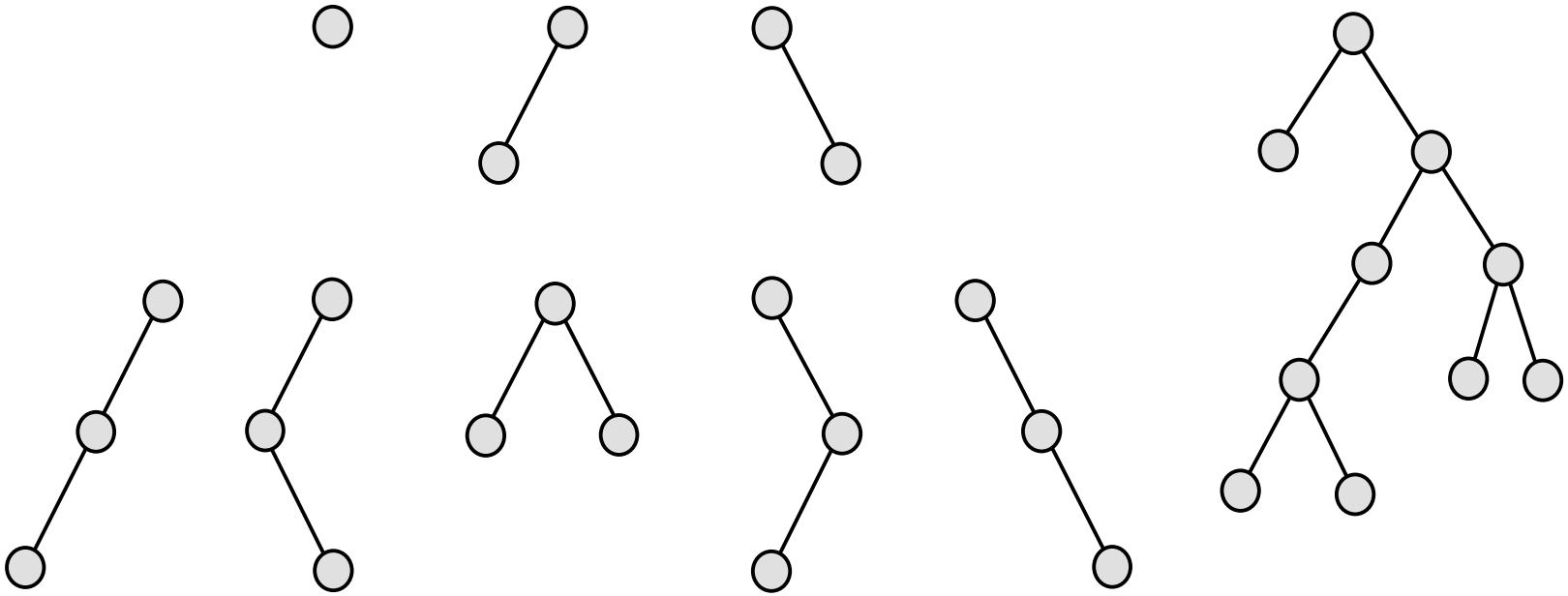
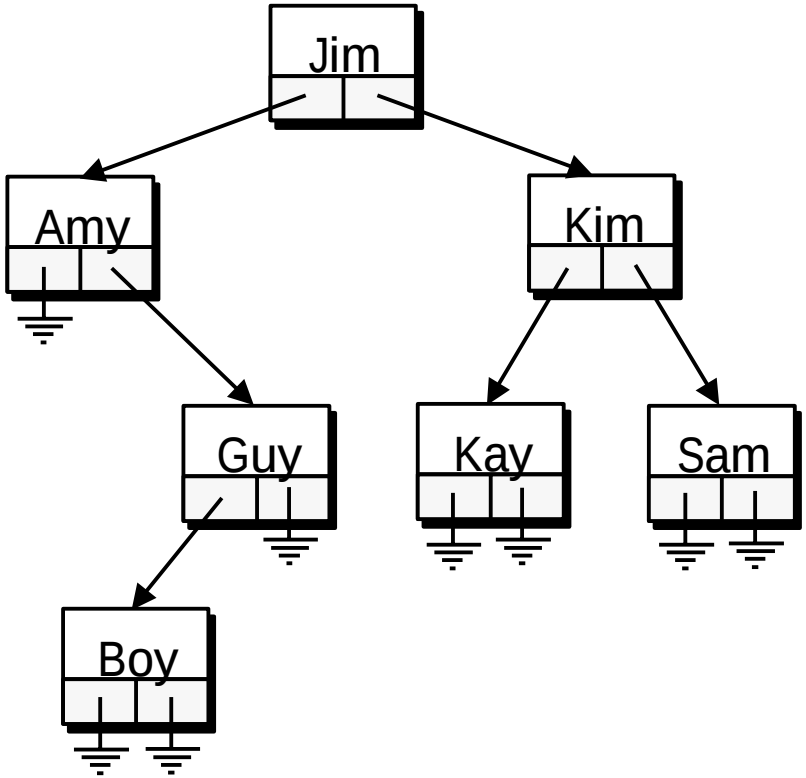
BINARY TREES

ต้นไม้แบบทวิภาค (Binary Trees)

A binary tree is either empty or it consists of a node called the root together with two binary trees called the left subtree and the right subtree of the root.



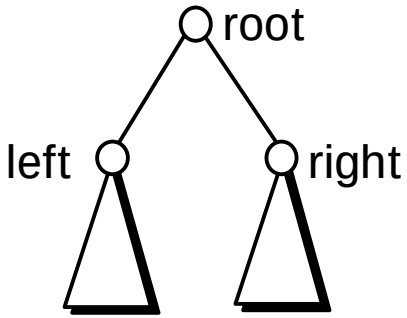
Binary Trees

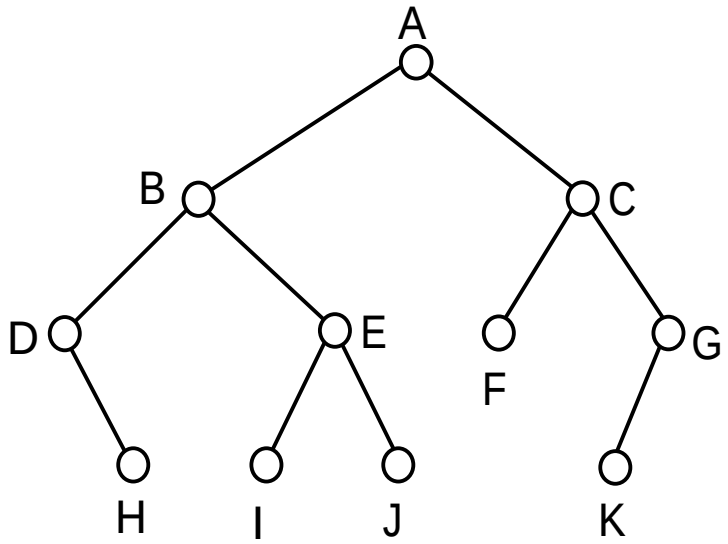



```
typedef struct NodeTag {
        ItemType  info;
        struct NodeTag *pLeft;
        struct NodeTag *pRight;
} NodeType;
```

Traversal of Binary Trees

- แบบก่อนลำดับ (preorder)
- แบบตามลำดับ (inorder)
- แบบหลังลำดับ (postorder)

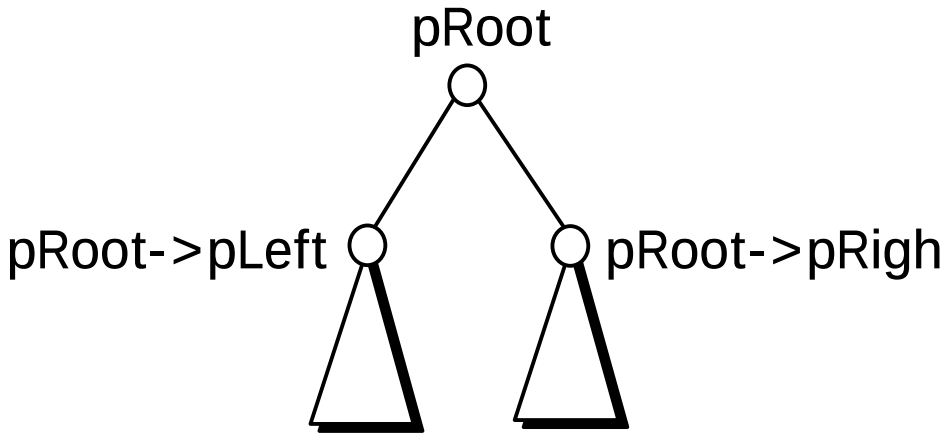




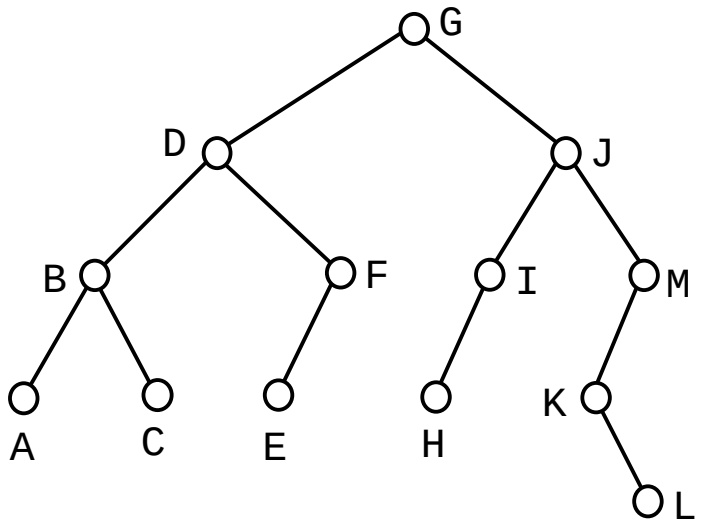

```
Pre(  root ) = root Pre( left ) Pre( right )
In(   root ) = In( left ) root In( right )
Post( root ) = Post( left ) Post( right ) root
```

Traversal of Binary Trees

```
void Preorder( NodeType *pRoot )
{
  if ( pRoot ) {
    Visit( pRoot );
    Preorder( pRoot->pLeft  );
    Preorder( pRoot->pRight );
  }
}
void Inorder( NodeType *pRoot )
{
  if ( pRoot ) {
    Inorder( pRoot->pLeft );
    Visit( pRoot );
    Inorder( pRoot->pRight );
  }
}
void Postorder( NodeType *pRoot )
{
  if ( pRoot ) {
    Postorder( pRoot->pLeft );
```



Traversal of Binary Trees

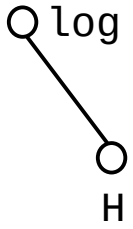


Preorder : G D B A C F E J I H N K L

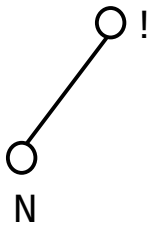
Inorder : A B C D E F G H I J K L M

Postorder : A C B E F D H I L K N J G

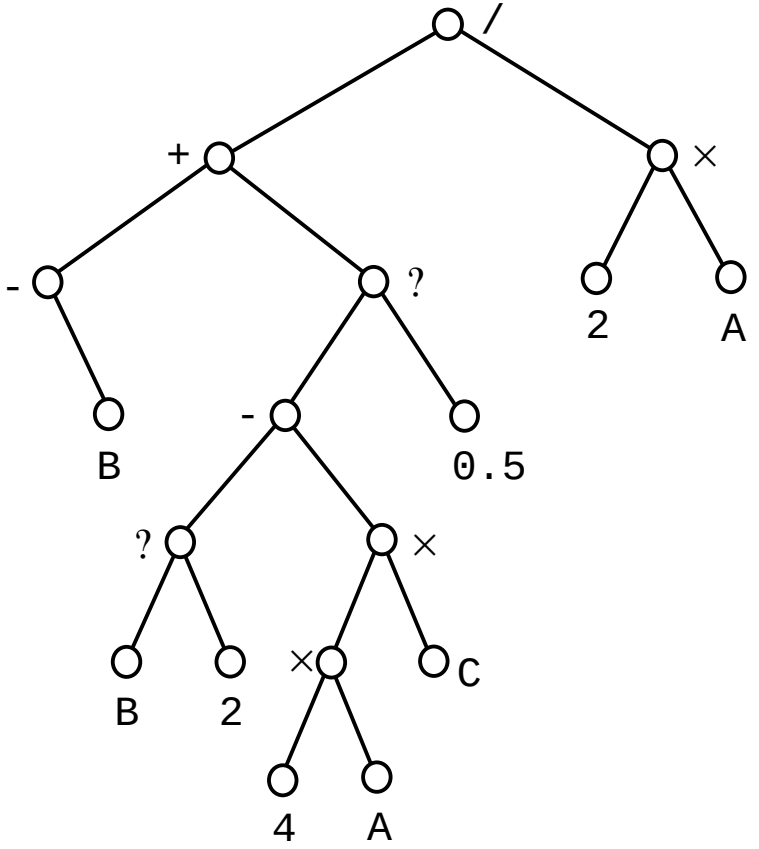
Expression Trees



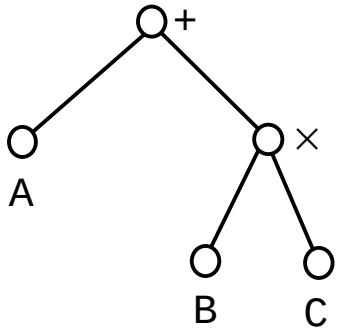
log H



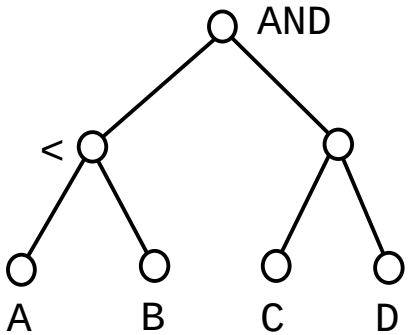
N!



$$\frac{-B+(B\ ?\ 2\ -\ 4\times A\times C)\ ?\ 0.5}{2\times A}$$

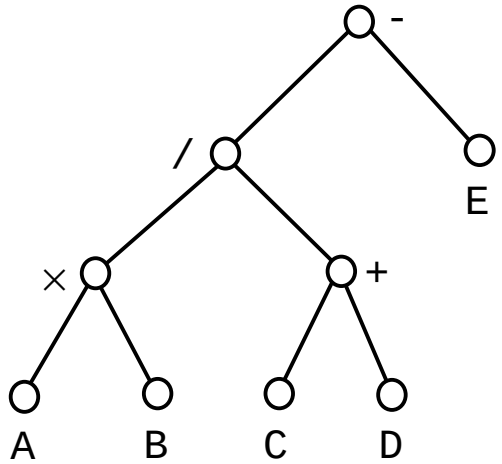


A + (B × C)



(A<B) AND (C>D)

Expression Tree & Tree Traversal

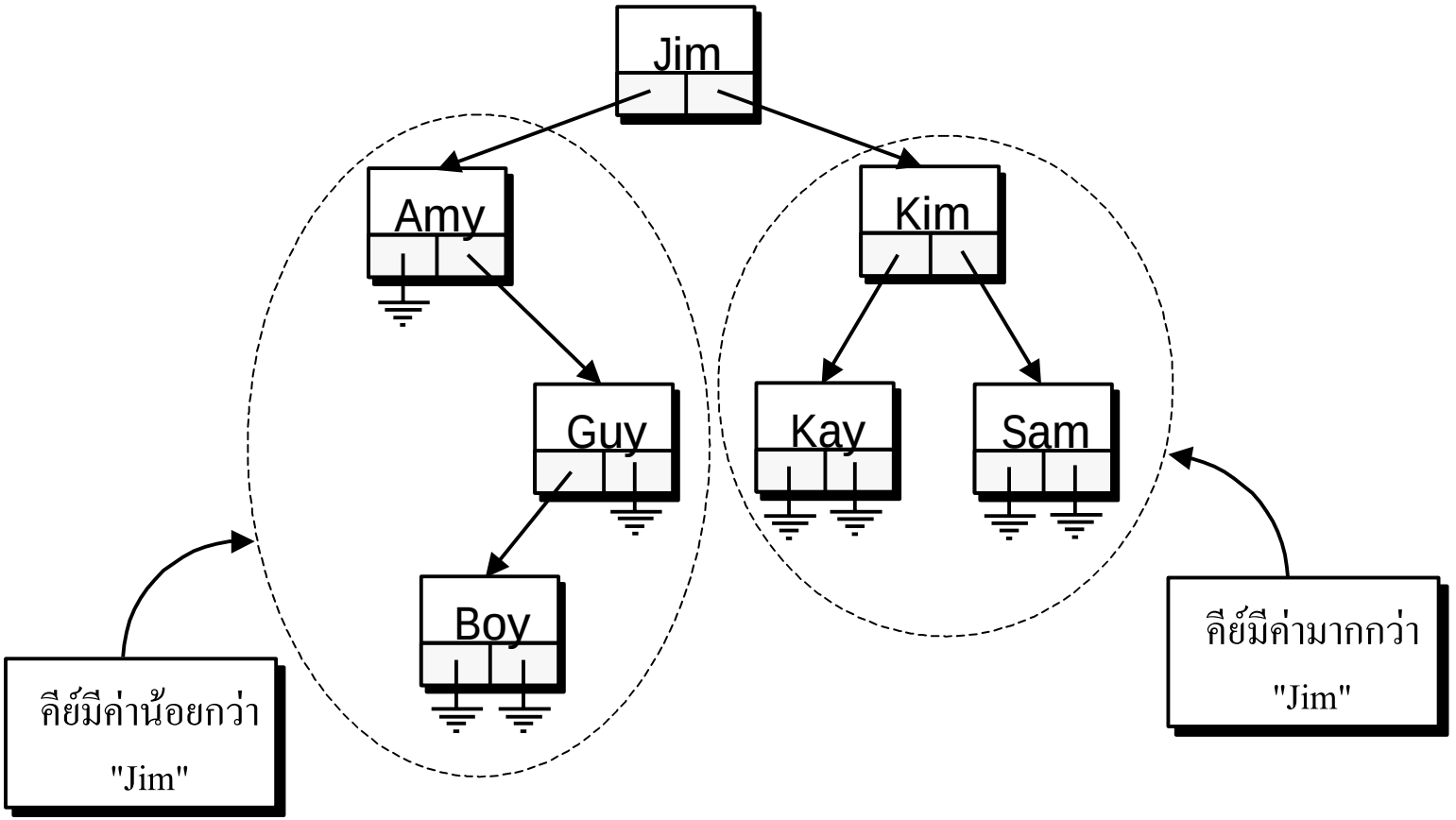


Inorder : A × B / C + D - E

Preorder : - / × A B + C D E

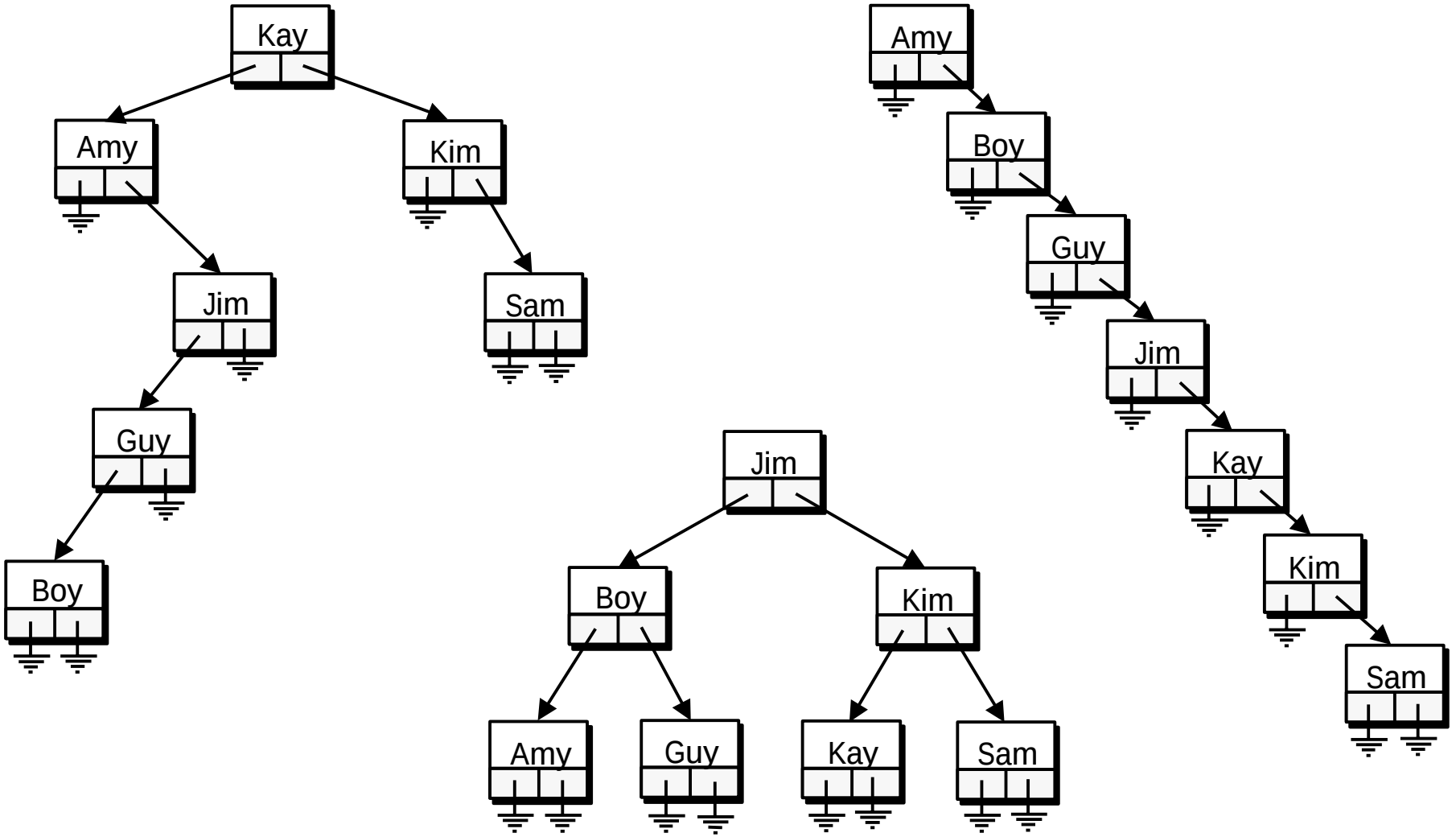
Postorder : A B × C D + / E -

ต้นไม้ค้นหาแบบทวิภาค (Binary Search Tree)




```
typedef struct NodeTag {
        ItemType  info;
        struct NodeTag *pLeft;
        struct NodeTag *pRight;
} NodeType;
```

Binary Search Trees

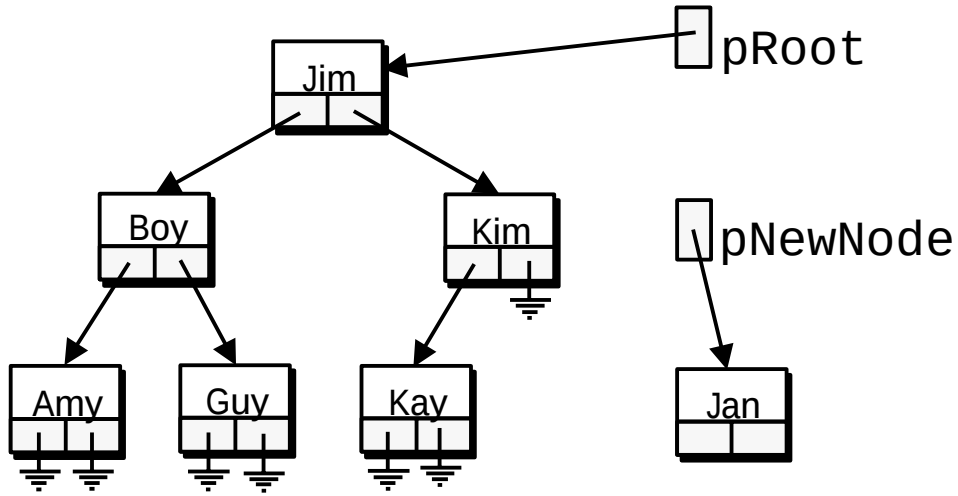


TreeSearch

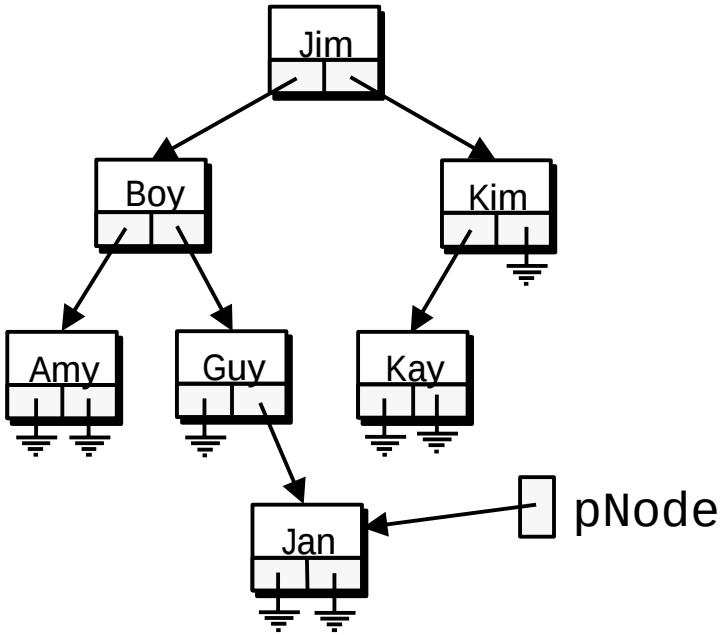
```
NodeType *TreeSearch( NodeType *p, KeyType Target )
{
  if ( p ) {
    if ( LT( Target, p->info.key ) )
      p = TreeSearch( p->pLeft,  Target );
    else if ( GT( Target, p->info.key ) )
      p = TreeSearch( p->pRight, Target );
  }
  return( p );
}

NodeType *TreeSearch( NodeType *p, KeyType Target )
{
  while ( p != NULL && NE( Target, p->info.key ) ) {
    if ( LT( Target, p->info.key ) )
      p = p->pLeft;
    else
      p = p->pRight;
  }
  return( p );
```

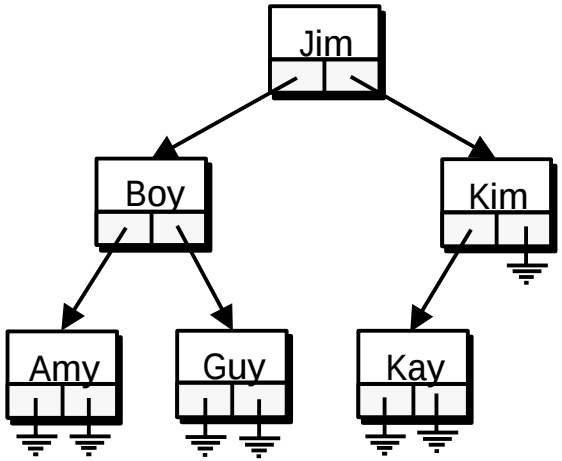
Insertion into a Binary Search Tree

```
NodeType *Insert( NodeType *pRoot, NodeType *pNewNode )
{
  if ( pRoot == NULL ) {
    pRoot = pNewNode;
    pRoot->pLeft = pRoot->pRight = NULL;
  }
  else if ( LT( pNewNode->info.key, pRoot->info.key ) )
    pRoot->pLeft = Insert( pRoot->pLeft, pNewNode );
  else if ( GT( pNewNode->info.key, pRoot->info.key ) )
    pRoot->pRight = Insert( pRoot->pRight, pNewNode );
  else
    Error( "Duplicate key" );
  return( pRoot );
}
```



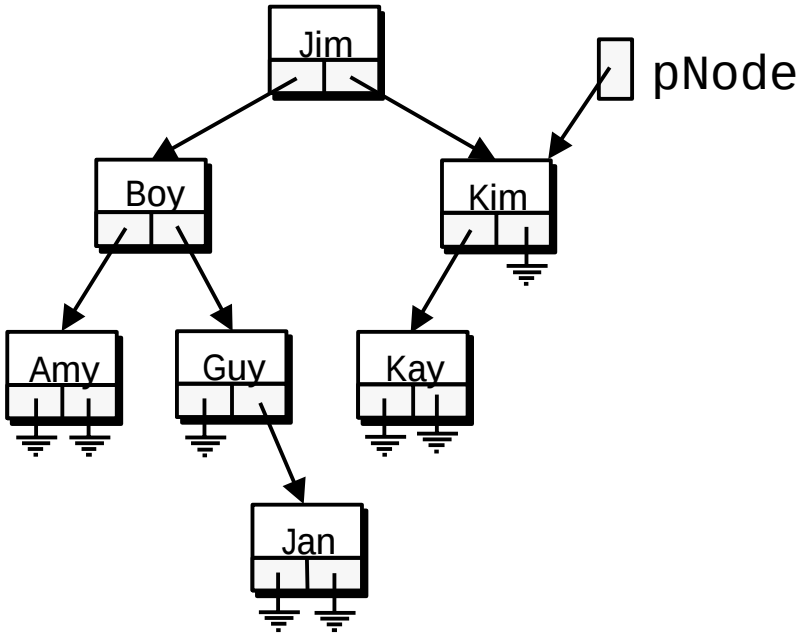
Deletion from a Binary Search Tree

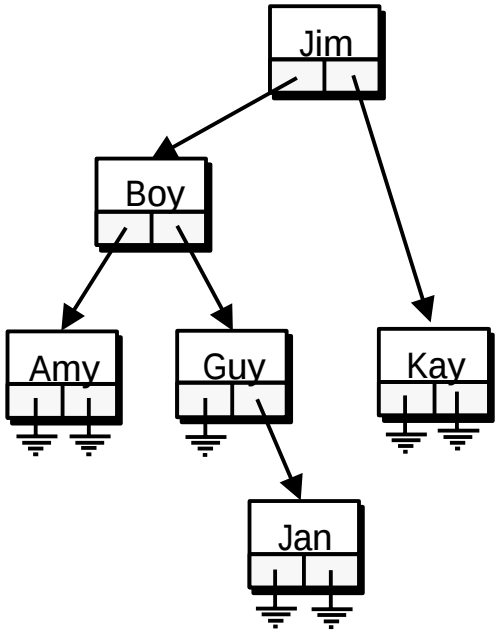




กรณีที่ *pNode เป็นใบ

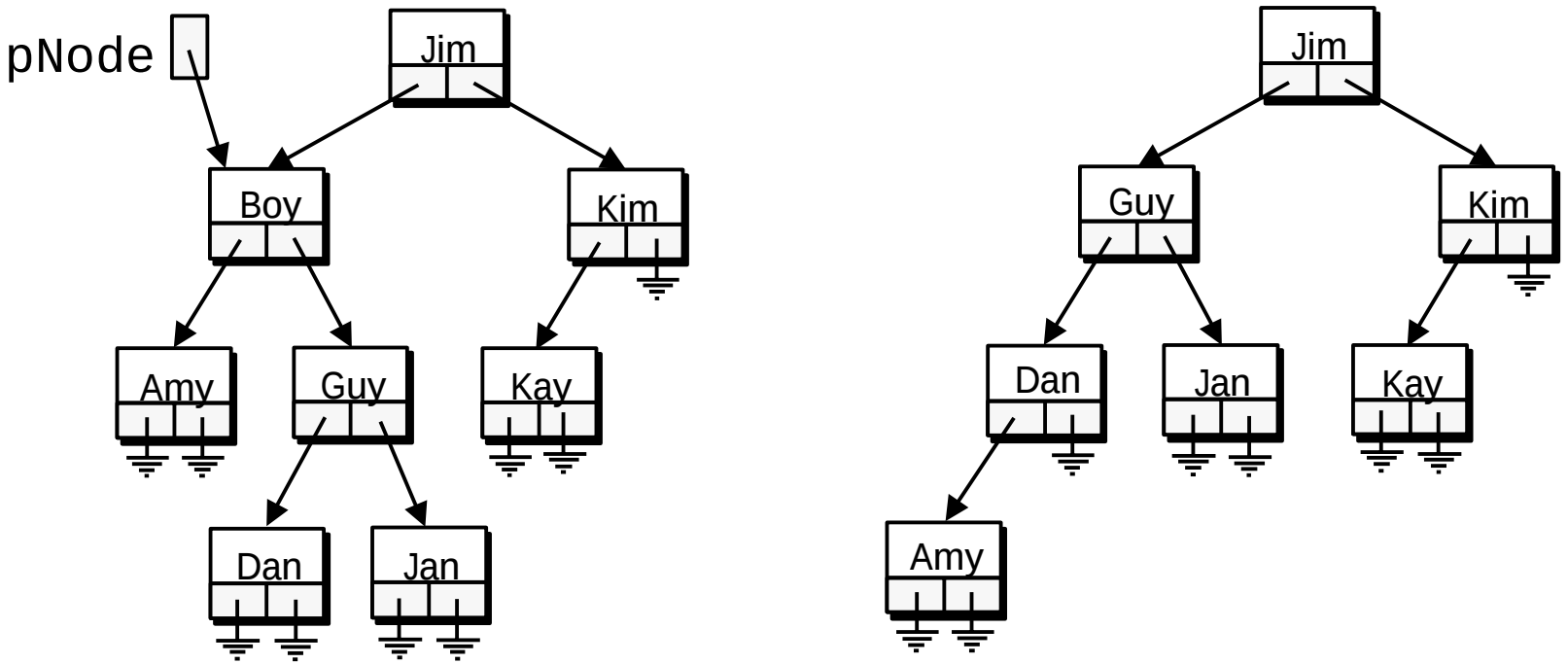
Deletion from a Binary Search Tree





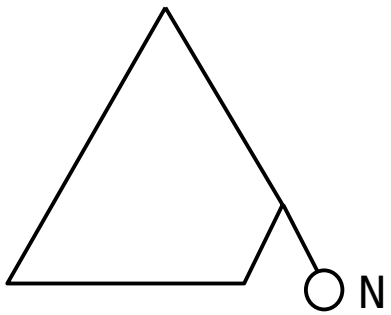
กรณีที่ต้นไม้ย่อยข้างหนึ่ง
ของ *pNode ไม่มี

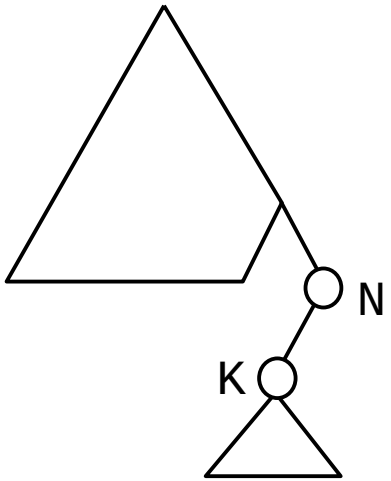
Deletion from a Binary Search Tree

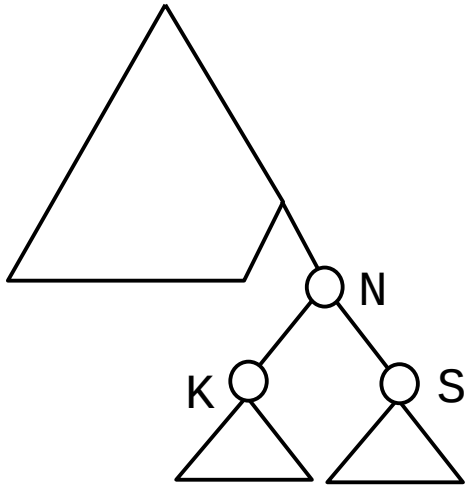


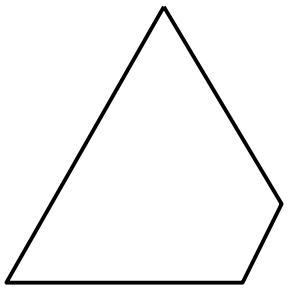
กรณีที่ *pNode มีต้นไม้
ย่อยทั้งสองข้าง

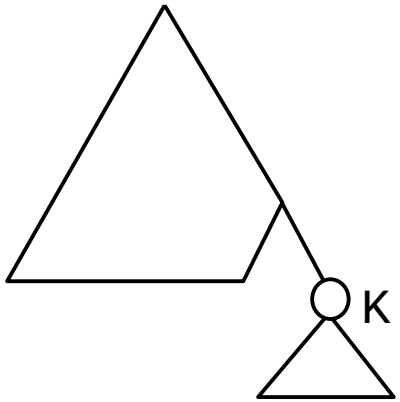
Deletion from a Binary Seach Tree

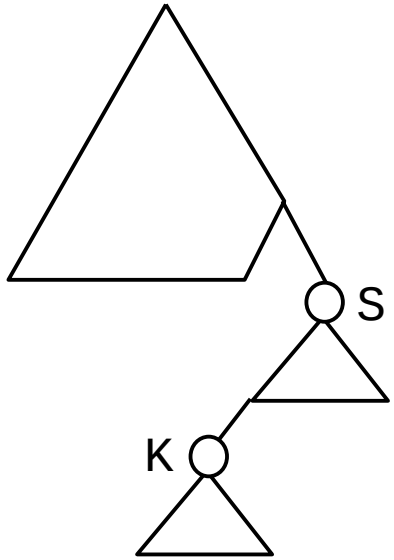




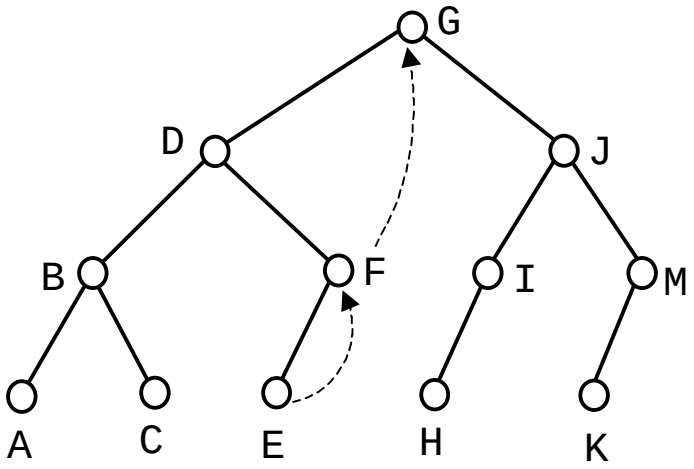


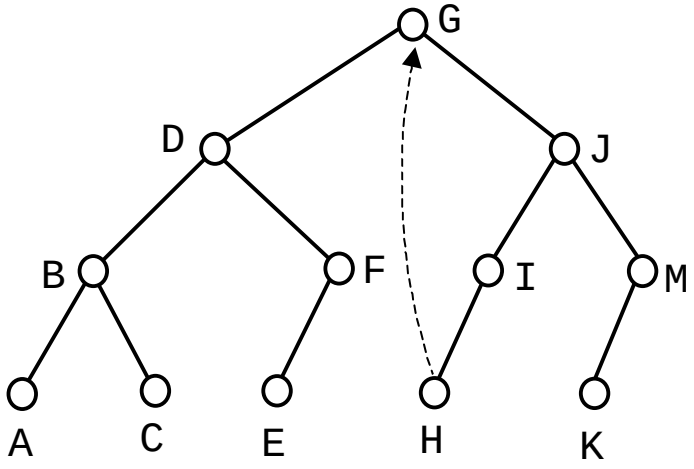






Deletion from a Binary Search Tree



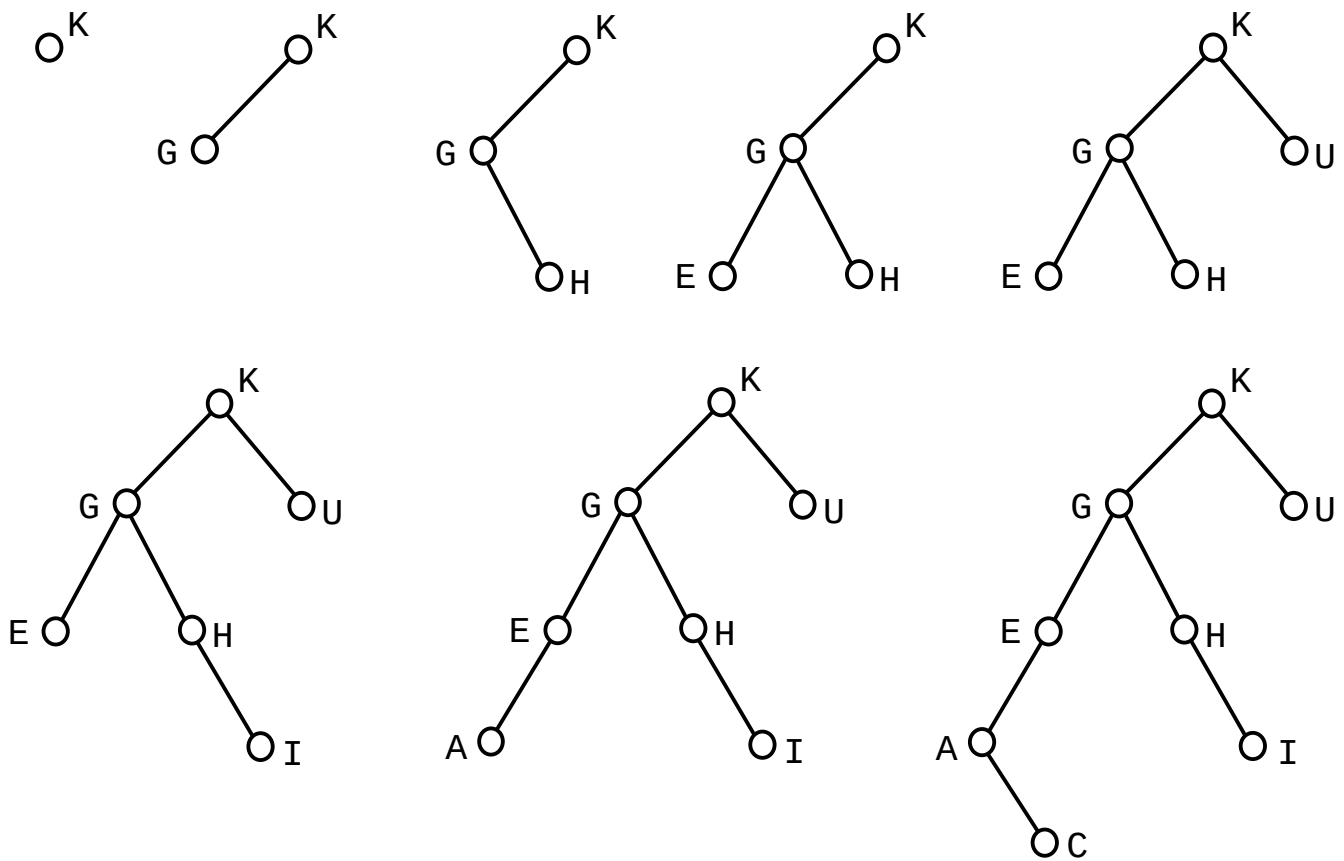


Inorder : A B C D E F G H I J K L

ถ้าจะลบ G จะสรุปได้เสมอว่า F และ H จะมี
ต้นไม้ย่อยอย่างมากเพียงหนึ่งต้น

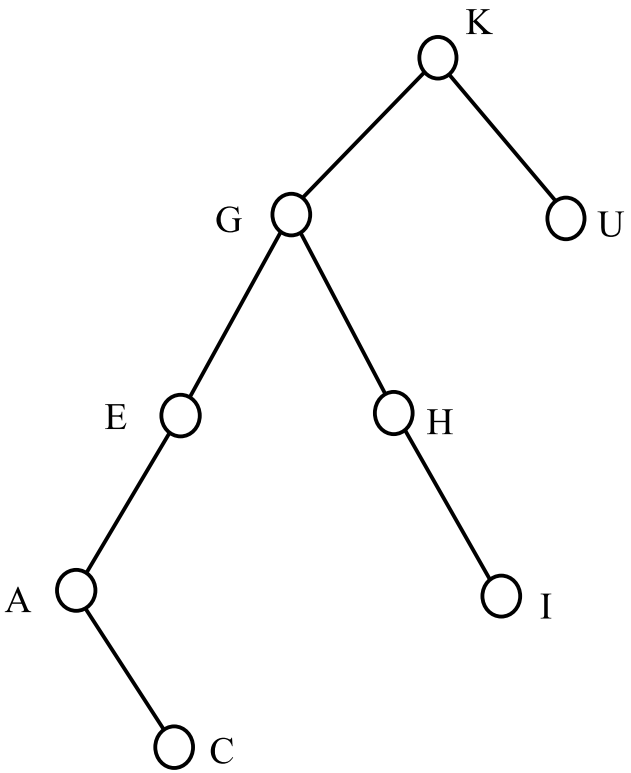
Treesort

1. รับข้อมูลที่ต้องการเรียงลำดับ
2. สร้างต้นไม้ค้นหาแบบทวิภาคจากข้อมูลเหล่านั้น
3. แวะผ่านข้อมูลในต้นไม้ แบบตามลำดับ



K G H E U I A C

Treesort vs. Quicksort



K	G	H	E	U	I	A	C

G	H	E	I	A	C	K	U

E	A	C	G	H	I	K	U

A	C	E	G	H	I	K	U

A	C	E	G	H	I	K	U

A	C	E	G	H	I	K	U

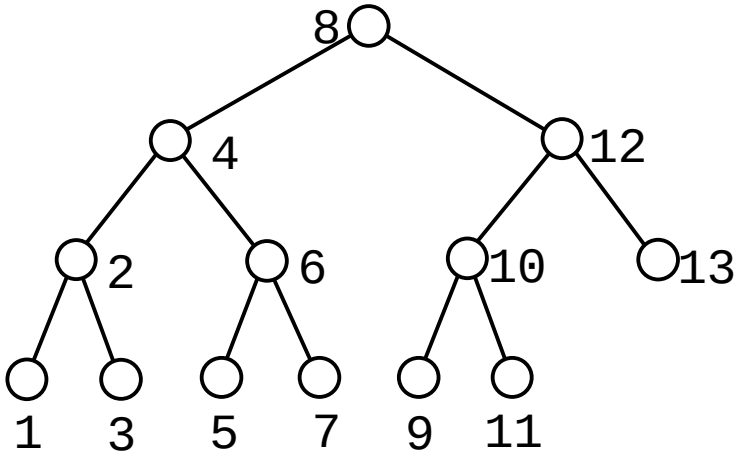
A	C	E	G	H	I	K	U

A	C	E	G	H	I	K	U

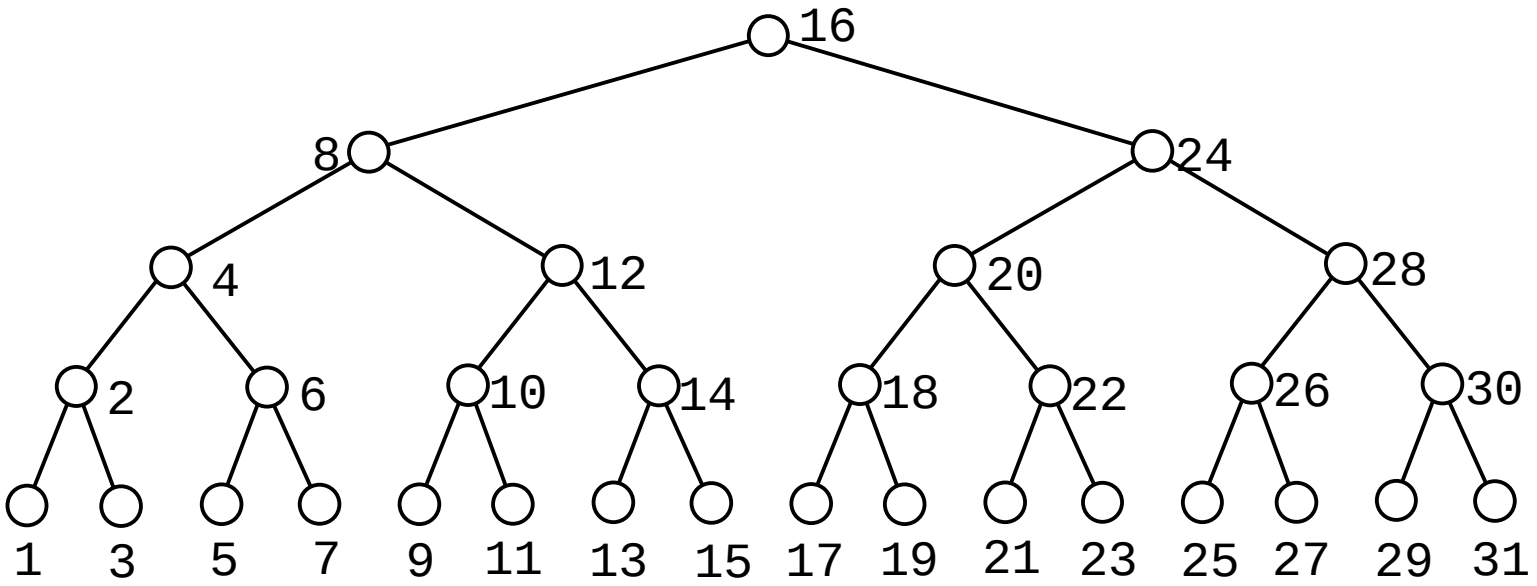
A	C	E	G	H	I	K	U

Building a Binary Search Tree

1	2	3	4	5	6	7	8	9	10	11	12	13

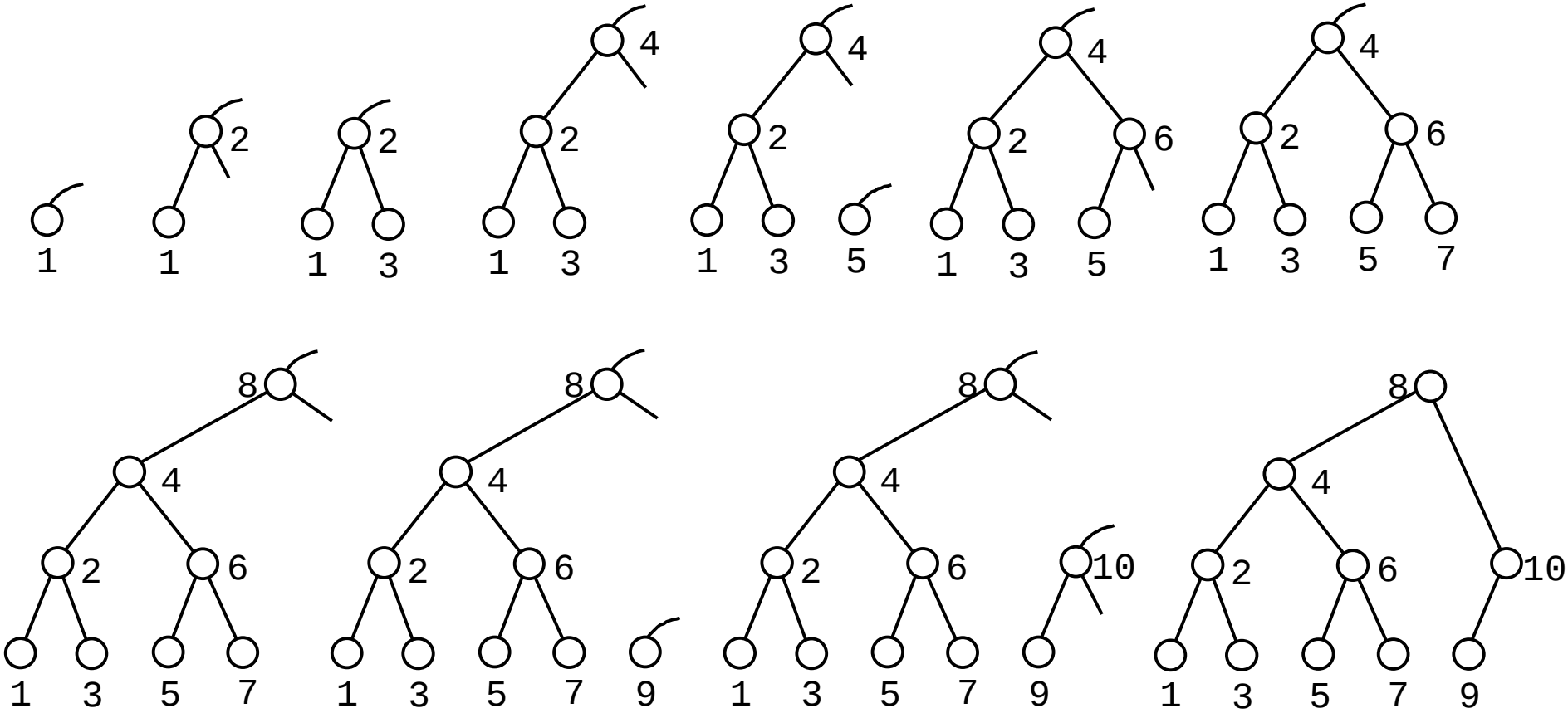


Building a Binary Search Tree



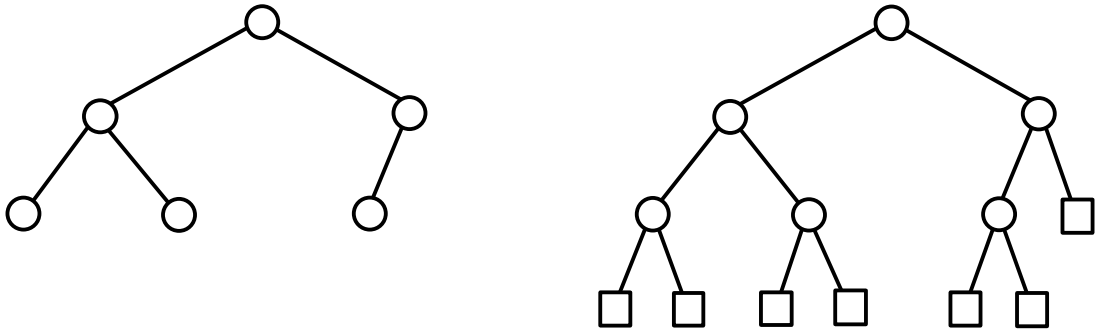
ถ้ากำหนดให้เลขประจำ node ต่างๆในต้นไม้แบบทวิภาคบริบูรณ์ (complete binary tree) ที่มีความสูง h ตรงตามลำดับของการแวะผ่านแบบตามลำดับ (inorder) จะได้ว่า เลขประจำ node ที่อยู่ในระดับที่ k จะหารด้วย 2^{h-k} ลงตัว แต่จะหารด้วย 2^{h-k} +1 ไม่ลงตัว

Building a Binary Search Tree



1	2	3	4	5	6	7	8	9	10

Randomly Built Binary Search Trees



$S(n)$ = จำนวนการเปรียบเทียบเฉลี่ย ในกรณีพบข้อมูล

$U(n)$ = จำนวนการเปรียบเทียบเฉลี่ย ในกรณีไม่พบข้อมูล

$$S(n) = \frac{2I}{n} + 1 \qquad U(n) = \frac{2E}{n+1} \qquad E = I + 2n$$

$$S(n) = \left(1 + \frac{1}{n}\right)U(n) - 3$$

$$S(n) = 1 + \frac{U(0) + U(1) + \ldots + U(n-1)}{n}$$

$$(n+1)U(n) = 4n + U(0) + U(1) + \ldots U(n-1)$$

$$nU(n-1) = 4(n-1) + U(0) + U(1) + \ldots + U(n-2)$$