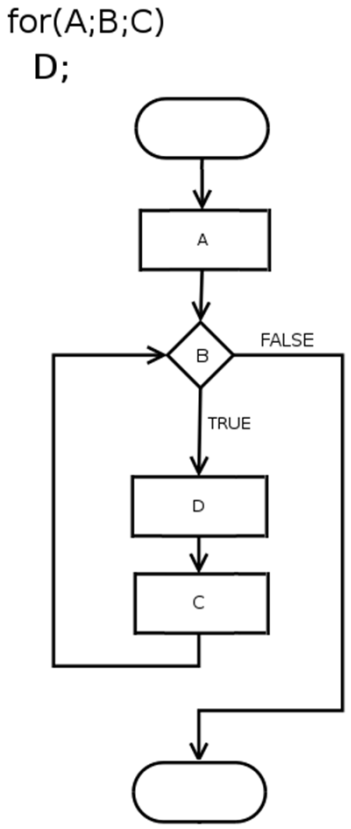
Clickseo.com

자료구조 & 알고리즘

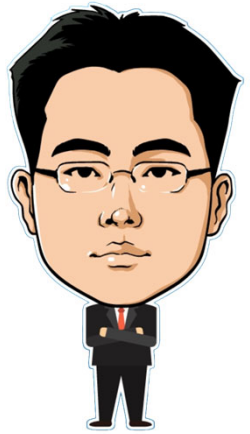


트리
(Tree)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



CC BY NC ND

목 차

CC BY NC ND

- 트리의 이해
- 이진 트리 구현
- 우선 순위 큐와 힙

트리의 이해 (1/3)

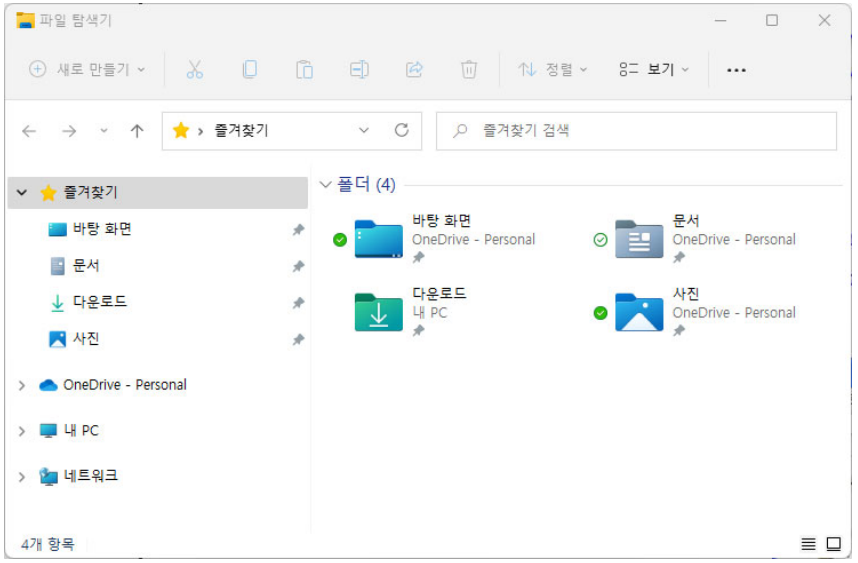
● 트리(Tree)

❍ 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**

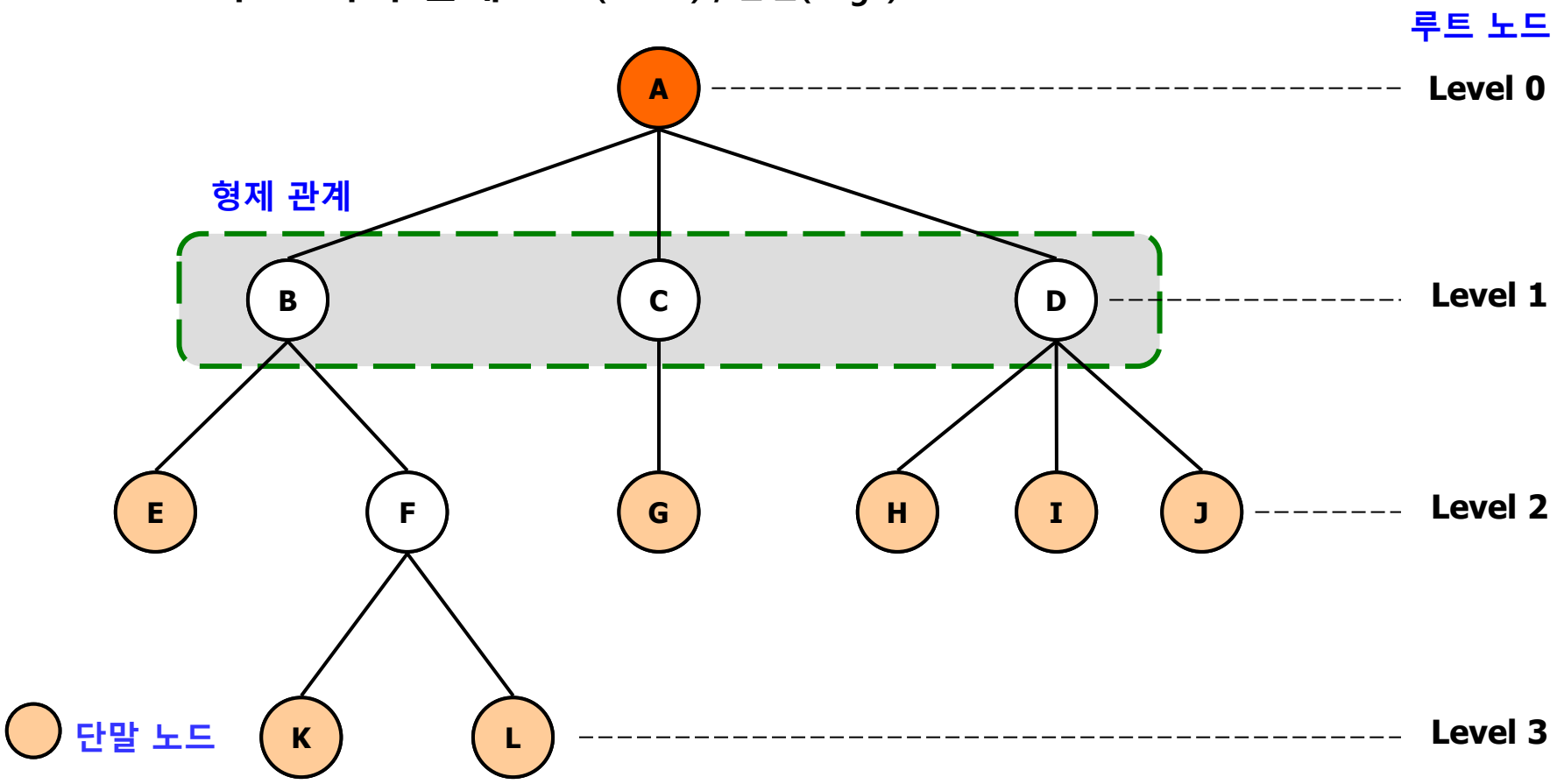
❍ 트리 구조의 예

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



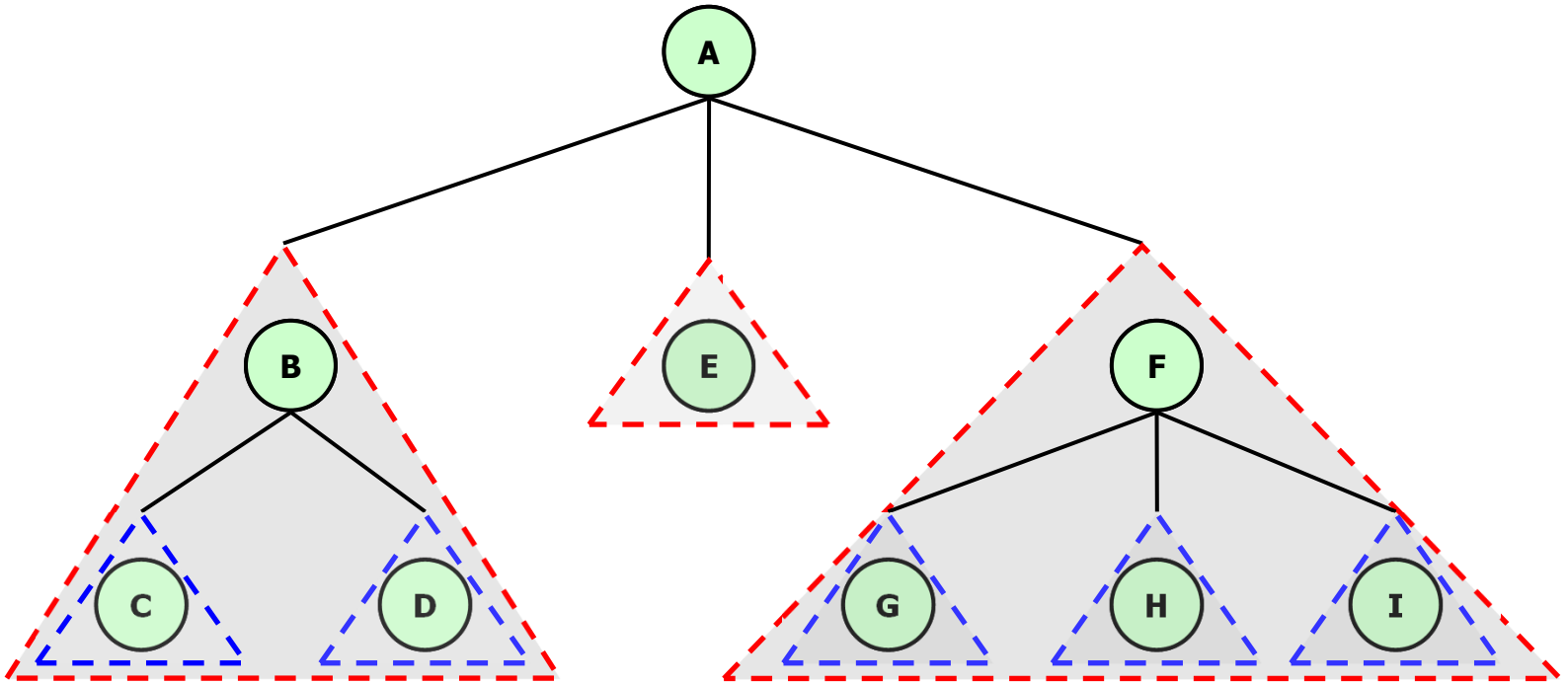
트리의 이해 (2/3)

- 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.



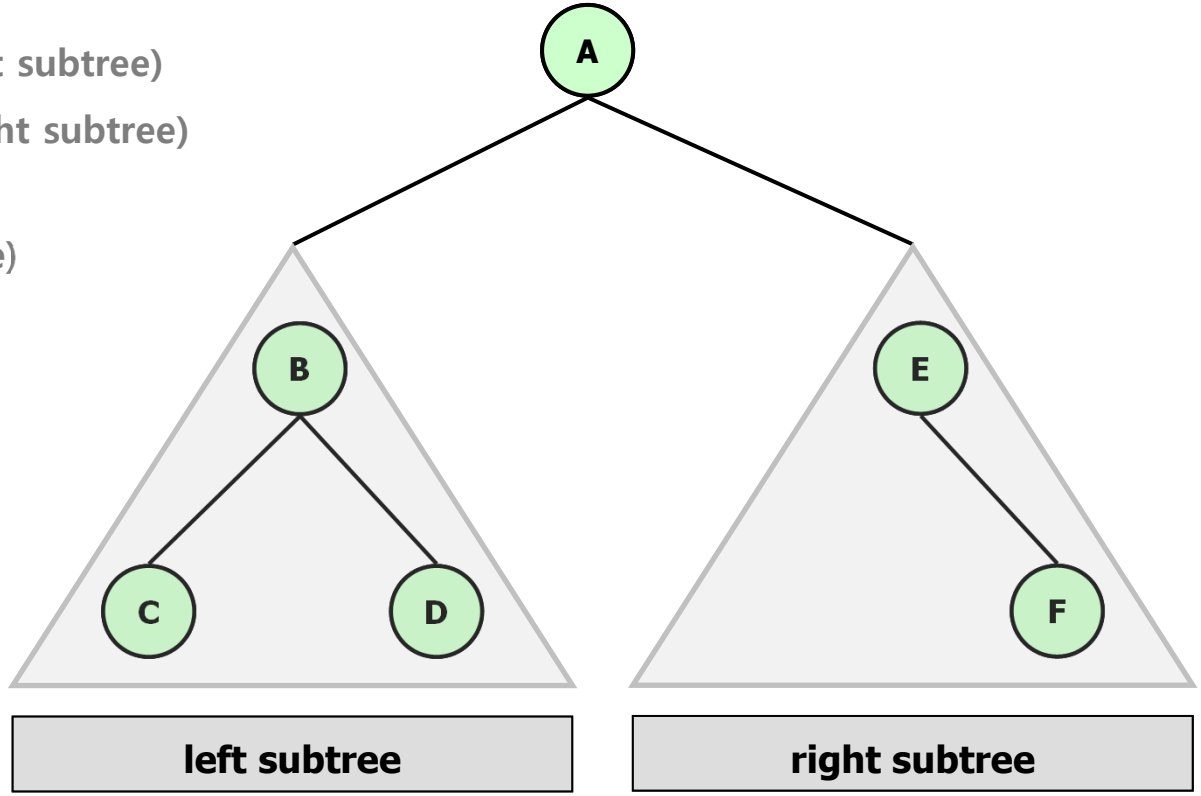
트리의 이해

CC BY NC ND

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

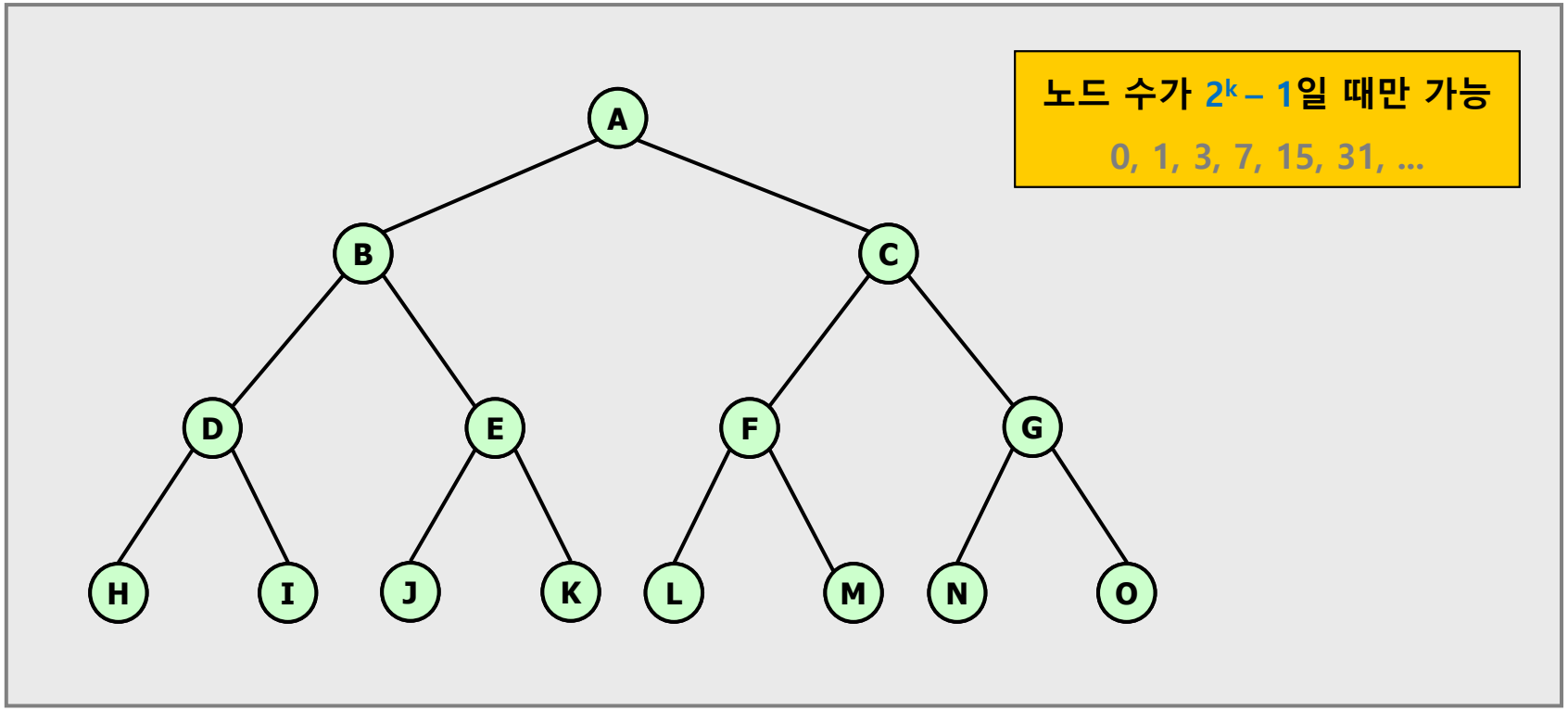
이진 트리 (1/4)

- **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)



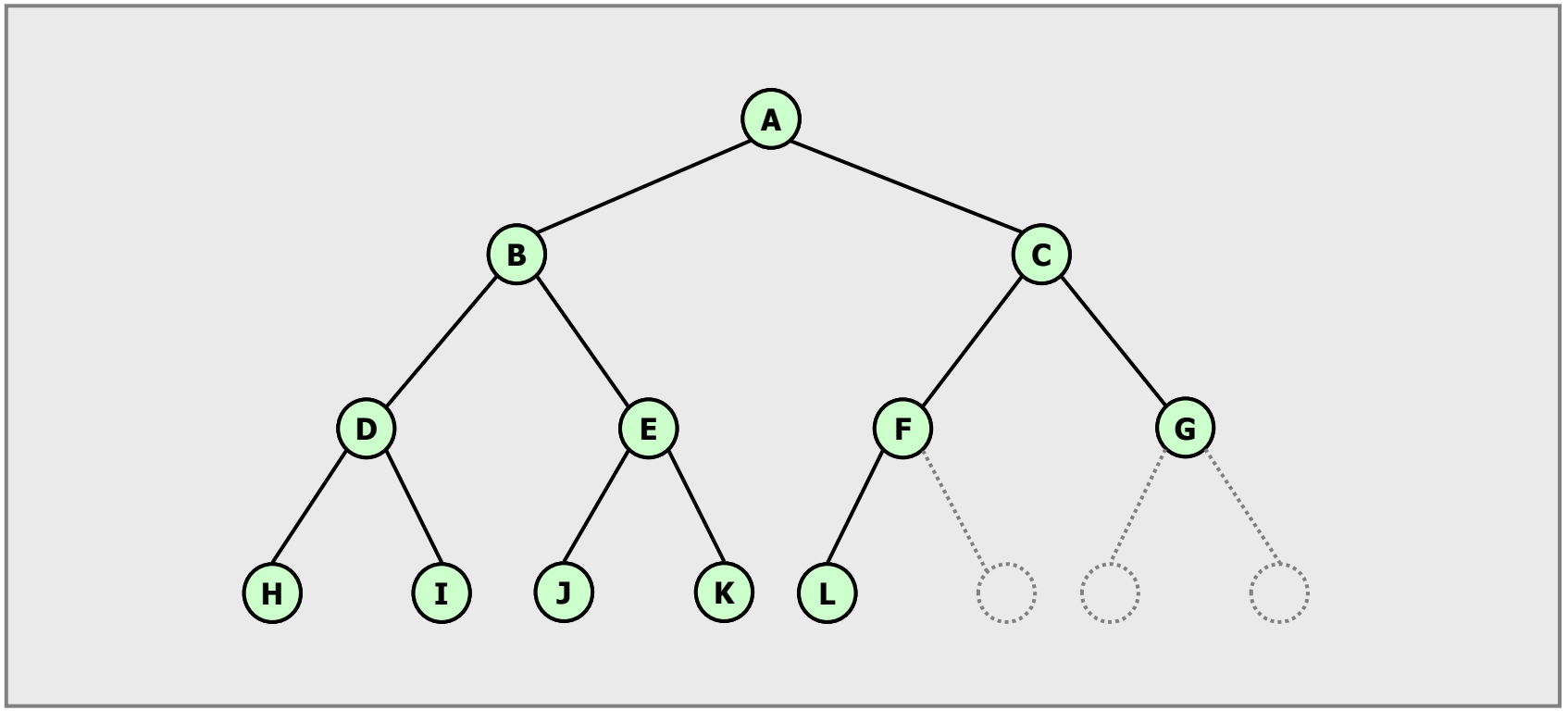
이진 트리 (2/4)

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



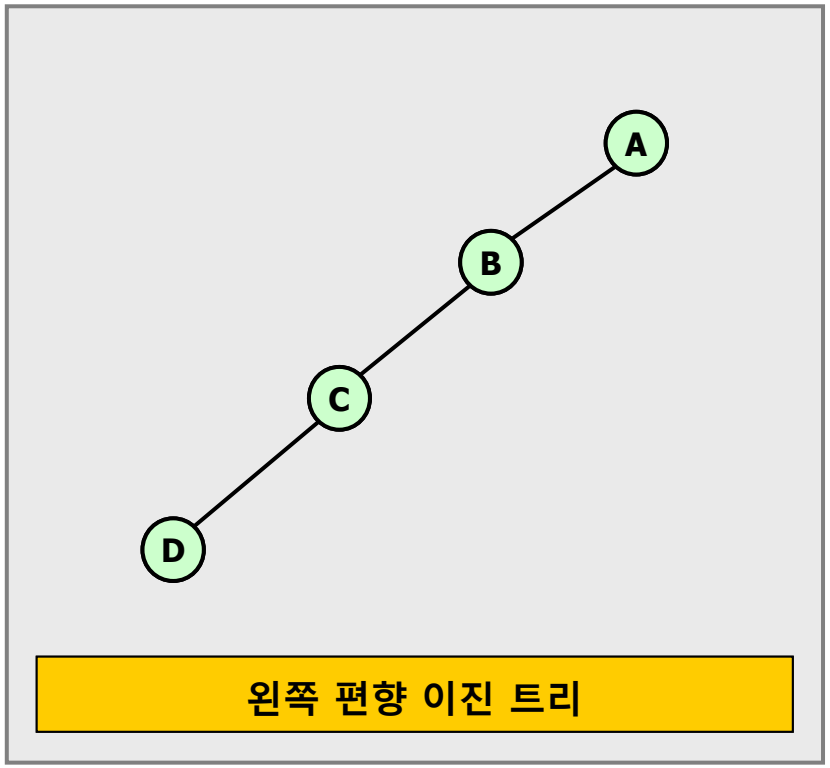
이진 트리 (3/4)

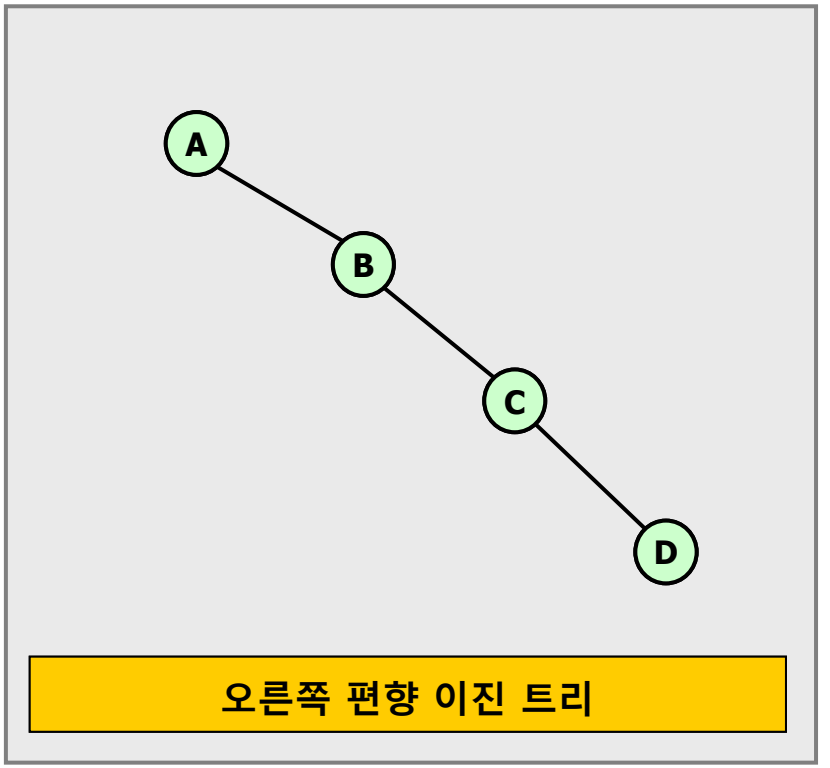
- **완전 이진 트리**(Complete Binary Tree)
  - 노드의 수가 맞지 않아 포화 이진 트리를 만들 수 없으면 맨 마지막 레벨은 왼쪽부터 채워 나간다.



이진 트리 (4/4)

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





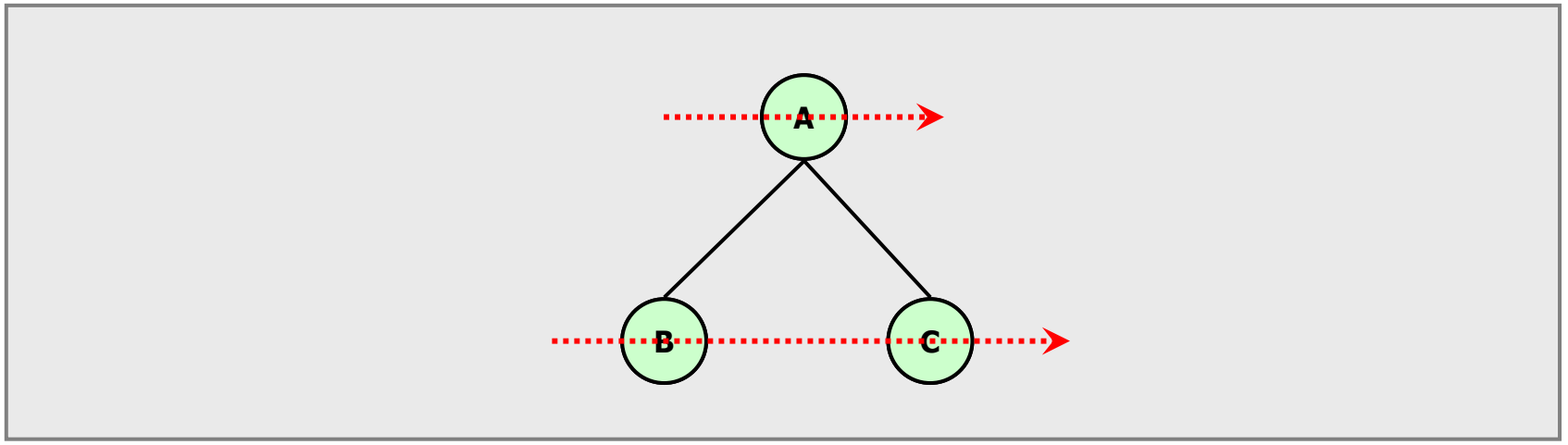
CC BY NC ND

이진 트리

이진 트리 순회

이진 트리 순회 (1/4)

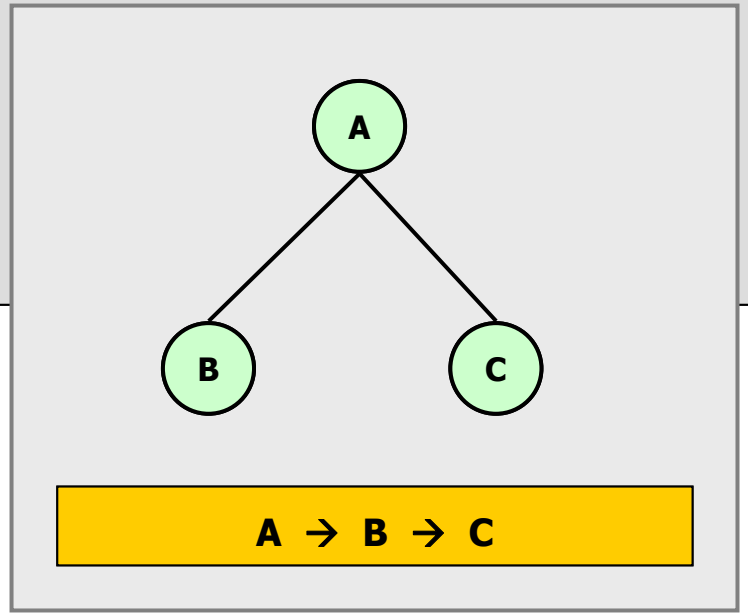
- **순회**(traversal)
  - 깊이 우선 순회: 스택을 이용하여 구현
    - 전위 순회(preorder traversal)
    - 중위 순회(inorder traversal)
    - 후위 순회(postorder traversal)
  - 너비 우선 순회: 큐를 이용하여 구현
    - 다음 레벨의 노드들을 처리하기 전에 노드의 자식 모두를 처리



이진 트리 순회 (2/4)

- **깊이 우선 순회:** 전위 순회
  - 전위 순회(preorder traversal)

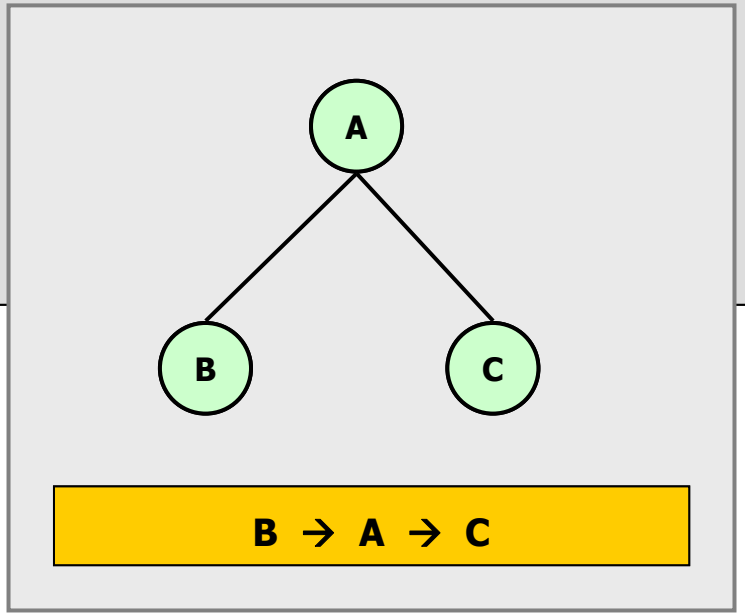
```
preorder(T)
    if(T ≠ NULL) then
    {
        visit T.data;
        preorder(T.Llink);
        preorder(T.Rlink);
    }
end preorder()
```



이진 트리 순회 (3/4)

- **깊이 우선 순회: 중위 순회**
  - 중위 순회(inorder traversal)

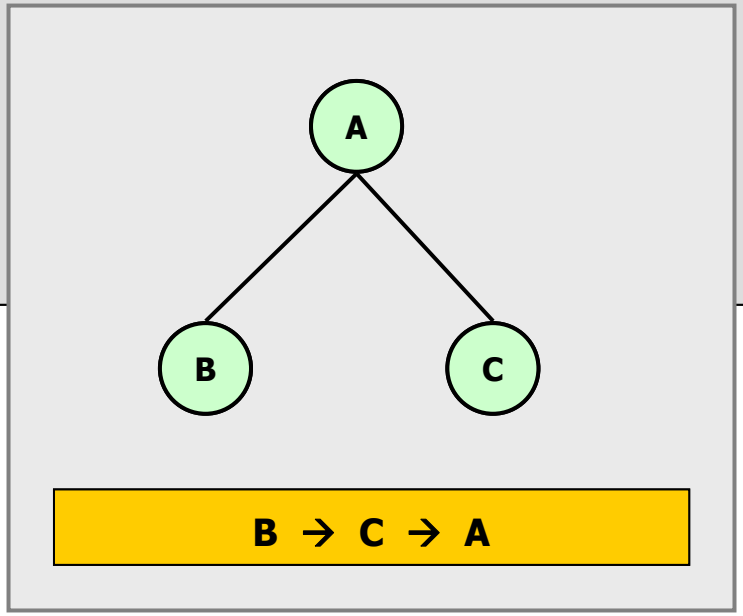
```
inorder(T)
    if(T ≠ NULL) then
    {
        inorder(T.Llink)
        visit T.data;
        inorder(T.Rlink);
    }
end inorder()
```



이진 트리 순회 (4/4)

- **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 구현

CC BY NC ND

- 트리의 이해
- **이진 트리 구현**
  - 순차 자료 구조
  - 연결 자료 구조
- 우선 순위 큐와 힙

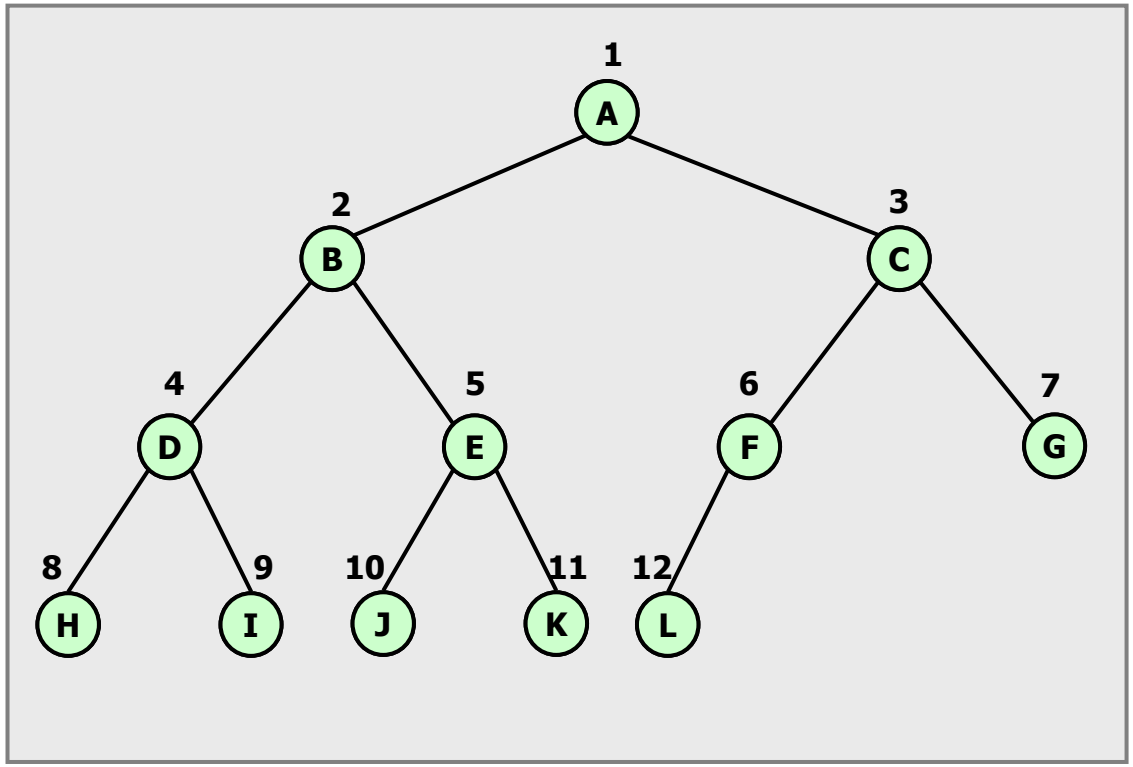
CC BY NC ND

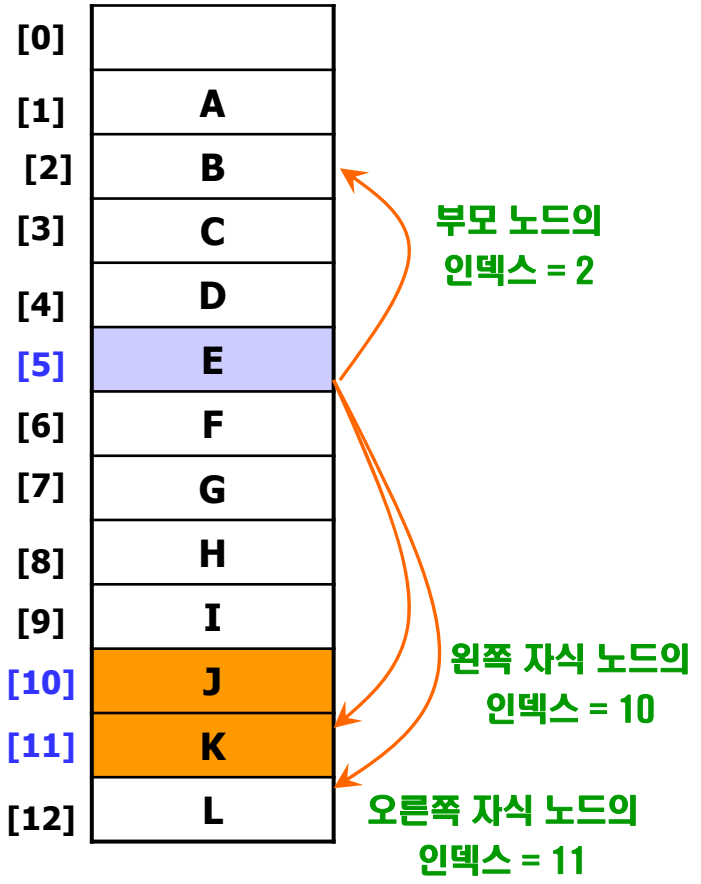
이진 트리 구현

순차 자료 구조

이진 트리 구현: 순차 자료 구조 (1/2)

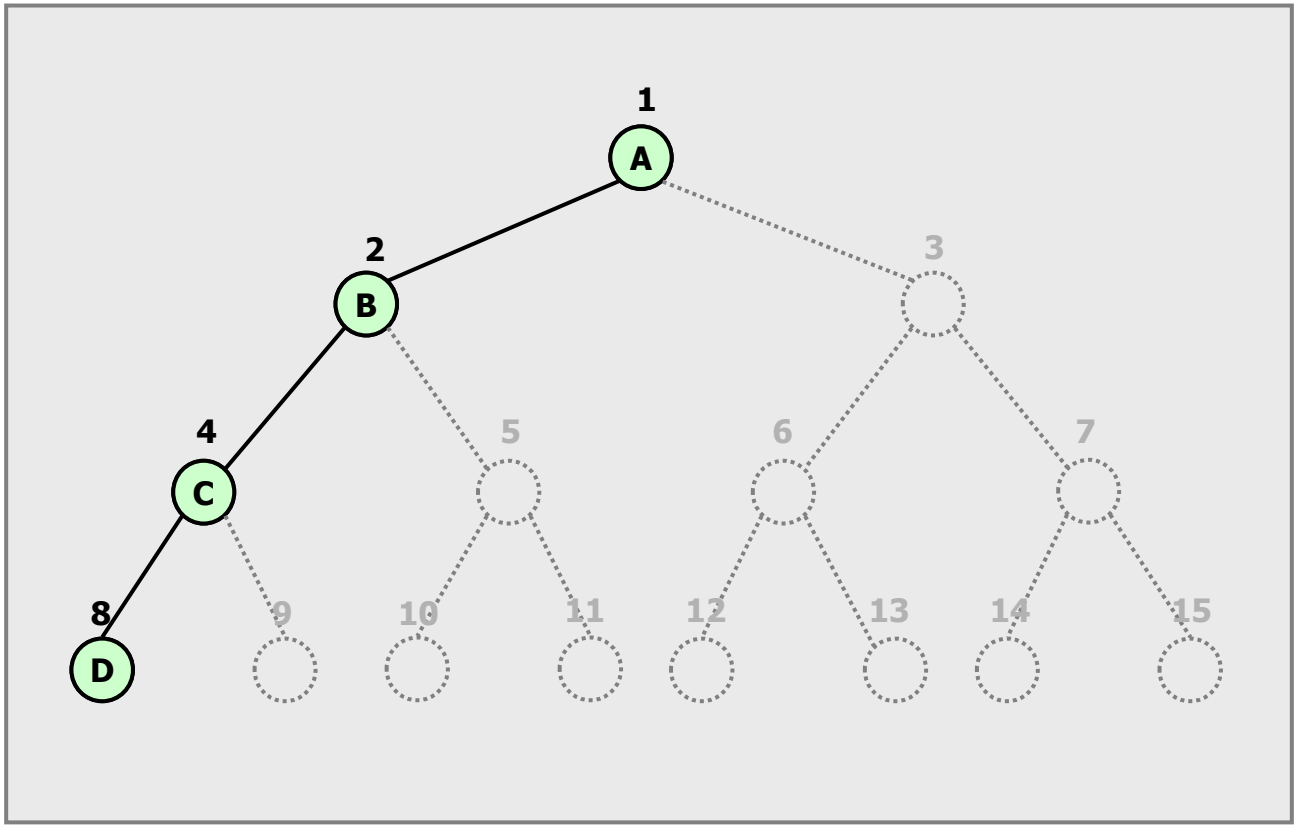
- **이진 트리 구현:** 순차 자료구조
  - 완전 이진 트리의 배열 표현





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]	

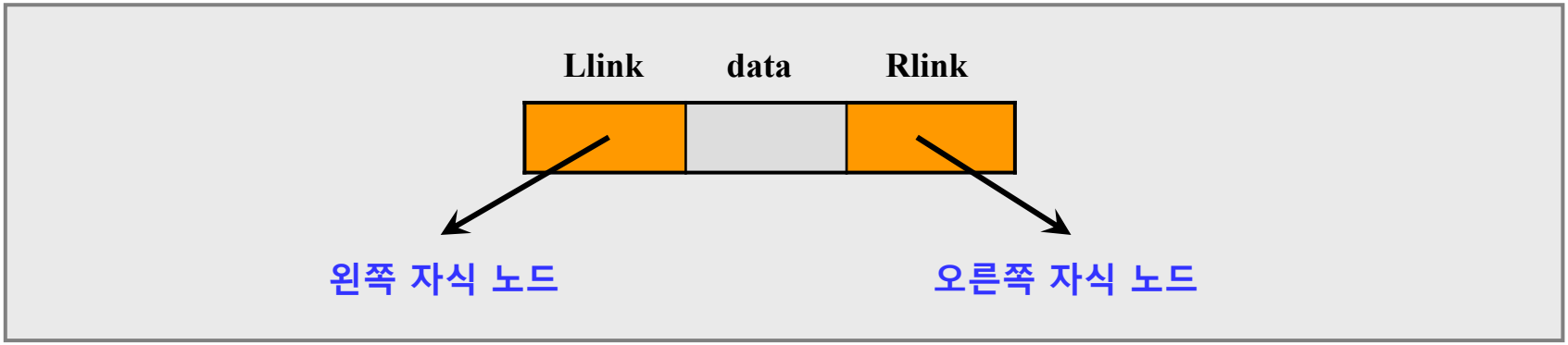
CC BY NC ND

이진 트리

이진 트리 구현: 연결 자료구조

이진 트리 구현: 연결 자료 구조 (1/6)

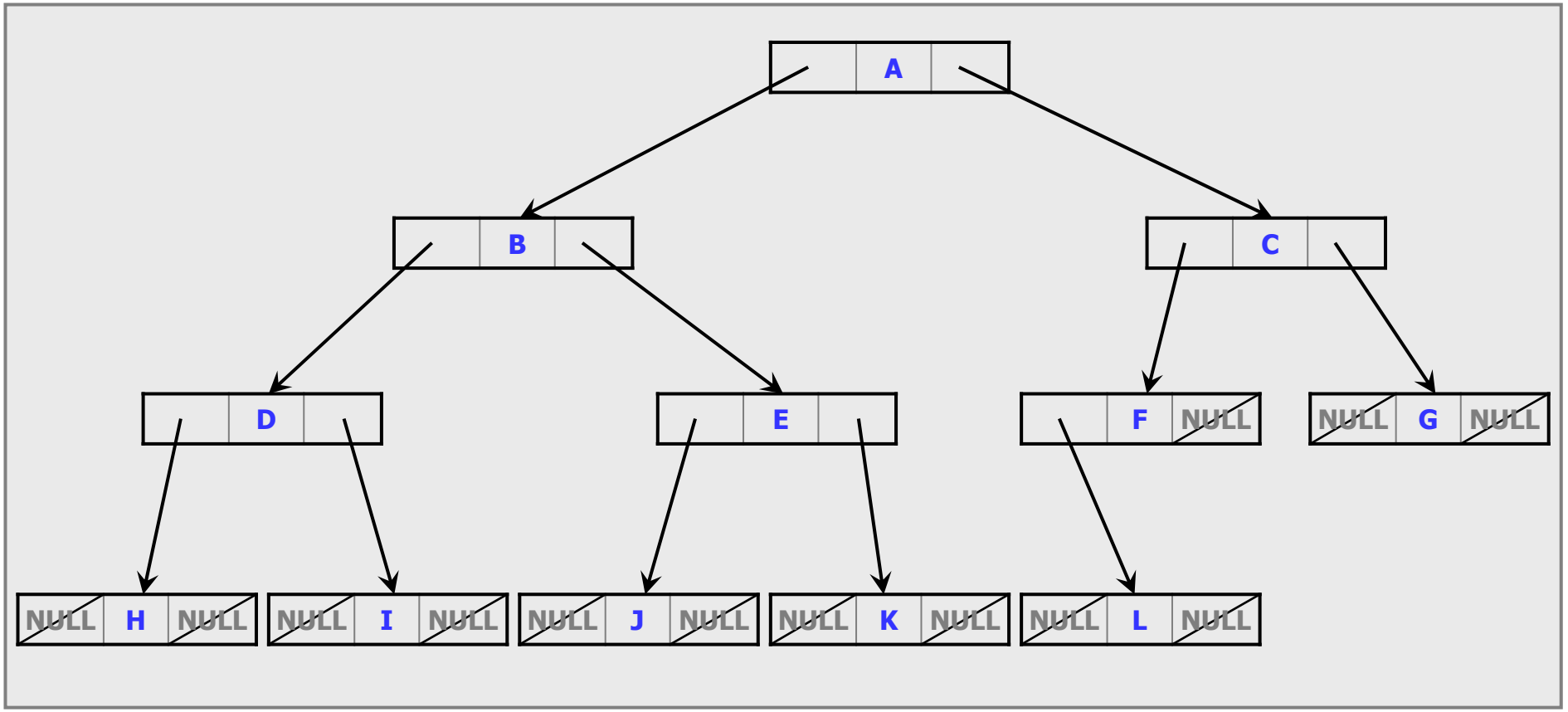
- **이진 트리 구현:** 연결 자료 구조




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

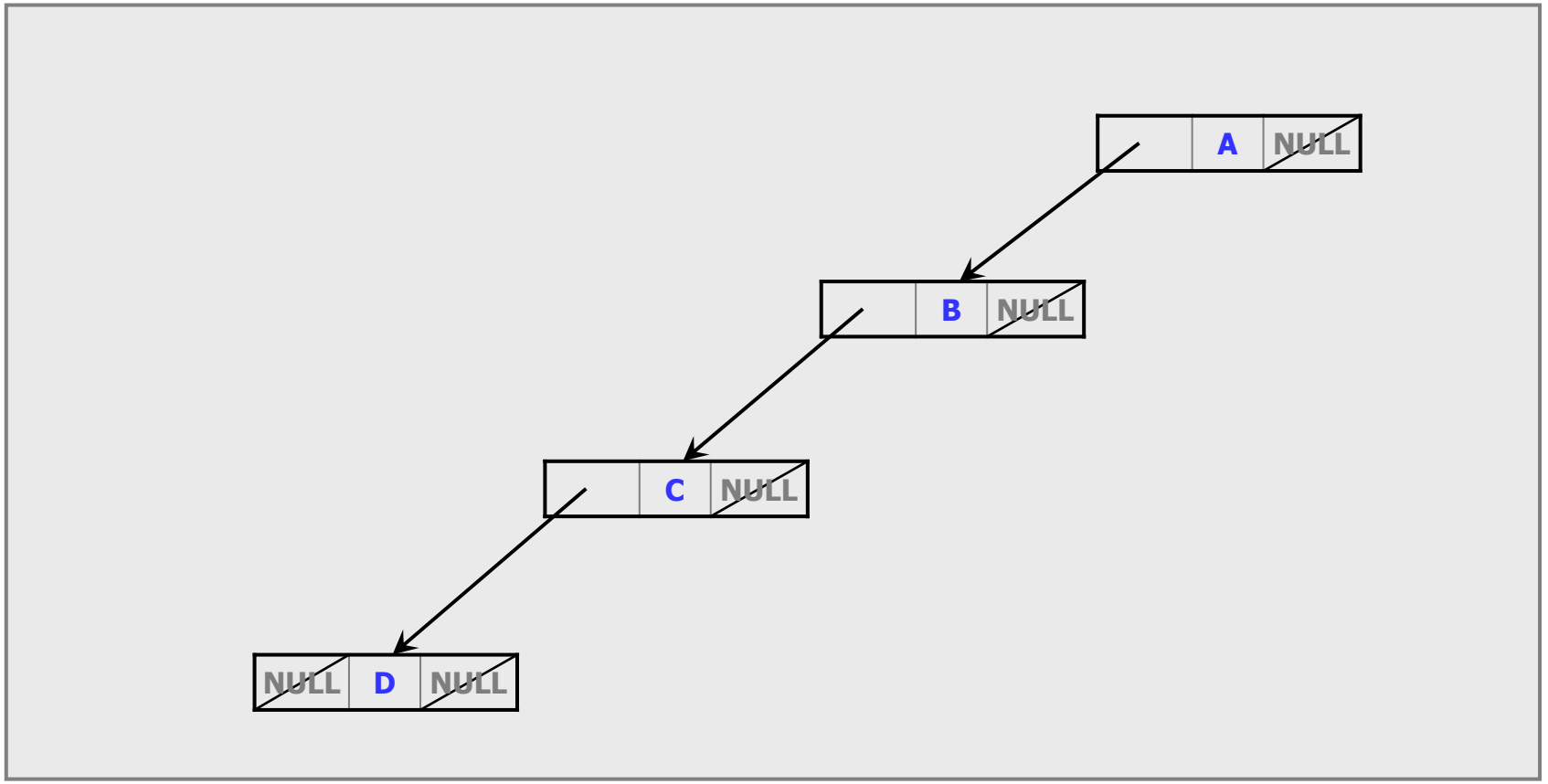
이진 트리 구현: 연결 자료 구조 (2/6)

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



이진 트리 구현: 연결 자료 구조 (3/6)

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



CC BY NC ND

이진 트리 구현

연결 자료 구조: C/C++, Python

이진 트리 구현: 연결 자료 구조 (4/6)

- **이진 트리 구현:** 연결 자료구조

```
// 이진 트리 구현: 단순 연결 리스트
// #pragma once
#ifndef __BinaryTree_H_
#define __BinaryTree_H_
// 이진 트리 노드: data, Llink, Rlink
typedef struct _treeNode {
        char               __data;
        struct _treeNode*  __Llink;
        struct _treeNode*  __Rlink;
} treeNode;
#endif

// 이진 트리 구현: 이진 트리 생성
treeNode* makeBinaryTree(char*);
treeNode* makeTreeNode(char);
int       isOperator(int);
int       isLegal(char*);

// 이진 트리 순회: 전위.중위.후위 순회
void      Preorder(treeNode*);
void      Inorder(treeNode*);
void      Postorder(treeNode*);

// 이진 트리 순회: 너비 우선 순회
void      Preorder(treeNode*);
```

이진 트리 구현: 연결 자료 구조 (5/6)

- **이진 트리 구현:** 연결 자료구조

```
template <typename E>
class BinaryTreeNode {
private:
    E                       data;
    BinaryTreeNode<E>*      Llink;
    BinaryTreeNode<E>*      Rlink;
    template <typename E> friend class LinkedBinaryTree;
};

template <typename E>
class LinkedBinaryTree {
private:
    BinaryTreeNode<E>* root;
public:
    LinkedBinaryTree();
    ~LinkedBinaryTree();
    BinaryTreeNode<E>* makeBinaryTree(const char& ch) const;
    BinaryTreeNode<E>* makeLinkedBinaryTree(const char* pStr) const;
    void  Preorder(void) const;
    void  Inorder(void) const;
    void  Postorder(void) const;
    void  Levelorder(void) const;
};
```

이진 트리 구현: 연결 자료 구조 (6/6)

- **이진 트리 구현:** 연결 자료구조

```
from LinkedStack import LinkedStack
from LinkedQueue import LinkedQueue

def isOperator(op) -> bool :

class TreeNode :
    def __init__ (self, data, Llink=None, Rlink=None):
        self.data = data
        self.Llink = Llink
        self.Rlink = Rlink

class LinkedBinaryTree :
    def __init__(self):
        self.__root = None

    def makeLinkedBinaryTree(self, postfix) -> TreeNode:
    def Preorder(self):
    def Inorder(self) :
    def Postorder(self) :
    def Levelorder(self) :
```

CC BY NC ND

이진 트리 구현

연결 자료구조: C++

THE C PROGRAMMING LANGUAGE

CC BY NC ND

이진 트리 구현

연결 자료구조: C++

CC BY NC ND

이진 트리 구현

연결 자료구조: Python

우선 순위 큐와 힙

cc BY NC ND

- 트리의 이해
- 이진 트리 구현
- **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.

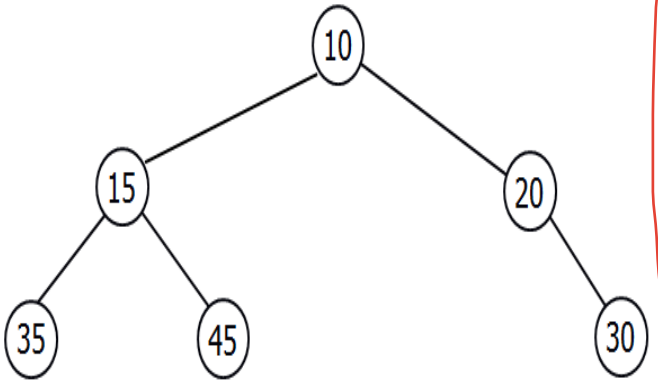
  - 우선 순위 큐(Priority Queue)
    - 가장 높은 우선순위를 가진 항목에 접근, 삭제와 임의의 우선순위를 가진 항목을 삽입을 지원하는 자료구조
    - 스택이나 큐도 일종의 우선 순위 큐
      - **스택:** 가장 마지막으로 삽입된 항목이 가장 높은 우선순위를 가진다.
        - » 따라서 최근 시간일수록 높은 우선순위를 부여한다.
      - **큐:** 먼저 삽입된 항목이 우선순위가 더 높다
        - » 따라서 이른 시간일수록 더 높은 우선순위를 부여한다.
      - 삽입되는 항목이 임의의 우선순위를 가지면 스택이나 큐는 새 항목이 삽입될 때마다 저장되어 있는 항목들을 우선순위에 따라 정렬해야 하는 문제점이 있음.

우선 순위 큐와 힙 (2/3)

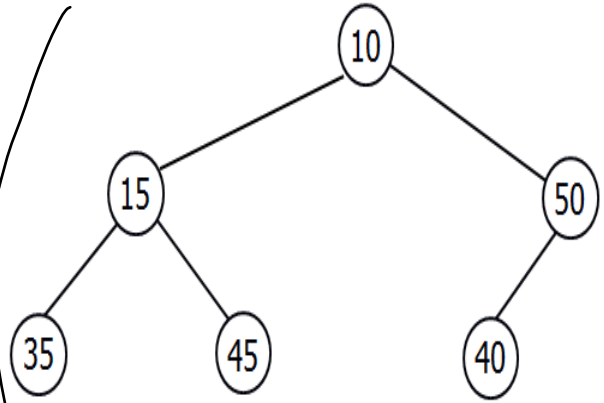
- **힙:** 완전 이진 트리
  - 완전 이진 트리로서 부모의 우선 순위가 자식의 우선 순위보다 높은 자료구조

어느 트리가 이진 힙 일까?

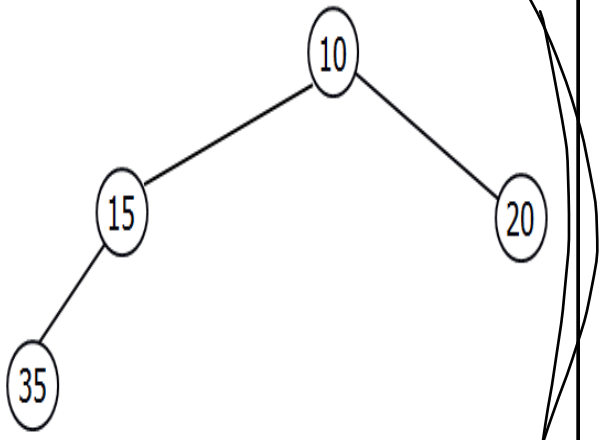
완전이진트리



(a)



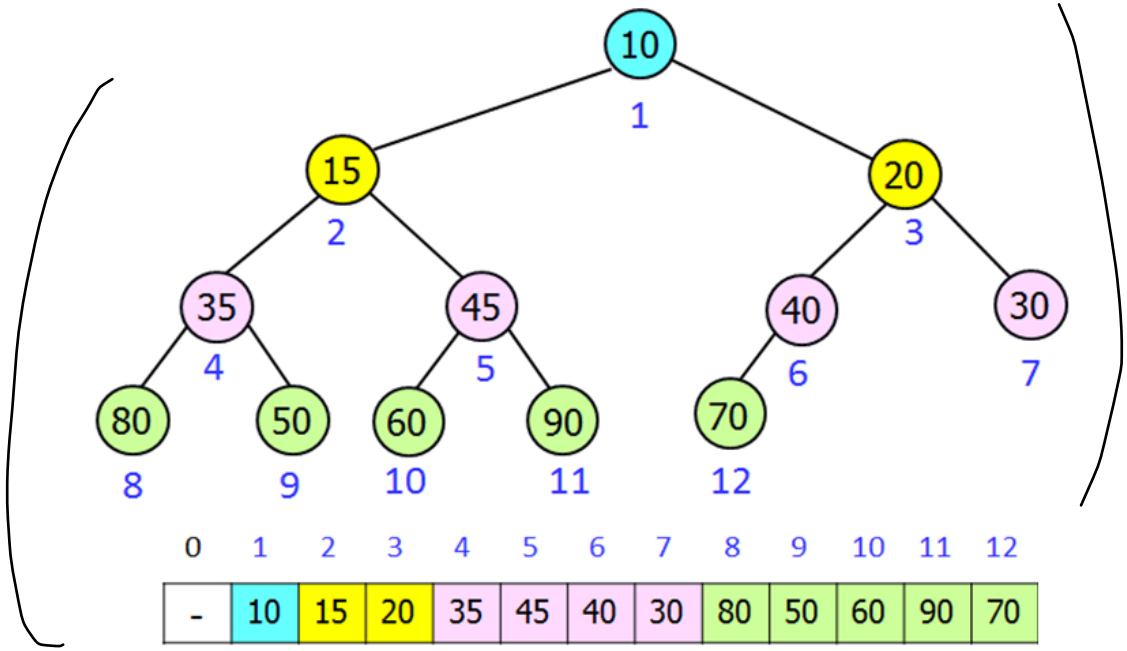
(b)



(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.

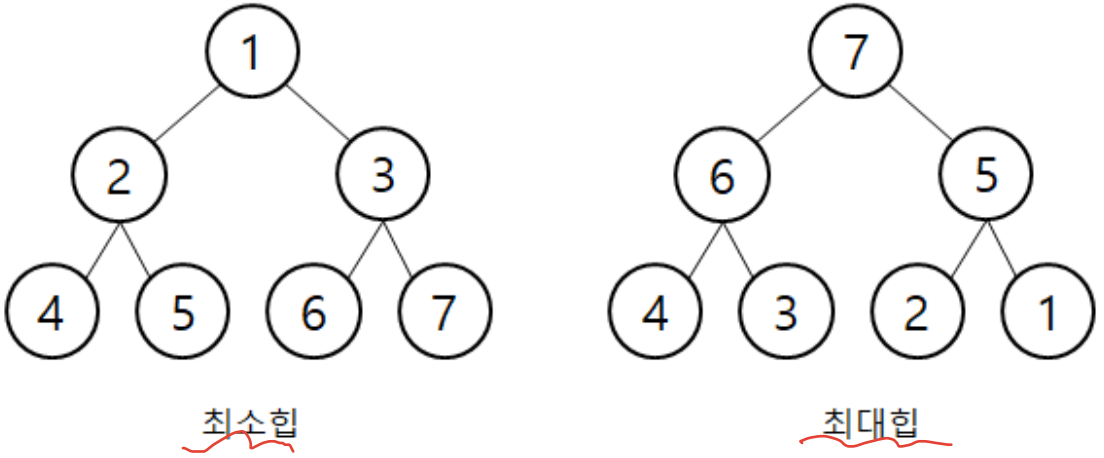
CC BY NC ND

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**



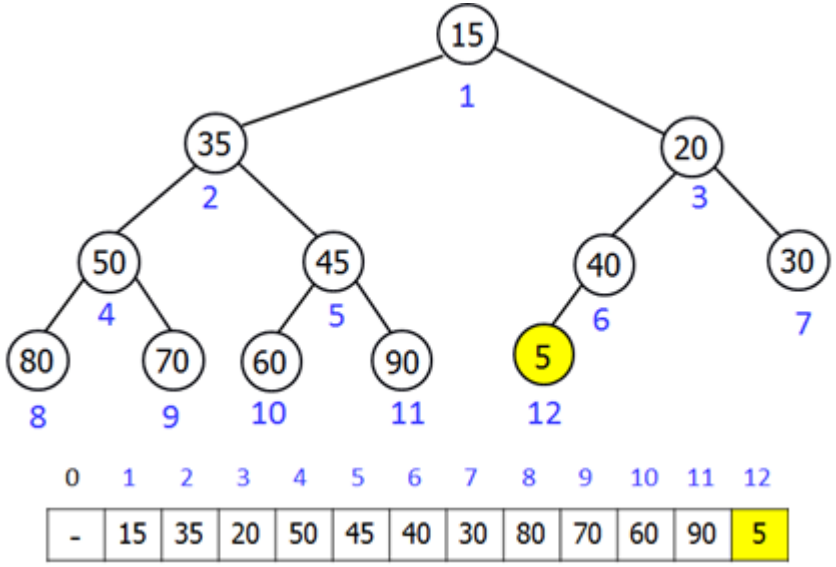
최소 힙 (2/11)

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

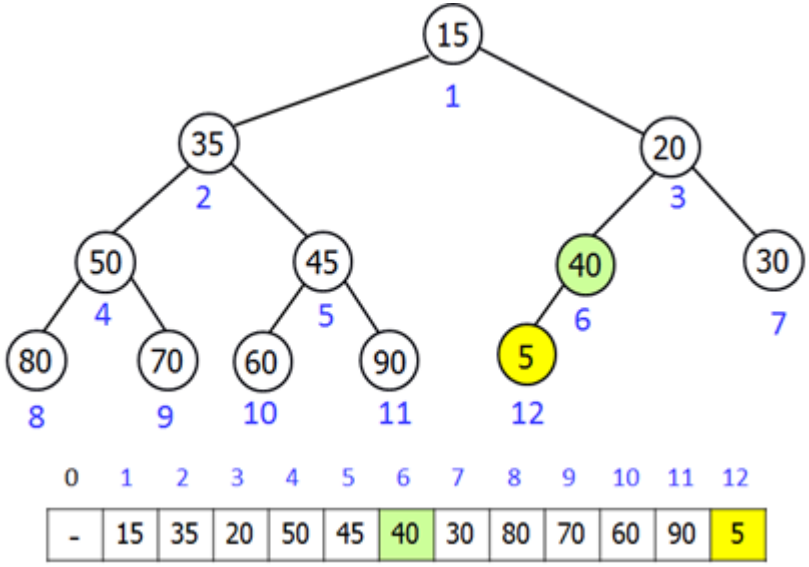
최소 힙 (3/11)

- **최소 힙:** 삽입 연산

최소힙에 5를 삽입하는 과정



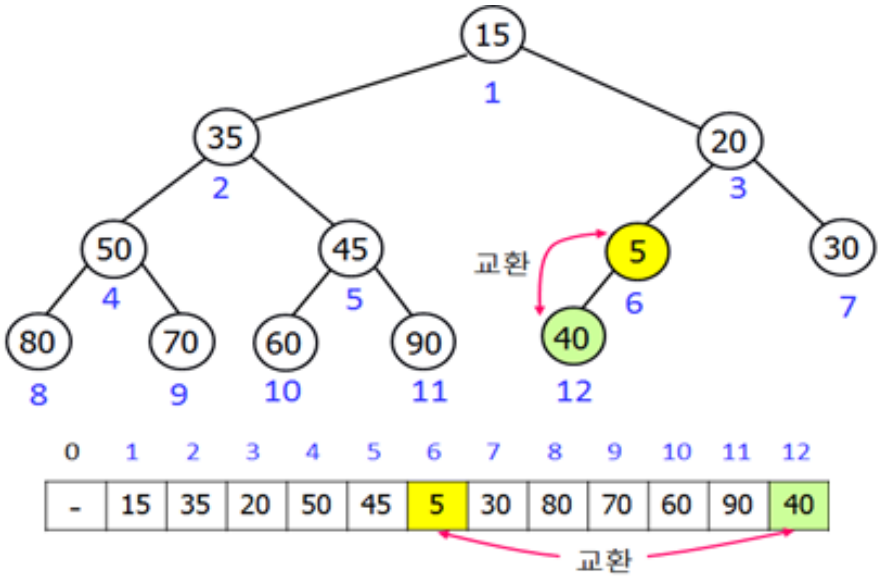
(a) 5를 마지막 항목(90) 다음에 저장



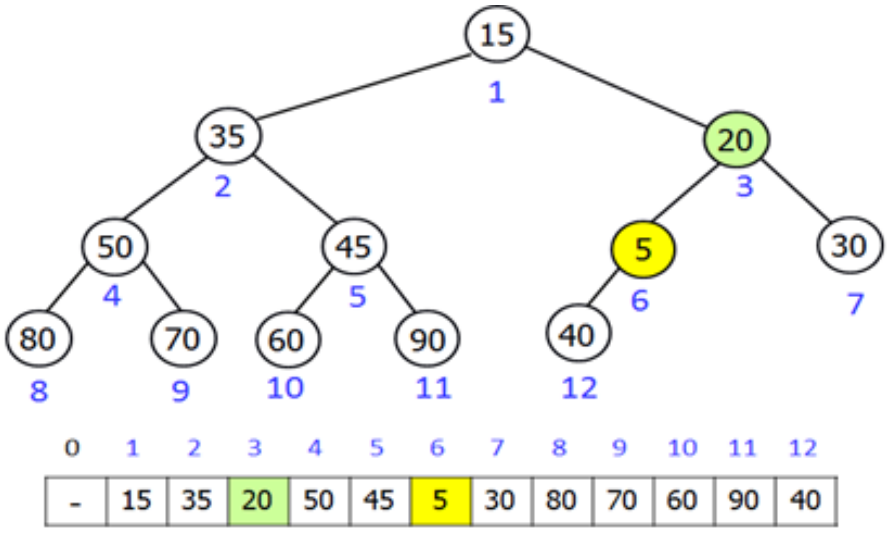
(b) a[12]의 5와 부모 a[6]의 40 비교

최소 힙 (4/11)

- **최소 힙:** 삽입 연산



(c) 5와 40을 교환

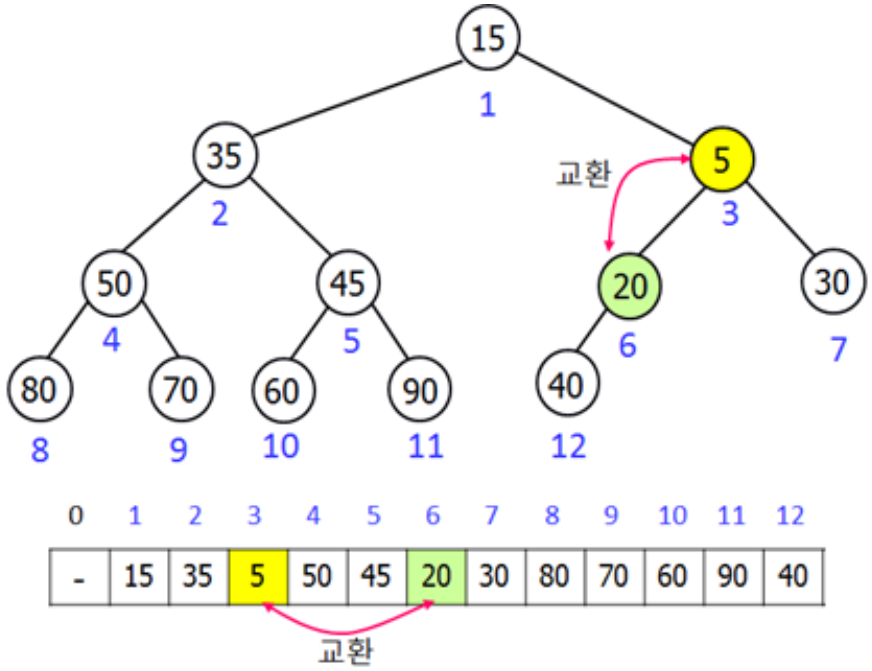


(d) a[6]의 5와 부모 a[3]의 20 비교

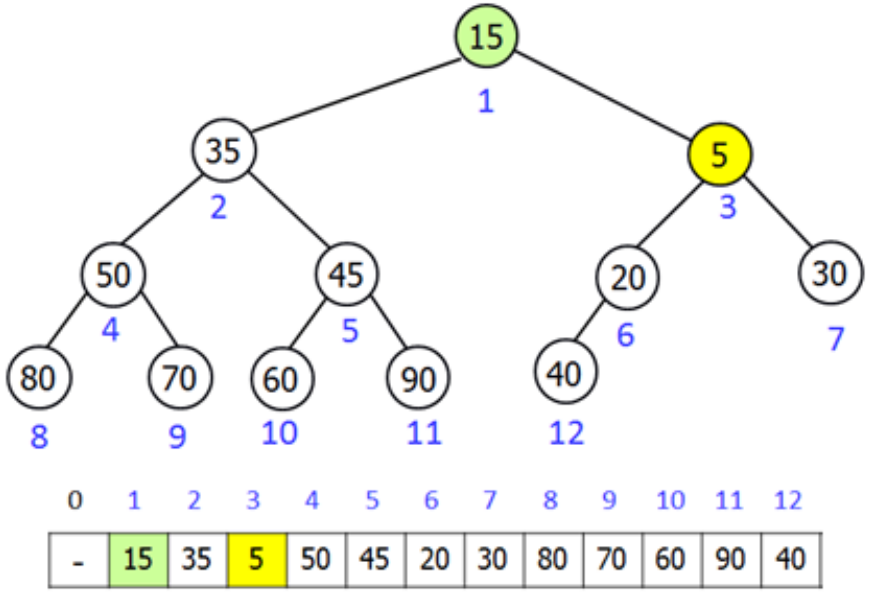
index / 2

최소 힙 (5/11)

- **최소 힙:** 삽입 연산



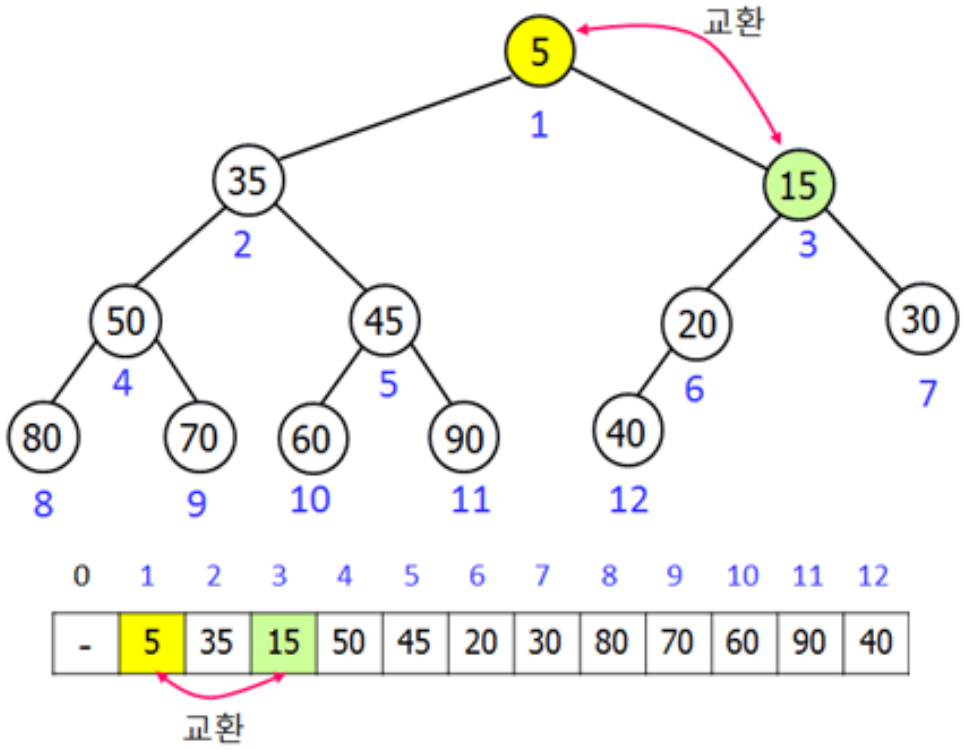
(e) 5와 20을 교환



(f) a[3]의 5와 부모 a[1]의 15 비교

최소 힙 (6/11)

- **최소 힙:** 삽입 연산



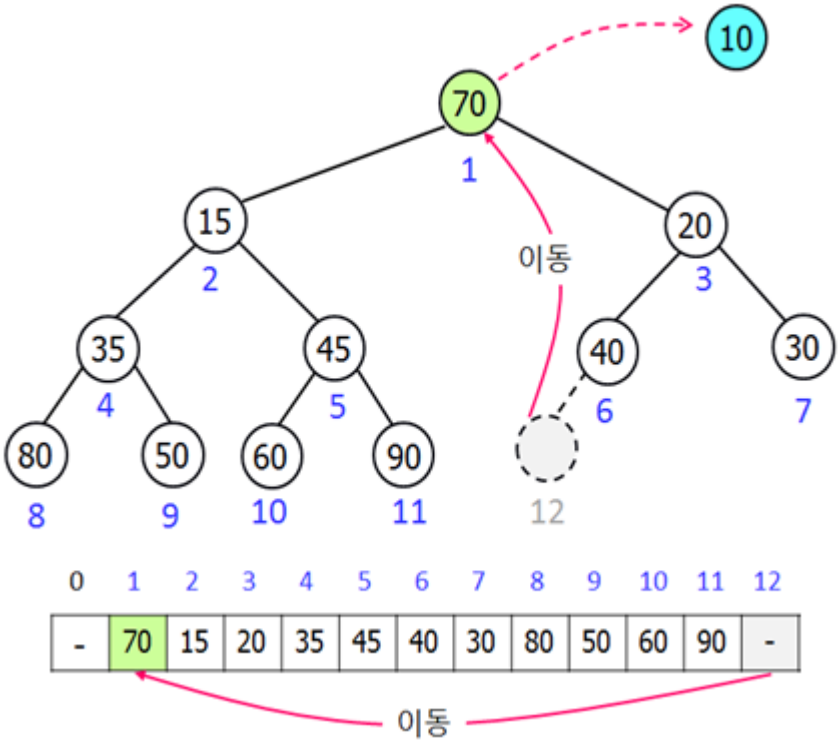
(g) 5와 15를 교환

최소 힙 (7/11)

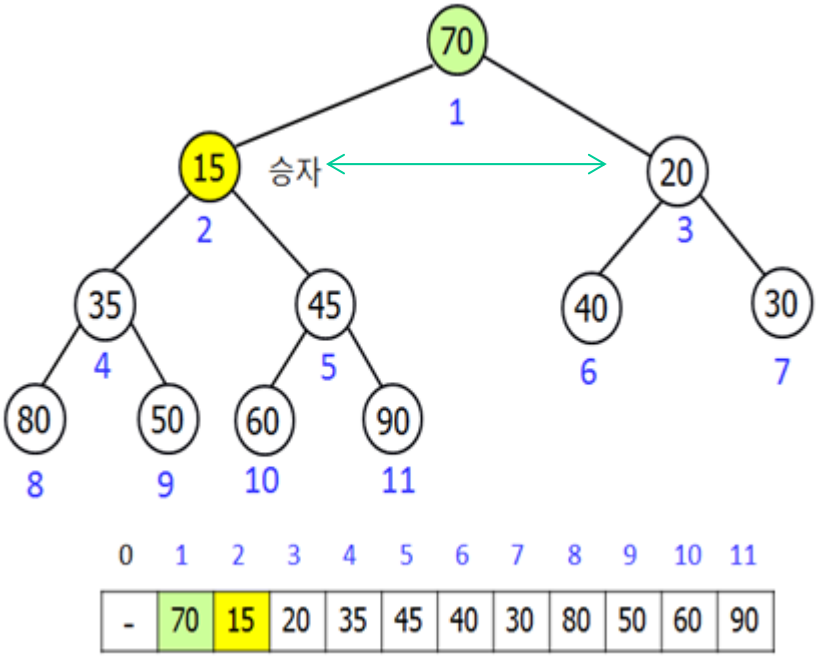
- **최소 힙: 삭제 연산**
  - **루트의 키를 삭제**
    1. **힙의 가장 마지막 노드,** 즉, 리스트의 가장 마지막 항목을 루트로 옮기고,
    2. **힙 크기를 1 감소** 시킨다.
    3. 루트로부터 자식들 중에서 작은 값을 가진 자식 (승자)과 키를 비교하여 힙 속성이 만족될 때까지 키를 교환하며 이파리 방향으로 진행
  - **[3]의 과정은 루트로부터 아래로 내려가며 진행되므로 downheap 이라 부름**

최소 힙 (8/11)

- **최소 힙:** 삭제 연산



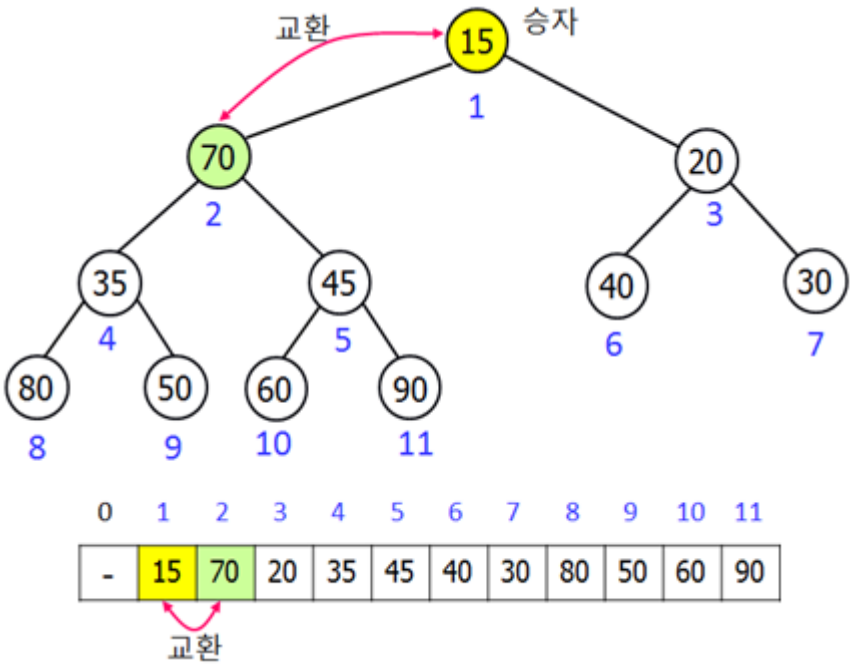
(a) 마지막 항목을 루트로 이동



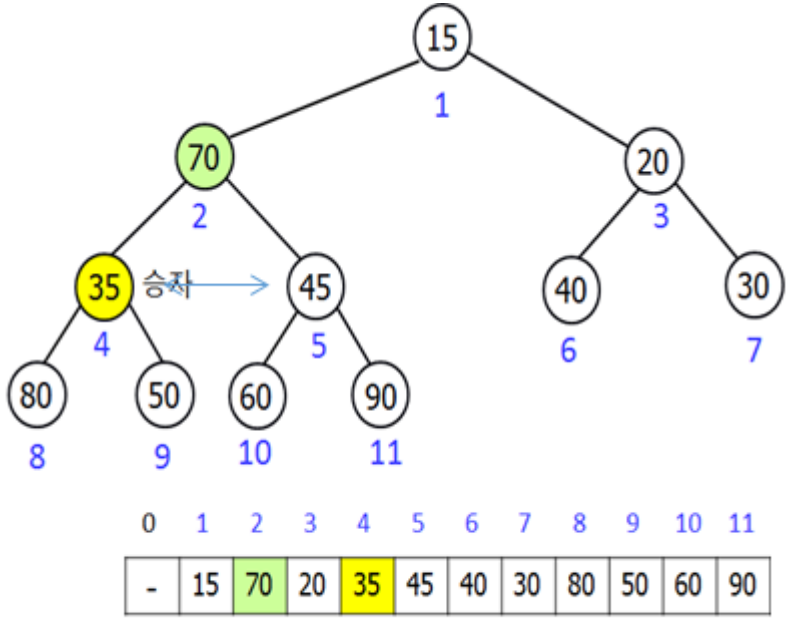
(b) 15와 20 중에 15가 승자

최소 힙 (9/11)

- **최소 힙:** 삭제 연산



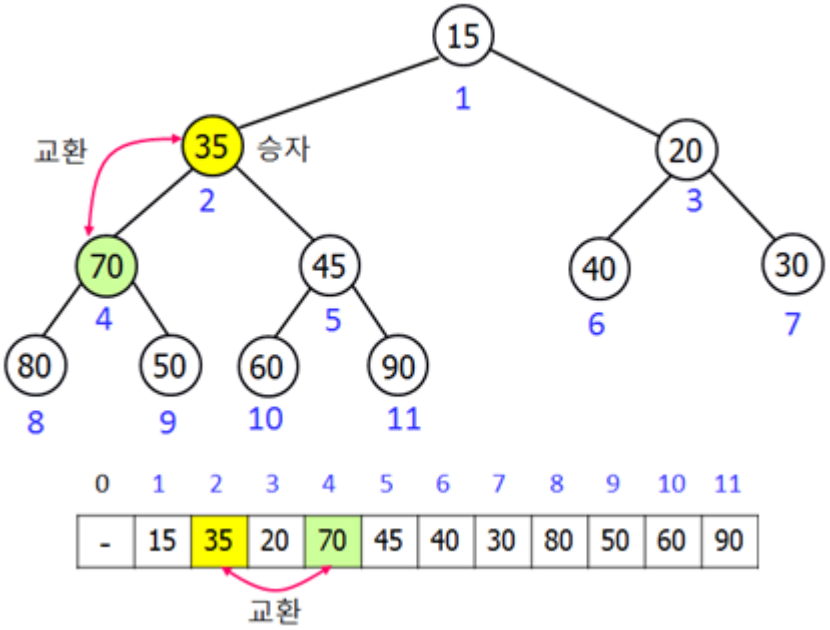
(c) 승자인 15와 루트를 교환



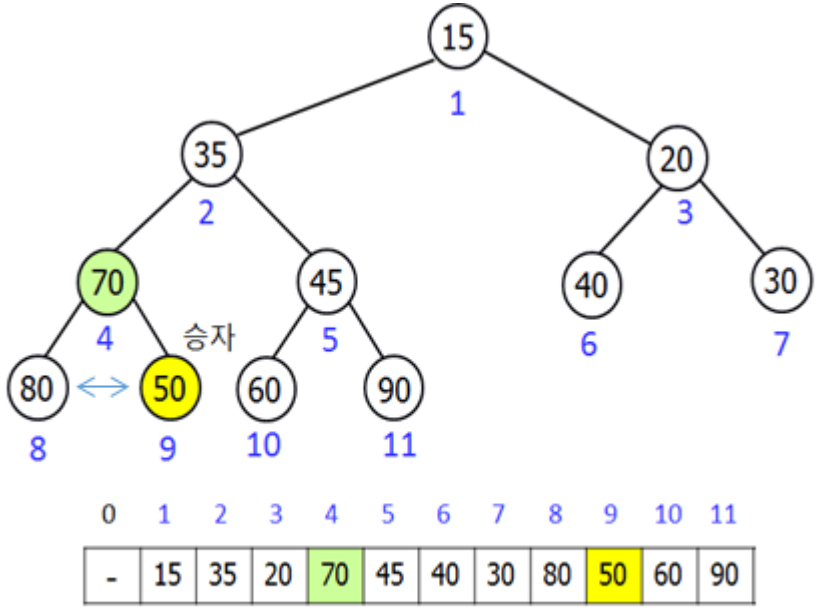
(d) 35와 45 중에 35가 승자

최소 힙 (10/11)

- **최소 힙:** 삭제 연산



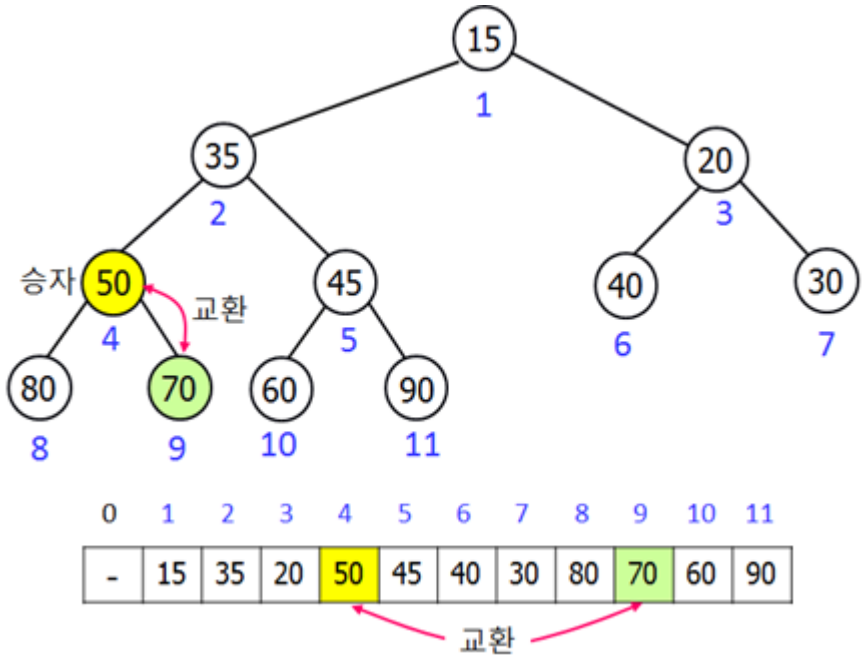
(e) 승자인 35와 70를 교환



(f) 80과 50 중에 50이 승자

최소 힙 (11/11)

- **최소 힙:** 삭제 연산



(g) 승자인 50과 70을 교환

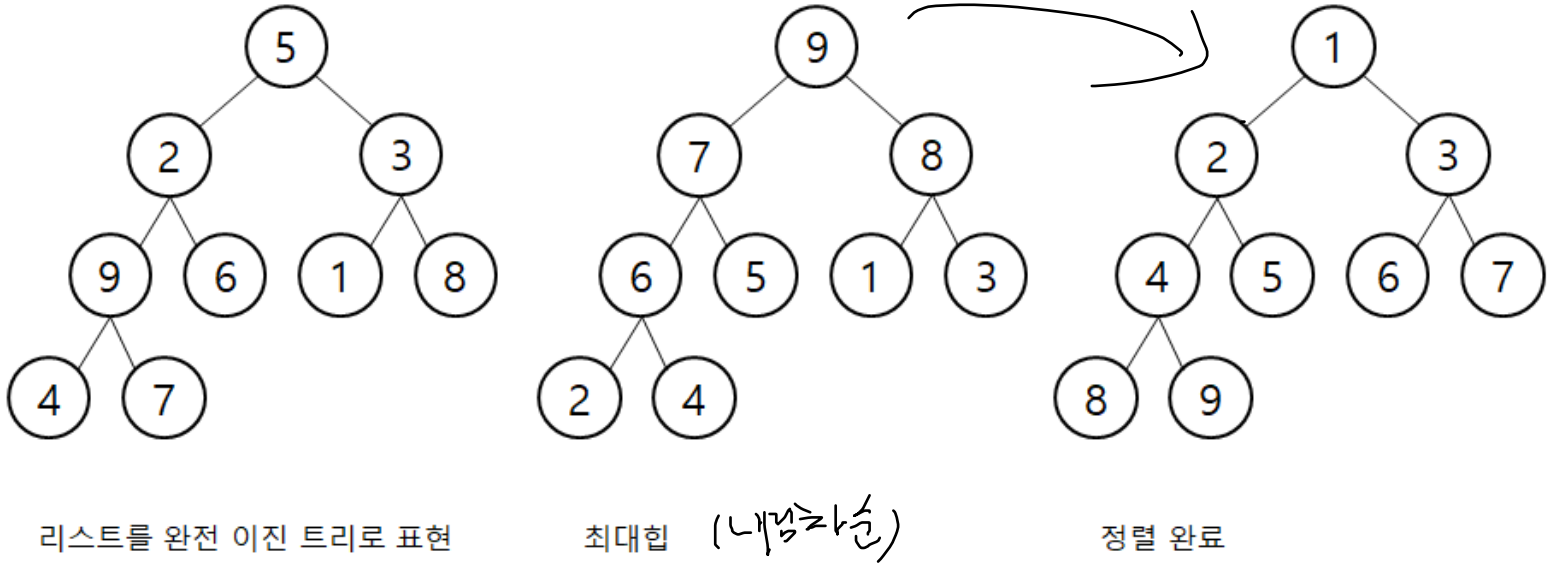
CC BY NC ND

우선 순위 큐와 힙

힙 정렬

힙 정렬 (1/1)

- **힙 정렬(Heap Sort)** $(n \log n)$
  - 힙을 이용한 정렬 과정



리스트를 완전 이진 트리로 표현 　 최대힙 (내림차순) 　 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: $\Theta(n)$

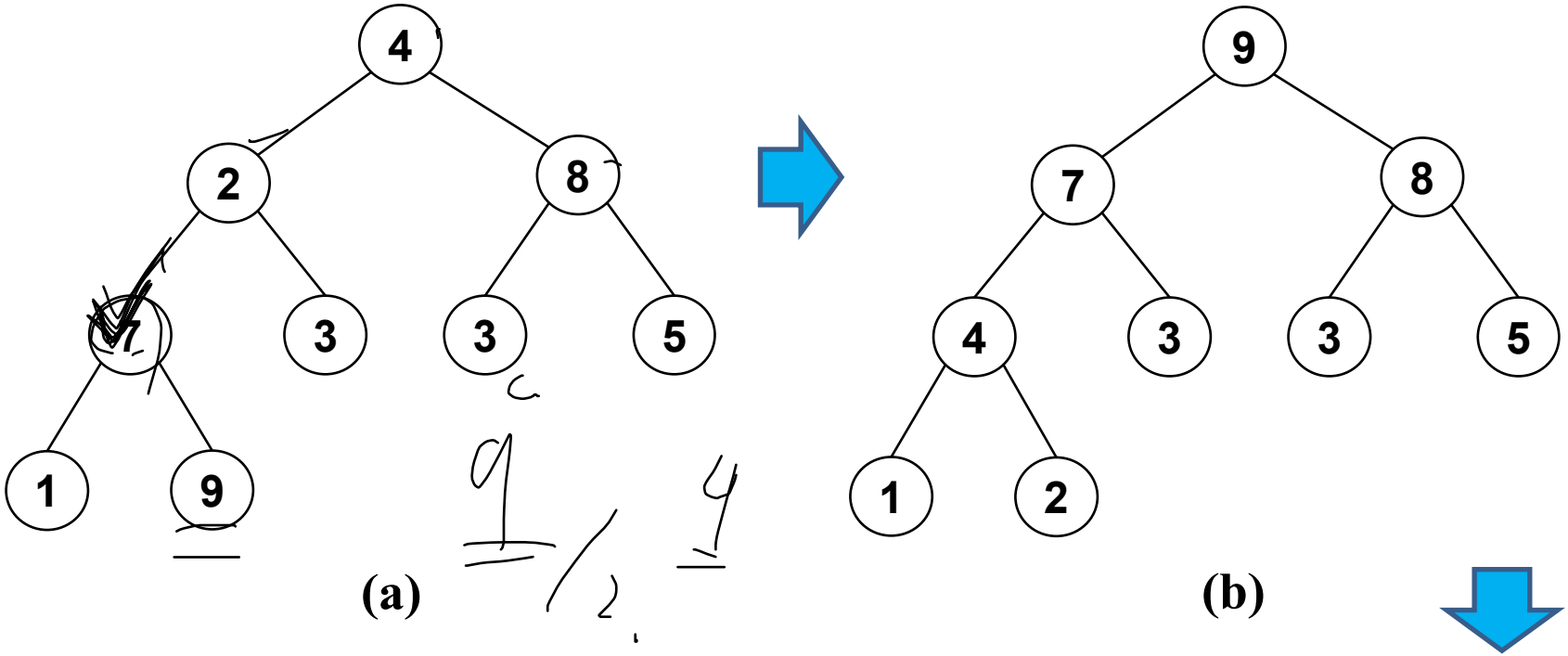
for loop: $O(n\log n)$

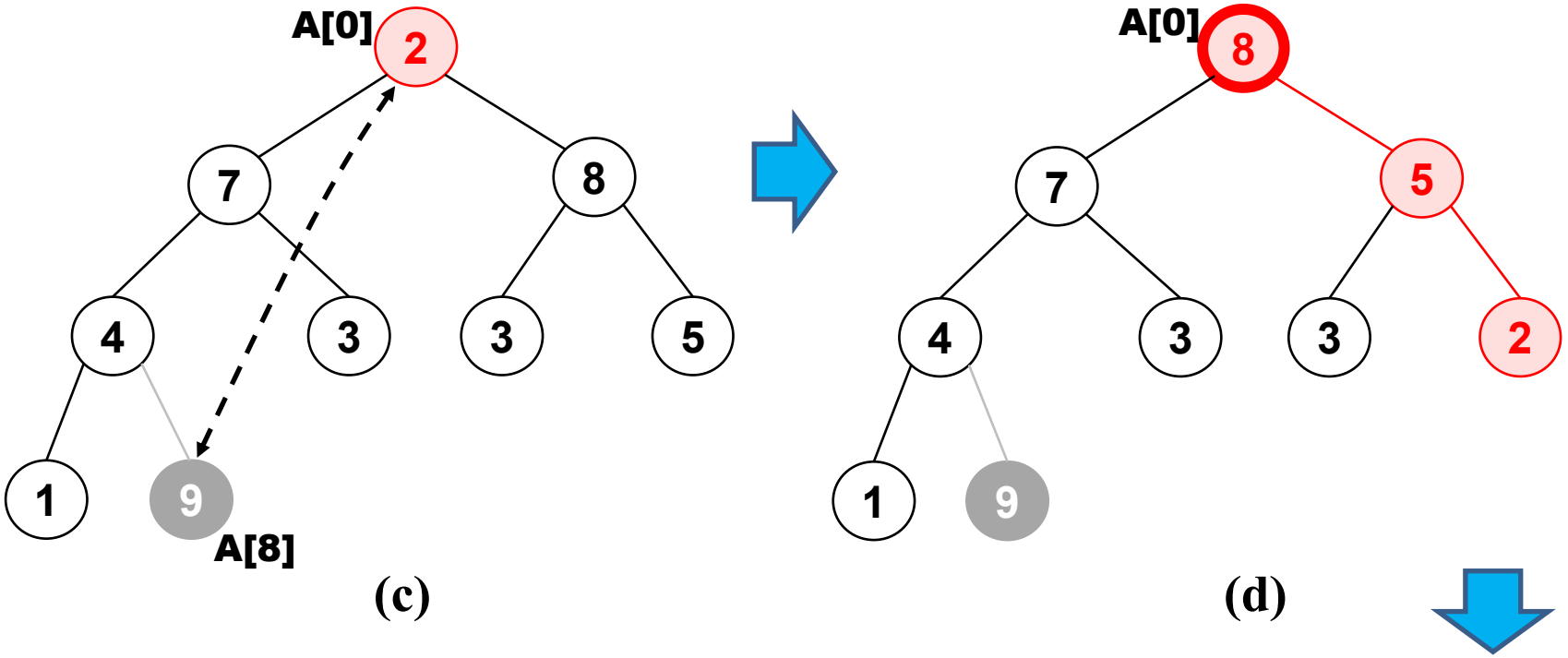
그냥 말하면: $\Omega(n) \sim O(n\log n)$

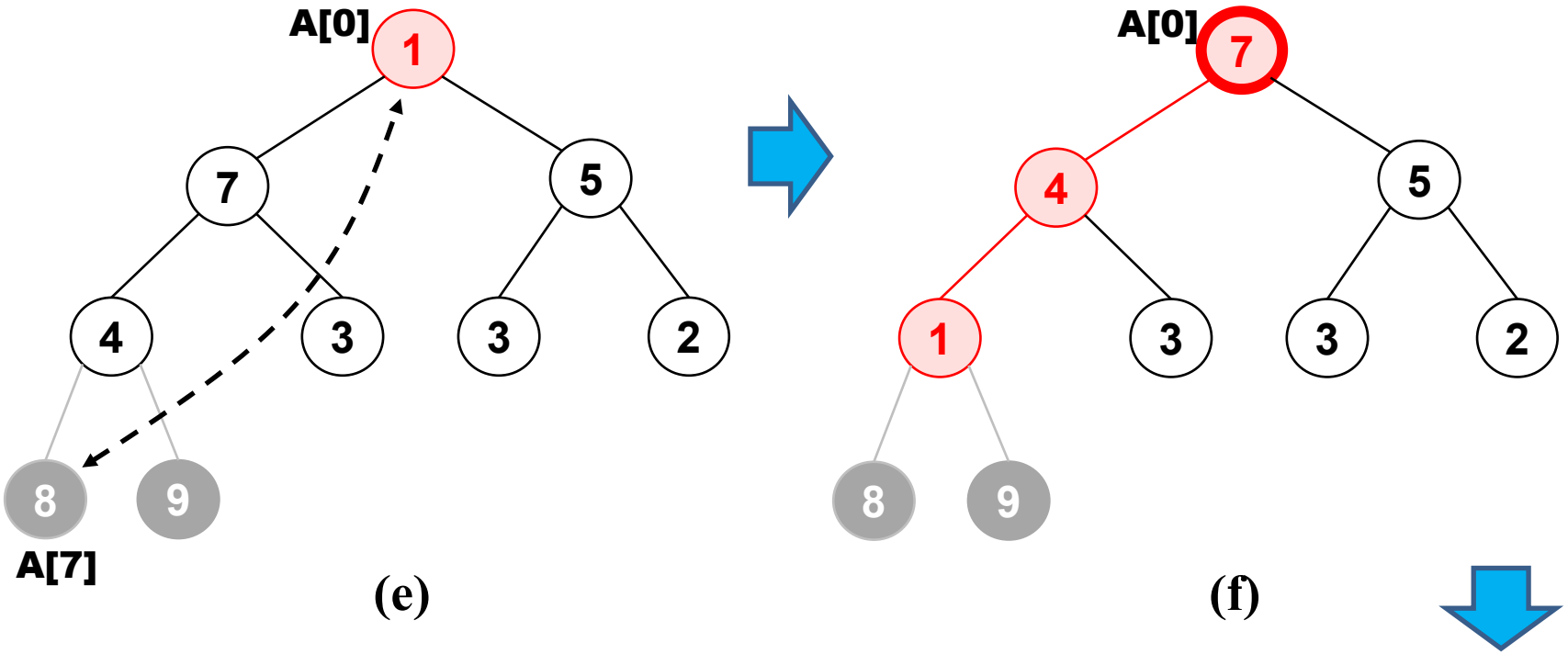
Worst: $\Theta(n\log n)$

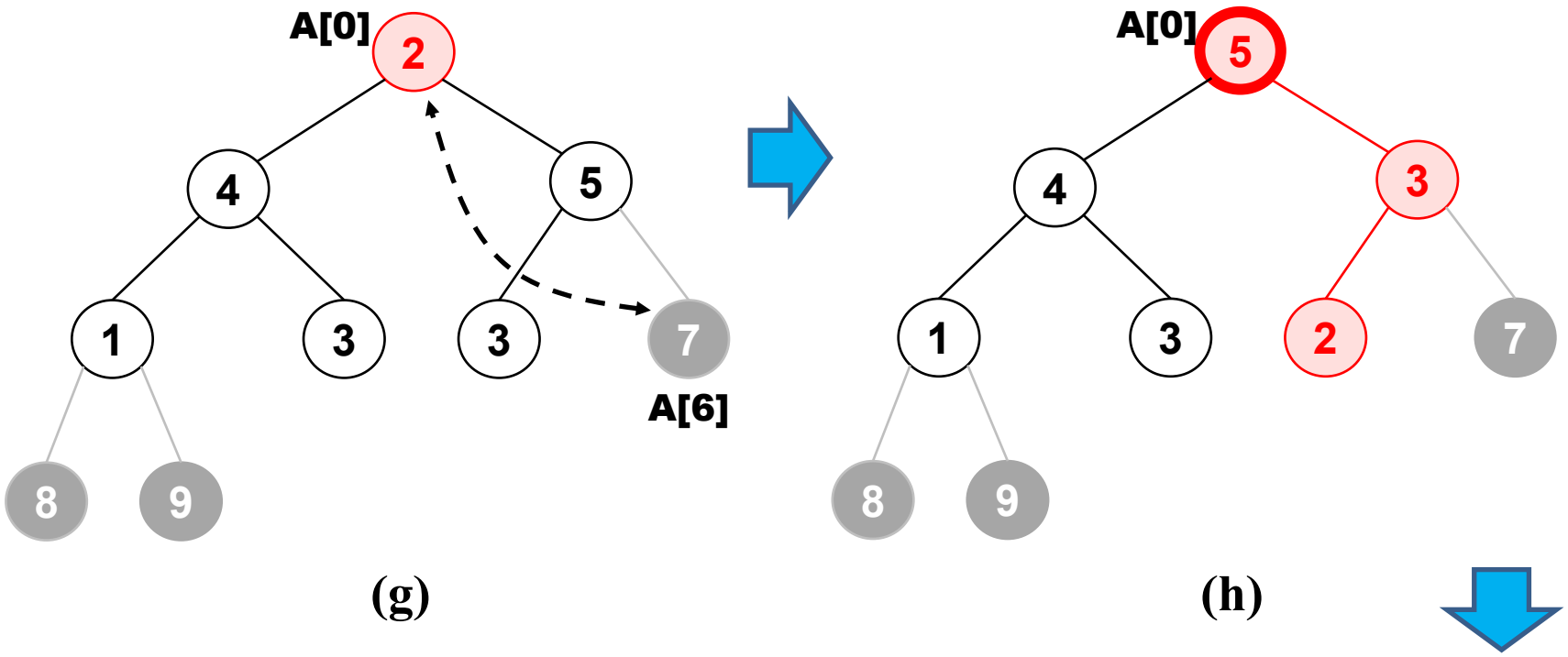
Best: $\Theta(n)$

6. 힙 정렬Heapsort

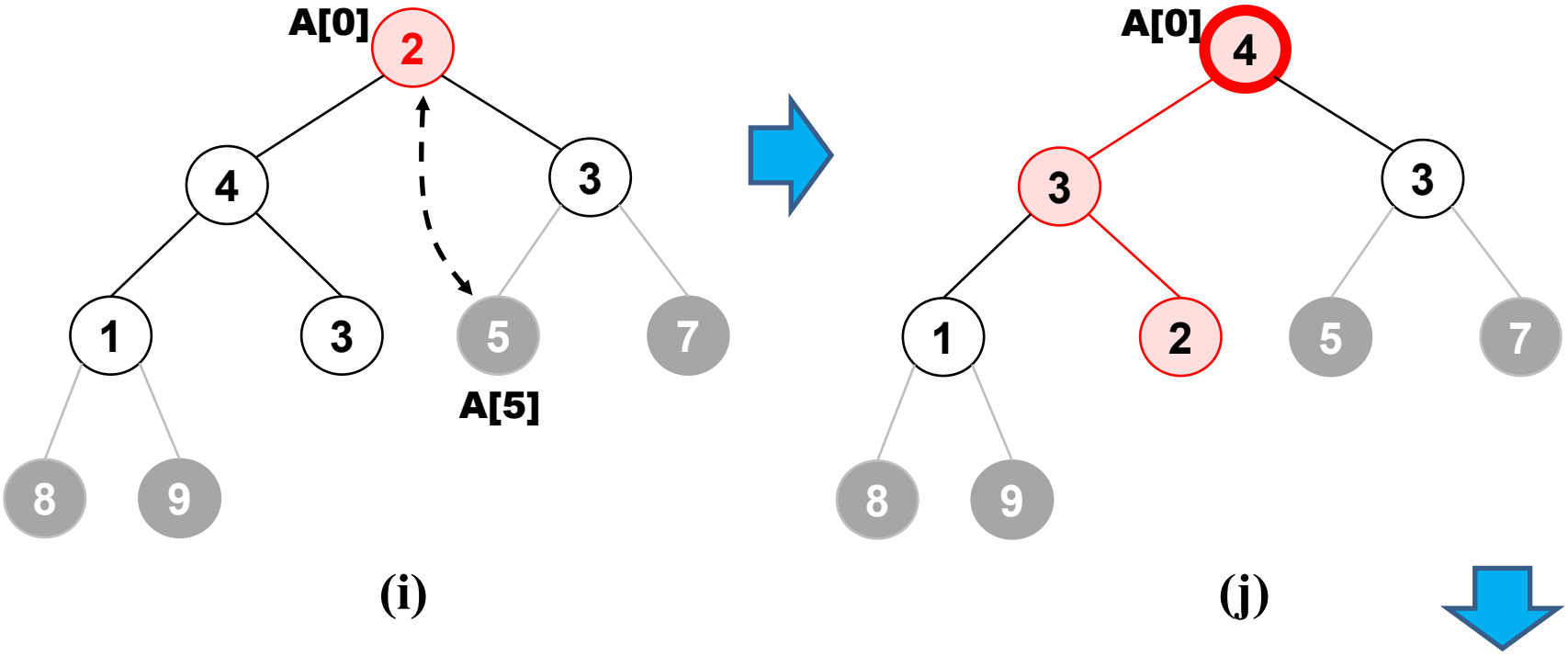


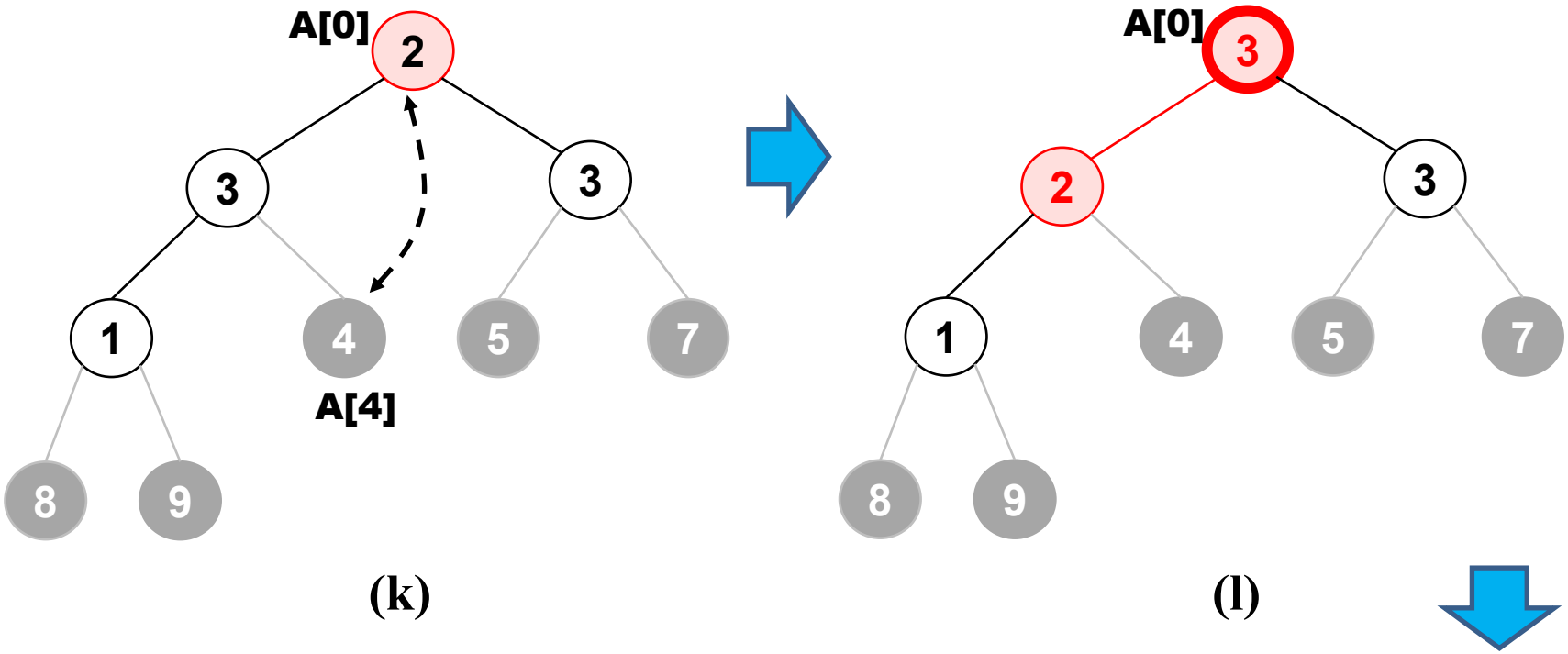

A[0]
2
7
8
4
3
3
5
1
9
A[8]
(c)
A[0]
8
7
5
4
3
3
2
1
9
(d)

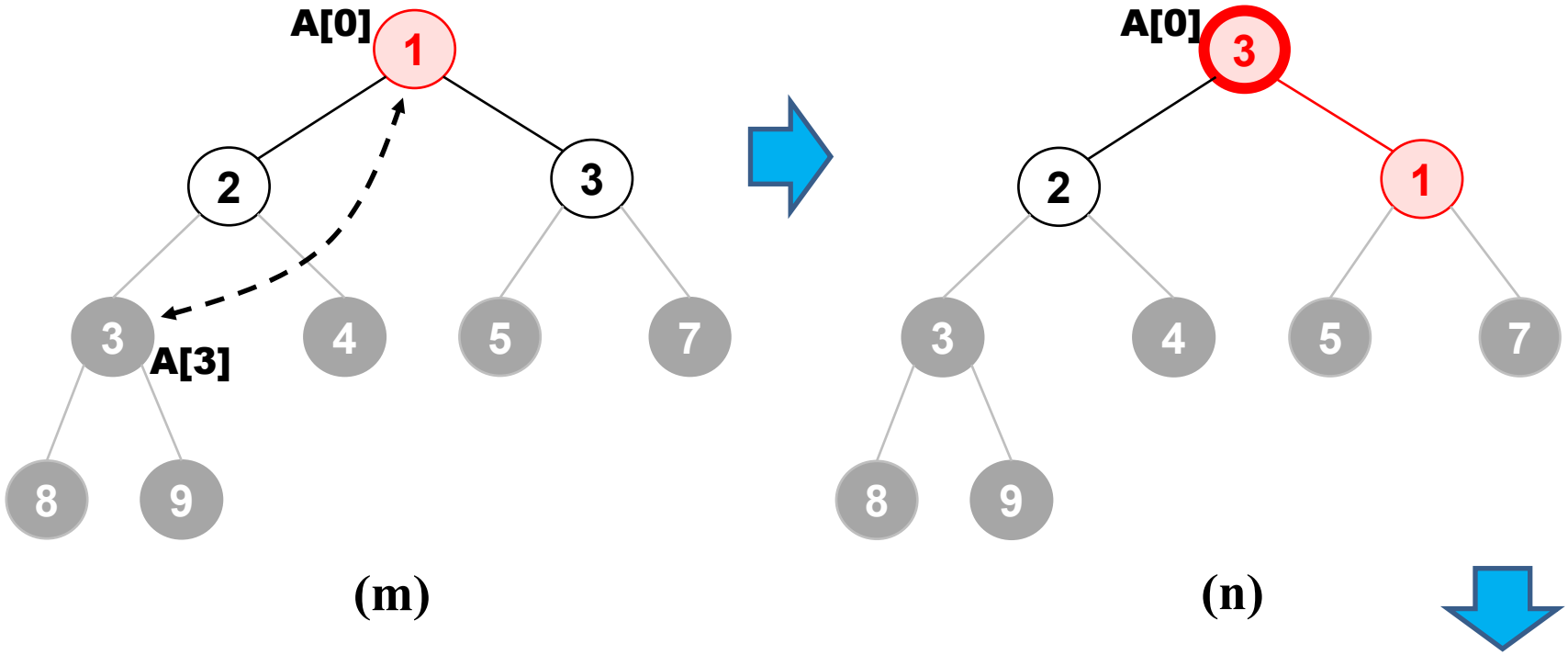

A[0]
1
7
5
4
3
3
2
8
9
A[7]
(e)
A[0]
7
4
5
1
3
3
2
8
9
(f)

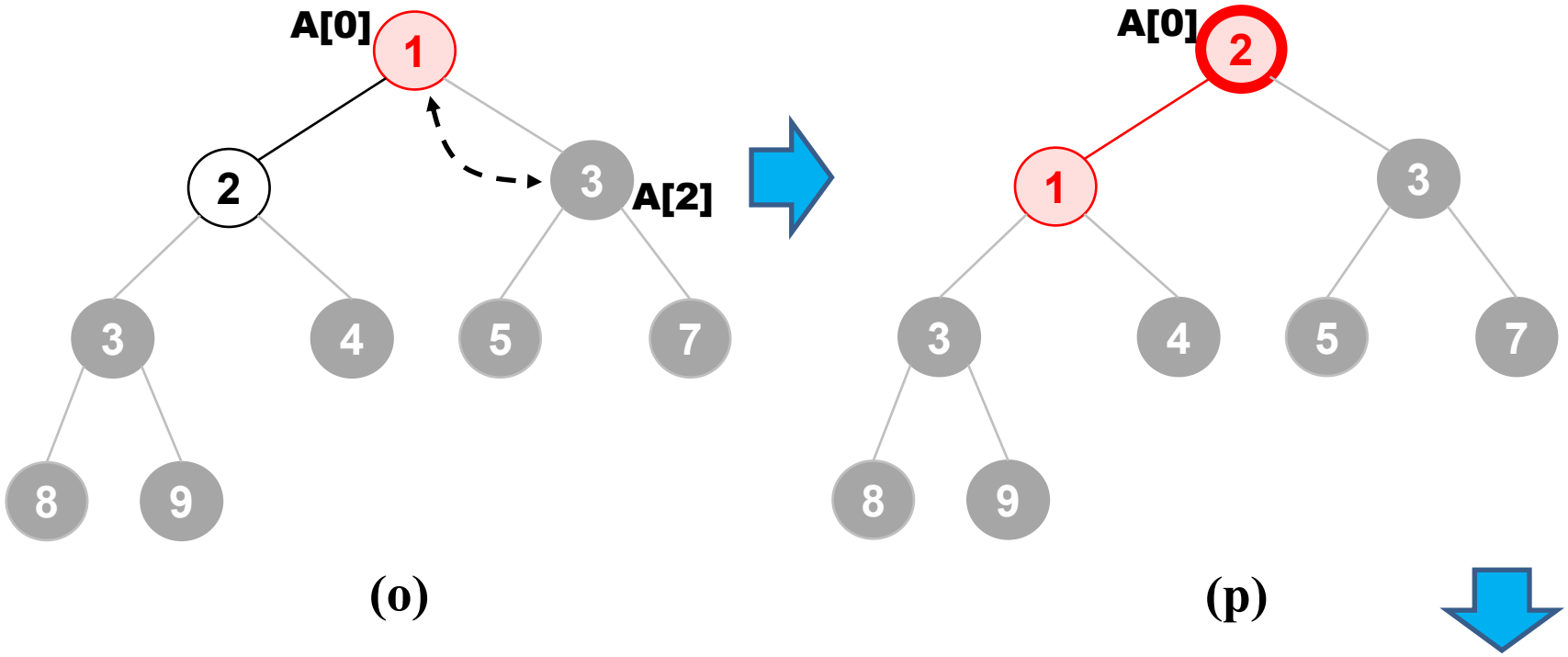


(g) (h)

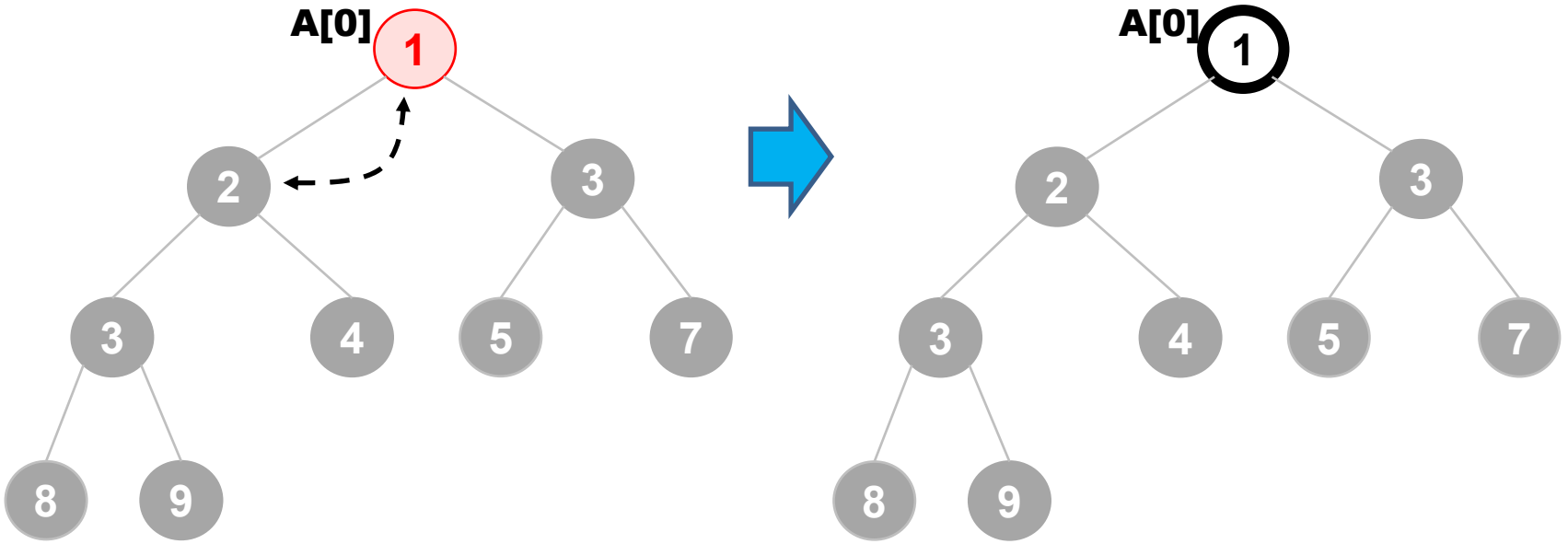

A[0]
2
4
3
1
3
5
7
A[5]
8
9
(i)
A[0]
4
3
3
1
2
5
7
8
9
(j)


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)

참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, “C++로 구현하는 자료구조와 알고리즘”, 한티에듀, 2020.

[2] “프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문”, 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

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

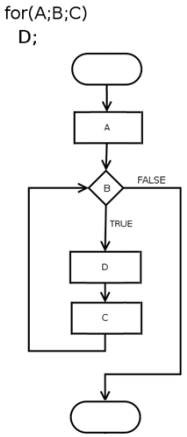
[4] “이것이 취업을 위한 코딩 테스트다 with 파이썬”, 나동빈, 한빛미디어, 2020.

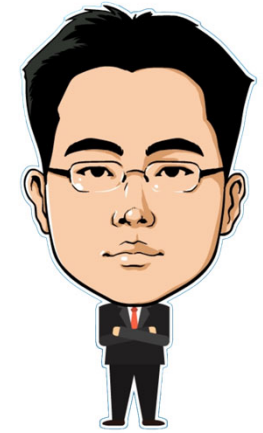
[5] 문병로, “IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법”(개정판), 개정판, 한빛아카데미, 2018.

[6] Richard E. Neapolitan, 도경구 역, “알고리즘 기초”, 도서출판 홍릉, 2017.

[7] 주우석, “IT CookBook, C・C++ 로 배우는 자료구조론”, 한빛아카데미, 2019.

[8] 이지영, “C 로 배우는 쉬운 자료구조”, 한빛아카데미, 2022.





이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.