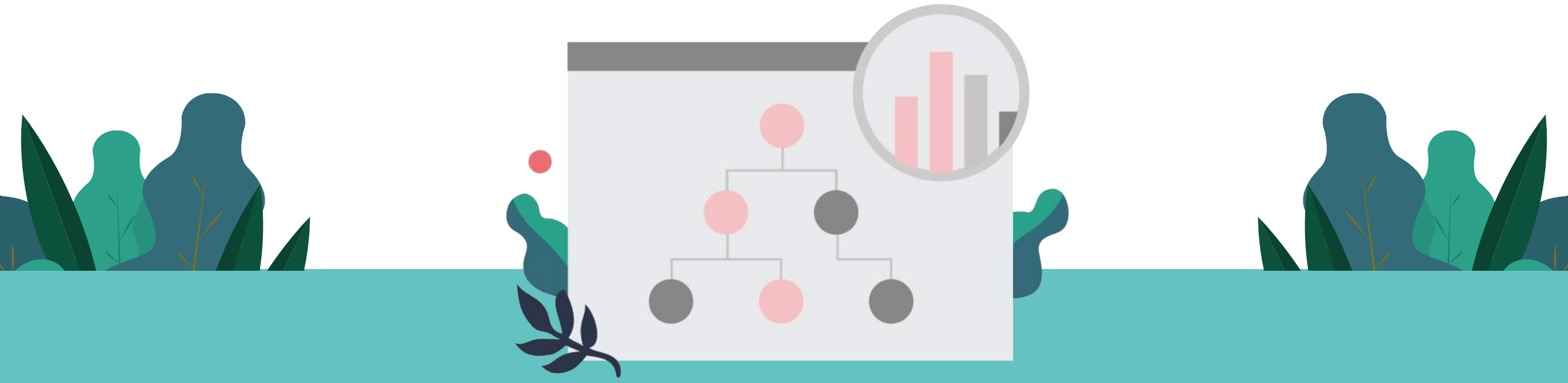
/* 데이터 구조 및 알고리즘 */

신현규 강사, 화/목 20:00

힙



/* elice */

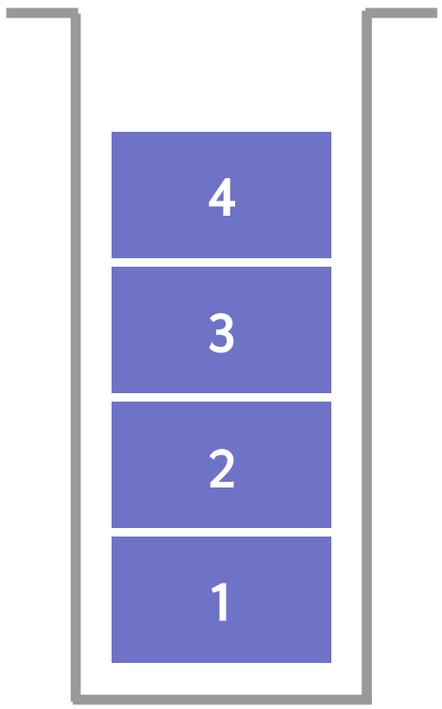
주차별 커리큘럼

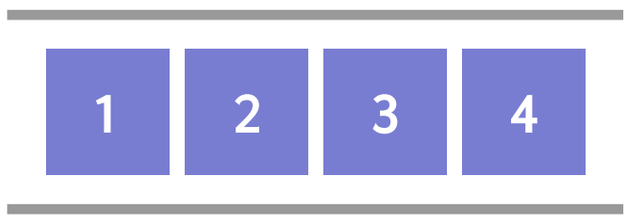
1주차	과정 소개, 배열, 연결리스트, 클래스
2주차	스택, 큐, 해싱
3주차	시간복잡도
4주차	트리, 트리순회, 재귀호출
5주차	**힙**
6주차	그래프 소개, DFS
7주차	그래프 심화, BFS
8주차	강의 요약, 알고리즘 과정 소개

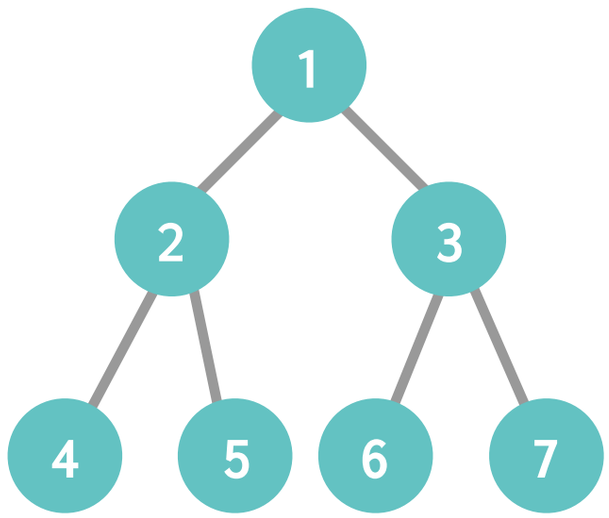
컴퓨터를 이용한 문제 해결 과정

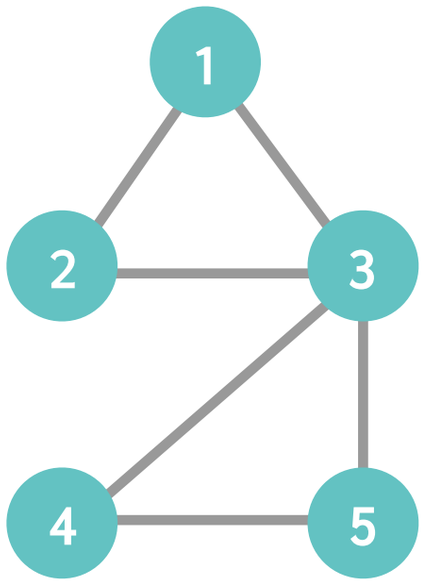
1. 문제를 정확히 이해한다
2. 문제를 해결하는 알고리즘을 개발한다
3. 알고리즘이 **문제를 해결한다는 것을 증명한다**
4. 알고리즘이 **제한시간 내에 동작한다는 것을 보인다**
5. 알고리즘을 코드로 작성한다
6. 제출 후 만점을 받고 매우 기뻐한다

대표적인 자료구조









스택 (Stack)

Last In First Out

큐 (Queue)

First In First Out

트리 (Tree)

그래프 (Graph)

재귀함수의 올바른 디자인 및 해석

재귀함수를 디자인하기 위해서는 다음 세 가지 단계를 명심하자

1. 함수의 정의를 명확히 한다.

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

[예제 3] 거듭제곱 구하기

m^n을 구하시오

(단, $1 \leq n \leq 1{,}000{,}000{,}000$)

입력의 예

```
3 4
```

출력의 예

```
81
```

재귀함수의 올바른 디자인 및 해석

재귀함수를 디자인하기 위해서는 다음 세 가지 단계를 명심하자

1. 함수의 정의를 명확히 한다.

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

1. 함수의 정의를 명확히 한다.

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

1. 함수의 정의를 명확히 한다.

getPower(m, n) : m^n 을 반환하는 함수

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

getPower(m, 0) = 1

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

getPower(m, n) = m x getPower(m, n-1)

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

getPower(m, 0) = 1

getPower(m, n) = m x getPower(m, n-1)

정의

점화식

기저조건
(Base condition)

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :
    if n == 0 :
        return 1
    else :
        return m * getPower(m, n-1)
```

[예제 3] 거듭제곱 구하기

$m^n = m \times m \times \ldots \times m$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :
    if n == 0 :
        return 1
    else :
        return m * getPower(m, n-1)
```

O(n)

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$m^n = (m^{(n/2)})^2$ n이 짝수일 경우

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$m^n = (m^{(n/2)})^2 \qquad n이 짝수일 경우$$

$$(m^{n-1}) \times m \qquad n이 홀수일 경우$$

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

getPower(m, n) = n이 짝수

getPower(m, n) = n이 홀수

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

getPower(m, n) = (getPower(m, n//2))^2 n이 짝수

getPower(m, n) = m x getPower(m, n-1) n이 홀수

[예제 3] 거듭제곱 구하기

$$m^n = m \times m \times \ldots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :
    if n == 0 :
        return 1
    elif n % 2 == 0 :
        temp = getPower(m, n//2)
        return temp * temp
    else :
        return getPower(m, n-1) * m
```

[예제 3] 거듭제곱 구하기

$m^n = m \times m \times \ldots \times m$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :
    if n == 0 :
        return 1
    elif n % 2 == 0 :
        temp = getPower(m, n//2)
        return temp * temp
    else :
        return getPower(m, n-1) * m
```

O(log n)

[예제 1] 퀵정렬 구현하기

숫자 n개를 오름차순으로 정렬하시오

(단, $1 \leq n \leq 1,000,000$)

입력의 예

```
10 2 3 4 5 6 9 7 8 1
```

출력의 예

```
1 2 3 4 5 6 7 8 9 10
```

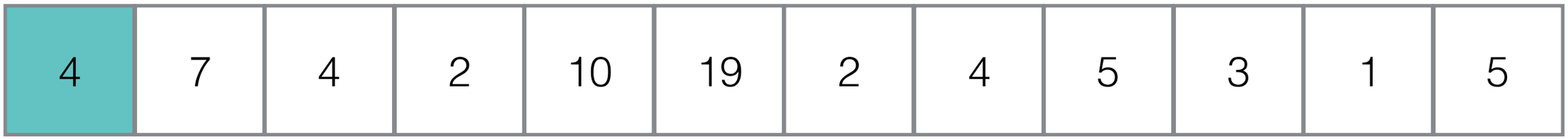
퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	7	4	2	10	19	2	4	5	3	1	5

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬



pivot

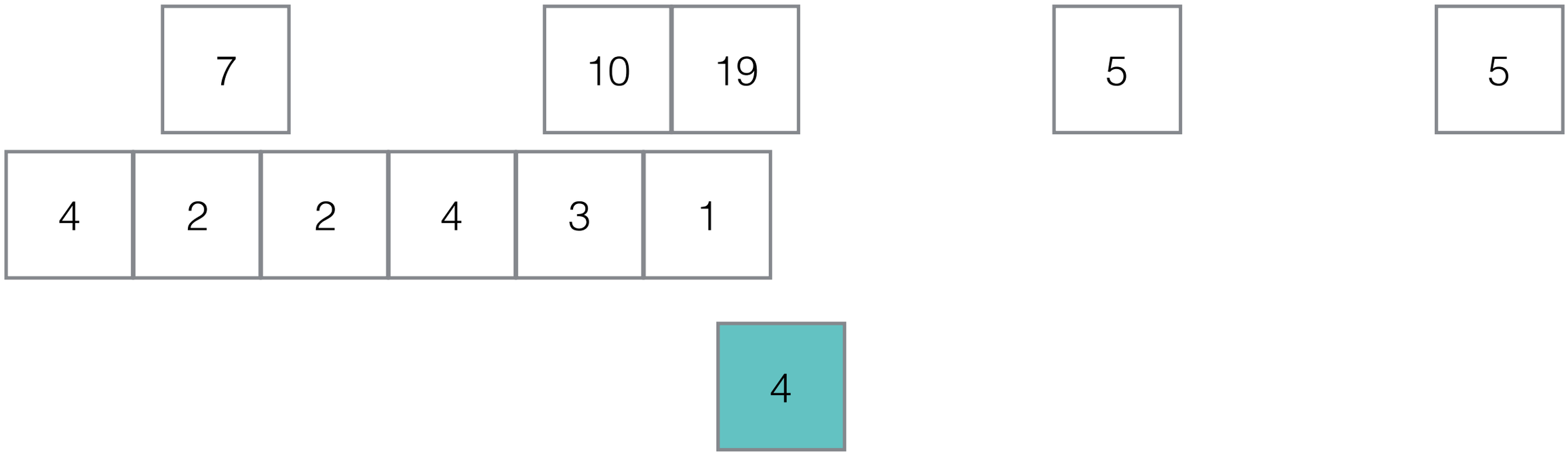
퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

7	4	2	10	19	2	4	5	3	1	5

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬



퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	2	2	4	3	1

7	10	19	5	5

4

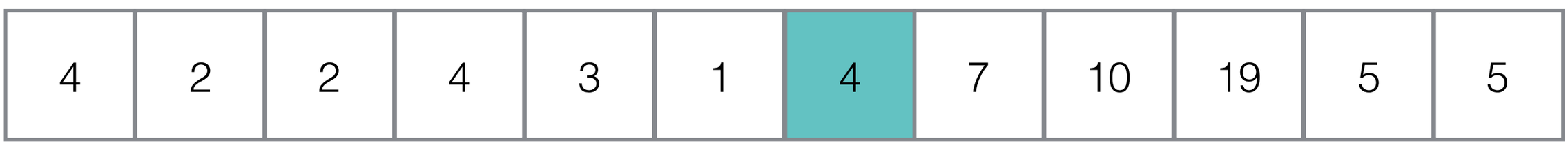
퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	2	2	4	3	1	4	7	10	19	5	5

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

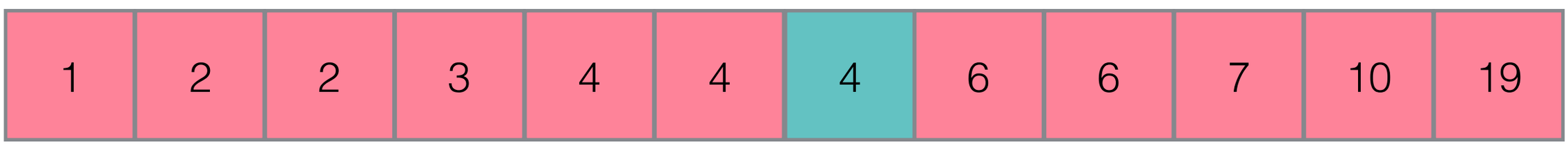


Quicksort!

Quicksort!

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬



Quicksort!

Quicksort!

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

1	2	2	3	4	4	4	6	6	7	10	19

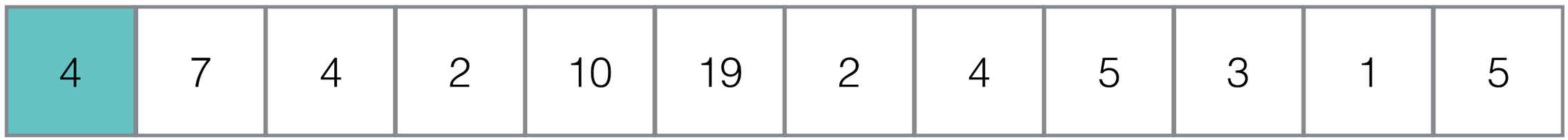
Quicksort!

Quicksort!

퀵정렬의 자세한 단계

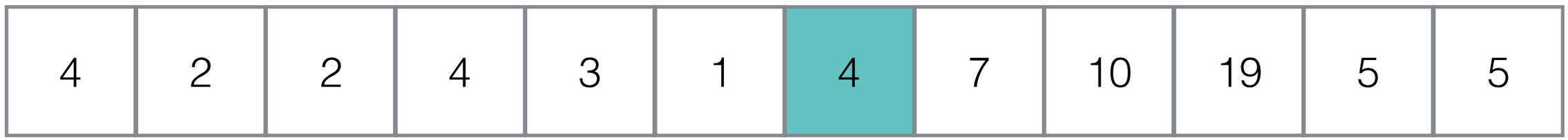
4	7	4	2	10	19	2	4	5	3	1	5

퀵정렬의 자세한 단계

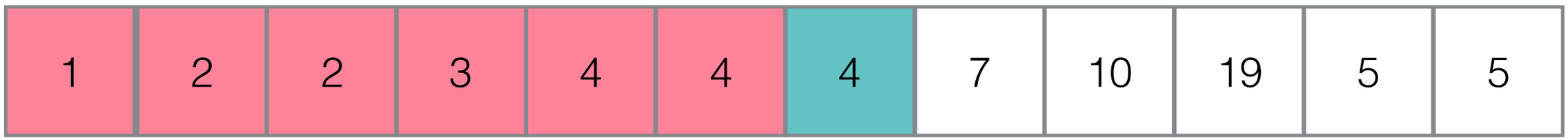


pivot

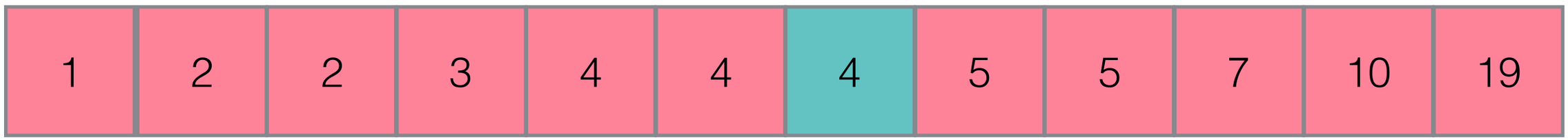
퀵정렬의 자세한 단계



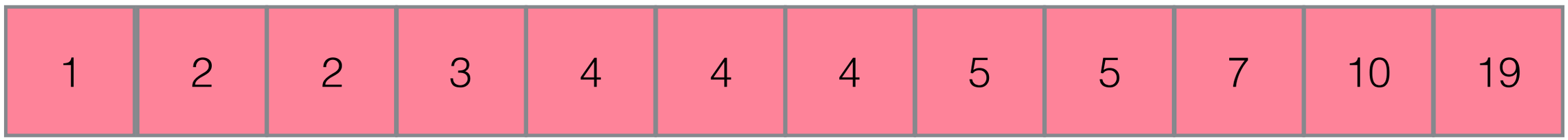
퀵정렬의 자세한 단계



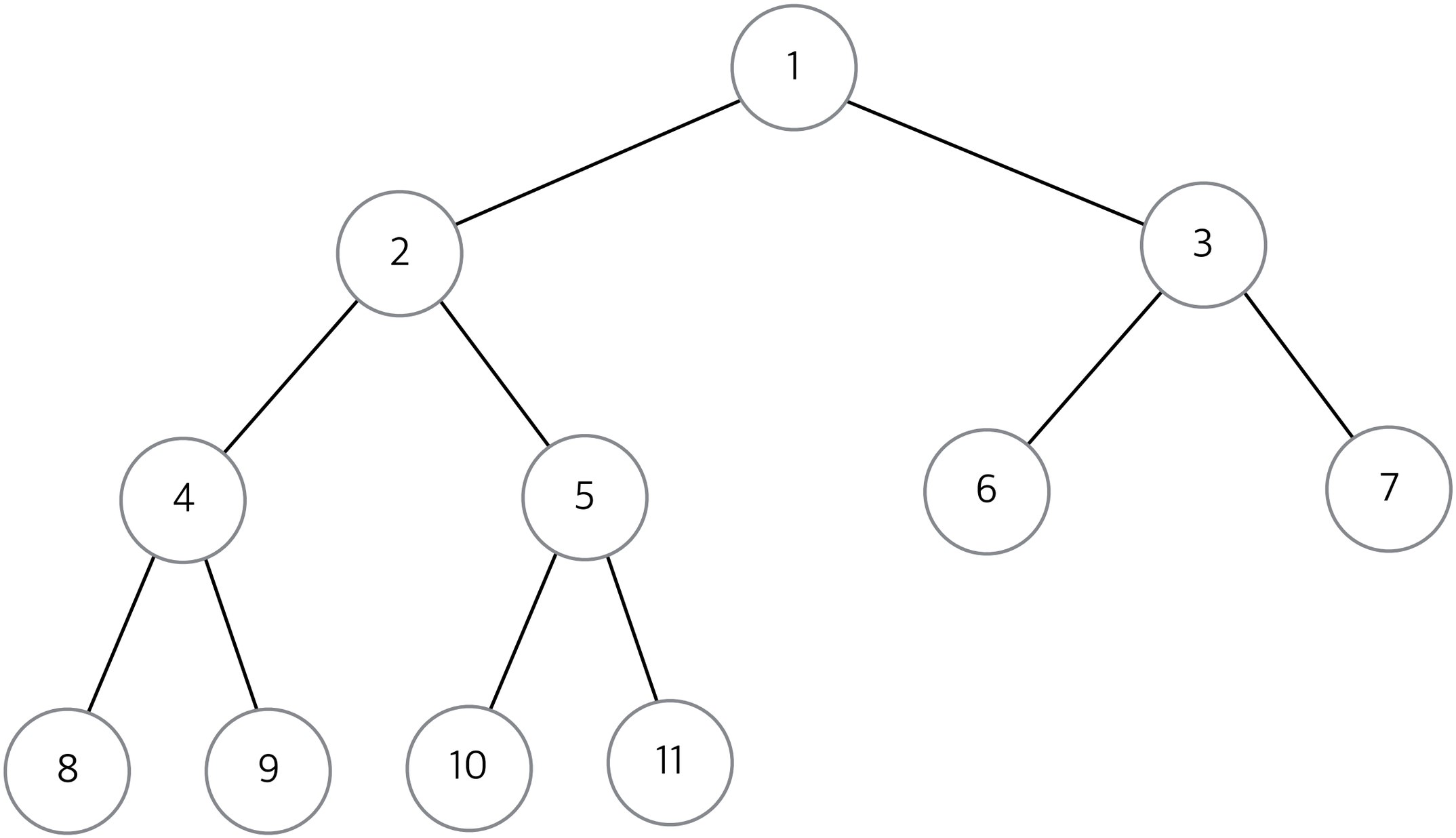
퀵정렬의 자세한 단계



퀵정렬의 자세한 단계



완전 이진 트리
(Complete binary tree)



[예제 1] 완전 이진 트리의 순회

완전 이진 트리의 노드 개수가 주어질 때,

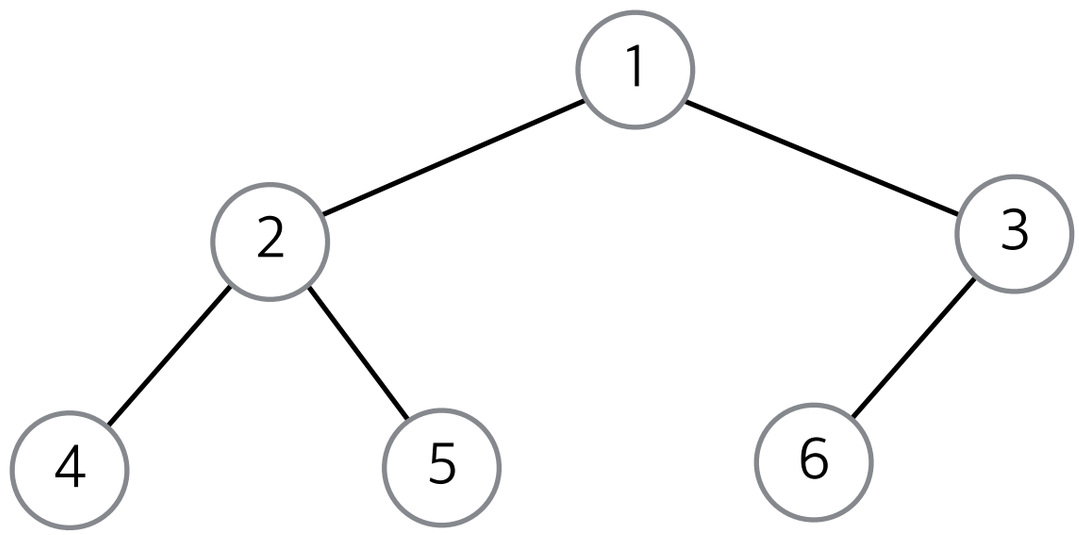
이 완전 이진 트리를 후위순회 한 결과를 출력하라.

입력의 예

```
6
```

출력의 예

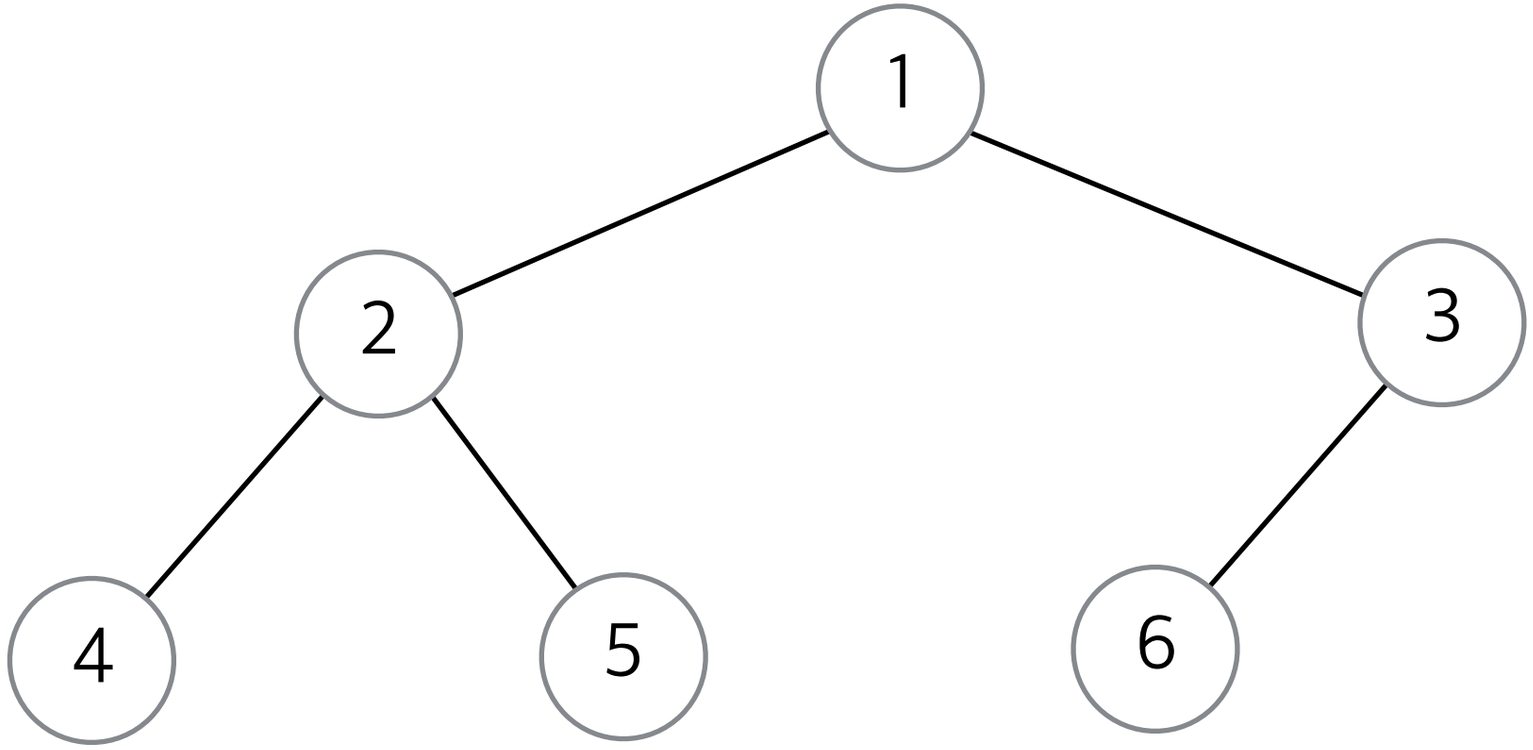
```
4 5 2 6 3 1
```



[예제 1] 완전 이진 트리의 순회

트리 문제라고 해서 항상 트리를

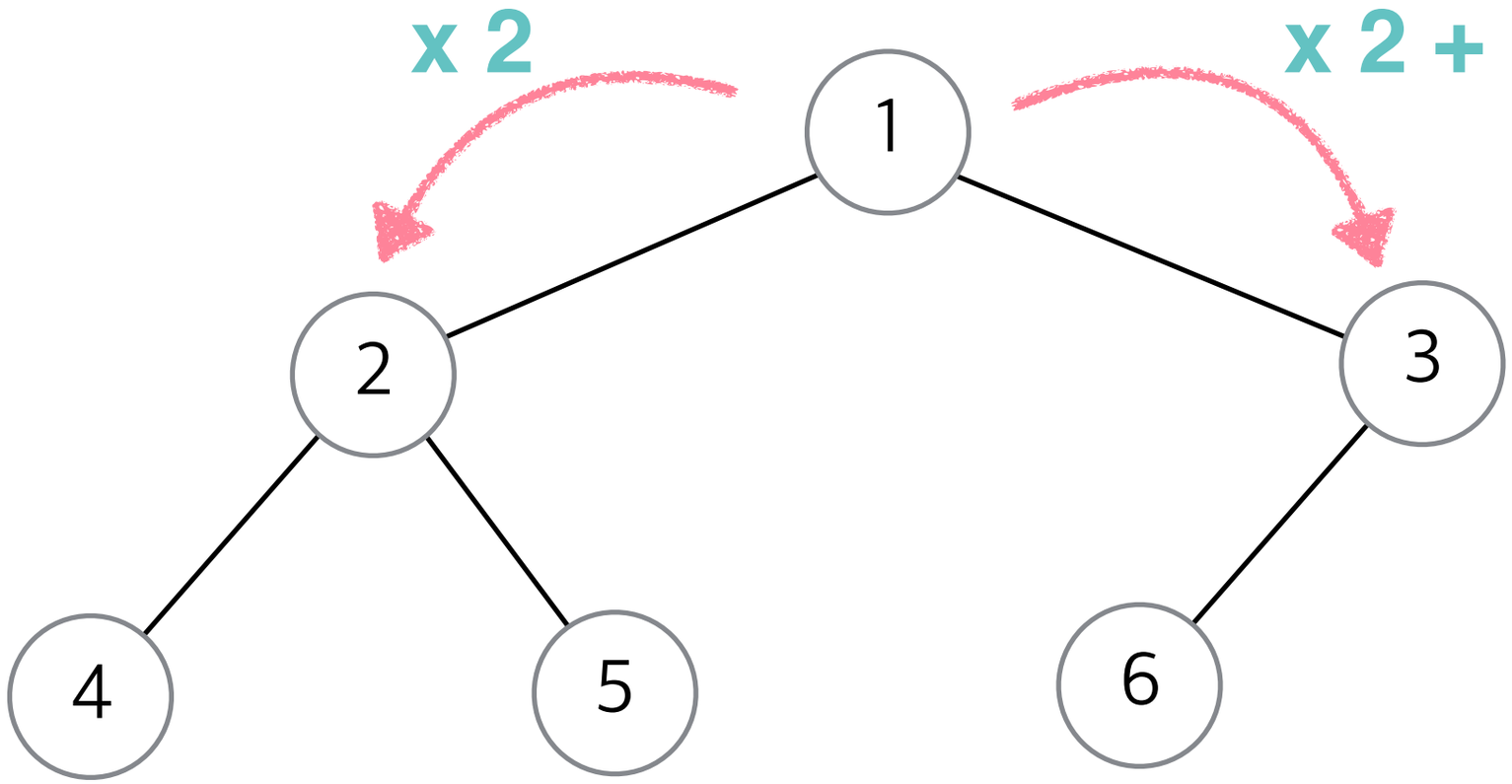
구현해야 하는 것이 아님



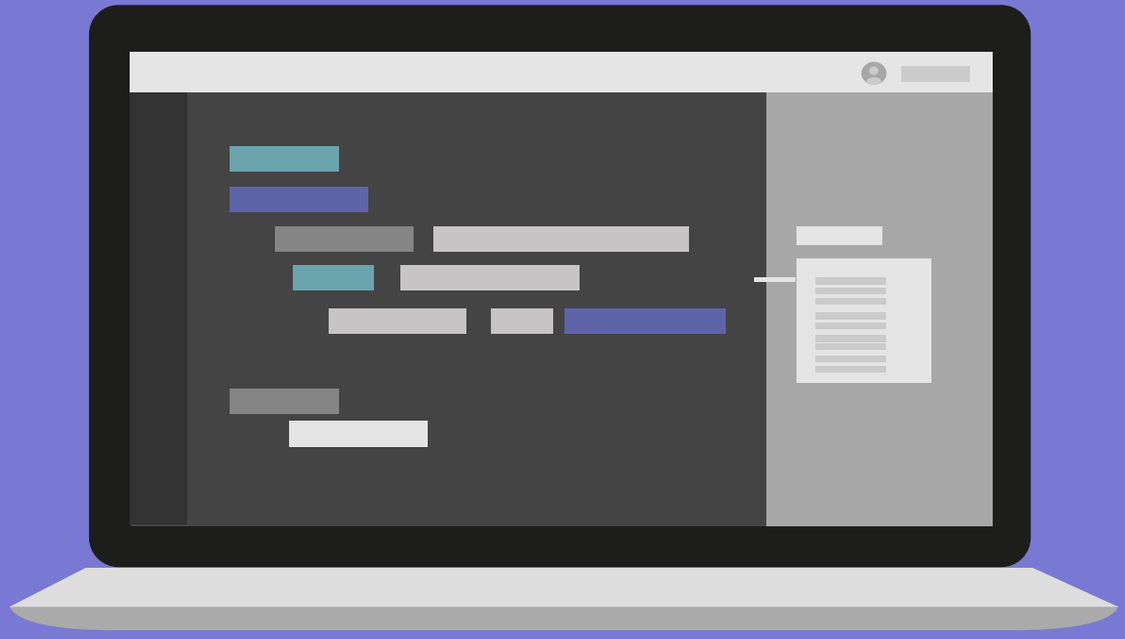
[예제 1] 완전 이진 트리의 순회

트리 문제라고 해서 항상 트리를

구현해야 하는 것이 아님



[예제 1] 완전 이진 트리의 순회



트리를 직접 구현 ?

사실 트리를 직접 class로

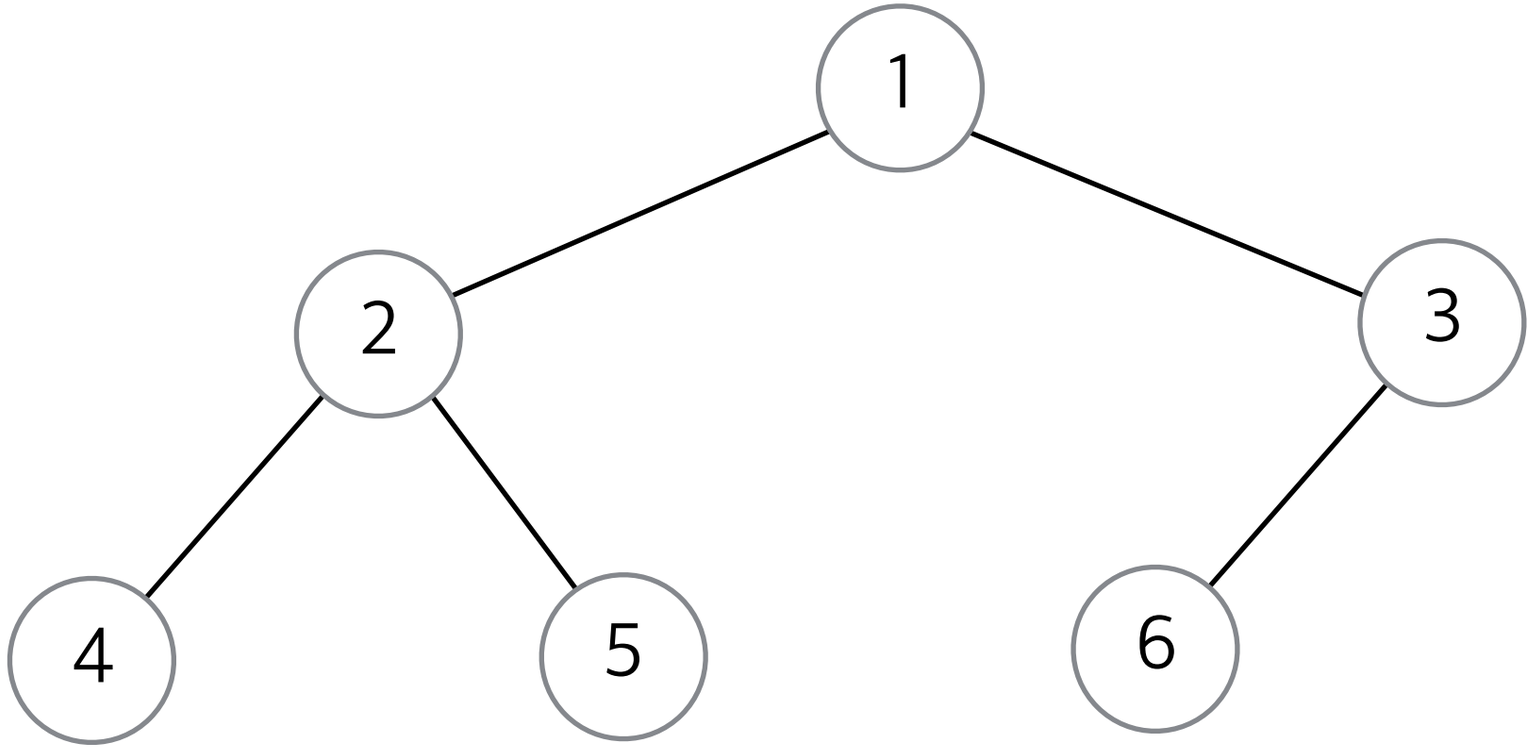
구현해야 할 일이 별로 없음

그래프 구현을 배우면

트리의 구현을 자연스럽게 습득

완전 이진 트리의 성질

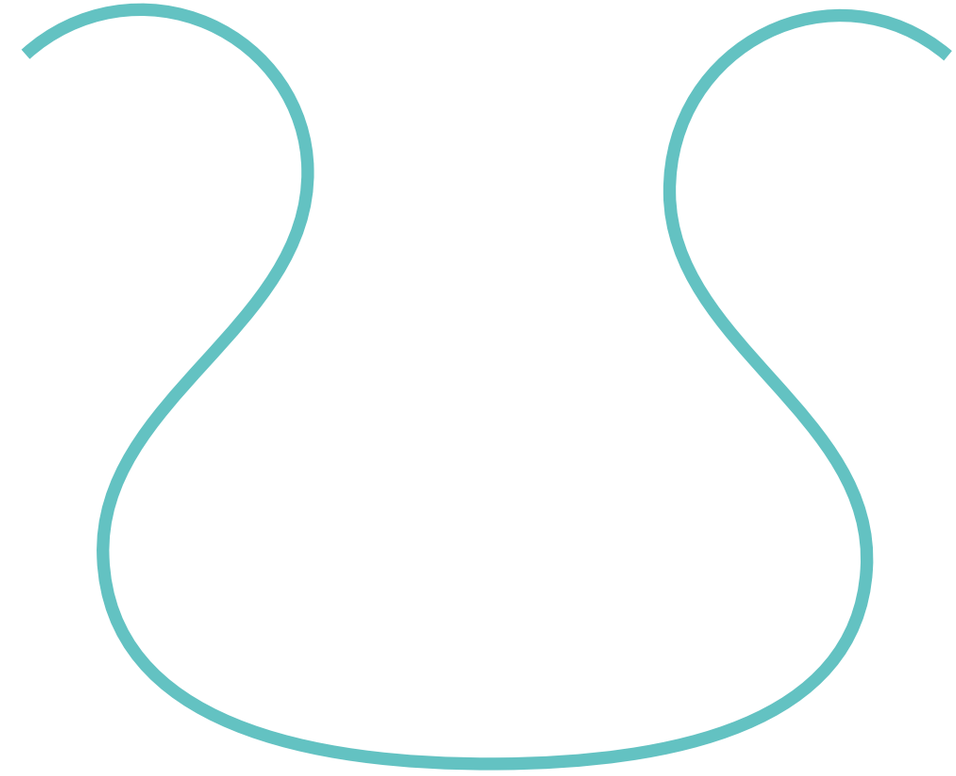
노드가 n개면, 그 높이는 약 log(n) 이다.



우선순위 큐

원소를 제거할 시,

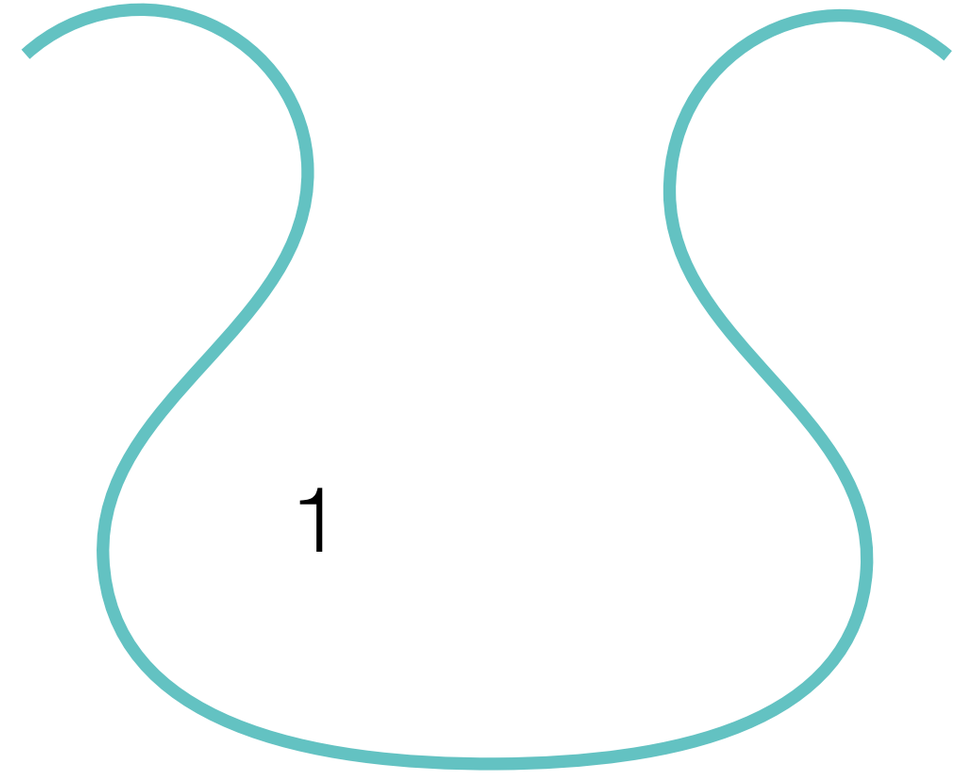
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

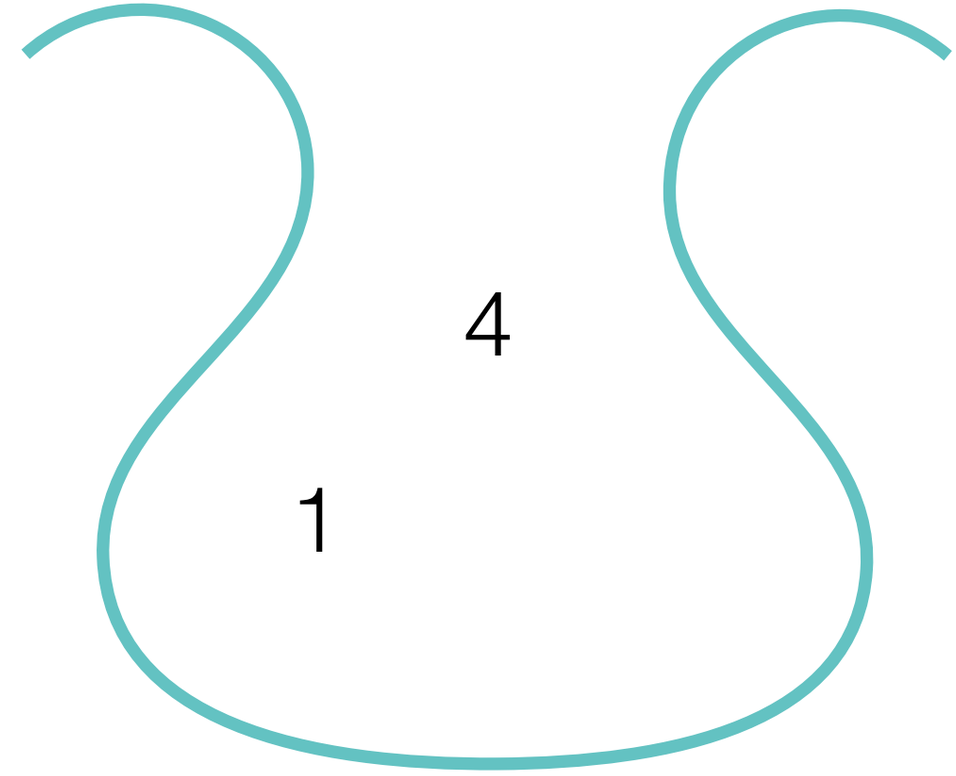
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

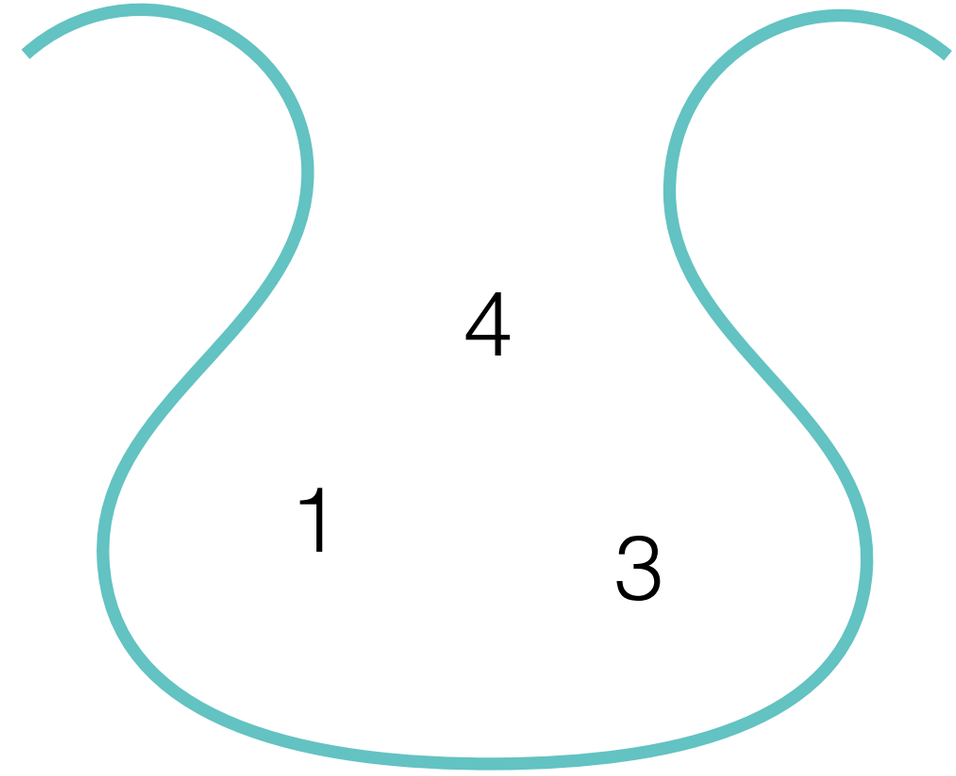
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

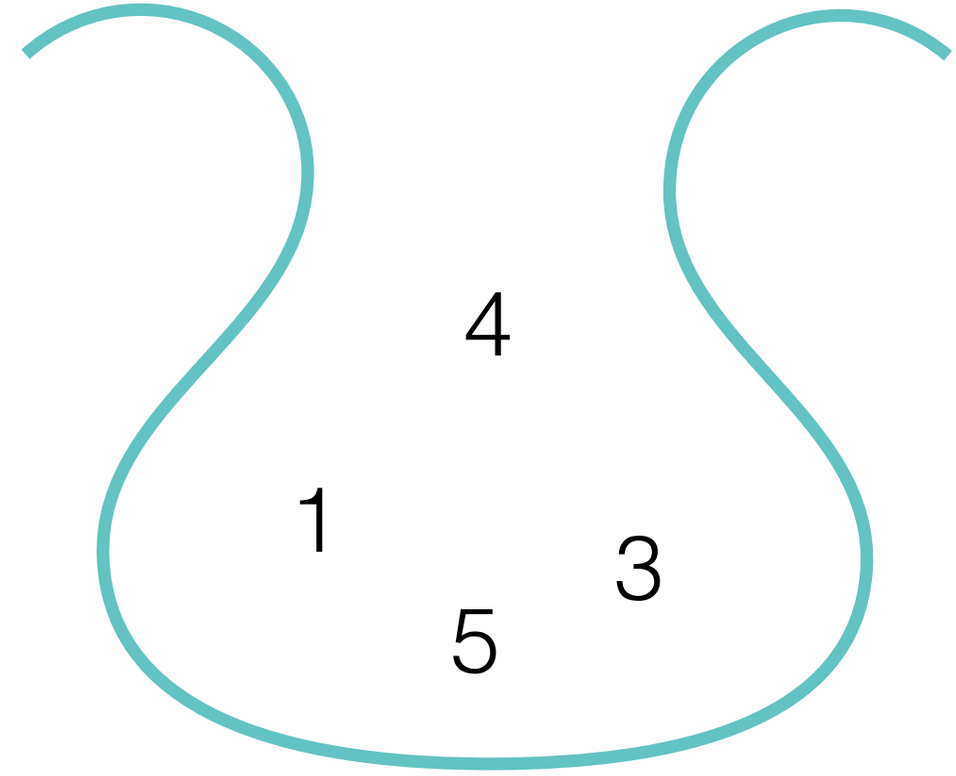
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

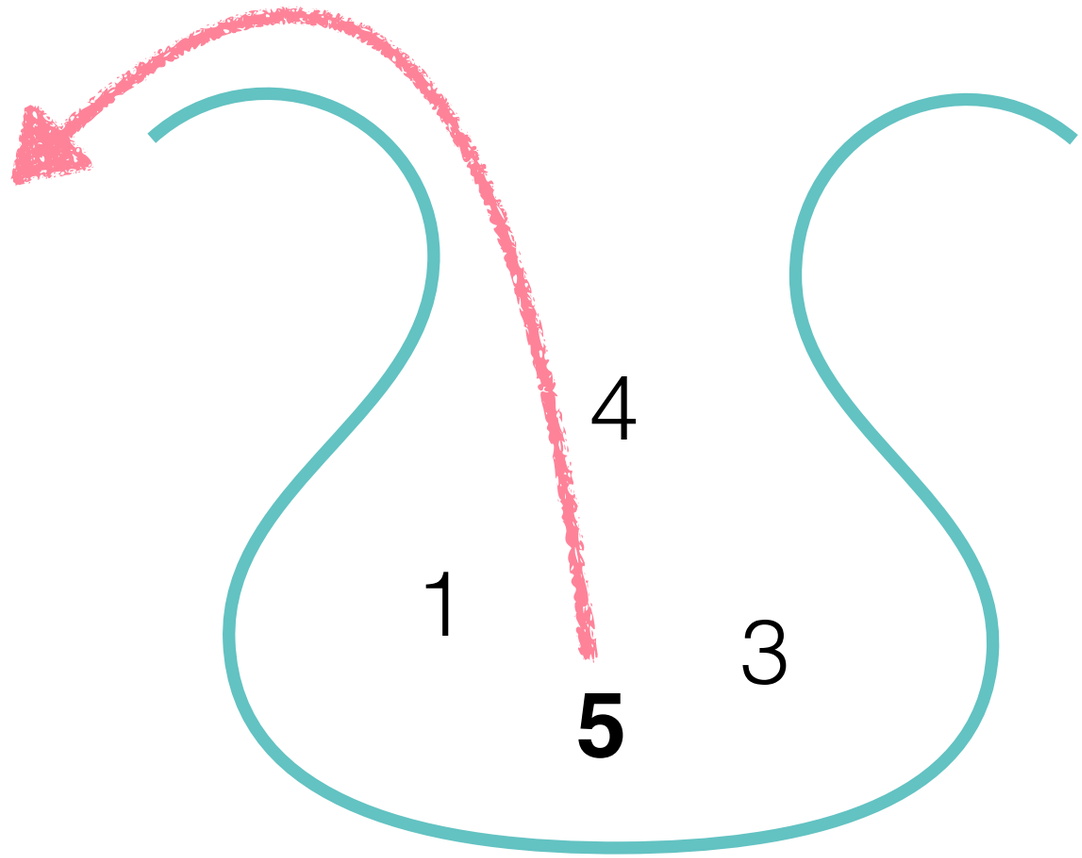
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

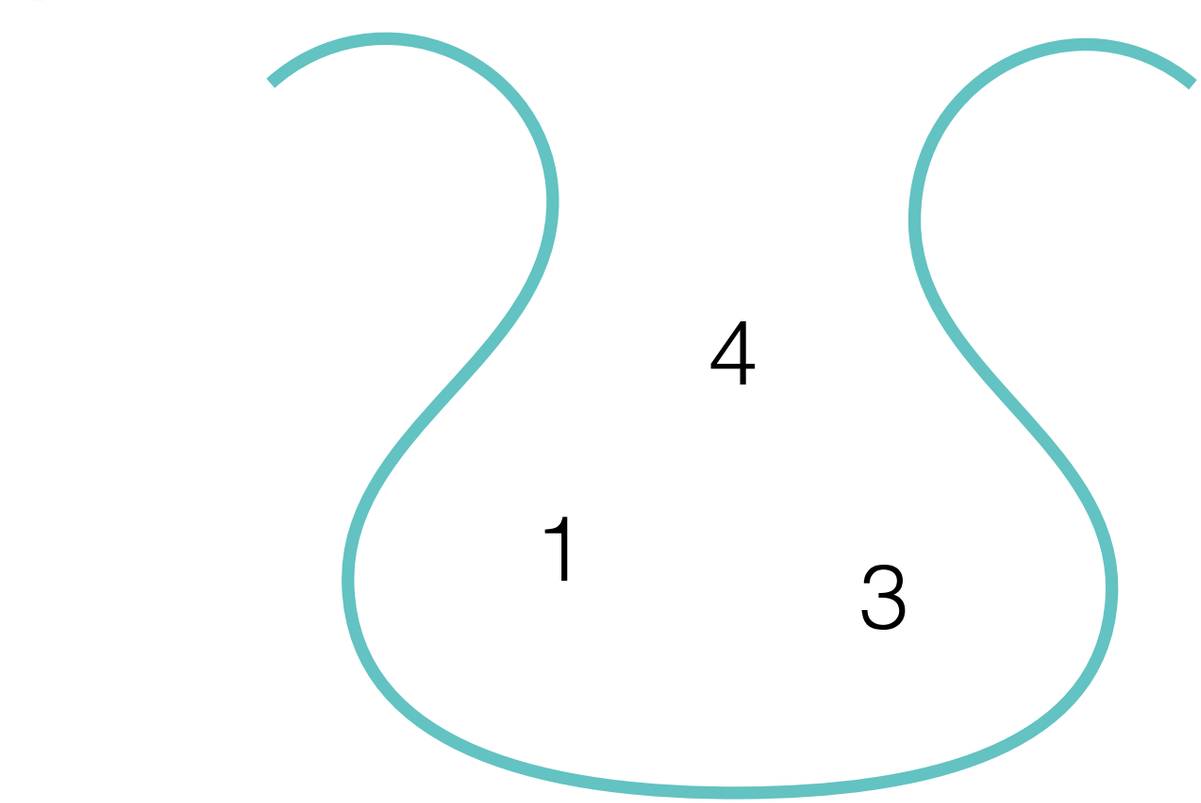
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

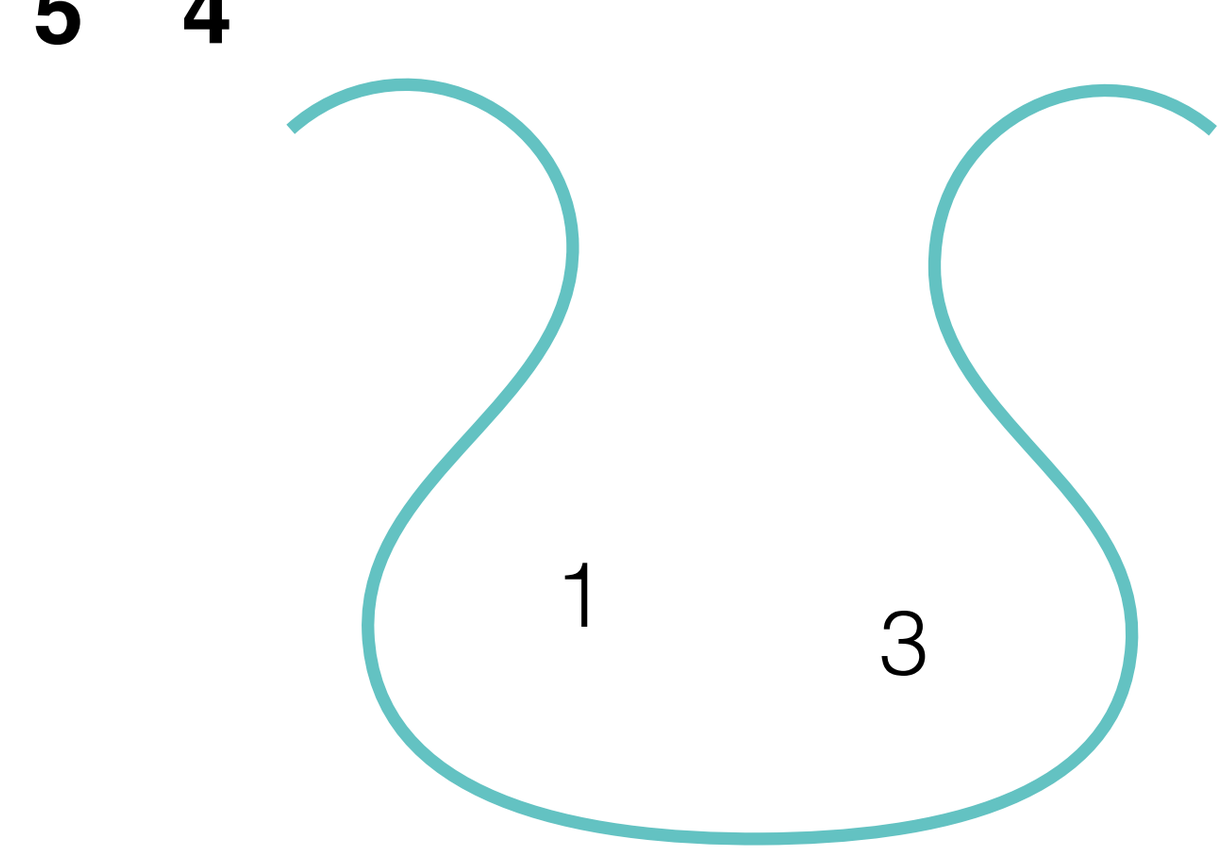
가장 우선순위가 높은 원소를 제거



우선순위 큐

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

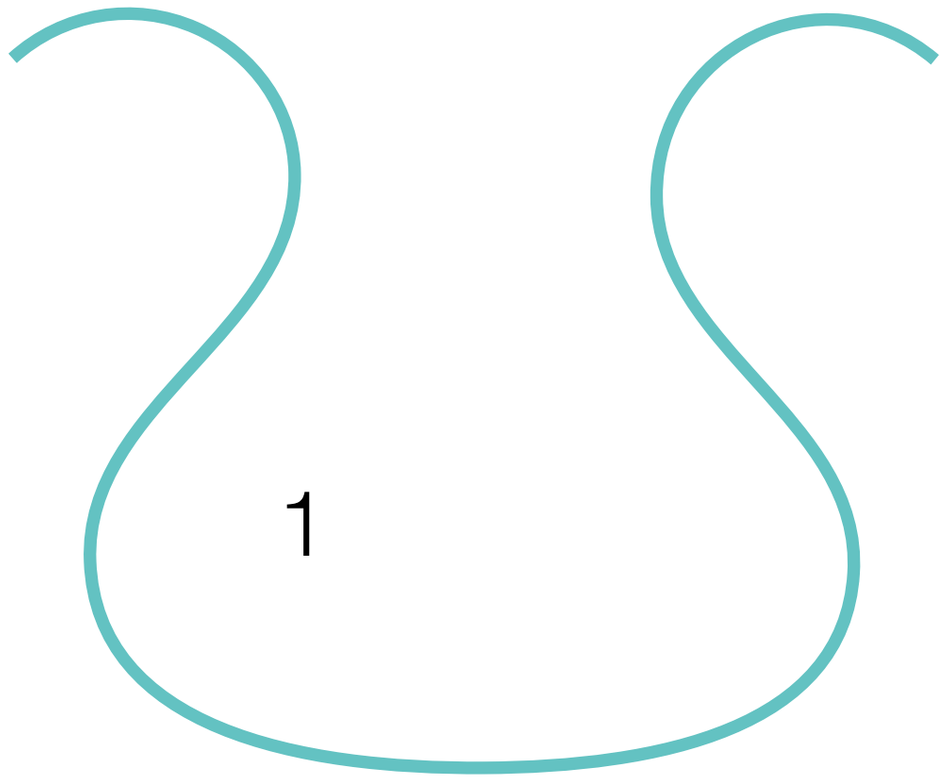


우선순위 큐

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3

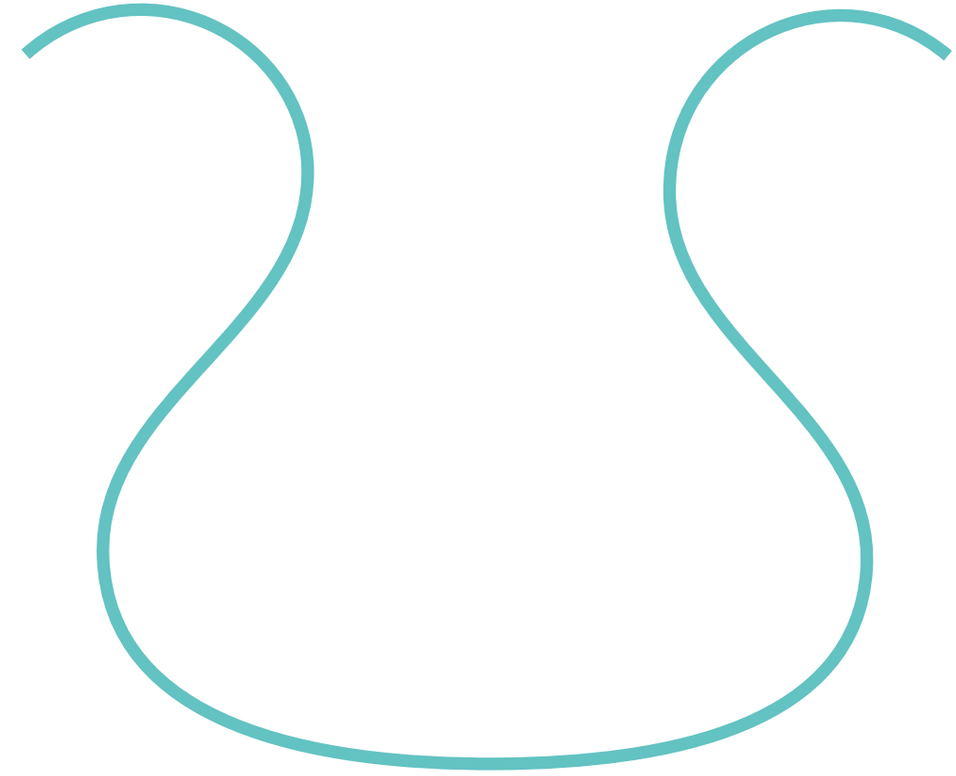


우선순위 큐

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

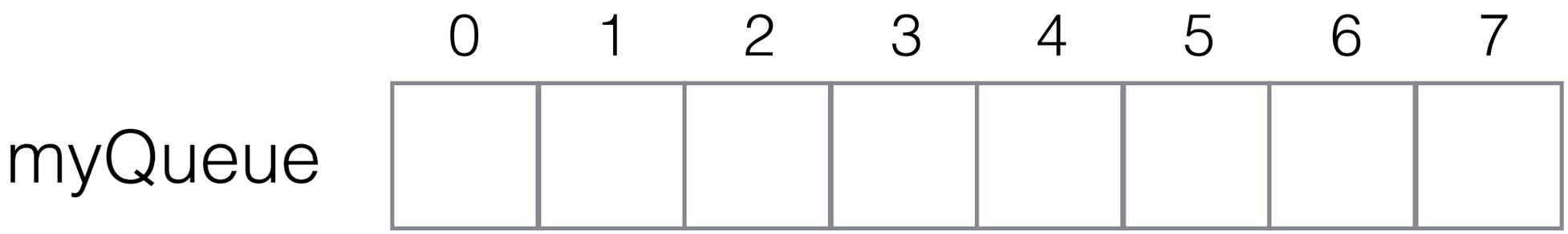
5 4 3 1



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

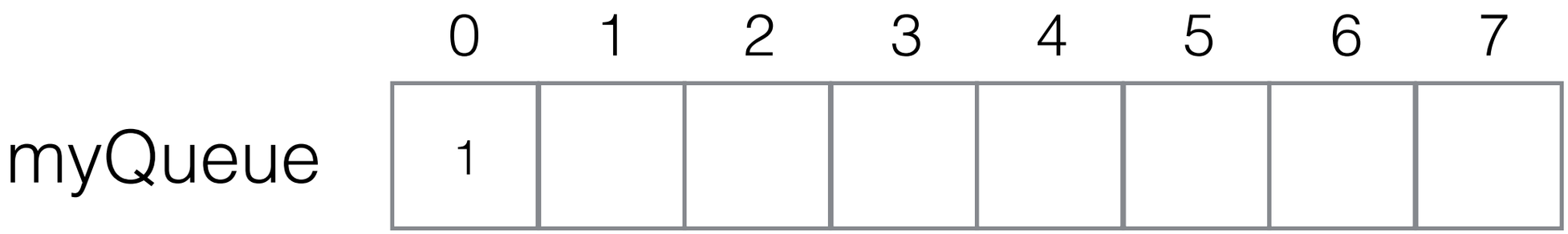
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

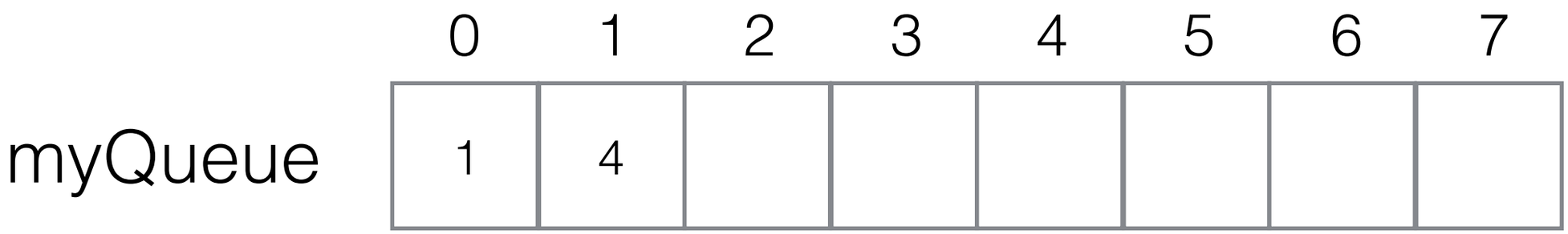
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

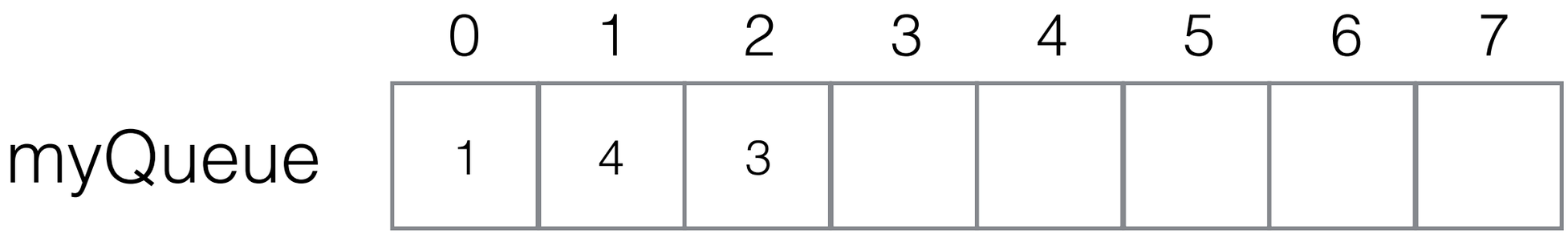
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

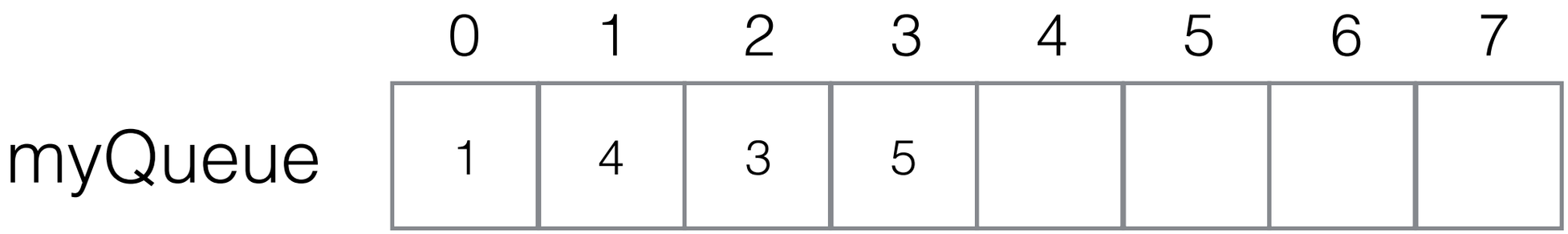
가장 우선순위가 높은 원소를 제거



우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

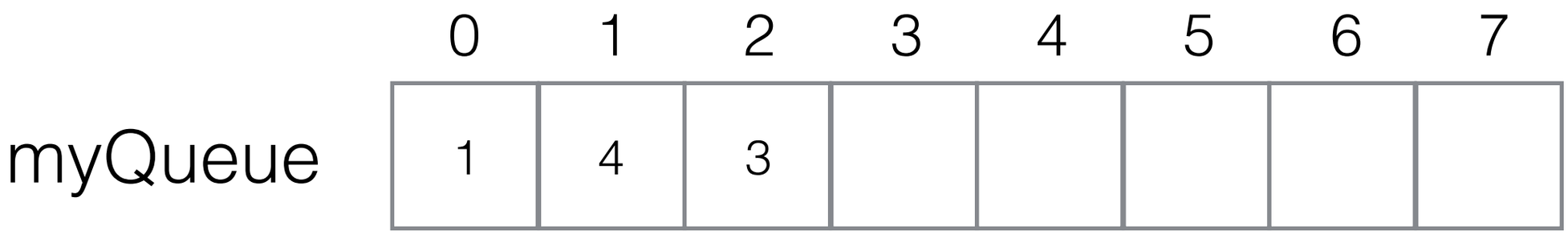


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5

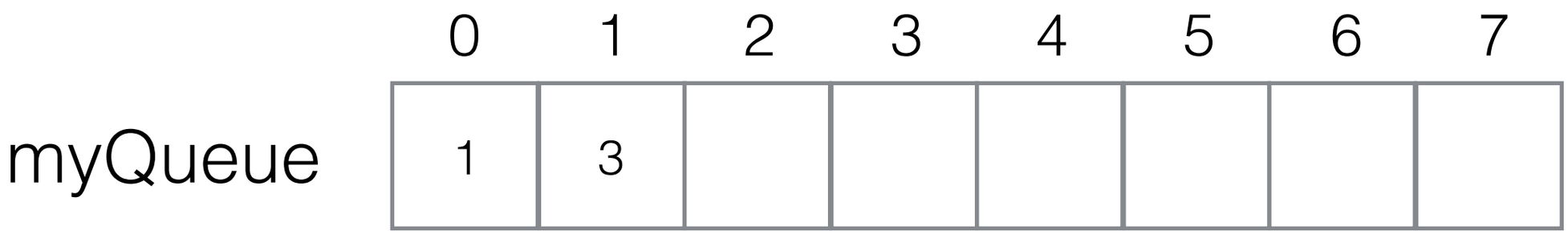


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4

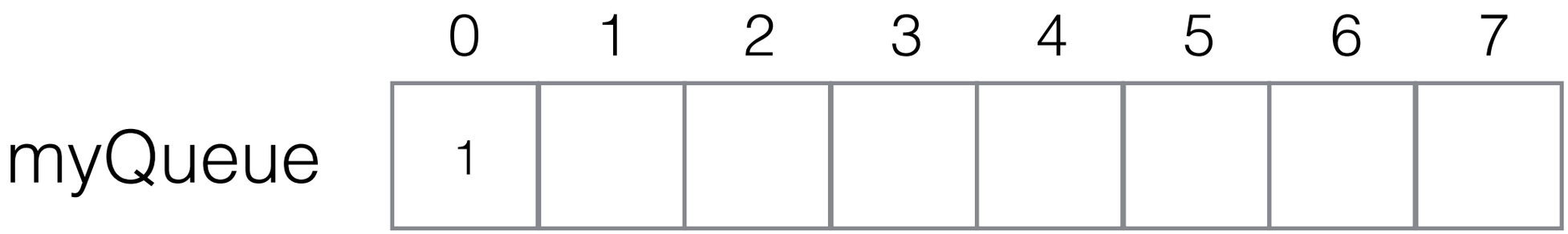


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3

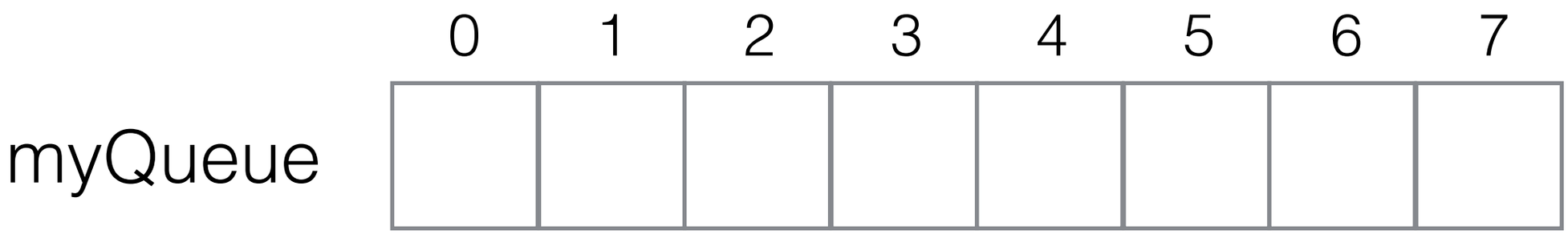


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3 1

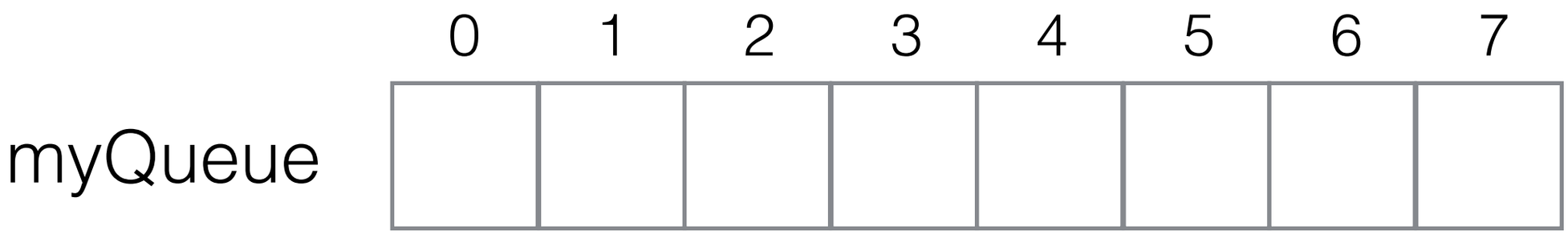


우선순위 큐 : 배열로 구현하기

원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3 1 삽입 : O(1)



우선순위 큐 : 배열로 구현하기

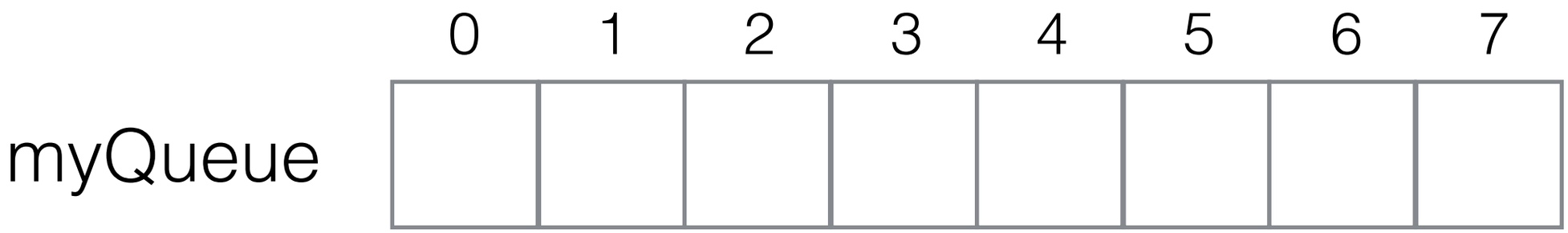
원소를 제거할 시,

가장 우선순위가 높은 원소를 제거

5 4 3 1

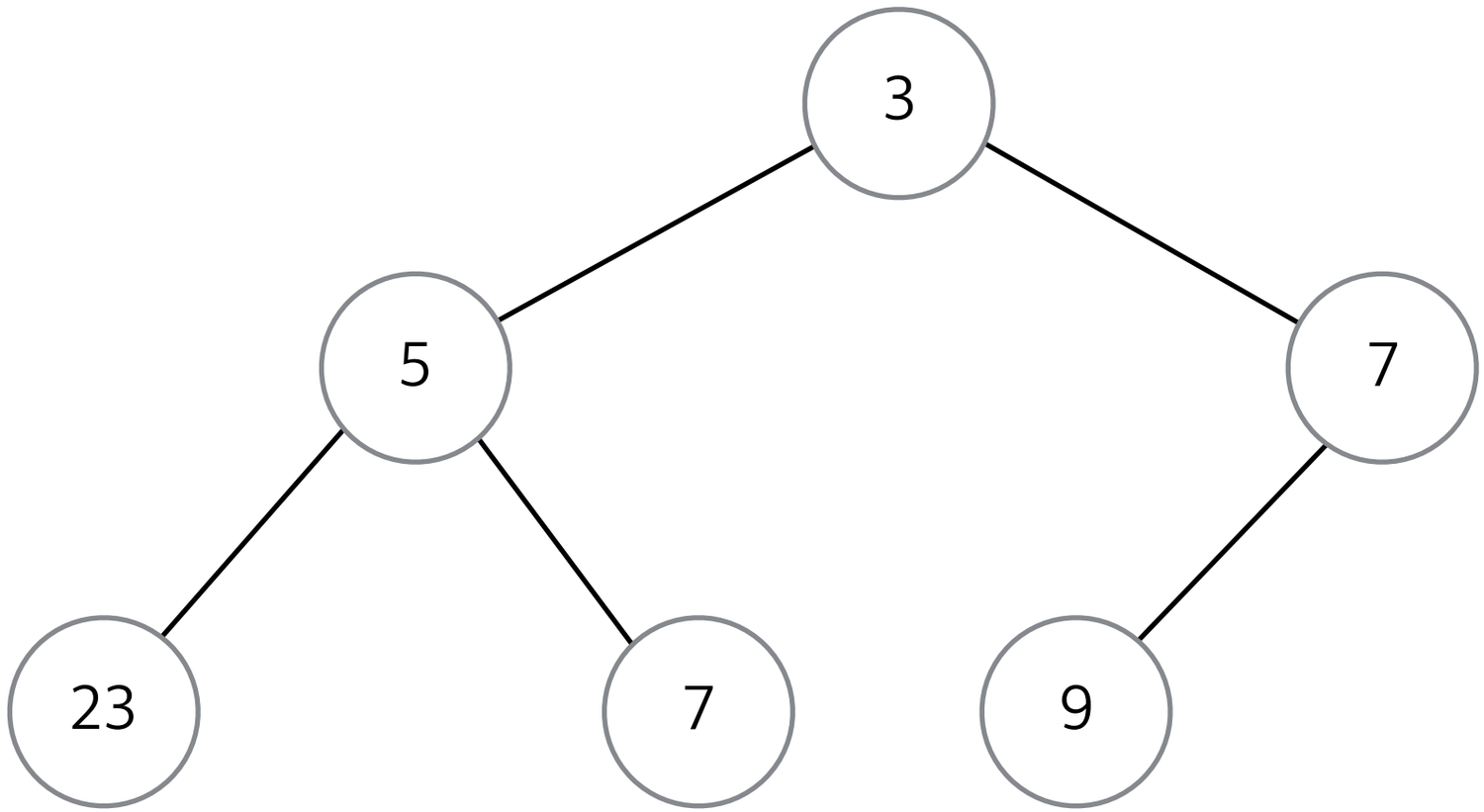
삽입 : O(1)

삭제 : O(n)



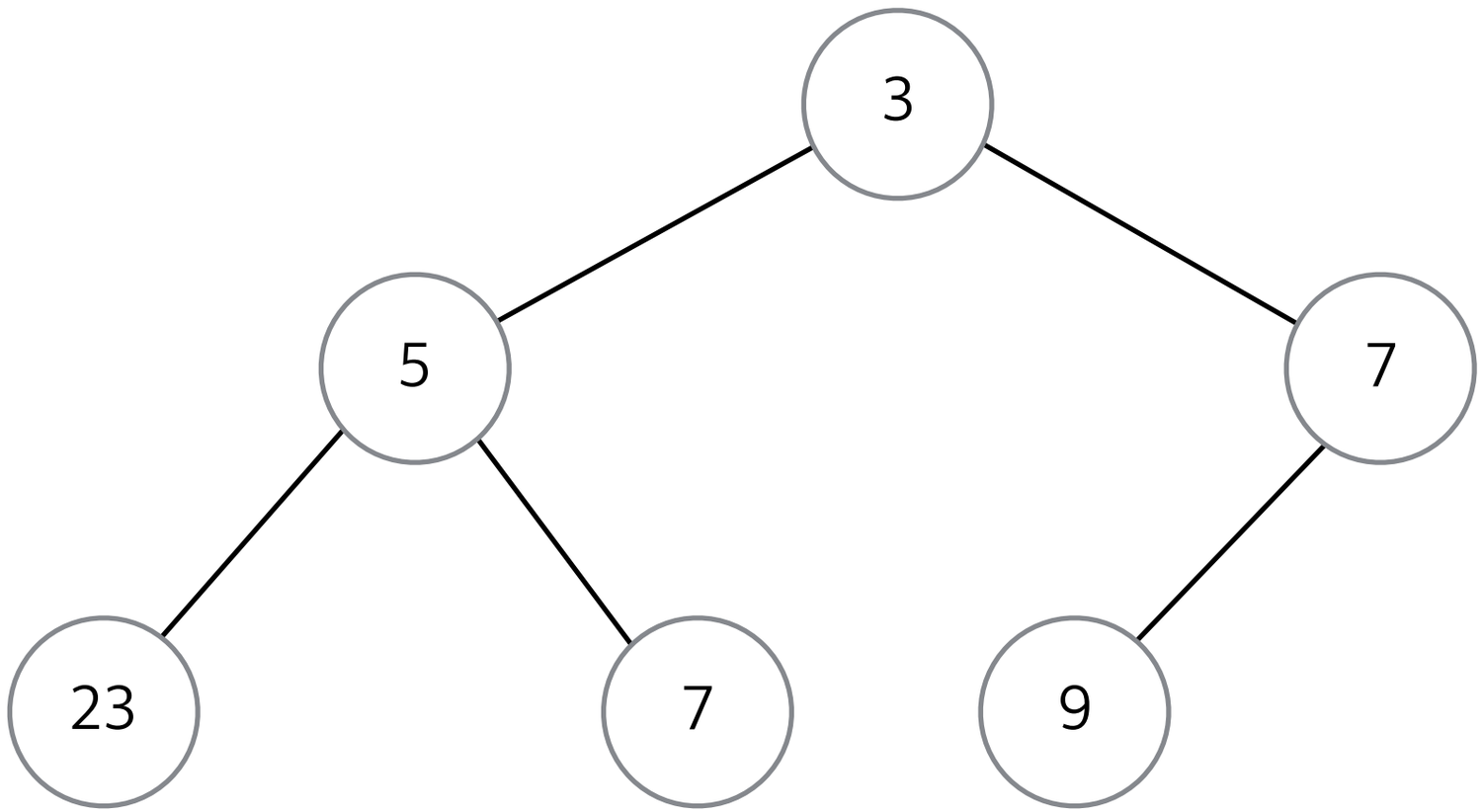
힙

부모의 값이 항상 자식보다 작은 완전 이진 트리



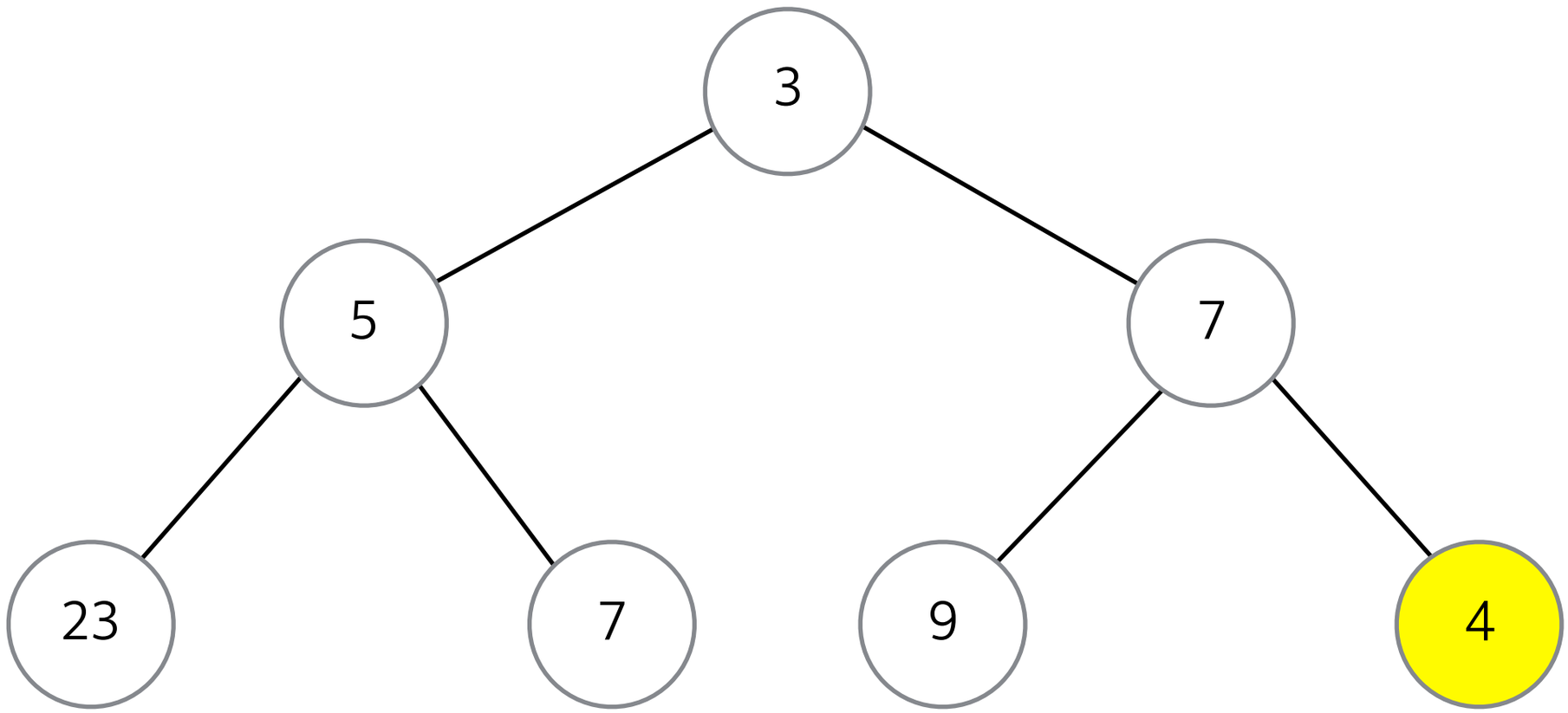
힙 : 값 삽입

heap.insert(4)



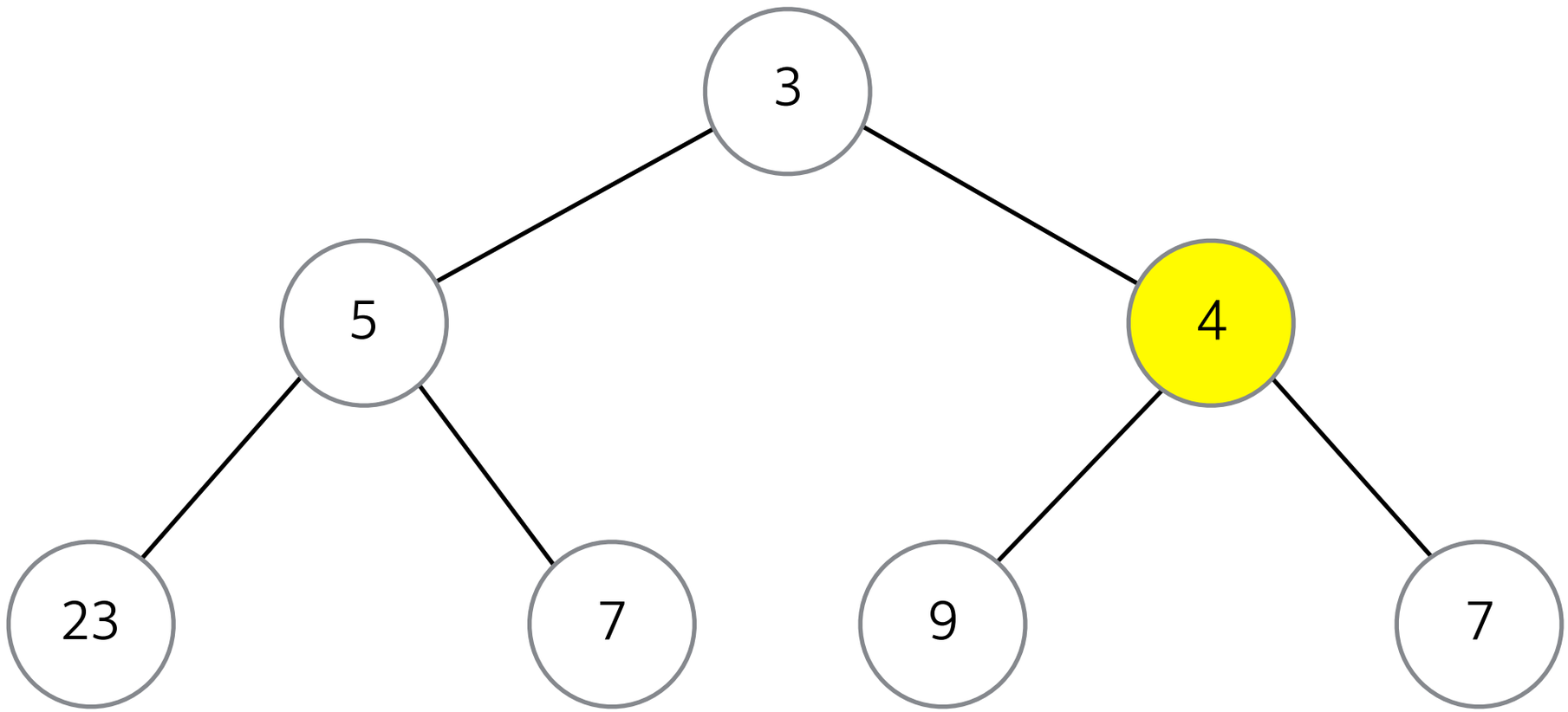
힙 : 값 삽입

heap.insert(4)



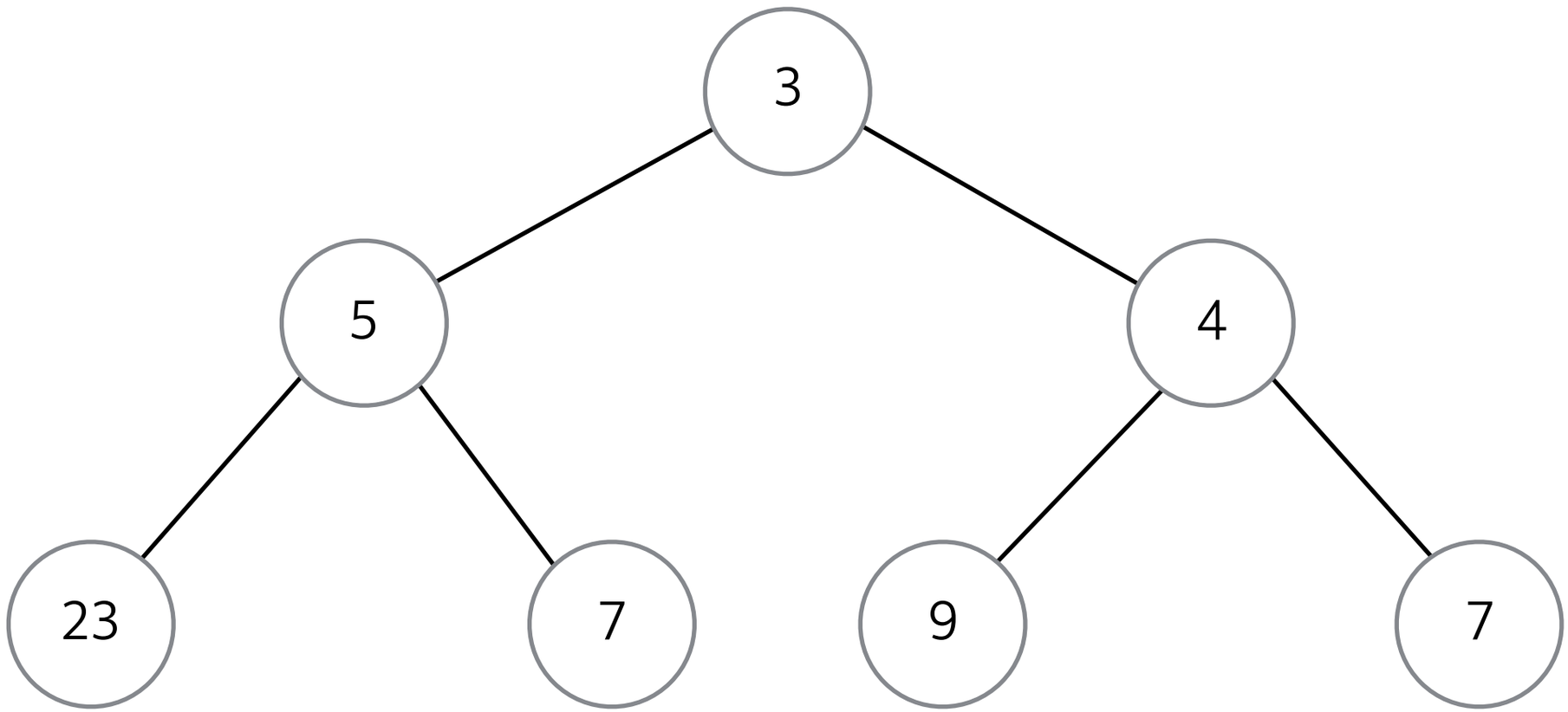
힙 : 값 삽입

heap.insert(4)



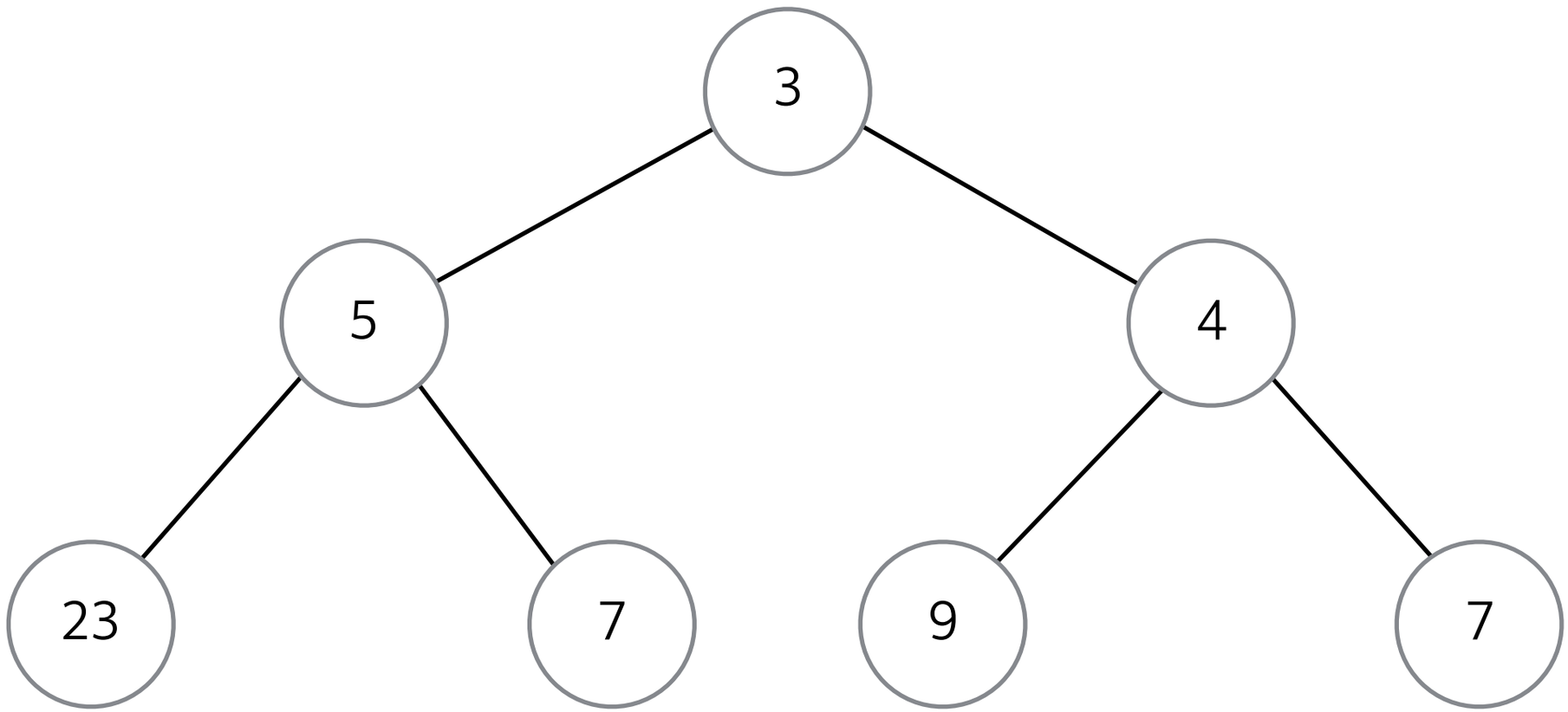
힙 : 값 삽입

heap.insert(4)



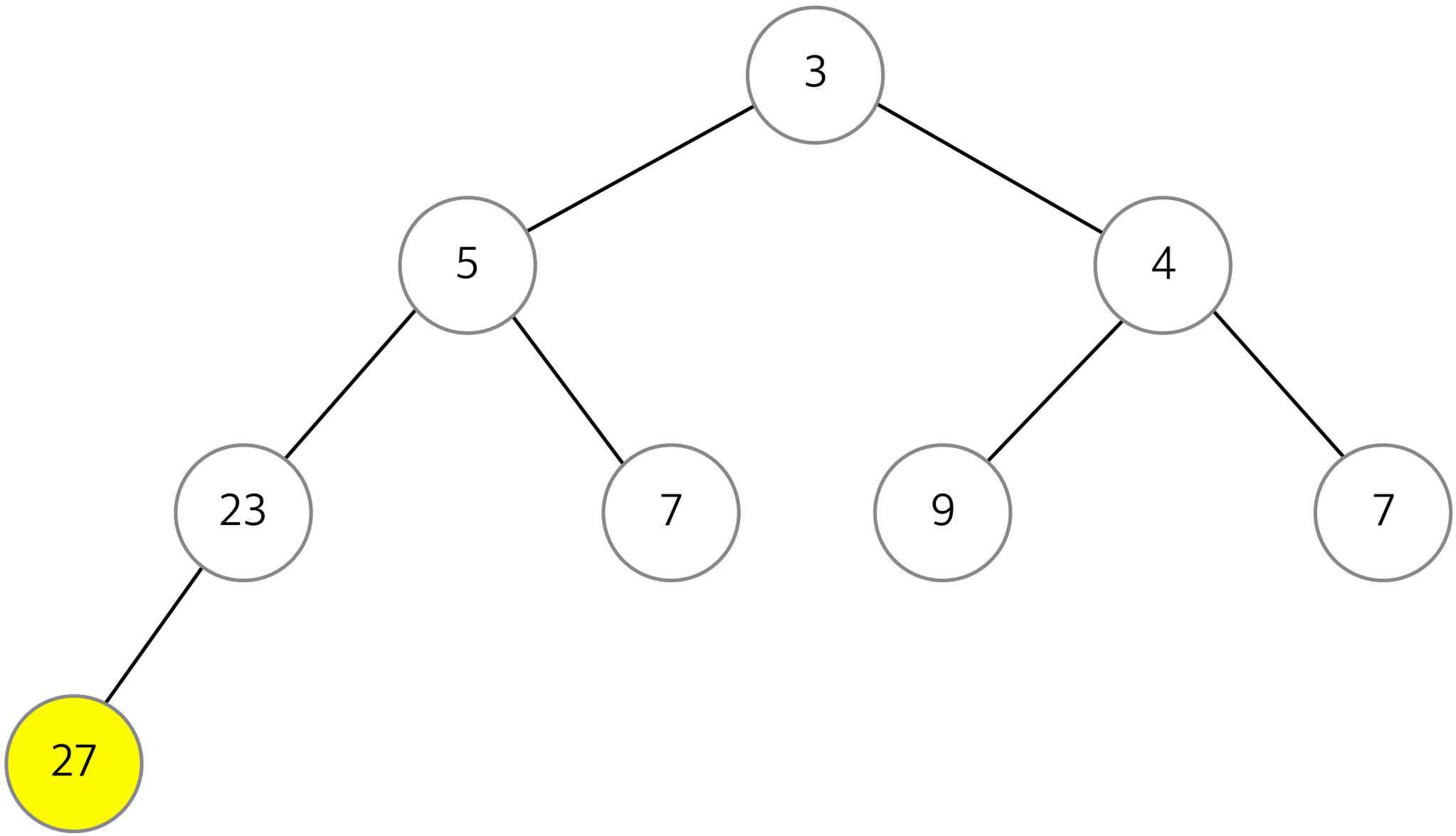
힙 : 값 삽입

heap.insert(27)



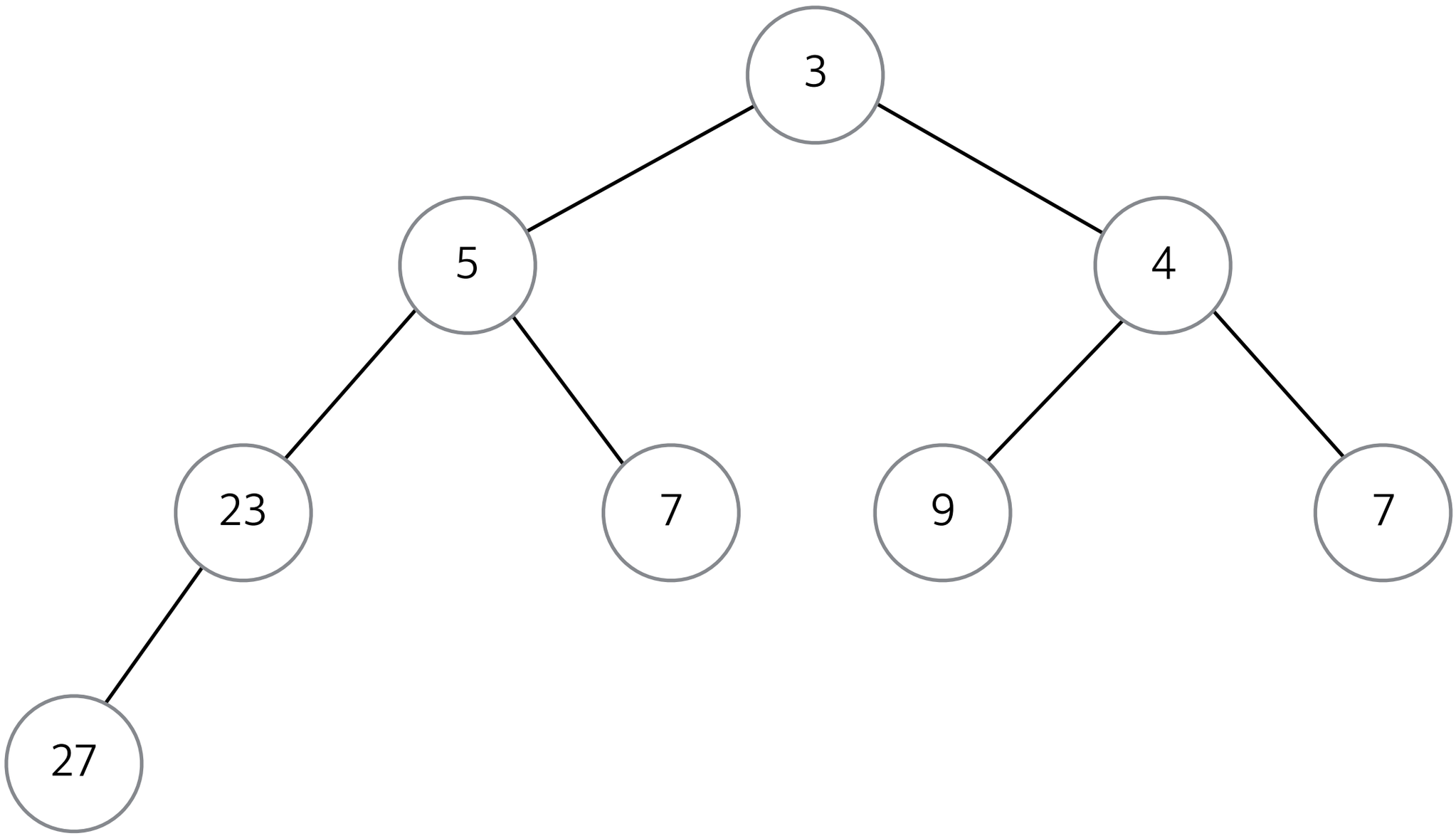
힙 : 값 삽입

```
heap.insert(27)
```



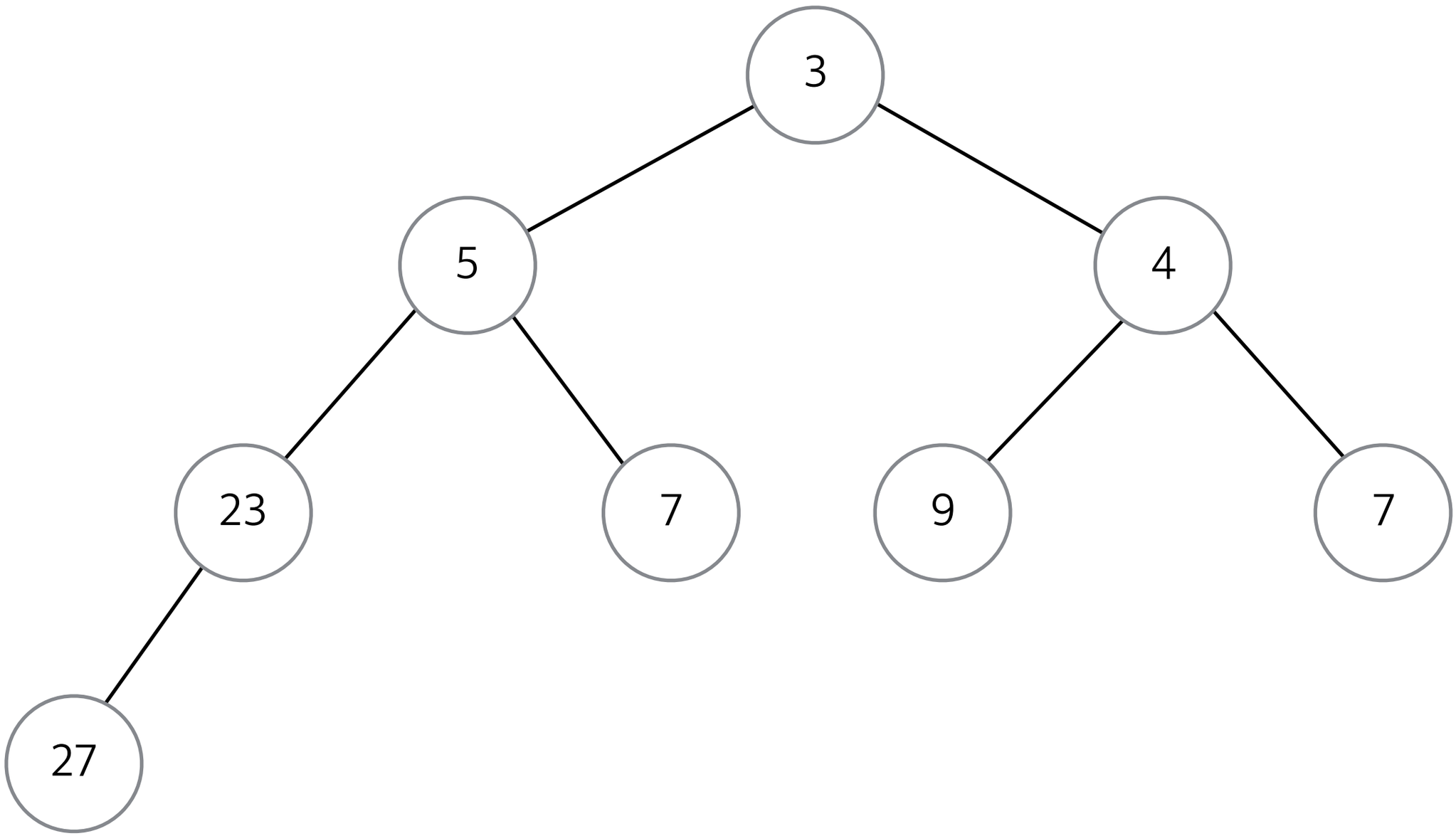
힙 : 값 삽입

heap.insert(27)



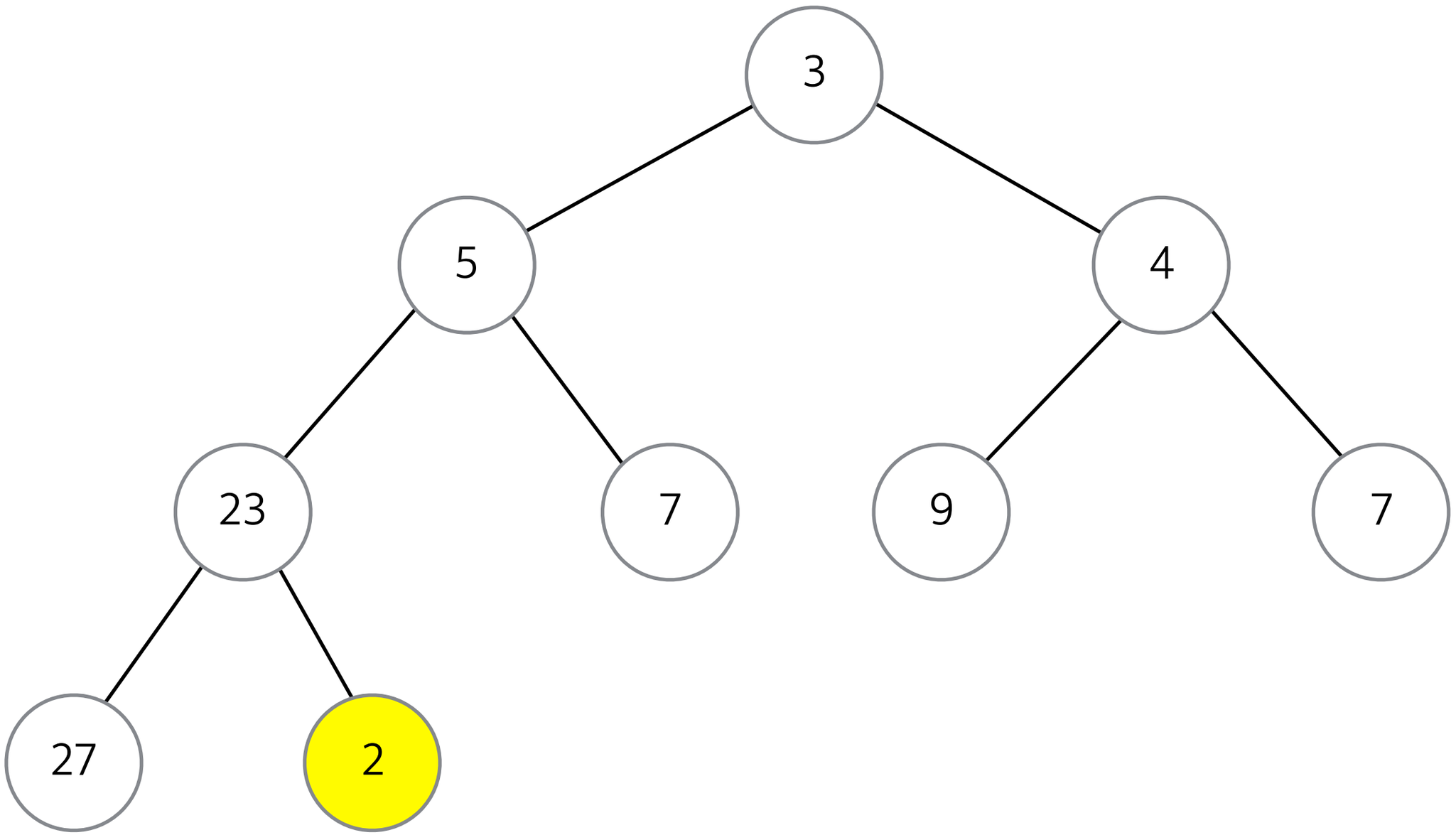
힙 : 값 삽입

heap.insert(2)



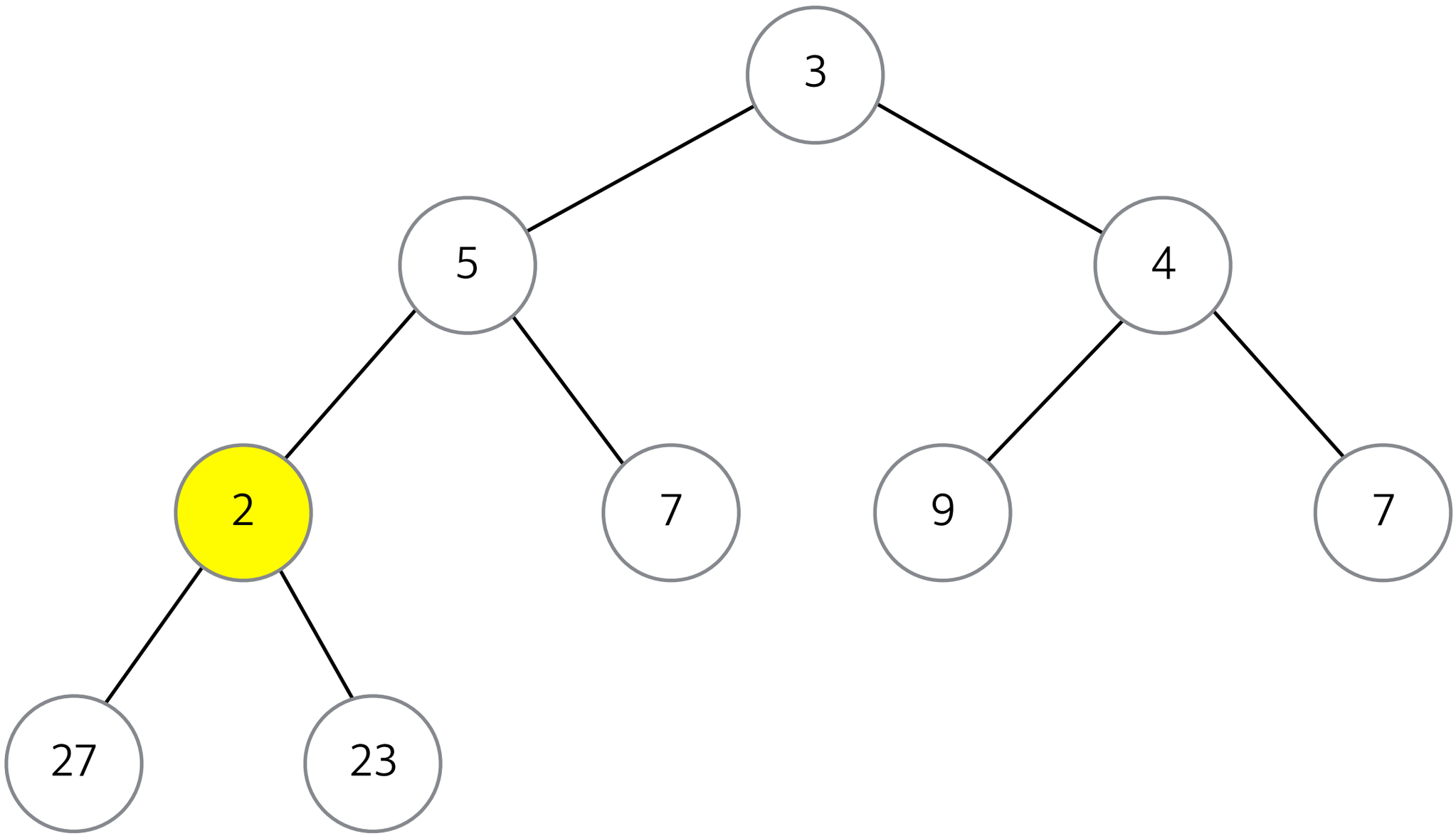
힙 : 값 삽입

heap.insert(2)



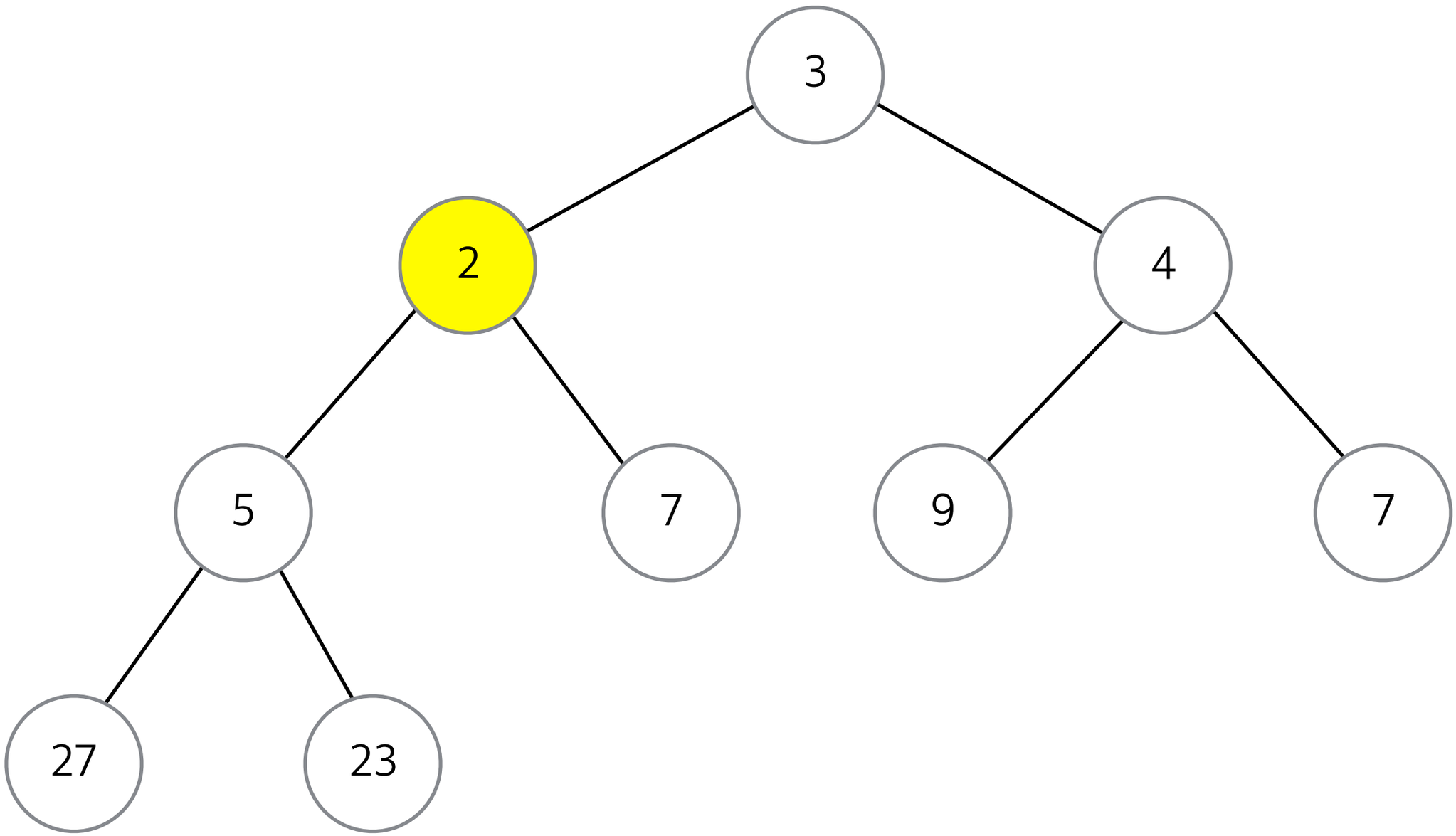
힙 : 값 삽입

heap.insert(2)



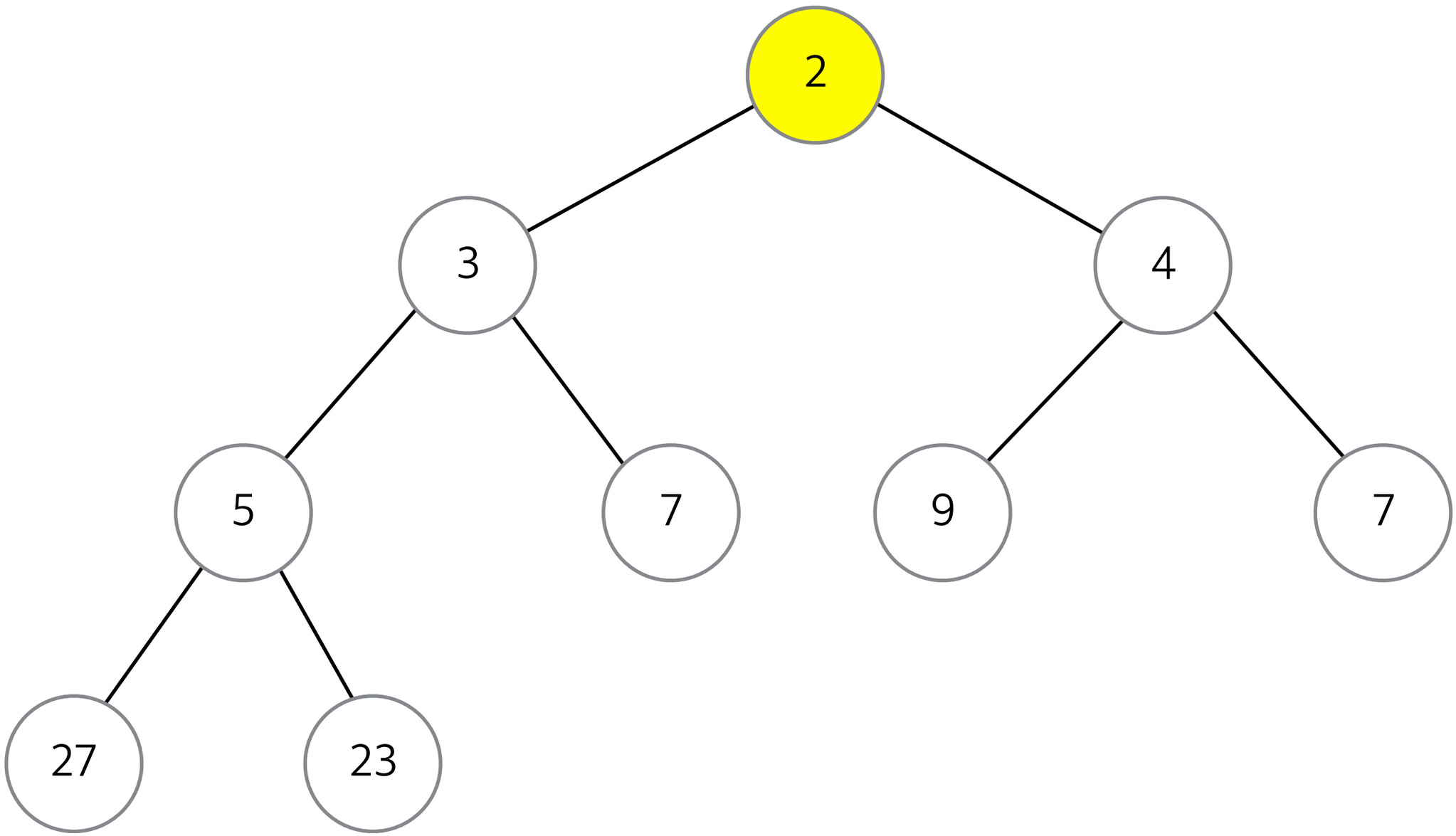
힙 : 값 삽입

heap.insert(2)



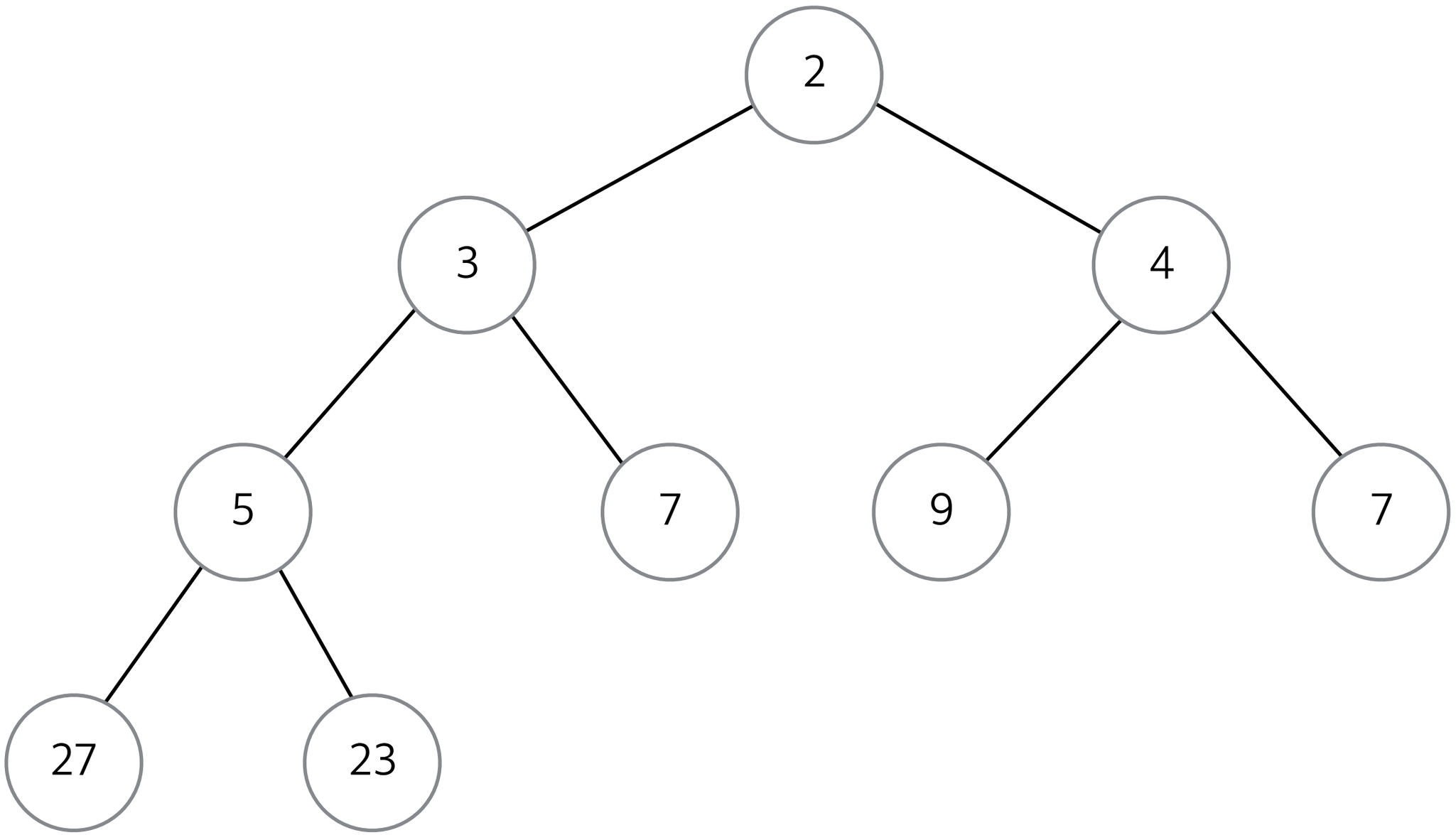
힙 : 값 삽입

heap.insert(2)

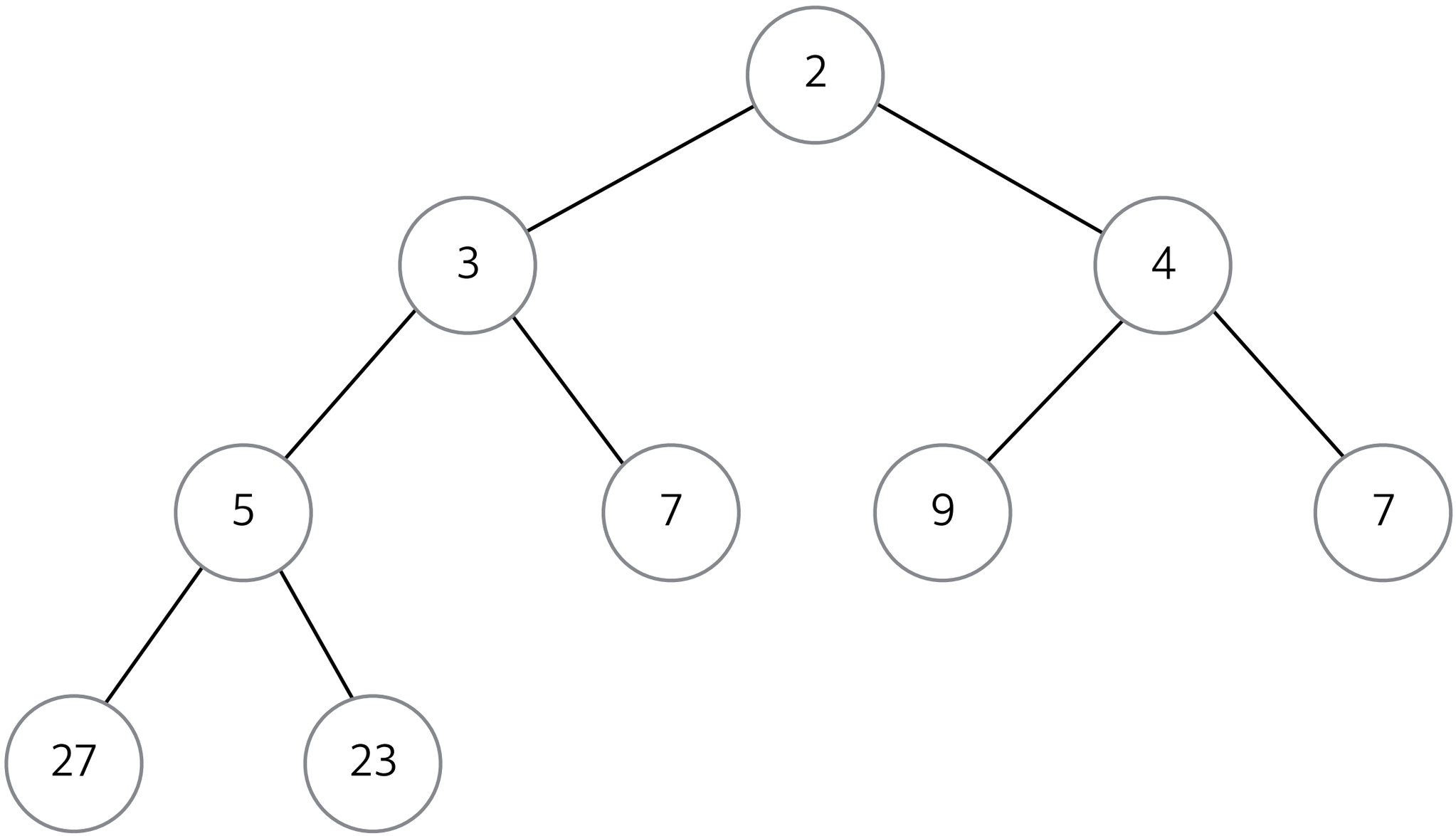


힙 : 값 삽입

heap.insert(2)

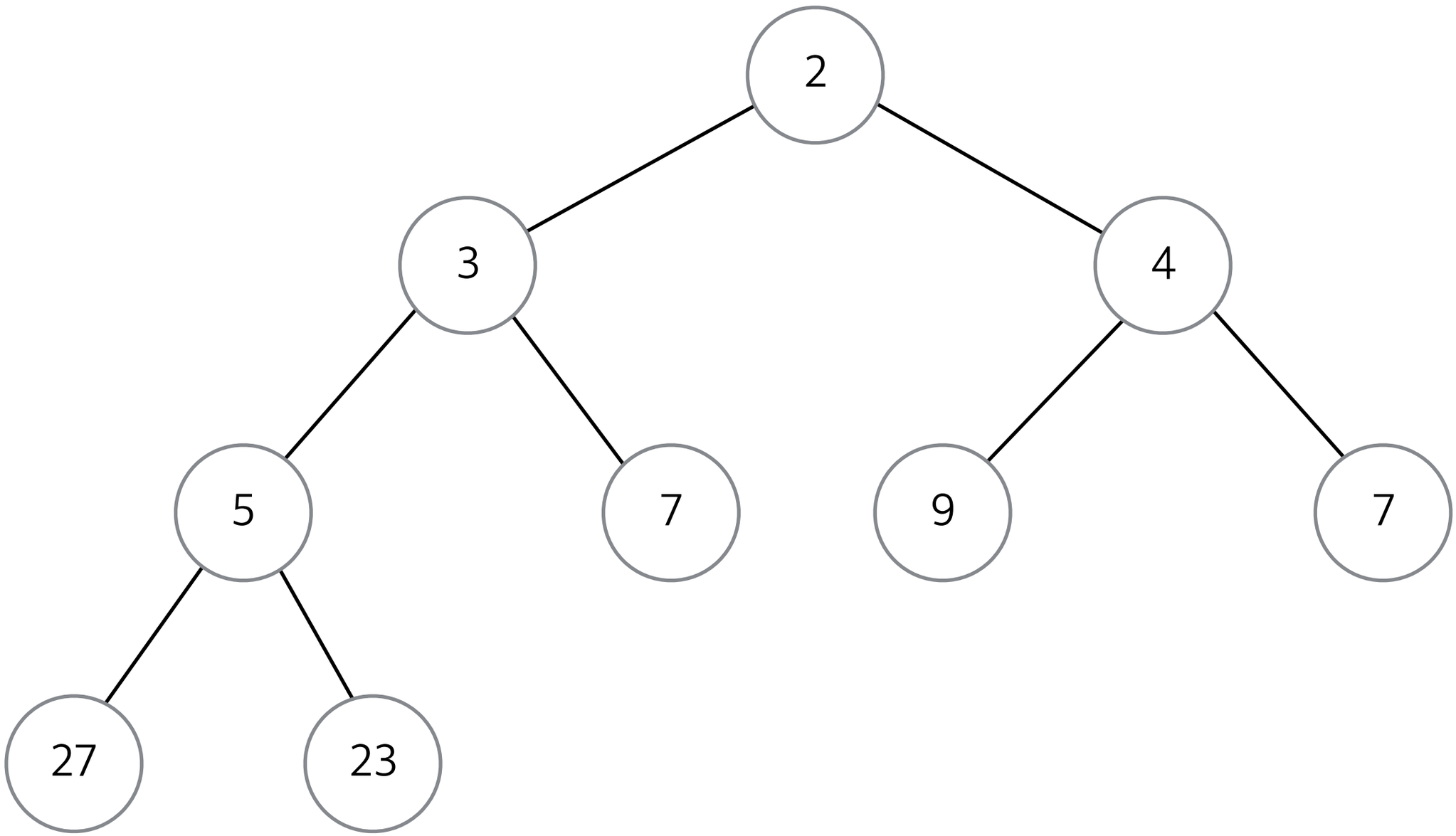


힙 : 값 삽입의 시간복잡도



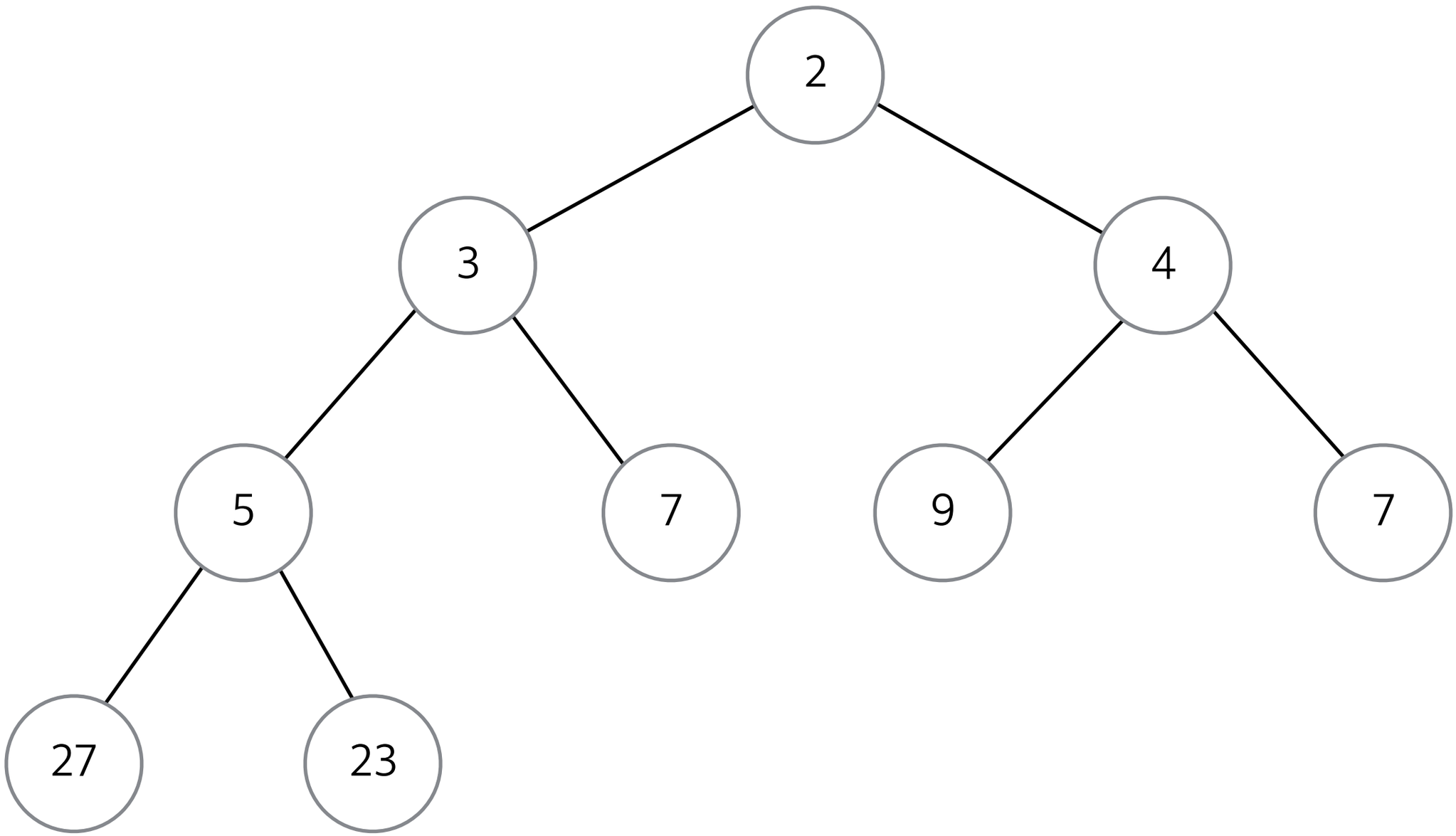
힙 : 값 삽입의 시간복잡도

O(log n)



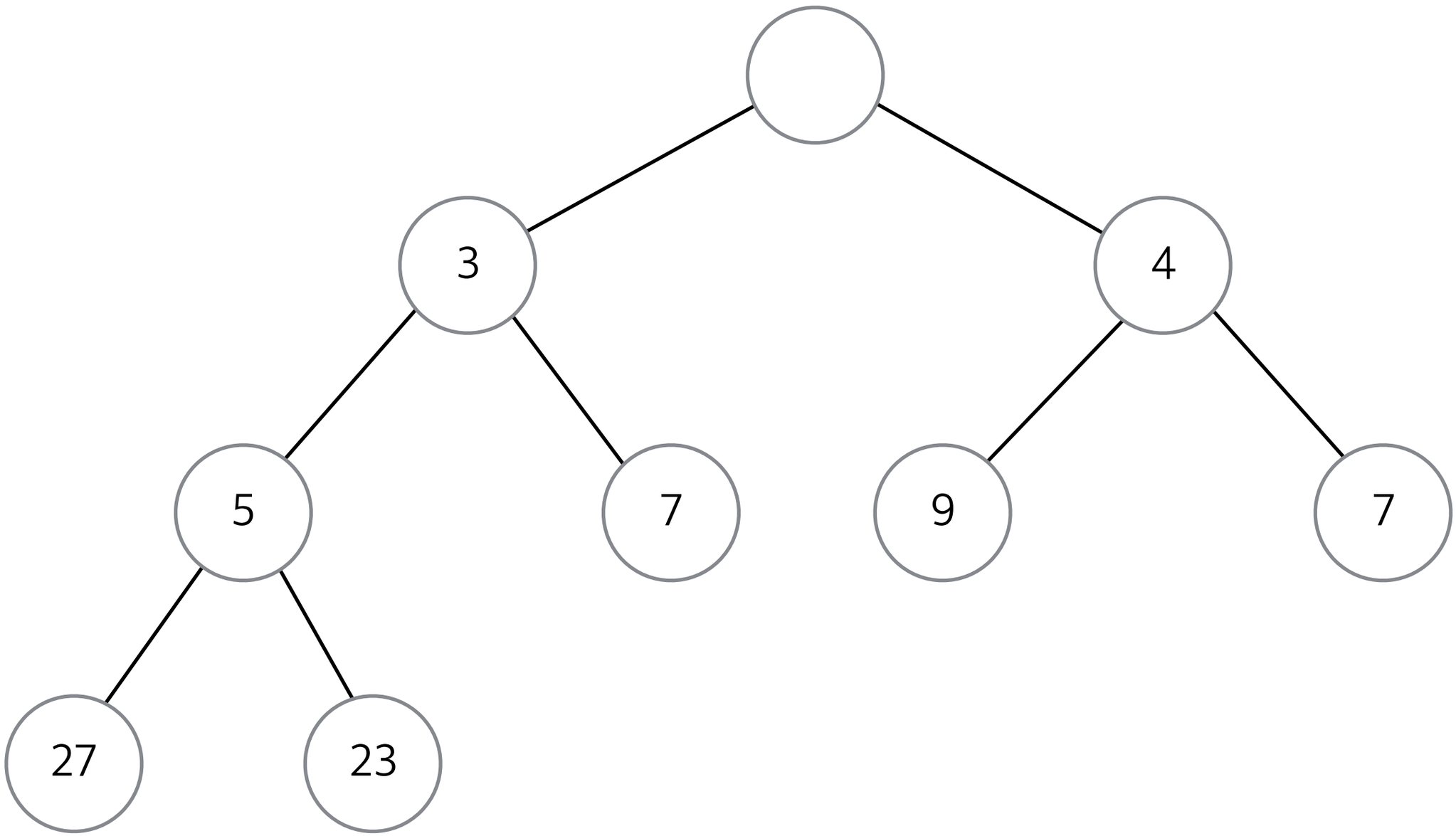
힙 : 값 삭제

heap.pop()



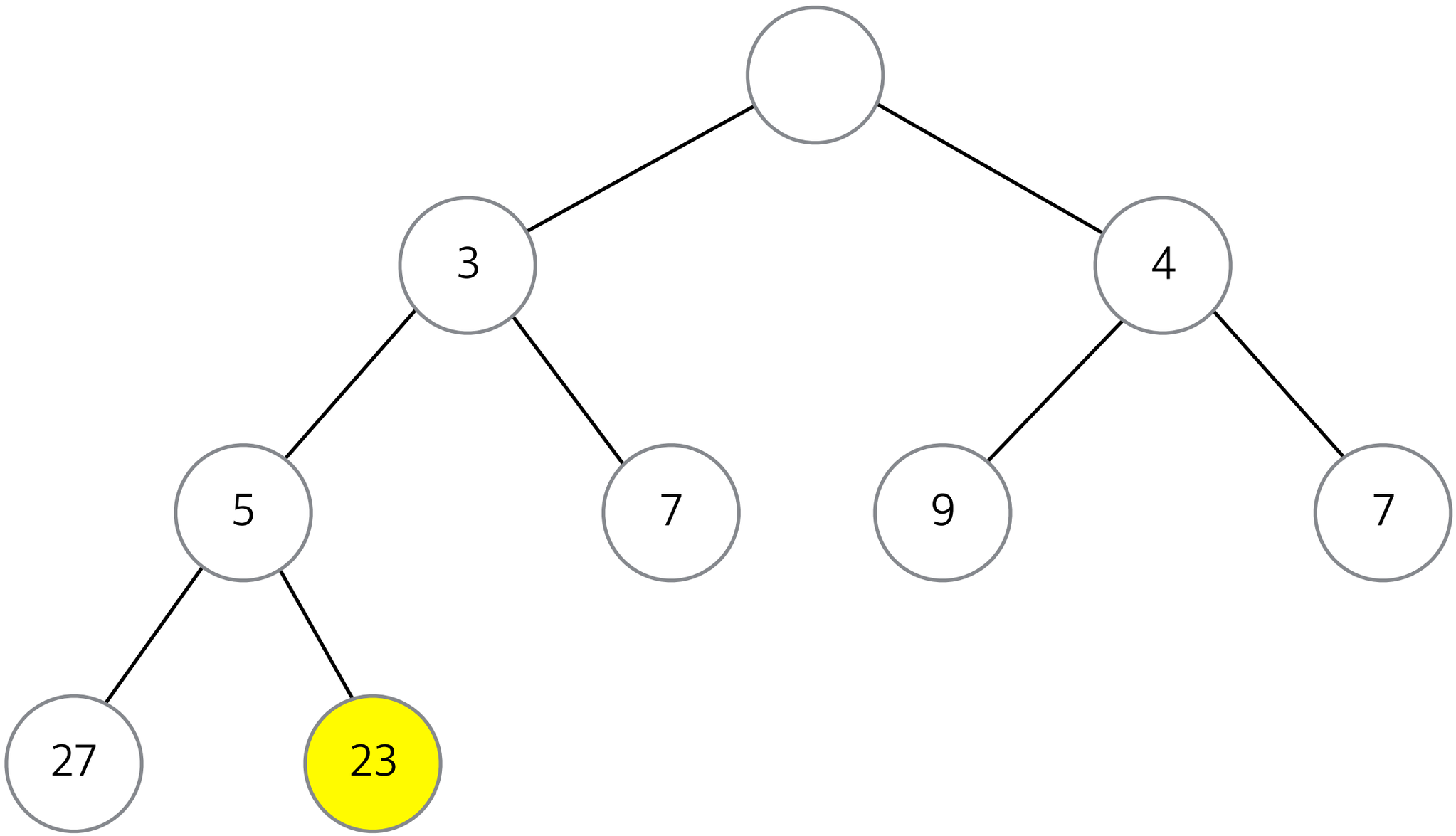
힙 : 값 삭제

heap.pop()



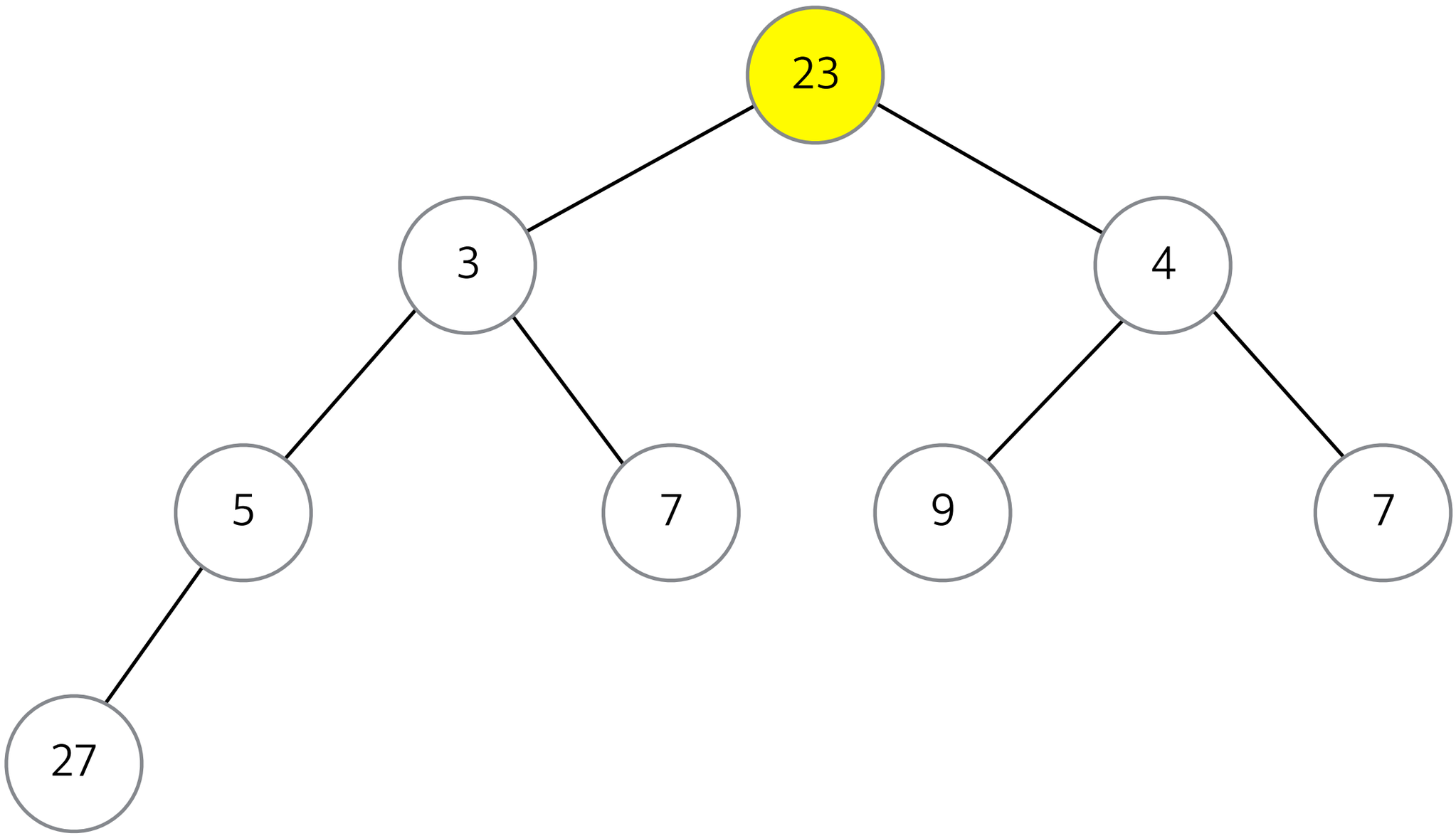
힙 : 값 삭제

heap.pop()



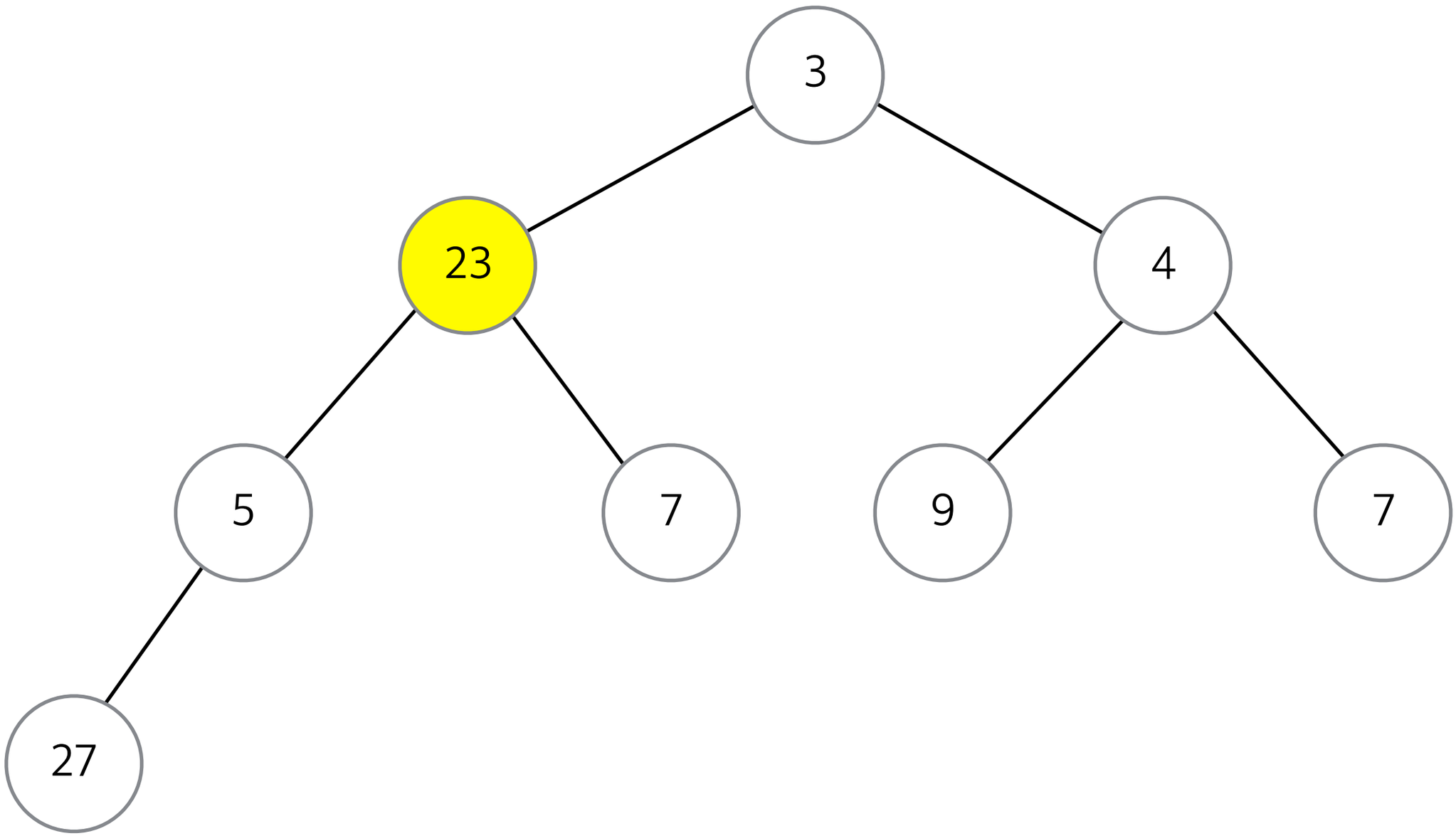
힙 : 값 삭제

heap.pop()



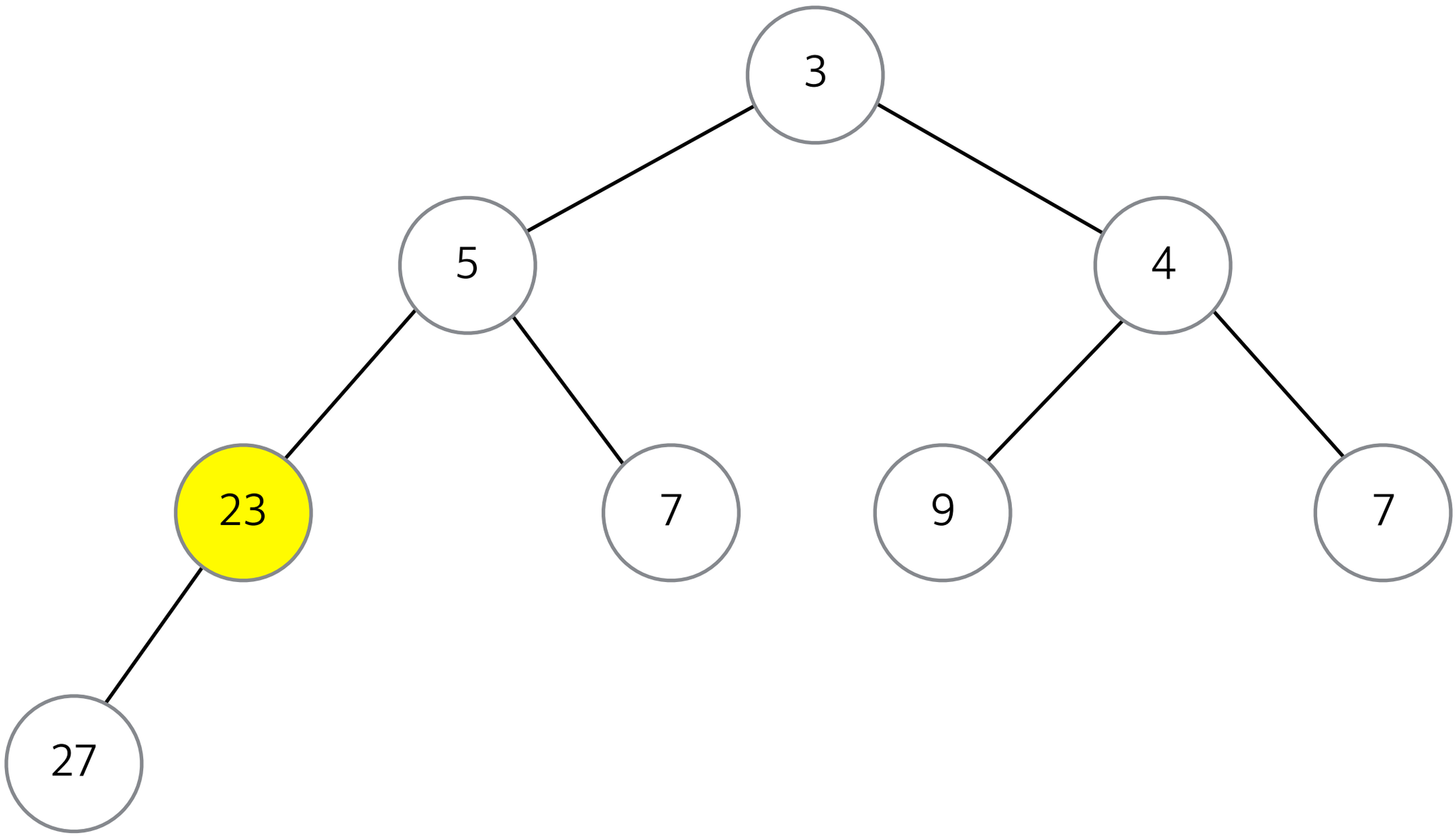
힙 : 값 삭제

heap.pop()



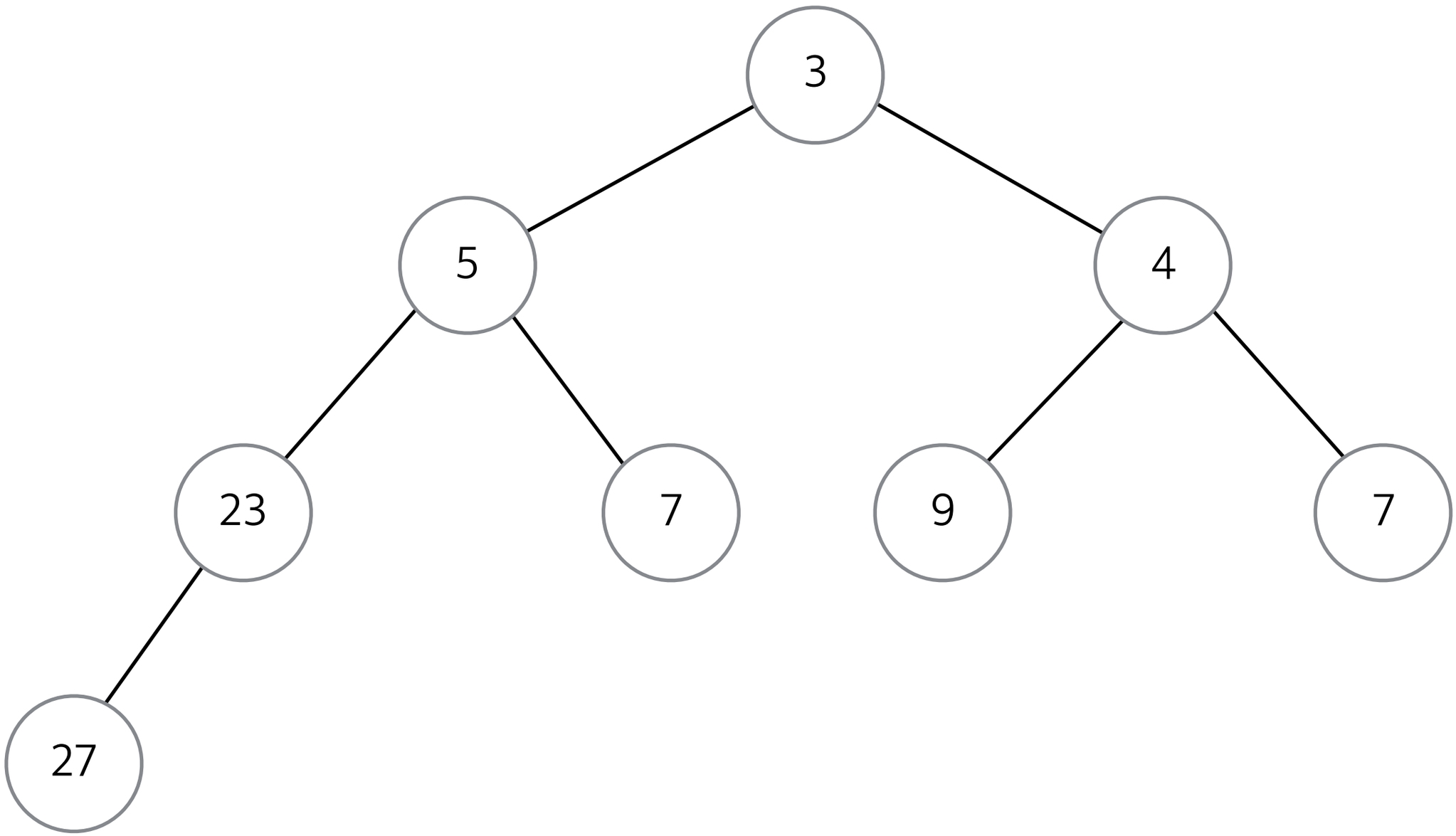
힙 : 값 삭제

heap.pop()



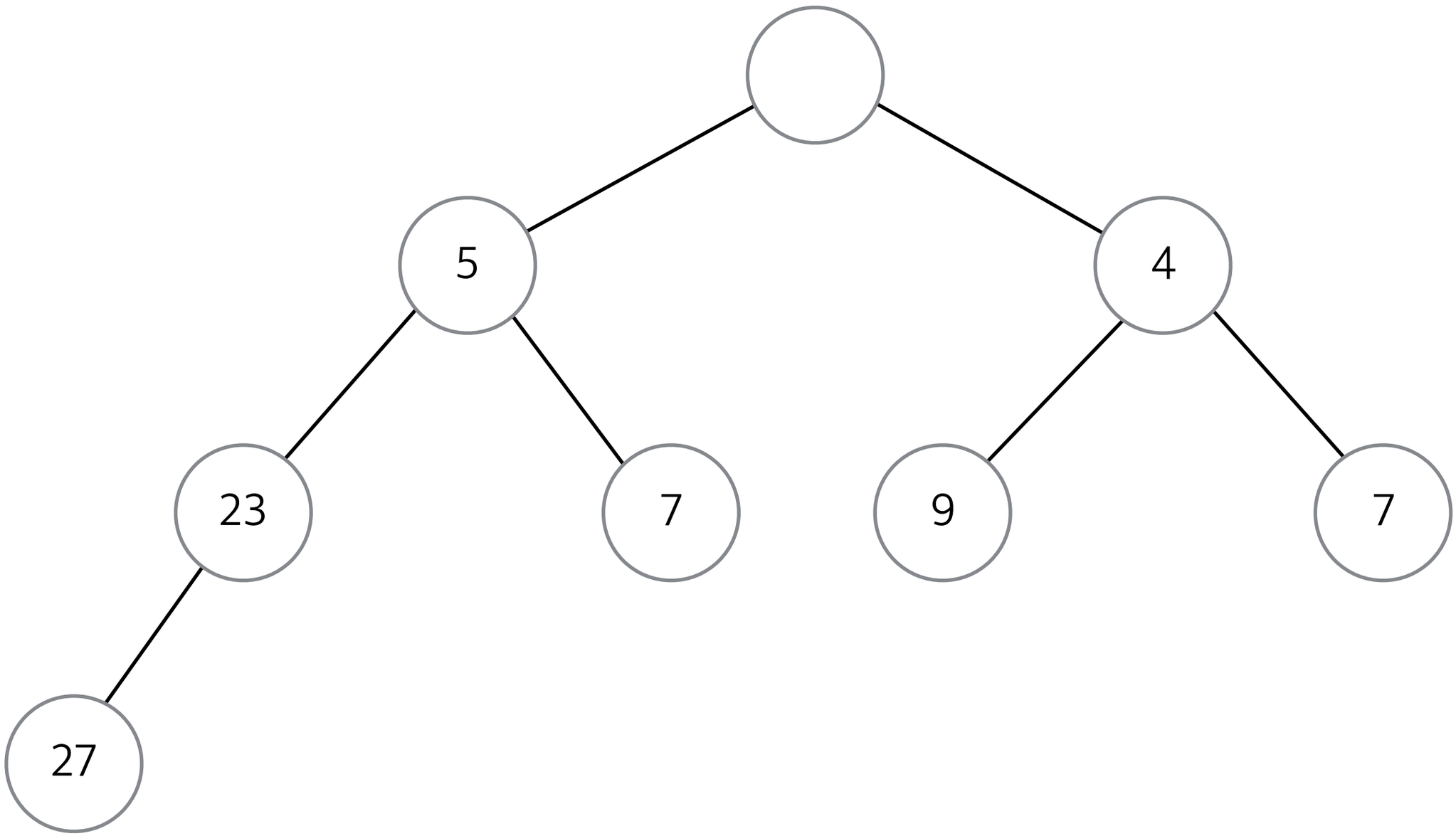
힙 : 값 삭제

heap.pop()



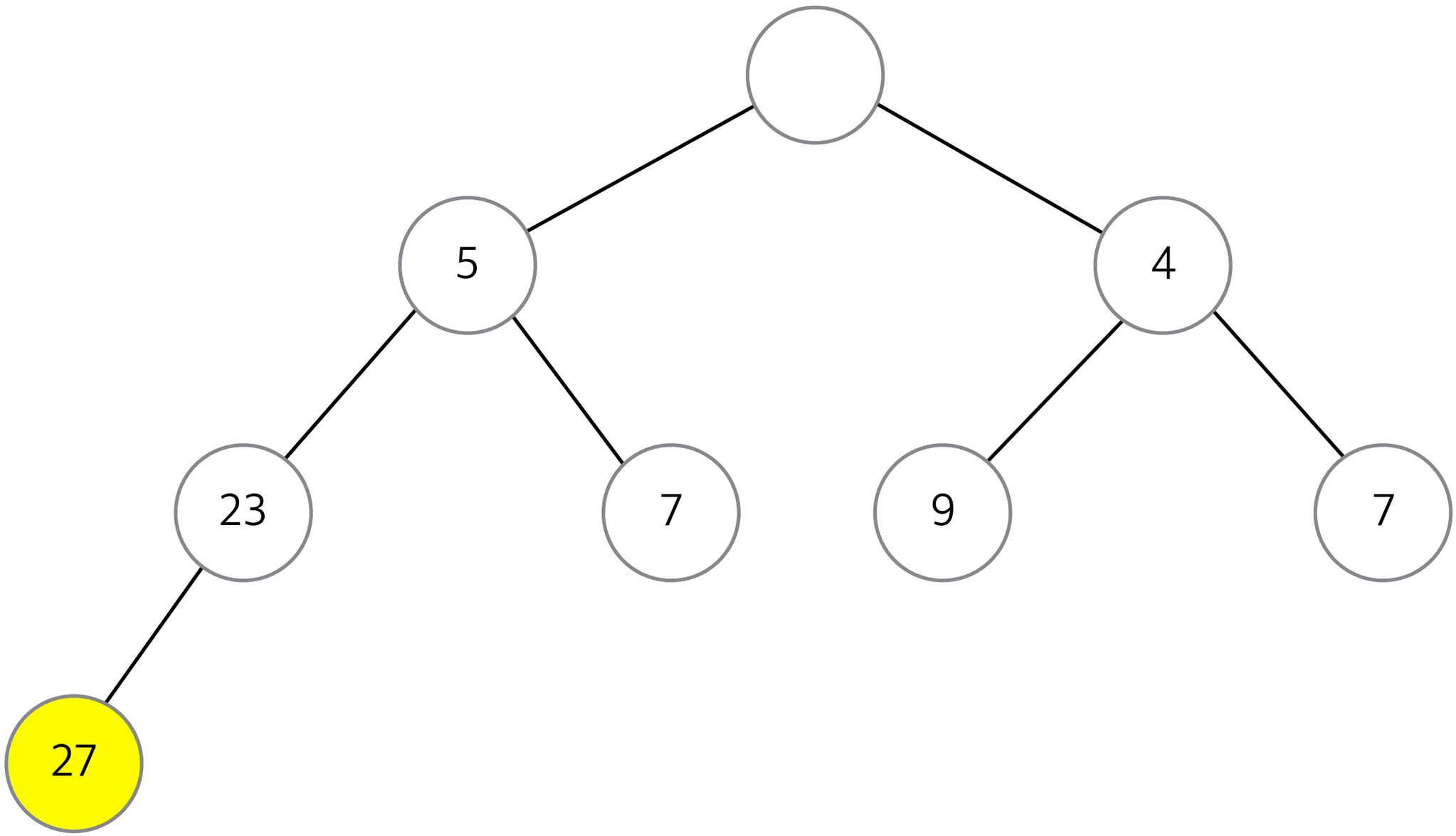
힙 : 값 삭제

heap.pop()



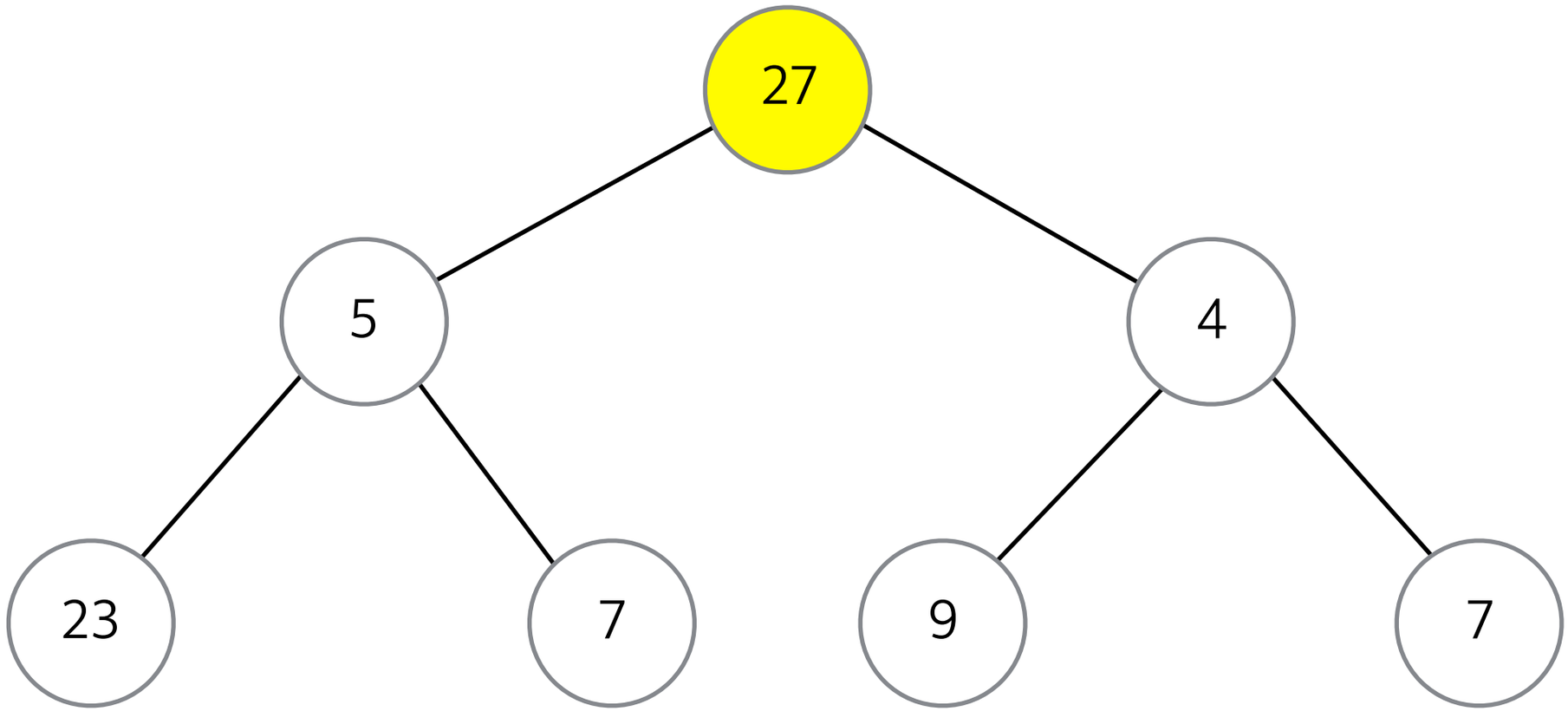
힙 : 값 삭제

heap.pop()



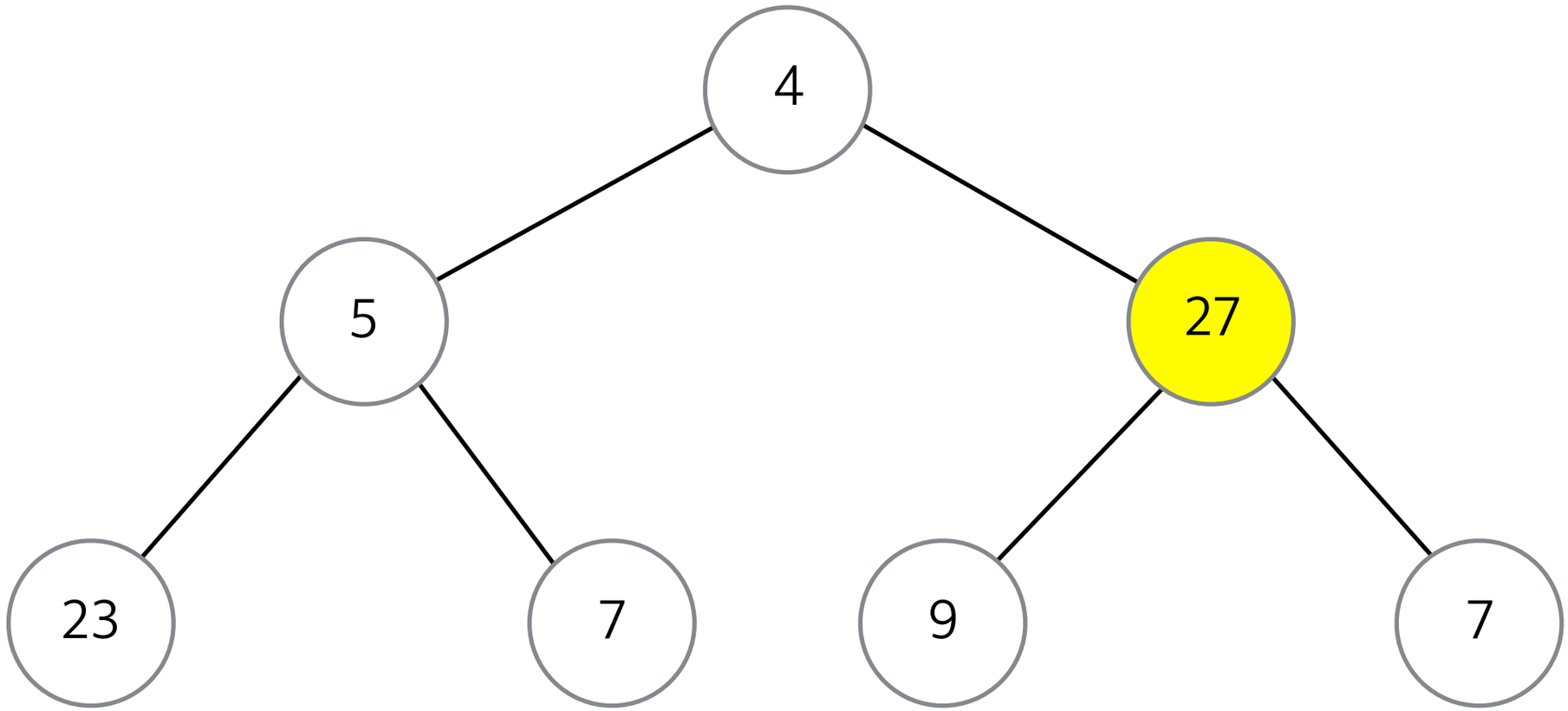
힙 : 값 삭제

heap.pop()



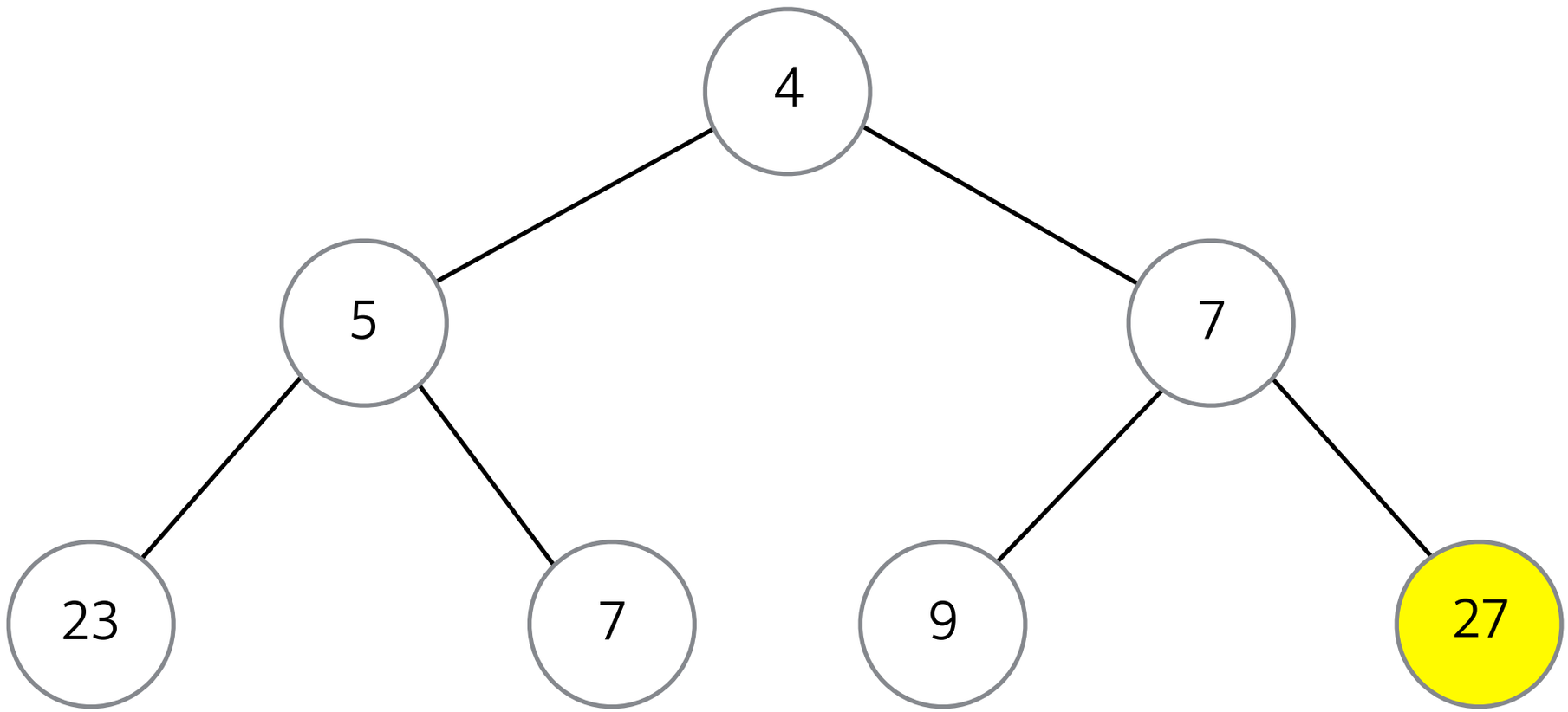
힙 : 값 삭제

heap.pop()



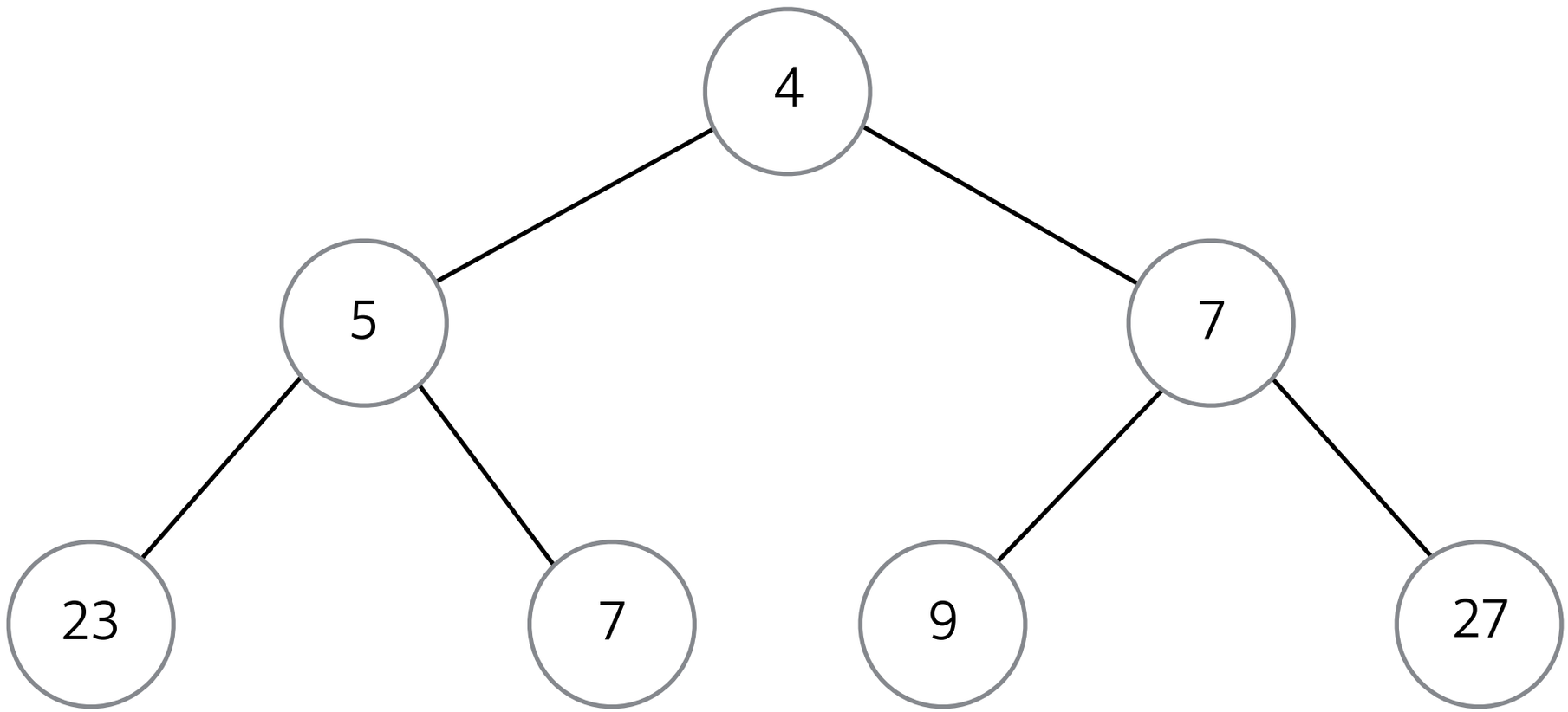
힙 : 값 삭제

heap.pop()

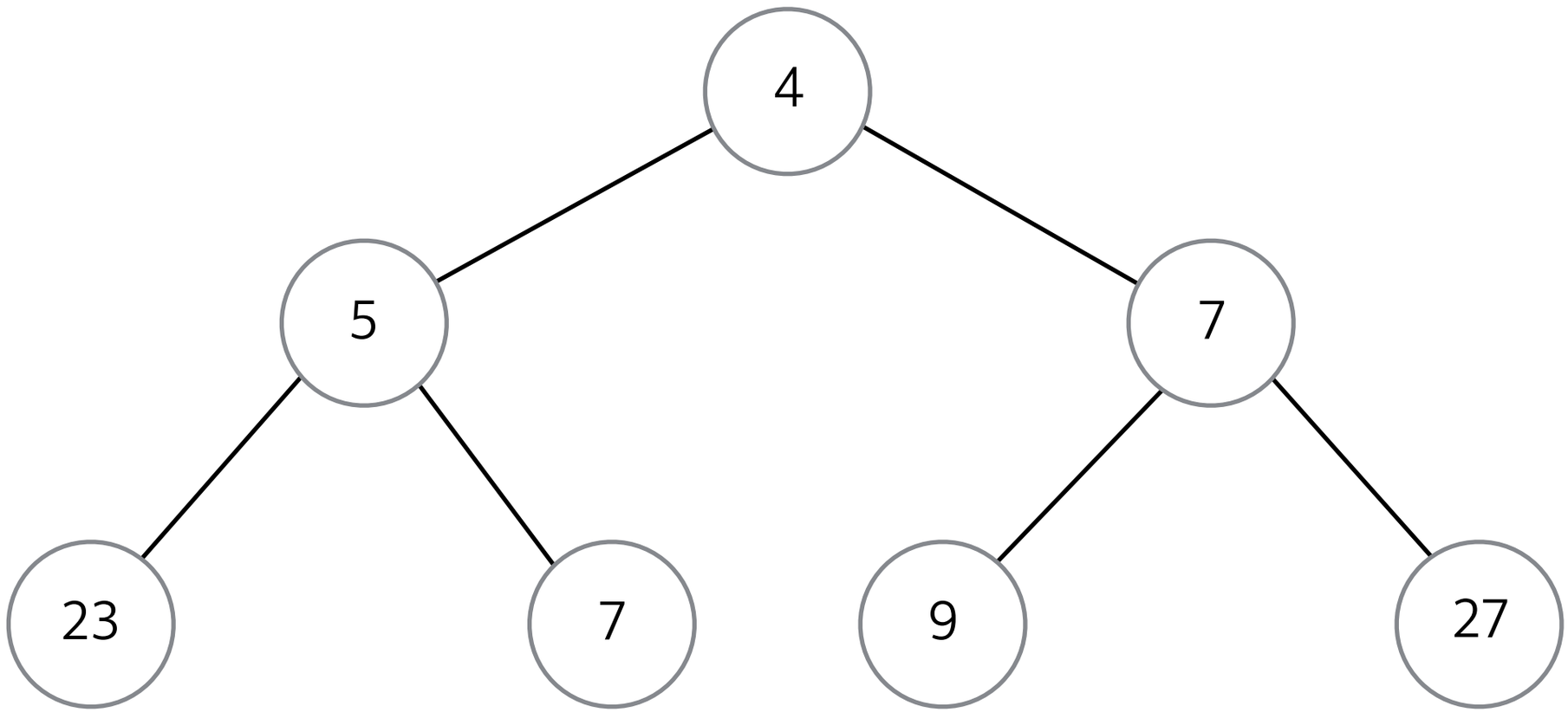


힙 : 값 삭제

heap.pop()

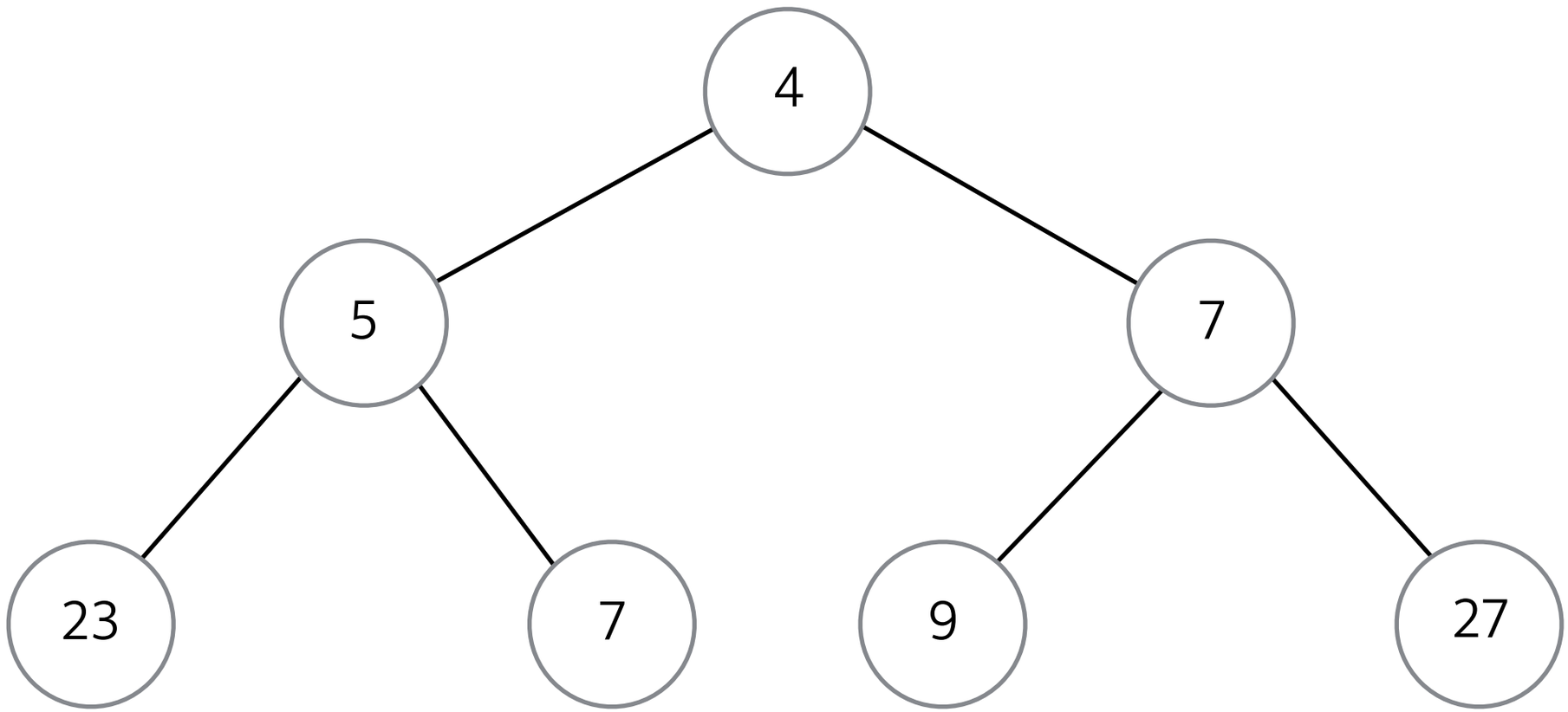


힙 : 값 삭제의 시간복잡도



힙 : 값 삭제의 시간복잡도

O(log n)



우선순위 큐의 구현 요약

	리스트	힙
값의 삽입	O(1)	O(log n)
값의 삭제	O(n)	O(log n)

[예제 2] 우선순위 큐 구현하기

힙을 구현하여 우선순위 큐를 완성하시오

시스템 입력의 예

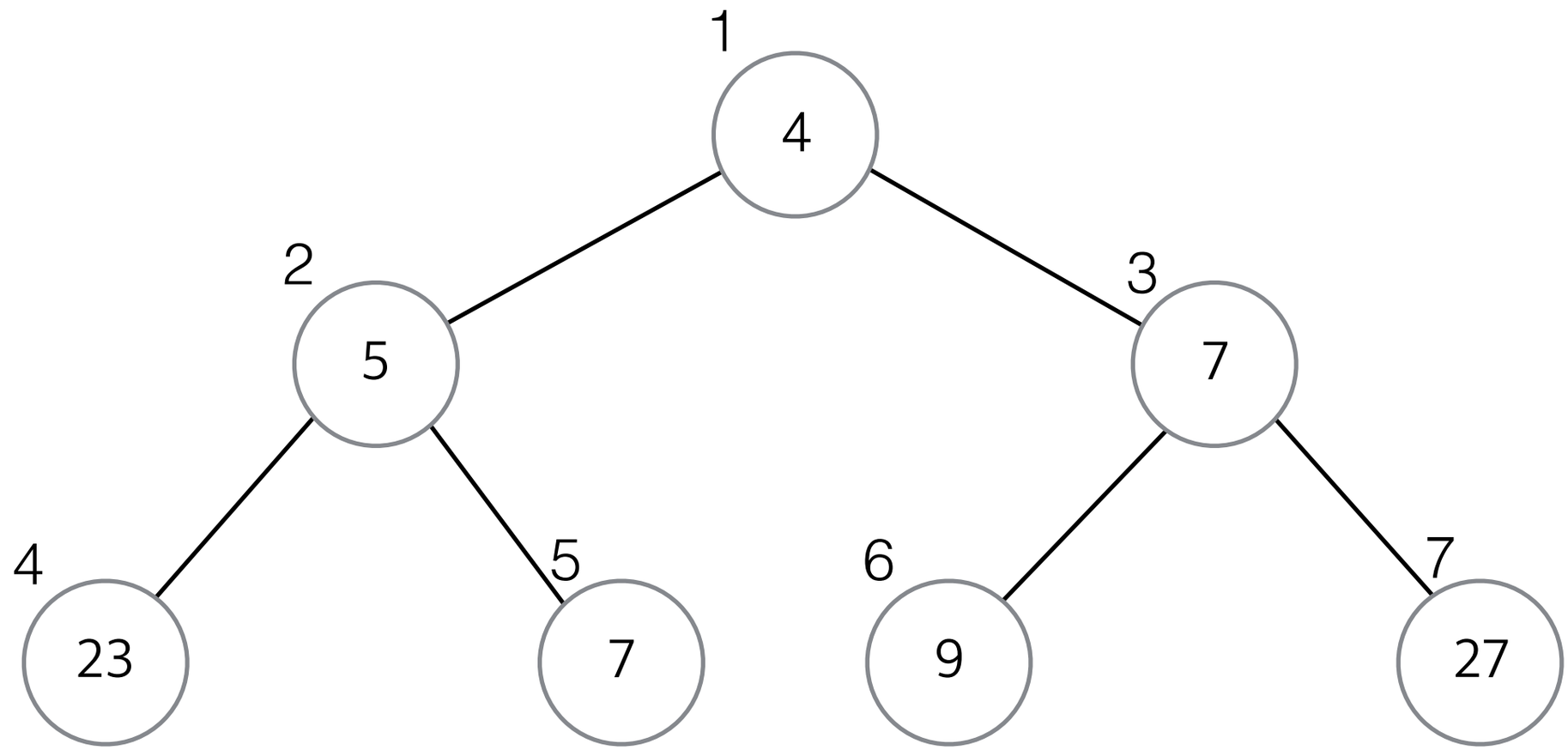
```
myPQ = priorityQueue()

myPQ.push(1)
myPQ.push(4)
myPQ.pop()

print(myPQ.top())
```

출력의 예

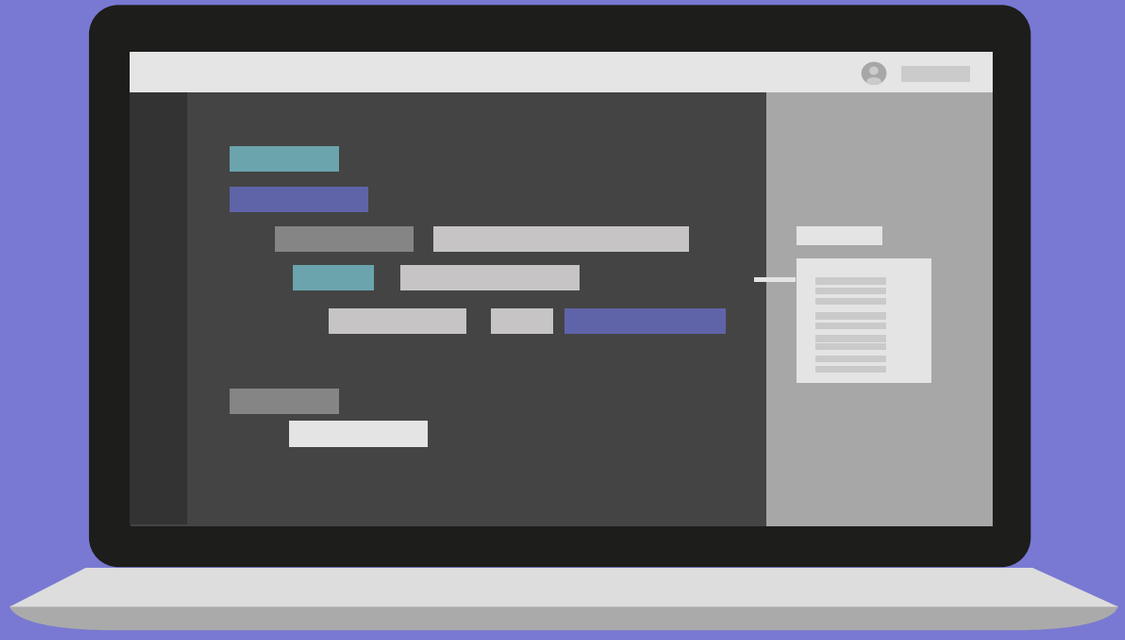
```
4
```

[예제 2] 우선순위 큐 구현하기

힙을 List를 이용하여 구현한다



[예제 2] 우선순위 큐 구현하기



요약

트리는 직접 구현할 일이 거의 없다

그래프의 구현을 배우면 트리 구현을 자연스럽게 습득

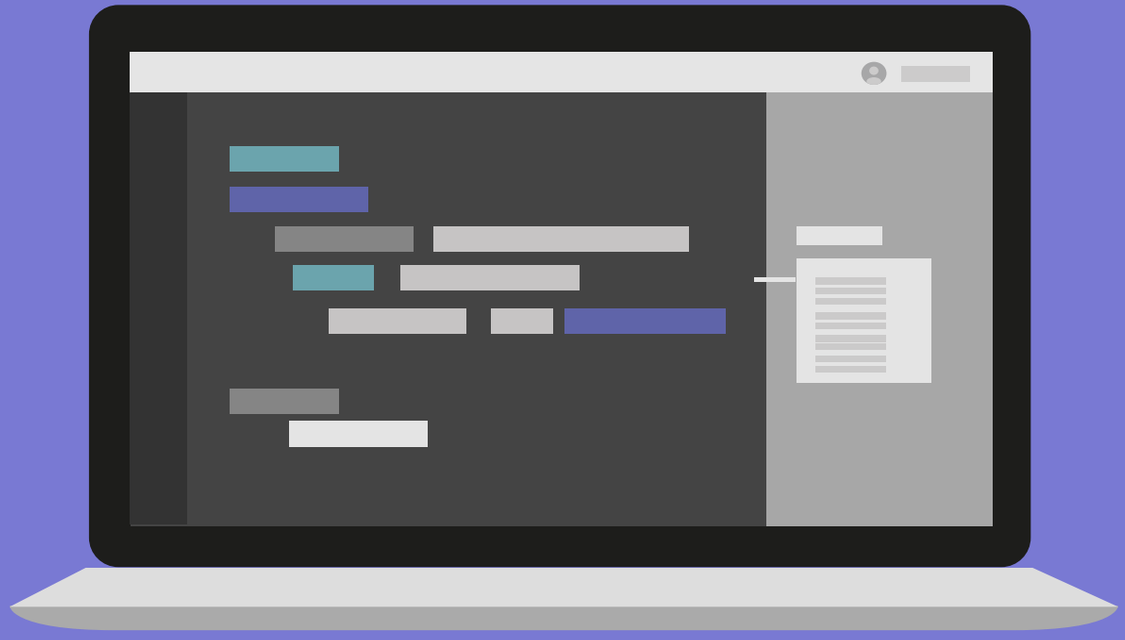
목적이 확실한 자료구조가 더 익히기 쉽다

힙은 스택/큐/트리처럼 심오한 의미가 있진 않다

트리 문제라도 트리를 구현할 필요가 없을 수 있다

굳이 트리를 저장하지 않아도 되는 경우가 있다

퀴즈 및 설문



감사합니다!

신현규

E-mail : hyungyu.sh@kaist.ac.kr

Kakao : yougatup

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

blog.naver.com/elicer