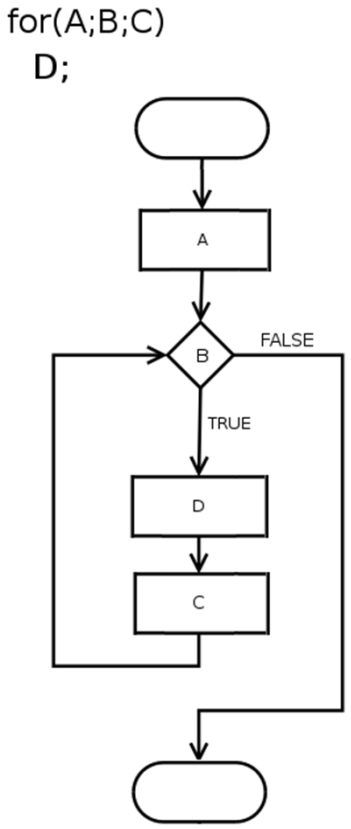
Clickseo.com

자료구조 및 알고리즘

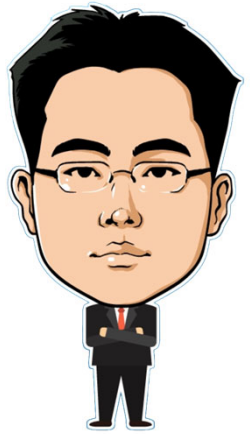


트리
(Tree)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

백문이불여일타(百聞而不如一打)

- 트리의 이해
- 이진 트리
- 우선 순위 큐와 힙

트리의 이해 (1/3)

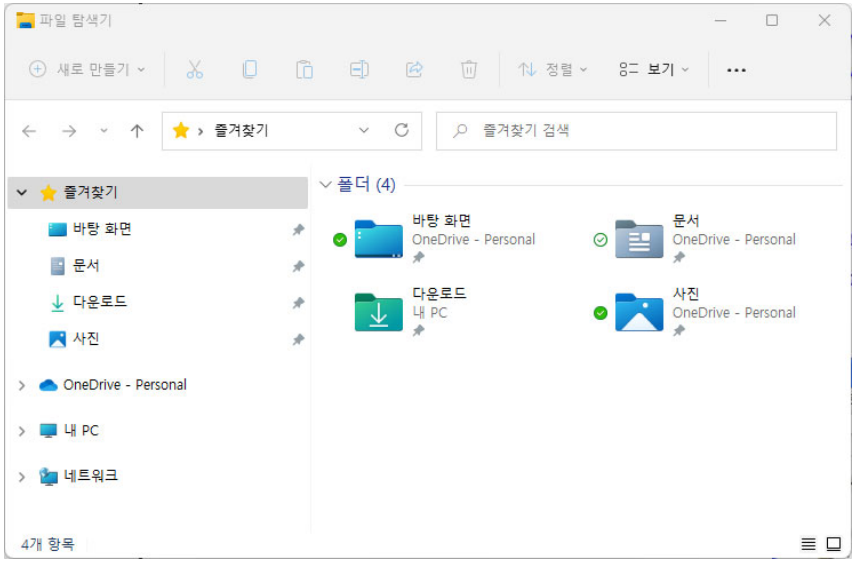
● 트리(Tree)

❍ 트리의 정의

- 원소들 간에 **1:多 관계**를 가지는 **비선형 자료구조**
- 원소들 간에 **계층 관계**를 가지는 **계층형 자료구조**
- 상위 원소에서 하위 원소로 내려가면서 확장되는 **나무 모양의 구조**

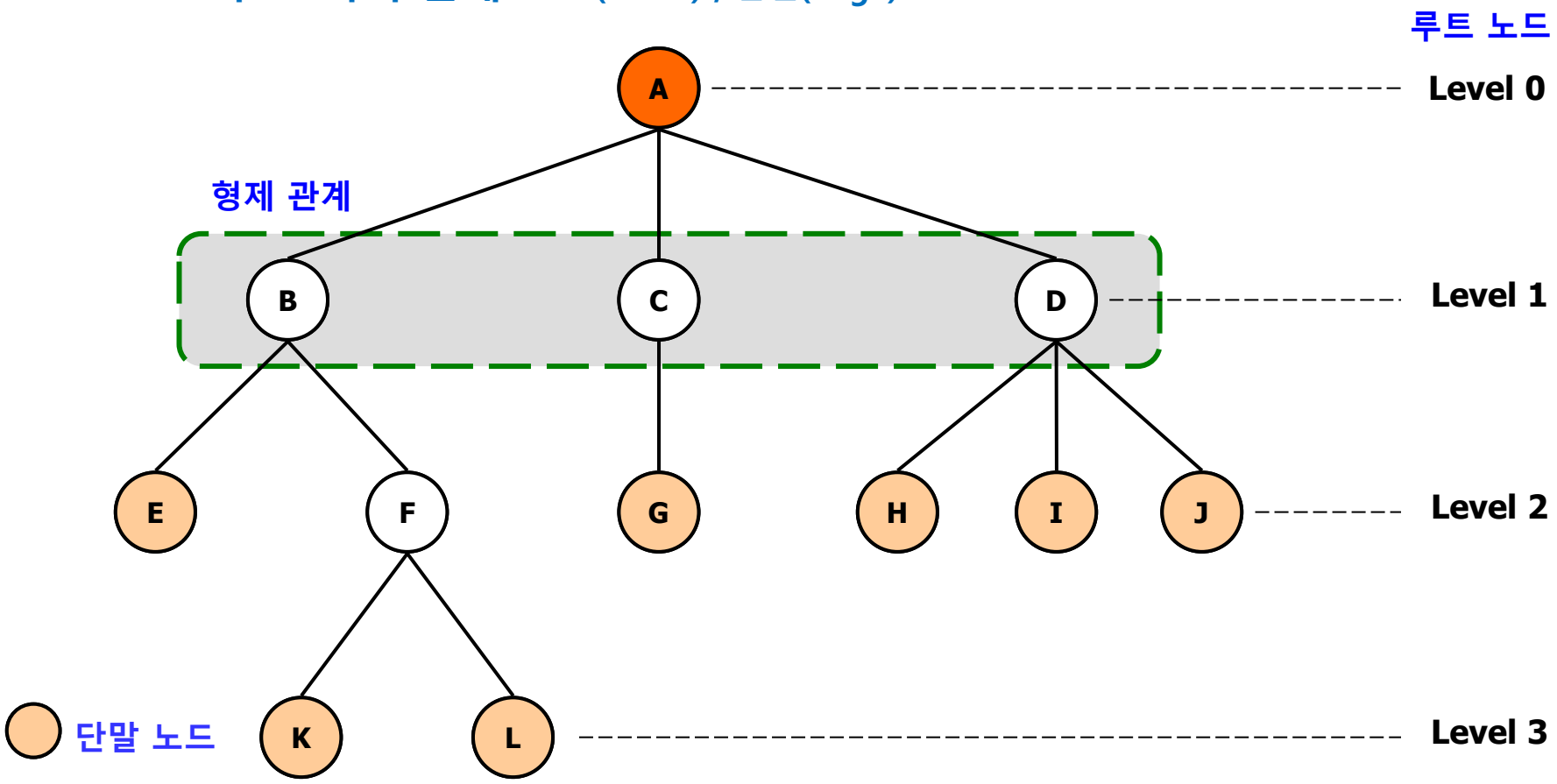
❍ 트리 구조의 예

- 컴퓨터 디렉터리(Directory) 구조
- 기업 구조(Organization Chart)
- 족보(Family Tree)
- 결정 트리(Decision Tree)



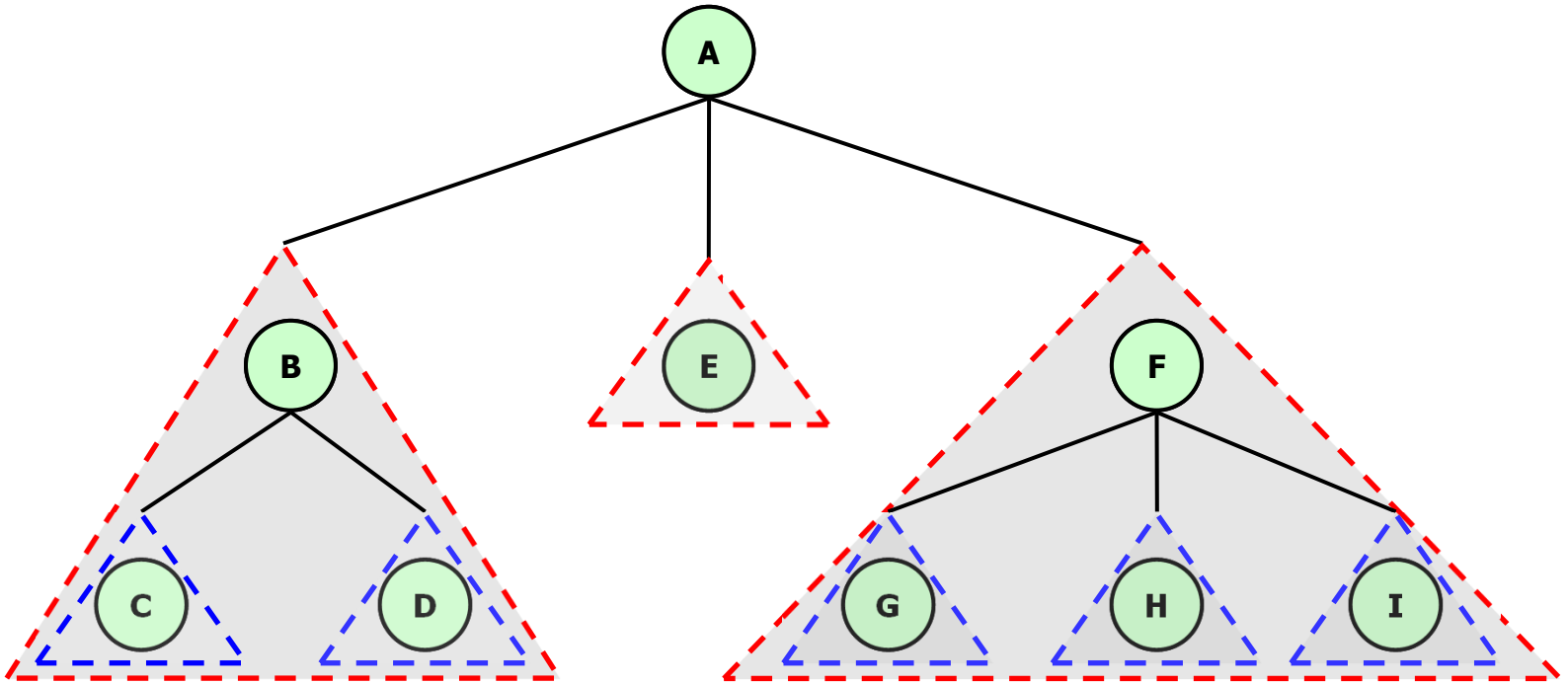
트리의 이해 (2/3)

- 트리 구조
  - 부모-자식 관계: 노드(Node) , 간선(Edge)



트리의 이해 (3/3)

- **트리 구조: 부분 트리**
  - **부분 트리(Subtree)**
    - 자식 노드들은 각각 독립하여 새로운 트리를 구성할 수 있다.
    - 각 노드는 자식 노드 수만큼의 서브 트리를 갖는다.



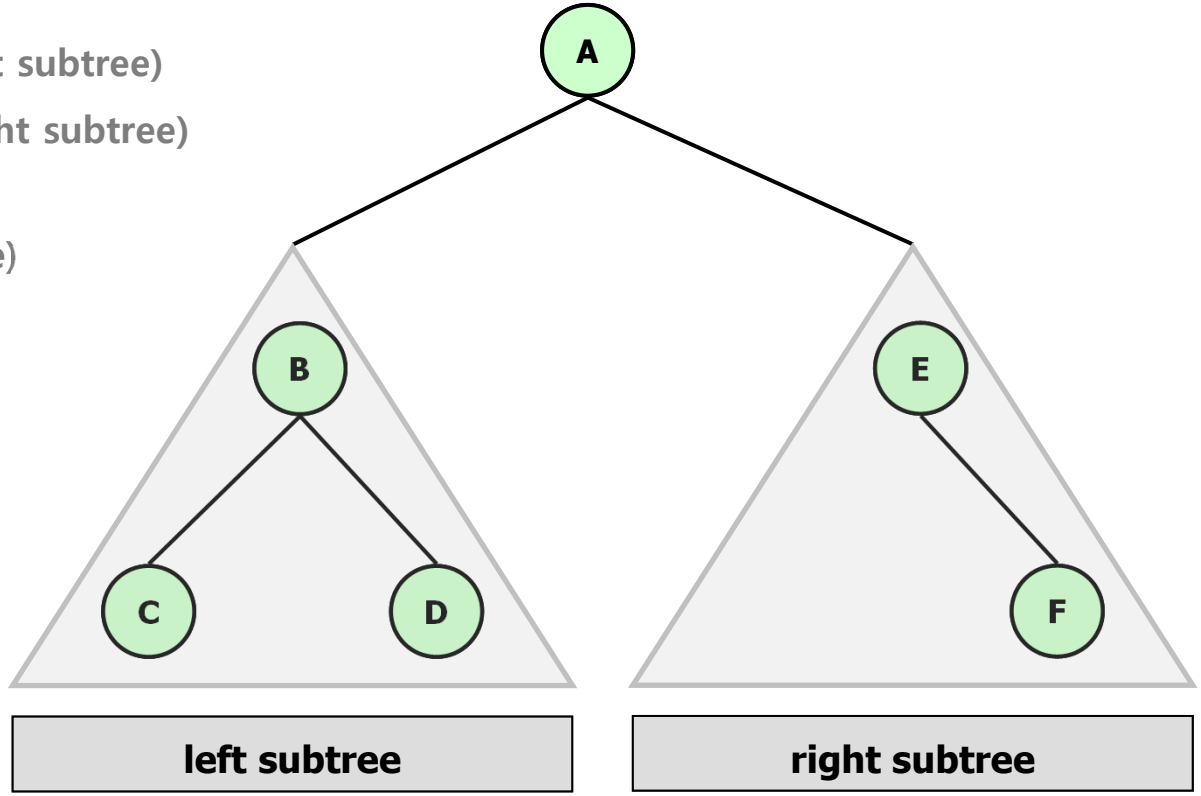
이진 트리

백문이불여일타(百聞而不如一打)

- 트리의 이해
- **이진 트리**
  - 이진 트리 순회
  - 이진 트리 구현
- 우선 순위 큐와 힙

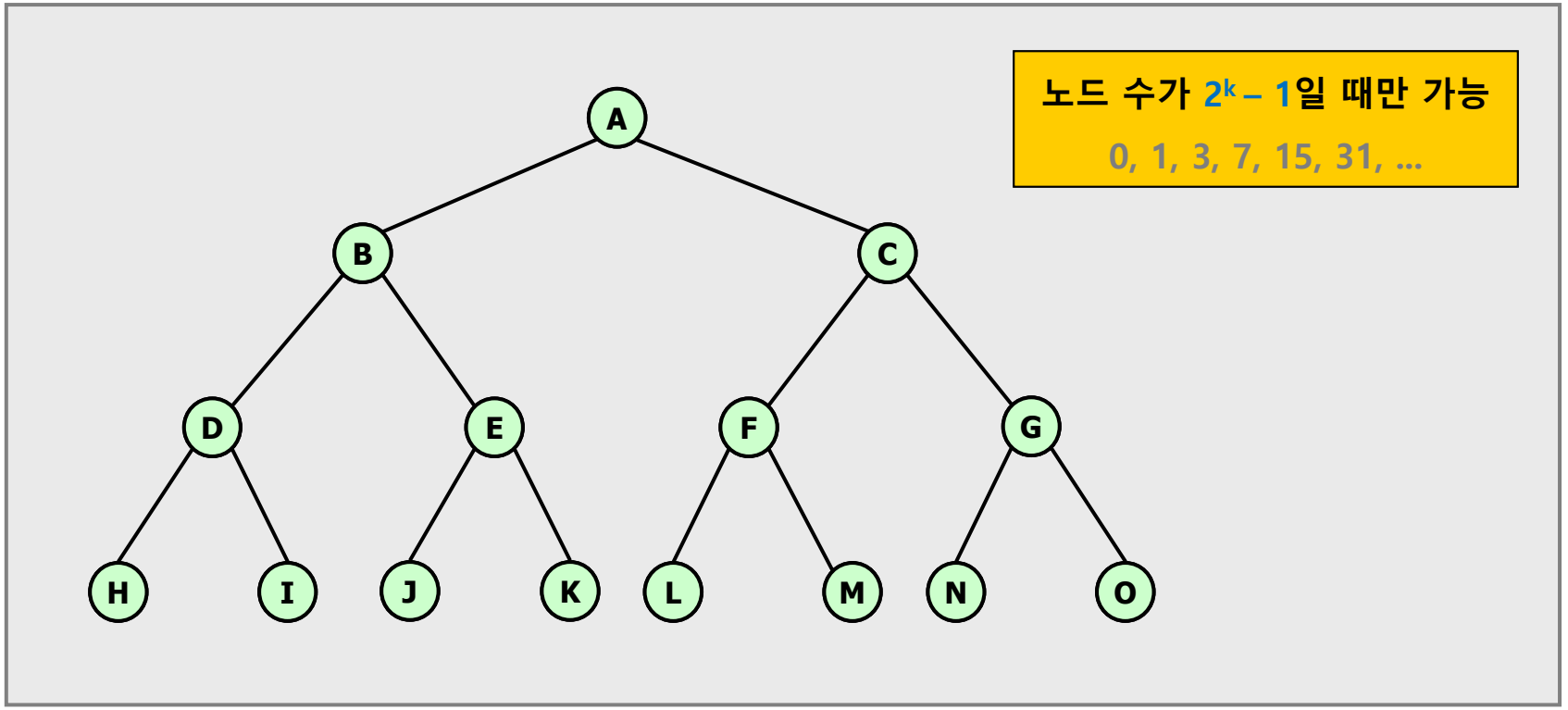
이진 트리 (1/4)

- **이진 트리**(Binary Tree)
  - 최대 두 개까지의 자식 노드를 가질 수 있는 트리
    - 하나의 노드는 0, 1, 혹은 2개의 서브 트리를 가질 수 있다.
    - 좌 서브 트리(left subtree)
    - 우 서브 트리(right subtree)
    - 널 트리(null tree)



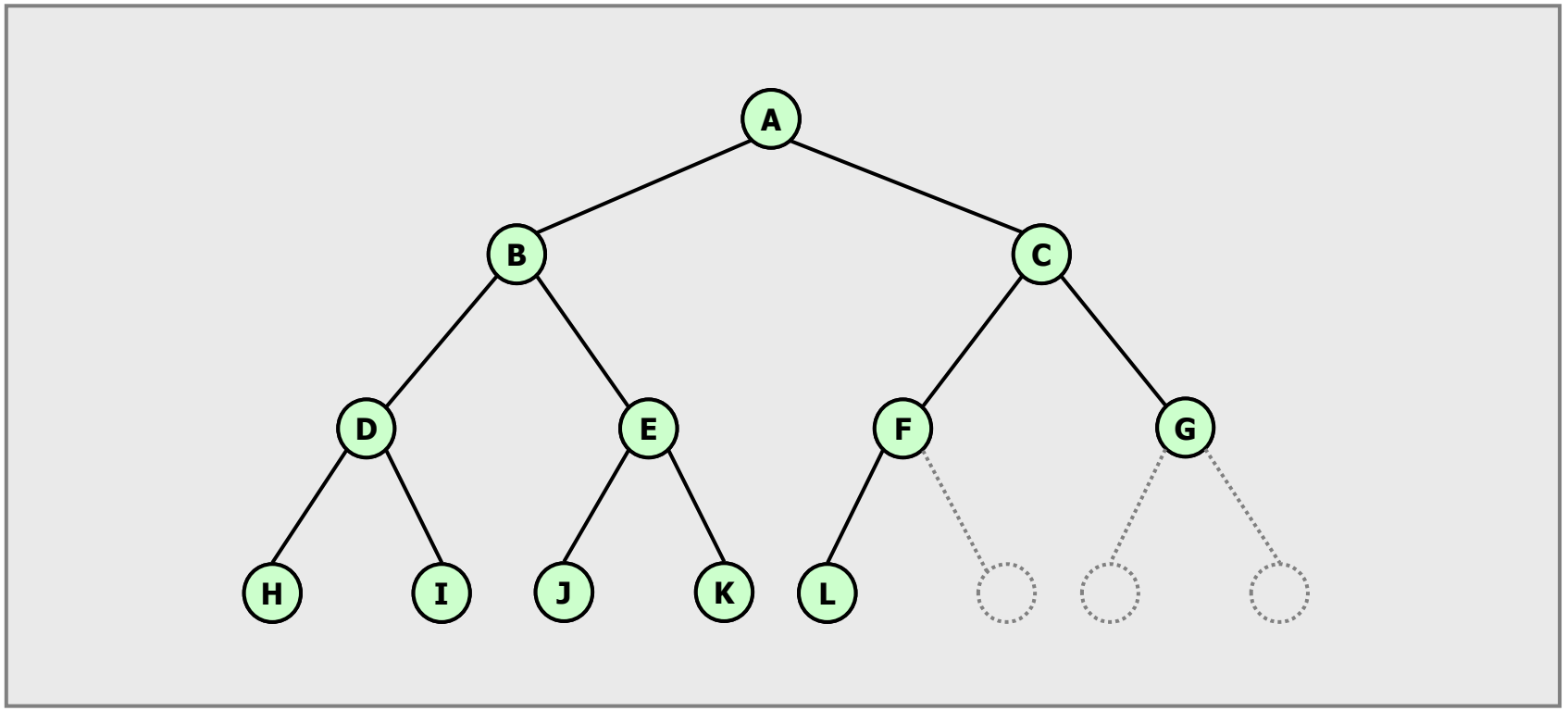
이진 트리 (2/4)

- **포화 이진 트리(Full Binary Tree)**
  - 루트로부터 시작해서 모든 노드가 정확히 두 개씩의 자식 노드를 가지도록 꽉 채워진 트리



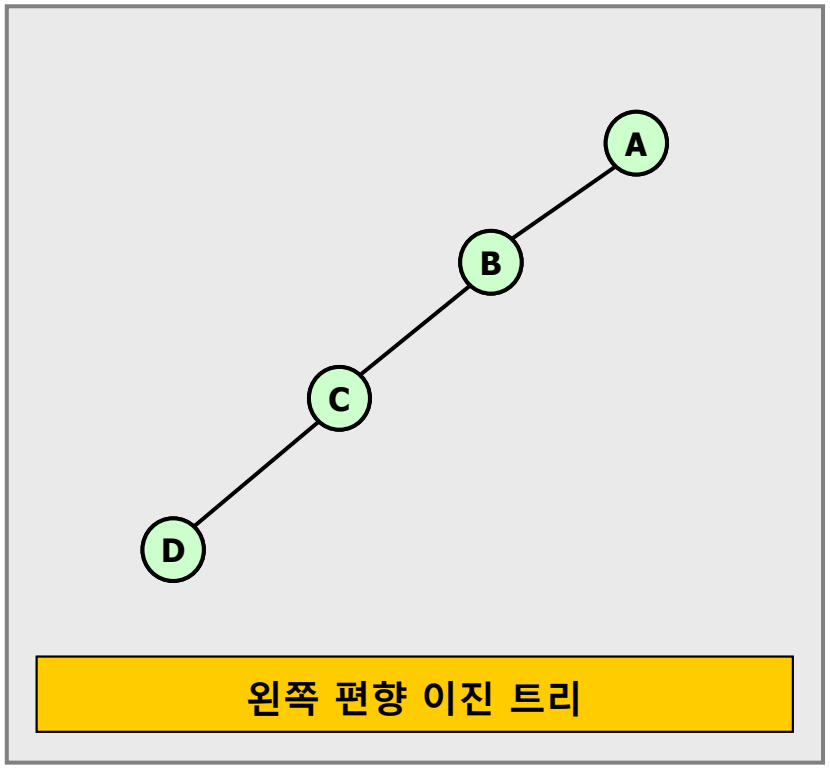
이진 트리 (3/4)

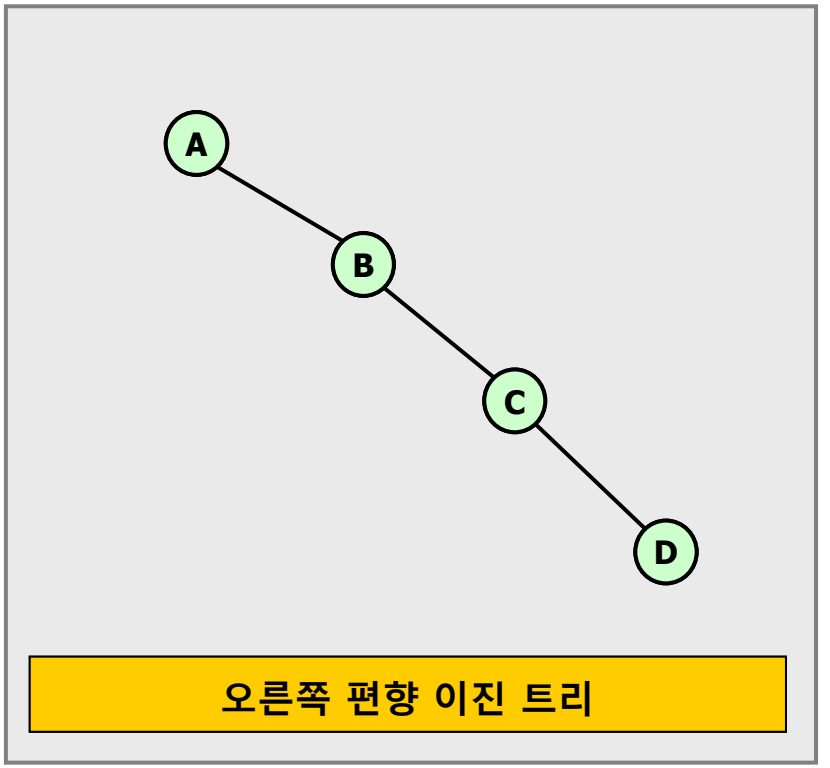
- **완전 이진 트리**(Complete Binary Tree)
  - 노드의 수가 맞지 않아 포화 이진 트리를 만들 수 없으면 맨 마지막 레벨은 왼쪽부터 채워 나간다.



이진 트리 (4/4)

- **편향 이진 트리**(Skewed Binary Tree)
  - 이진 트리 중에서 최소 개수의 노드를 가지면서 왼쪽이나 오른쪽 서브 트리만 가지고 있는 트리



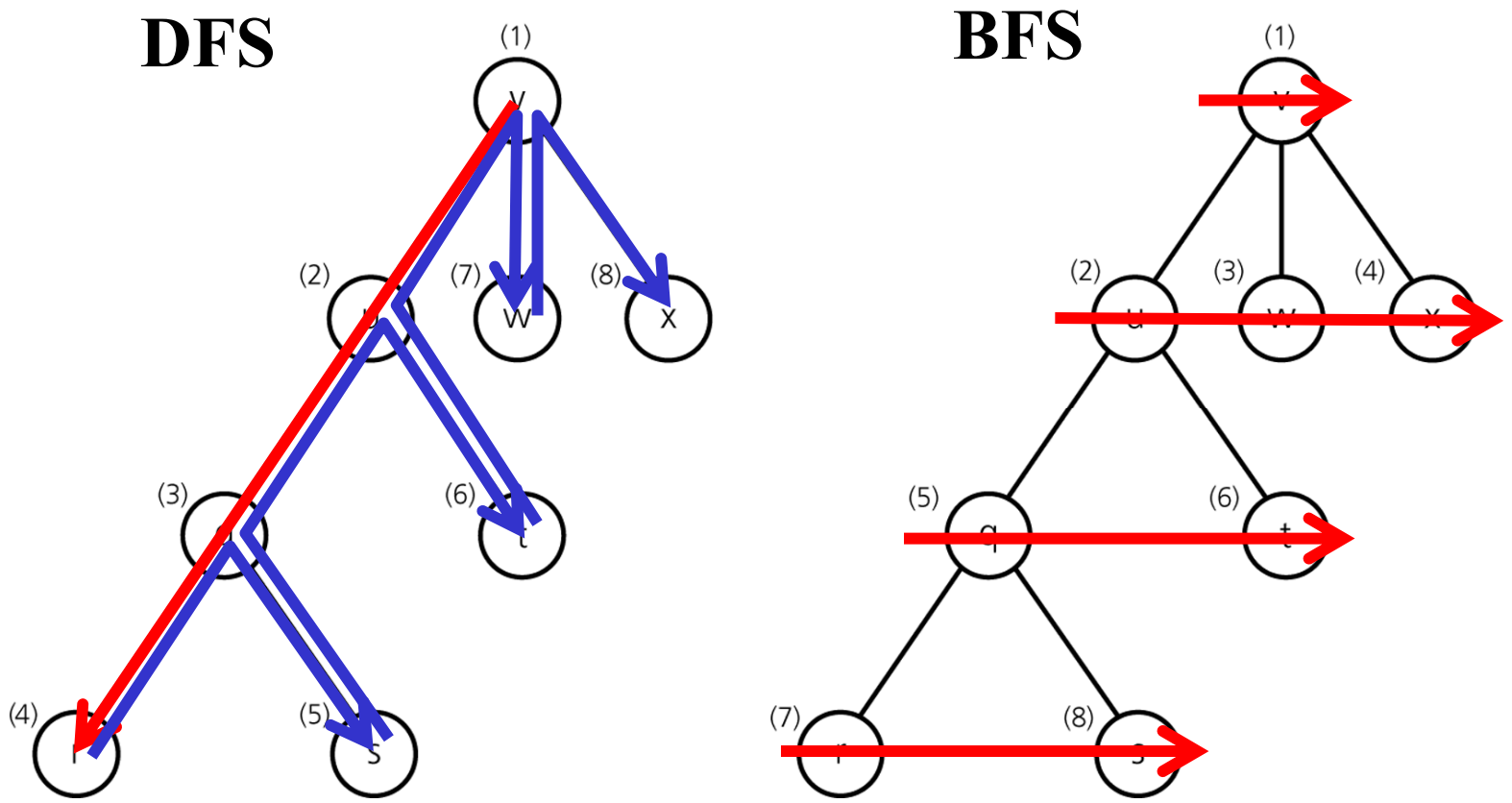


이진 트리

이진 트리 순회

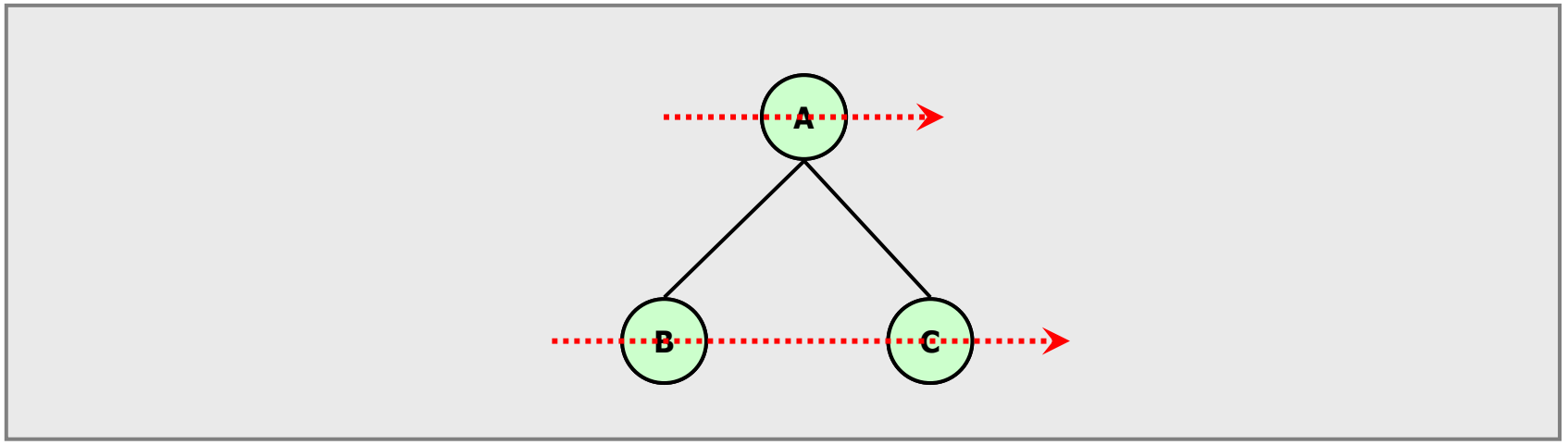
이진 트리 순회 (1/5)

- (이진) **트리 순회**(Traversal)
  - 깊이 우선 순회(DFS)와 너비 우선 순회(BFS)



이진 트리 순회 (2/5)

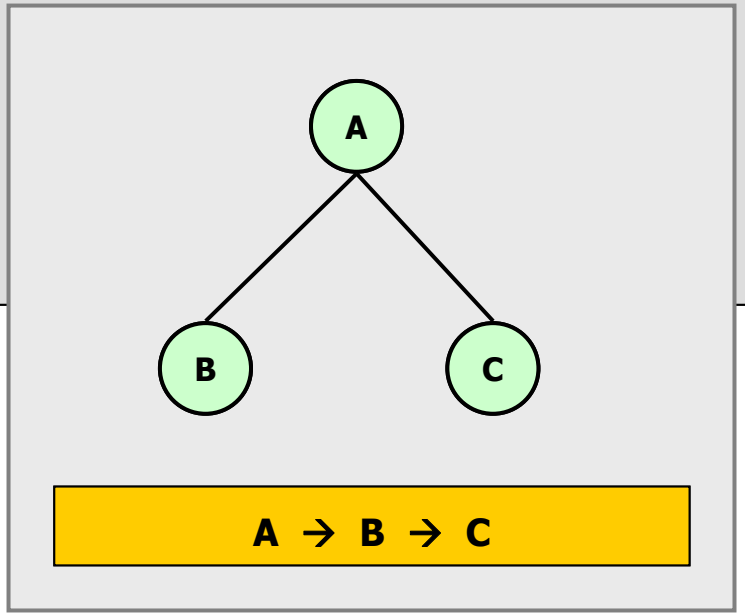
- **이진 트리 순회:** DFS, BFS
  - 깊이 우선 순회: 스택을 이용하여 구현
    - 전위 순회(preorder traversal)
    - 중위 순회(inorder traversal)
    - 후위 순회(postorder traversal)
  - 너비 우선 순회: 큐를 이용하여 구현
    - 다음 레벨의 노드들을 처리하기 전에 노드의 자식 모두를 처리



이진 트리 순회 (3/5)

- **깊이 우선 순회:** 전위 순회
  - 전위 순회(Preorder Traversal)

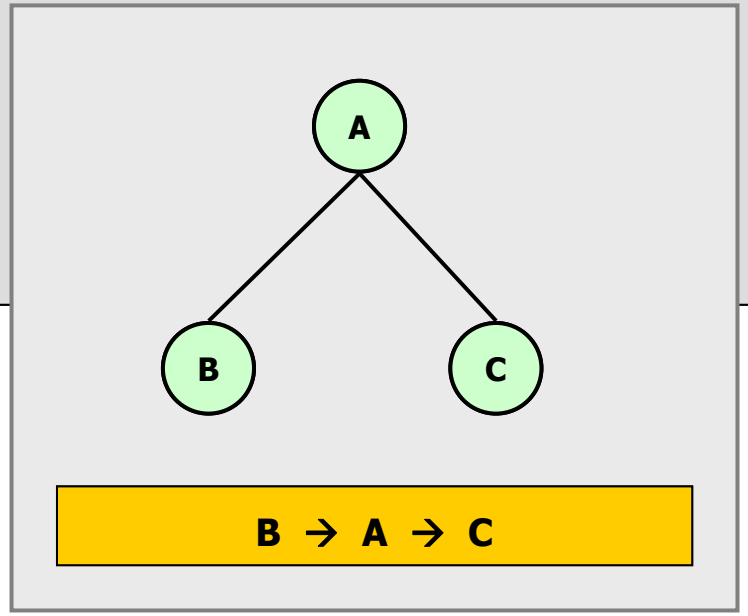
```
preorder(T)
    if(T ≠ NULL) then
    {
        visit T.data;
        preorder(T.Llink);
        preorder(T.Rlink);
    }
end preorder()
```



이진 트리 순회 (4/5)

- **깊이 우선 순회:** 중위 순회
  - 중위 순회(Inorder Traversal)

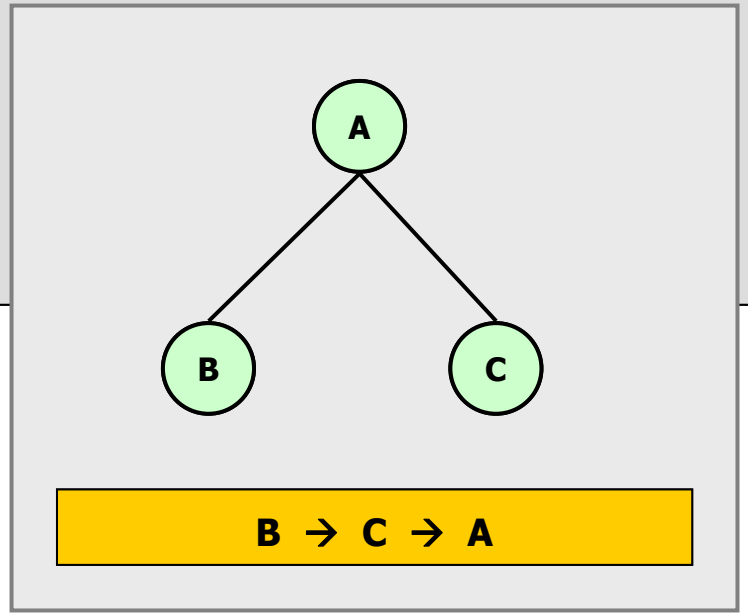
```
inorder(T)
    if(T ≠ NULL) then
    {
        inorder(T.Llink)
        visit T.data;
        inorder(T.Rlink);
    }
end inorder()
```



이진 트리 순회 (5/5)

- **깊이 우선 순회: 후위 순회**
  - 후위 순회(Postorder Traversal)

```
postorder(T)
    if(T ≠ NULL) then
    {
          postorder(T.Llink)
          postorder(T.Rlink);
          visit T.data;
    }
end postorder()
```

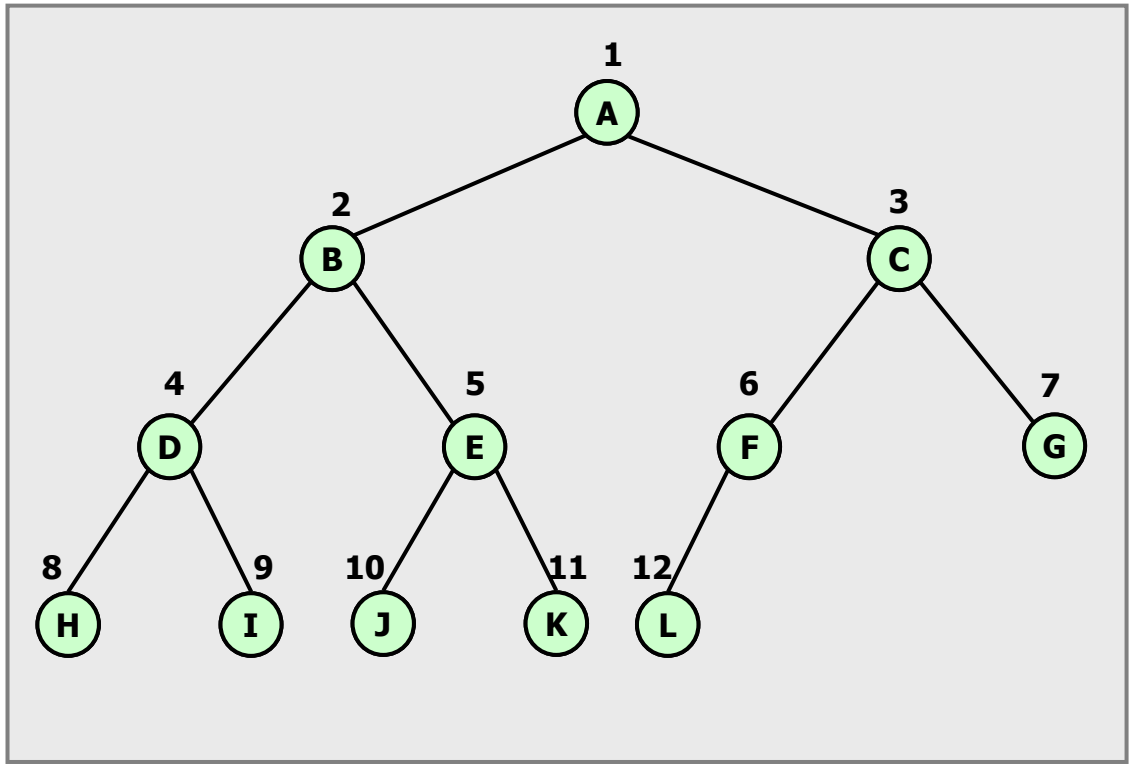


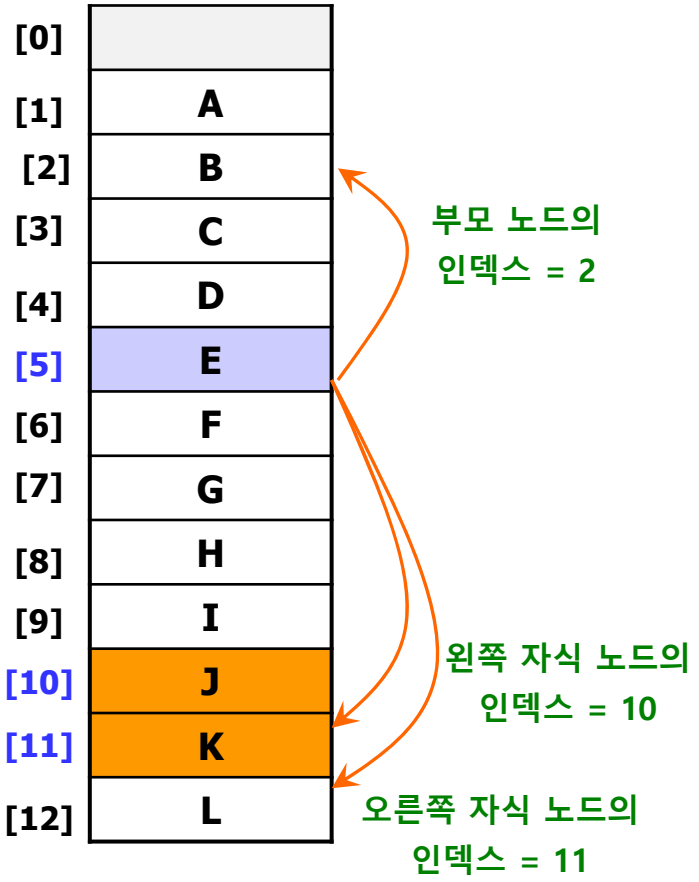
이진 트리

이진 트리 구현: 순차 자료 구조

이진 트리 구현: 순차 자료 구조 (1/2)

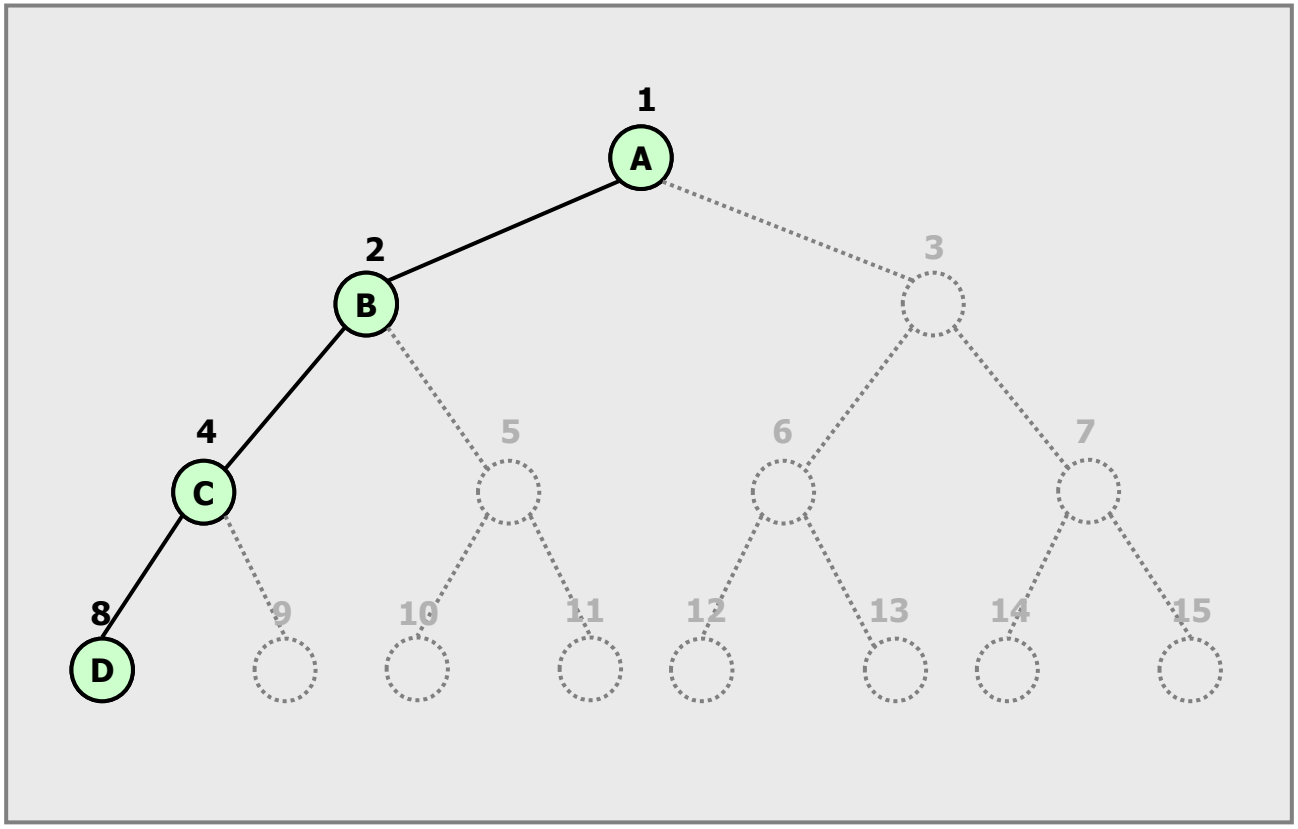
- **이진 트리 구현:** 순차 자료구조
  - 완전 이진 트리의 배열 표현





이진 트리 구현: 순차 자료 구조 (2/2)

- **이진 트리 구현:** 순차 자료구조
  - 편향 이진 트리의 배열 표현



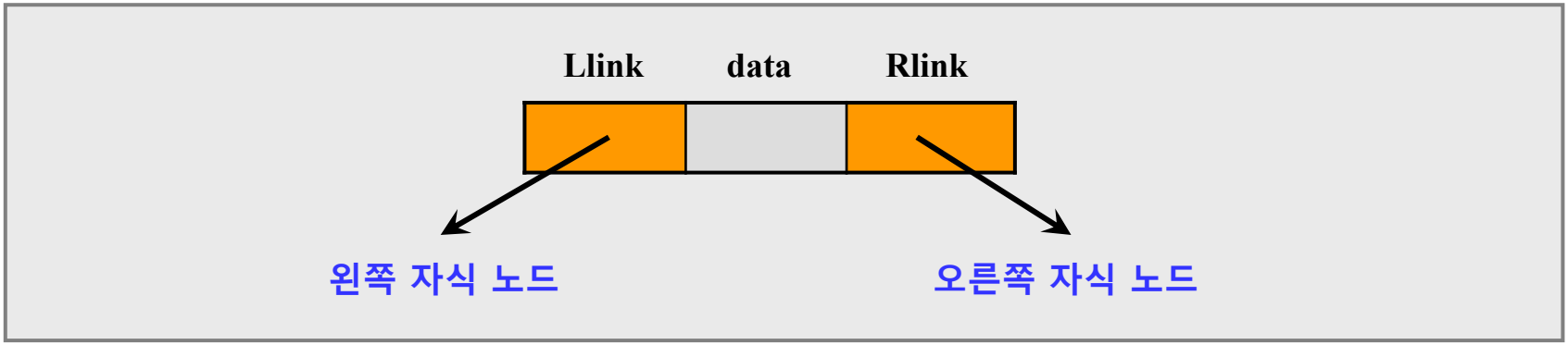
[0]	
[1]	A
[2]	B
[3]	
[4]	C
[5]	
[6]	
[7]	
[8]	D
[9]	
[10]	
[11]	
[12]	

이진 트리

이진 트리 구현: 연결 자료구조

이진 트리 구현: 연결 자료 구조 (1/3)

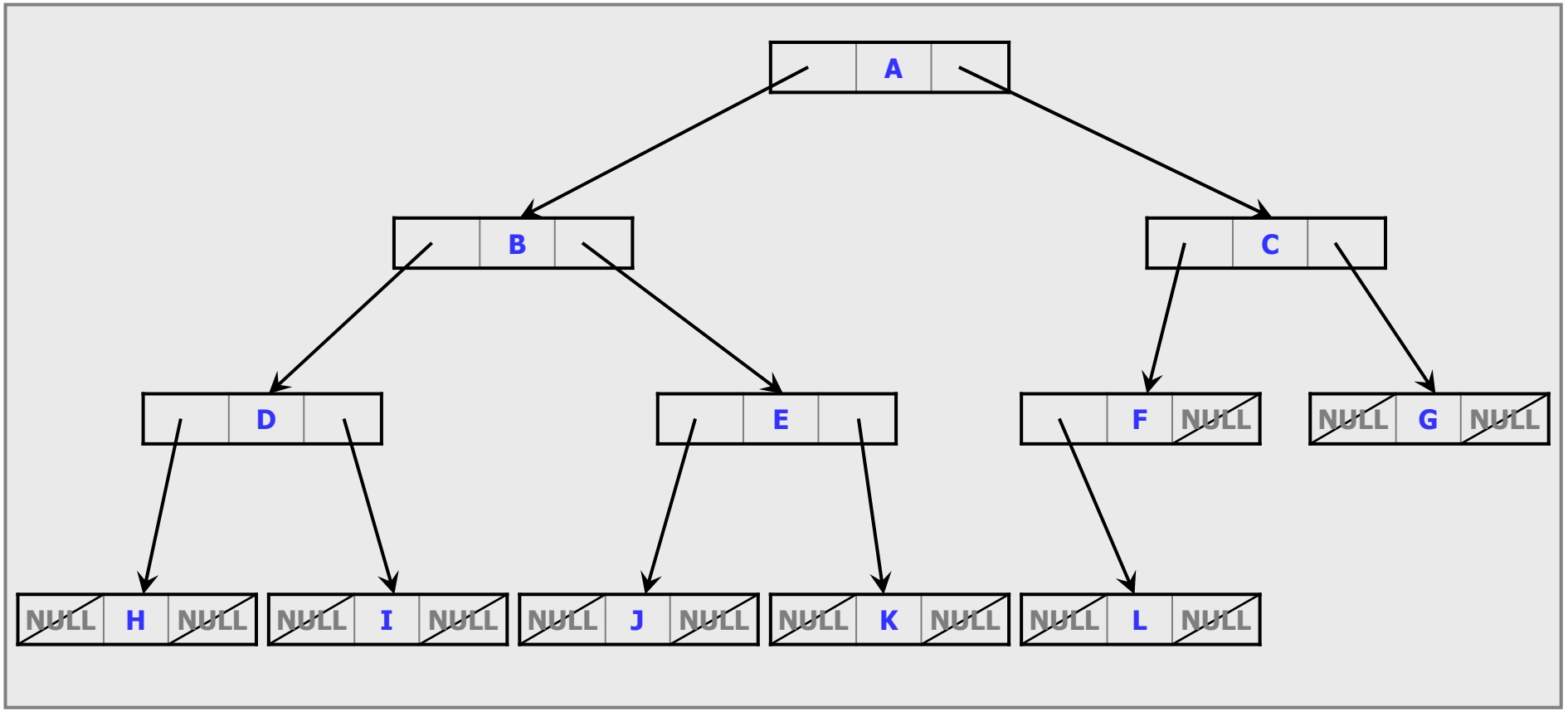
- **이진 트리 구현:** 연결 자료 구조




```
typedef struct _BTreeNode
{
    int                     data;
    struct _BTreeNode       *Llink;
    struct _BTreeNode       *Rlink;
}BTreeNode;
```

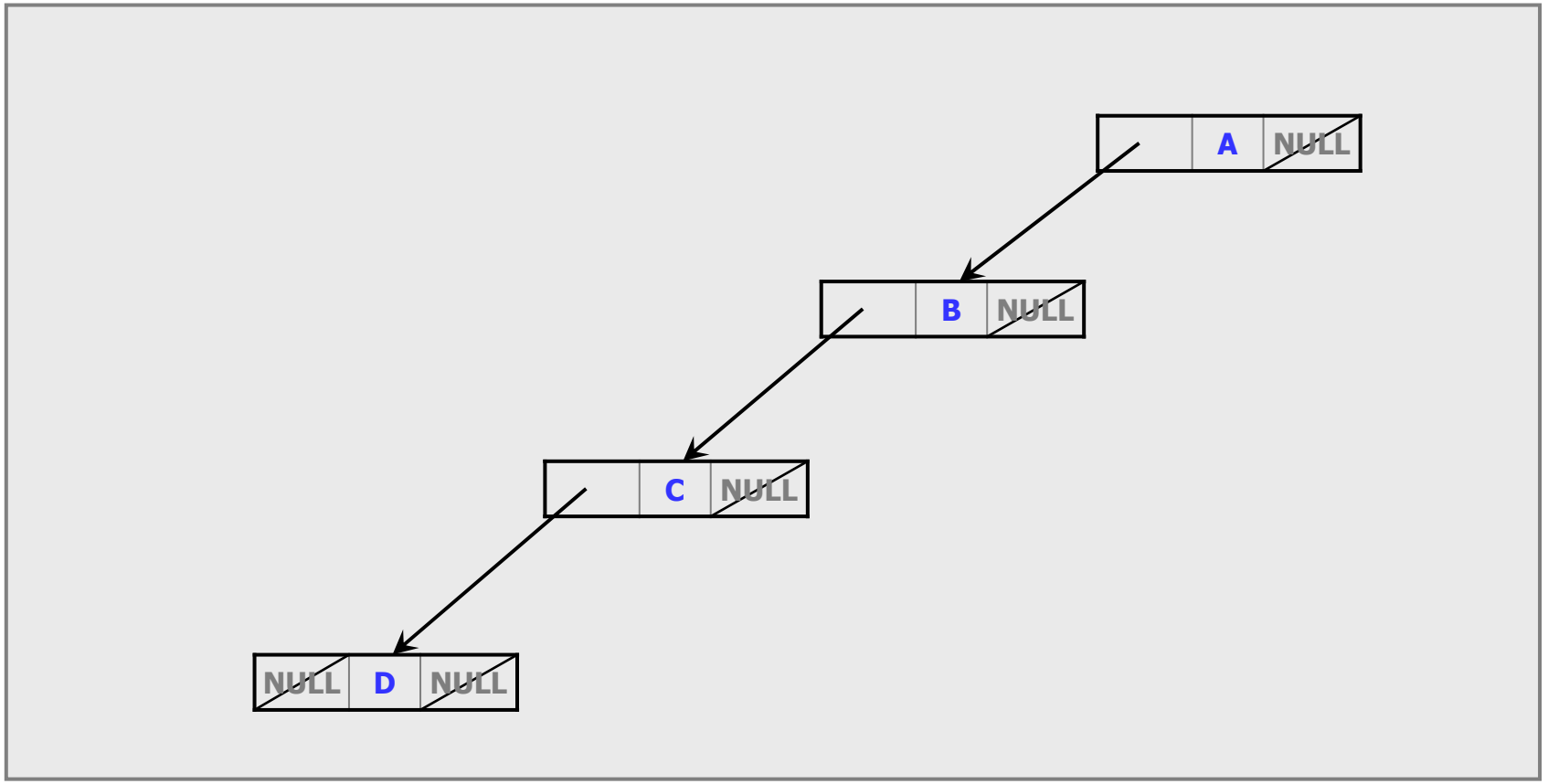
이진 트리 구현: 연결 자료 구조 (2/3)

- **이진 트리 구현:** 연결 자료 구조
  - 완전 이진 트리의 연결 자료 구조 형태



이진 트리 구현: 연결 자료 구조 (3/3)

- **이진 트리 구현:** 연결 자료 구조
  - 편향 이진 트리의 연결 자료 구조 형태



이진 트리

이진 트리 구현: 연결 자료 구조

– C/C++, Python

이진 트리 구현: 연결 자료 구조 (6/6)

- **이진 트리 구현:** 연결 자료구조

```
# 클래스 설계: LinkedBTree
class LinkedBTree:
    class DNode:
        def __init__(self, data, Llink=None, Rlink=None):
            self.data = data
            self.Llink = Llink
            self.Rlink = Rlink

    def __init__(self):
        self.__root = None

    # 깊이 우선 순회: 전위.중위.후위 순회
    def Preorder(self) -> None:
    def Inorder(self) -> None:
    def Postorder(self) -> None:

    # 너비 우선 순회
    def Levelorder(self) -> None:
    def __del__(self):

    # 이진 트리(수식 트리) 생성: 스택 구조 활용
    def makeLinkedBTree(self, postfix) -> DNode:
```

이진 트리 구현: 연결 자료 구조 (5/6)

- 이진 트리 구현: 연결 자료

```
#include "LinkedNode(template).cpp"    // DNode

// #pragma once
#ifndef __LinkedBTree_Template_H__
#define __LinkedBTree_Template_H__

// 클래스 설계: LinkedBTree<T>
template <typename T>
class LinkedBTree {
public:
        LinkedBTree();
        ~LinkedBTree();
        void            Preorder(DNode<T> *root) const;
        void            Inorder(DNode<T> *root) const;
        void            Postorder(DNode<T> *root) const;
        void            Levelorder(DNode<T> *root) const;
        DNode<T>        *makeLinkedBTree(const string str);
        int             isOperator(int op);
private:
        DNode<T>        *root_;
};
```

```
// LinkedNode(template).cpp
// #pragma once
#ifndef __DNode_Template_H__
#define __DNode_Template_H__

template <typename T>    class LinkedBTree;

// 클래스 설계: DNode(data, Llink, Rlink)
template <typename T>
class DNode {
public:
        DNode(const T &data);
private:
        T               data_;
        DNode<T>        *Llink_;
        DNode<T>        *Rlink_;
        template <typename T> friend class LinkedBTree;
};

#endif
```

이진 트리 구현: 연결 자료 구조 (4/6)

- **이진 트리 구현:** 연결 자료구조

```
// #pragma once
#include "LinkedNode.h"              // DNode, makeDNode

// 이진 트리 생성: 후위 표기법으로...
DNode    *makeLinkedBTree(char *postfix);
int      isOperator(int ch);
int      isLegal(char *pStr);

// 이진 트리 순회: 깊이 우선 순회(전위.중위.후위 순회)
void     Preorder(DNode *root);
void     Inorder(DNode *root);
void     Postorder(DNode *root);

// 이진 트리 순회: 너비 우선 순회
void     Levelorder(DNode *root);
```

이진 트리 구현

연결 자료구조: Python

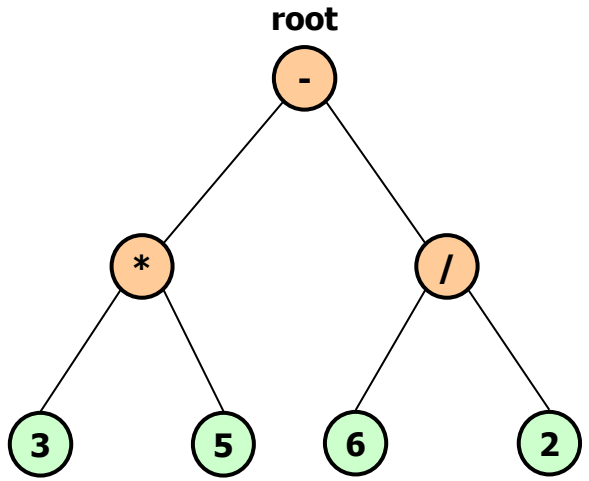
이진 트리 구현(Python): 연결 자료 구조 (1/4)

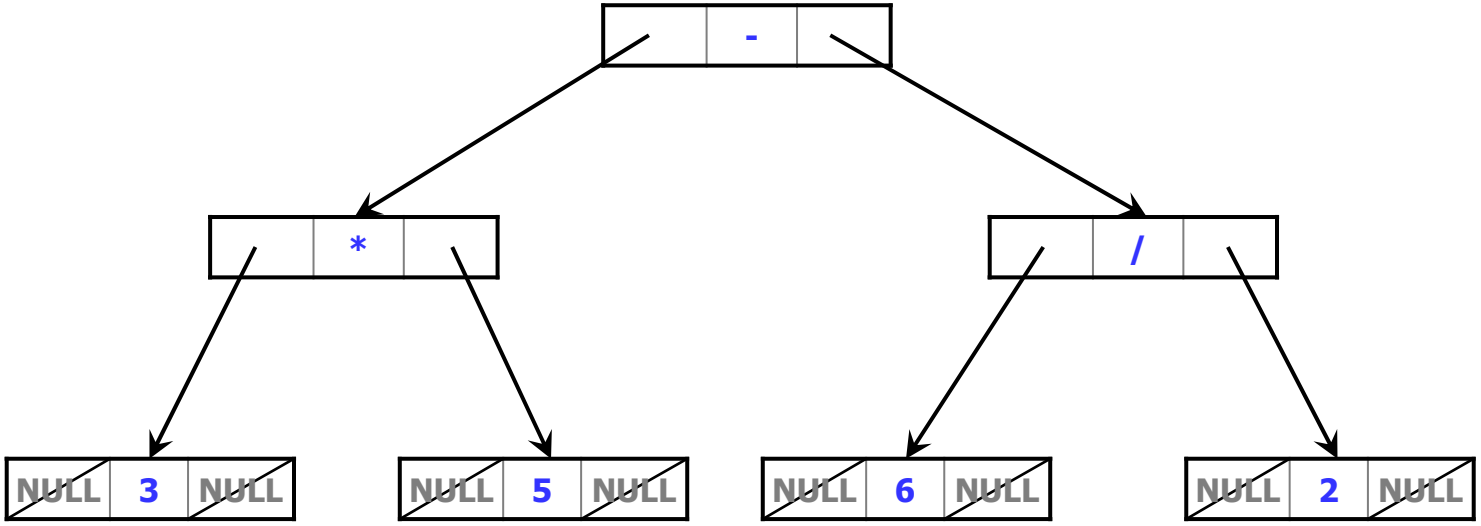
- **이진 트리 구현:** 연결 자료구조

```
# 클래스 설계: LinkedBTree
class LinkedBTree:
    class DNode:
        def __init__(self, data, Llink=None, Rlink=None):
            self.data = data
            self.Llink = Llink
            self.Rlink = Rlink

    def __init__(self):
        self.__root = None

    # 깊이 우선 순회: 전위.중위.후위 순회
    def Preorder(self) -> None:
    def Inorder(self) -> None:
    def Postorder(self) -> None:

    # 너비 우선 순회
    def Levelorder(self) -> None:
    def __del__(self):

    # 이진 트리(수식 트리) 생성: 스택 구조 활용
    def makeLinkedBTree(self, postfix) -> DNode:
```

이진 트리 구현(Python): 연결 자료 구조 (2/4)

- **이진 트리 구현:** 연결 자료구조
  - 프로그램 실행 결과는 다음과 같다.

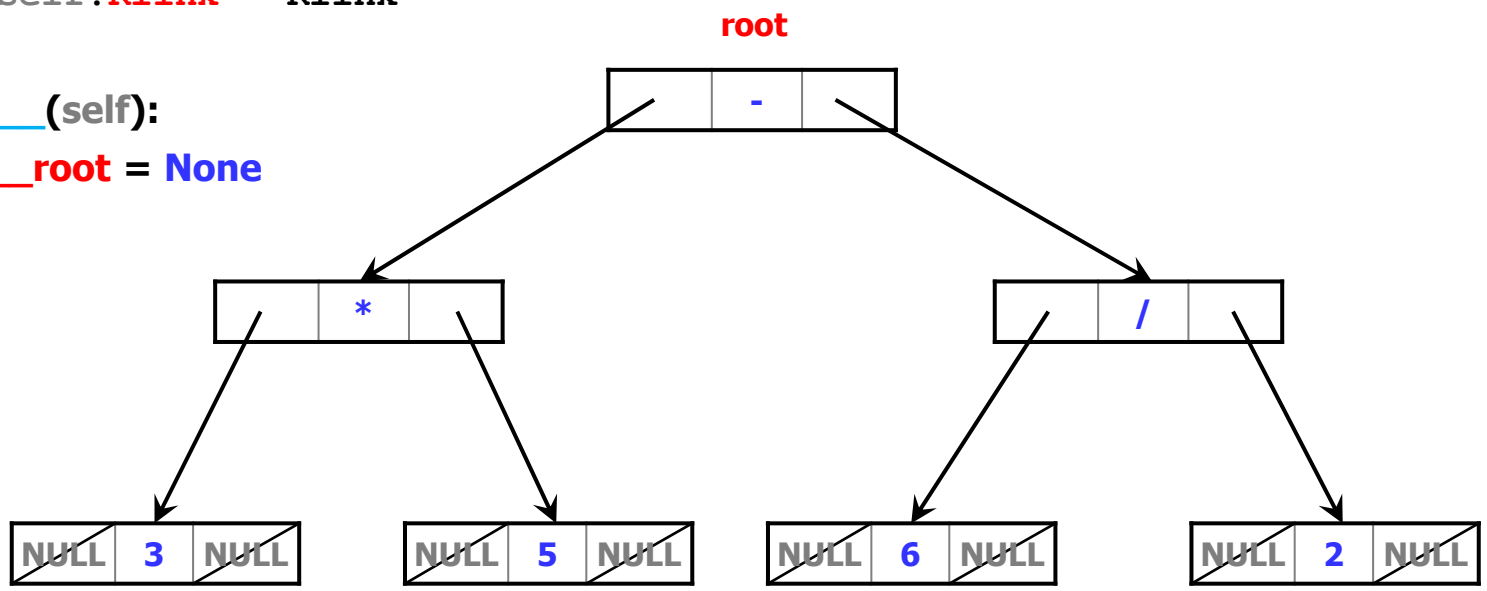




이진 트리 구현(Python): 연결 자료 구조 (3/4)

예제 7-1: 이진 트리 구현 -- 연결 자료 구조 LinkedBTree.py (1/2)

```
# 클래스 설계: LinkedBTree
class LinkedBTree:
    class DNode:
        def __init__(self, data, Llink=None, Rlink=None):
            self.data = data
            self.Llink = Llink
            self.Rlink = Rlink

    def __init__(self):
        self.__root = None
```



이진 트리 구현(Python): 연결 자료 구조 (4/4)

예제 7-1: 이진 트리 구현 -- 연결 자료 구조 LinkedBTree.py (2/2)

```
    # 깊이 우선 순회: 전위,중위,후위 순회
    def Preorder(self) -> None:
    def Inorder(self) -> None:
    def Postorder(self) -> None:

    # 너비 우선 순회
    def Levelorder(self) -> None
    def __del__(self):

    # 이진 트리(수식 트리) 생성: 스택 구조 활용
    def makeLinkedBTree(self, postfix) -> None:

if __name__ == '__main__':
    postfix = input('트리를 구성할 후위 수식: ')
    BTree = LinkedBTree()
    BTree.makeLinkedBTree(postfix)

    # 깊이 우선 순회: 전위,중위,후위 순회
    print('Preorder  : ', end=' '); BTree.Preorder(); print('')
    print('Inorder   : ', end=' '); BTree.Inorder(); print('')
    print('Postorder : ', end=' '); BTree.Postorder(); print('')

    # 너비 우선 순회
    print('Levelorder: ', end=' '); BTree.Levelorder(); print('')
```

이진 트리 구현

연결 자료구조: C++

이진 트리 구현(C++): 연결 자료 구조 (1/5)

- 이진 트리 구현: 연결 자료

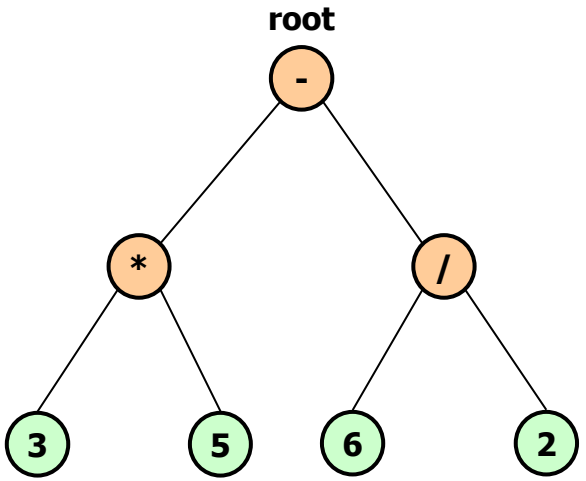
```
#include "LinkedNode(template).cpp"    // DNode

// #pragma once
#ifndef __LinkedBTree_Template_H__
#define __LinkedBTree_Template_H__

// 클래스 설계: LinkedBTree<T>
template <typename T>
class LinkedBTree {
public:
        LinkedBTree();
        ~LinkedBTree();
        void            Preorder(DNode<T> *root) const;
        void            Inorder(DNode<T> *root) const;
        void            Postorder(DNode<T> *root) const;
        void            Levelorder(DNode<T> *root) const;
        DNode<T>        *makeLinkedBTree(const string str);
        int             isOperator(int op);
private:
        DNode<T>        *root_;
};
```

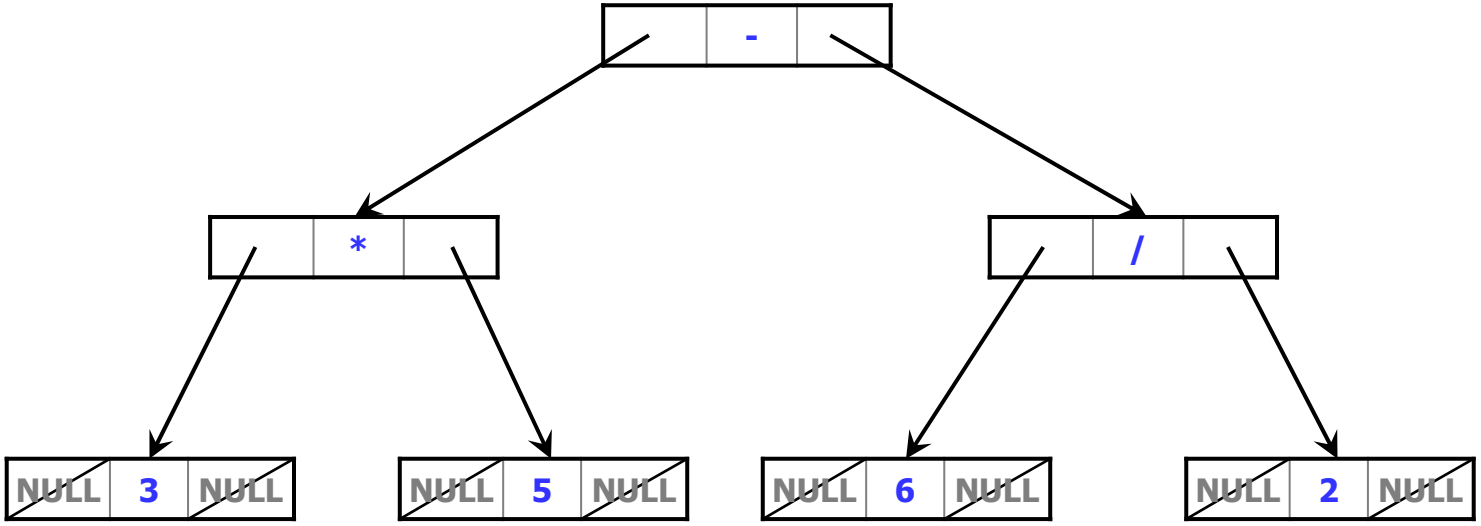
```
// LinkedNode(template).cpp
// #pragma once
#ifndef __DNode_Template_H__
#define __DNode_Template_H__

template <typename T>    class LinkedBTree;

// 클래스 설계: DNode(data, Llink, Rlink)
template <typename T>
class DNode {
public:
        DNode(const T &data);
private:
        T               data_;
        DNode<T>        *Llink_;
        DNode<T>        *Rlink_;
        template <typename T> friend class LinkedBTree;
};

#endif
```

이진 트리 구현(C++): 연결 자료 구조 (2/5)

- **이진 트리 구현:** 연결 자료구조
  - 프로그램 실행 결과는 다음과 같다.

```
Microsoft Visual Studio 디버그
트리를 구성할 후위 수식: 3 5 * 6 2 / -

Preorder  :   - * 3 5 / 6 2
Inorder   :   3 * 5 - 6 / 2
Postorder :   3 5 * 6 2 / -
Levelorder :   - * / 3 5 6 2
```





이진 트리 구현(C++): 연결 자료 구조 (3/5)

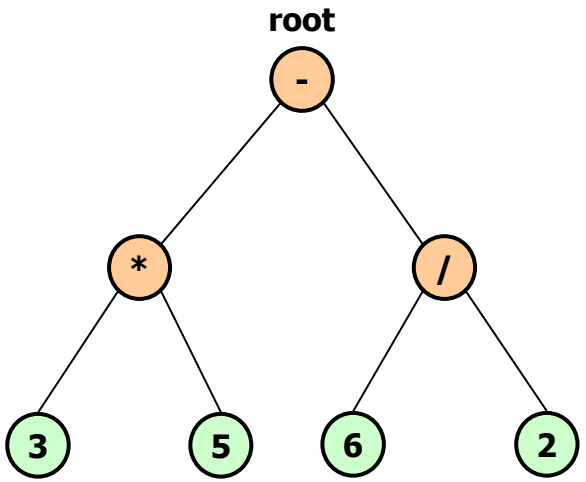
예제 7-1: 이진 트리 구현 -- 연결 자료 구조 LinkedBTree(demo).cpp

```
#include <iostream>
#include <string>
#include "LinkedBTree.cpp"          // LinkedBTree<T>, DNode<T>
using namespace std;

int main(void)
{
        string postfix;

        cout << "트리를 구성할 후위 수식: ";
        getline(cin, postfix);

        // 이진 트리 생성
        LinkedBTree<char>     BTree;
        DNode<char>           *root = BTree.makeLinkedBTree(postfix);

        // 깊이 우선 순회: 전위.중위.후위 순회
        cout << "\nPreorder  : ";    BTree.Preorder(root);          // 전위 순회
        cout << "\nInorder   : ";    BTree.Inorder(root);           // 중위 순회
        cout << "\nPostorder : ";    BTree.Postorder(root);   // 후위 순회

        // 너비 우선 순회
        cout << "\nLevelorder : ";   BTree.Levelorder(root);

        return 0;
}
```

이진 트리 구현(C++): 연결 자료 구조 (4/5)

예제 7-1: 이진 트리 구현 -- 연결 자료 구조 LinkedBTree.cpp (1/2)

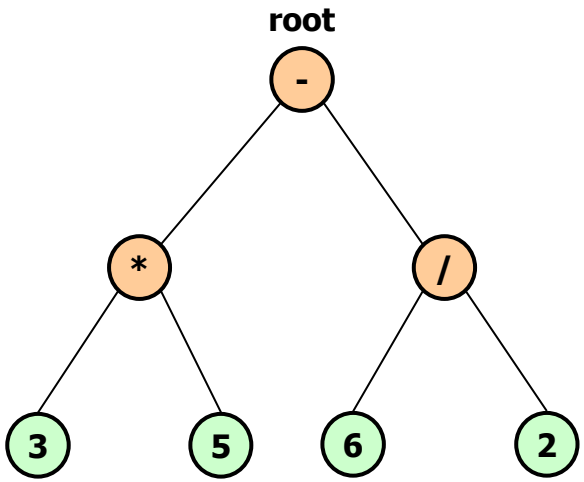
```
#include <iostream>
#include <string>
#include <stack>
#include <queue>
#include "LinkedNode(template).cpp" // SNode<T>
using namespace std;

// #pragma once
#ifndef __LinkedBTree_Template_H__
#define __LinkedBTree_Template_H__

// 클래스 설계: LinkedBTree<T>
template <typename T>
class LinkedBTree {
public:
        LinkedBTree();
        ~LinkedBTree();
        void            Preorder(DNode<T> *root) const;
        void            Inorder(DNode<T> *root) const;
        void            Postorder(DNode<T> *root) const;
        void            Levelorder(DNode<T> *root) const;
        DNode<T>        *makeLinkedBTree(const string str);
        int             isOperator(int op);
private:
        DNode<T>        *root_;
};
```



이진 트리 구현(C++): 연결 자료 구조 (5/5)

예제 7-1: 이진 트리 구현 -- 연결 자료 구조 LinkedBTree.cpp (2/2)

```
// 이진 트리 생성
template <typename T>
LinkedBTree<T>::LinkedBTree()
        : root_(nullptr) { }
```




```
// 연산자 여부 판단
template <typename T>
int  LinkedBTree<T>::isOperator(int op) {
        return op == '+' || op == '-' || op == '*' || op == '/';
}

#endif
```

이진 트리 구현

연결 자료구조: C

THE C PROGRAMMING LANGUAGE

이진 트리 구현(C): 연결 자료 구조 (1/5)

- **이진 트리 구현:** 연결 자료구조

```
// #pragma once
#include "LinkedNode.h"              // DNode, makeDNode

// 이진 트리 생성
DNode    *makeLinkedBTree(char *postfix);
int      isOperator(int ch);
int      isLegal(char *pStr);

// 이진 트리 순회: 깊이 우선 순회(전위.중위.후위 순회)
void     Preorder(DNode *root);
void     Inorder(DNode *root);
void     Postorder(DNode *root);

// 이진 트리 순회: 너비 우선 순회
void     Levelorder(DNode *root);
```

이진 트리 구현(C): 연결 자료 구조 (2/5)

예제 7-1: 이진 트리 구현(C) -- 연결 자료 구조 LinkedBTree(demo).c

```c
#include <stdio.h>
#include <stdlib.h>             // exit
#include "LinkedBtree.h"        // DNode
#define bufferMAXSIZE 1024
int main(void)
{
        DNode   *root;
        char    postfix[bufferMAXSIZE];

        printf("트리를 구성할 후위 수식: ");
        gets_s(postfix, sizeof(postfix));
        if(!isLegal(postfix)) {
                printf("\n잘못된 수식... \n\n");
                exit(100);
        }

        // 이진 트리 구성
        root = makeLinkedBTree(postfix);

        // 깊이 우선 순회(이진 트리): 전위.중위.후위 순회
        printf("\nPreorder   : ");    Preorder(root);        // 전위 순회
        printf("\nInorder    : ");    Inorder(root);         // 중위 순회
        printf("\nPostorder  : ");    Postorder(root);       // 후위 순회

        // 너비 우선 순회(이진 트리)
        printf("\nLevelorder : ");    Levelorder(root);
        printf("\n");
        return 0;
}
```

```
Microsoft Visual Studio 디버그
트리를 구성할 후위 수식: 3 5 * 6 2 / -

Preorder  :   - * 3 5 / 6 2
Inorder   :   3 * 5 - 6 / 2
Postorder :   3 5 * 6 2 / -
Levelorder :   - * / 3 5 6 2
```

이진 트리 구현(C): 연결 자료 구조 (3/5)

예제 7-1: 이진 트리 구현(C) -- 연결 자료 구조 LinkedBTree.c (1/3)

```
#include <stdio.h>
#include <stdlib.h>                    // malloc
#include "LinkedStack.h"               // LinkedStack
#include "LinkedQueue.h"               // LinkedQueue
#include "LinkedBTree.h"               // DNode
// #include "LinkedNode.h"             // SNode, DNode
#include "Operators.h"                 // isOperator,precedence, isLegal

// 이진 트리 생성
DNode  *makeLinkedBTree(char *pStr) {
        DNode           *temp;
        LinkedStack     *Stack = stackCreate();
        while(*pStr) {
                while(*pStr == ' ')     // 공백 제거
                        ++pStr;
                temp = makeDNode(*pStr);
                // 연산자일 경우: 스택에서 자식 노드를 구성할 주소를 pop
                if(isOperator(*pStr)) {
                        temp->Rlink = (DNode*)top(Stack);   pop(Stack);
                        temp->Llink = (DNode*)top(Stack);   pop(Stack);
                }
                push(Stack, (long long)temp);
                ++pStr;
        }
        temp = (DNode*)top(Stack);    // 루트 노드
        pop(Stack);

        stackDestroy(Stack);
        return  temp;
}
```

이진 트리 구현(C): 연결 자료 구조 (4/5)

예제 7-1: 이진 트리 구현(C) -- 연결 자료 구조 LinkedBTree.c (2/3)

```
// 깊이 우선 순회: 전위 순회(재귀적 용법)
void Preorder(DNode *root) {
        if(root) {
                printf("%3c", root->data);
                Preorder(root->Llink);
                Preorder(root->Rlink);
        }
}

// 깊이 우선 순회: 중위 순회(재귀적 용법)
void Inorder(DNode *root) {
        if(root) {
                Inorder(root->Llink);
                printf("%3c", root->data);
                Inorder(root->Rlink);
        }
}

// 깊이 우선 순회: 후위 순회(재귀적 용법)
void Postorder(DNode *root) {
        if(root) {
                Postorder(root->Llink);
                Postorder(root->Rlink);
                printf("%3c", root->data);
        }
}
```

이진 트리 구현(C): 연결 자료 구조 (5/5)

예제 7-1: 이진 트리 구현(C) -- 연결 자료 구조 LinkedBTree.c (3/3)

```
// 너비 우선 순회: 비재귀적 용법
void Levelorder(DNode* root) {
        DNode          *temp;
        LinkedQueue    *Queue = queueCreate();

        enQueue(Queue, (long long)root);
        while(!queueEempty(Queue))   {
                temp = (DNode*)front(Queue); deQueue(Queue);
                printf("%3c", temp->data);
                if(temp->Llink)        enQueue(Queue, temp->Llink);
                if(temp->Rlink)        enQueue(Queue, temp->Rlink);
        }
        queueDestroy(Queue);
}
```

우선 순위 큐와 힙

백문이불여일타(百聞而不如一打)

- 트리의 이해
- 이진 트리
- **우선 순위 큐와 힙**
  - 최소 힙
  - 힙 정렬

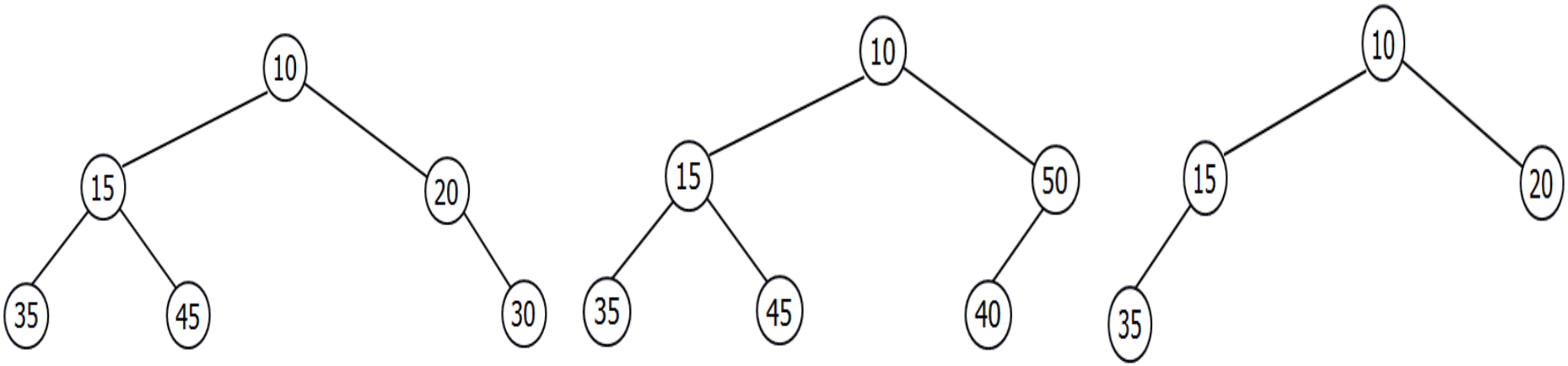
우선 순위 큐와 힙 (1/3)

- **힙(Heap)**
  - 우선 순위 큐를 구현하는 가장 기본적인 자료구조
    - 힙은 다음 두 조건을 만족해야 한다
      1. 완전 이진 트리
      2. 모든 노드는 값을 갖고, 자식 노드(들) 값보다 크거나 같다.
  - 우선 순위 큐(Priority Queue)
    - 가장 높은 우선순위를 가진 항목에 접근, 삭제와 임의의 우선순위를 가진 항목을 삽입을 지원하는 자료구조
    - 스택이나 큐도 일종의 우선 순위 큐
      - **스택:** 가장 마지막으로 삽입된 항목이 가장 높은 우선순위를 가진다.
        - » 따라서 최근 시간일수록 높은 우선순위를 부여한다.
      - **큐:** 먼저 삽입된 항목이 우선순위가 더 높다
        - » 따라서 이른 시간일수록 더 높은 우선순위를 부여한다.
      - 삽입되는 항목이 임의의 우선순위를 가지면 스택이나 큐는 새 항목이 삽입될 때마다 저장되어 있는 항목들을 우선순위에 따라 정렬해야 하는 문제점이 있음.

우선 순위 큐와 힙 (2/3)

- **힙:** 완전 이진 트리
  - 완전 이진 트리로서 부모의 우선 순위가 자식의 우선 순위보다 높은 자료구조

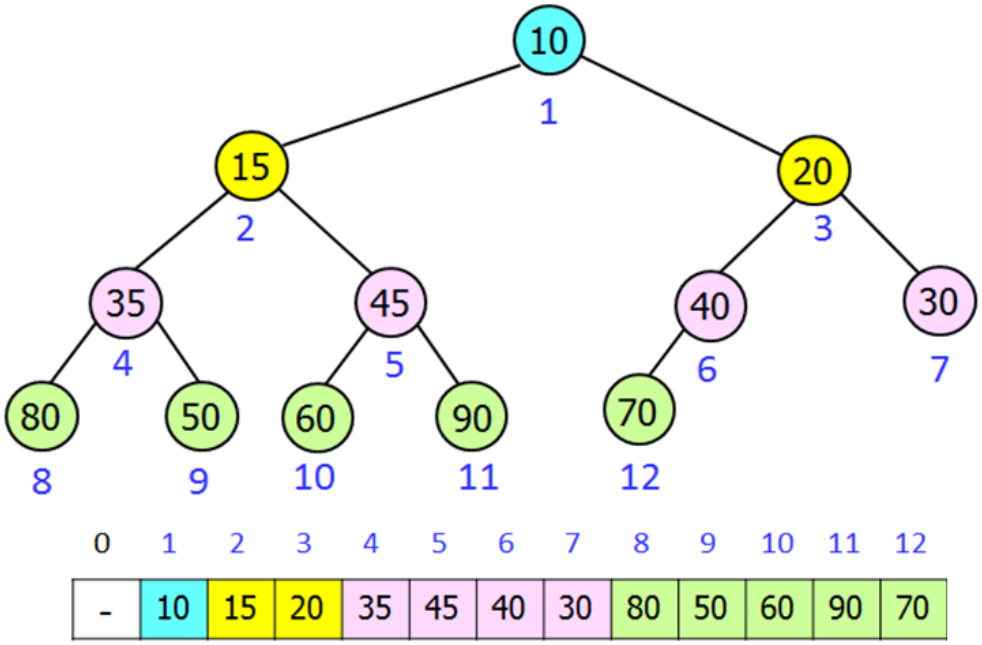
어느 트리가 이진 힙 일까?



(a) (b) (c)

우선 순위 큐와 힙 (3/3)

- **힙:** 순차 자료 구조
  - 완전 이진 트리의 노드들이 저장된 리스트



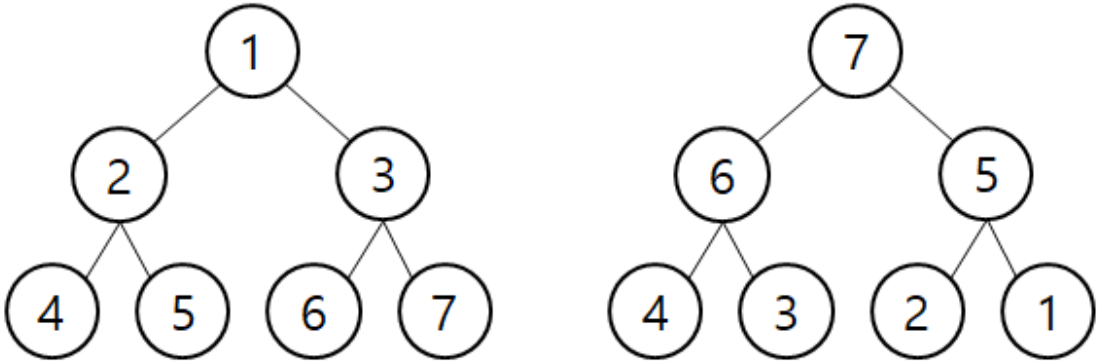
- a[i]의 자식은 a[2i]와 a[2i+1]에 있고,
- a[j]의 부모는 a[j//2]에 있다, j > 1.

우선 순위 큐와 힙

최소 힙

최소 힙 (1/11)

- **최소 힙(Minimum Heap)**
  - 키 값이 작을수록 높은 우선순위
    - 최소 힙의 루트에는 항상 가장 작은 키가 저장된다.
      - 부모에 저장된 키가 자식의 키보다 작다는 규칙
      - 루트는 a[1] 에 있으므로, O(1) 시간에 min 키를 가진 노드 접근
  - 최대 힙: 키 값이 클수록 더 높은 우선순위



최소힙 최대힙

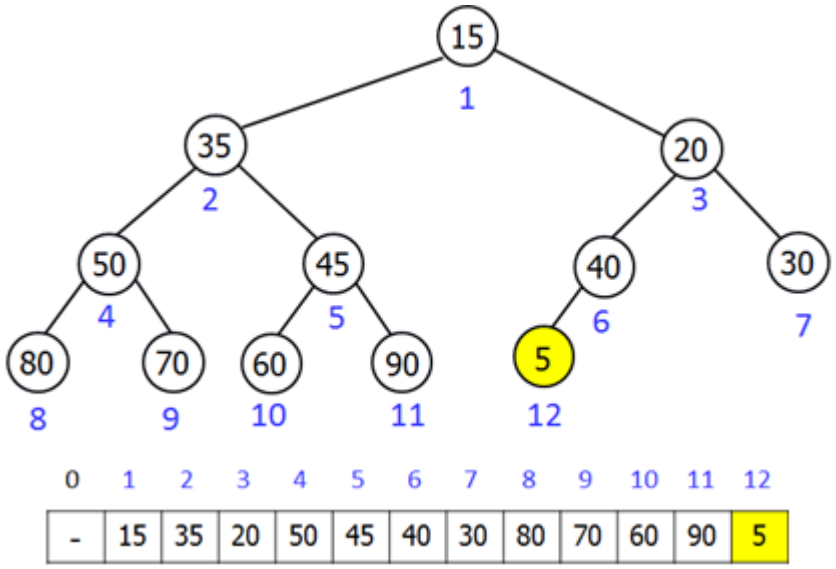
최소 힙 (2/11)

- **최소 힙: 삽입 연산**
  - **삽입 연산**
    1. **힙의 마지막 노드**(즉, 리스트의 마지막 항목)의 바로 다음에 비어 있는 공간에 새로운 항목을 저장한다.
    2. **루트 방향으로 올라가면서** 부모의 키와 비교하여 힙 속성이 만족될 때까지 노드를 교환한다.
  - **[2] 의 과정은 위로 올라가며 수행되므로 upheap 이라 부른다.**

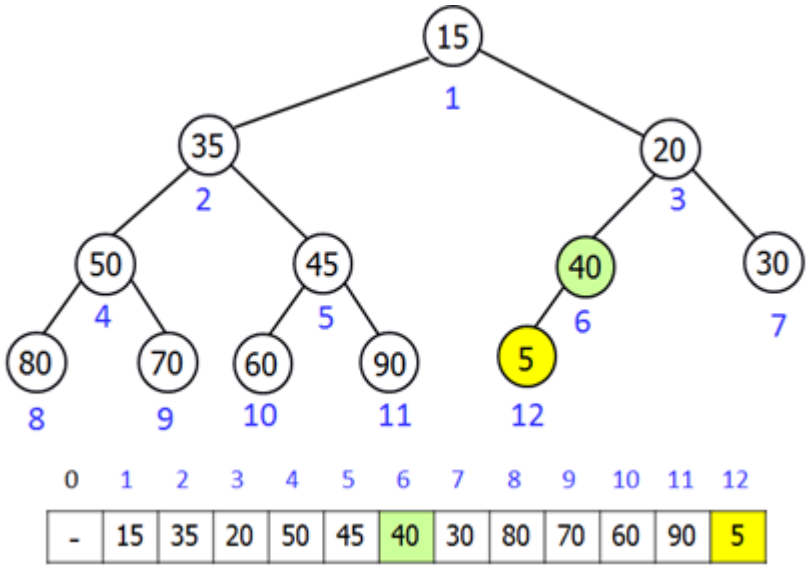
최소 힙 (3/11)

- **최소 힙:** 삽입 연산

최소힙에 5 를 삽입하는 과정



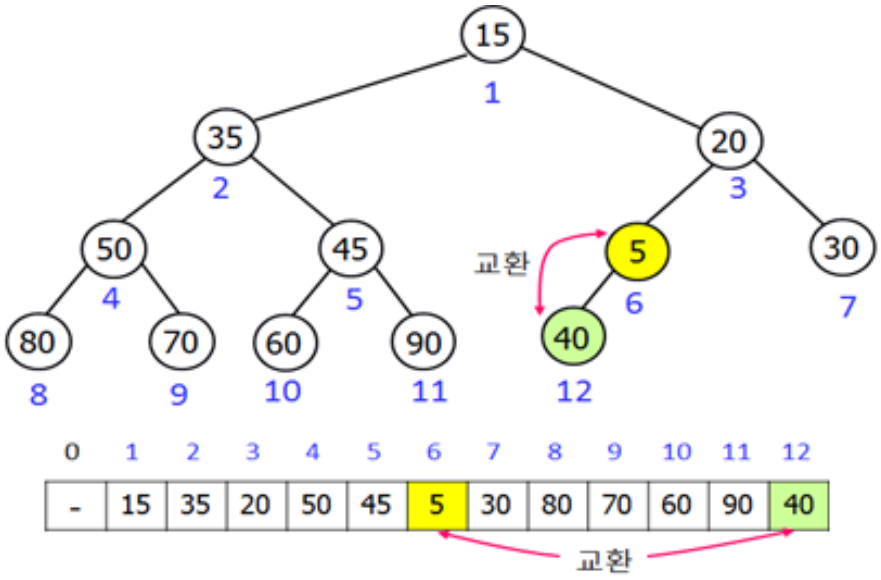
(a) 5를 마지막 항목(90) 다음에 저장



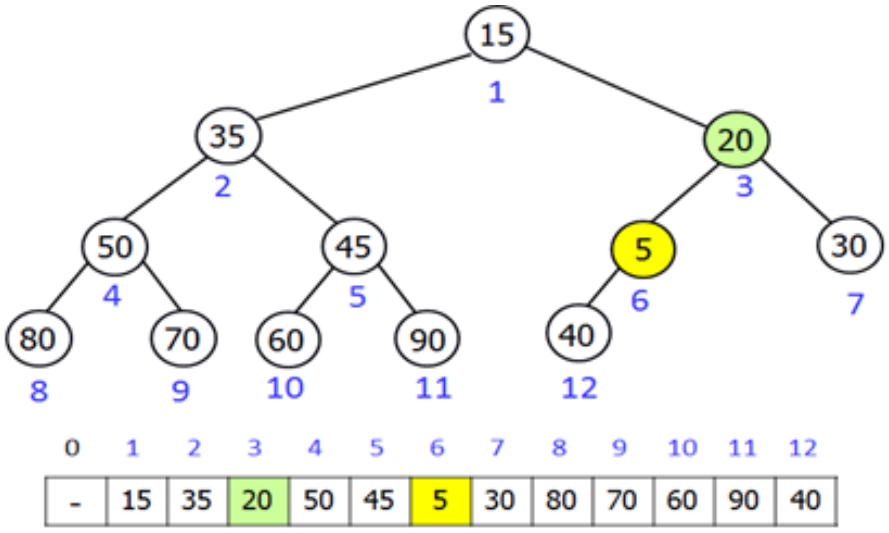
(b) a[12]의 5와 부모 a[6]의 40 비교

최소 힙 (4/11)

- **최소 힙:** 삽입 연산



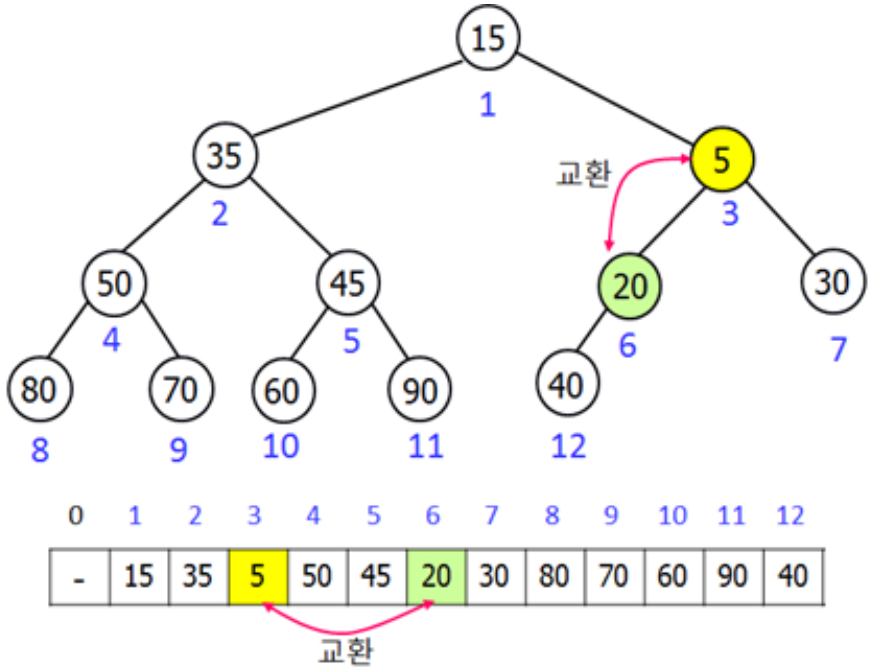
(c) 5와 40을 교환



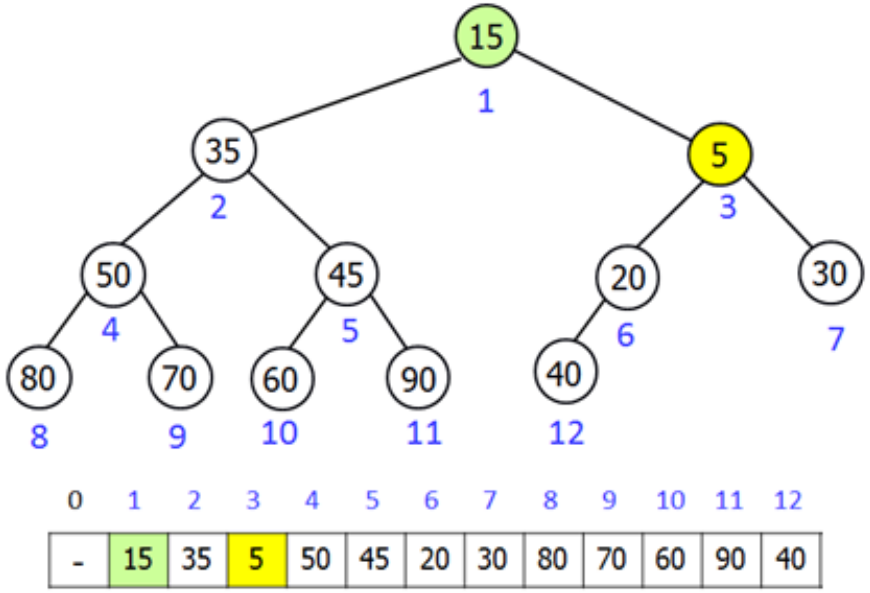
(d) a[6]의 5와 부모 a[3]의 20 비교

최소 힙 (5/11)

- **최소 힙:** 삽입 연산



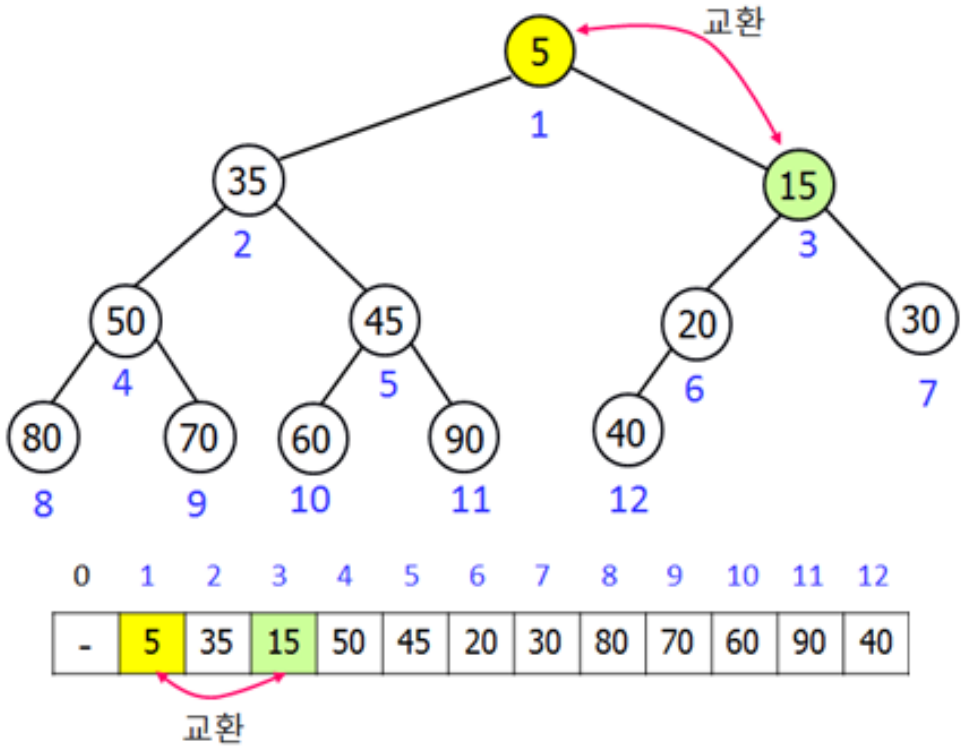
(e) 5와 20을 교환



(f) a[3]의 5와 부모 a[1]의 15 비교

최소 힙 (6/11)

- **최소 힙:** 삽입 연산



(g) 5와 15를 교환

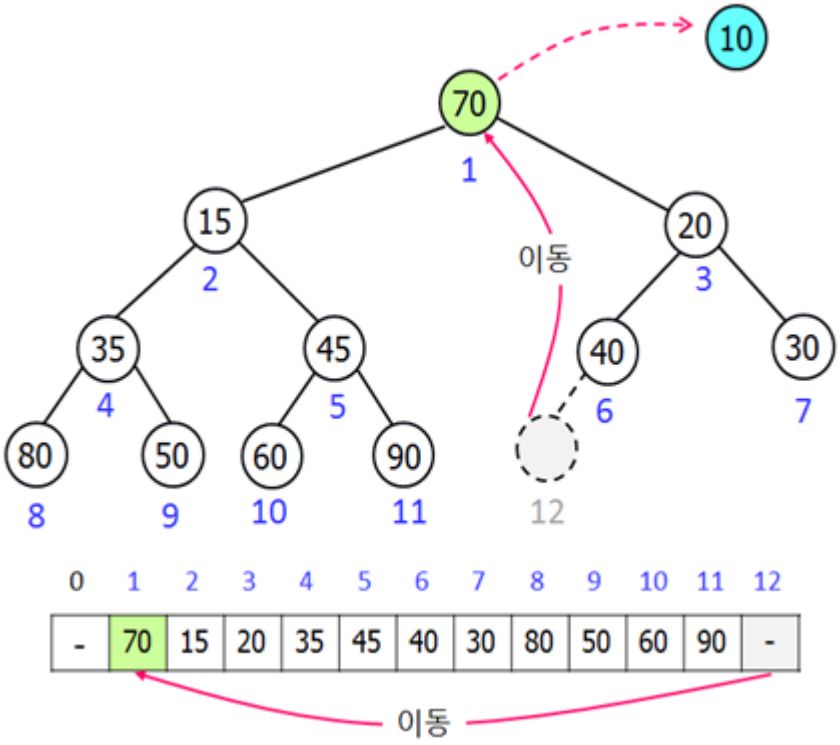
최소 힙 (7/11)

- **최소 힙: 삭제 연산**
  - **루트의 키를 삭제**
    1. **힙의 가장 마지막 노드,** 즉, 리스트의 가장 마지막 항목을 루트로 옮기고,
    2. **힙 크기를 1 감소** 시킨다.
    3. 루트로부터 자식들 중에서 작은 값을 가진 자식 (승자)과 키를 비교하여 힙 속성이 만족될 때까지 키를 교환하며 이파리 방향으로 진행한다.

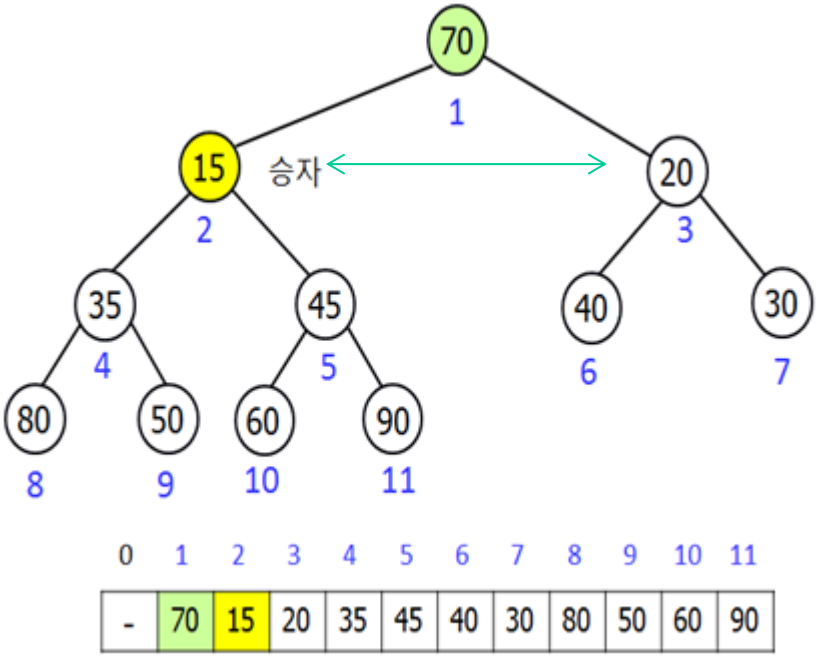
  - **[3]의 과정은 루트로부터 아래로 내려가며 진행되므로 downheap 이라 부른다.**

최소 힙 (8/11)

- **최소 힙:** 삭제 연산



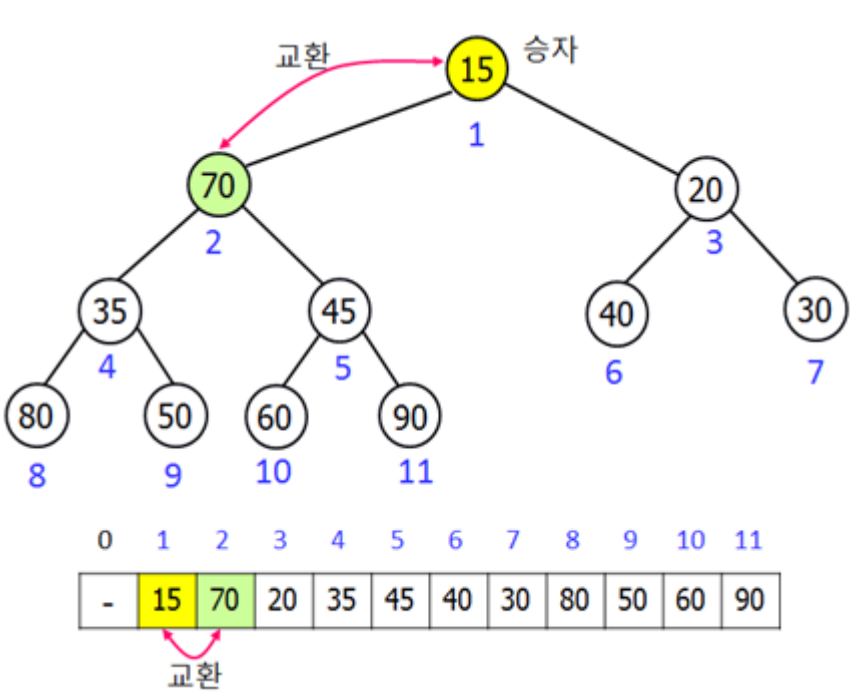
(a) 마지막 항목을 루트로 이동



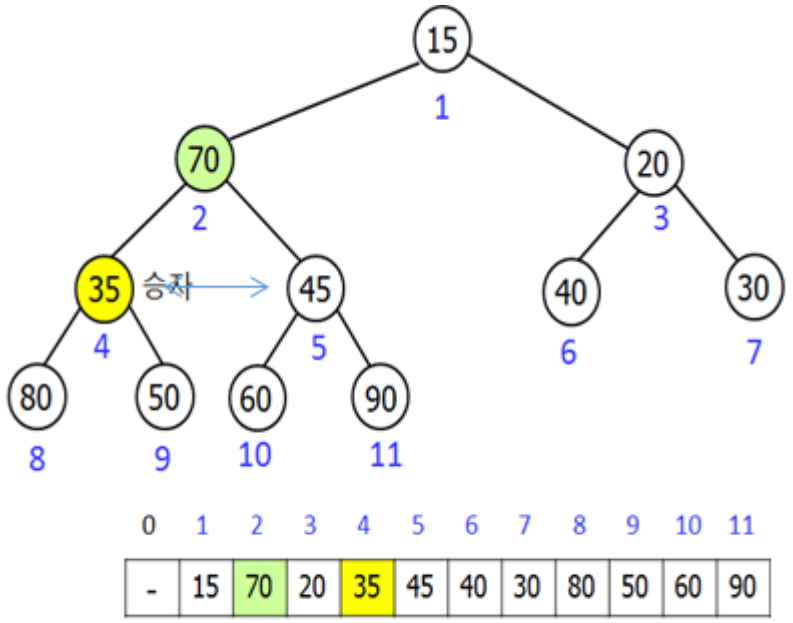
(b) 15와 20 중에 15가 승자

최소 힙 (9/11)

- **최소 힙:** 삭제 연산



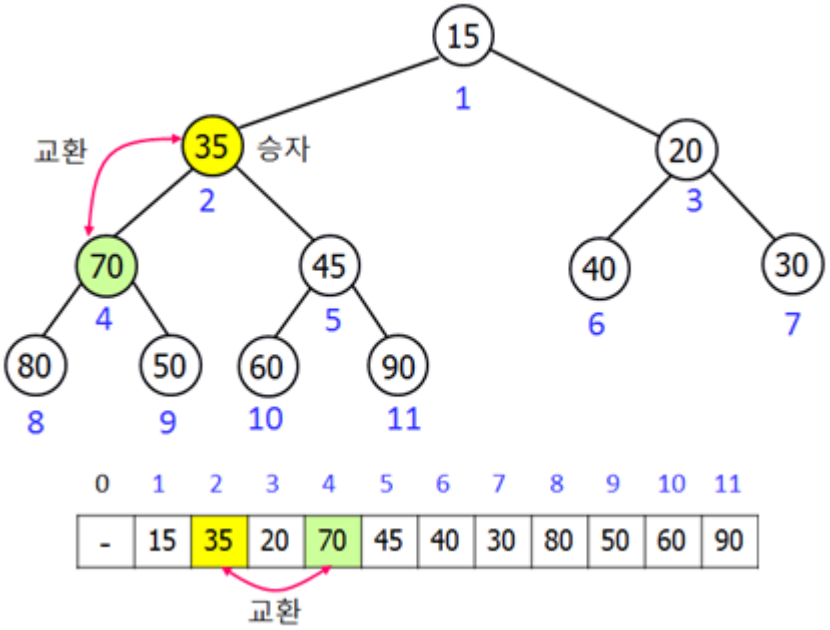
(c) 승자인 15와 루트를 교환



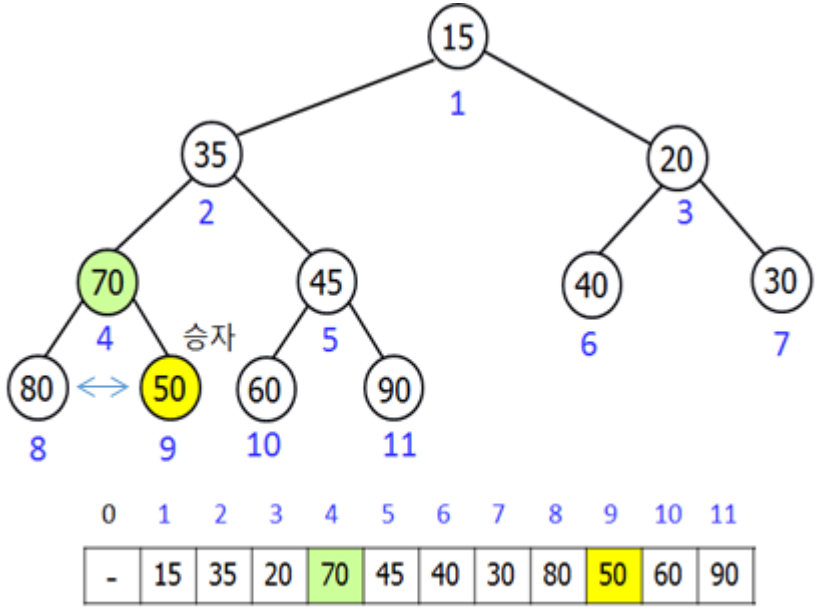
(d) 35와 45 중에 35가 승자

최소 힙 (10/11)

- **최소 힙:** 삭제 연산



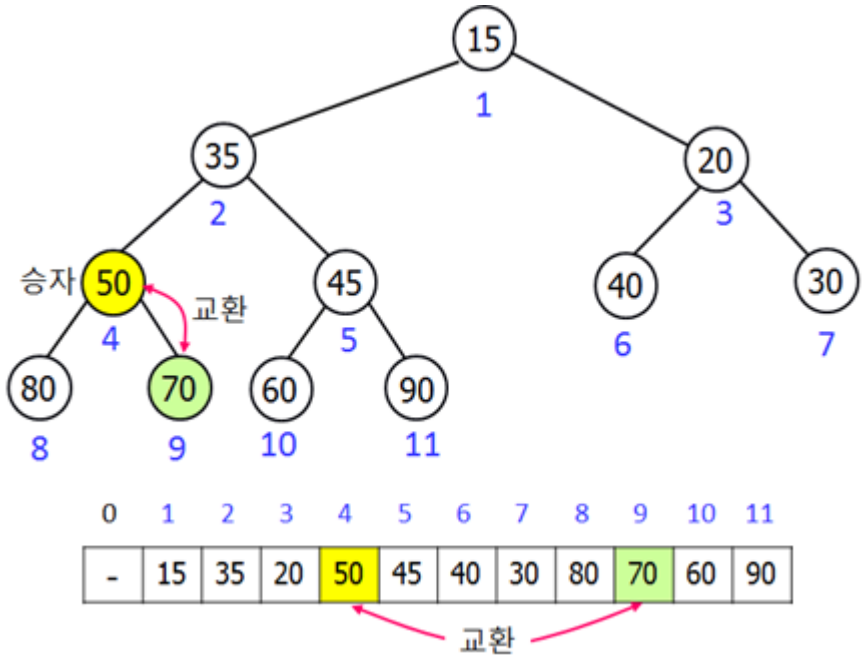
(e) 승자인 35와 70를 교환



(f) 80과 50 중에 50이 승자

최소 힙 (11/11)

- **최소 힙:** 삭제 연산



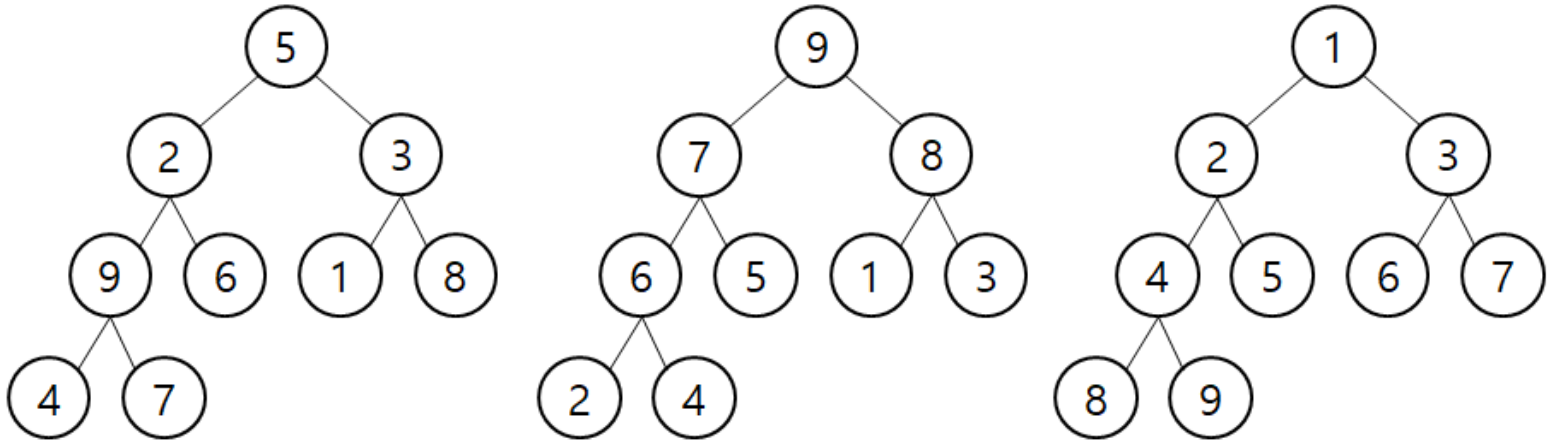
(g) 승자인 50과 70을 교환

우선 순위 큐와 힙

힙 정렬

힙 정렬 (1/1)

- **힙 정렬**(Heap Sort)
  - 힙을 이용한 정렬 과정



리스트를 완전 이진 트리로 표현　　최대힙　　정렬 완료

알고리즘 heapsort()

알고리즘 9-10 힙 정렬

```
◀ 리스트 A[0...n-1]을 정렬한다
heapSort():
    buildHeap()
    for i ← n-1 downto 1
        ❶ A[i] ← deleteMax()
```

buildHeap: $\Theta(n)$

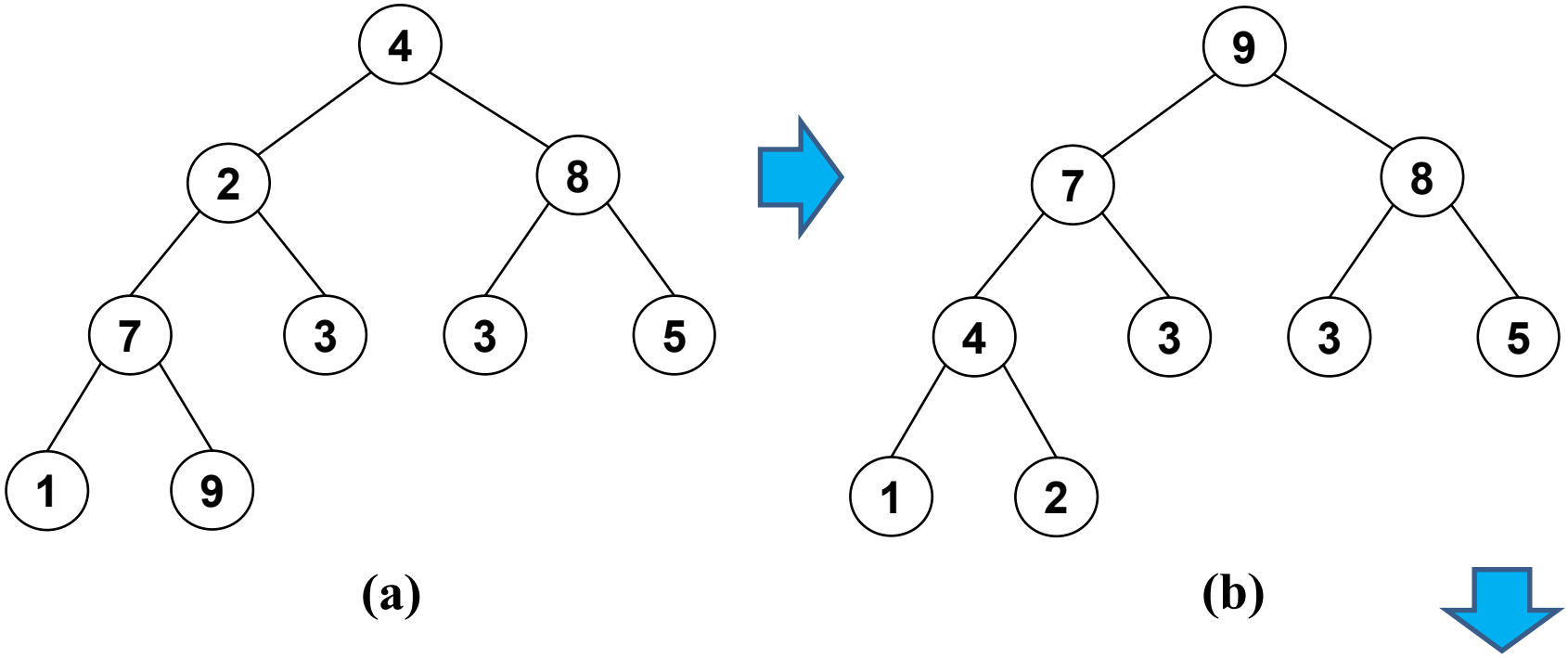
for loop: $O(n\log n)$

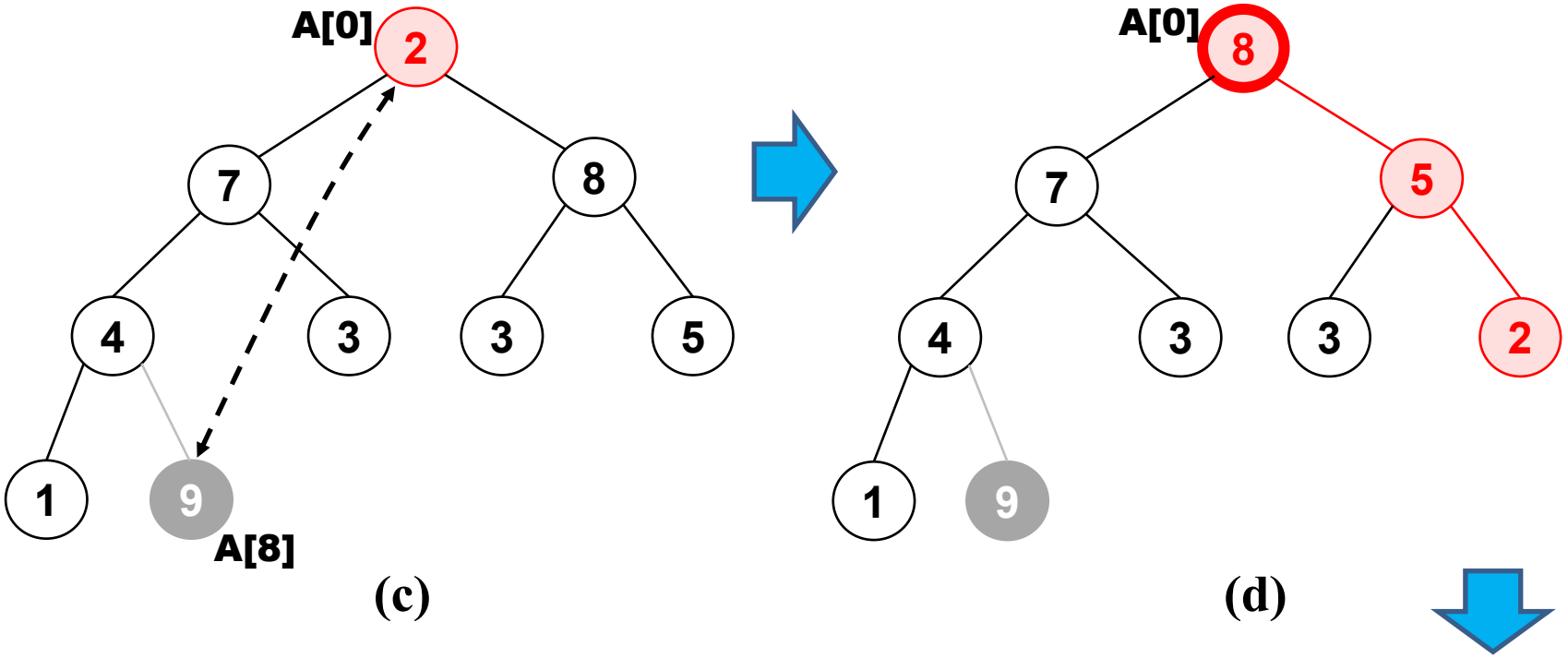
그냥 말하면: $\Omega(n) \sim O(n\log n)$

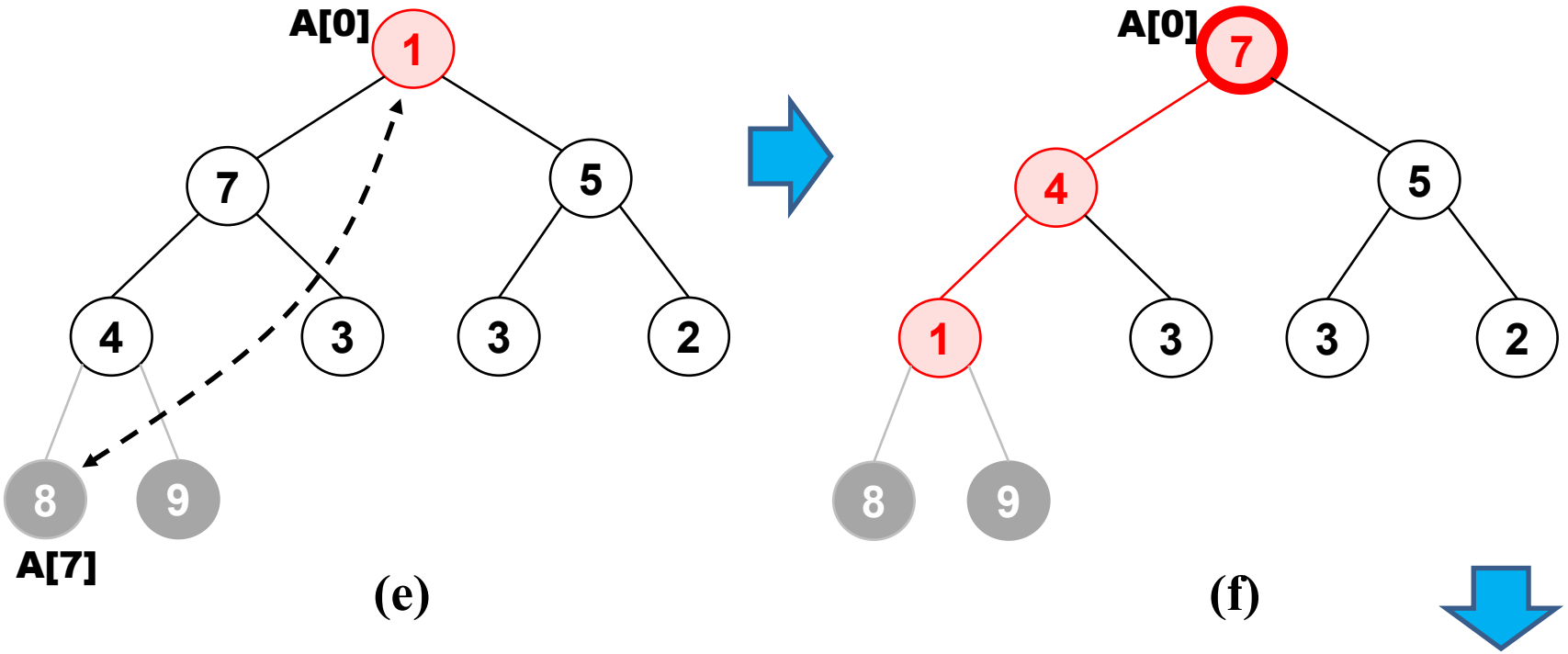
Worst: $\Theta(n\log n)$

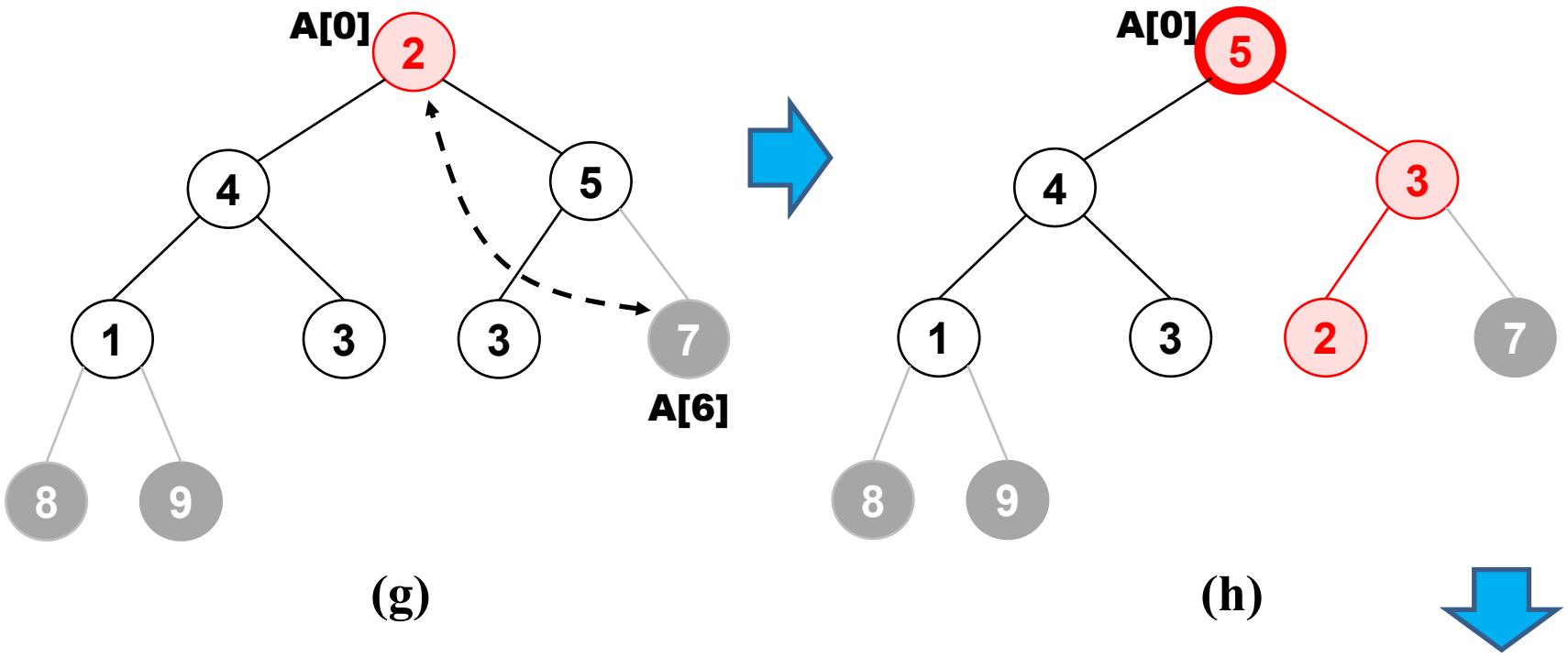
Best: $\Theta(n)$

6. 힙 정렬Heapsort

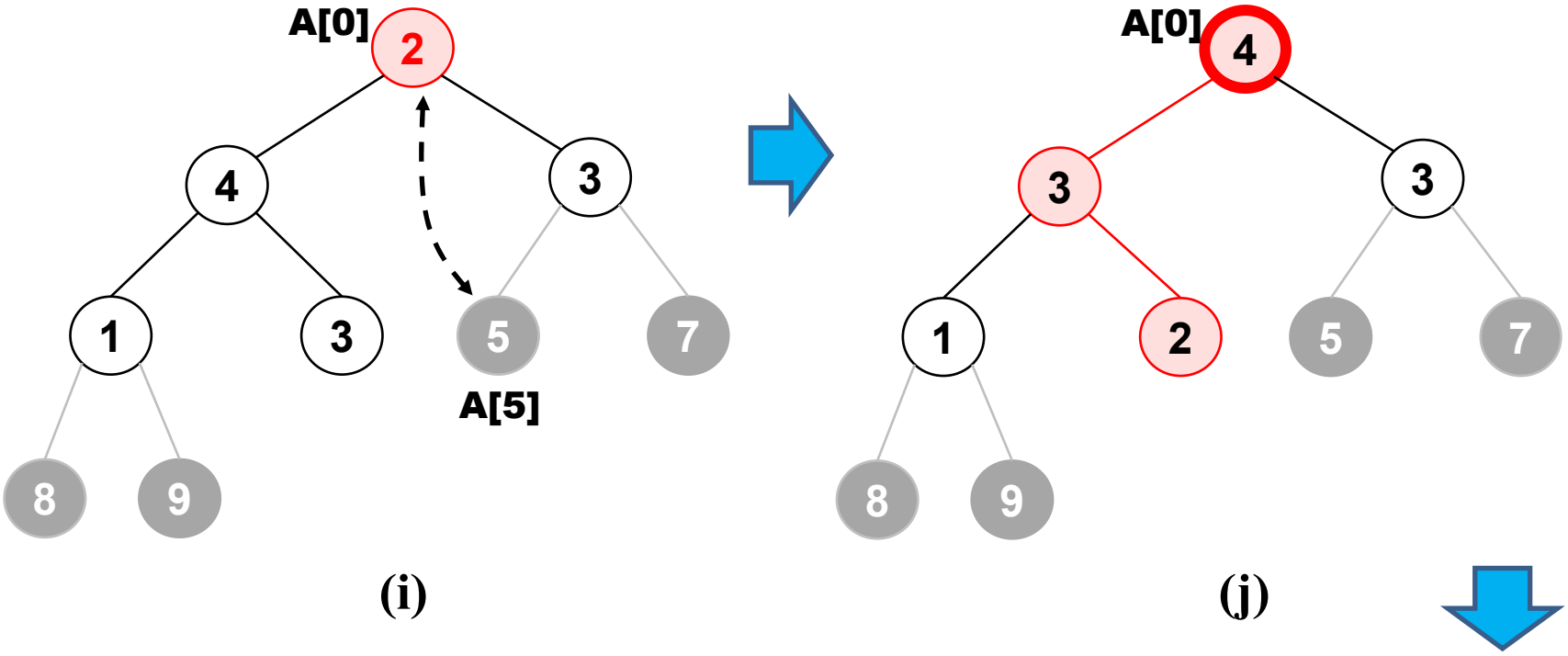


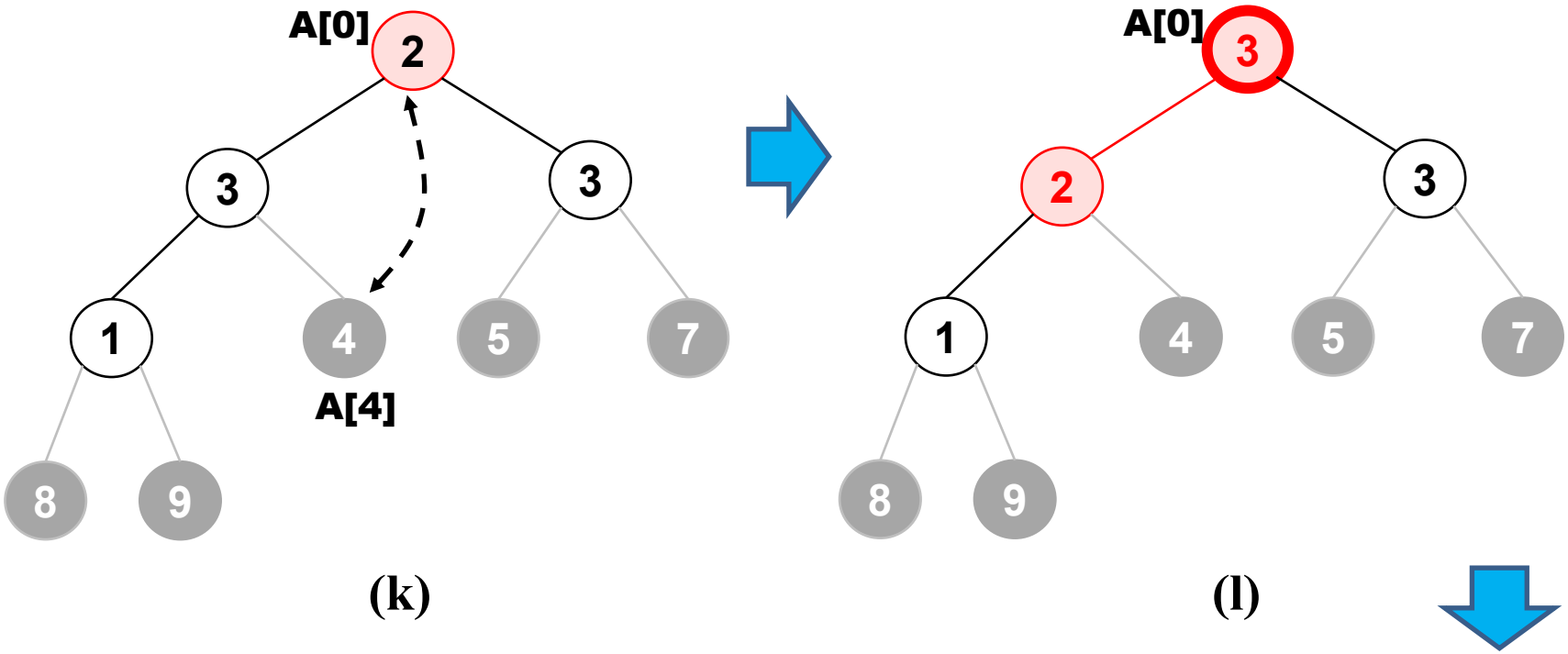

A[0]
2
7
8
4
3
3
5
1
9
A[8]
(c)
A[0]
8
7
5
4
3
3
2
1
9
(d)

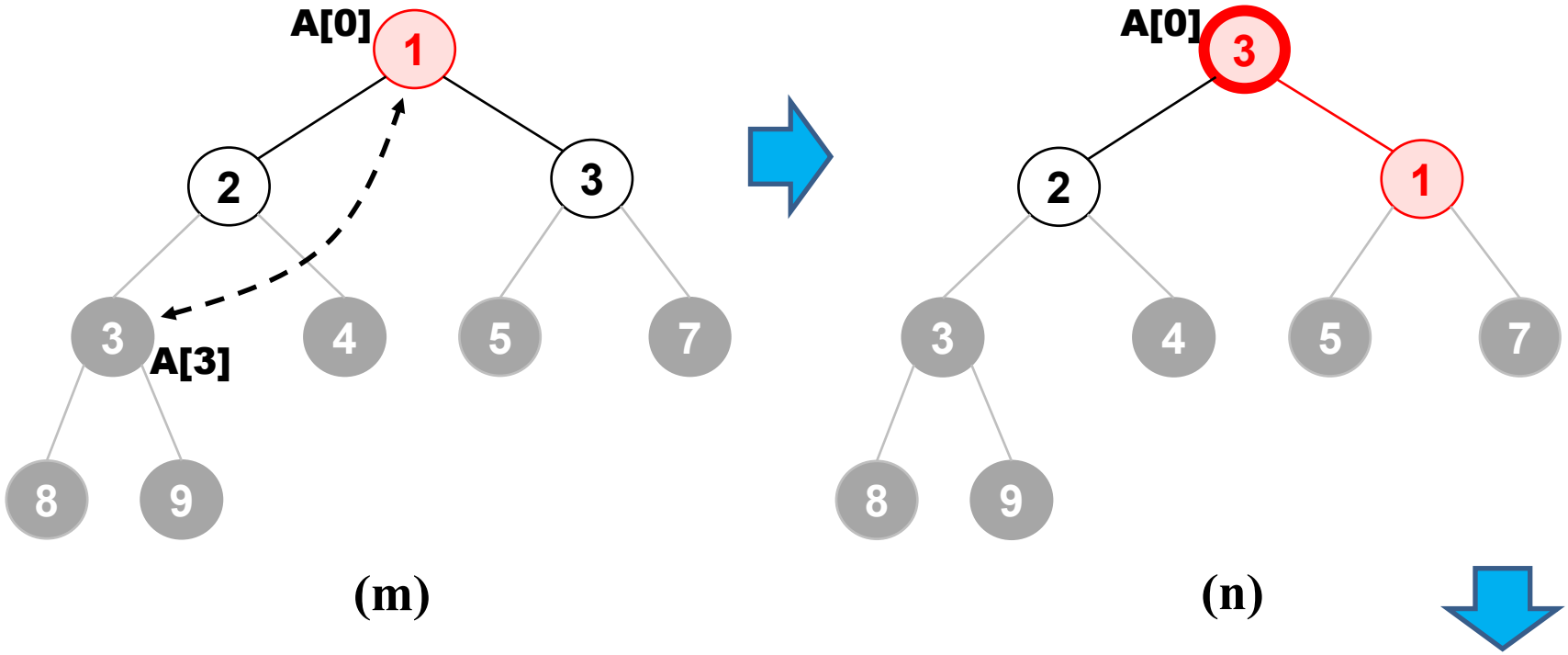

A[0]
1
7
5
4
3
3
2
8
9
A[7]
(e)
A[0]
7
4
5
1
3
3
2
8
9
(f)

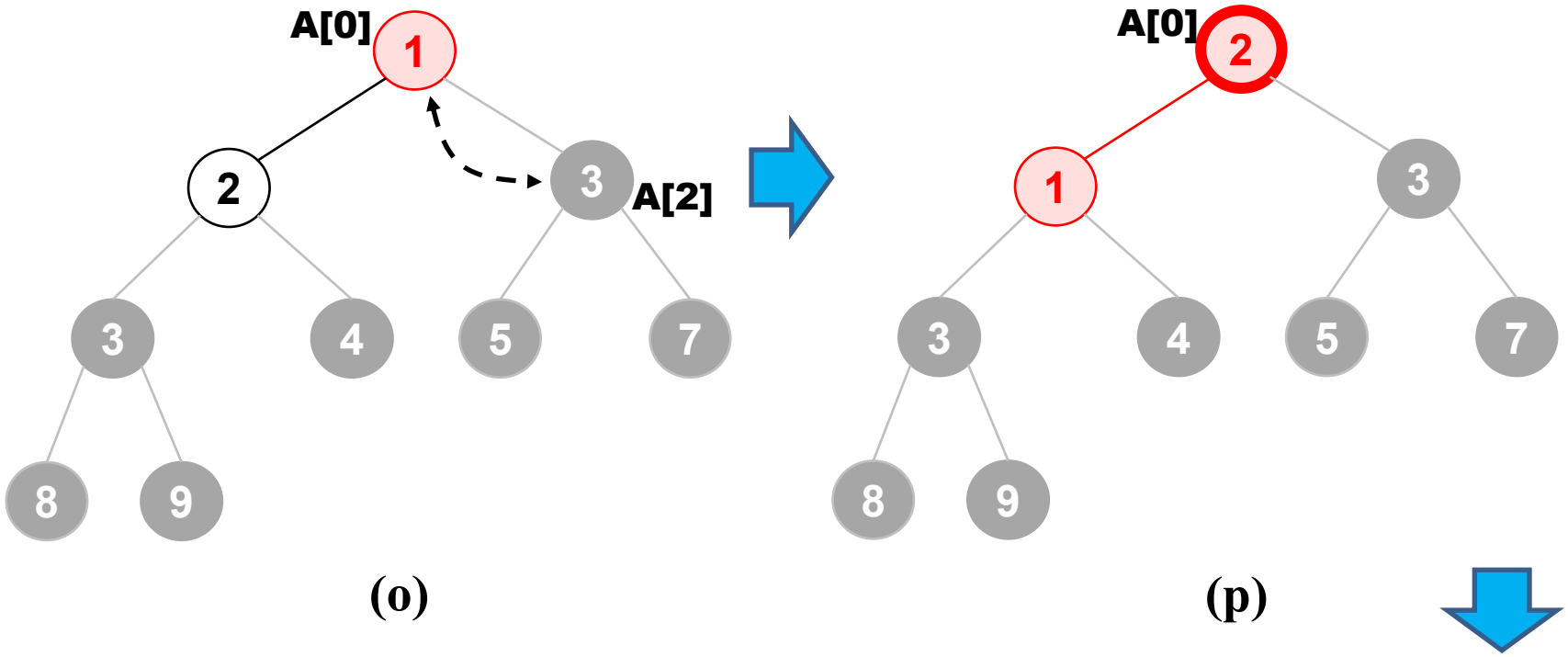


(g) (h)

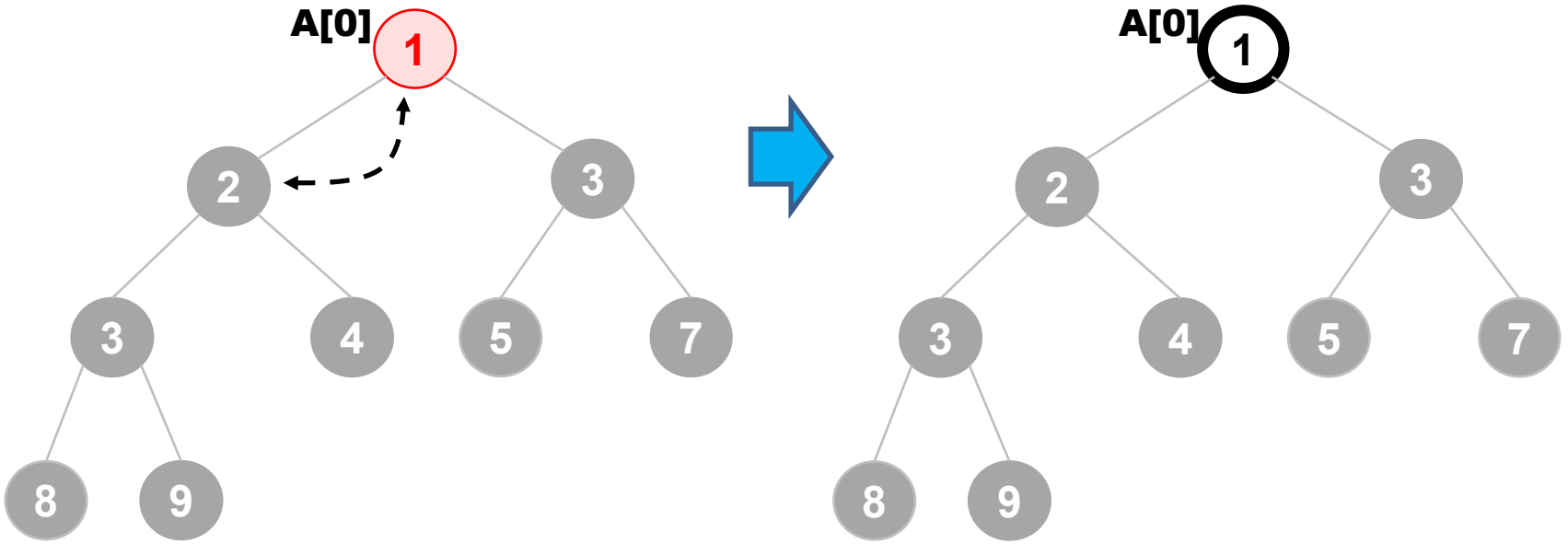

A[0]
2
4
3
1
3
5
7
A[5]
8
9
(i)
A[0]
4
3
3
1
2
5
7
8
9
(j)


A[0]
2
3
3
1
4
5
7
A[4]
8
9
(k)
A[0]
3
2
3
1
4
5
7
8
9
(l)


A[0]
1
2
3
3
A[3]
4
5
7
8
9
(m)
A[0]
3
2
1
3
4
5
7
8
9
(n)



(o) (p)



(q) (r)

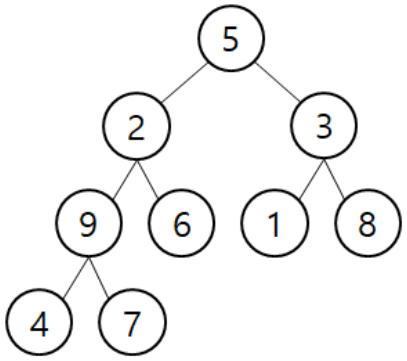
우선 순위 큐와 힙

힙 정렬: 알고리즘 구현

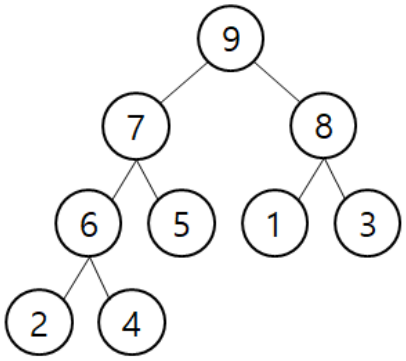
힙 정렬: 알고리즘 구현

힙 정렬: 알고리즘 구현

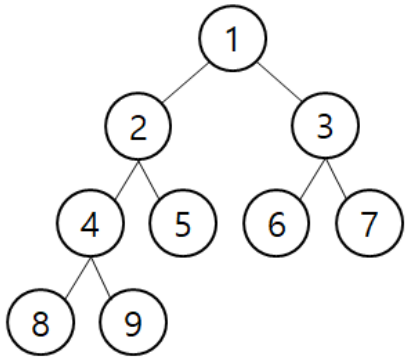
- 임의의 정수(난수)를 힙 정렬 알고리즘을 이용하여 정렬하는 프로그램을 작성하세요.
  - 힙을 이용한 정렬 과정은 다음과 같다.
    - (최대)힙 구성
    - 힙 정렬: 오름차순



리스트를 완전 이진 트리로 표현



최대힙



정렬 완료

[0]	5
[1]	2
[2]	3
[3]	9
[4]	6
[5]	1
[6]	8
[7]	4
[8]	7
[9]	
[10]	
[11]	
[12]	
[13]	
[14]	

우선순위 큐와 힙

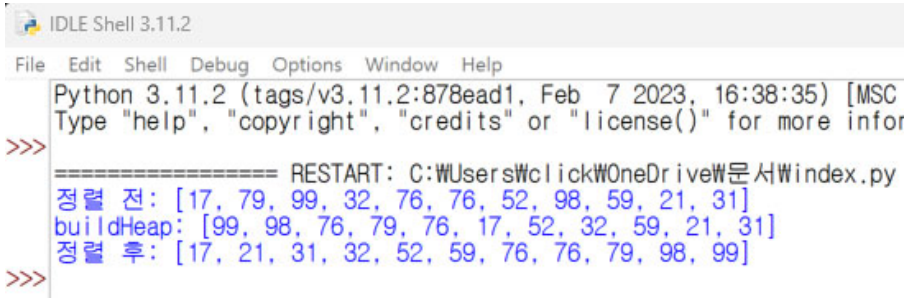
힙 정렬

: 알고리즘 구현(Python)

힙 정렬: 알고리즘 구현(Python)

연습문제 7-2: 힙 정렬 -- 알고리즘 구현(Python)

```
# def heapSort(SList):
# def buildHeap(SList):
# def percolateDown(SList, k:int, end:int):

if __name__ == '__main__' :
    sList = []
    while len(sList)<=10 :
        num = random.randint(0, 99)
        sList.append(num)

    print(f'정렬 전: {sList}')
    heapSort(sList)
    print(f'정렬 후: {sList}')
```



우선 순위 큐와 힙

힙 정렬

: 알고리즘 구현(C++)

힙 정렬: 알고리즘 구현(C++)

연습문제 7-2: 힙 정렬 -- 알고리즘 구현(C++)

```
#include <iostream>
#include <random>          // C++ 11에서 추가
using namespace std;       // C++11 이전: C 스타일 난수 생성(srand와 rand 함수)

#define arrMAXSIZE 10

template <typename E> void HeapSort(E *pArr, const int num);
template <typename E> void buildHeap(E* pArr, const int num);
template <typename E> void percolateDown(E *pArr, const int root, const int num);
template <typename E> void SWAP(E *pa, E *pb);
template <typename E> void PRINT(E *pArr, const int num);

int main(void)
{
    int         arr[arrMAXSIZE] = { 0 };

    // C++ 스타일 난수 생성
    random_device rd;          // 시드 설정: random_device 생성
    mt19937 gen(rd());         // 난수 생성 엔진(mt19937) 초기화
    uniform_int_distribution<int> dis(0, 99);      // 균등 분포 정의: 범위 지정
    for(int i = 0; i < arrMAXSIZE; ++i)
        *(arr + i) = dis(gen);

    cout << "정렬 전: ";          PRINT(arr, arrMAXSIZE);
    HeapSort(arr, arrMAXSIZE);
    cout << "정렬 후: ";          PRINT(arr, arrMAXSIZE);
    return 0;
}
```

우선 순위 큐와 힙

힙 정렬

: 알고리즘 구현(C)

THE C PROGRAMMING LANGUAGE

힙 정렬: 알고리즘 구현(C) (1/4)

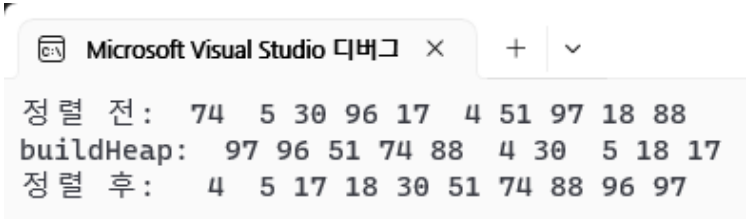
연습문제 7-2: 힙 정렬 -- 알고리즘 구현(C) (1/4)

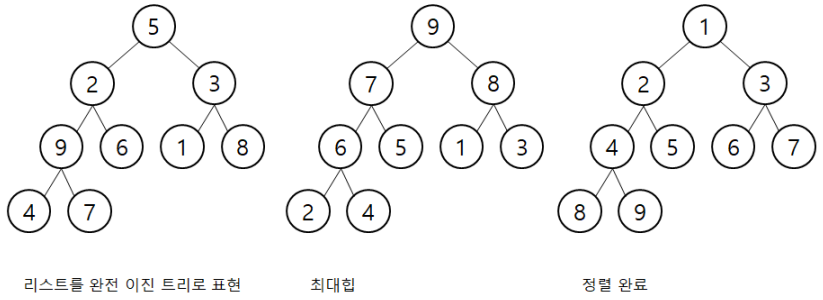
```c
#include <stdio.h>
#include <stdlib.h>   // srand, rand, malloc, calloc, free
#include <stdbool.h>  // bool, true, false
#include <time.h>     // time

#define arrMAXSIZE 10
typedef int element;

void HeapSort(element *pArr, const int nun);
void buildHeap(element *pArr, const int num);
void percolateDown(element *pArr, const int root, const int end);
void SWAP(element *pa, element *pb);
void PRINT(element*pArr, const int num);

int main(void)
{
    int        arr[arrMAXSIZE] = { 0 };

    // 임의의 난수 생성: 0 ~ 99사이의 정수
    srand((unsigned int)time(NULL));
    for(int i = 0; i < arrMAXSIZE; i++)
        *(arr + i) = rand() % 100;

    printf("정렬 전: ");            PRINT(arr, arrMAXSIZE);
    HeapSort(arr, arrMAXSIZE);
    printf("정렬 후: ");            PRINT(arr, arrMAXSIZE);

    return 0;
}
```

힙 정렬: 알고리즘 구현(C) (2/4)

연습문제 7-2: 힙 정렬 -- 알고리즘 구현(C) (2/4)

```
// HeapSort: 힙 정렬
void HeapSort(element *pArr, const int num) {
    buildHeap(pArr, num);

    for(int i = end - 1; i >= 0; --i) {
        // A[i] = deleteMax();
        // i번째 위치에 제일 큰 값을 저장
        SWAP(pArr, pArr + i);

        // pArr[0]의 자료 이동으로 pArr[0:i-1]를 최대 힙으로 재구성
        percolateDown(pArr, 0, i - 1);
    }
}
```

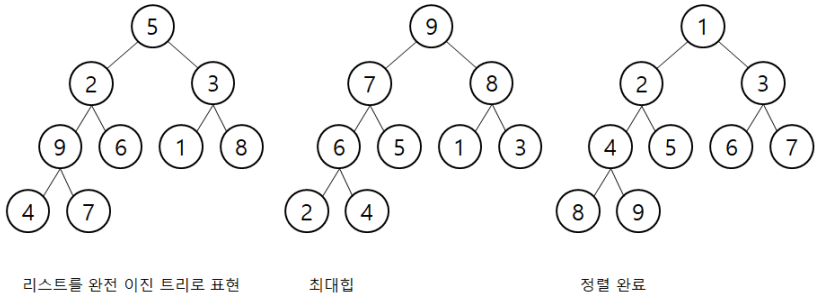




힙 정렬: 알고리즘 구현(C) (3/4)

연습문제 7-2: 힙 정렬 -- 알고리즘 구현(C) (3/4)

```
// buildHeap: 최대 힙 구성
void buildHeap(element *pArr, const int num) {
    for(int i = num / 2; i >= 0; --i)
        percolateDown(pArr, i, num);
    // printf("buildHeap: ");          PRINT(pArr, arrMAXSIZE);
}

void percolateDown(element *pArr, const int root, const int end) {
    int         child = 2 * root + 1;  // 왼쪽 자식
    int         right = 2 * root + 2;  // 오른쪽 자식
    if(child <= end) {
        if(right <= end && pArr[child] < pArr[right])
            child = right;
        // child: A[2k + 1]와 A[2k + 2] 중에 큰 원소의 인덱스

        if(pArr[root] < pArr[child]) {
            SWAP(pArr + root, pArr + child);
            percolateDown(pArr, child, end);
        }
    }
}
```

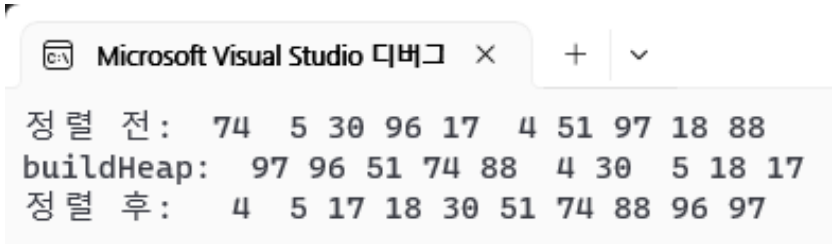


리스트를 완전 이진 트리로 표현 최대힙 정렬 완료

힙 정렬: 알고리즘 구현(C) (4/4)

연습문제 7-2: 힙 정렬 -- 알고리즘 구현(C) (4/4)

```
void SWAP(element *pa, element *pb) {
    element     temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
    return;
}

void PRINT(element *pArr, const int num) {
    for(int i = 0; i < num; ++i)
        printf("%3d", *(pArr + i));
    printf("\n");
    return;
}
```



참고문헌

[1] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛미디어, 2022.

[2] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.

[3] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.

[4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(3판), 개정판, 한빛아카데미, 2024.

[5] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.

[6] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.

[7] "SW Expert Academy", SAMSUNG, 2025 of viewing the site, https://swexpertacademy.com/.

[8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2025 of viewing the site, https://www.acmicpc.net/.

[9] "programmers", grepp, 2025 of viewing the site, https://programmers.co.kr/.

[10] "goormlevel", goorm, 2025 of viewing the siteh, https://level.goorm.io/