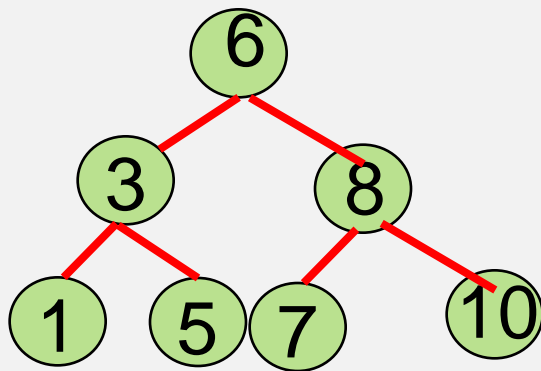




## การ insert Binary Search Trees

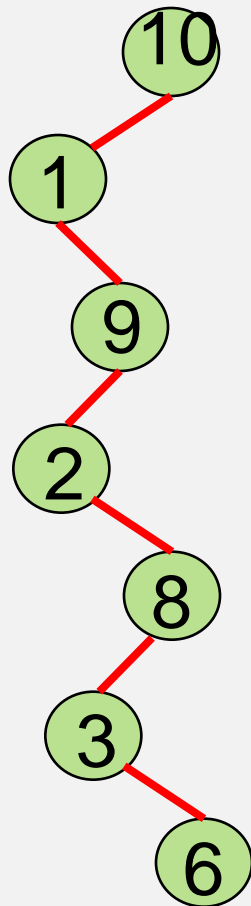
6 3 8 1 5 7 10



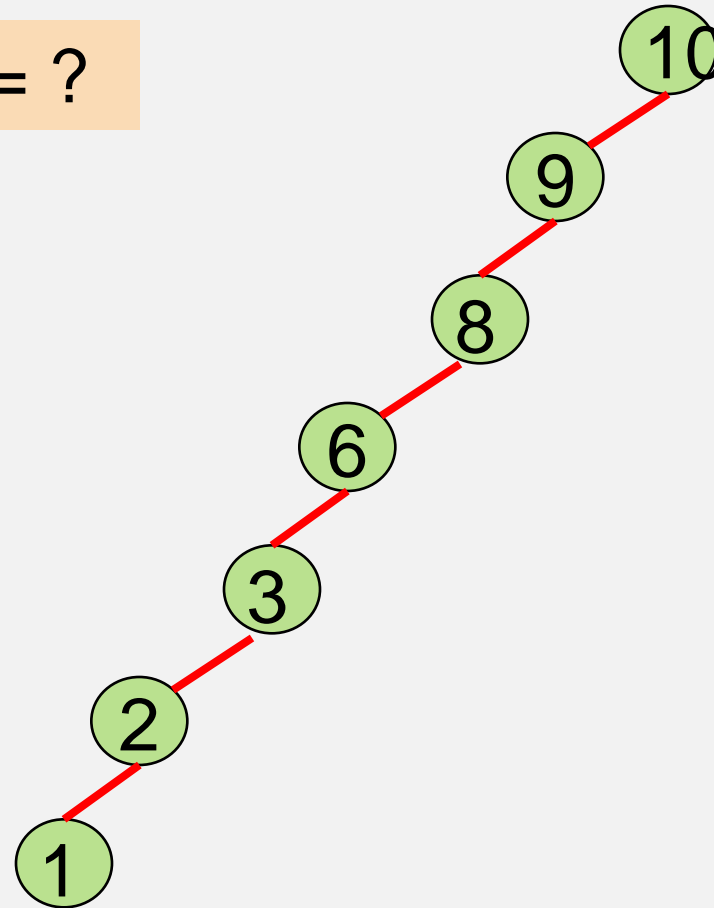


Insert : 10 1 9 2 8 3 6

Insert : 10 9 8 6 3 2 1



Big O = ?





## ปัญหา Big O ของ Binary Search Trees

worse case  $O(n)$

### แก้ไข

2.1 ต้องทำให้ทรี Perfect Balance Trees : ทำยาก

2.2 AVL Trees

AVL Binary Search Trees ที่มี Balance condition  
(ความสูงต่างกันไม่เกิน 1)

- Single Rotation
- Double Rotation



### **3.4 AVL Trees**

A binary search tree with **balance condition**, it ensures that the depth of the tree is  $O(\log_n)$ .

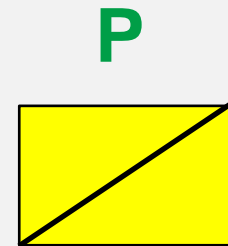
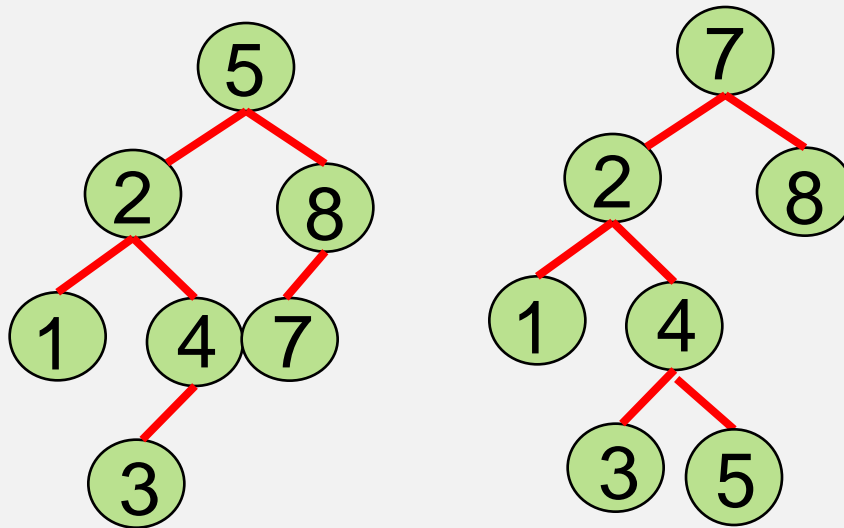
**Balance condition** : Every node must have left and right subtree of the same height.

**Problem** : Only **perfect balance trees** of  $2^k - 1$  node would satisfy this criterion.



**Definition** : An AVL(Adelson-Velskii and Landis) tree is identical to a binary search tree, except that for every node in the tree, the height of the left and the right subtree **can differ by at most 1**.

AVL Operation      All =  $O(\log_N)$  except insertion



**The height of an empty subtree is defined to be -1.**



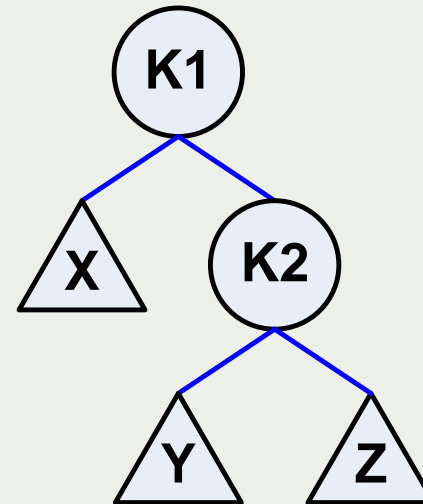
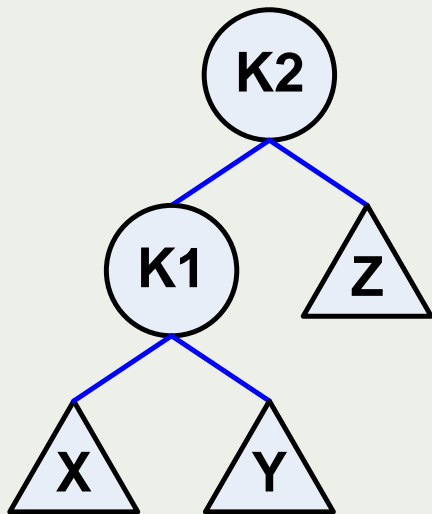
### **3.4.1 Insertion operation :**

1. Update all the balancing information for the nodes on the path back to the root
2. The insertion a node could violate the AVL tree property, then property has to be restored before the insertion step. Called a *rotation*
  - **Single rotation**
  - **Double rotation**



### 3.4.2 Single Rotation

- A rotation involves only a few pointer changes, and changes the structure of the tree while preserving tree property.





**Example** insert 1, 2, 3, 4, 5, 6, 7

นอกช่วง





**Example** insert 15, 14

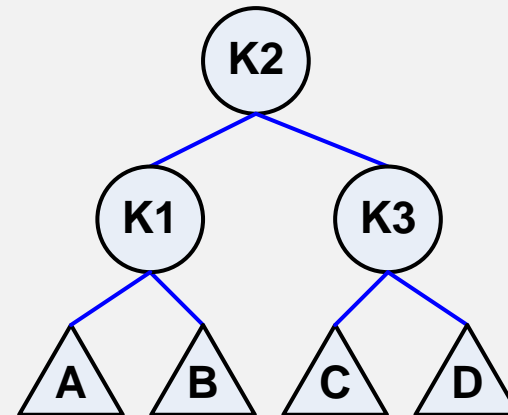
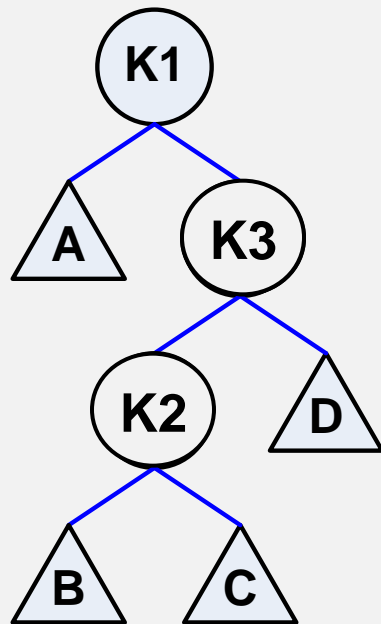
insert 13, 12, 11, 10, 9, 8, 8.5

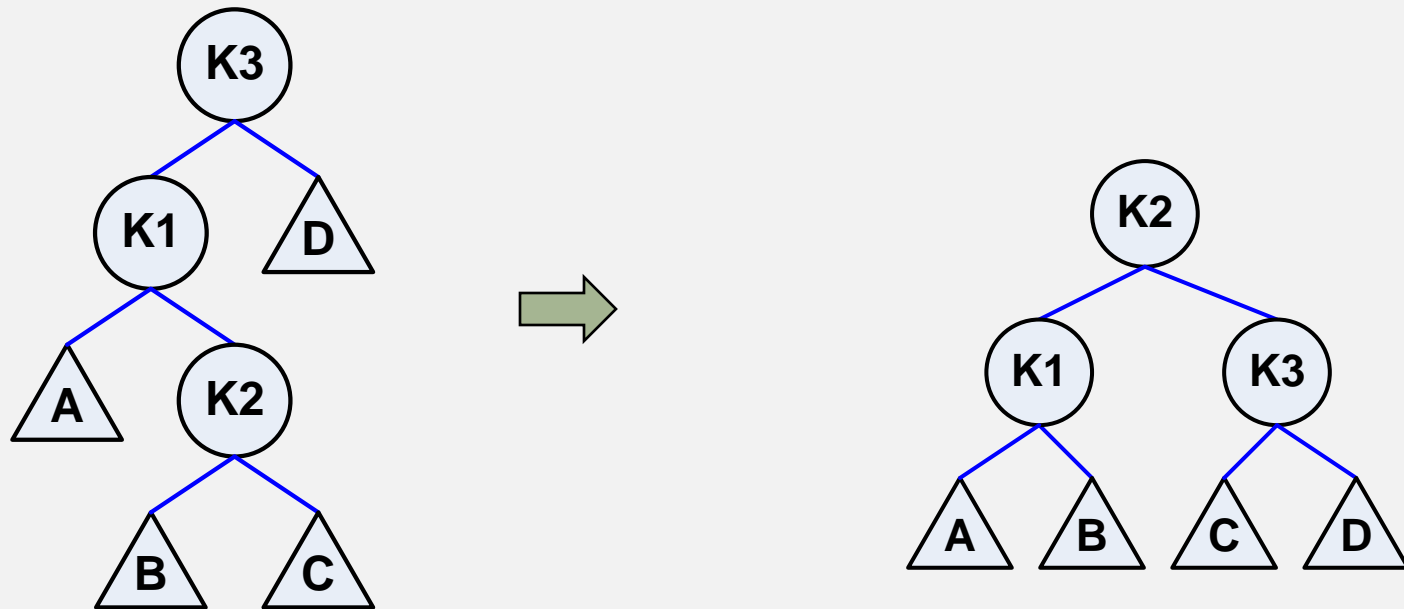
นอกช่วง



### 3.4.3 Double rotation

**Problem** : Single rotation has not fixed the height imbalance by a node inserted into the tree containing the middle elements.





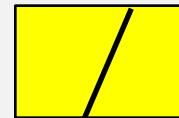


```
struct node
{
    int value;
    int height;
    struct node *left;
    struct node *right;
};
```

```
1. int fheight( struct node *P)
2. { if ( P==NULL)
3.     return -1;
4.     else
5.     return P->height;
}
```



P



P



1024



## Code Insert

ถ้า tree == NULL

สร้าง node ใหม่ ใส่ค่า

else

ถ้า x น้อยกว่า tree->value ให้ recursive ลงไปทางซ้าย  
tree->left = insert (tree->left)

ถ้าความสูงของทรีทางซ้ายและทางขวาต่างกันเกิน 2

ถ้า x น้อยกว่า tree->left->value  
single rotation (อยู่นอกช่วง)

else  
double rotation (อยู่ในช่วง)

update ความสูง

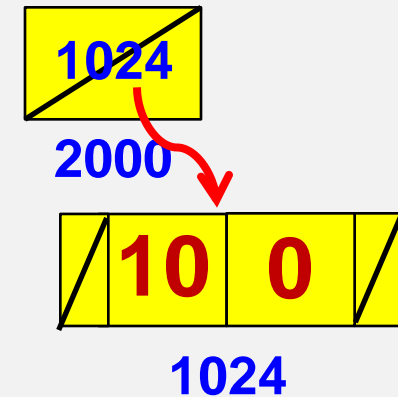
return tree

เมื่อ insert แล้ว return กลับ  
มาเช็คความสูง



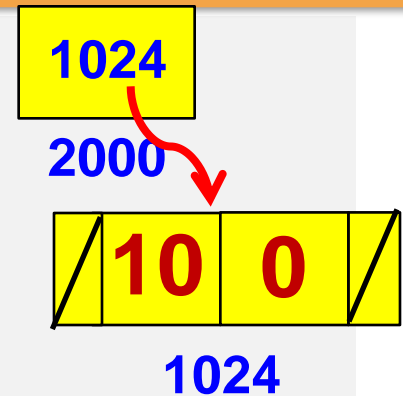
```
struct node *insert(int x, struct node *T)
1{   if(T == NULL)
2   {   T=new struct node;
3       T->value=x;
4       T->left=T->right=NULL;
5       T->height=0;
6   }
   else
```

tree





```
8 if(x < T->value )
9     { T->left = insert(x,T->left);
10         if(fheight(T->left) - fheight(T->right) == 2)
11             if( x < T->left->value)
12                 T=srleft(T);
13             else
14                 T=drleft(T);
15     }
16 else
17     ถึง 29 if(x > T->value )
18         { ข้างขวาบ้าง ให้เขียนเอง ... }
19
20         max(-1,-1) = -1 +1
21
22     T->height = max(fheight(T->left), fheight(T->right)) + 1;
23
24     return T;
25 }
```





```
struct node *insert(int x, struct node *T)
```

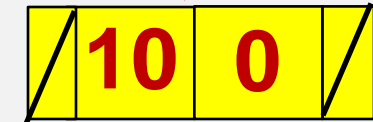
```
1{  if(T == NULL)
2    {  T=new struct node;
3        T->value=x;
4        T->left=T->right=NULL;
5        T->height=0;
6    }
7    else
```



tree



2000



1024





ถ้า x น้อยกว่า tree->value  
ให้ recursive ลงไปทางซ้าย

# Rule3 – Tree

tree

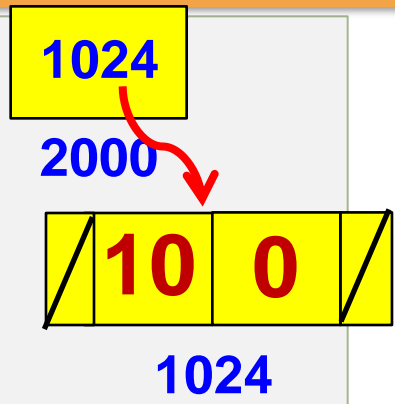
5, NULL

5

```
8 if(x < T->value )
9   { T->left = insert(x,T->left);
10   height(T->right) == 2)
11   ค้างบรรทัด 9 1024
12   T=srleft(T);
13   else
14   T=drleft(T);
15   }

16 else
17   ถึง 29 if(x > T->value )
18   { ข้างขวาบ้าง ให้เขียนเอง ... }

30 T->height = max(fheight(T->left), fheight(T->right)) + 1;
31 return T;
}
```





## Recursive 1

5, NULL

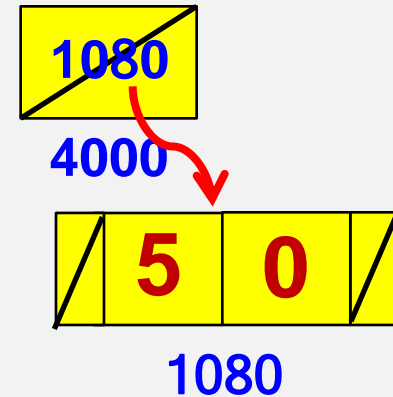
```
struct node *insert(int x, struct node *T)
```

```
1{   if(T == NULL)
2     {   T=new struct node;
3         T->value=x;
4         T->left=T->right=NULL;
5         T->height=0;
        }
7   else
```

.....

```
30   T->height =  $\max(-1, -1) = -1$   $+1$ 
31   return T;
}
```

tree



1080



ค้ำงบรรทัด 9 1024

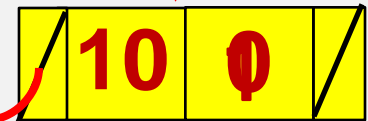
tree

1080

```
8 if(x < T->value )
9   { T->left = insert(x,T->left);
10      if(fheight(T->left) - fheight(T->right) == 2)
11          if( x < T->left->value)
12              T=srleft(T);
13          else
14              T=drleft(T);
15      }
16 29 if(x > T->value )
17     { ข้างขวาบ้าง ให้เขียนเอง ... }
30 T->height = max(fheight(T->left), fheight(T->right)) + 1;
31 return T;
}
```

1024

2000



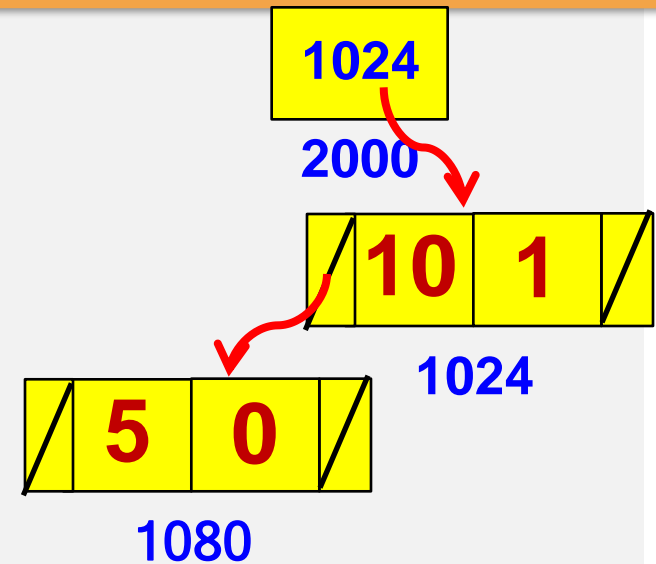
1024



1080



```
main()
{
    .....
    tree=insert(2,tree);
}
```



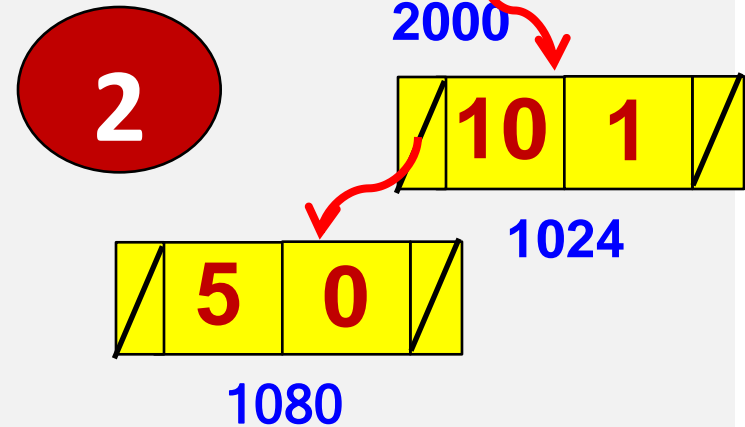


จาก main

2, 1024

```
struct node *insert(int x, struct node *T)
```

```
1{   if(T == NULL)
2   {   T=new struct node;
3       T->value=x;
4       T->left=T->right=NULL;
5       T->height=0;
        }
7   else
        ....
```





ถ้า x น้อยกว่า tree->value  
ให้ recursive ลงไปทางซ้าย

# Module 3 – Tree

tree

8 if(x < T->value )

2, 1080

9 { T->left = insert(x,T->left);

10 if(fheight(T->left) - fheight(T->right) == 2)

A ค้างบรรทัด 9 1024

12 T=srleft(T);

13 else

14 T=drleft(T);

15 }

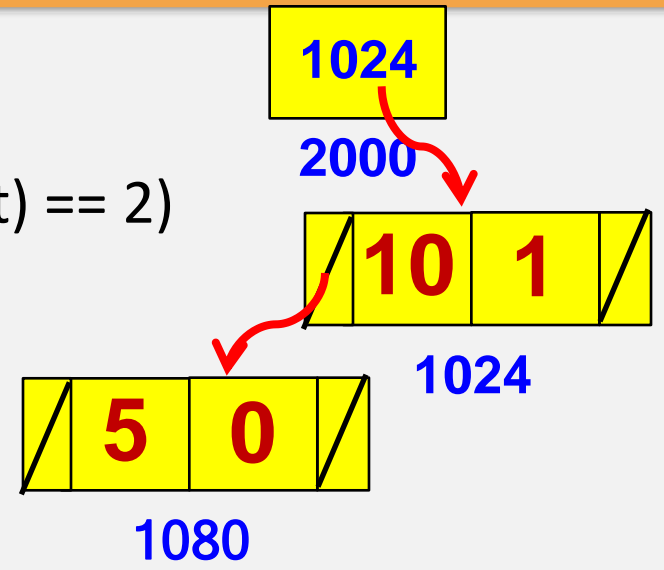
16 ถึง 29 if(x > T->value )

{ ข้างขวาบ้าง ให้เขียนเอง ... }

30 T->height = max(fheight(T->left), fheight(T->right)) + 1;

31 return T;

}



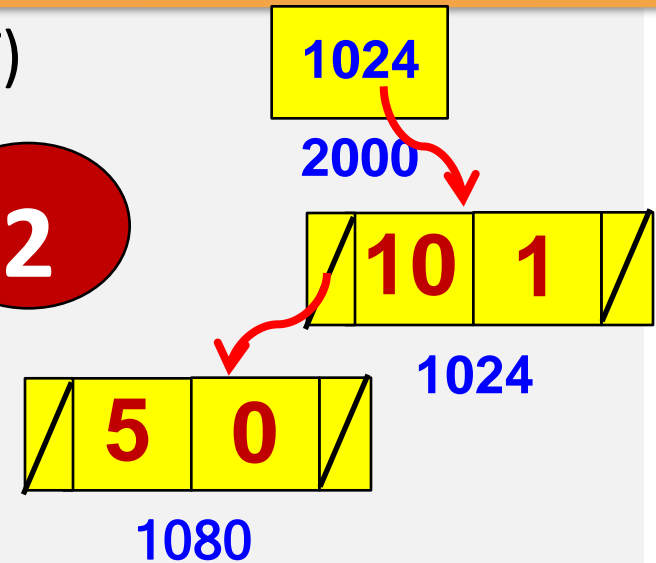


## Recursive 1

2, 1080

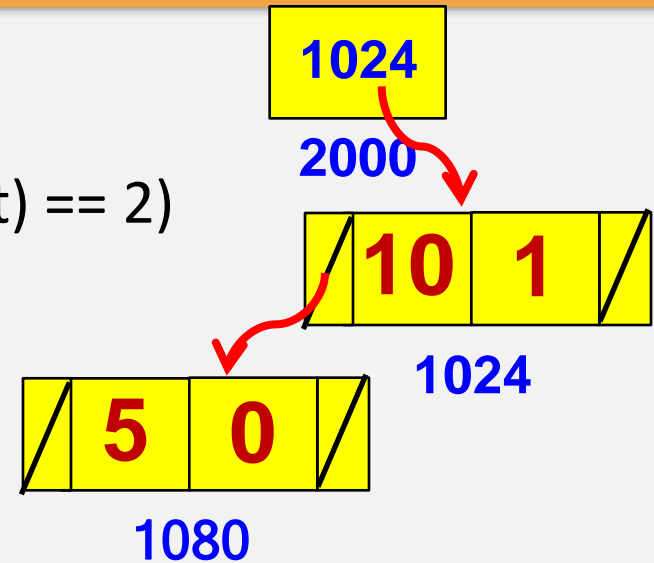
```
struct node *insert(int x, struct node *T)
```

```
1{   if(T == NULL)
2   {   T=new struct node;
3       T->value=x;
4       T->left=T->right=NULL;
5       T->height=0;
6   }
7   else
    ....
```



```

8 if(x < T->value )
9     { T->left = insert(x,T->left);
10         if(fheight(T->left) - fheight(T->right) == 2)
11             B ค้างบรรทัด 9 1080
12             T=srleft(T);
13         else
14             T=drleft(T);
15     }
16 ถึง 29 if(x > T->value )
17     { ข้างขวาบ้าง ให้เขียนเอง ... }
30 T->height = max(fheight(T->left), fheight(T->right)) + 1;
31 return T;
    }
```







## Recursive 2

## 2, NULL

```
struct node *insert(int x, struct node *T)
```

```
1{   if(T == NULL)
2     {   T=new struct node;
3         T->value=x;
4         T->left=T->right=NULL;
5         T->height=0;
6     }
7   else
```

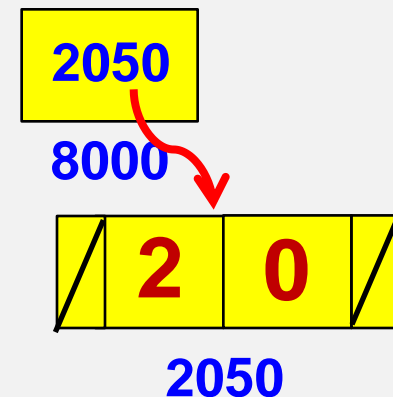
.....

```
30   T->height =  $\max(-1, -1) = -1$   $+1$  fheight(T->right)) + 1;
31   return T;
}
```

กลับไป B ค้างบรรทัด 9 1080



tree





# B คำบรรทัด 9 1080 e3 – Tree

tree

26

8 if(x < T->value )

2050

9 { T->left = insert(x,T->left);

10 if(fheight(T->left) - fheight(T->right) == 2)

11 if( x < T->left->value)

12 T=srleft(T);

13 else

14 T=drleft(T);

15 }

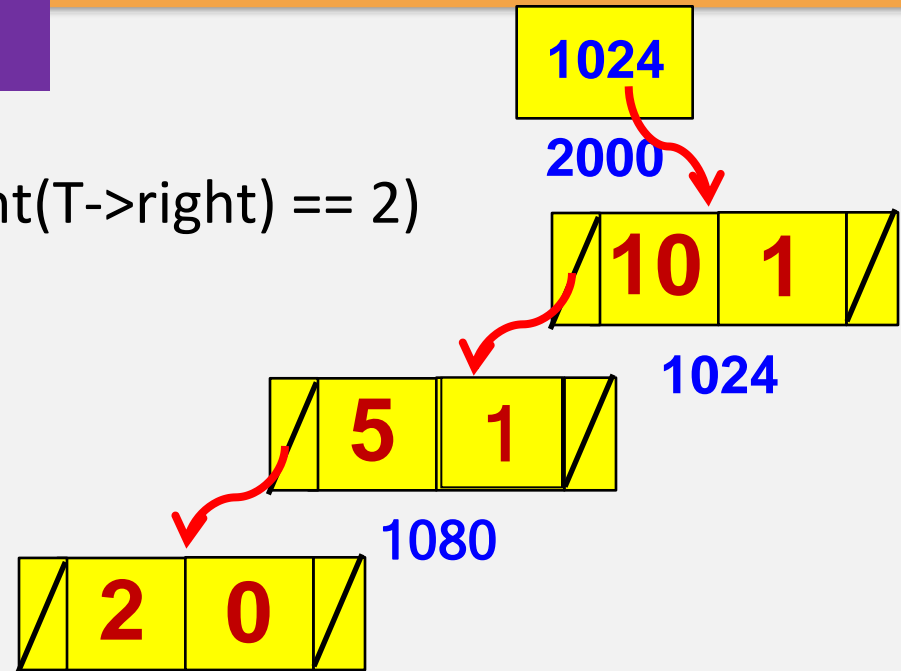
16 ถึง 29 if(x > T->value )

{ ข้างขวาบ้าง ให้เขียนเอง ... }

30 T->height = max(fheight(T->left), fheight(T->right)) + 1;

31 return T;

}





# A ค้างบรรทัด 9 1024

le3 - Tree

tree

27

8 if(x < T->value )

1080

9 { T->left = insert(x,T->left);

10 if(fheight(T->left) - fheight(T->right) == 2)

11 if( x < T->left->value)

12 T=srleft(T);

13 else

14 T=drleft(T);

15 }

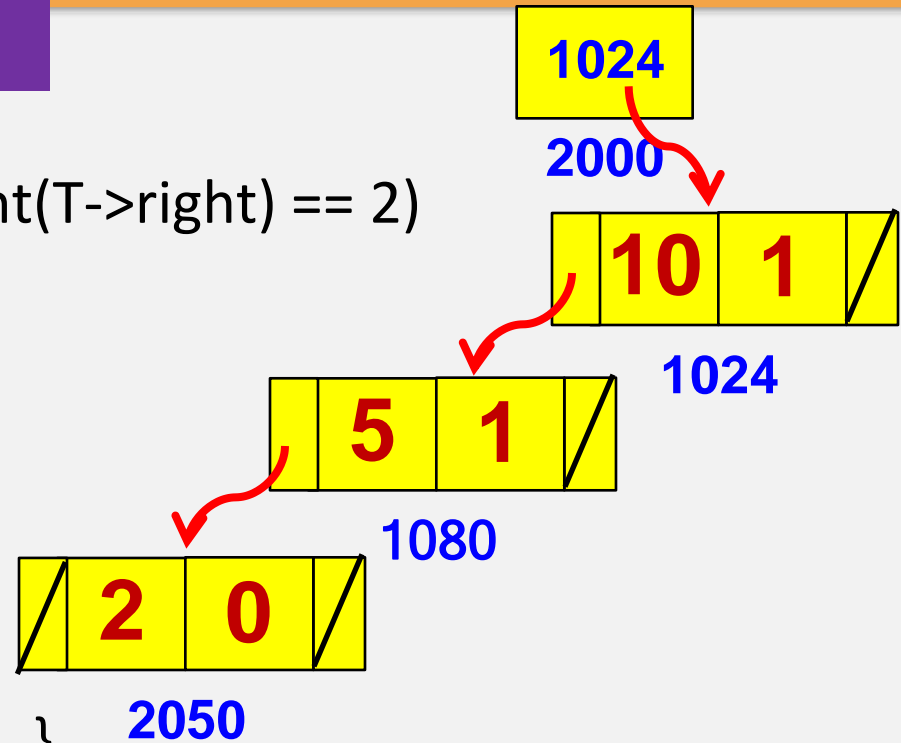
16ถึง 29 if(x > T->value )

{ ข้างขวาบ้าง ให้เขียนเอง ... }

30 T->height = max(fheight(T->left), fheight(T->right)) + 1;

31 return T;

}





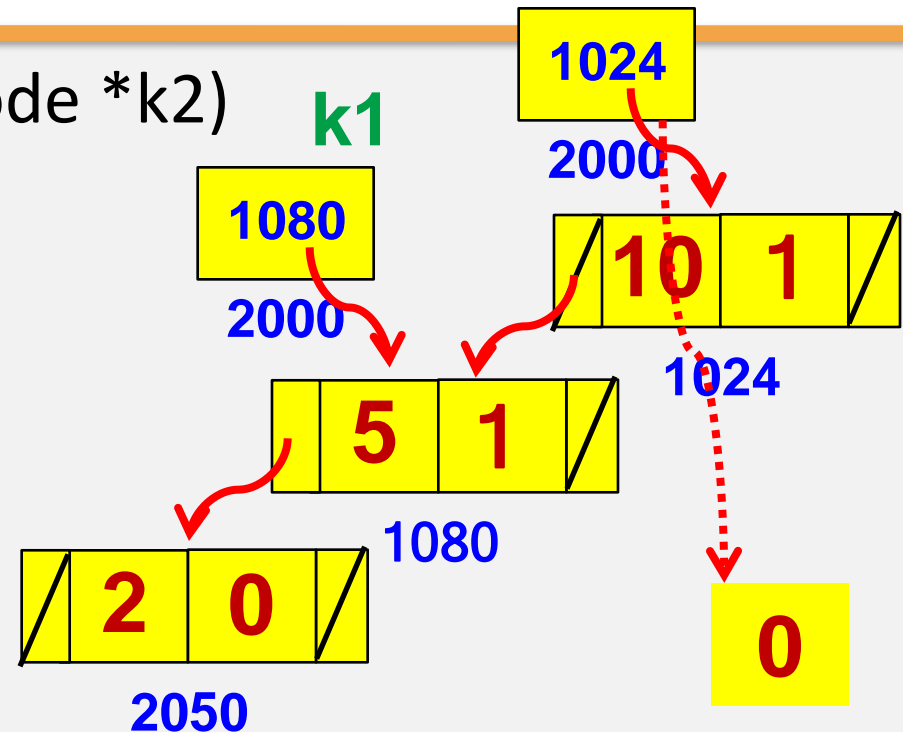
```
struct node *srleft(struct node *k2)
```

```
{ struct node *k1;
```

```
  k1=k2->left;
```

```
  k2->left = k1->right;
```

```
  k1->right = k2;
```



```
  k2->height = max(fheight(k2->left), fheight(k2->right)) + 1;
```

```
  k1->height = max(fheight(k1->left), k2->height) + 1;
```

```
  return k1;
```

```
}
```



# 1080

```
10    if(fheight(T->left) - fheight(T->right) == 2)
```

```
12      T=srleft(T);
```

```
14      T=drleft(T);
```

15 }

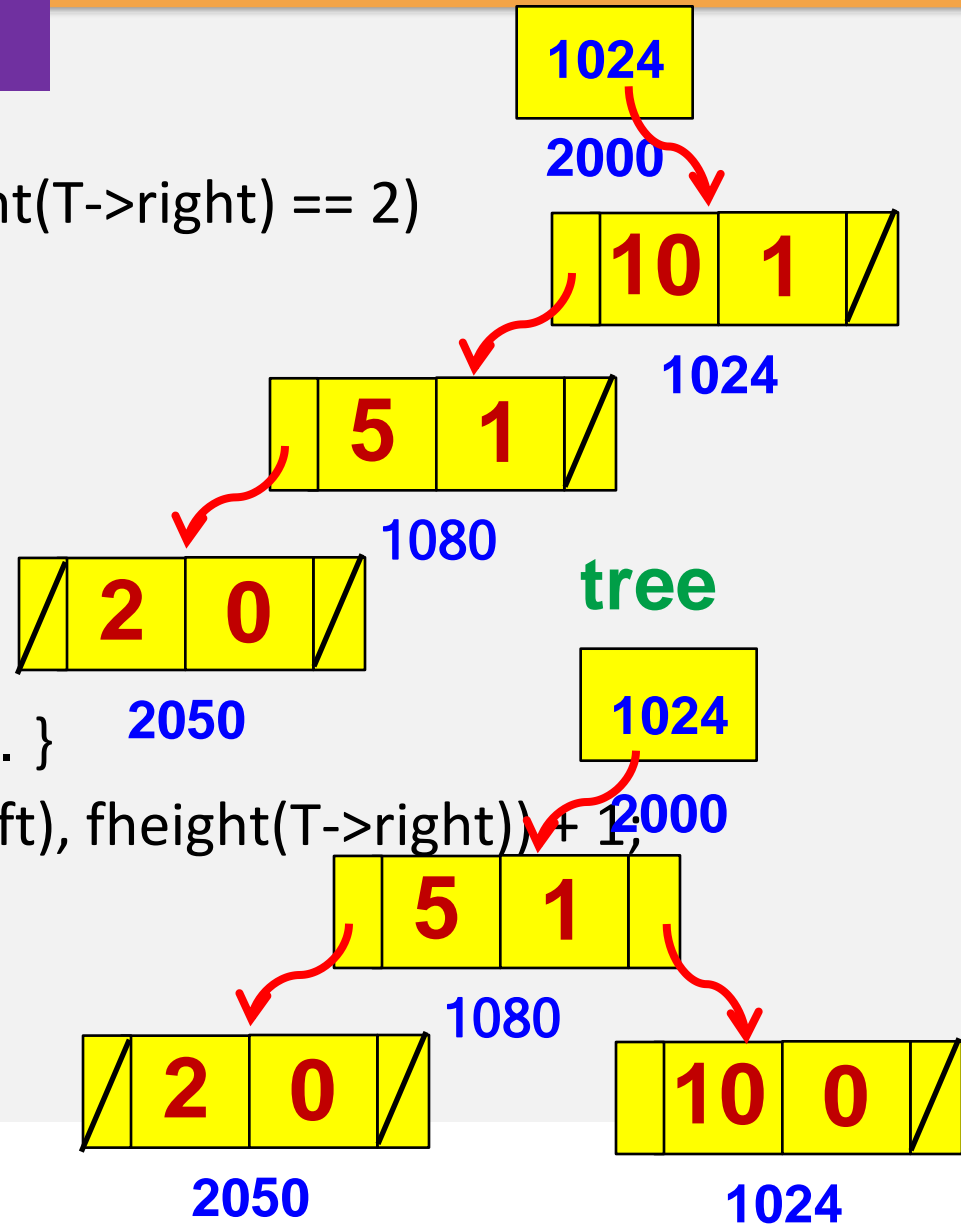
16ถึง 29 if(x > T->value )

{ ข้างขวบบ้าง ให้เขียนเอง ... }

```
30  T->height = max(fheight(T->left), fheight(T->right)) + 1;
```

```
31 return T;
```

}

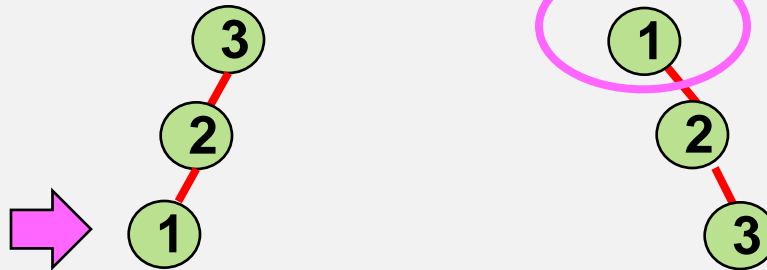




```
struct node *drleft(struct node *k3)
{
    k3->left = srright(k3->left);
    return srleft(k3);
}
```

### 3.5 Splay Trees

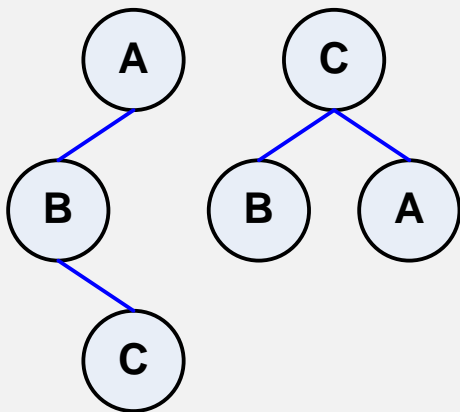
- $O(N)$  worst-case time per operation for binary search trees.
- After a node is accessed, it is pushed to the root



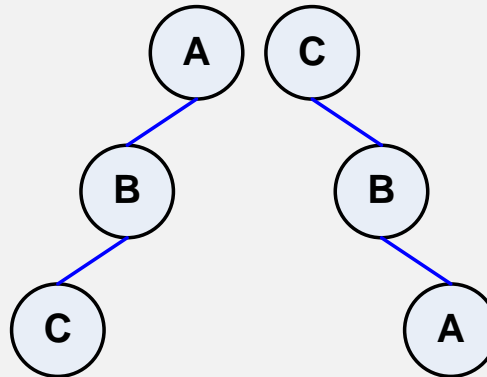
## Rotation idea

### Transform

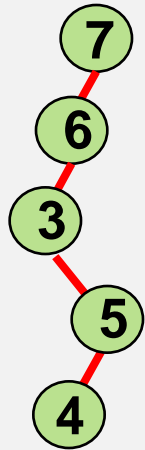
1. zig-zag

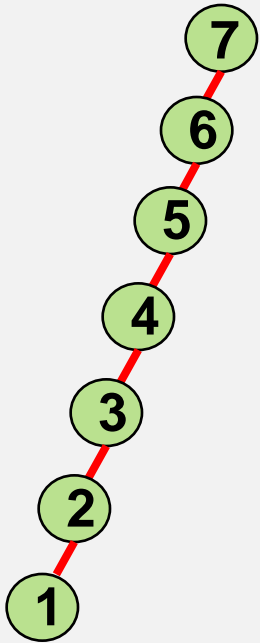


2. zig-zig







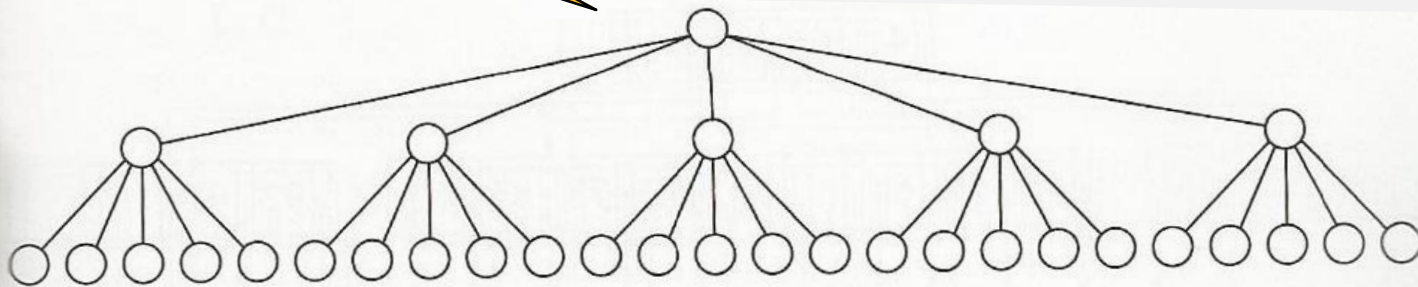
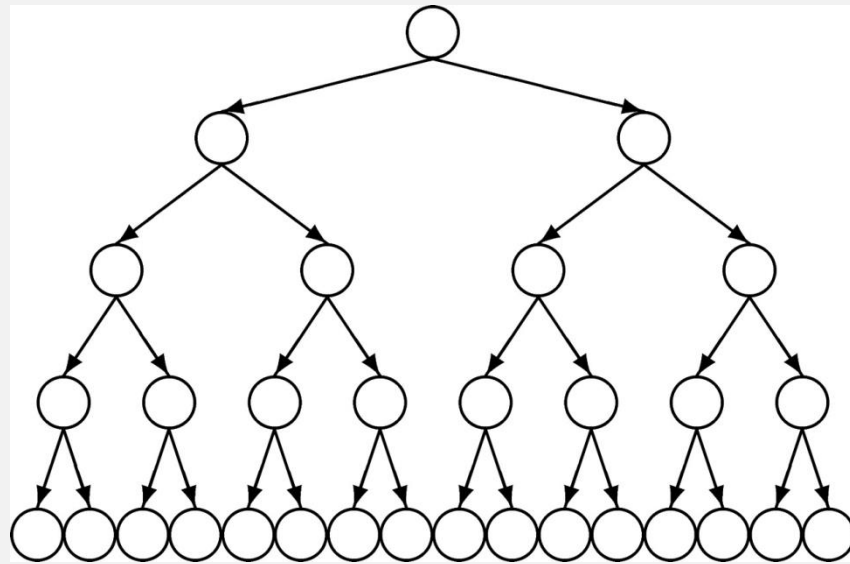




## 3.6 B-Trees

Complete binary  
Tree 31 nodes

สร้างทรีรูปแบบใหม่  
ให้มีคุณสมบัติตามที่เร  
ต้องการ ที่ไม่ใช่  
Binary Search, AVL

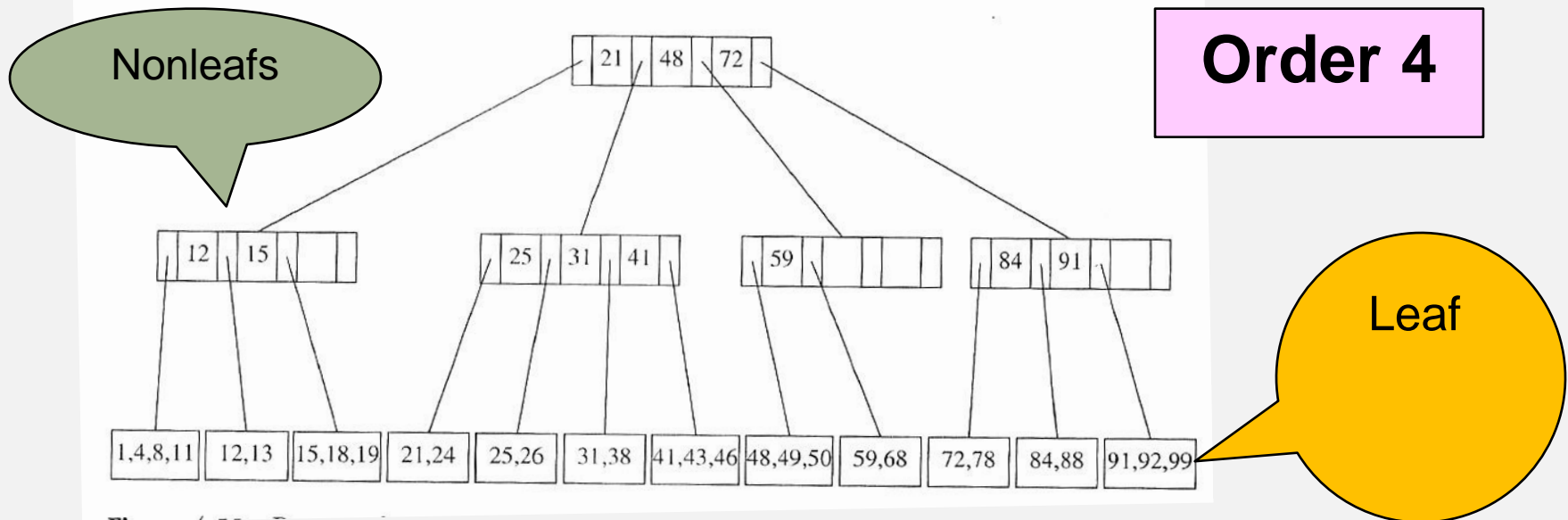




## 3.6 B-Trees

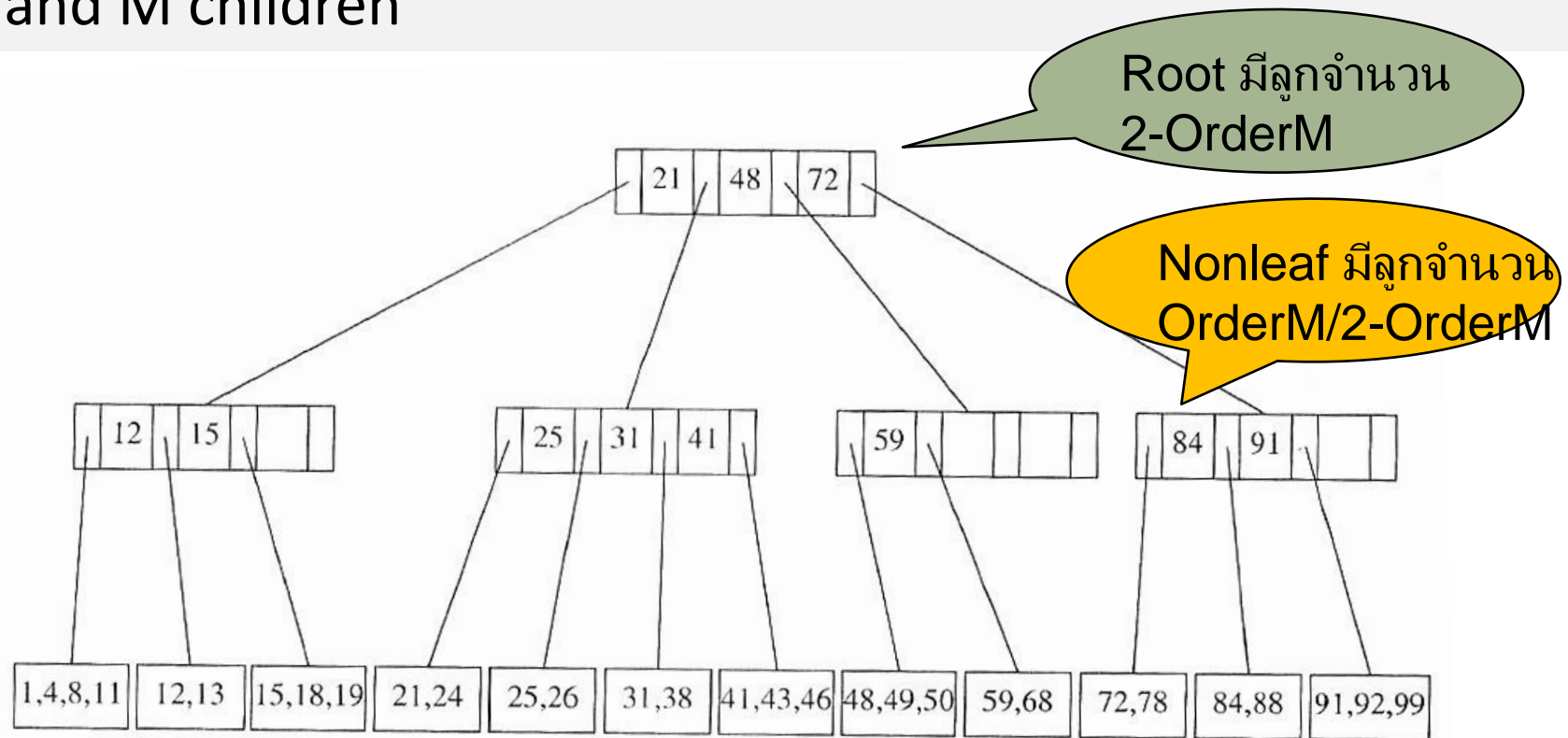
A B-tree of **order  $m$**  is a tree with the following structural properties:

- 1) The data items are stores at leaves.
- 2) The nonleaves nodes store up to  $M-1$  keys to guide the searching; key  $i$  represents the smallest key in subtrees  $i+1$



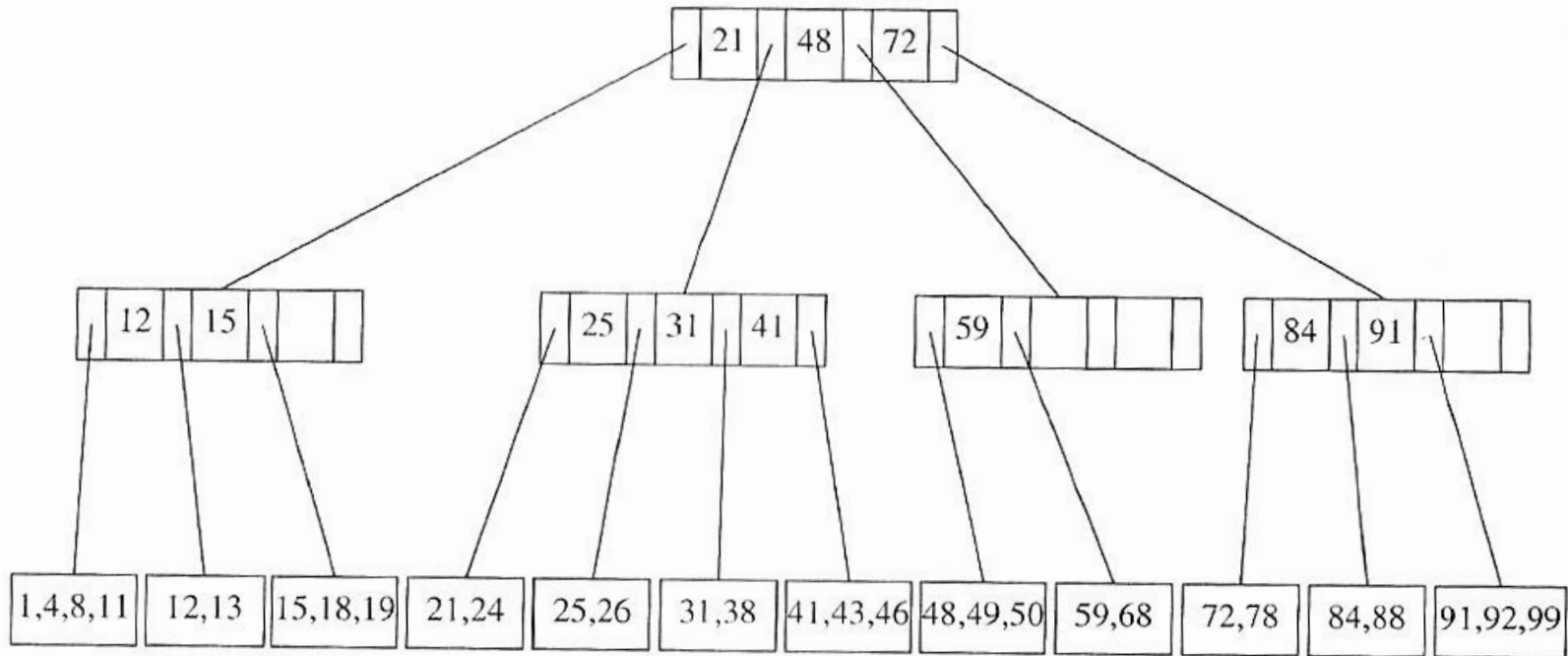


- 3) The root is either a leaf or has between two and M children.
- 4) All nonleaf nodes (Except the root) have between  $\lceil M/2 \rceil$  and M children





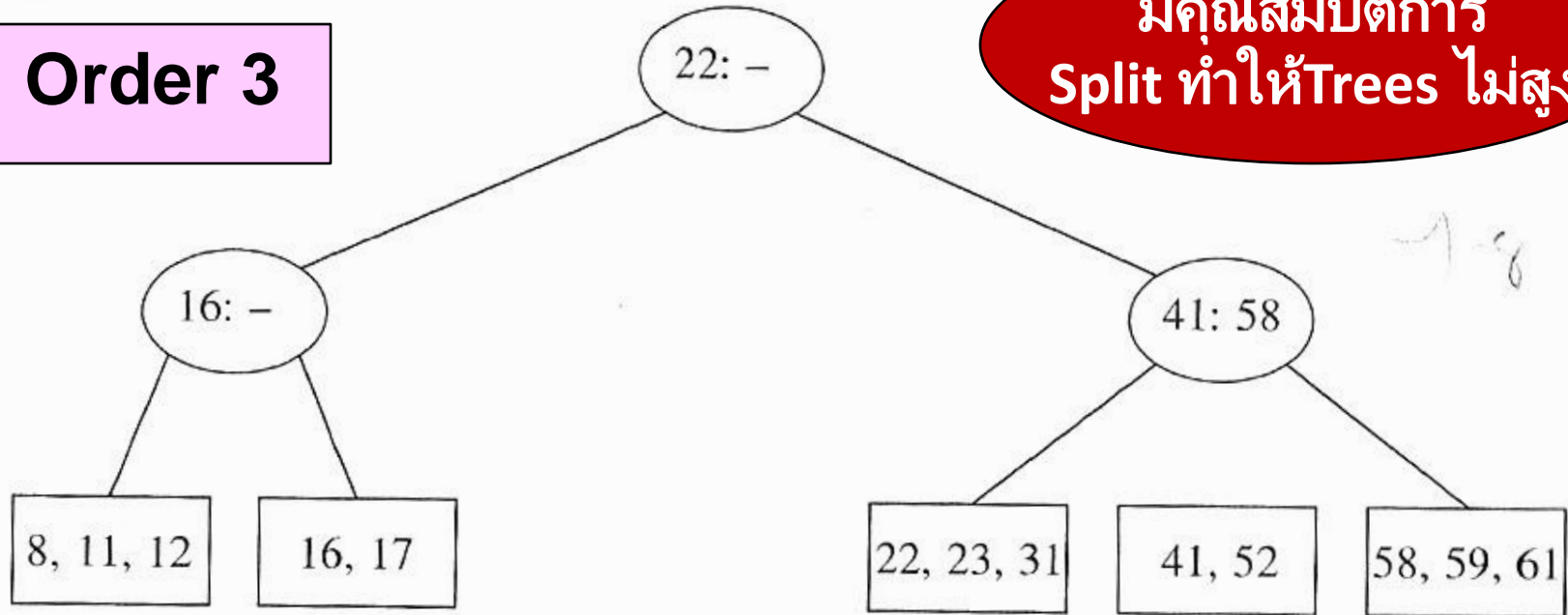
5) All leaves are at the same depth and have between  $\lceil L/2 \rceil$  and  $L$  children.



leaf มีลูกจำนวน  
 $L/2$ -OrderM



**Order 3**



มีคุณสมบัติการ  
Split ทำให้Trees ไม่สูง

**Insert 18, 1, 19, 28**



## Order 5

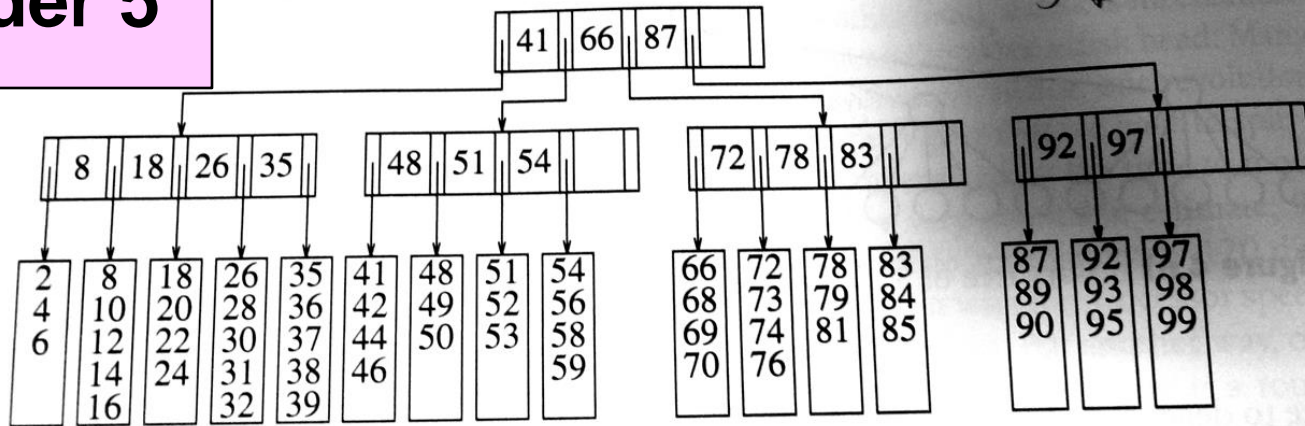
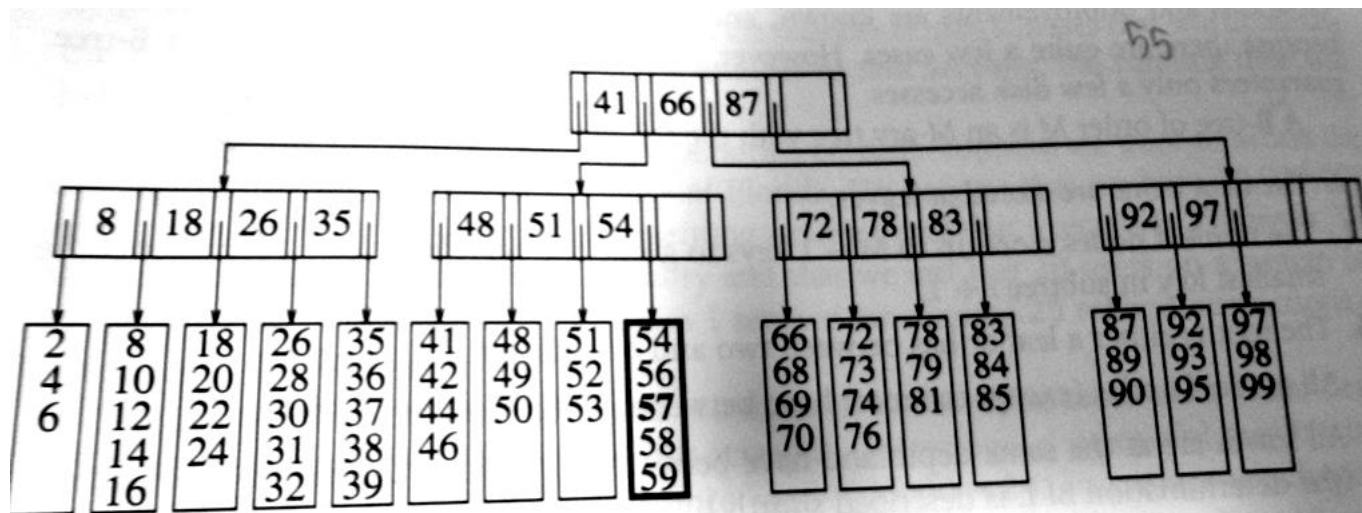


Figure 4.62 B-tree of order 5



57 is the tree in Figure 4.62



