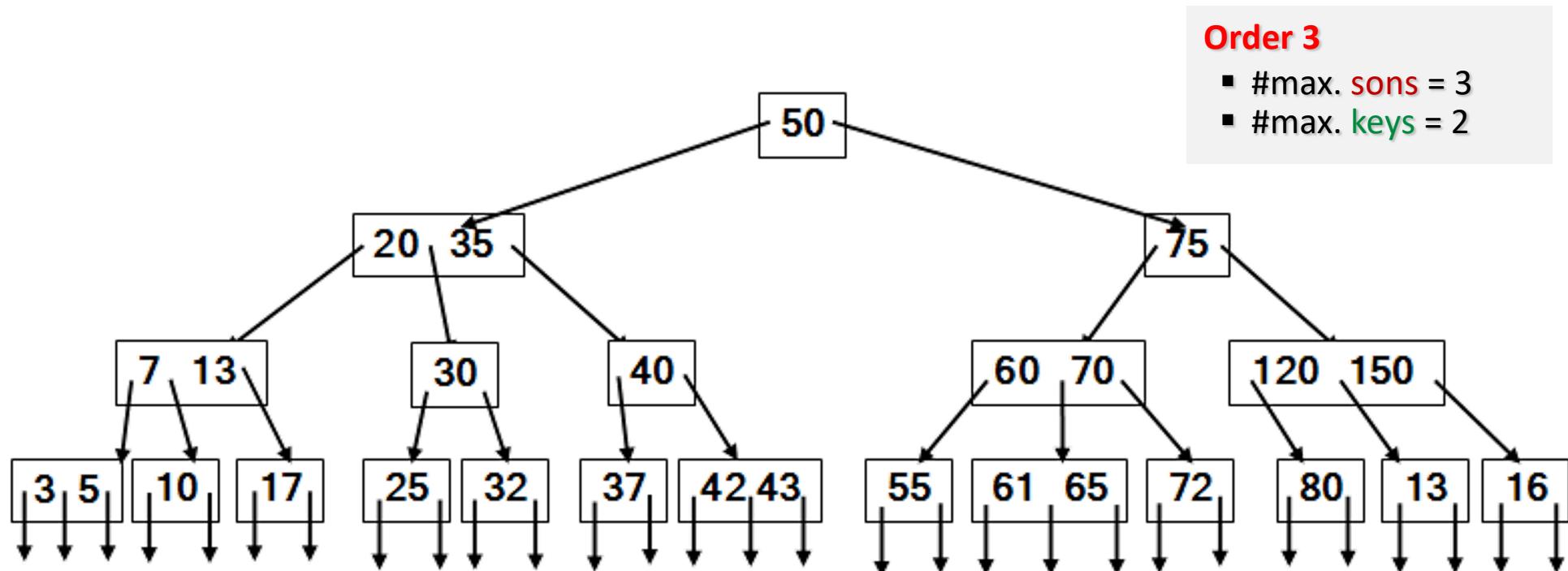
ทุก leaf อยู่ level เดียวกัน

(Perfect balance , perfectly height-balanced)



B-tree order n

1. Multi-way search tree order n : #max son = n, #max keys = n-1
2. **Balanced tree** : ทุก leaf อยู่ใน level เดียวกัน
(Perfect balance , perfectly height-balanced)
3. => next page



B-tree order n

1. Multi-way search tree order n

#max son = n, #max keys = n-1

2. Balanced tree

3. ทุก node มีจำนวน non-empty subtree (มีลูก)อย่างน้อยที่สุด เท่ากับครึ่งหนึ่งของ n (มีน้อยกว่านี้ไม่ได้) คือ Half Full
ยกเว้น root node ไม่ต้อง half full ก็ได้

#min sons = $\lceil n/2 \rceil$ ยกเว้น root



- > ไม่มี node ที่เล็กมากๆ
- > ดังนั้น จำนวน node หั้งหมดไม่มากนัก
- > height ไม่ยาว

EX.

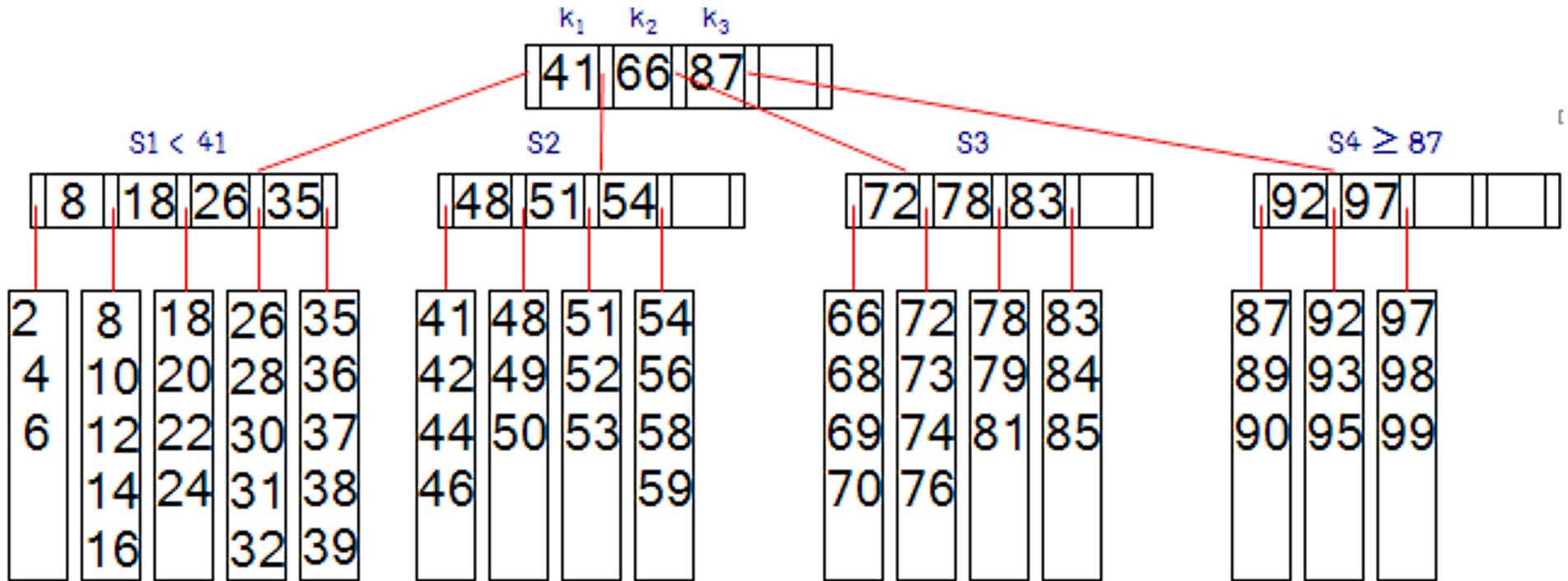
1. order 2 : #max sons = 2 , #min sons = $\lceil 2/2 \rceil = 1$

\therefore Binary ที่เป็น (almost) complete binary tree จะเป็น B-tree

2 order 5 : #max sons = 5 , #min sons = $\lceil 5/2 \rceil = \lceil 2.5 \rceil = 3$

3 order 201 : #max sons = 201 , #min sons = $\lceil 201/2 \rceil = 101$

B-Tree Variation



order 5 #max son = 5, #max keys = 4

- Half Full #min sons = $\lceil 5/2 \rceil = \lceil 2.5 \rceil = 3$ (ยกเว้น root node)

B-tree -> variations

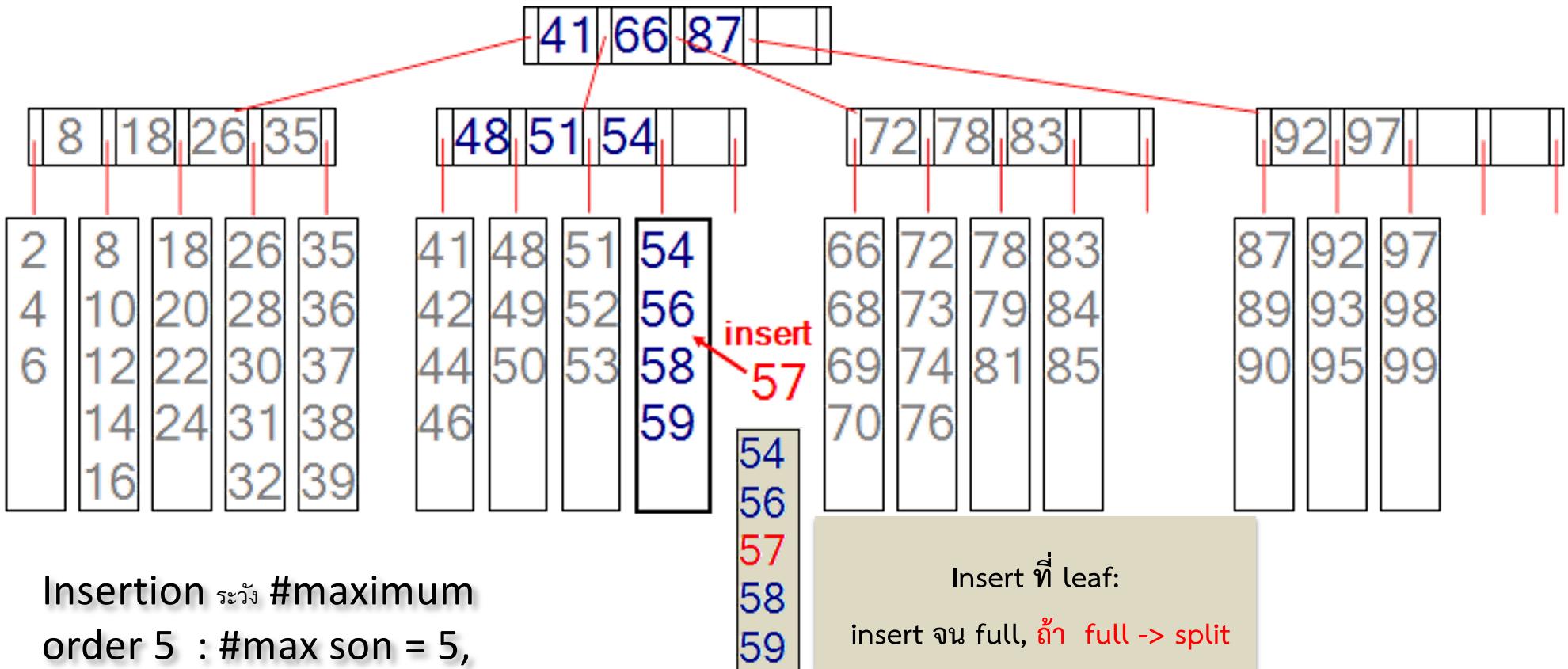
• **internal nodes** : เก็บเฉพาะ key เพื่อประยัดพื้นที่ (เก็บ data ได้มากกว่า เก็บทั้ง record), **order** กำหนดจำนวนลูก

• **external nodes** : เก็บ data ทั้ง record

ค่า **L** กำหนดจำนวน record/node ของ leaf

ถ้า **L = 5** : #max data = 5, #min data = $\lceil 5/2 \rceil = \lceil 2.5 \rceil = 3$

B-Tree (Variation) Insertion



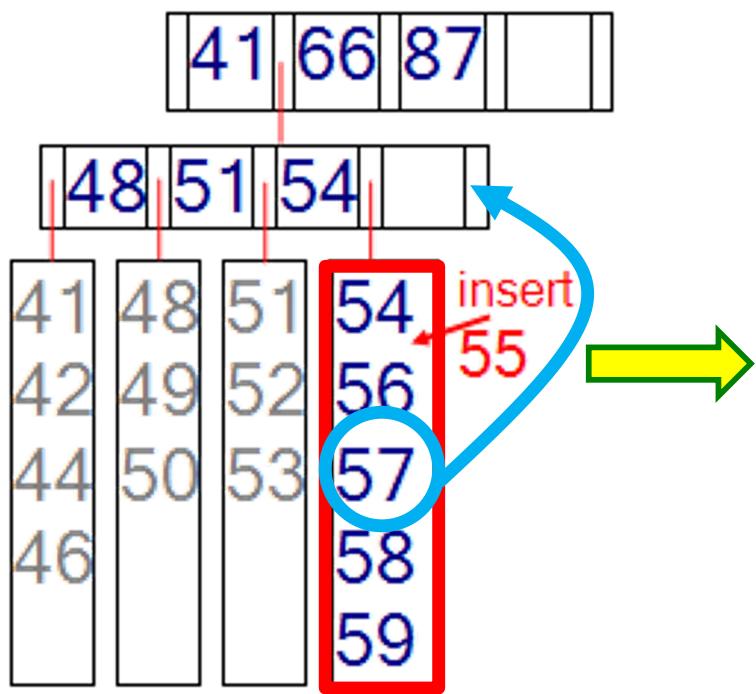
จะ insert ต้องระวังไม่ให้เกินจำนวนที่กำหนด (max)

\rightarrow Insert ที่ leaf ดูค่า L

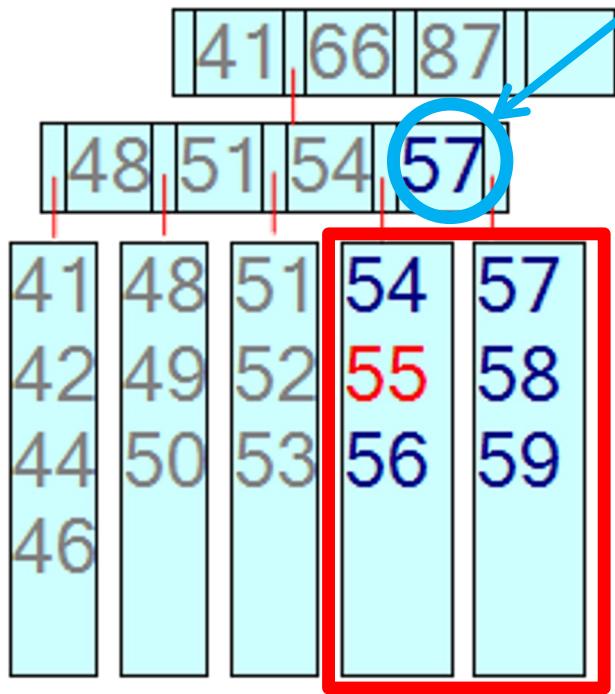
Leaf L = 5: #max data = 5

B-Tree (Variation) Insertion

order 5, #max key = 5-1=4
L=5 #leaf max data = 5



Insert 55
Insert ที่ leaf จน full :
L = 5: #max data = 5
Full



order 5 : internal node
#max son = 5
: **#max keys = 4**,
not full -> can insert

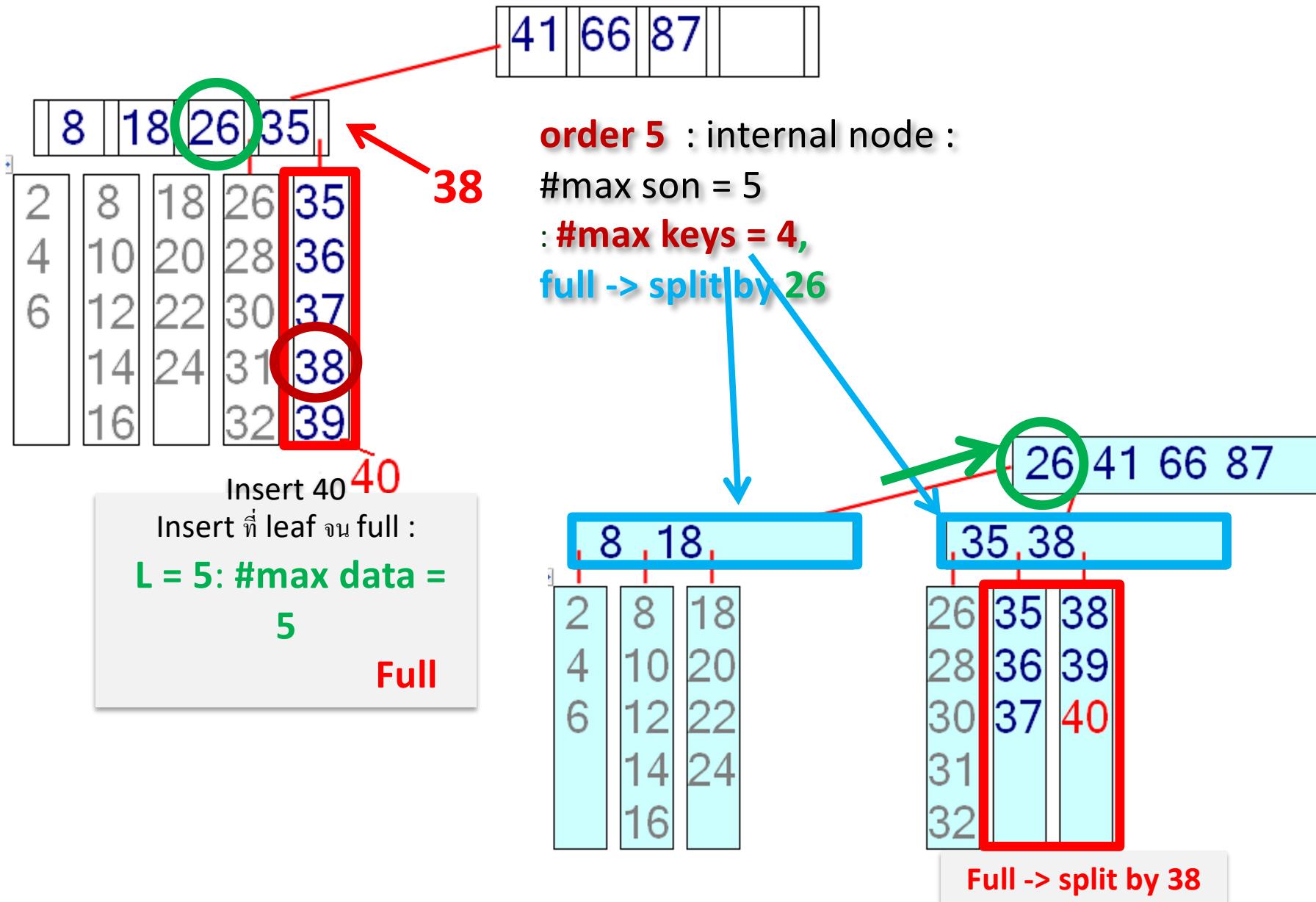
key 57 splits 2 nodes
key 57 ต้องถูก insert ใน father node

Full -> split half-half ที่ 57

B-Tree (Variation) Insertion

Insert at leaf:

until full, if full -> split



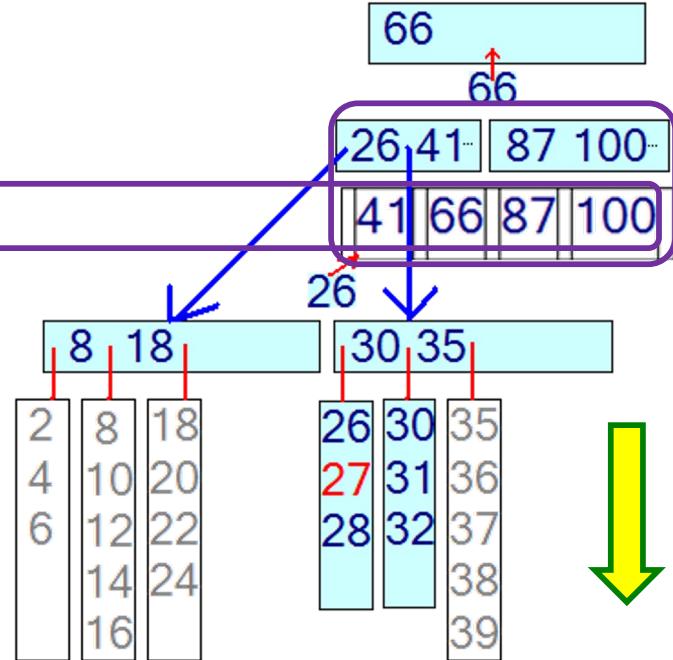
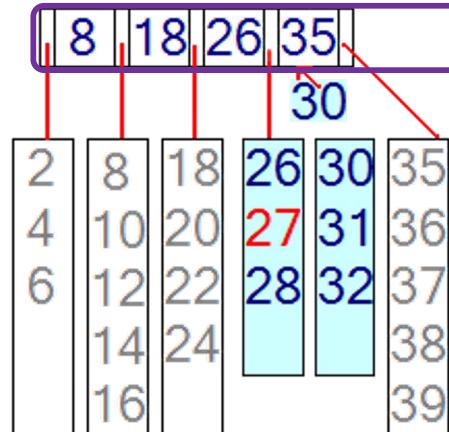
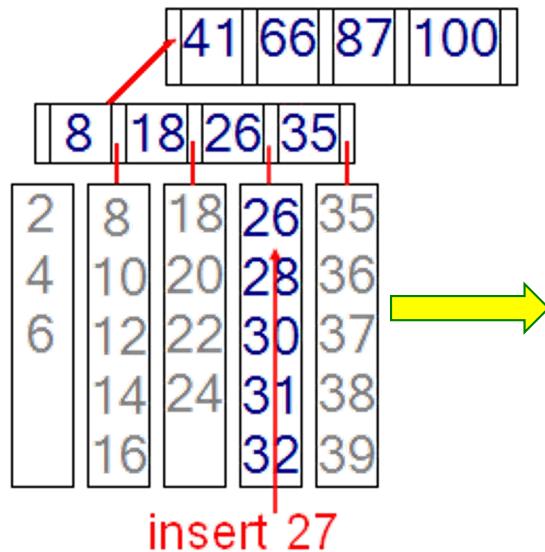
B-Tree (Variation) Insertion

Insert at leaf:

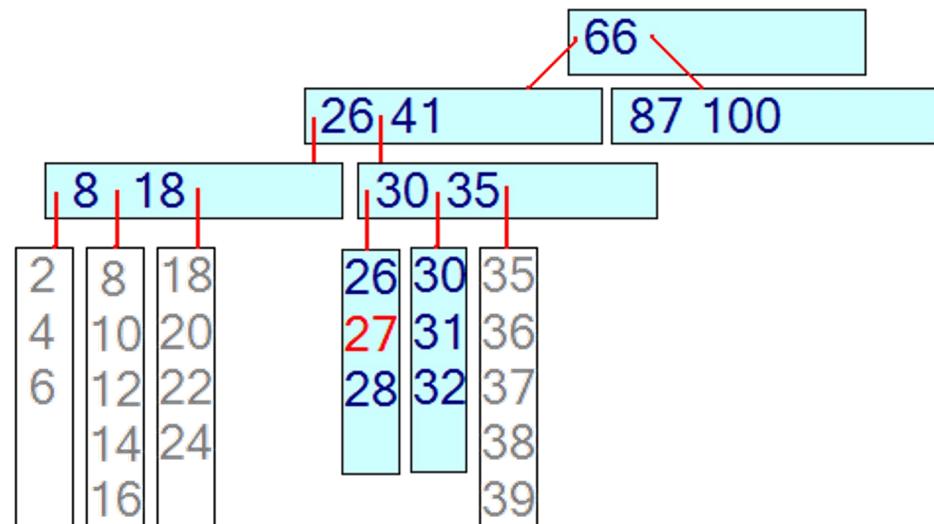
until full, if full -> split

order 5, #max key = $5-1=4$

L=5 #leaf max data = 5



Insertion 挿入 #maximum
order 5 : #max son = 5,
Internal : **#max keys = 4**,
#min sons = $\lceil 5/2 \rceil = 3$
Leaf L = 5: **#max data = 5**,
#min data = $\lceil 5/2 \rceil = 3$



B-Tree (Variation) Deletion

Deletion consider

#minimum

Internal order 5 :

$$\# \text{min sons} = \lceil 5/2 \rceil =$$

3

#min keys = 2

Leaf L = 5:

#max data = 5,

$$\# \text{min data} = \lceil 5/2 \rceil$$

= 3

Delete at leaf:

insert ระวังไม่ให้เกิน ต้องดู max

delete ระวังไม่ให้ขาด ต้องดู min

until low, if low -> borrow

Borrowing

1. ยืมพี่น้อง (ผ่านพ่อ)

2. ยืมพ่อ ต้อง consolidate

พ่อไม่มี พ่อยืม

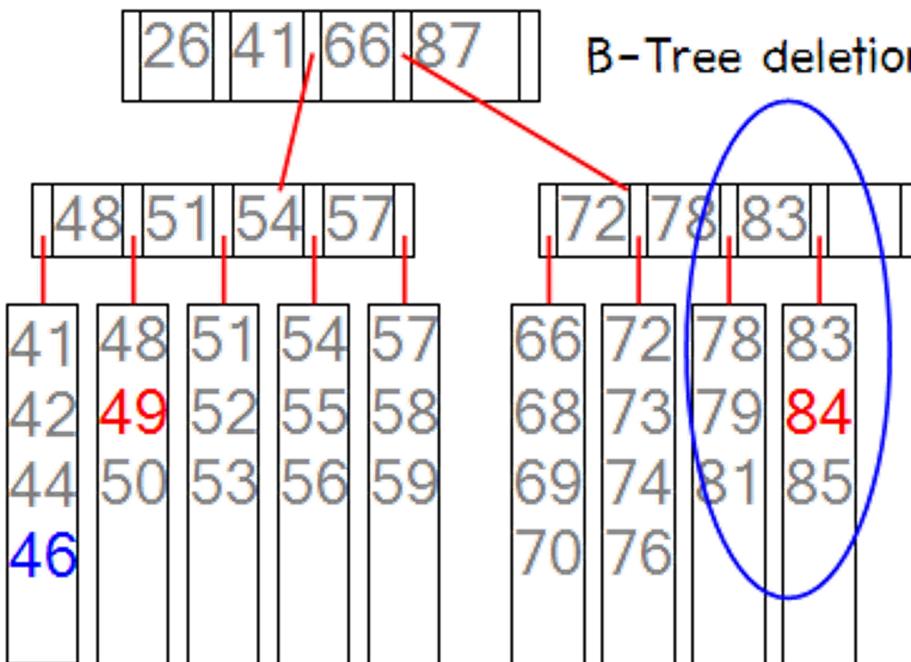
-พี่น้องพ่อ (ผ่านปู่)

...

B-Tree (Variation) Deletion

Delete at leaf:

until low, if low -> borrow



B-Tree deletion : **delete 49,84**

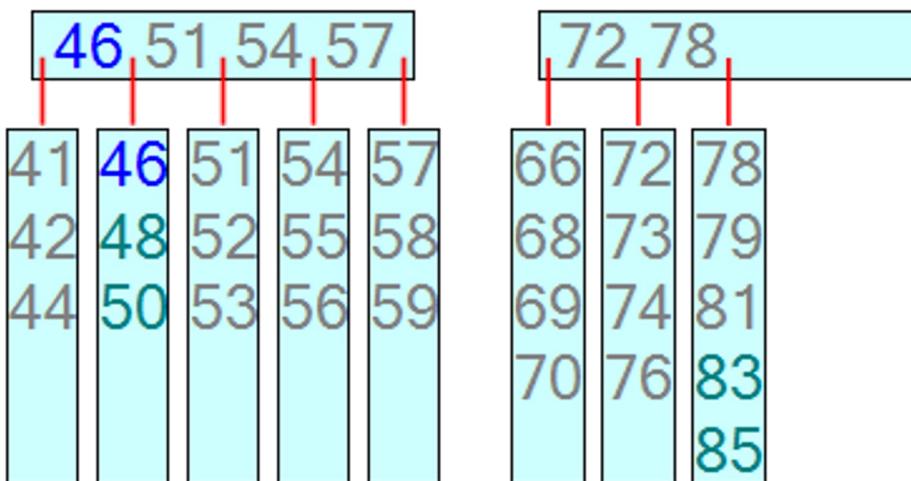
Deletion consider #minimum

Internal order 5 : #min sons = $\lceil 5/2 \rceil = 3$

#min keys = 2

Leaf L = 5: #max data = 5,

#min data = $\lceil 5/2 \rceil = 3$



Borrowing

1. ยืมพี่น้อง(ผ่านพ่อ) เช่น del 49 เอา 46 ของพี่มา
2. ยืมพ่อ ต้อง consolidate เช่น del 84 พี่น้องไม่มีให้ยืม ยืมพ่อ โดยรวม consolidate กับพ่อ

ถ้าพ่อไม่มี พ่อยืม
-พี่น้องพ่อ(ผ่านปู่)

...

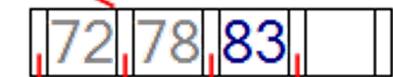
B-Tree (Variation) Deletion

Delete at leaf:

until low, if low -> borrow



B-Tree deletion : **delete 99**



66	72	78	83
68	73	79	84
69	74	81	85
70	76		



87	92	97	
89	93	98	
90	95	99	

Deletion consider

#minimum

Internal order 5 :

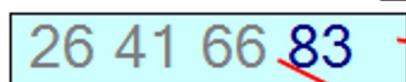
$$\# \text{min sons} = \lceil 5/2 \rceil = 3$$

#min keys = 2

Leaf L = 5:

#max data = 5,

$$\# \text{min data} = \lceil 5/2 \rceil = 3$$



66	72	78	83
68	73	79	84
69	74	81	85
70	76		



87	92		
89	93		
90	95		
97	98		

Borrowing

1. ยืมพี่น้อง (ผ่านพ่อ)

2. ยืมพ่อ ต้อง consolidate

พ่อไม่มี พ่อยืม

-พี่น้องพ่อ (ผ่านปู่)

...