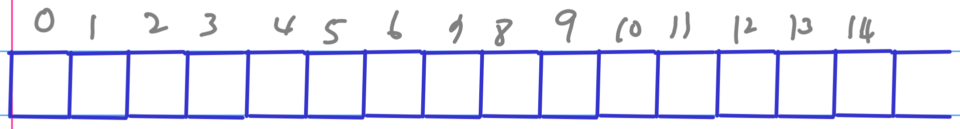
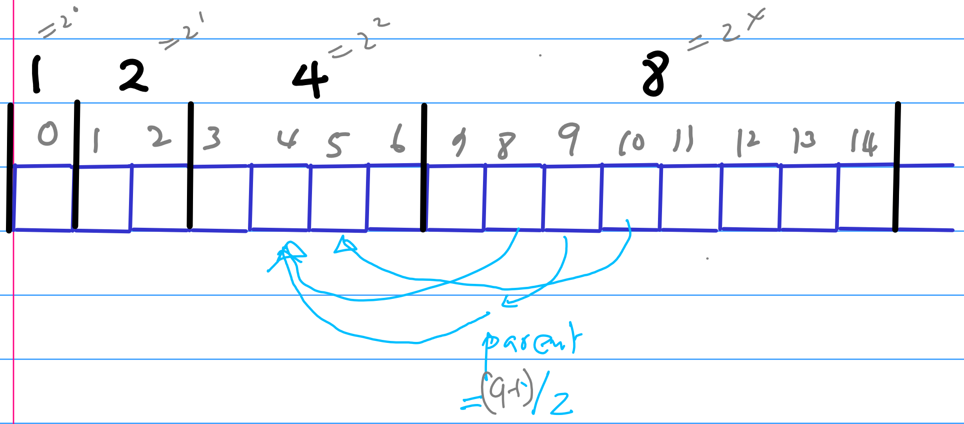
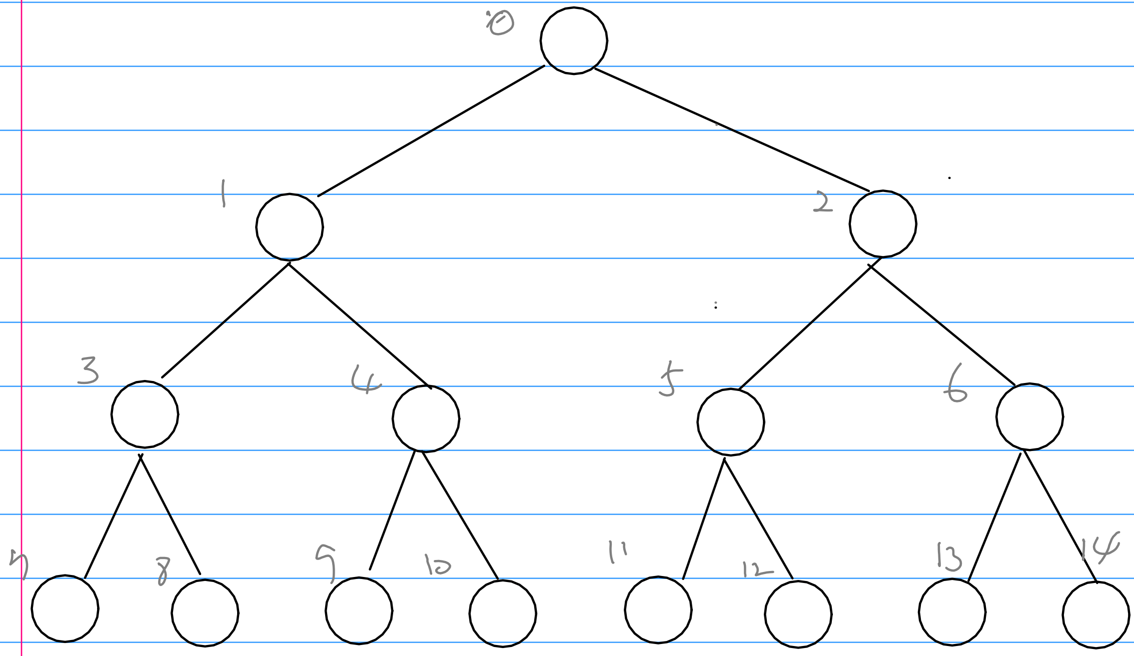
Tree (H1)





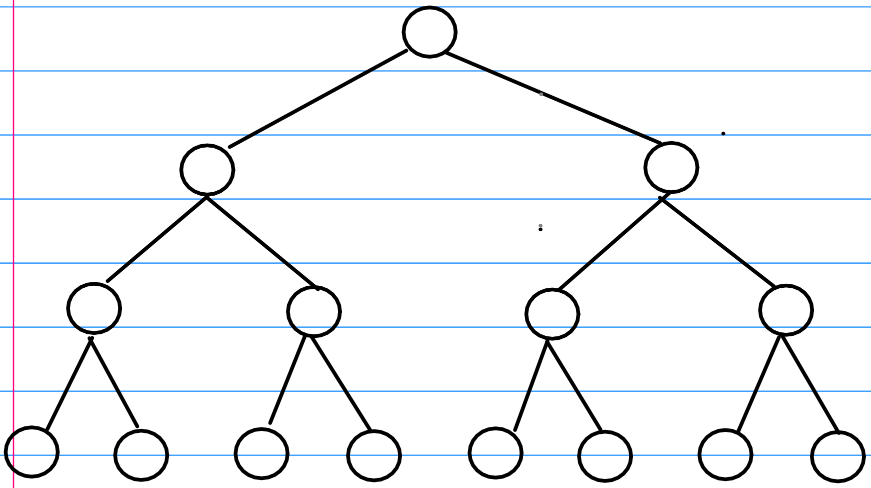


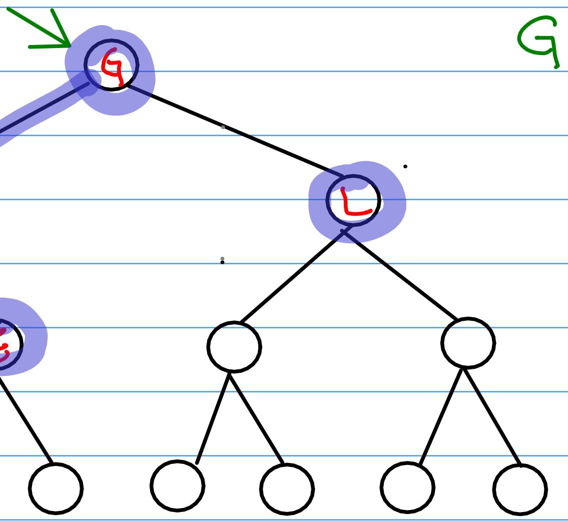
$\frac{(7-1)}{2}\frac{(8-1)}{2}$

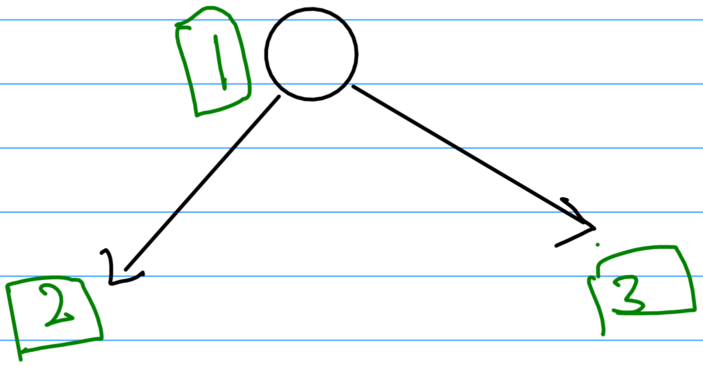
$\frac{9-1}{2}$ $\frac{10-1}{2}$

$5\times2+1$

$5\times2+2$




G
C
L
A
E
G


1
2
3

```
void PreOrder (Nptr T) {

if (T != NULL)
        {      Visit (T->Name);
               PreOrder (T->L);
               PreOrder (T->R);
        }
 }
```

G → C → A → E → L

```
void PreOrder (G) {

if (G != NULL)
        {      Visit (G);
               PreOrder (C);
               PreOrder (L);
        }
 }

1  void PreOrder (C) {

if (C != NULL)
        {      Visit (C);
               PreOrder (A);
               PreOrder (E);
        }
 }

2  void PreOrder (A) {

if (A != NULL)
        {      Visit (A);
               PreOrder (NULL);
               PreOrder (NULL);
        }
 }

3  void PreOrder (NULL) {

return

4  void PreOrder (NULL) {

return
```

```
5  void PreOrder (E) {

if (E != NULL)
        {      Visit (E);
               PreOrder (NULL);
               PreOrder (NULL);
        }
 }

6  void PreOrder (NULL) {

return

7  void PreOrder (NULL) {

return
```

```
void PreOrder (G) {

if (G != NULL)
        {      Visit (G);
               PreOrder (C);
               PreOrder (L);
        }
}
```

1

```
void PreOrder (C) {

if (C != NULL)
        {      Visit (C);
               PreOrder (A);
               PreOrder (E);
        }
}
```

8

```
void PreOrder (L) {

if (C != NULL)
        {      Visit (C);
               PreOrder (NULL);
               PreOrder (NULL);
        }
}
```

9

```
void PreOrder (NULL) {

return
```

10

```
void PreOrder (NULL) {

return
```

```
void PreOrder (Nptr T) {

if (T != NULL)
        {       Visit (T->Name);
                PreOrder (T->L);
                PreOrder (T->R);
        }
}
```

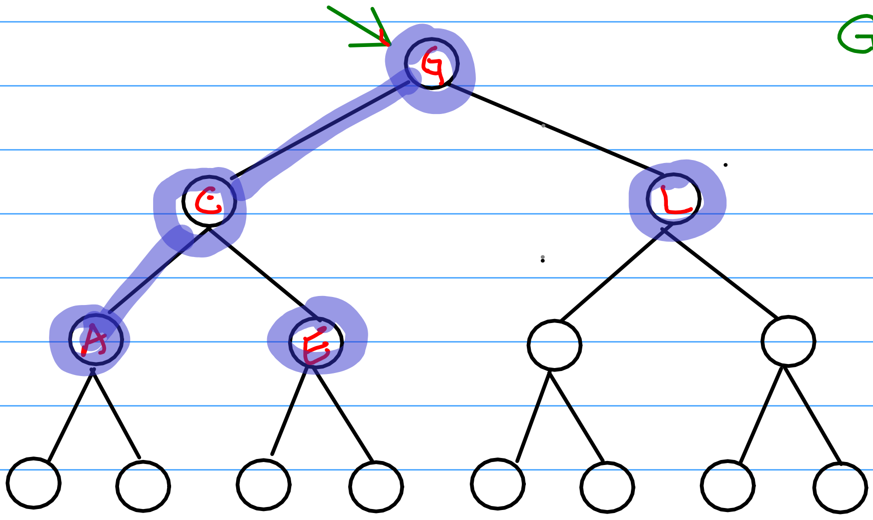
```
void InOrder (Nptr T) {

if (T != NULL)
        {       InOrder (T->L);
                Visit (T->Name);
                InOrder (T->R);
        }
}
```

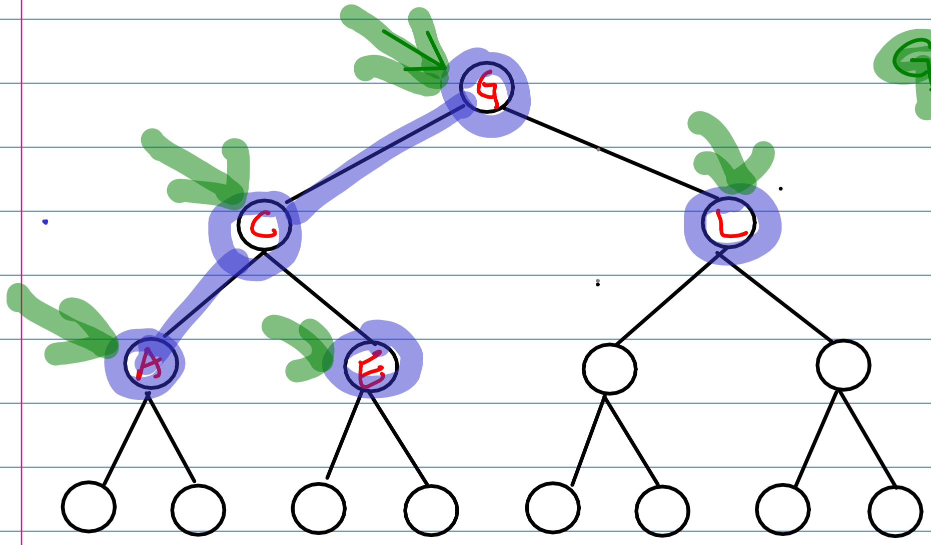
```
void PostOrder (Nptr T) {

if (T != NULL)
        {       PostOrder (T->R);
                PostOrder (T->L);
                Visit (T->Name);
        }
}
```

G-C-A-E-L

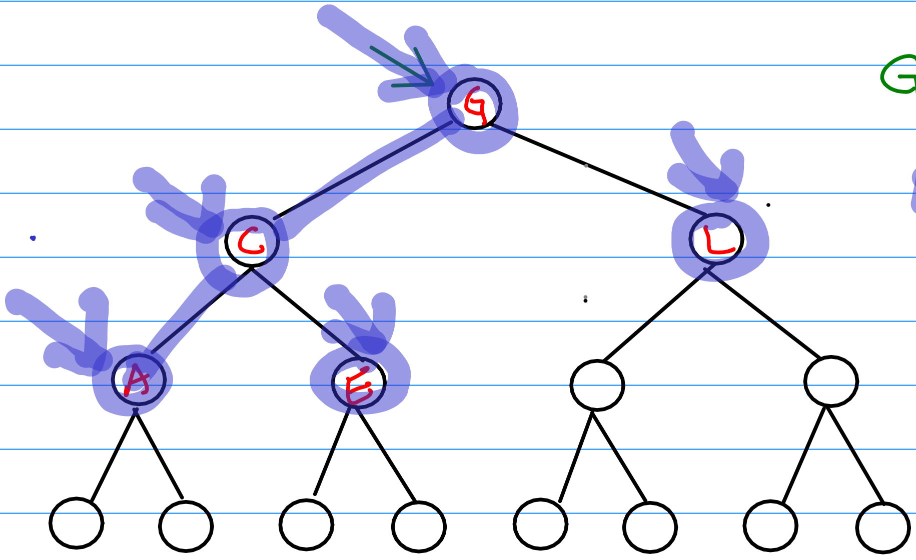
A C E G L

G



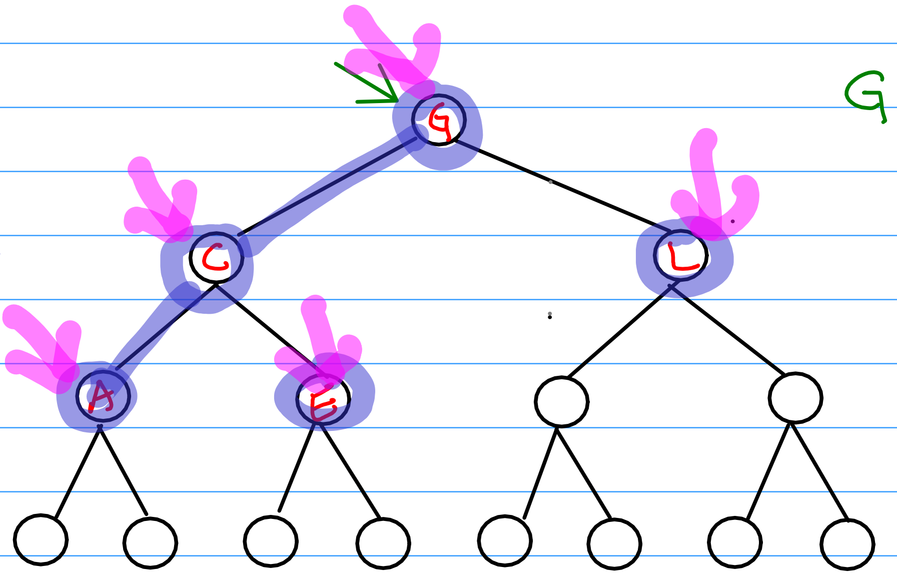


G C A E L



G

A C E G L

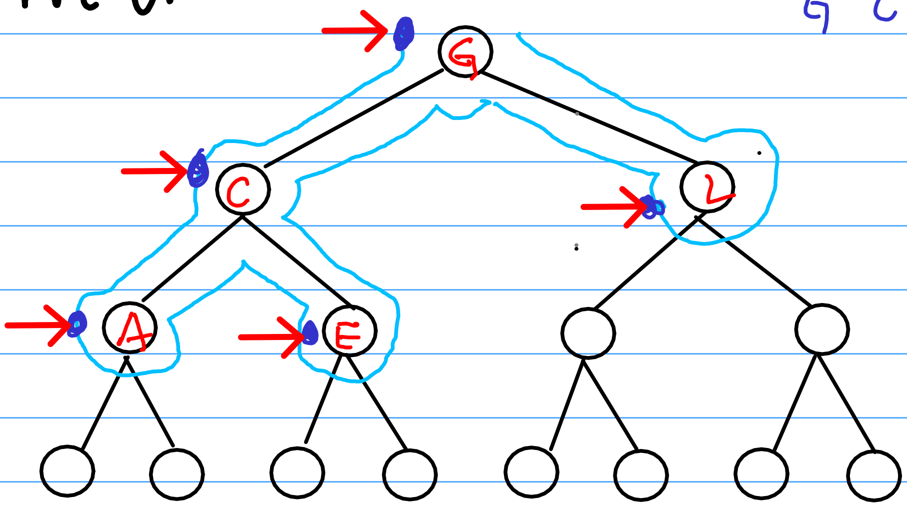


A E C L G

Left

Pre Order

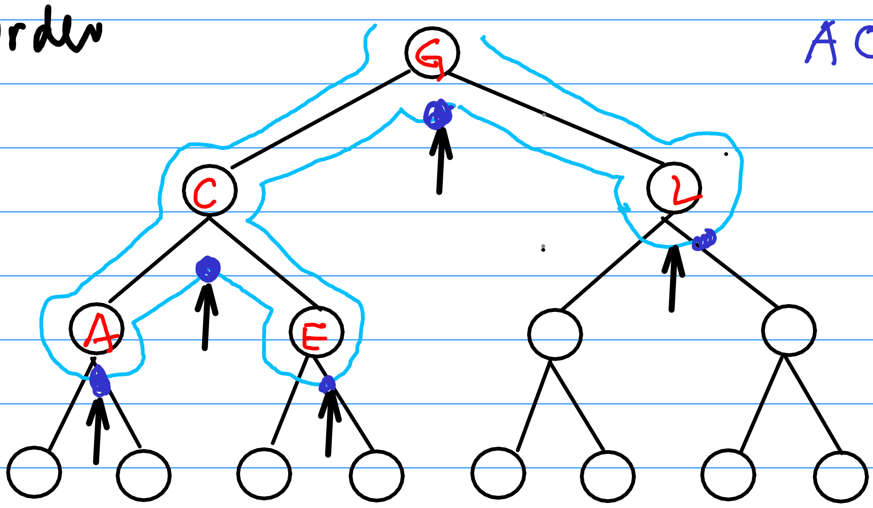
G C A E L



Mid

In Order

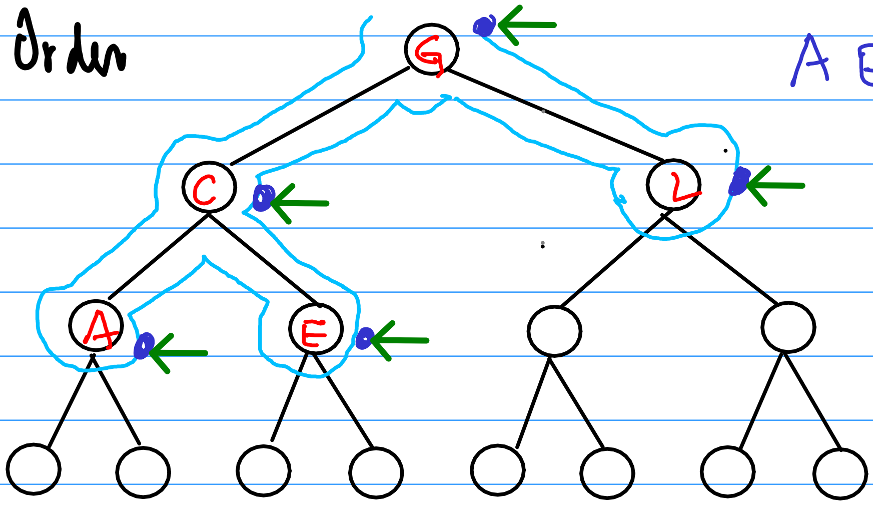
A C E G L



Right

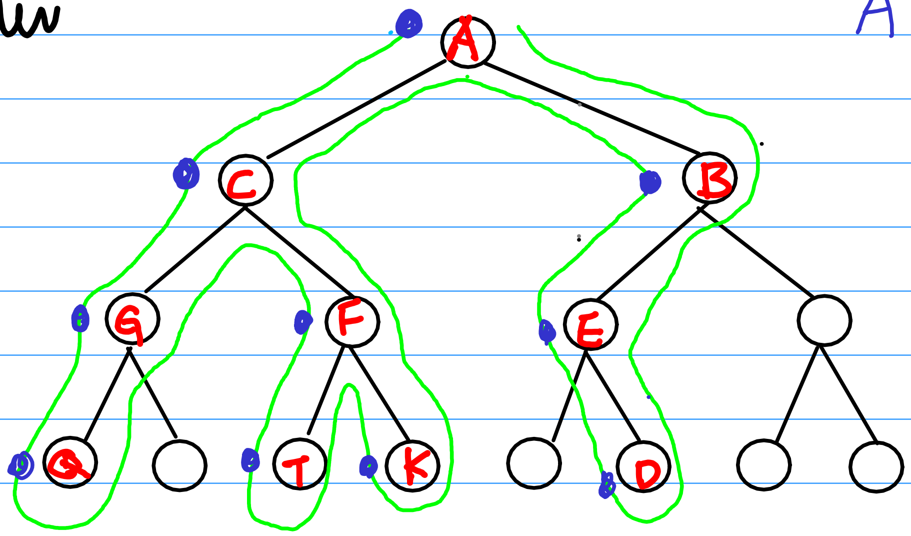
Post Order

A E C L G



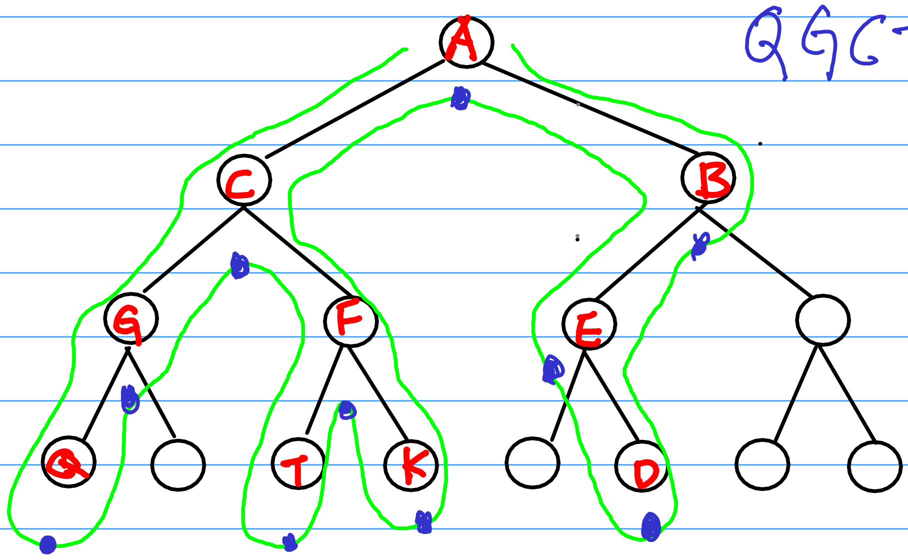
Pre Order

ACGQFTKBED



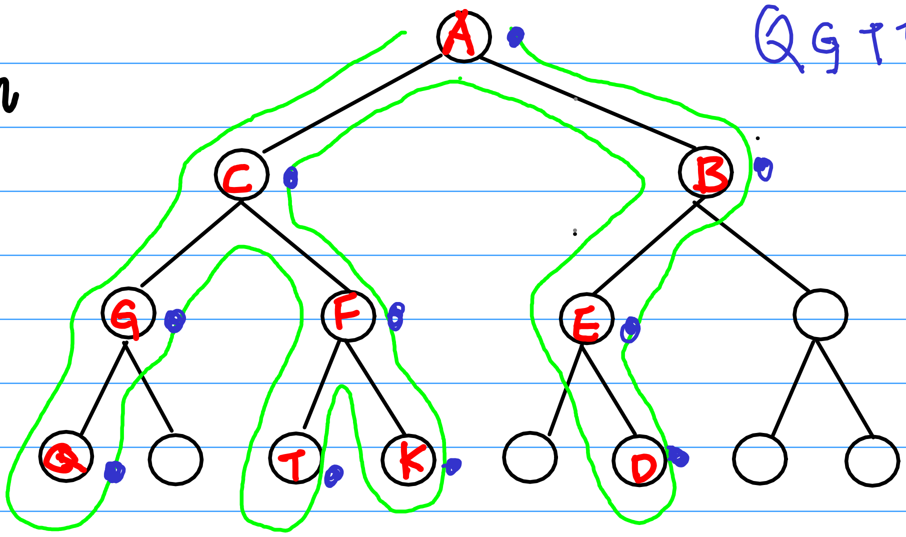
In Order

QGCTFKAEDB

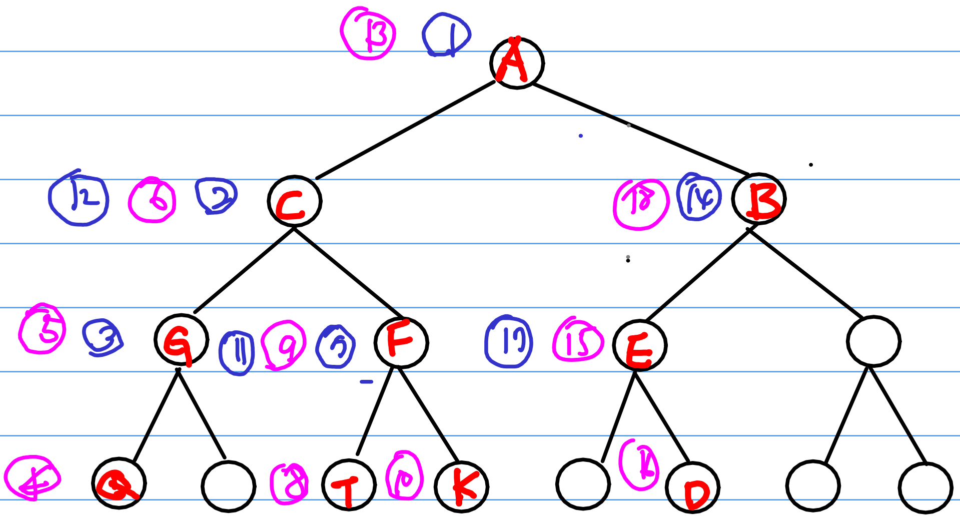


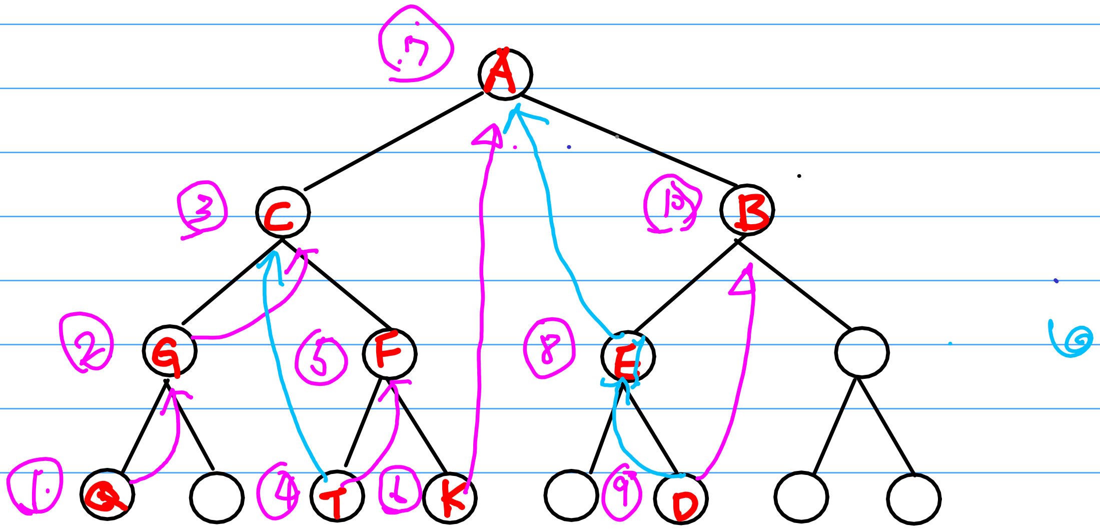
Post Order

QGTFKFCDEBA

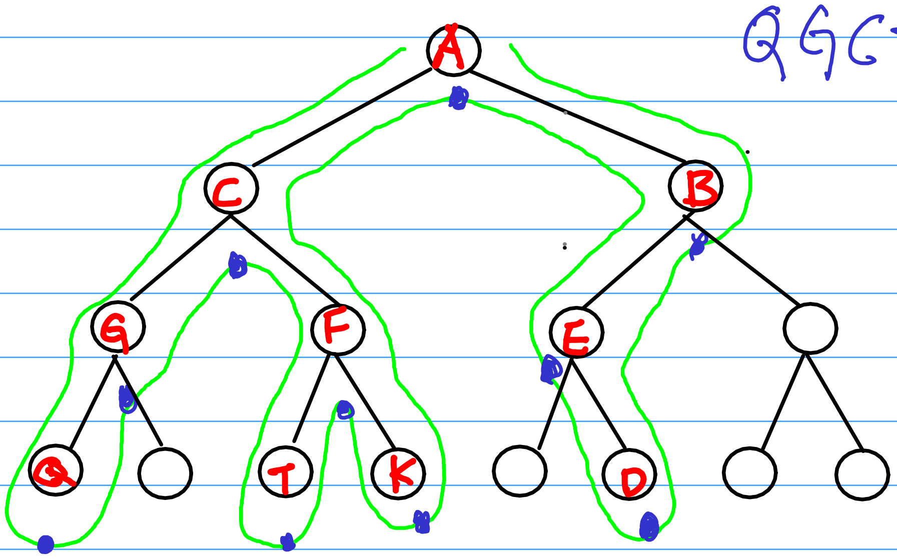


IN-ORDER

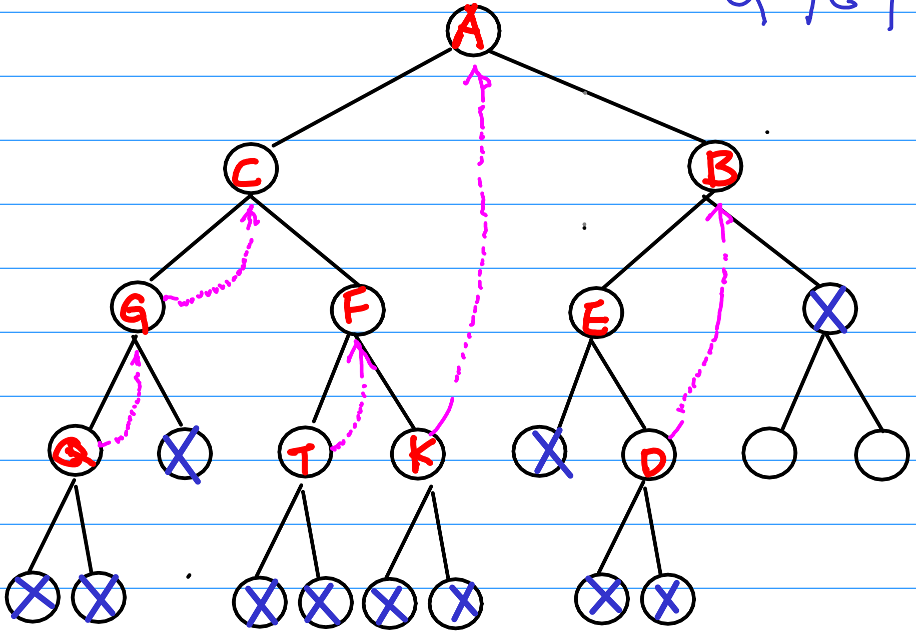




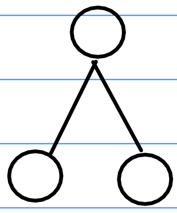
In Order



QGCTFKAEDB



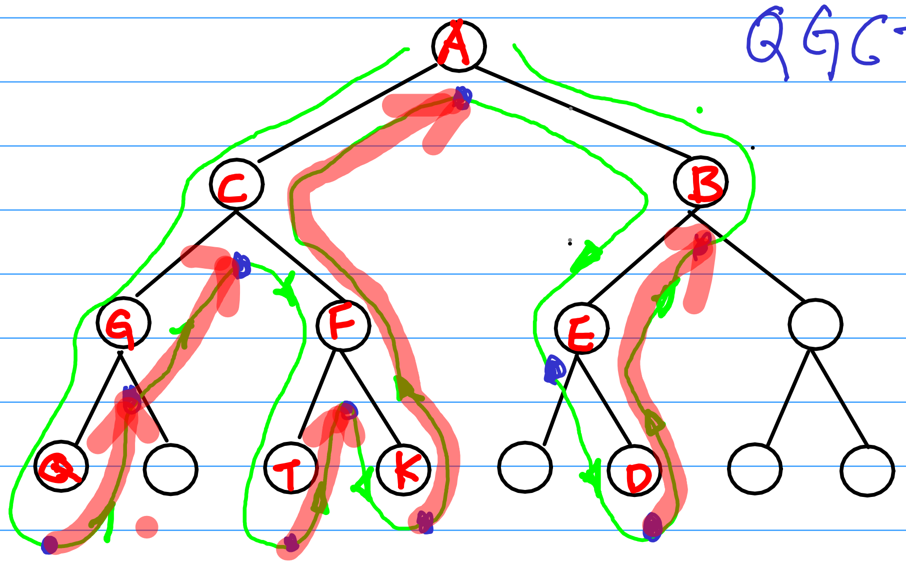
QGCTFKAEDB

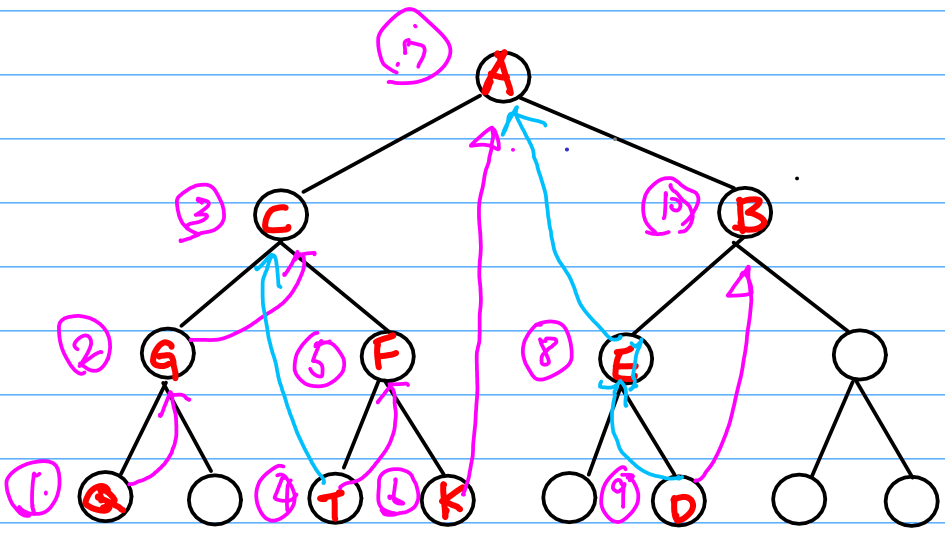


Without Recursion

InOrder

QGCTFKAEDR

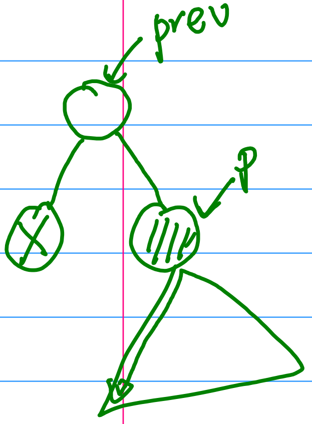


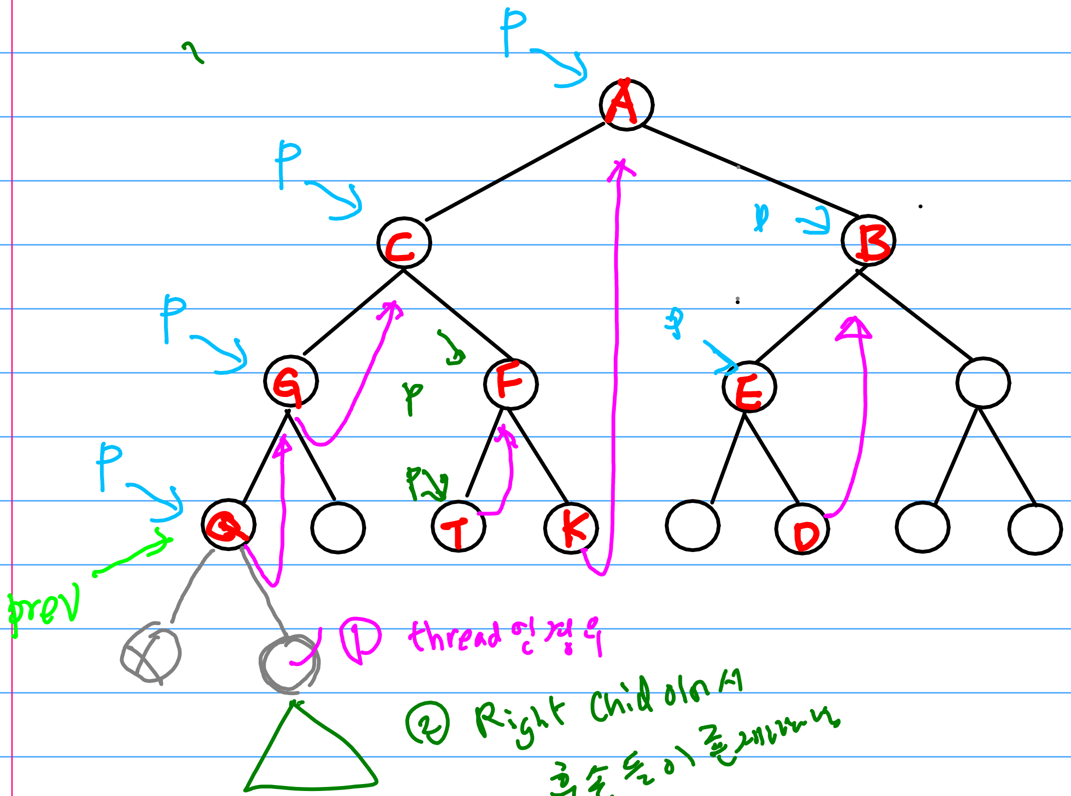


```
void InOrder(Nptr T) {

        if (T != NULL) {
                InOrder(T->L);
                Visit(T->Name);
                InOrder(T->R);
        }

}
```

```
void ThreadInOrder (Nptr p)
{
        if (p != NULL) {

                while (p->L != NULL)
                        p = p->L;

                while (p != NULL) {
                        Visit(p);
                        Prev = p;
                        p = p->R;

                        if (p != NULL && Prev->IsThread == FALSE)
                                while (p->L != NULL)
                                        p = p->L;
                }
        }
}
```

오른 Right child





① thread인 경우

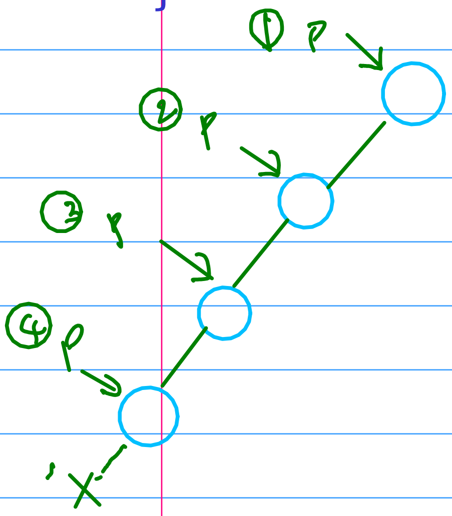
② Right Child에서 후손들이 존재하는

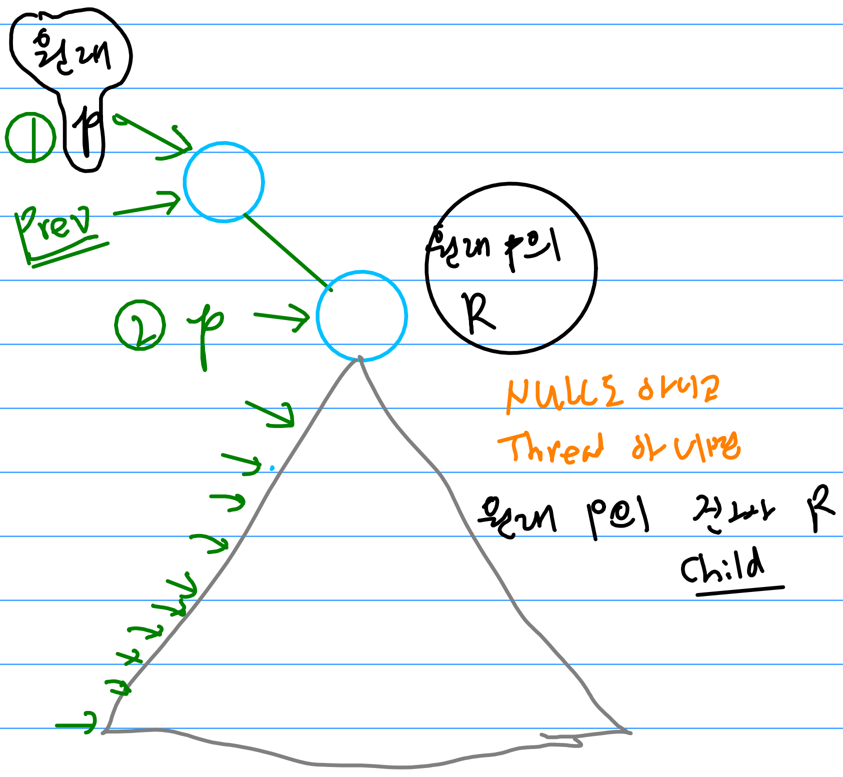
```
void ThreadInOrder (Nptr p)
{
        if (p != NULL) {

                while (p->L != NULL)
                        p = p->L;

                while (p != NULL) {
                        Visit(p);
                        Prev = p;
                        p = p->R;

                        if (p != NULL && Prev->IsThread == FALSE)
                                while (p->L != NULL)
                                        p = p->L;
                }
        }
}
```

왼쪽 끝 까지 간다 (leaf node)

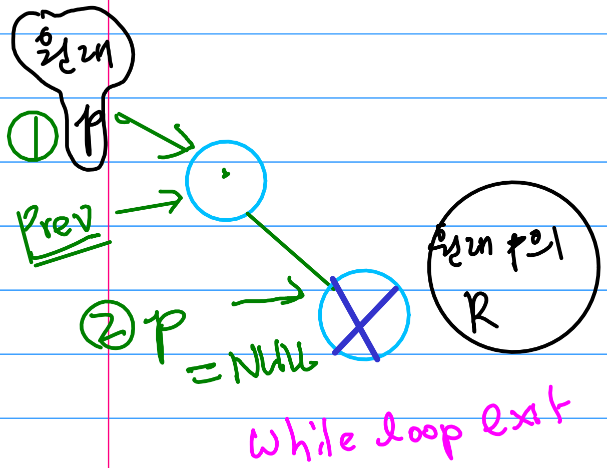
p가 NULL이 아니고 thread도 아닐때

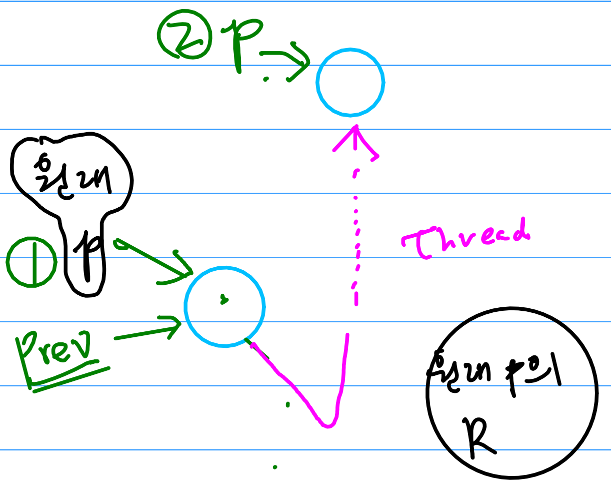
왼쪽 끝 까지 간다 (leaf node)





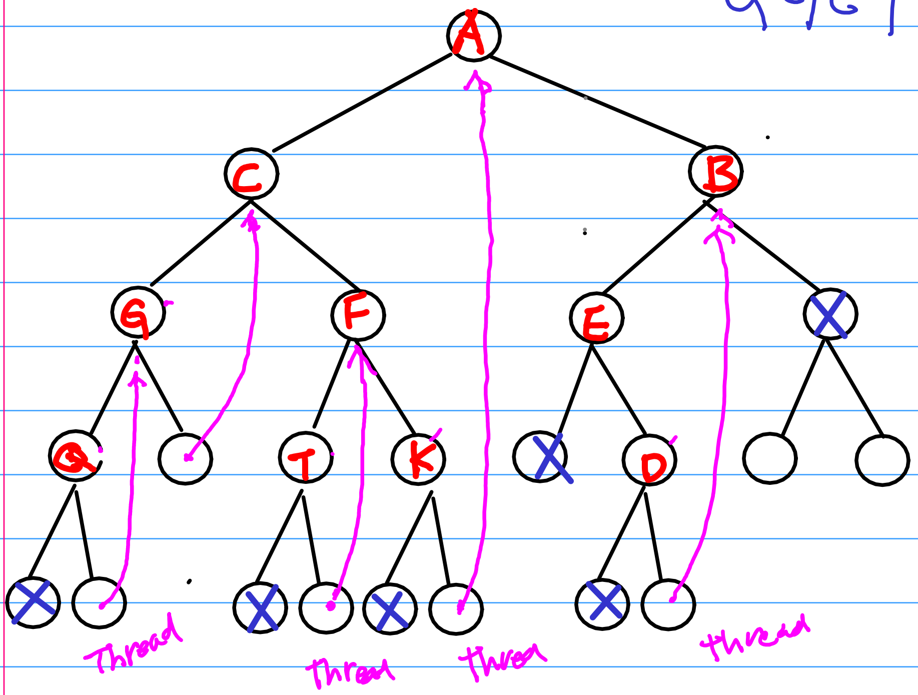
```
void ThreadInOrder (Nptr p)
{
        if (p != NULL) {
            while (p->L != NULL)                    왼쪽 끝까지 간다
                    p = p->L;                       (leaf node)

            while (p != NULL) {
                    Visit(p);
                    Prev = p;
                    p = p->R
                                                          p가 NULL이 아니고
                                                            thread도 아닐 때
                    if (p != NULL && Prev->IsThread == FALSE)
                        while (p->L != NULL)        왼쪽 끝까지 간다
                                p = p->L;           (leaf node)
            }
        }
}
```

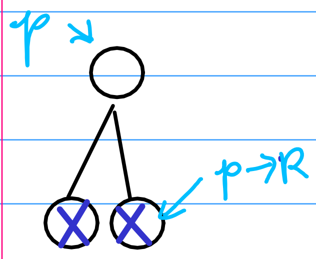


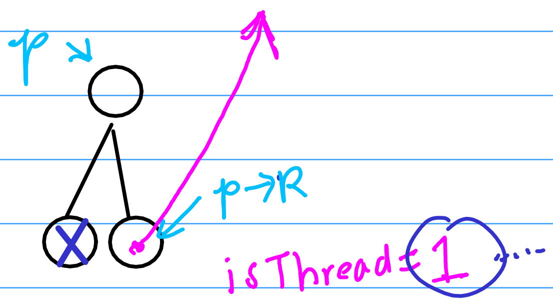


if 문장은 수행하지 않고
while loop 문장은 계속 수행

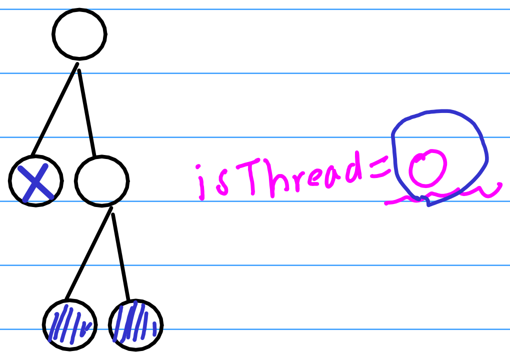
QGCTFKAEDB





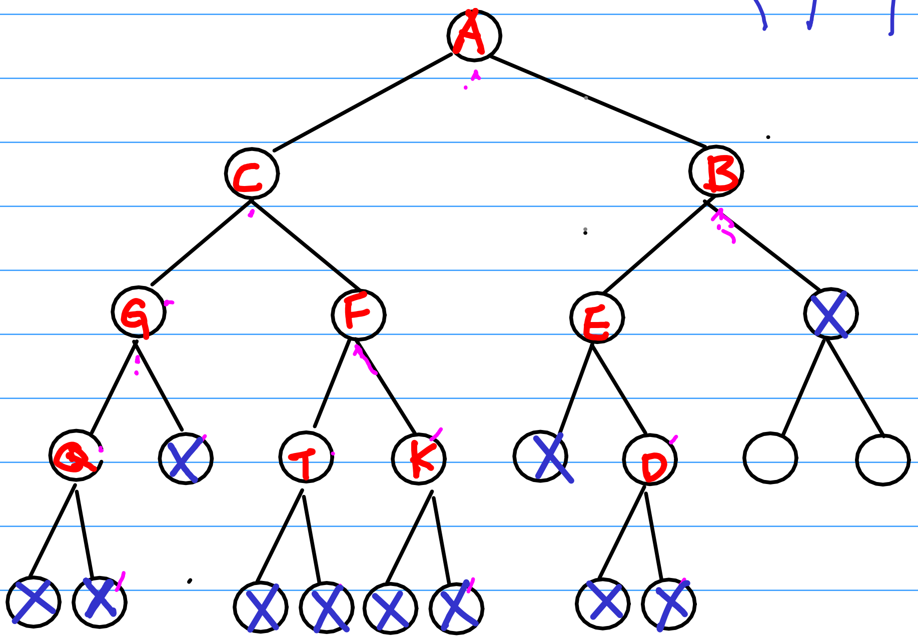


Ⓡ이 Thread일시
inOrder에서
successor 될시



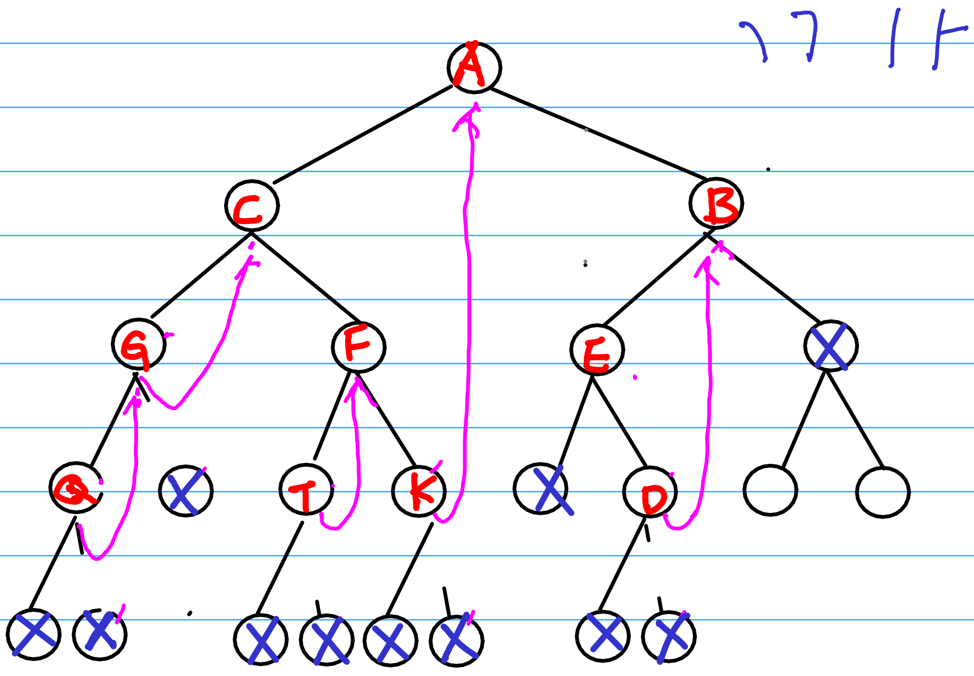
Ⓡ이 thread가
아니고 오른
Right Child로
될시

[illegible]



```
typedef struct TType {
        char Id;  = Q
        struct TType * L; =NULL
        struct TType * R; = NULL
} node;

typedef node * Nptr;
```

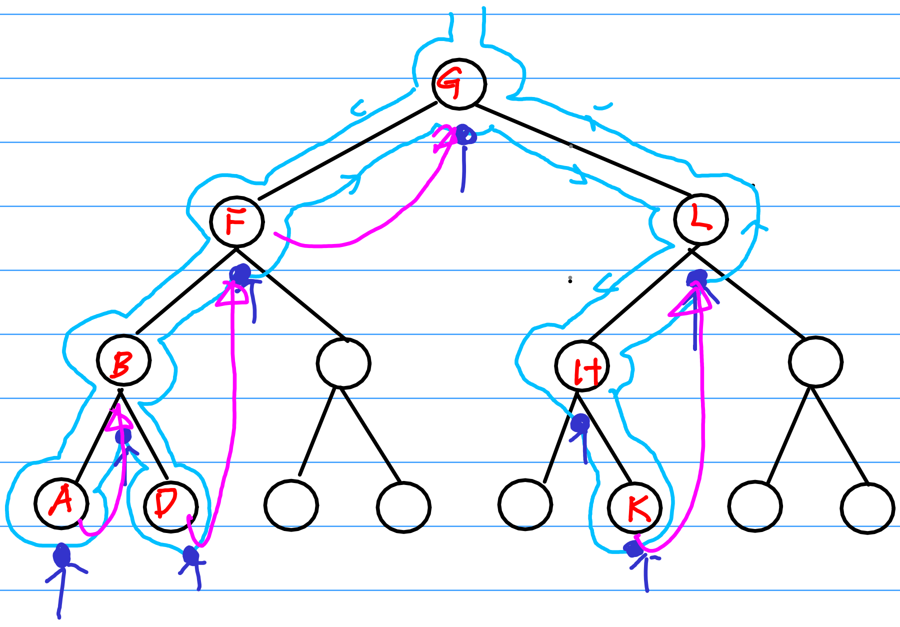
```
typedef struct TType {
        char Id;
        int isThread;
        struct TType * L;
        struct TType * R;
} node;

typedef node * Nptr;
```



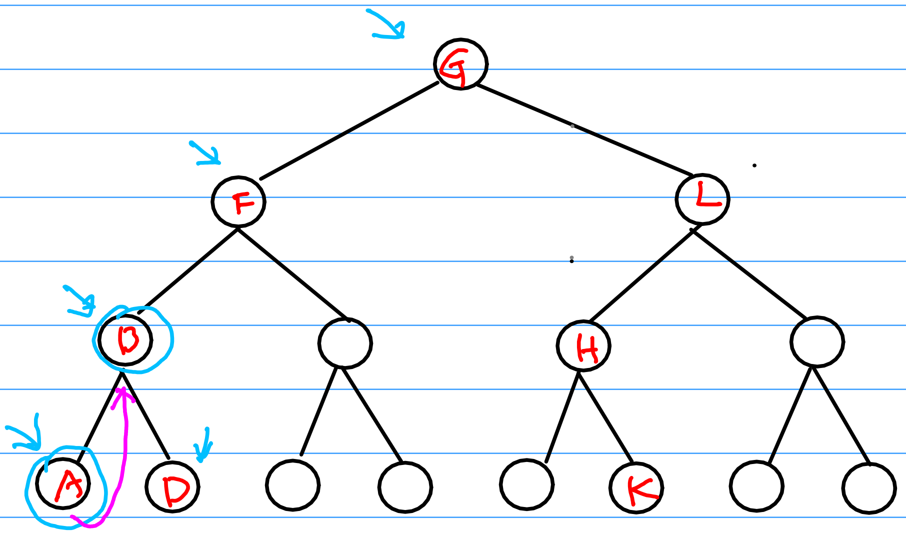
```
typedef struct TType {
        char Id; = Q
        struct TType * L; = NULL
        struct TType * R; = NULL
} node;

typedef node * Nptr;
```

```
typedef struct TType {
        char Id;
        int isThread;
        struct TType * L;
        struct TType * R;
} node;

typedef node * Nptr;
```

Inorder

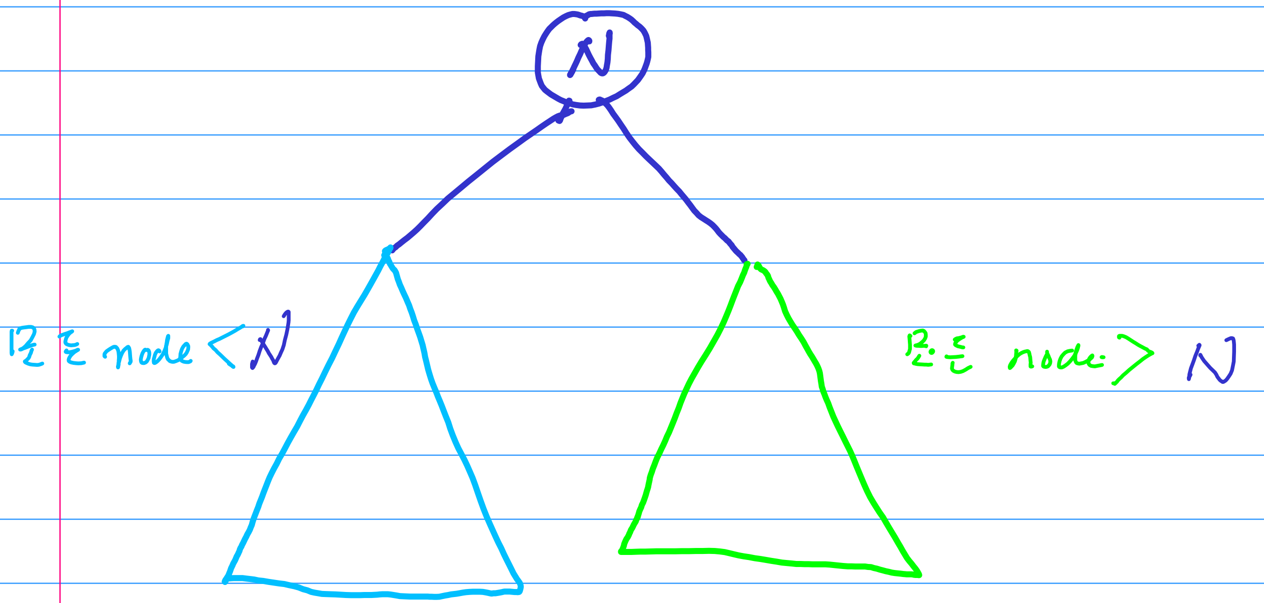


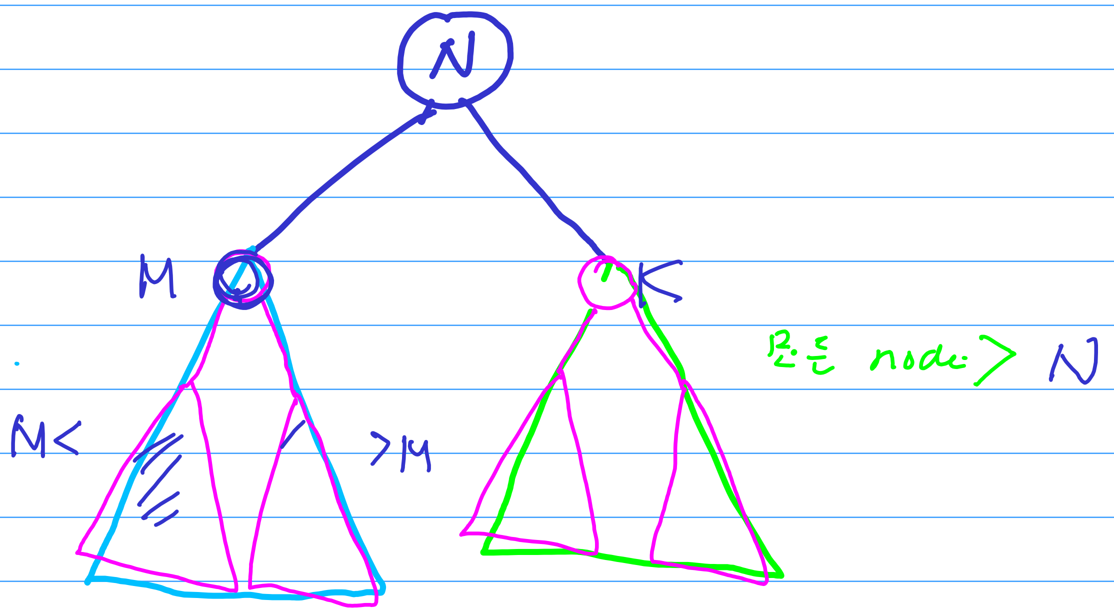
ABDFGHKL



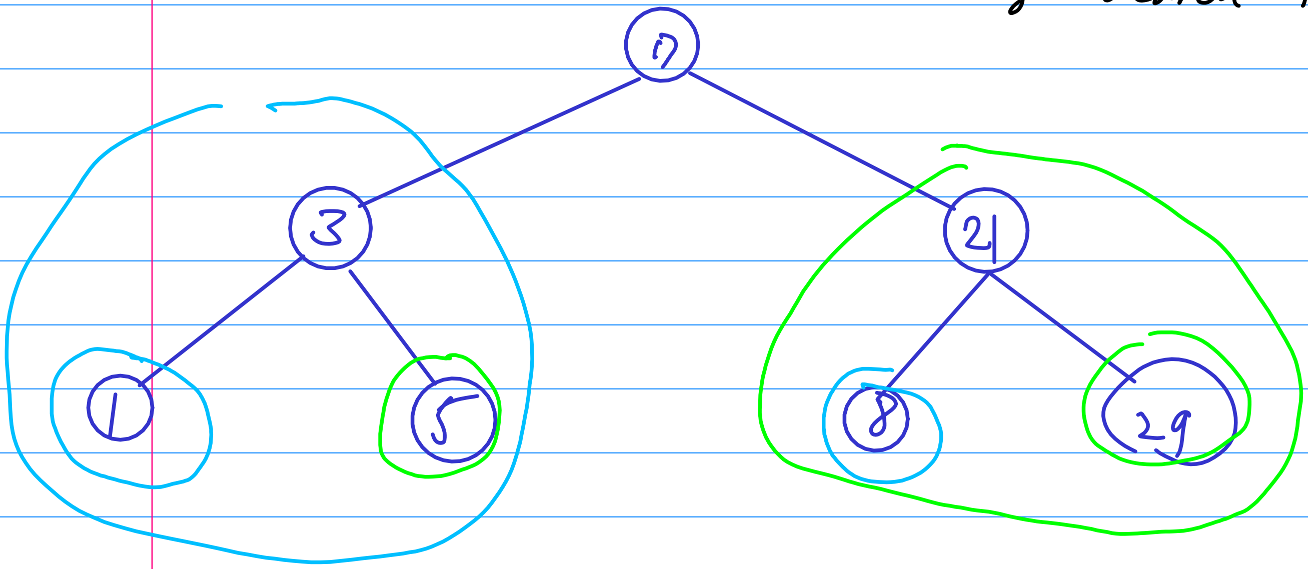
A

Binary Search Tree

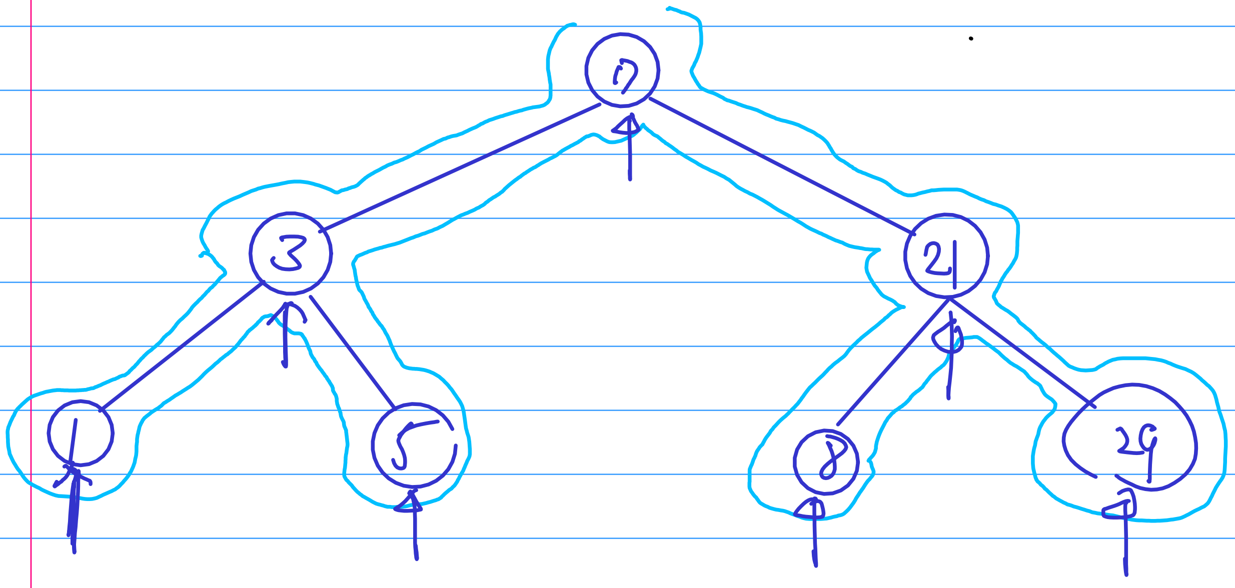




Binary Search Tree



Binary Search Tree 에서 In Order → $\boxed{\text{Sorting}}$

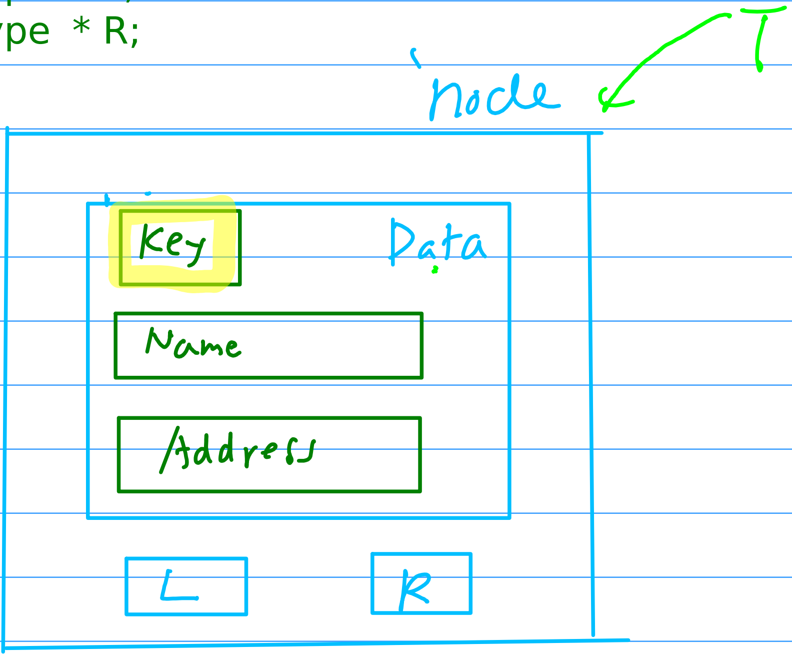


1 3 5 7 8 21 29

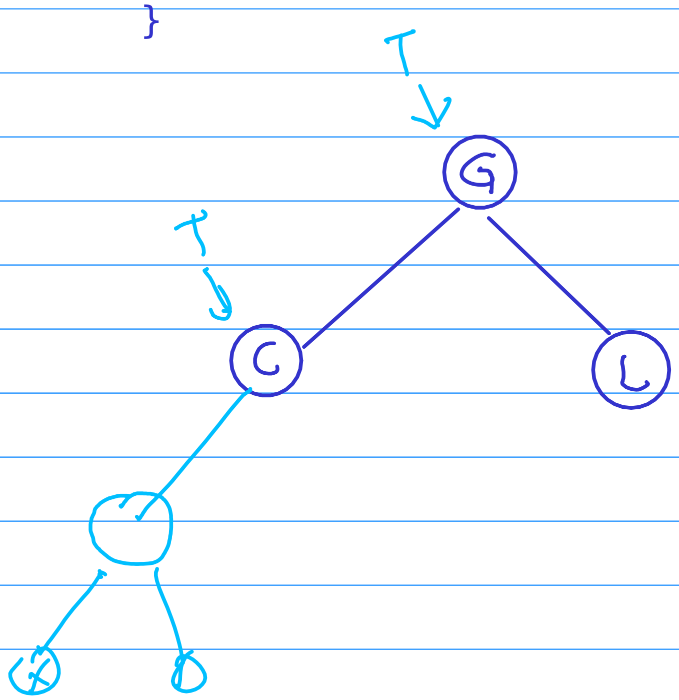
```
Nptr Search (Nptr T, int Key) {

        if (T == NULL)
              printf("No Such Node");

        else if (T->Data.Key == Key)
              return T;

        else if (T->Data.Key > Key)
              return Search (T->L, Key);

        else
              return Search (T->R, Key);

}
```

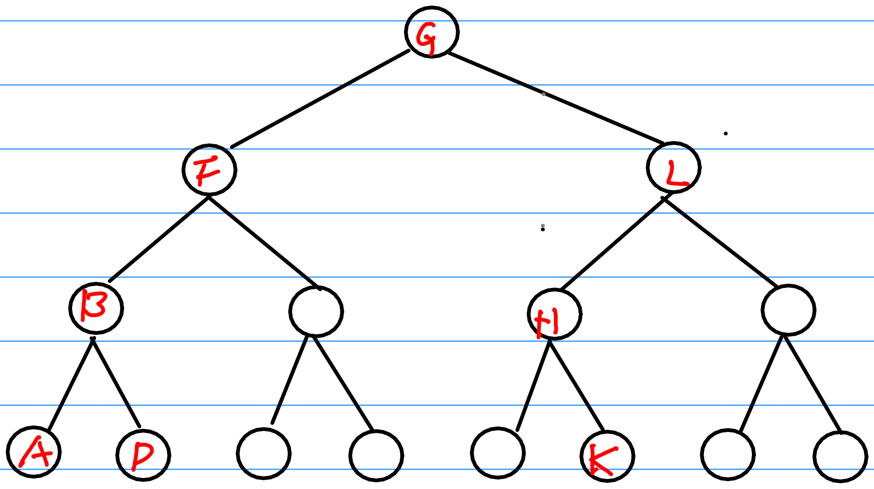
```
typedef struct dataRecord {
        int   Key;
        char Name[12];
        char Address[200];
} dataType;

typedef struct TType {
        dataType      Data;
        struct TType  * L;
        struct TType  * R;
} node;
```

T → Data. Key

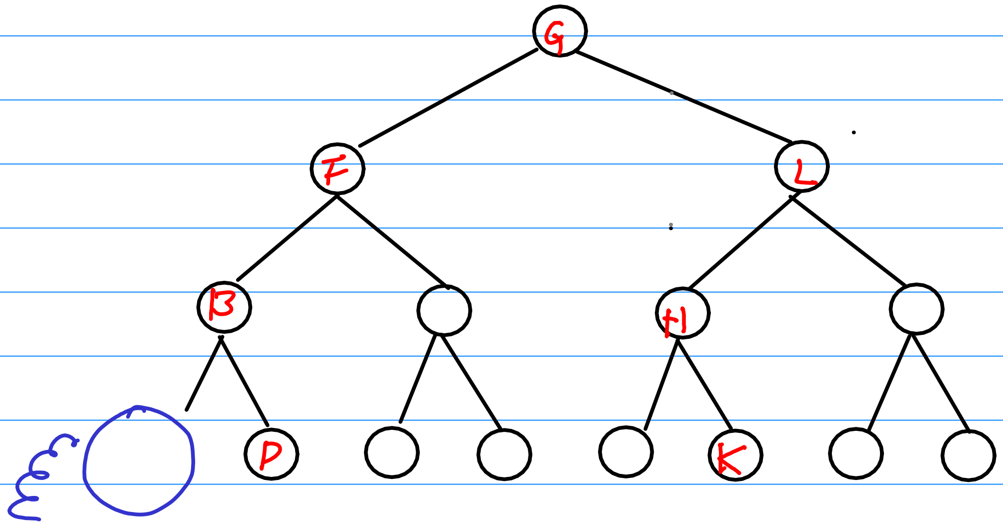


```
Nptr Insert(Nptr T, inst Key) {
    if (T == NULL) {
        T = (node *) malloc (sizeof (node));
        T->Data.Key = Key;
        T-> L = NULL;
        T-> R = NULL:
    }
    else if (T-> Data.Key > Key)
        T->L = ~~Insert~~(T->L, Key);
    else
        T->R = ~~Inset~~(T->R, Key);

    retrun T;
}
```




G
F
L
B
H
A
D
K

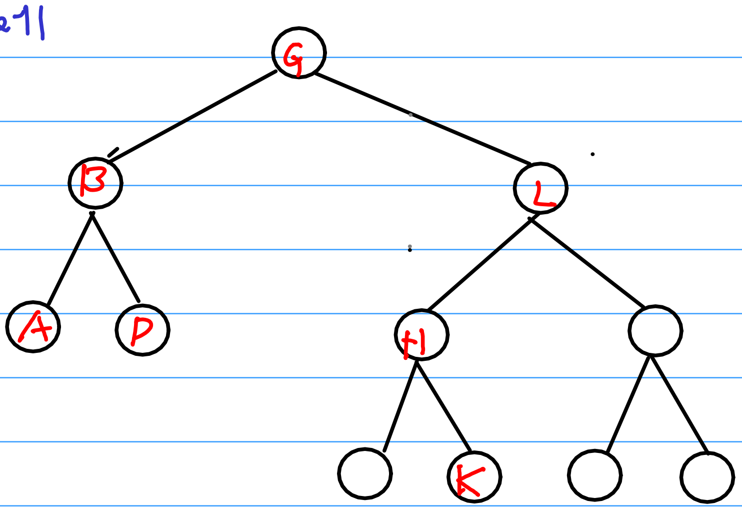
① leaf node A 삭제

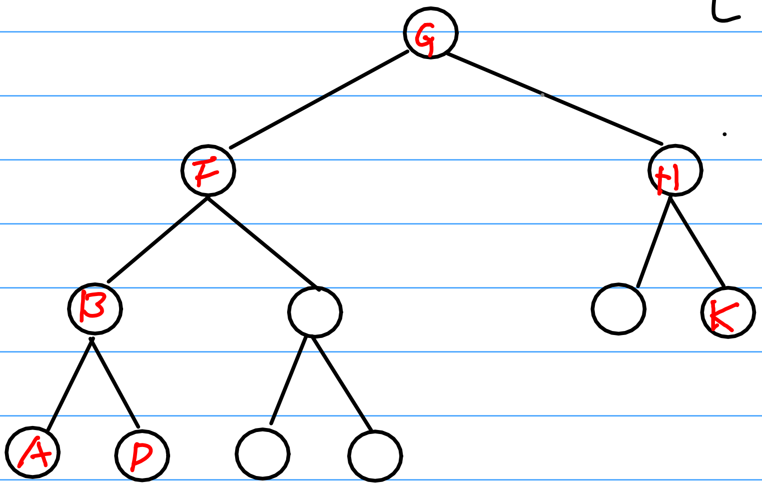


② 자식 node가 1개 인 경우

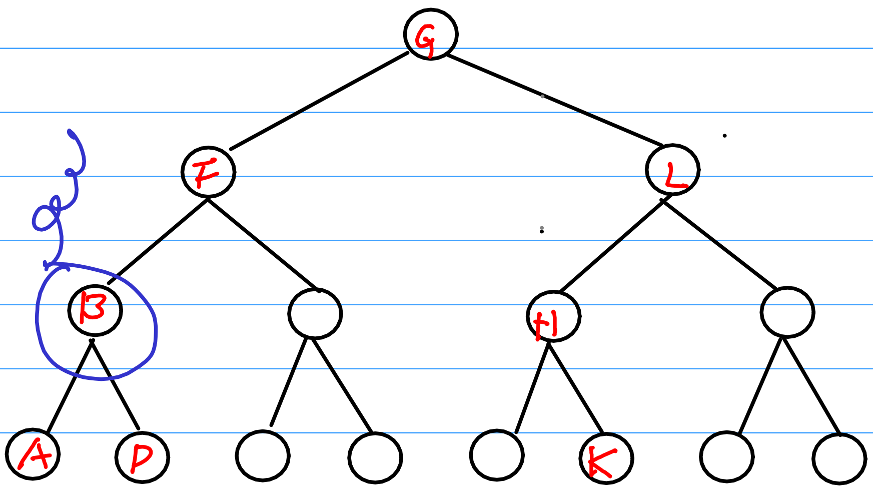
L 만 있거나 R 만 있는 경우 삭제

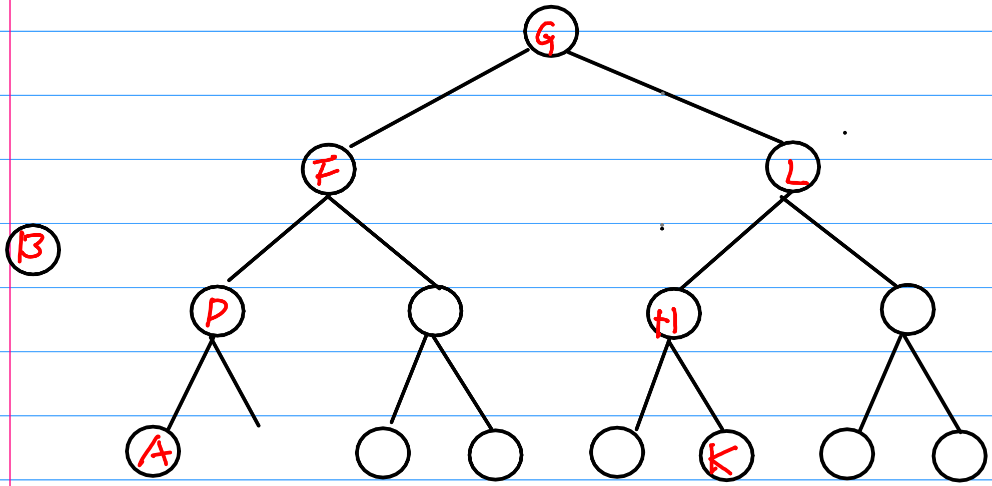
F node 삭제

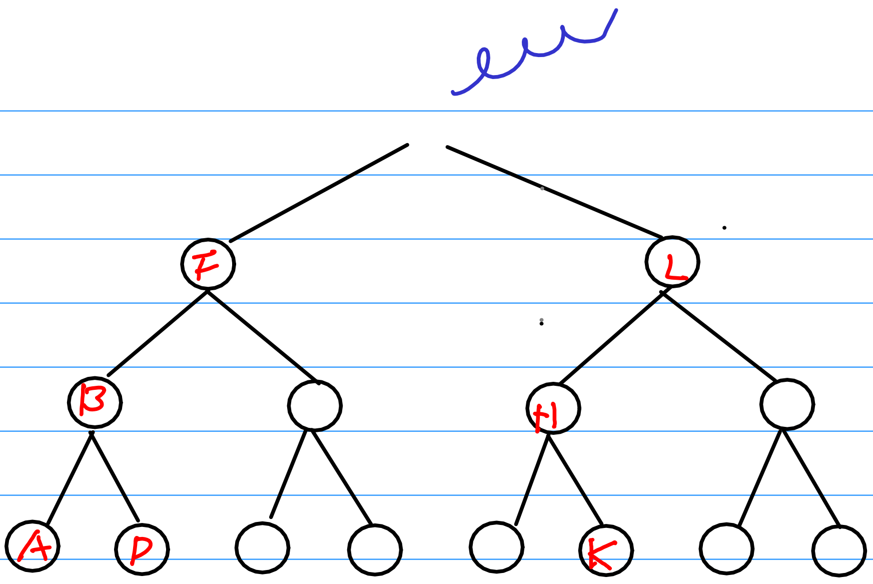


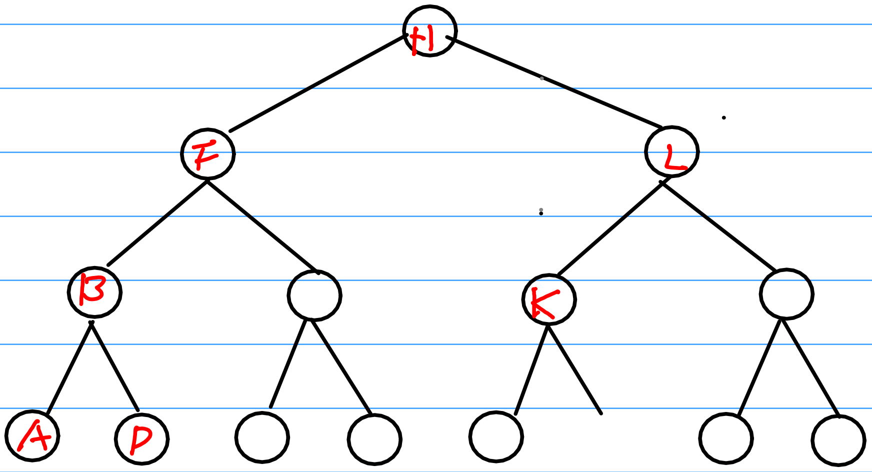


L node 삭제 결과


G
F
L
B
H
A
D
K


G
F
L
B
D
H
A
K


F
L
B
H
A
D
K


H
F
L
B
K
A
D