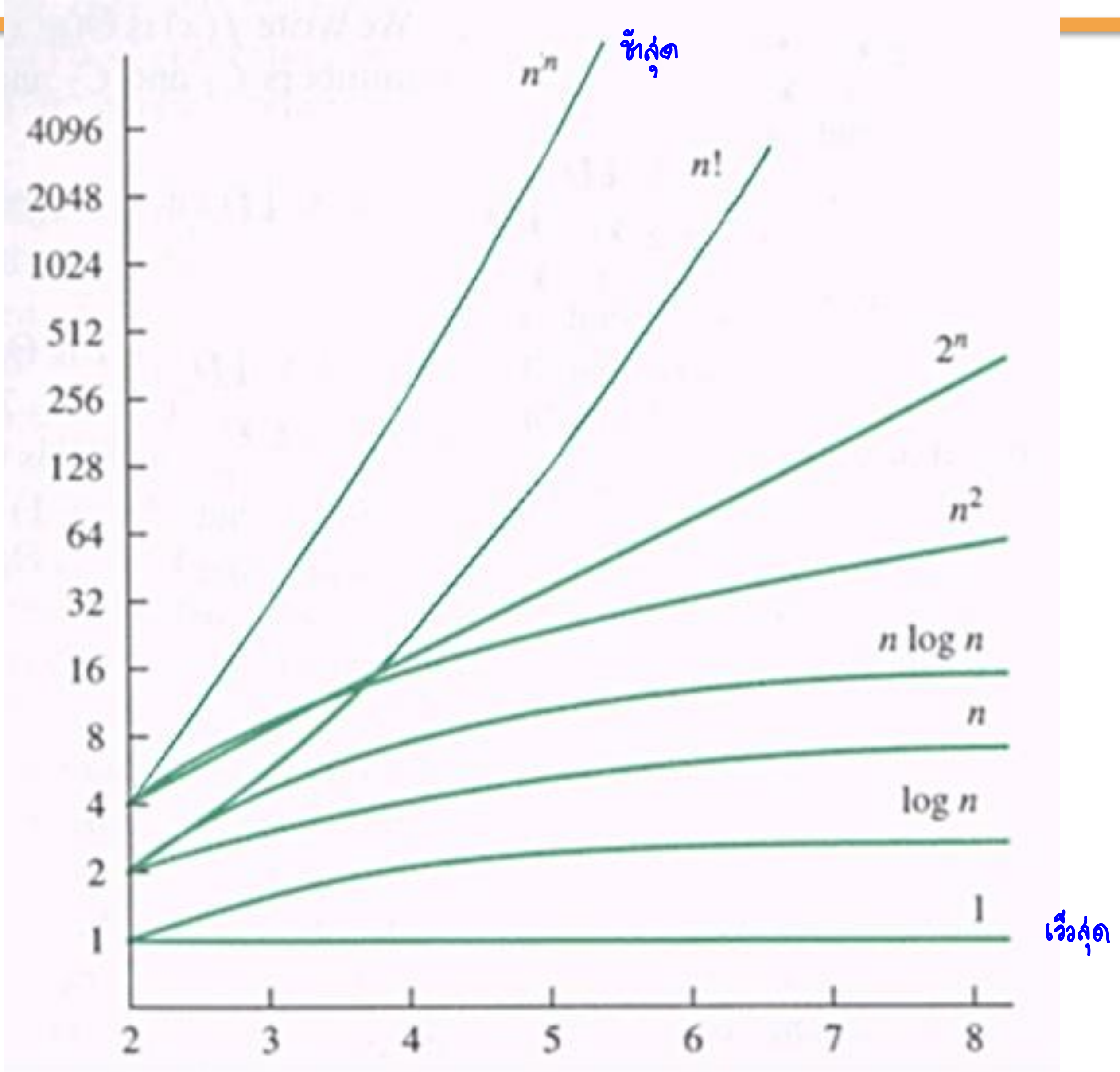
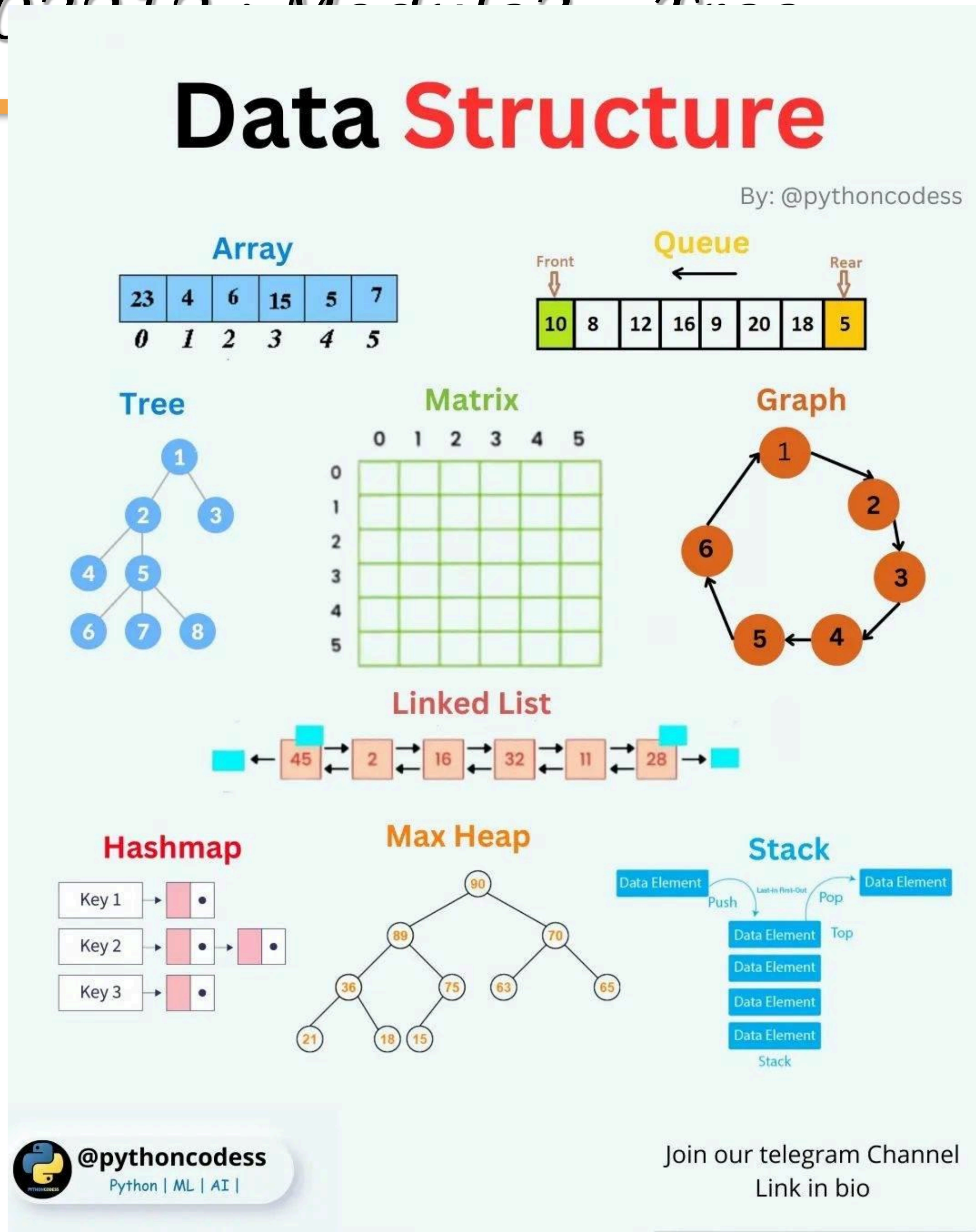
Module3—Tree

ค่าคงที่ log n ส่วนใหญ่เขียน $O(N^2)$

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84×10^{19}

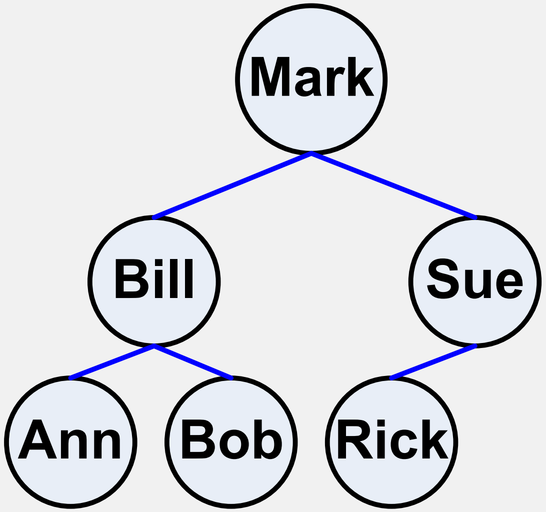

ช้าสุด
n^n
n!
2^n
n^2
n log n
n
log n
1
เร็วสุด
4096
2048
1024
512
256
128
64
32
16
8
4
2
1
2
3
4
5
6
7
8


Data Structure
By: @pythoncodess
Array
23 4 6 15 5 7
0 1 2 3 4 5
Queue
Front
Rear
10 8 12 16 9 20 18 5
Tree
Matrix
0 1 2 3 4 5
Graph
Linked List
45 2 16 32 11 28
Hashmap
Key 1
Key 2
Key 3
Max Heap
90 89 70 36 75 63 65 21 18 15
Stack
Data Element
Push
Pop
Top
Stack
@pythoncodess
Python | ML | AI |
Join our telegram Channel
Link in bio

3. Trees

Problem

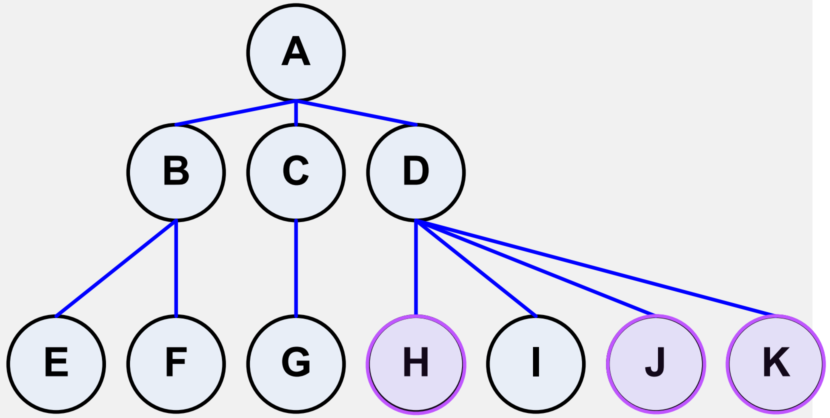
- Linear time access of linked list.
- Running time of operation O(n).



Correct : Trees

- Average time $O(\log_n)$.
- Worst case O(n).

3.1 Tree Definition



โครงสร้างข้อมูลต้นไม้ (Tree Data Structure) หรือเรียกสั้นๆว่าทรี (Tree) เป็นโครงสร้างข้อมูลรูปแบบหนึ่งในลักษณะ

- โครงสร้างข้อมูลชนิดไม่เชิงเส้น (Non-Linear)
- node
  สมาชิกแต่ละตัวในทรีสามารถเชื่อมโยงไปยังสมาชิกตัวถัดไป (Successor) ได้มากกว่าหนึ่งตัว
- และเชื่อมโยงถึงกันในลักษณะเป็นระดับคล้ายกับการแตก กิ่งก้านสาขาออกไปของต้นไม้
- ความสัมพันธ์ของสมาชิกข้อมูลในทรี จึงมีลักษณะลำดับชั้น (Hierarchical Relationship) คือ มีการเชื่อมโยงของแต่ละโหนดเป็นแบบทางเดียวจากบนลงล่าง
- โครงสร้างข้อมูลทรีประกอบด้วย**โหนด** (Node) สำหรับจัดเก็บข้อมูล และกิ่งหรือ**เส้น**ที่เชื่อมโยง

https://www2.cs.science.cmu.ac.th/courses/204251/lib/exe/fetch.php?media=tree.pdf

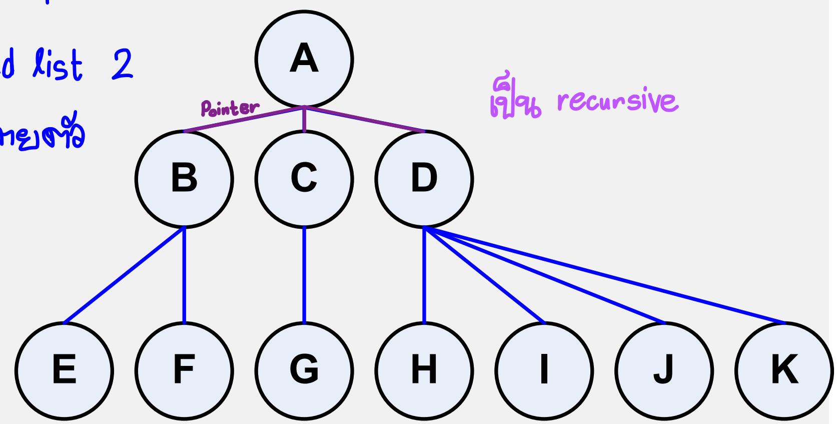
3.1 Tree Definition

linked List 1

Doubly linked list 2

tree หลายตัว

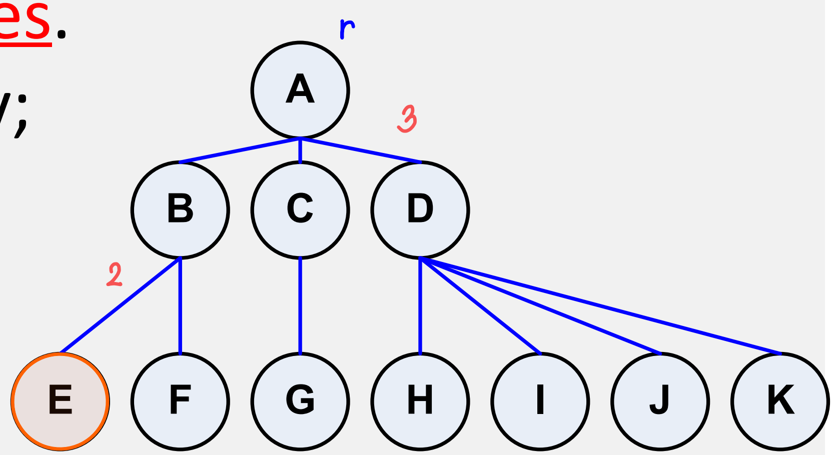
เป็น recursive



A tree data structure can be defined recursively as a collection of nodes (starting at a root node), where each **node** is a data structure consisting of a value,

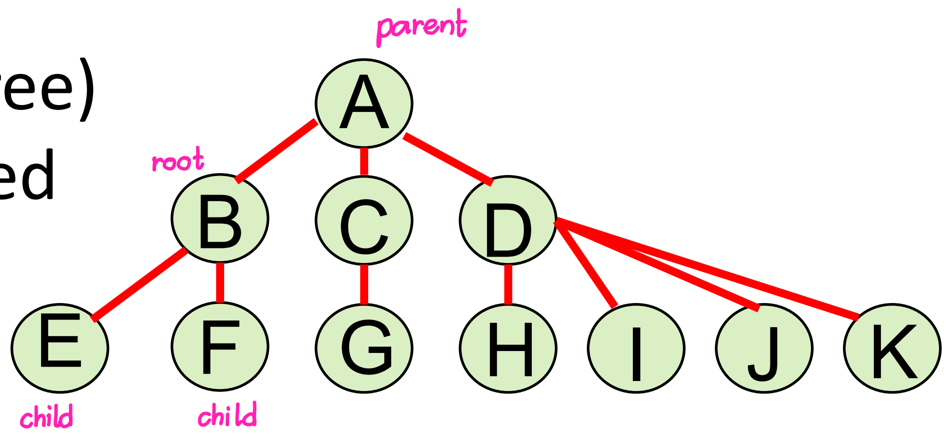
together with **a list of references** to nodes (the "children"), with the constraints that no reference is duplicated, and none points to the root.

- A tree is a collection of nodes.
- The collection can be empty;
- Otherwise,
  - a tree consists of a distinguished node r, called the root,
  - and zero or more nonempty (subtrees), $T_1, T_2, \ldots, T_k$



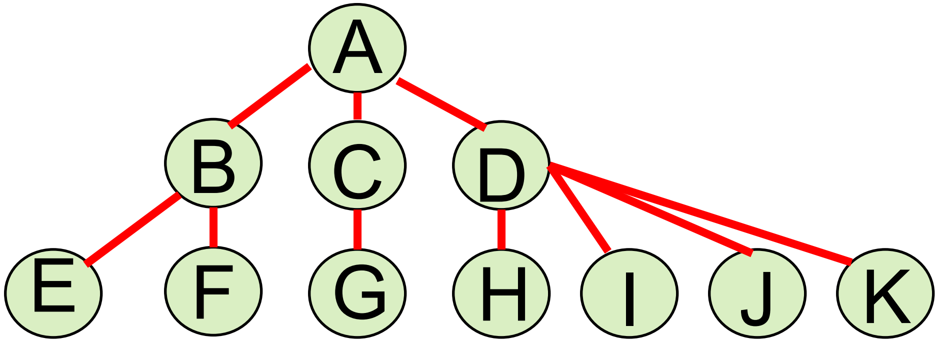
- each of whose root(Sub tree) are connected by a directed edge from r.

เส้น



- A root of each subtree is said to be a child of r,
- And r is the parent of each subtree root.

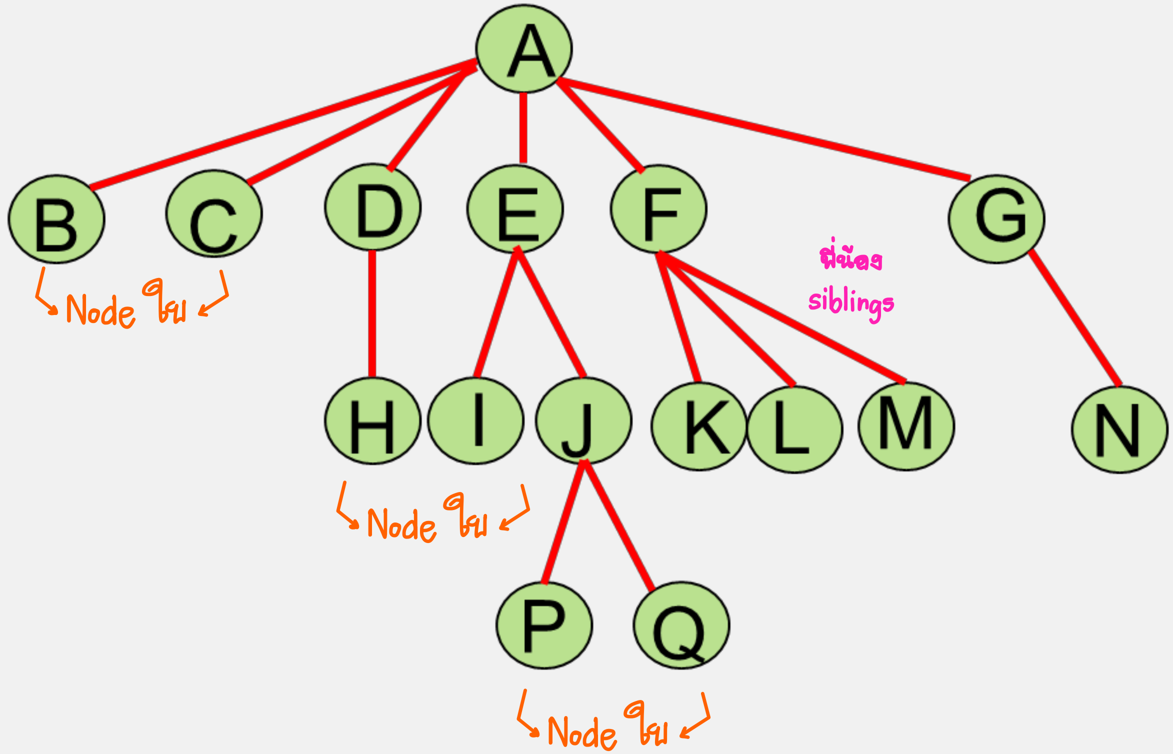
Recursive definition



- A tree is a collection of N nodes,
- one of which is the root, and N-1 edges.
- That there are N-1 edges follows from the fact that each edge connects some node to its parents.
- And every node except the root has one parent.

นิยามที่ใช้กับ Tree

- Leaves (Terminal)
- Parents
- Siblings
- Non Leaves (Non terminal)



นิยามที่ใช้กับ Tree

- **Degree** :The number of children of a node x in a rooted tree T.

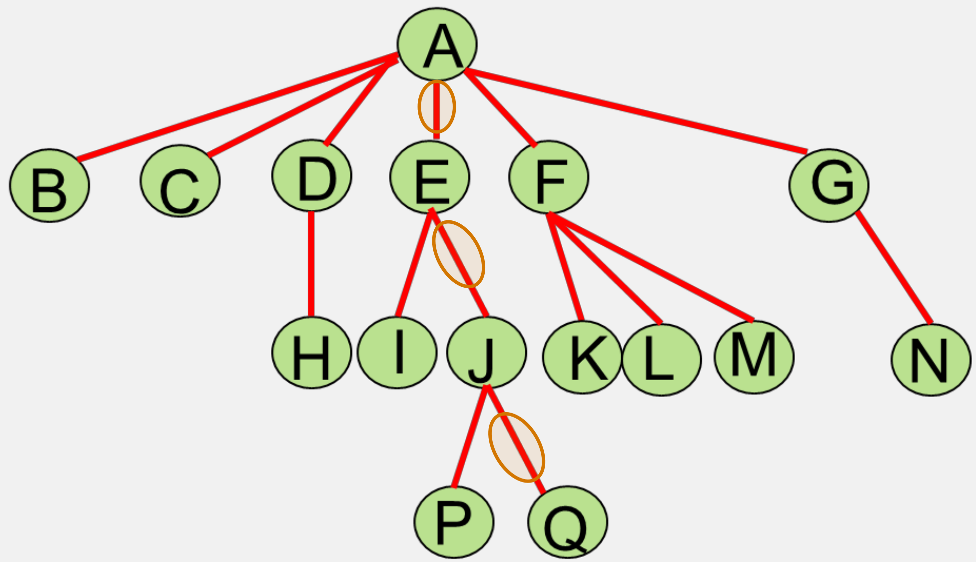
จำนวนลูก

A =6

B =0

เส้นทาง

- **Path** from node n_1 to n_k : Sequence of nodes $n_1, n_2, .., n_k$ such that n_i is the parent of n_{i+1} for $1 <= i <= k$.



PATH จาก A ถึง Q

A, E, J, K

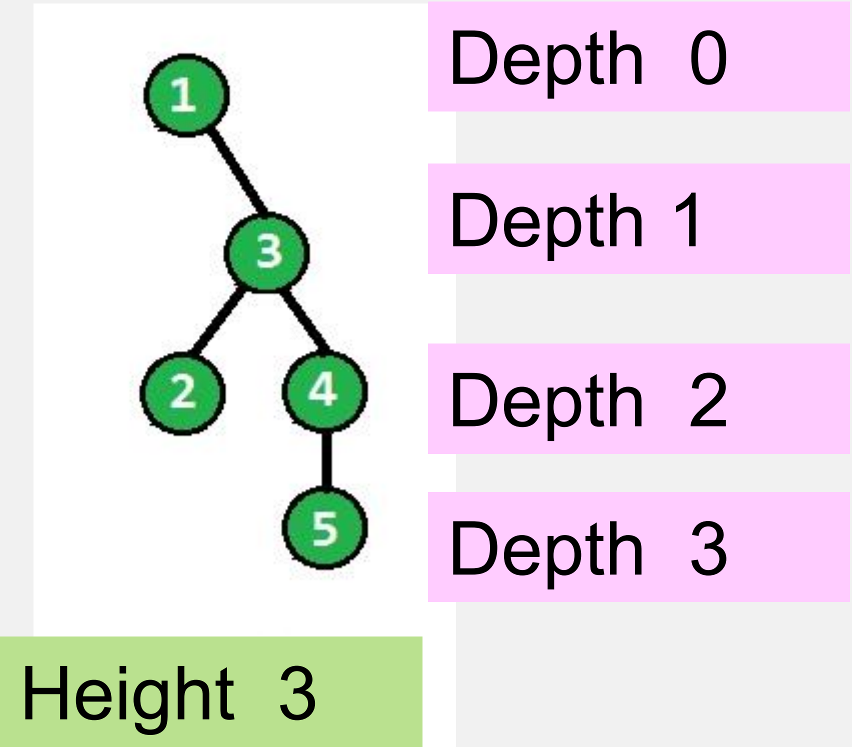
- lenght A - Q 3

ความลึก

- **Depth** : For Any node n_i, the **depth** of n_i is the length of the unique path from the root to n_i.

ความสูง

- **Height** : Is the longest path from n_i to a leaf. All leaves are at height 0. The height of a tree is equal to the Height of the root.



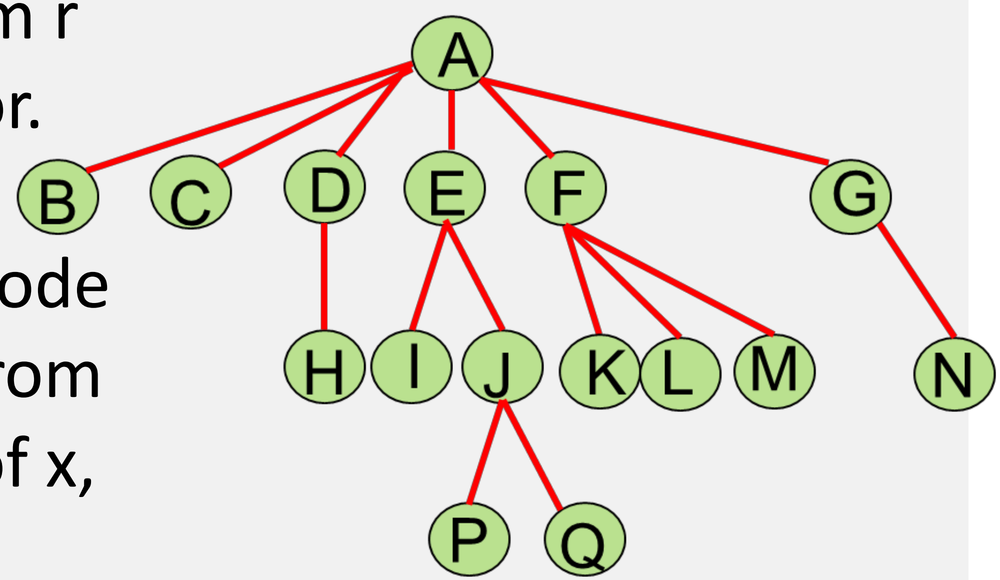
<u>Notice that</u> in a tree there is exactly one path from the root to each node.

ผู้เฒ่าแก่ บรรพบุรุษ

- **Ancestor** of x : Any node y on the unique path from r to x is called an ancestor.

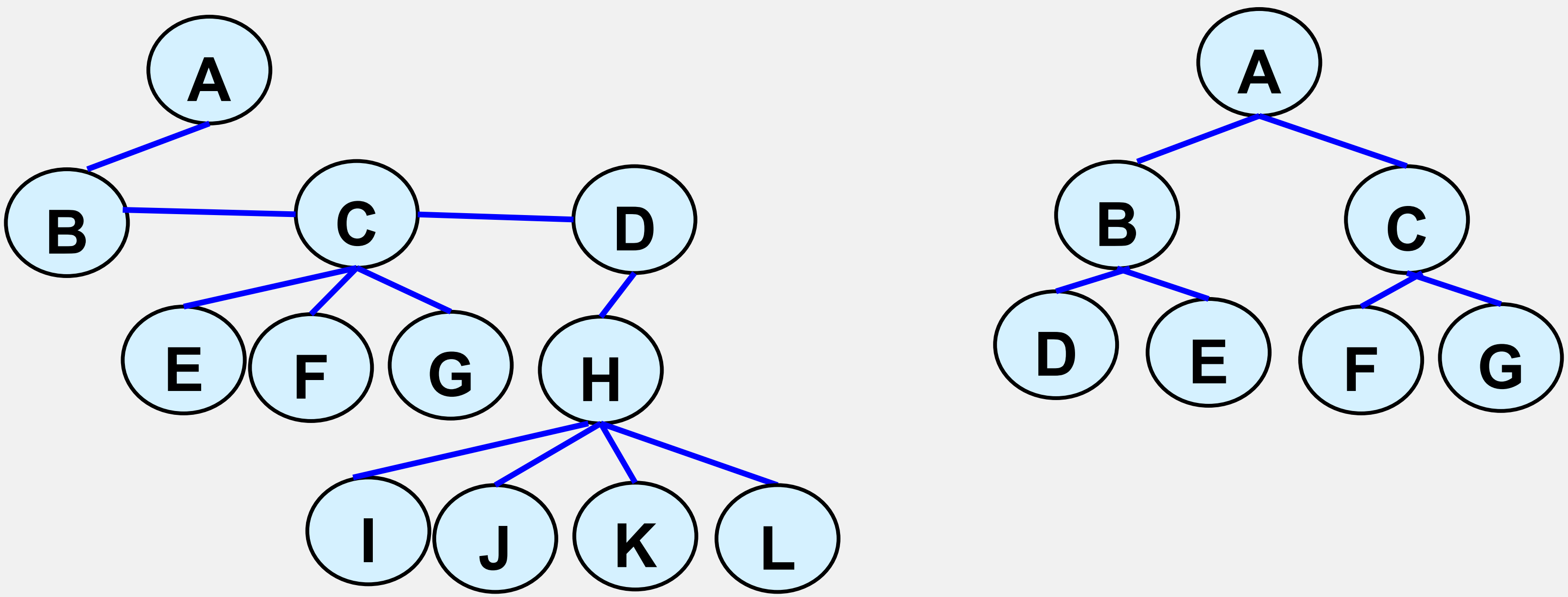
ลูกหลาน

- **Descendant** of y :Any node y on the unique path from r to x, y is descendant of x, Every node is both an ancestor of and a descendant of itself.



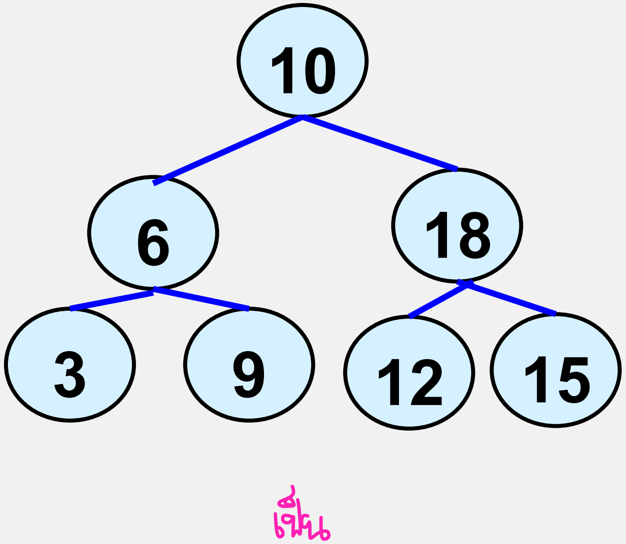
5.2 Binary tree

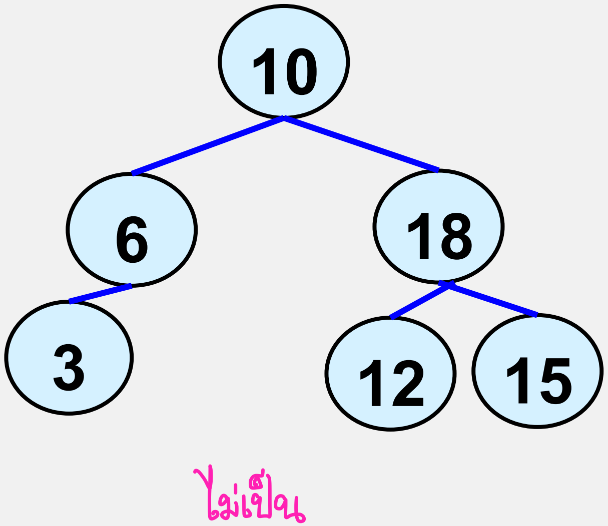
1) A Binary tree is a tree in which no node can have more than two children. ไม่มีมากกว่า 2 ลูก

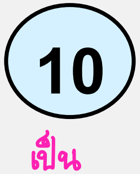


2) Full Binary tree (Complete Binary tree) :Binary tree which each node is either a leaf or has a degree exactly2

leaf หรือ degree 2 (ไม่ 0 ก็ 2)





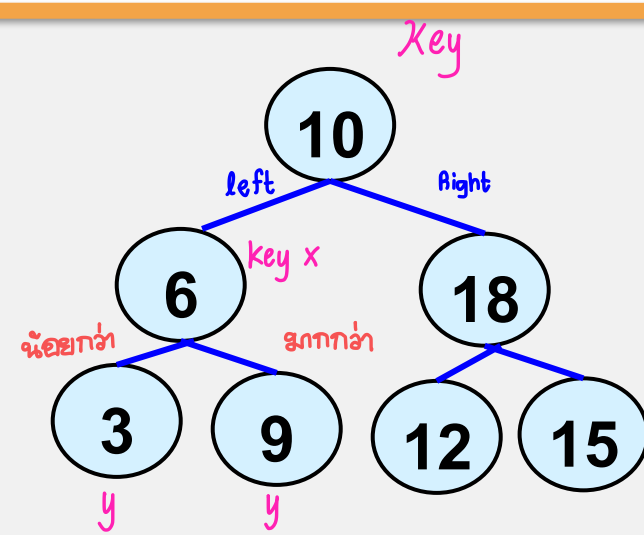


5.3 Binary search tree

พิเศษ

- Special type of binary tree,
- The keys in a binary search tree are always stored in such a way as to satisfy the binary search tree property:
  - Let x be a node in a binary search tree.
  - If y is a node in the left subtree of x, then key ซ้าย y <= key x. If y is a node in the right subtree of x, then key x <= key y.

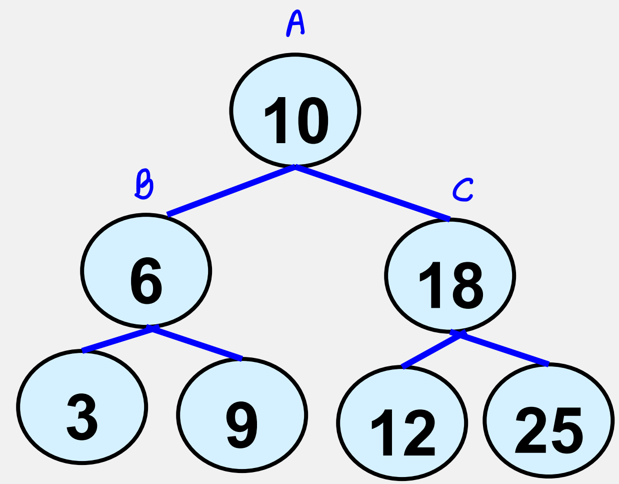
ขวา



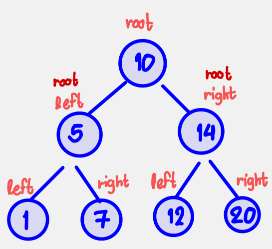
5.3.1 Tree Traversal

ออกสอบ

Binary search tree property allow us to print out all the keys in a tree in sorted order by a simple recursive algorithm called inorder **tree walk**.



1) **Preorder** Root Left Right ABC
2) **Inorder** Left Root Right BAC
3) **Postorder** Left Right Root BCA



root L L R root R L R

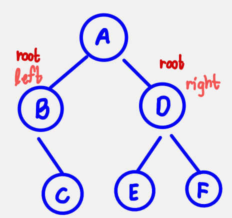
Pr : 10 5 1 7 14 12 20

I : 1 5 7 10 12 14 20

Post : 1 7 5 12 20 14 10

เดินให้สุดๆ จาก node

น้อยไปมาก



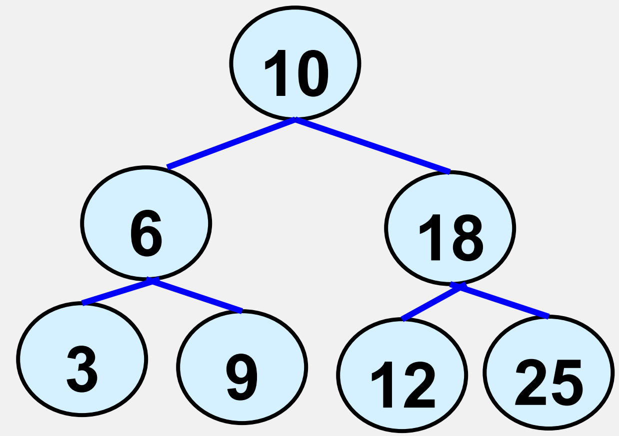
Pre : A B C D E F

In : B C A E D F

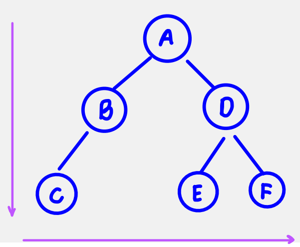
Post : C B E F D A

5.3.1 Tree Traversal

Binary search tree property allow us to print out all the keys in a tree in sorted order by a simple recursive algorithm called inorder **tree walk**.



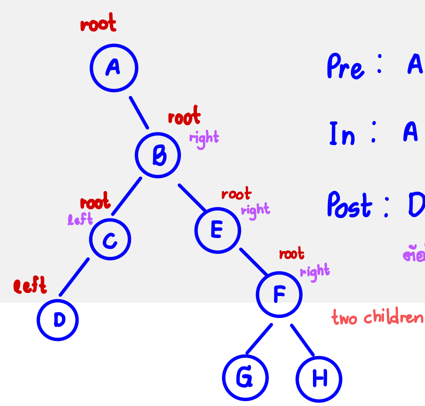
1) **Preorder** Root Left Right
2) **Inorder** Left Root Right
3) **Postorder** Left Right Root



Pre : A B C D E F

In : C B A E D F

Post : C B E F D A



Pre : A B C D E F G H

In : A D C B E G F H

Post : D C G H F E B A

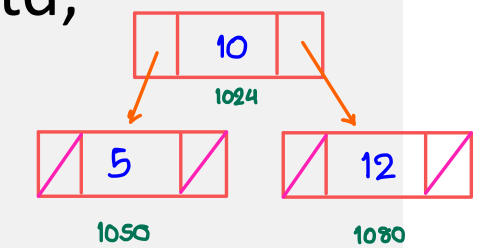
ต้องให้สุดทุกตัว

5.3.2 Operation

1. **Insert**
2. **Delete**
3. **Print :**
   - Preorder,
   - Inorder,
   - Postorder
4. **Find**

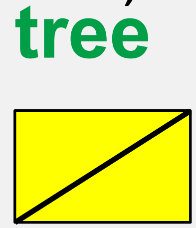
Example 1

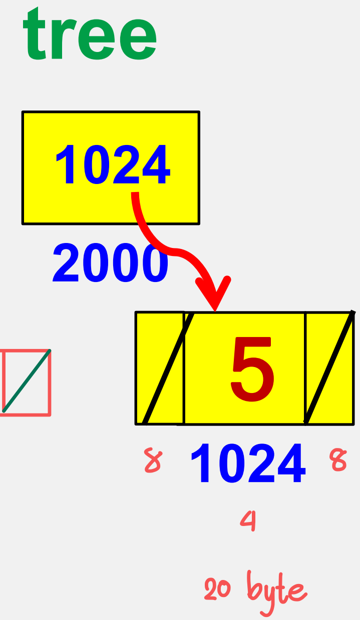
```
#include <iostream>
#include <stdio.h>
using namespace std;
struct node
{ int value;        เก็บ Value
  struct node *left;
  struct node *right;
};
```



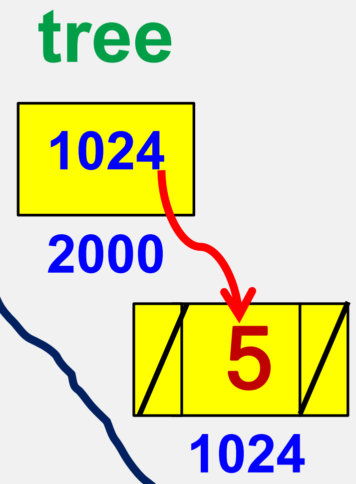
NULL 5

```
struct node *insert(struct node *tree, int x)
{   if(tree==NULL)
    {   tree = new struct node;                          ...... 1
        tree->value = x;                                 ...... 2
        tree->left =  tree->right = NULL;                ...... 3
    }                                                    ... 4,5
    else
    {     if( x < tree->value )                          ...... 6
              tree->left = insert(tree->left, x);        ...... 7
          else  if(x > tree->value)
              tree->right = insert(tree->right, x);      ...... 8
    }                                                    ....... 9
    return tree; }   return 1024                         ...... 10
```





```
struct node *insert(struct node *tree, int x)
{   if(tree==NULL)
    {   tree = new struct node;
        tree->value = x;
        tree->left =  tree->right = NULL;
    }
    else
    {     if( x < tree->value )
                tree->left = insert(tree->left, x);
           else  if(x > tree->value)
                tree->right = insert(tree->right, x);
    }
   return tree;  }
```



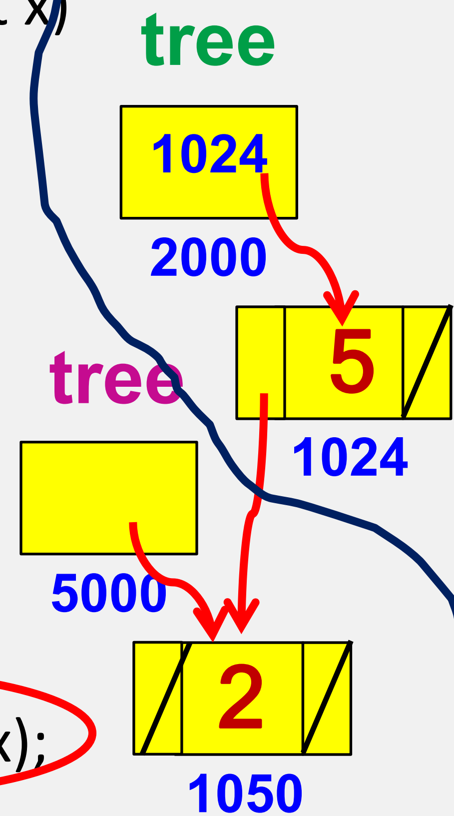
...... 1
...... 2
...... 3
... 4,5
...... 6
...... 7
...... 8
.... 9
...... 10

1024	2
NULL	2

```
struct node *insert(struct node *tree, int x)
{   if(tree==NULL)                                  ...... 1
    {   tree = new struct node;                     ...... 2
        tree->value = x;                            ...... 3
        tree->left =  tree->right = NULL;           ... 4,5
    }
    else
    {     if( x < tree->value )                     ...... 6
                tree->left = insert(tree->left, x); ...... 7
          else  if(x > tree->value)                 ...... 8
                tree->right = insert(tree->right, x); ....... 9
    }
    return tree; }                                  ...... 10
```

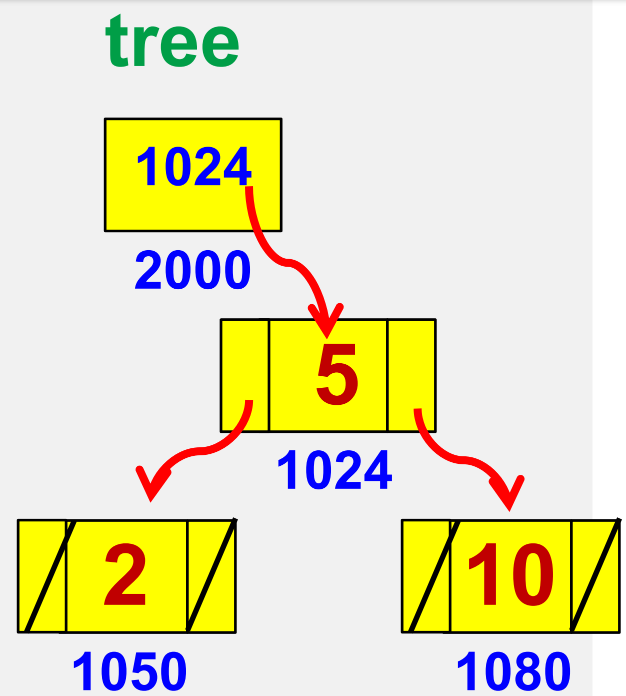
1050

return 1050 ไป 8



```
void print(struct node *tree)
{     if ( tree == NULL )
          return;
      else
      {    cout << tree->value << endl;
           print(tree->left);
           print(tree->right);
      }
      return;
}
```

(Preorder) 5 2 10



tree 1024	tree 1050	tree 1080
1	1	1
3	3	3
4	4	4
5	5	5
ค้าง Print (1050)	6	6
6	return ไป 5;	return ไป 6;
Print (1080)		

```
int main()
{     struct record *tree = NULL;
      tree = insert(tree, 5);
      tree = insert(tree, 2);
```

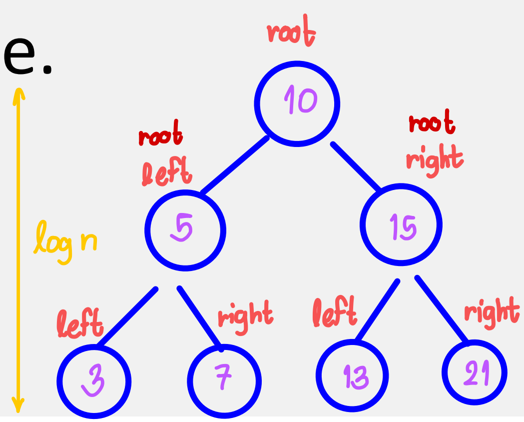
Time complexity:

Best case: O(1)

Average case: When there is a balanced binary search tree(a binary search tree is called balanced if height difference of nodes on left and right subtree is not more than one), so height becomes logN where N is number of nodes in a tree.

searching is O(logN)

Worst case: O(N)



Trees

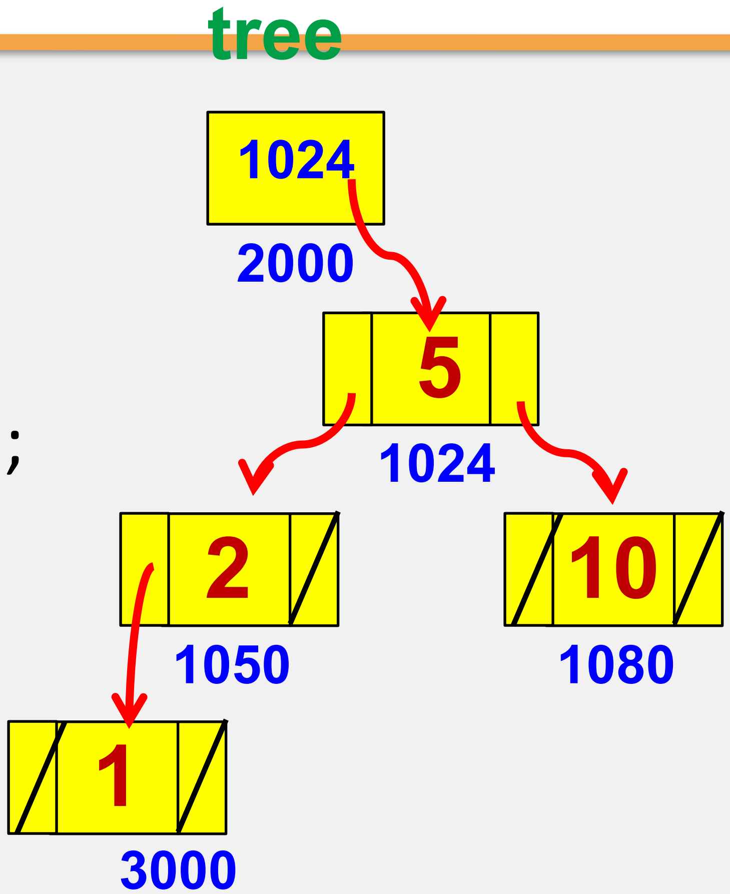
- Insert
- Print
- Search
- Find Min
- Delete

tree ใช้ 3 ครั้งเปรียบเทียบ

linked List นับตามจำนวนข้อมูล Big O O log n

```
void print(struct node *tree)
{    if ( tree == NULL )
          return;
     else
     {    cout << tree->value << endl;
           print(tree->left);
           print(tree->right);
     }
    return;
}
```

Preorder



```
void print(struct node *tree)
{     if ( tree == NULL )
            return;
      else
      {   print(tree->left);
          cout << tree->value << endl;
          print(tree->right);
      }
     return;
}
```