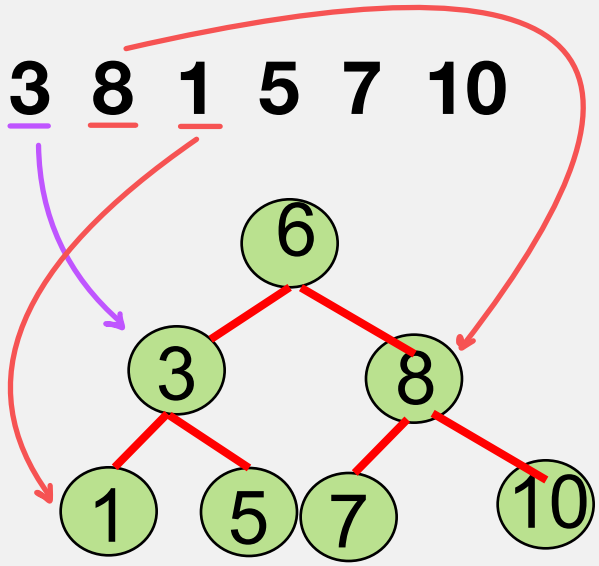
การ insert Binary Search Trees

6 3 8 1 5 7 10

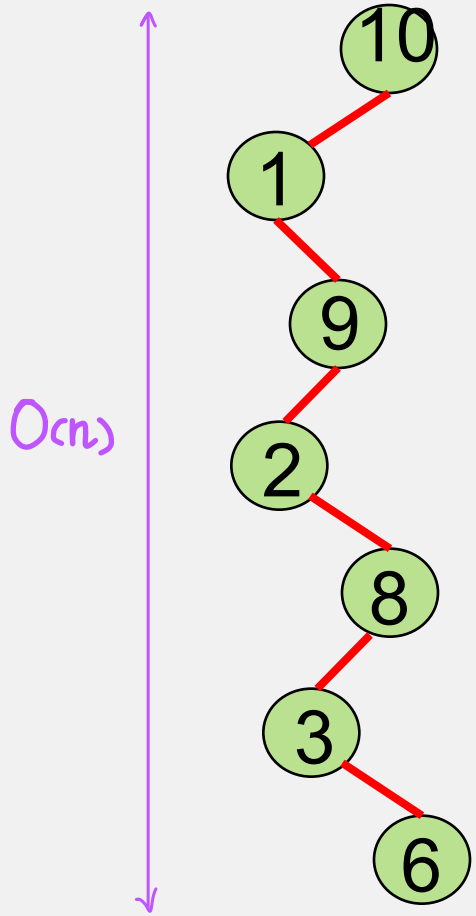


น้อยกว่าไปซ้าย
มากกว่าไปขวา

ตามหลัก
Binary Search tree

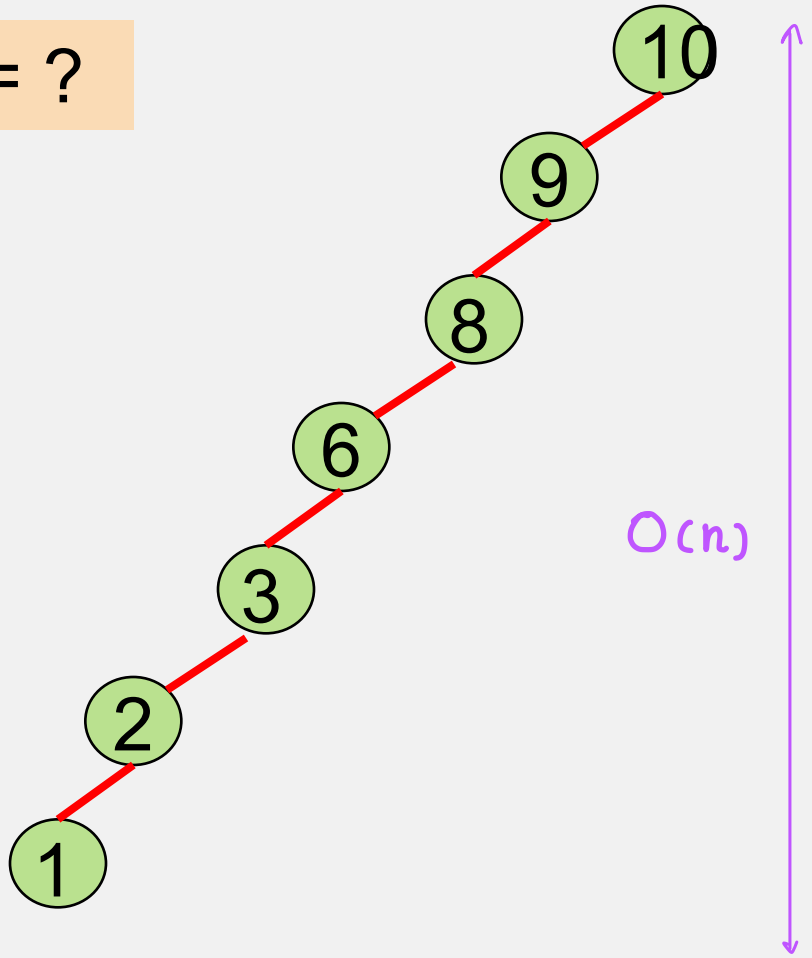
O(log n)

Insert : 10 1 9 2 8 3 6



Big O = ?

Insert : 10 9 8 6 3 2 1



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

every node have not higher than 1

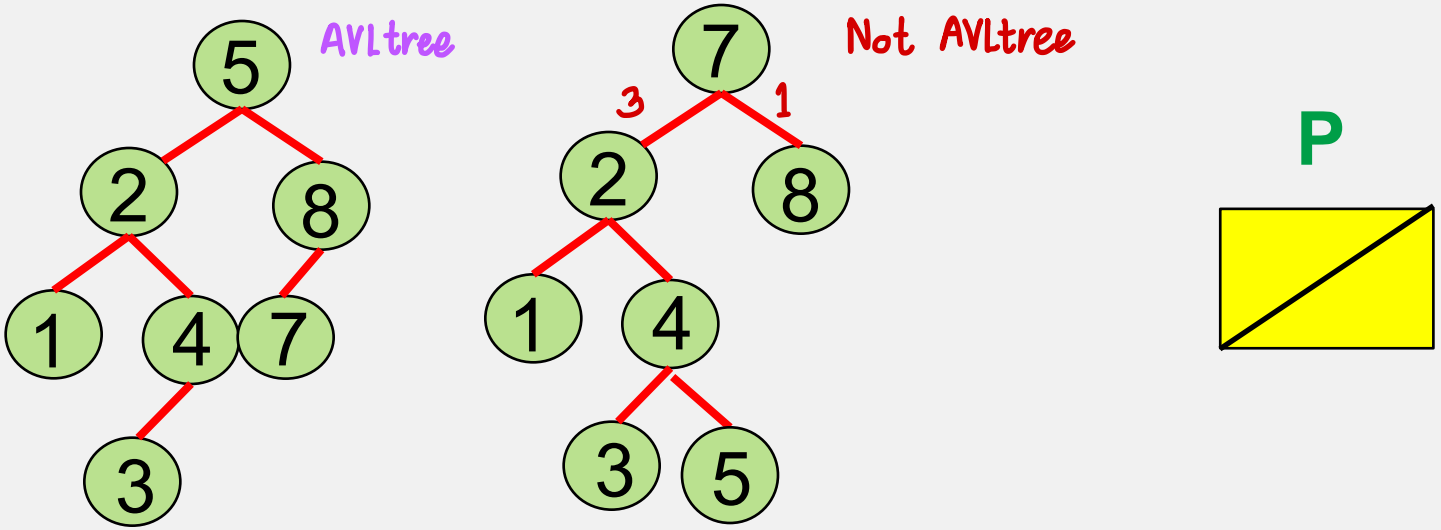
A binary search tree with balance condition, it ensures that the depth of the tree is $O(\log_n)$.

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.** ต่างกันไม่เกิน 1

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



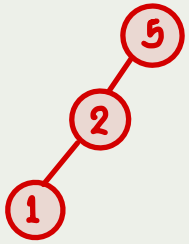
The height of an empty subtree is defined to be -1.

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*

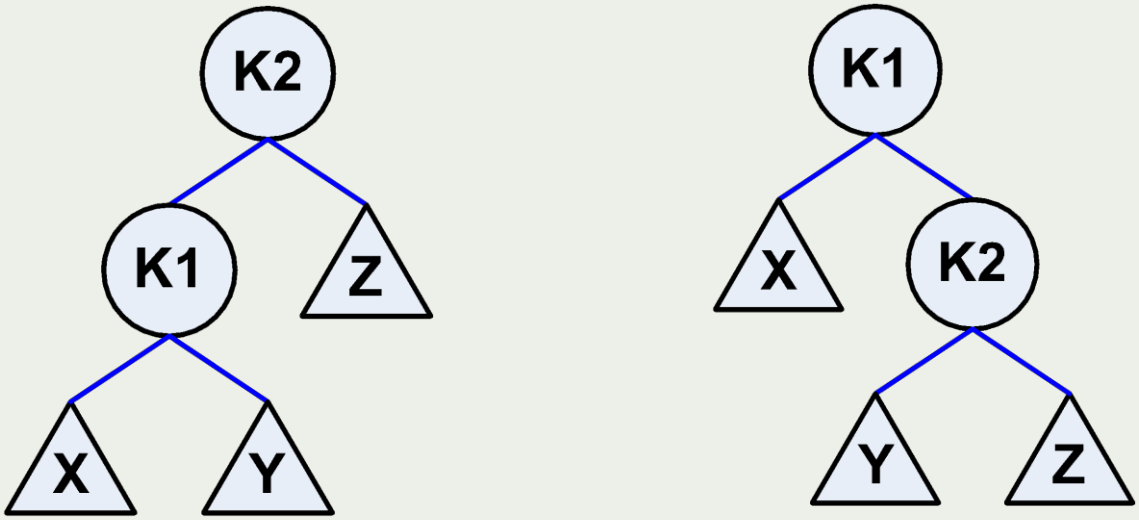
- Single rotation

- Double rotation

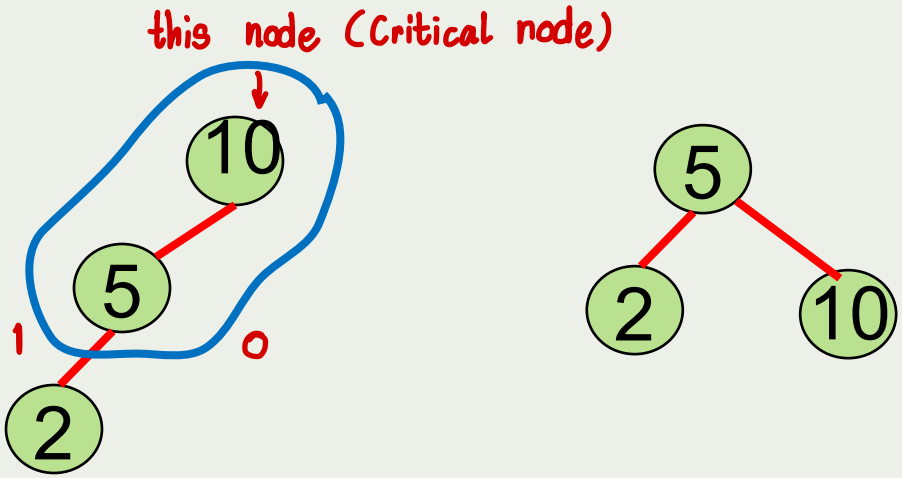


3.4.2 Single Rotation

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

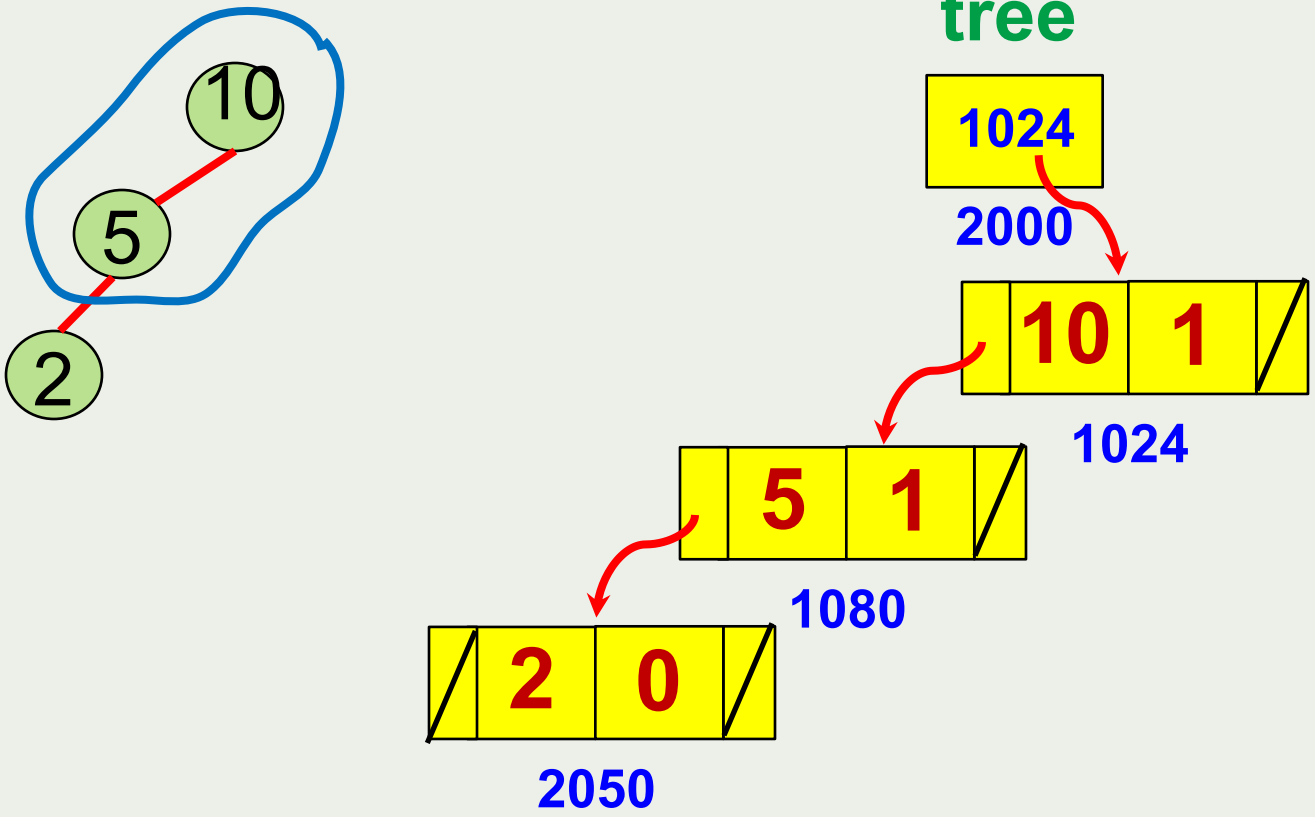


Example insert 10, 5, 2

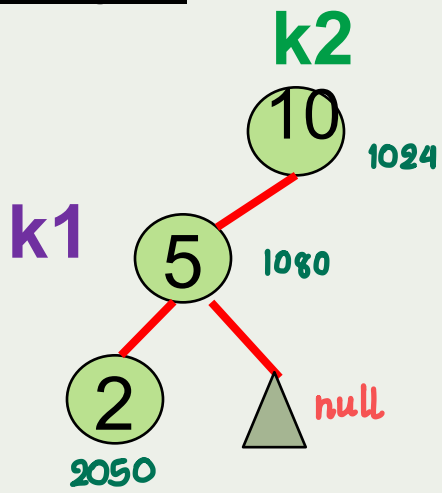


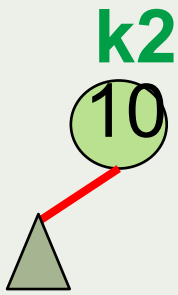
Example insert 10, 5, 2

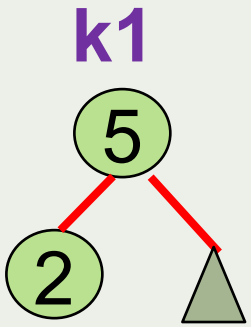
นอกช่วง



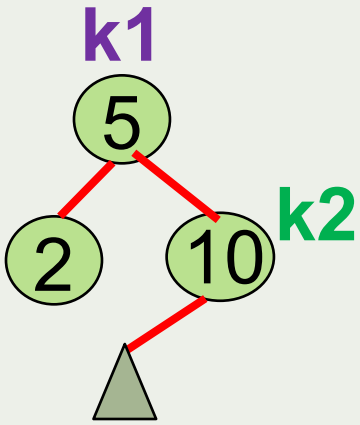
Example insert 10, 5, 2





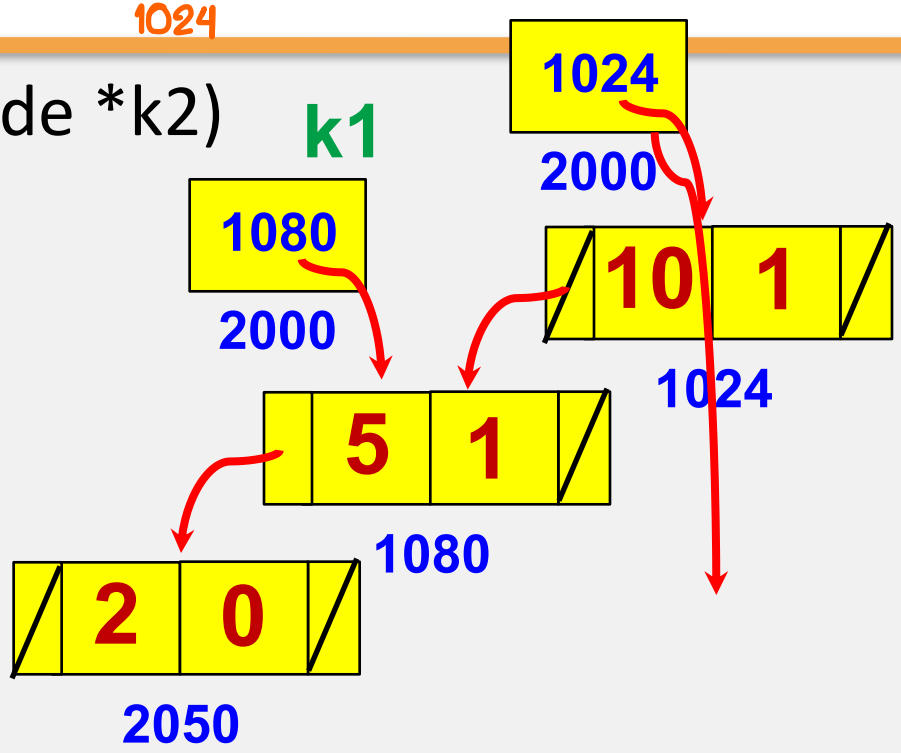


single rotate left




```
struct node *srleft(struct node *k2)
{    struct node *k1;
     k1=k2->left;      // 1080 เกี่ยวลูกซ้าย
     k2->left = k1->right;
     k1->right = k2;
```

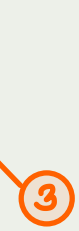
```
struct node *srleft(struct node *k2)
{    struct node *k1;
     k1=k2->left;
     k2->left = k1->right;
     k1->right = k2;


}
```



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

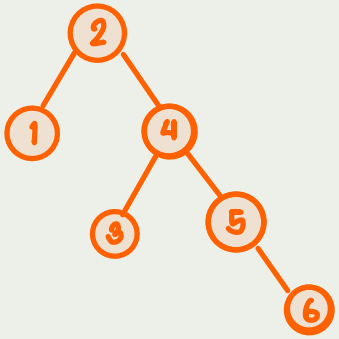
นอกช่วง

1)

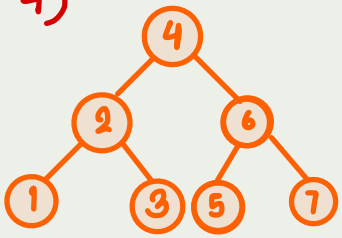


2)

3)



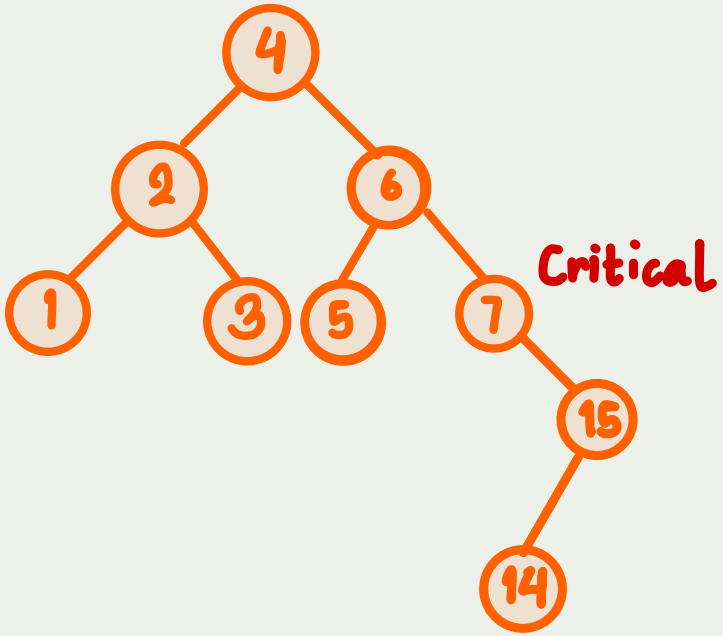
4)

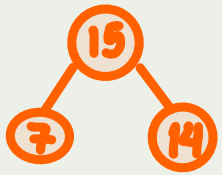


Example insert 15, 14

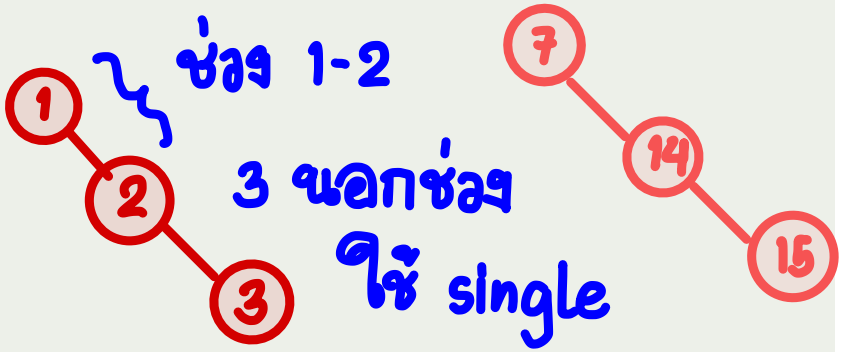
insert 13, 12, 11, 10, 9, 8, 8.5

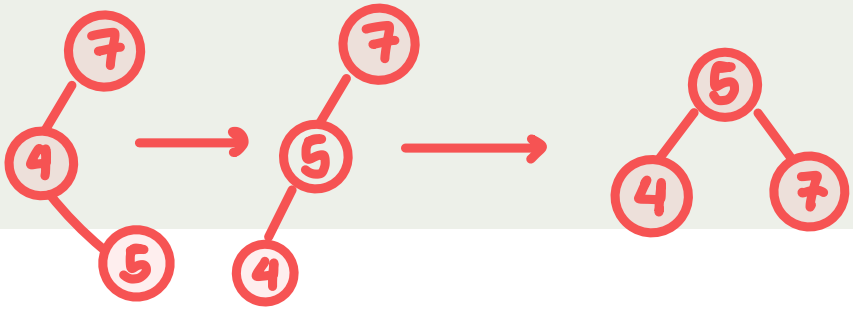
นอกช่วง

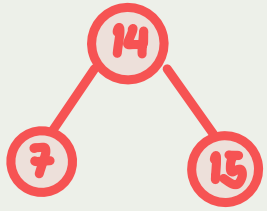




ผิด 14 < 15



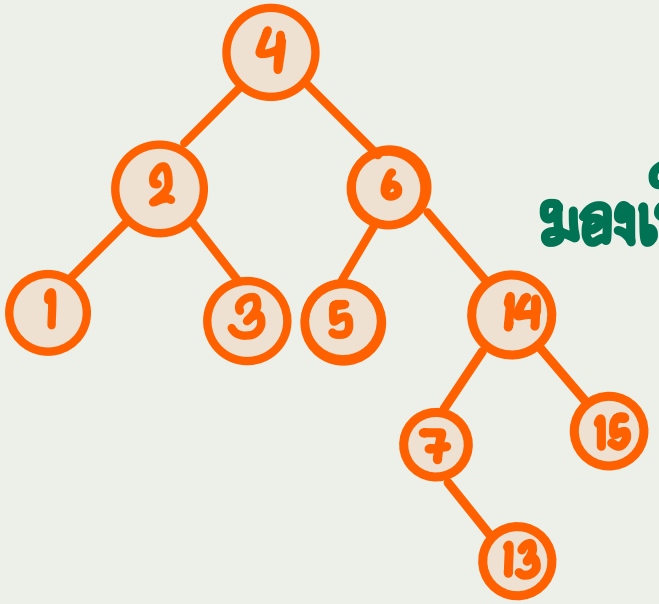




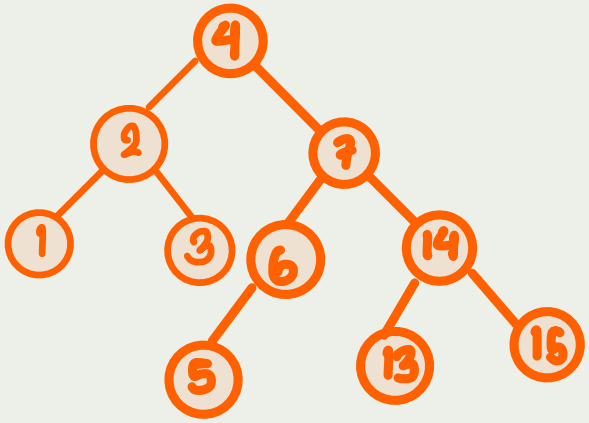
Example insert 15, 14

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

นอกช่วง



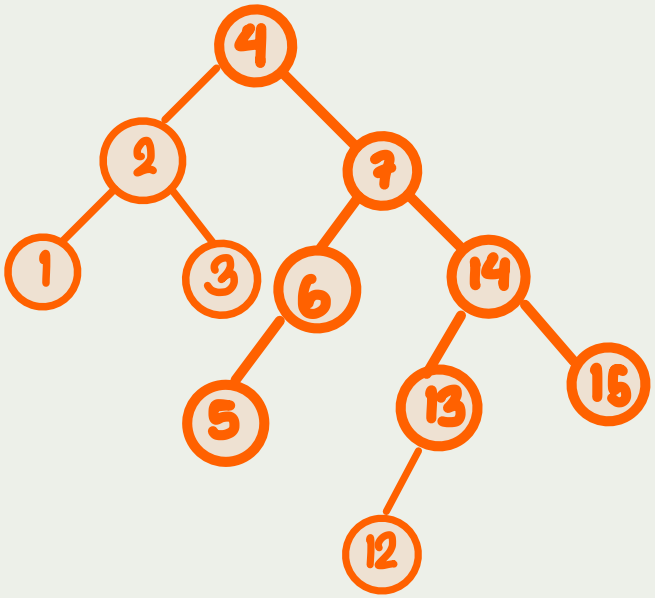
มองเป็นฮีปซิกแซก

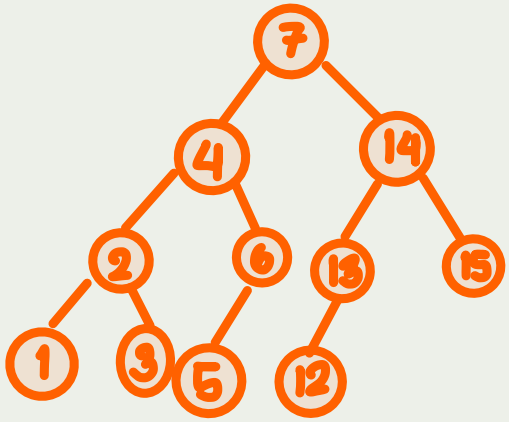


Example insert 15, 14

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

นอกช่วง

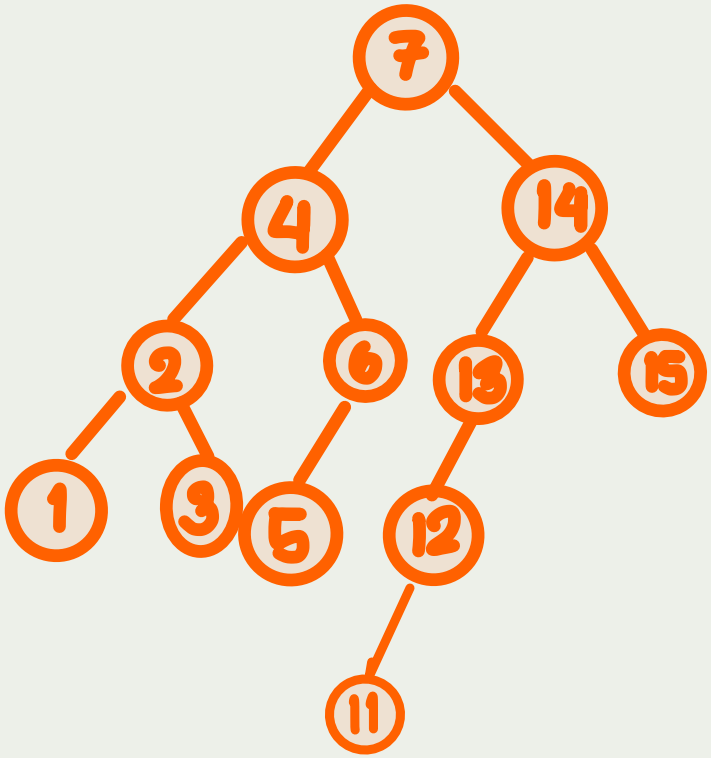


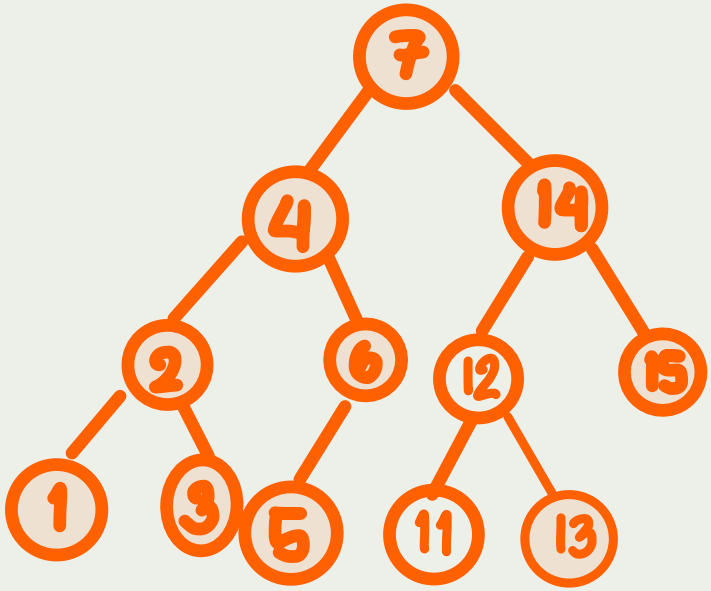


Example insert 15, 14

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

นอกช่วง

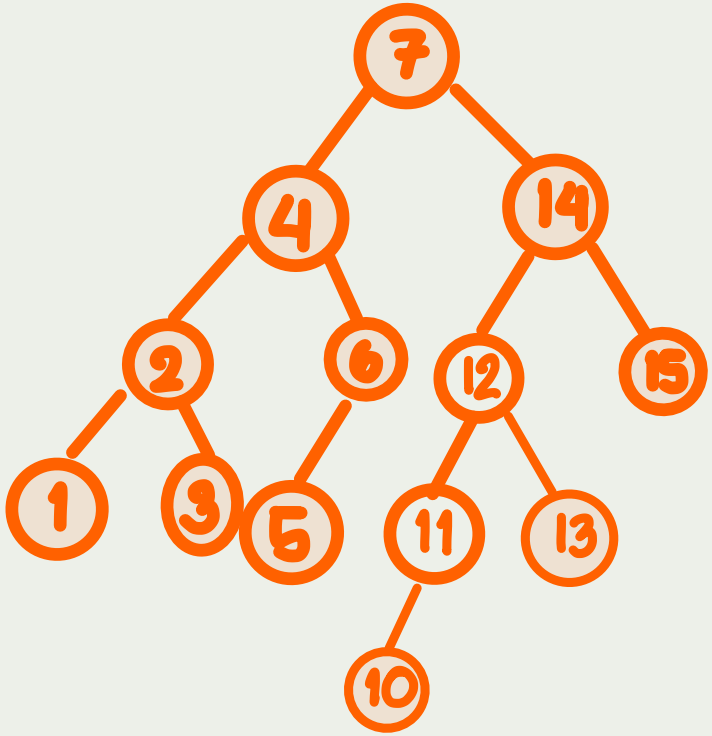


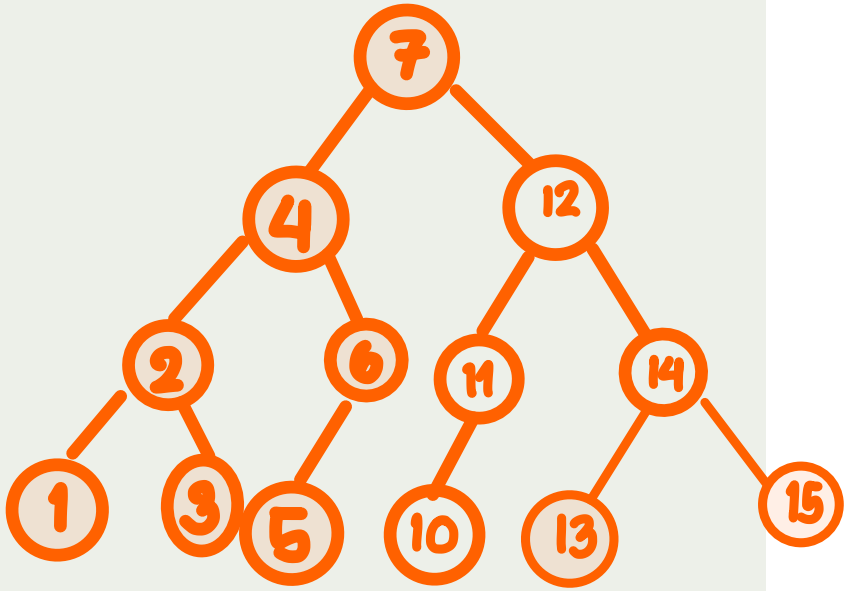


Example insert 15, 14

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

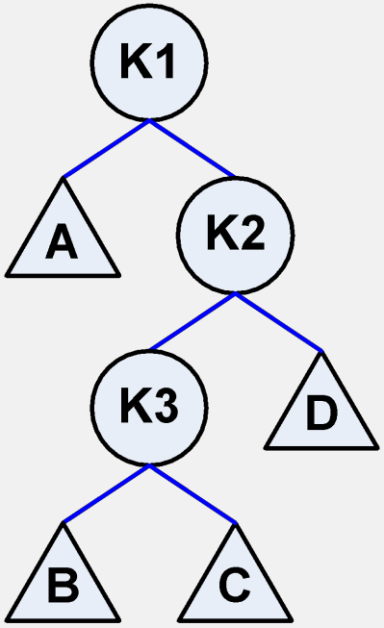
นอกช่วง

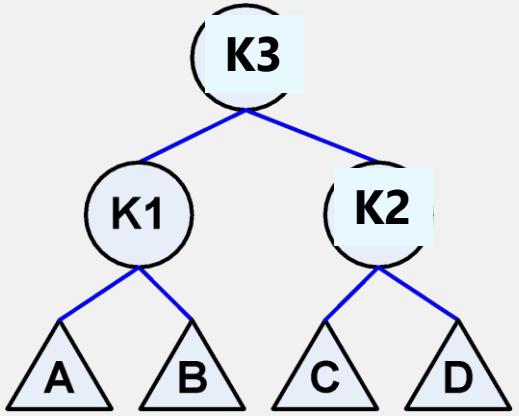


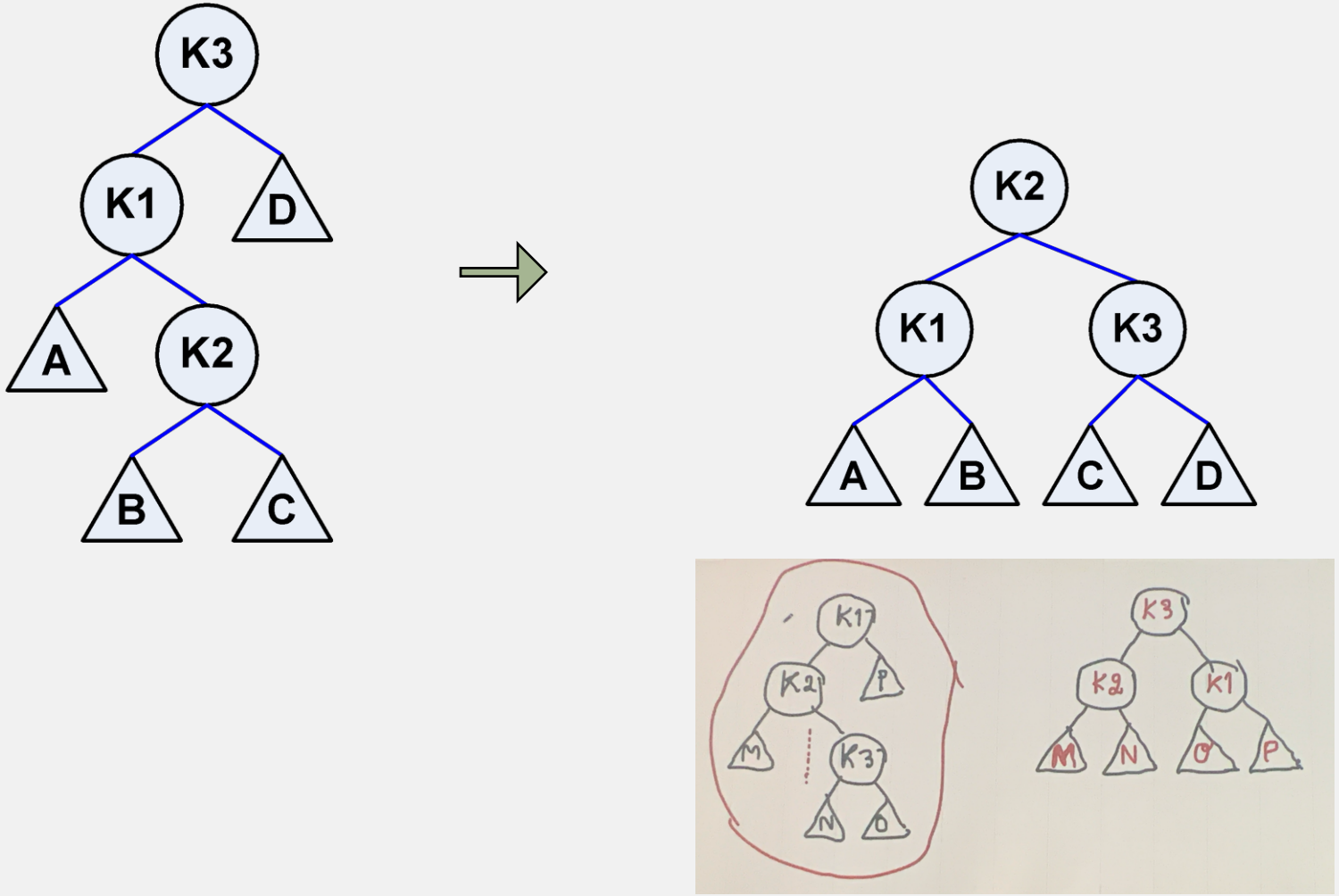


3.4.3 Double rotation

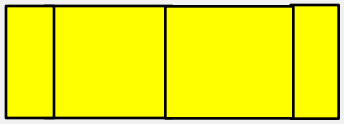
Problem : Single rotation has not fixed the height imbalance by a node inserted into the tree containing the middle elements.



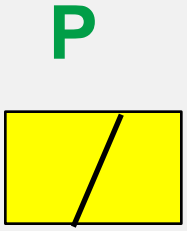


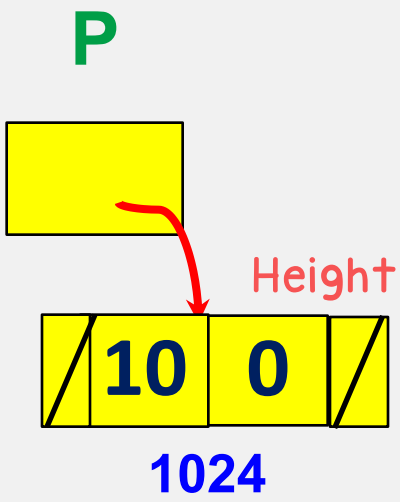

K3
K1
D
A
K2
B
C
K2
K1
K3
A
B
C
D
K1
K2
P
M
K3
N
O
K3
K2
K1
M
N
O
P

```
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;
6.   }
```



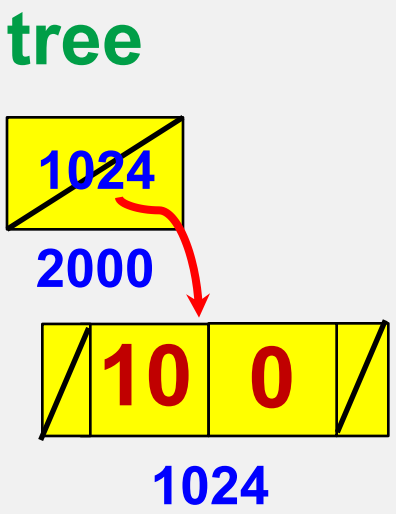


Code Insert

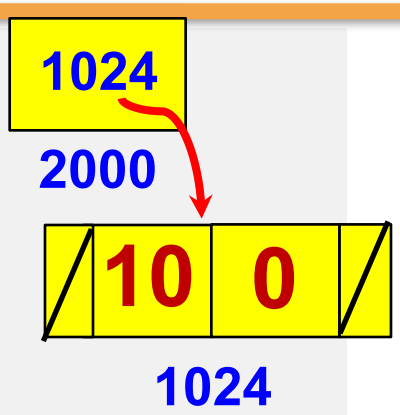
```
1 ถ้า tree == NULL
2     สร้าง node ใหม่ ใส่ค่า

3 else
4     ถ้า x น้อยกว่า tree->value  ให้ recursive ลงไปทางซ้าย
5         tree->left = insert (tree->left)
6     ถ้าความสูงของทรีทางซ้ายและทางขวาต่างกันเกิน 2
7         ถ้า x น้อยกว่า tree->left->value
8             single rotation  (อยู่นอกช่วง)
9         else
10            double rotation (อยู่ในช่วง)

11 update ความสูง
12 return tree
```

เมื่อ insert แล้ว return กลับมาเช็คความสูง

```
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
```



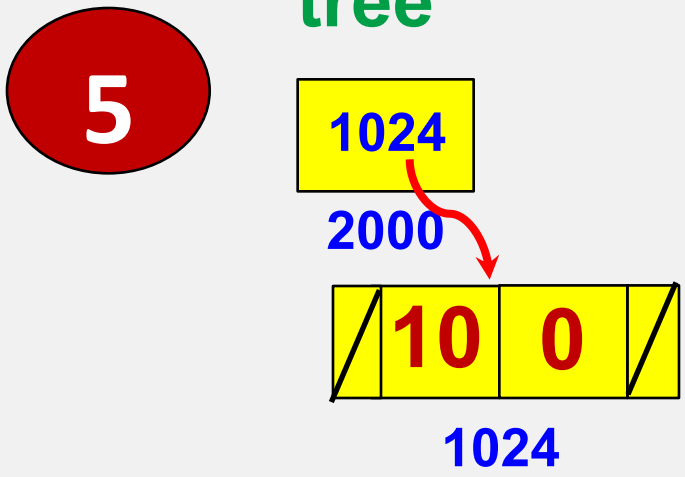
tree




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

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

max(-1,-1) = -1 +1

```
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
```

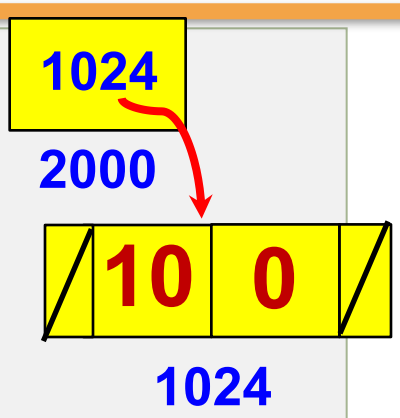


ถ้า x น้อยกว่า tree->value
ให้ recursive ลงไปทางซ้าย

5, NULL

5

tree

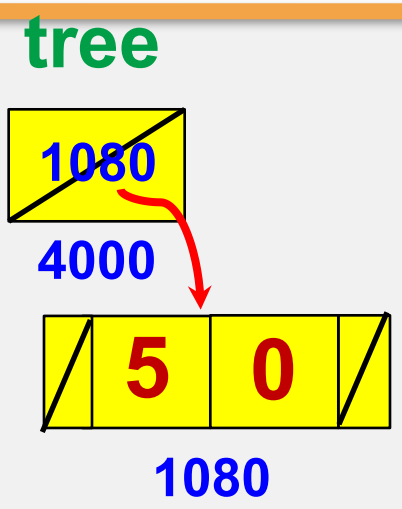



```
8 if(x < T->value )
9       {   T->left = insert(x,T->left);
10           if(fheight(T->left) - fheight(T->right) == 2)
11               if( x < T->left->value)
12                  T=srleft(T);
13               else
14                   T=drleft(T);
15      }

16else
17ถึง 29 if(x > T->value )
          {  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(fheight(T->left), fheight(T->right)) + 1;
31    return T;
 }
```



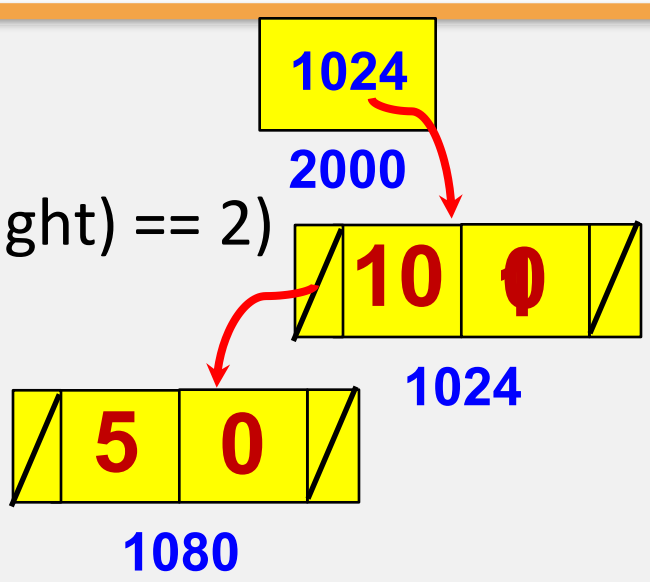
max(-1,-1) = -1 +1

1080

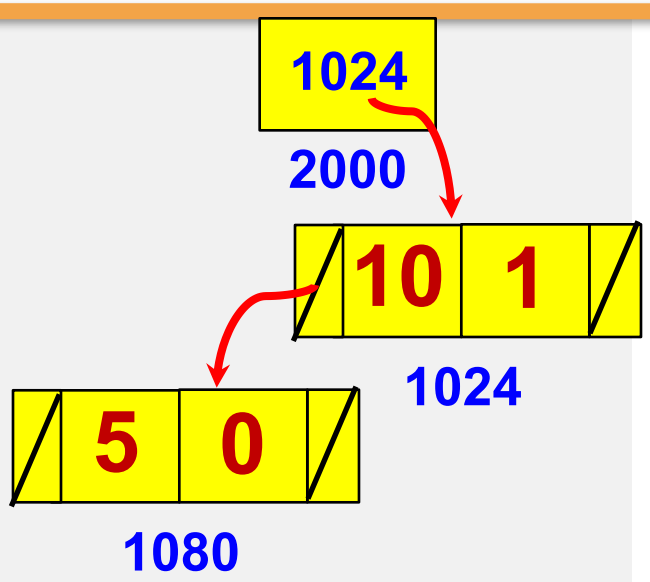
ค้างบรรทัด 9 T 1024

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      }
16ถึง 29 if(x > T->value )
          {  ข้างขวาบ้าง  ให้เขียนเอง … }
30   T->height = max(fheight(T->left), fheight(T->right)) + 1;
31    return T;
 }
```



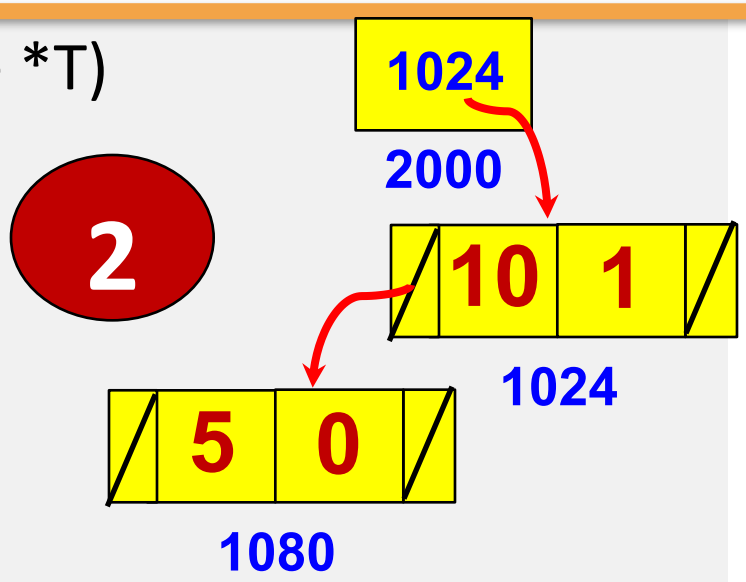
```
main()
{
    ……
    tree=insert(2,tree);
```



จาก main

2, 1024

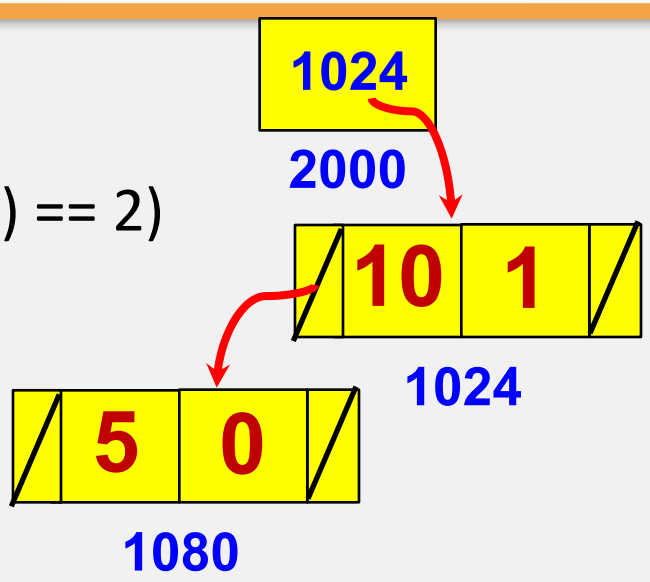
```
struct node *insert(int x, struct node *T)
1{     if(T == NULL)
2      {    T=new struct node;
3           T->value=x;
4           T->left=T->right=NULL;
5           T->height=0;
       }
7   else
              .....
```



ถ้า x น้อยกว่า tree->value
ให้ recursive ลงไปทางซ้าย

2, 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       }
16ถึง 29 if(x > T->value )
          {  ข้างขวาบ้าง  ให้เขียนเอง ... }
30   T->height = max(fheight(T->left), fheight(T->right)) + 1;
31    return T;
 }
```

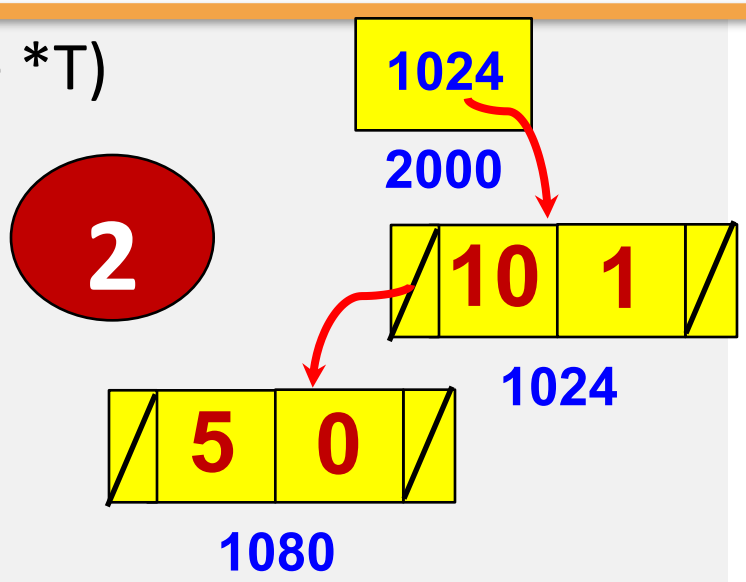


(A) T = 1024 ค้าง 9 หน้า 26

Recursive 1

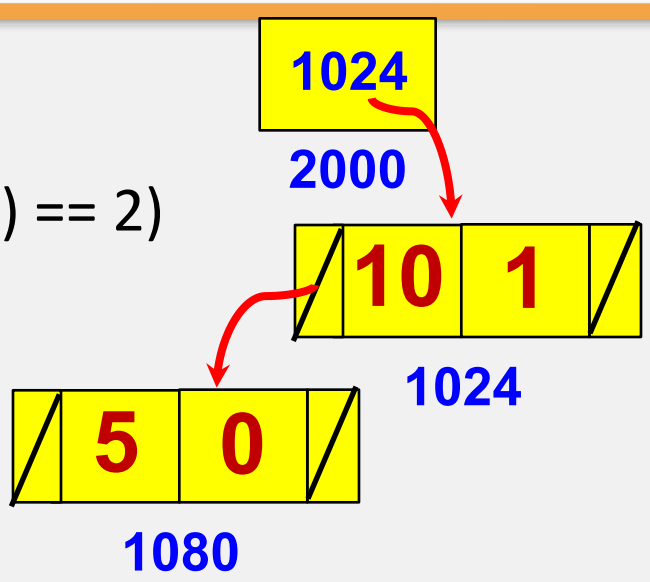
2, 1080

```
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;
       }
7   else
            .....
```



2, NULL

```
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       }
16ถึง 29 if(x > T->value )
          {  ข้างขวาบ้าง  ให้เขียนเอง … }
30   T->height = max(fheight(T->left), fheight(T->right)) + 1;
31    return T;
 }
```



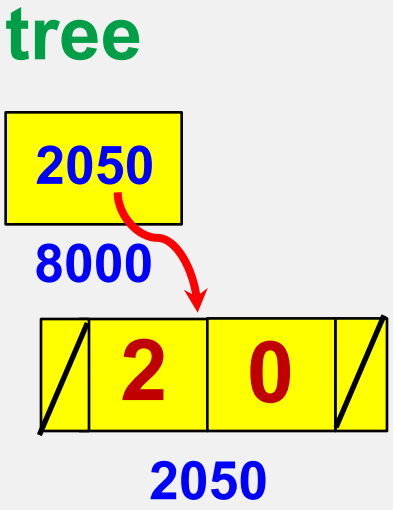
(B) T = 1080 ค้างบรรทัด 9 หน้า 28

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;
       }
7   else
            .....

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

2

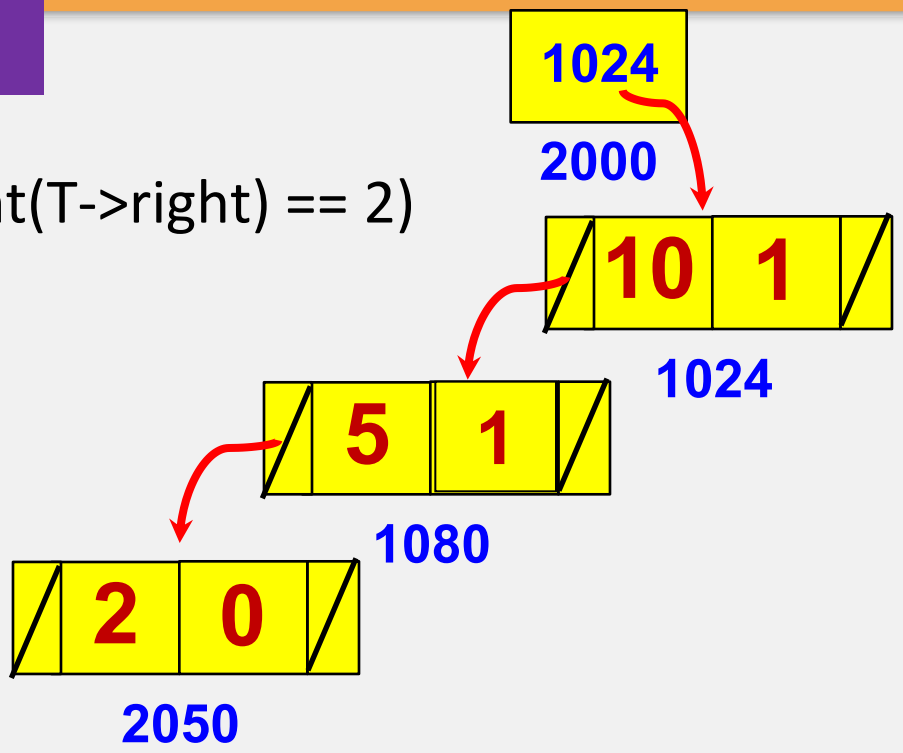


กลับไปที่ **(B) T = 1080 ค้างบรรทัด 9**

(B) T = 1080 ค้างบรรทัด 9

2050

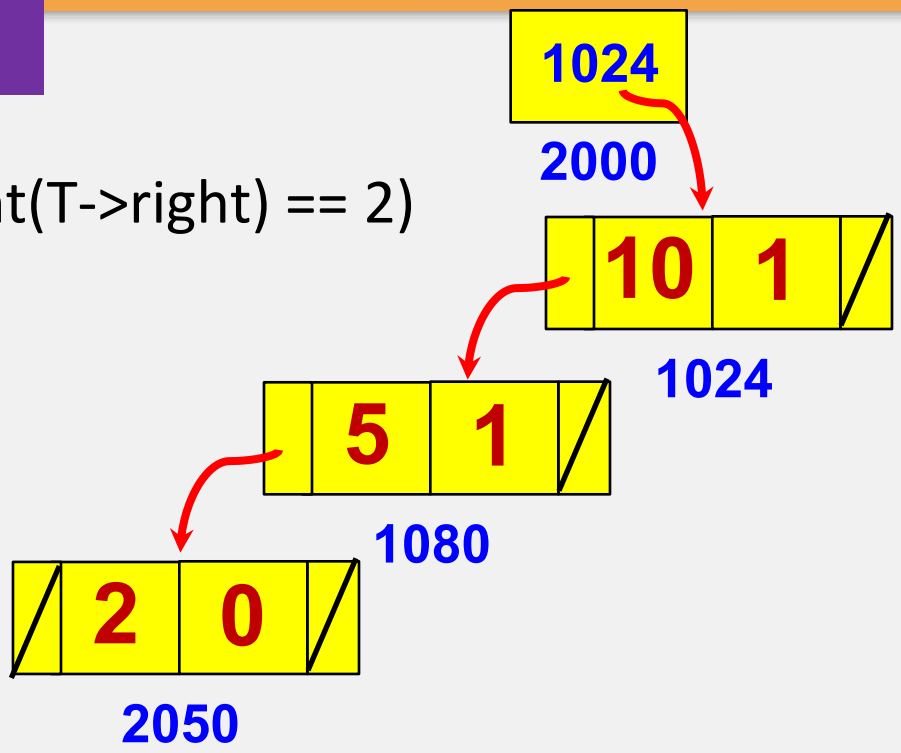
```
8 if(x < T->value )
9       {   T->left = insert(x,T->left);
10           if(fheight(T->left) - fheight(T->right) == 2)
11               if( x < T->left->value)
12                  T=srleft(T);
13               else
14                   T=drleft(T);
15      }
16ถึง 29 if(x > T->value )
        {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



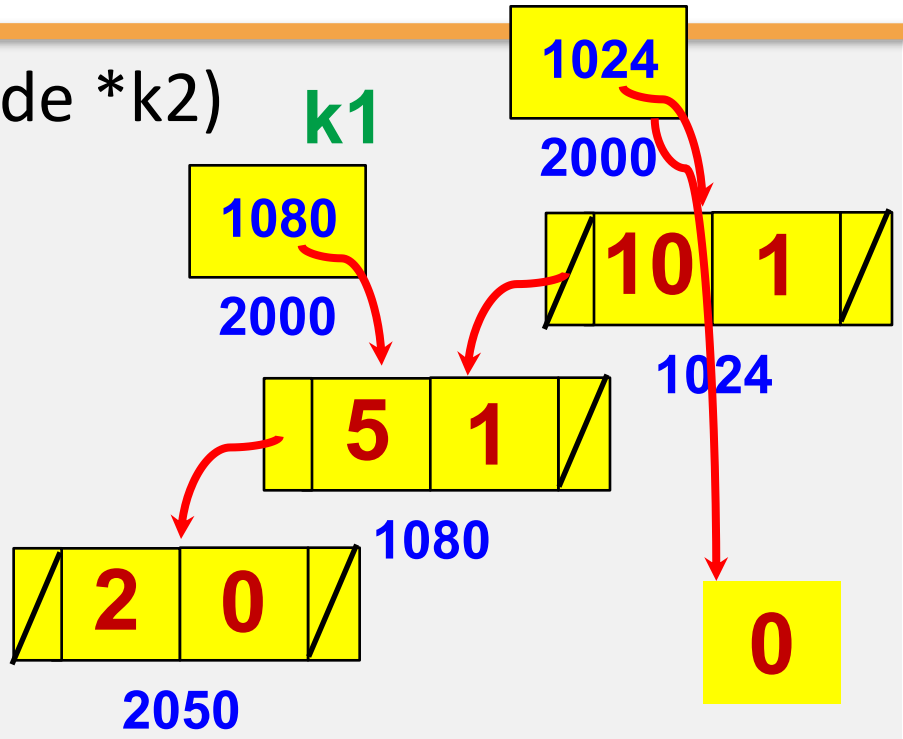
(A) ค้างบรรทัด 9 1024

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       }
16ถึง 29 if(x > T->value )
         { ข้างขวาบ้าง ให้เขียนเอง ... }
30   T->height = max(fheight(T->left), fheight(T->right)) + 1;
31    return T;
 }
```



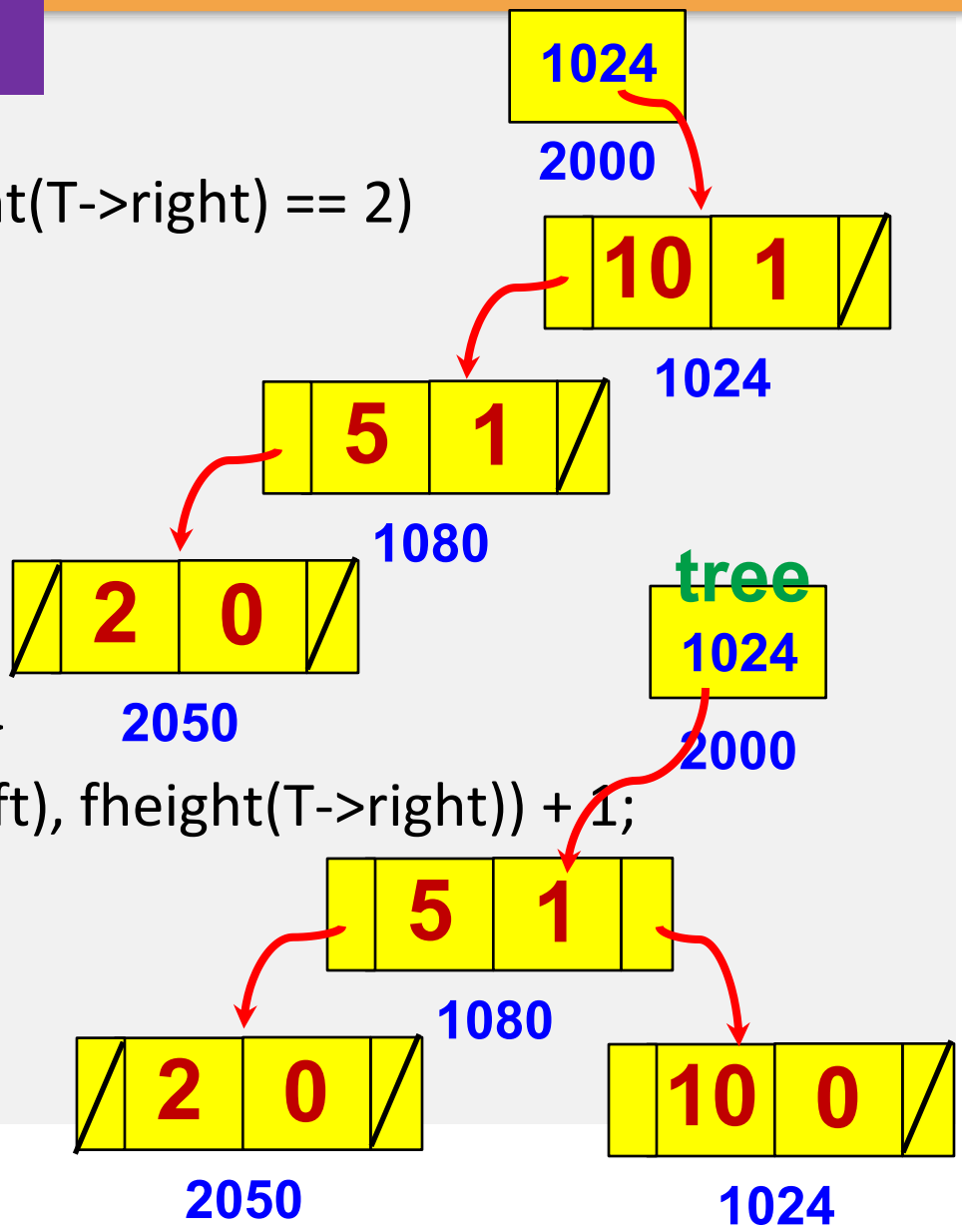
```
struct node *srleft(struct node *k2)
{    struct node *k1;
     k1=k2->left;
     k2->left = k1->right;
     k1->right = k2;
```




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

1080

```
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      }
16ถึง 29 if(x > T->value )
        { ข้างขวาบ้าง ให้เขียนเอง ... }
30   T->height = max(fheight(T->left), fheight(T->right)) + 1;
31    return T;
 }
```

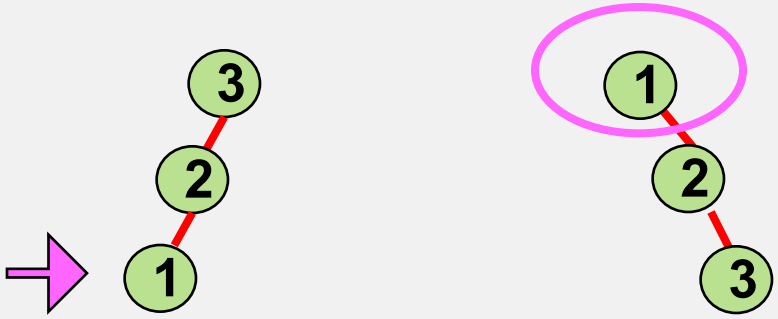


```
struct node *drleft(struct node *k3)
{    k3->left = srright(k3->left);
     return srleft(k3);
}
```

มีทรีอยู่แล้ว
แต่ไม่
Balance

3.5 Splay Trees

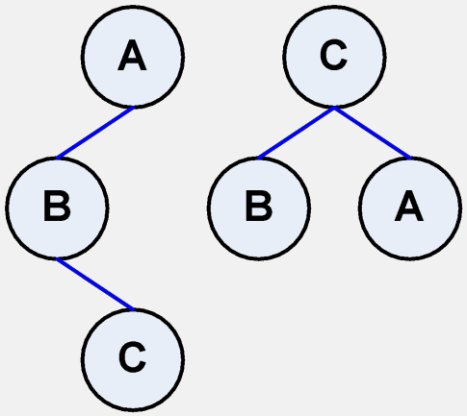
- O(N) worst-case time per operation for binary search trees.
- After a node is accessed, it is pushed to the root



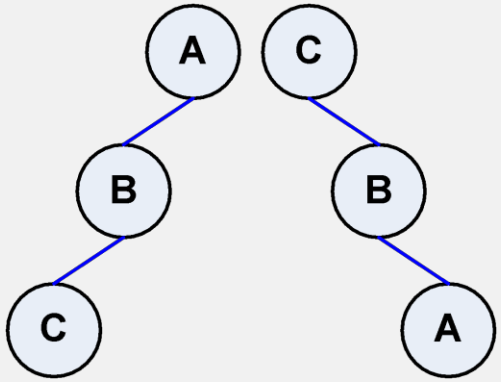
Rotation idea

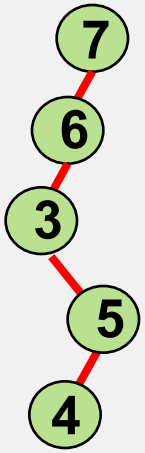
Transform

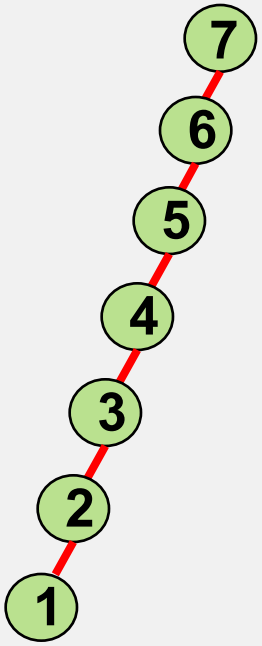
1. zig-zag



2. zig-zig



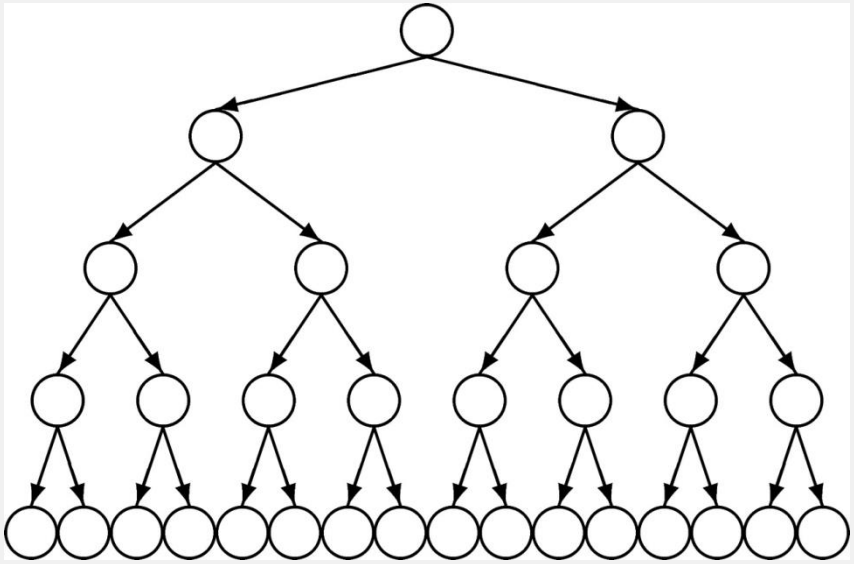

7
6
3
5
4

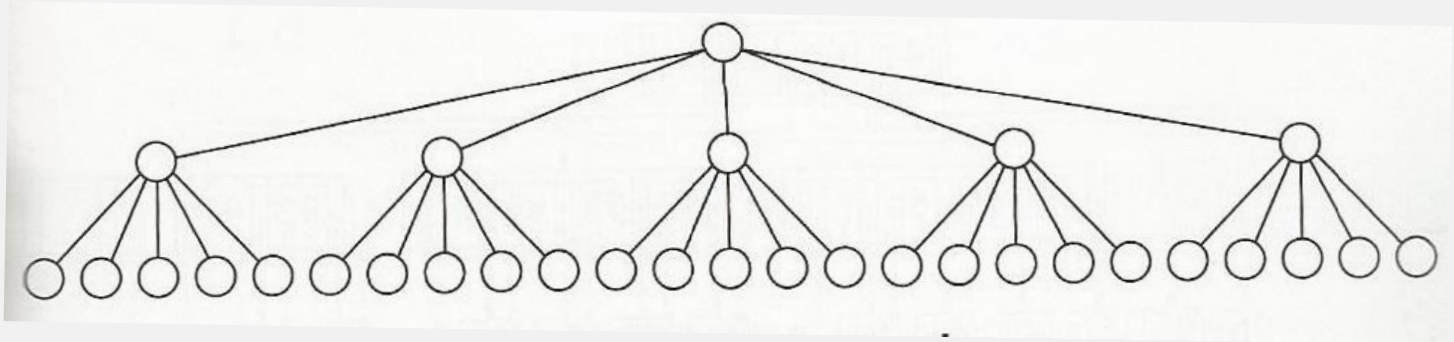

7
6
5
4
3
2
1

3.6 B-Trees

Complete binary

Tree 31 nodes



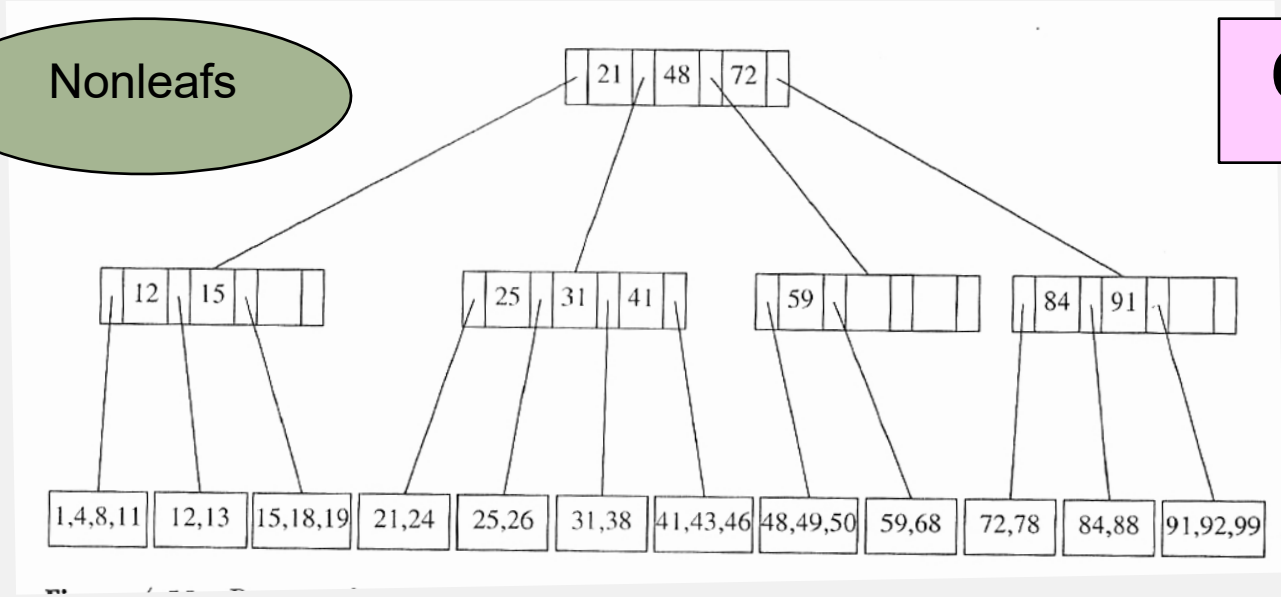


- สำหรับข้อมูลขนาดใหญ่
- ส่วนใหญ่ใช้ในระบบฐานข้อมูลและระบบไฟล์

3.6 B-Trees

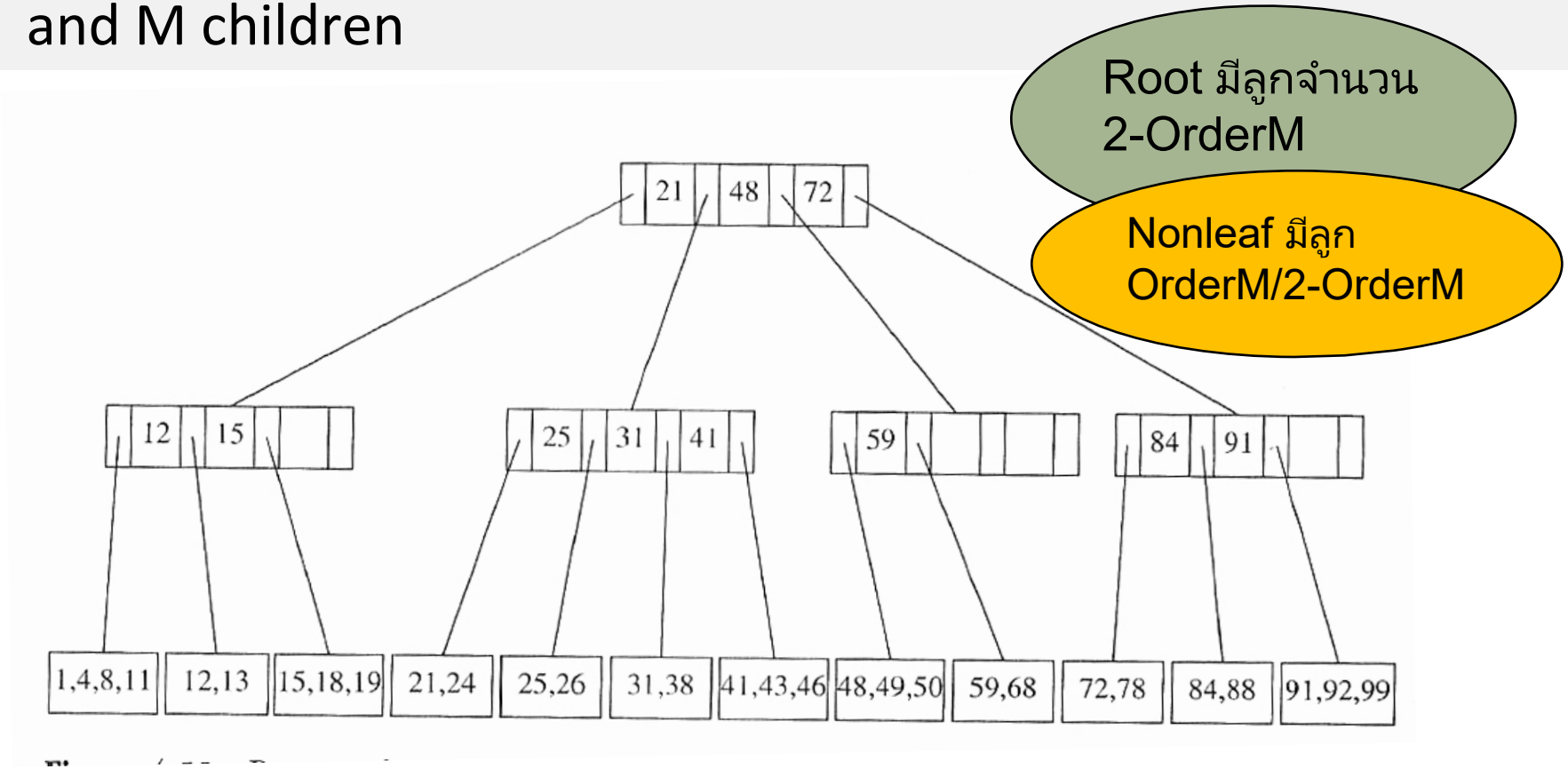
A B-tree of ***order m*** is a tree with the following structural properties:

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

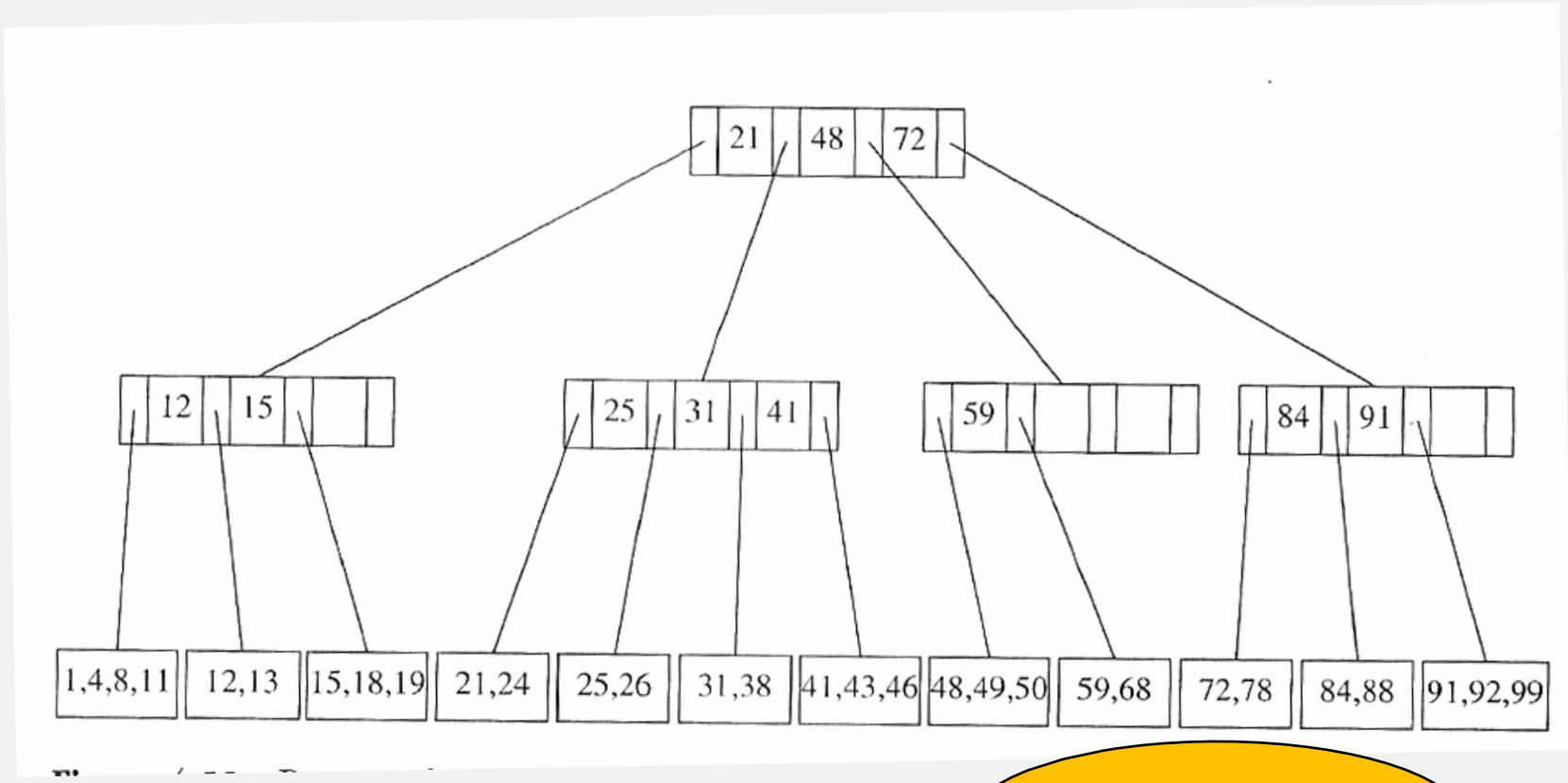


3) The root is either a leaf or has between two and M children.

4) All nonleaf nodes(Except the root) have between ⌈M/2⌉ and M children



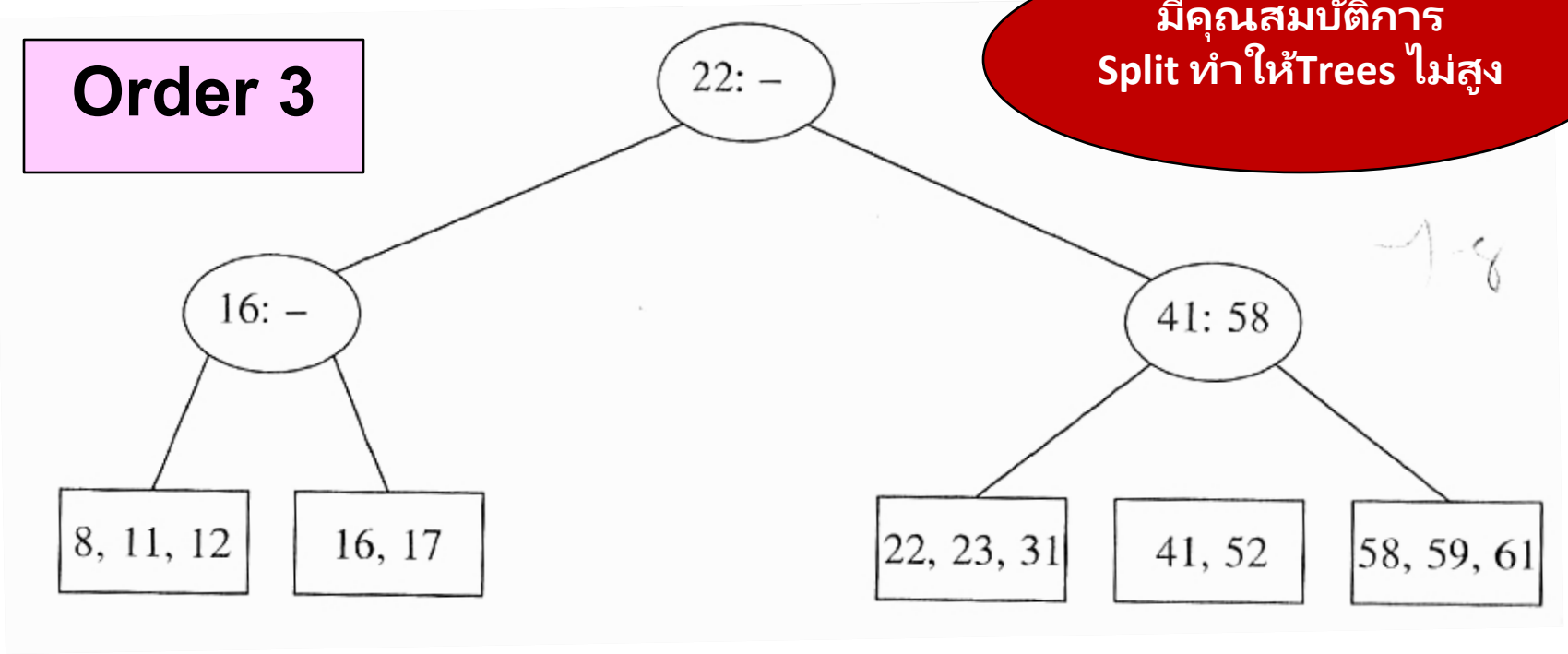
5) All leaves are at the same depth and have between ⌈L/2⌉ and L children.



leaf มีลูกจำนวน L/2-OrderM

Order 3

มีคุณสมบัติการ
Split ทำให้Trees ไม่สูง



Insert 18, 1, 19, 28

Order 5

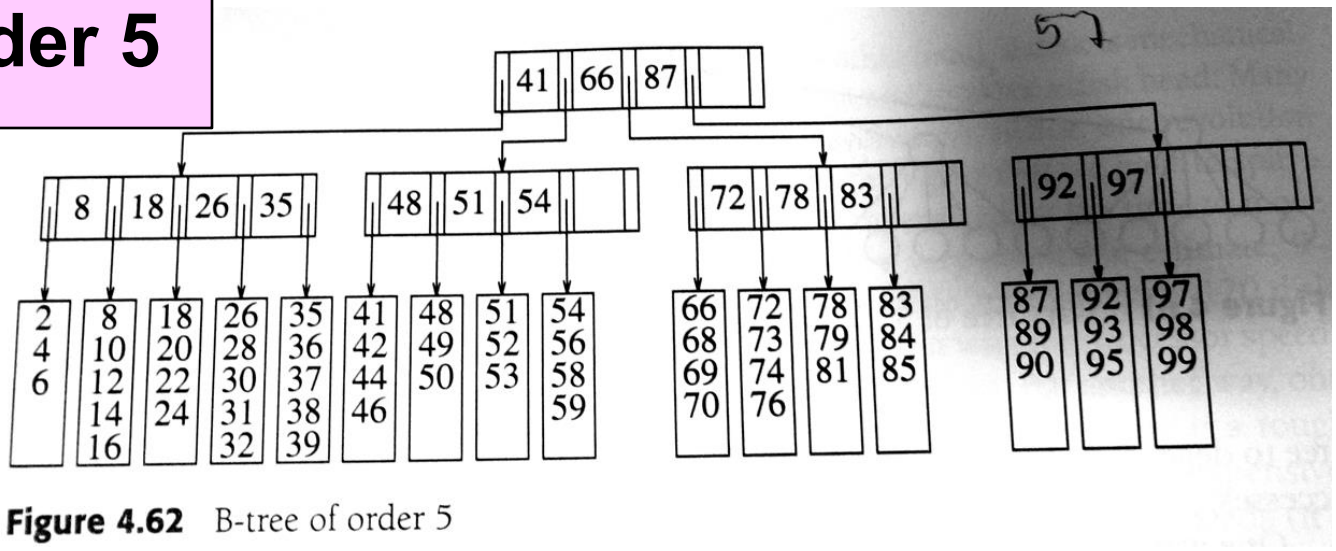
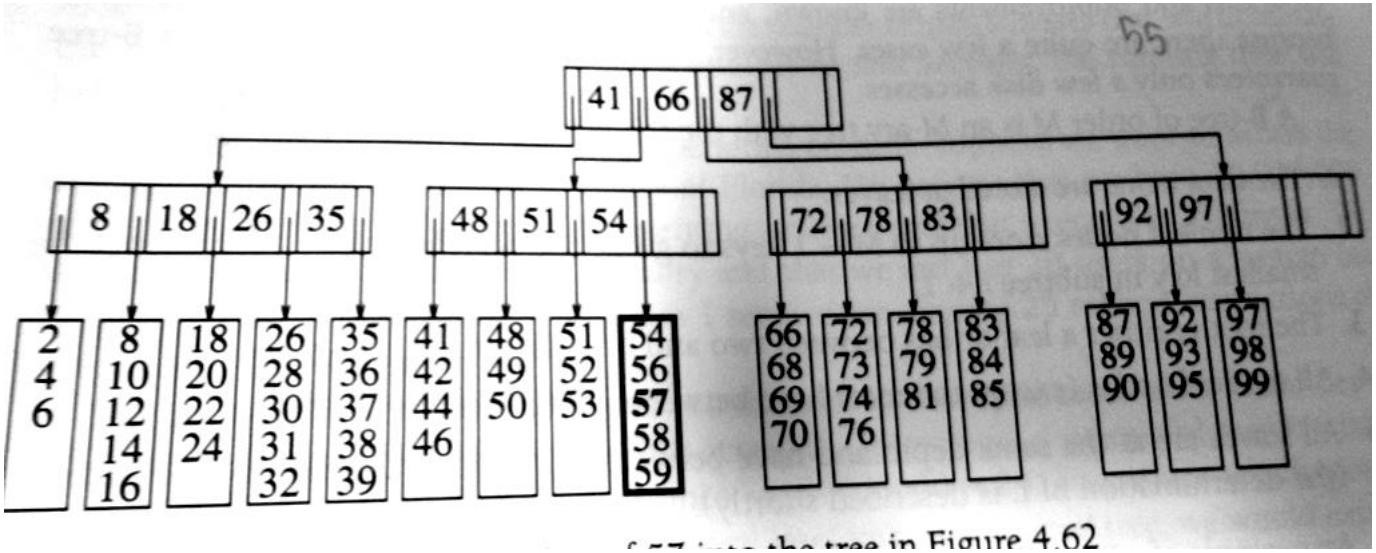
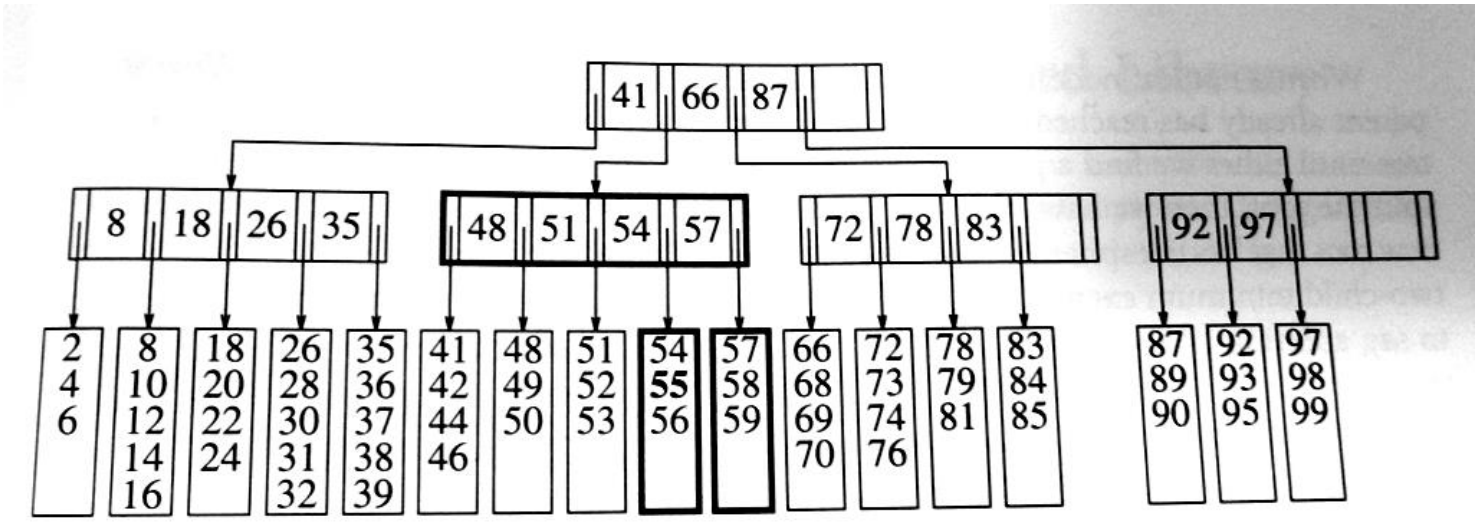
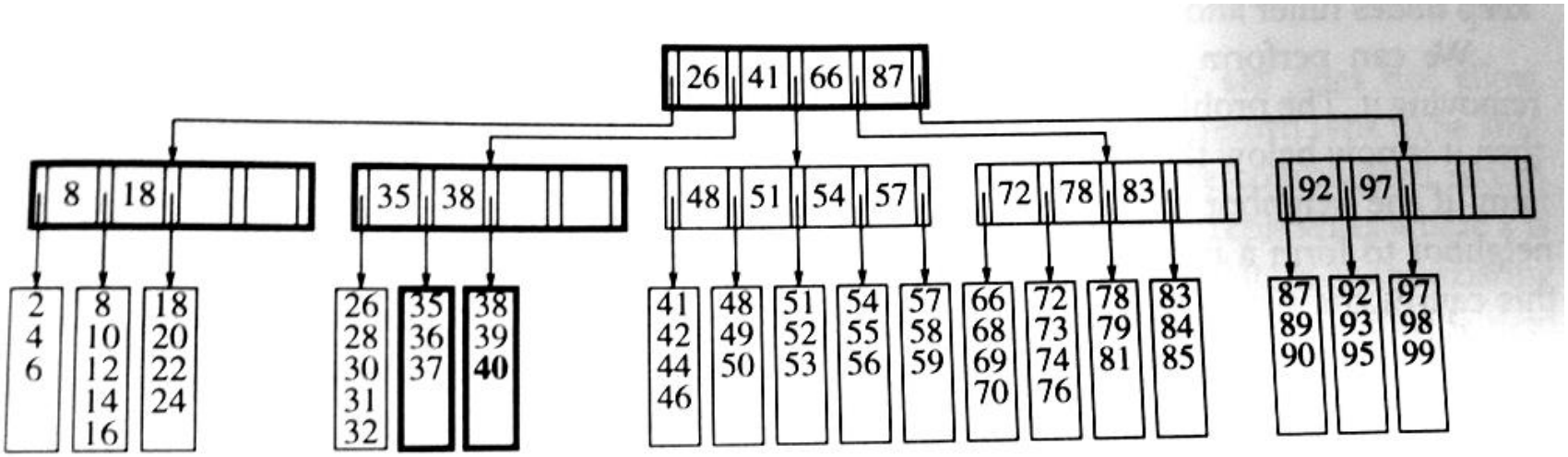


Figure 4.62 B-tree of order 5




41 66 87
8 18 26 35
48 51 54 57
72 78 83
92 97
2 4 6
8 10 12 14 16
18 20 22 24
26 28 30 31 32
35 36 37 38 39
41 42 44 46
48 49 50
51 52 53
54 55 56
57 58 59
66 68 69 70
72 73 74 76
78 79 81
83 84 85
87 89 90
92 93 95
97 98 99


26 41 66 87
8 18
35 38
48 51 54 57
72 78 83
92 97
2 4 6
8 10 12 14 16
18 20 22 24
26 28 30 31 32
35 36 37
38 39 40
41 42 44 46
48 49 50
51 52 53
54 55 56
57 58 59
66 68 69 70
72 73 74 76
78 79 81
83 84 85
87 89 90
92 93 95
97 98 99