Fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

선수 지식 - 자료구조
우선순위 큐

우선순위 큐 | 다양한 알고리즘의 기본이 되는 자료구조 이해하기

강사 나동빈

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

선수 지식 - 자료구조

우선순위 큐

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

우선순위 큐(Priority Queue)

- 우선순위 큐는 우선순위에 따라서 데이터를 추출하는 자료구조다.
- 컴퓨터 운영체제, 온라인 게임 매칭 등에서 활용된다.
- 우선순위 큐는 일반적으로 **힙(heap)을 이용해 구현**한다.

자료구조	추출되는 데이터
스택(stack)	가장 나중에 삽입된 데이터
큐(queue)	가장 먼저 삽입된 데이터
우선순위 큐(priority queue)	가장 우선순위가 높은 데이터

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

우선순위 큐를 구현하는 방법

- 우선순위 큐는 다양한 방법으로 구현할 수 있다.
- 데이터의 개수가 N개일 때, 구현 방식에 따른 시간 복잡도는 다음과 같다.

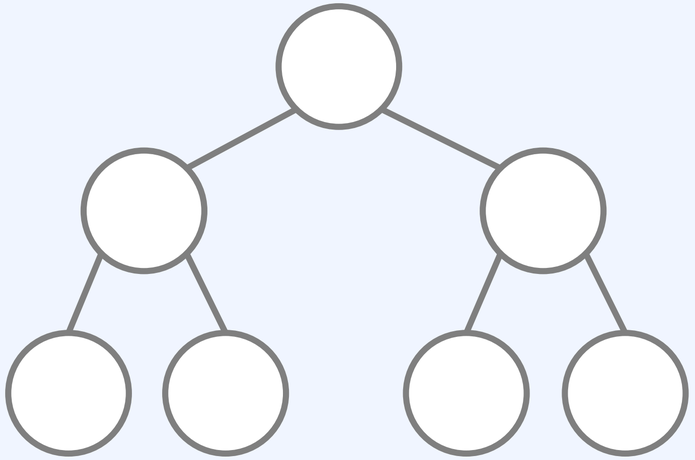
우선순위 큐 구현 방식	삽입 시간	삭제 시간
리스트 자료형	$O(1)$	$O(N)$
힙(Heap)	$O(logN)$	$O(logN)$

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

우선순위 큐를 구현하는 방법

- 일반적인 형태의 큐는 선형적인 구조를 가진다.
- 반면에 **우선순위 큐**는 **이진 트리**(binary tree) 구조를 사용하는 것이 일반적이다.

일반적인 큐

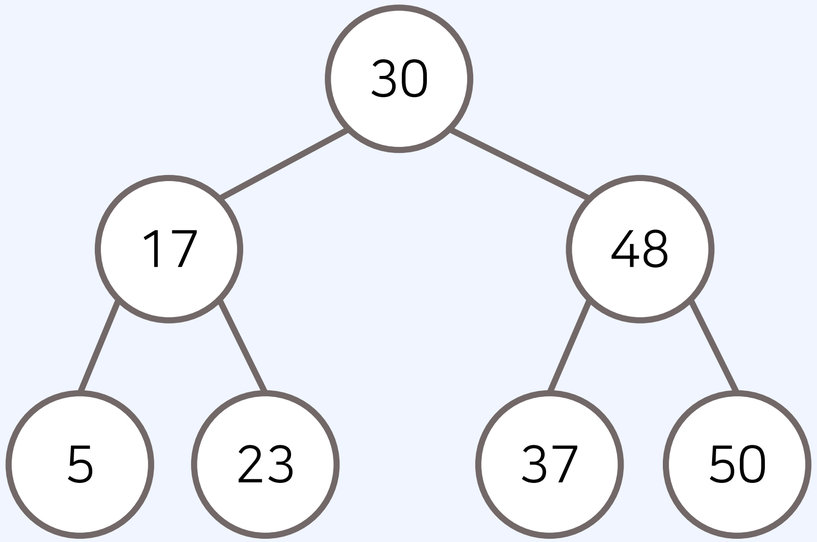


우선순위 큐

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

이진 트리(Binary Tree)

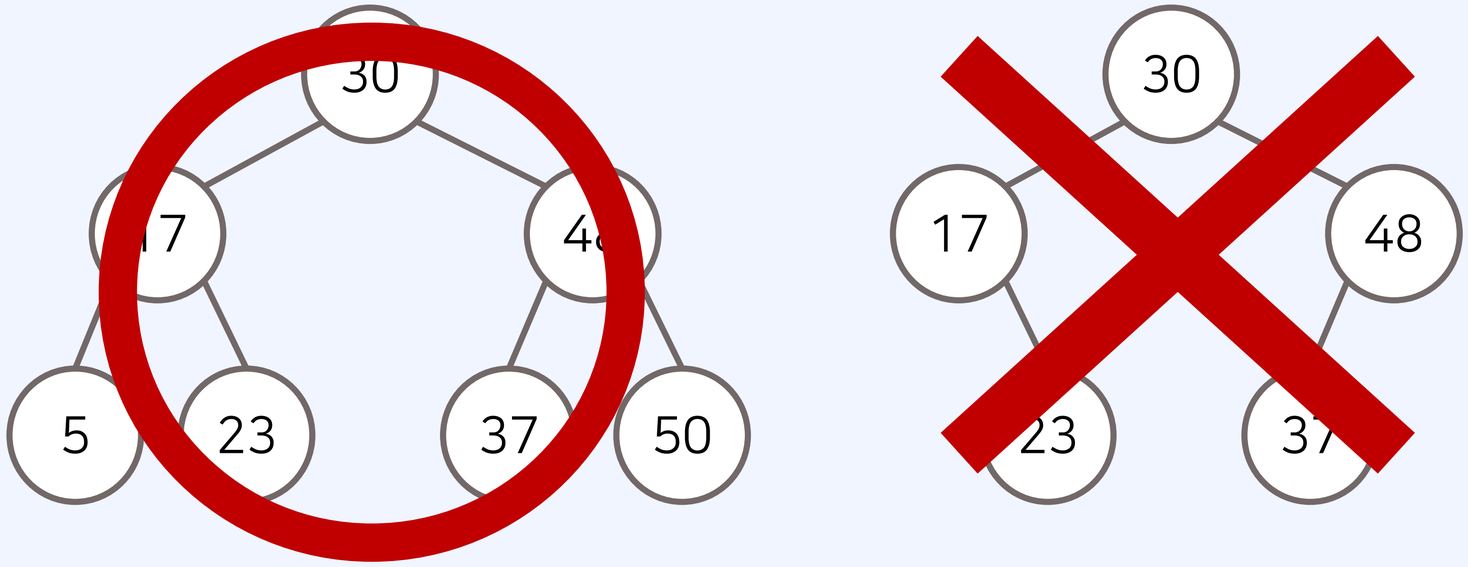
- 이진 트리(binary tree)는 최대 2개까지의 자식을 가질 수 있다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

포화 이진 트리(Full Binary Tree)

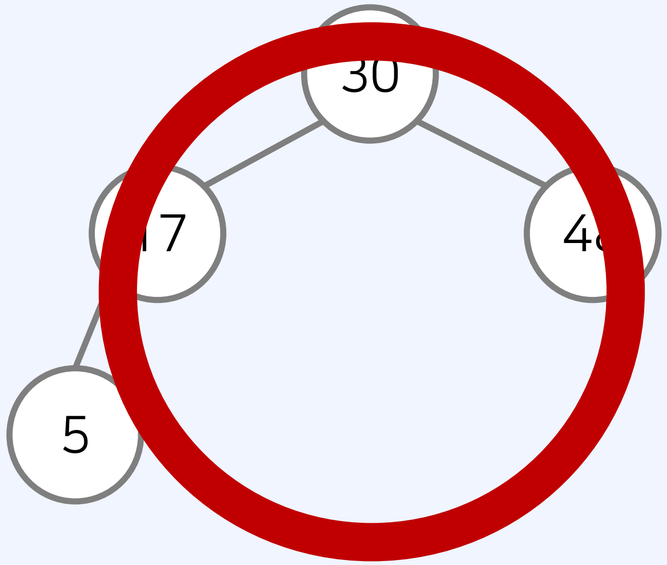
- **포화 이진 트리**는 리프 노드를 제외한 모든 노드가 두 자식을 가지고 있는 트리다.

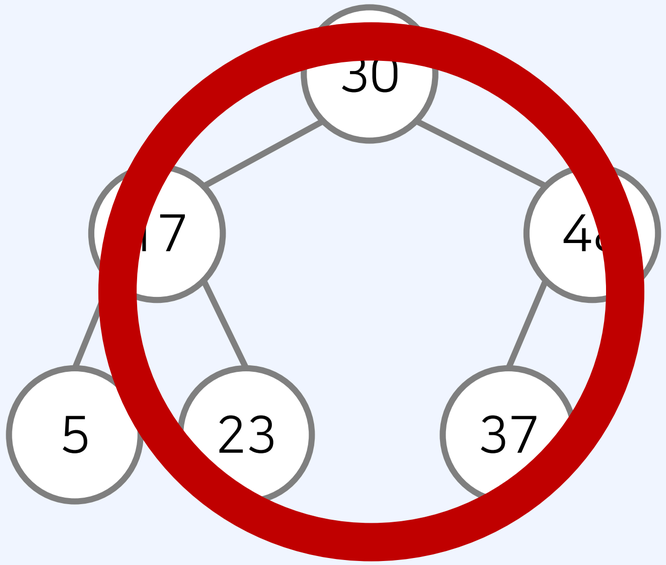


fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

완전 이진 트리(Complete Binary Tree)

- **완전 이진 트리**는 모든 노드가 왼쪽 자식부터 차근차근 채워진 트리다.

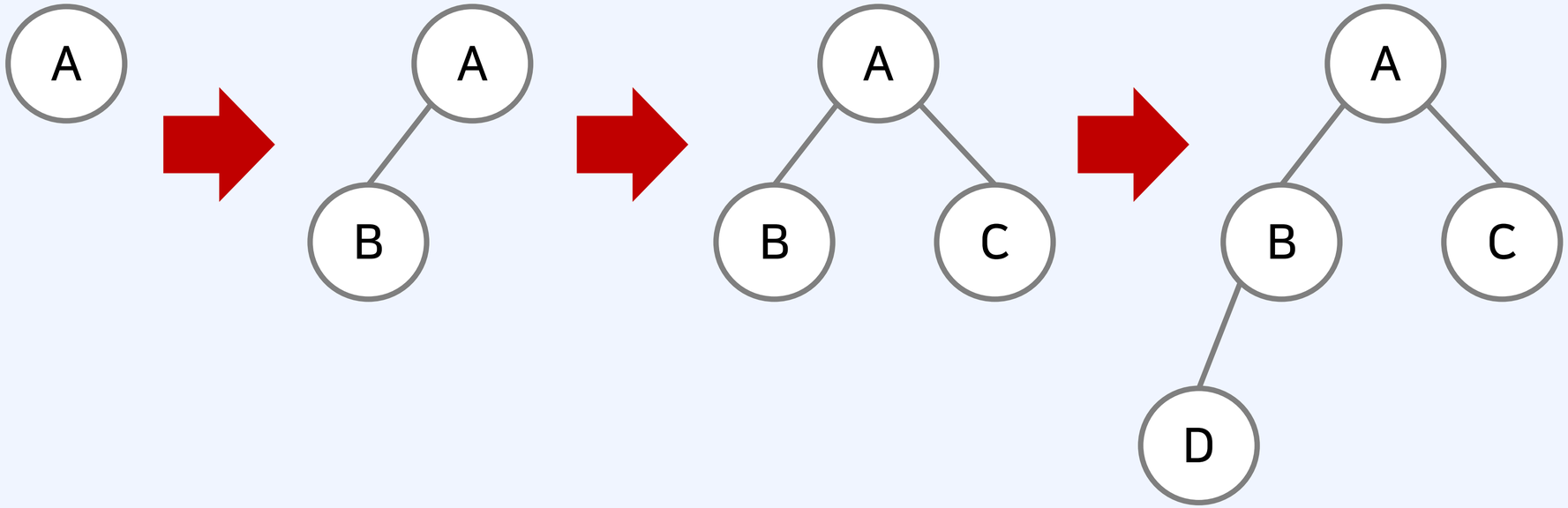




fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

완전 이진 트리(Complete Binary Tree)

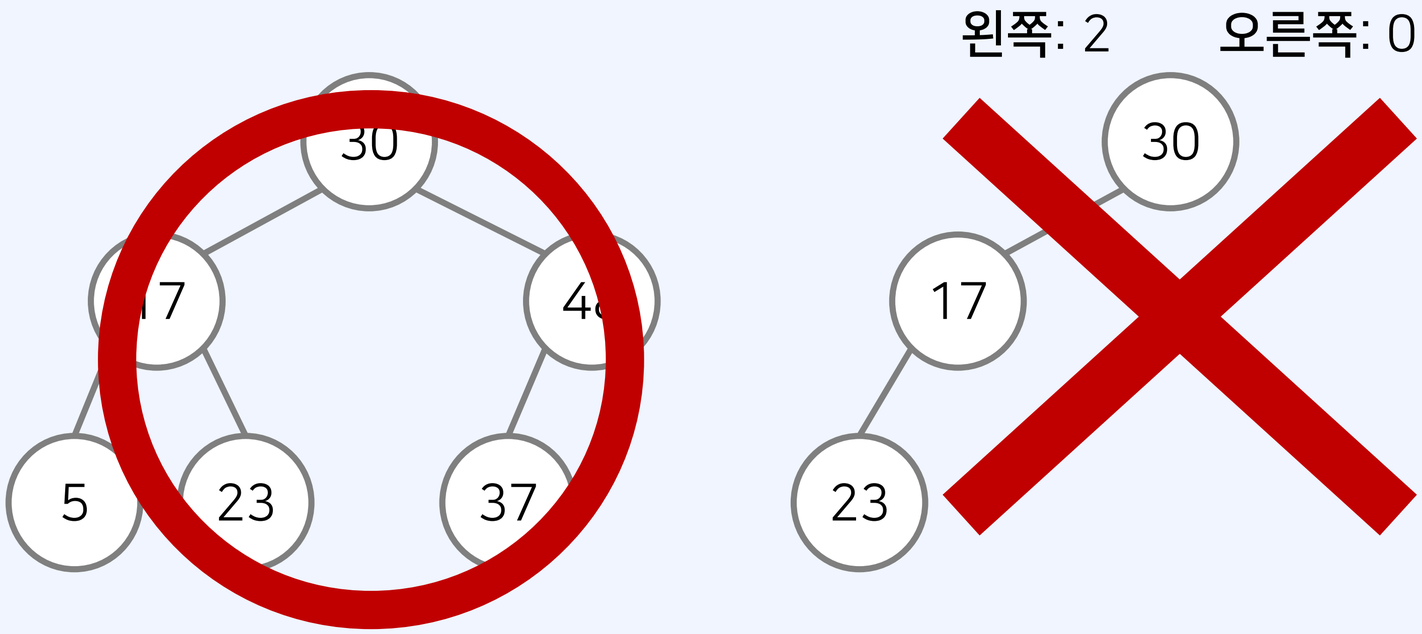
- **완전 이진 트리**는 모든 노드가 왼쪽 자식부터 차근차근 채워진 트리다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

높이 균형 트리(Height Balanced Tree)

- 왼쪽 자식 트리와 오른쪽 자식 트리의 높이가 1 이상 차이 나지 않는 트리다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

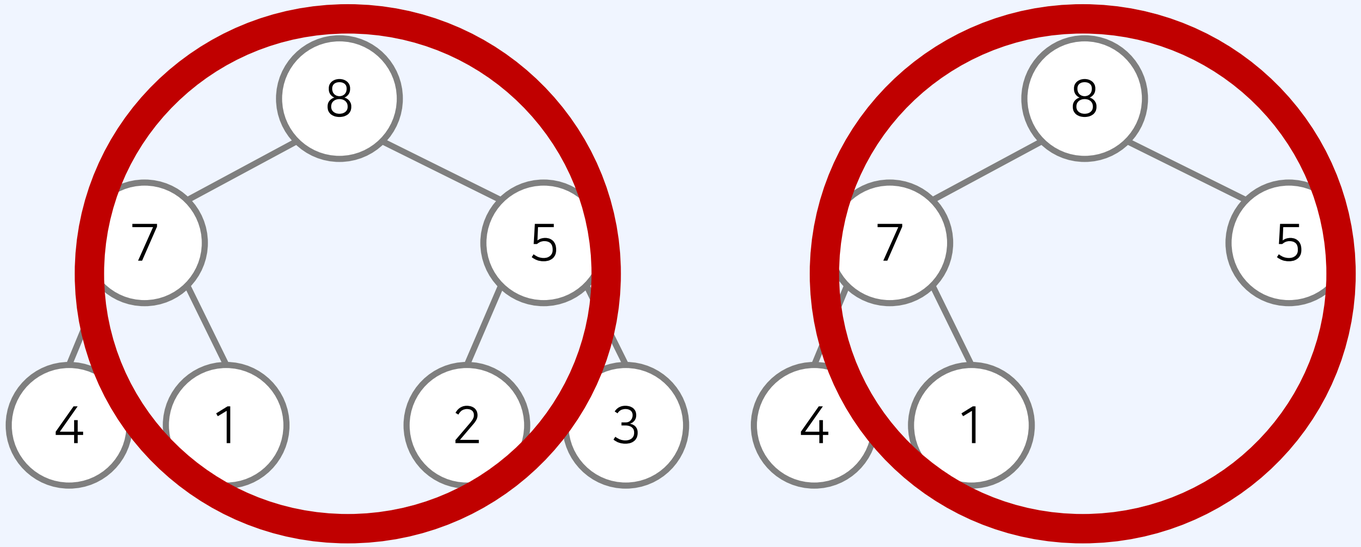
힙(Heap)

- 힙(Heap)은 원소들 중에서 최댓값 혹은 최솟값을 빠르게 찾아내는 자료구조다.
- **최대 힙(Max Heap)**: 값이 큰 원소부터 추출한다.
- **최소 힙(Min Heap)**: 값이 작은 원소부터 추출한다.
- 힙은 원소의 삽입과 삭제를 위해 $O(logN)$의 수행 시간을 요구한다.
- 단순한 N개의 데이터를 힙에 넣었다가 모두 꺼내는 작업은 정렬과 동일하다.
- 이 경우 시간 복잡도는 $O(NlogN)$이다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

최대 힙(Max Heap)

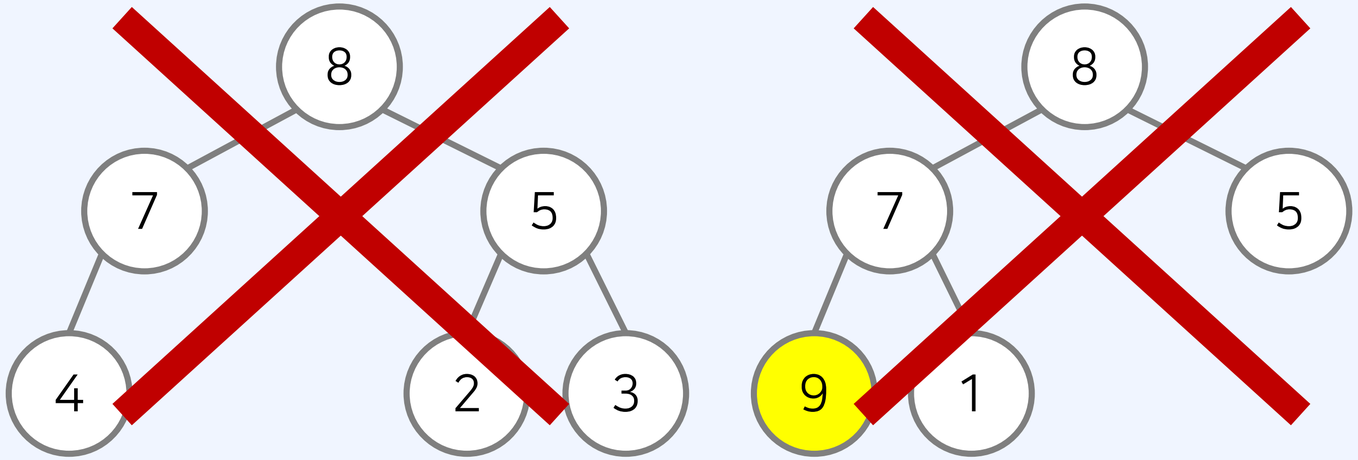
- **최대 힙(max heap)**은 부모 노드가 자식 노드보다 값이 큰 **완전 이진 트리**를 의미한다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

최대 힙(Max Heap)

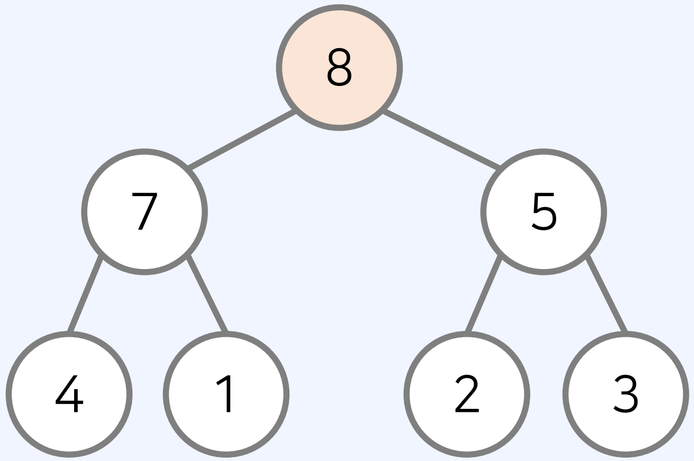
- **최대 힙(max heap)**은 부모 노드가 자식 노드보다 값이 큰 **완전 이진 트리**를 의미한다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

최대 힙(Max Heap)

- 최대 힙(max heap)의 루트 노드는 전체 트리에서 가장 큰 값을 가진다는 특징이 있다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

힙(Heap)의 특징

- 힙은 **완전 이진 트리** 자료구조를 따른다.
- 힙에서는 우선순위가 높은 노드가 루트(root)에 위치한다.

1. **최대 힙(max heap)**

- 부모 노드의 키 값이 자식 노드의 키 값보다 항상 크다.
- 루트 노드가 가장 크며, 값이 큰 데이터가 우선순위를 가진다.

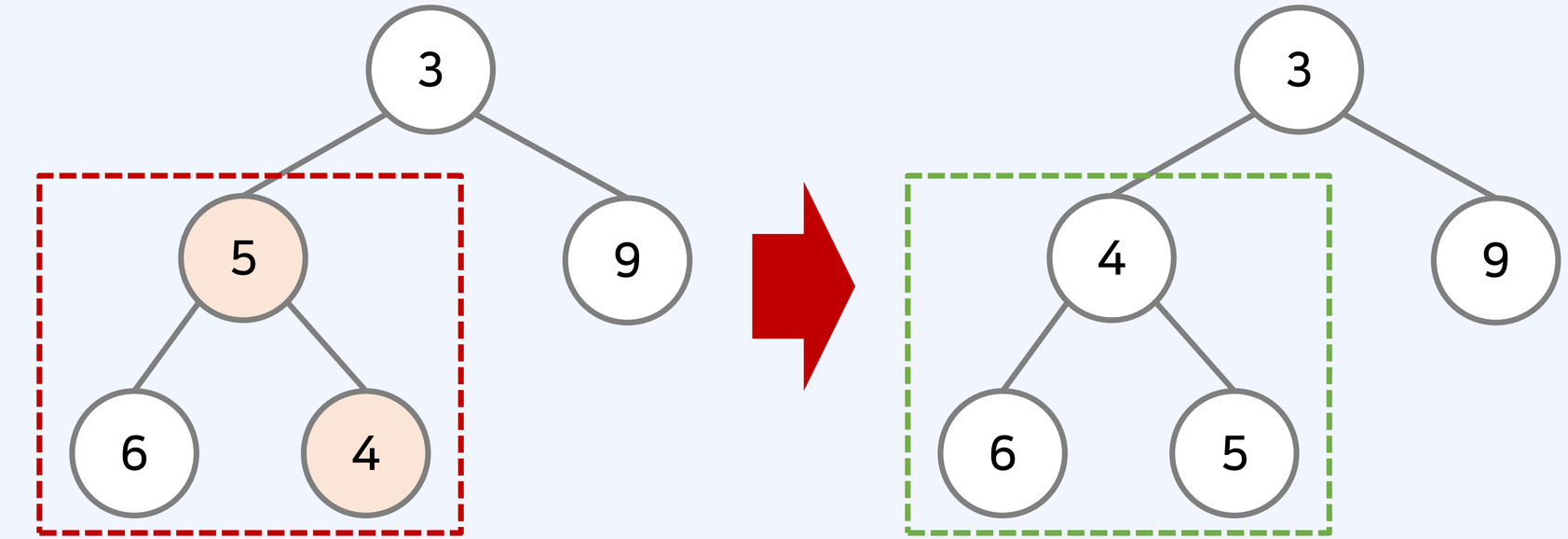
2. **최소 힙(min heap)**

- 부모 노드의 키 값이 자식 노드의 키 값보다 항상 작다.
- 루트 노드가 가장 작으며, 값이 작은 데이터가 우선순위를 가진다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

최소 힙 구성 함수: Heapify

- **(상향식)** 부모로 거슬러 올라가며, 부모보다 자신이 더 작은 경우에 위치를 교체한다.



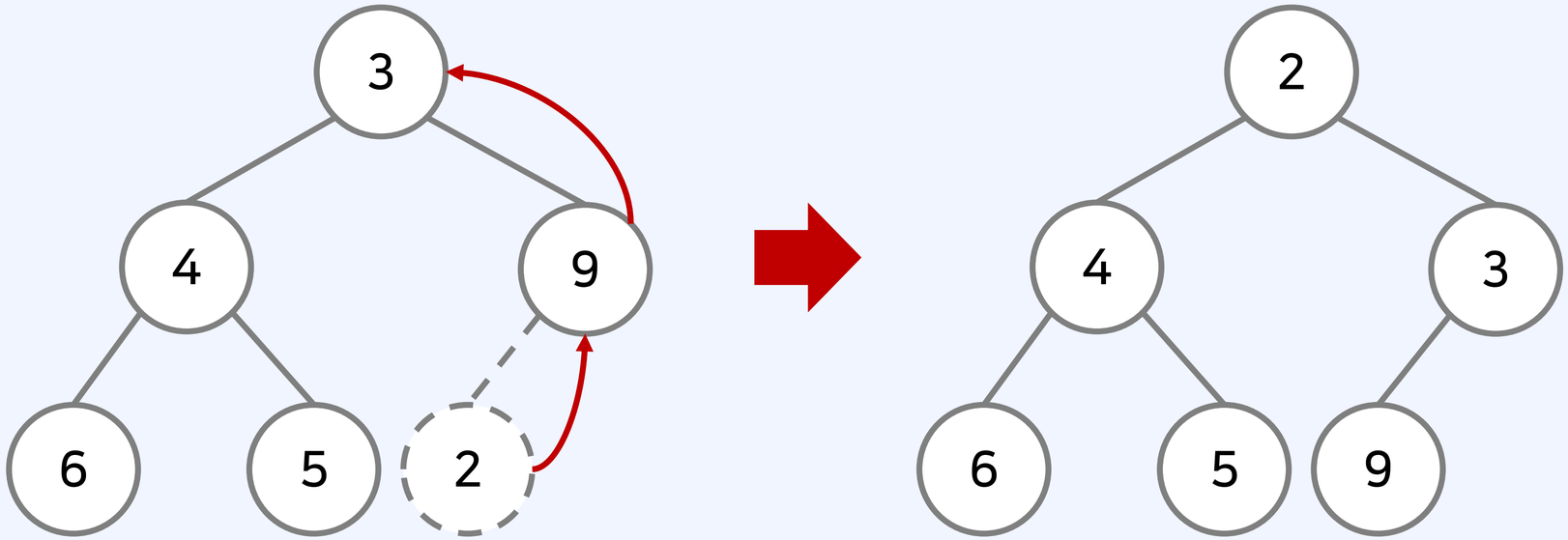
최소 힙 성질을
만족하지 않는 서브 트리

최소 힙 성질을
만족하는 서브 트리

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

힙에 새로운 원소가 삽입될 때

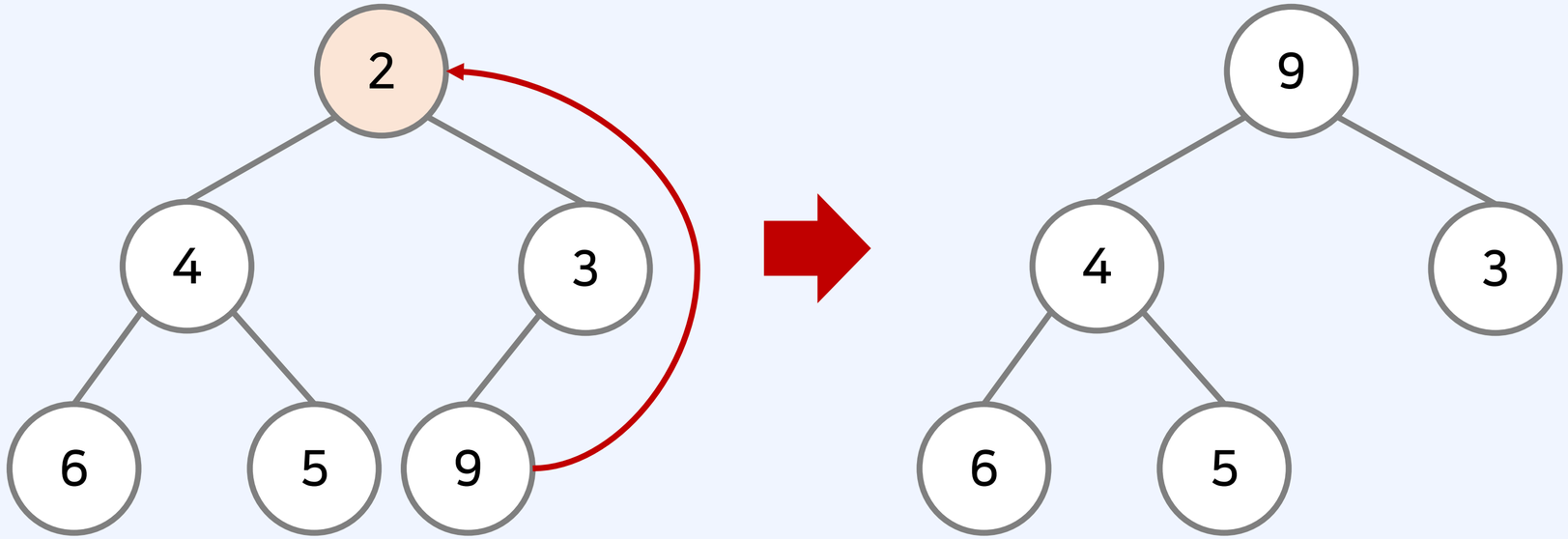
- **(상향식)** 부모로 거슬러 올라가며, 부모보다 자신이 더 작은 경우에 위치를 교체한다.
- 새로운 원소가 삽입되었을 때 $O(logN)$의 시간 복잡도로 힙 성질을 유지하도록 할 수 있다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

힙에 새로운 원소가 삭제될 때

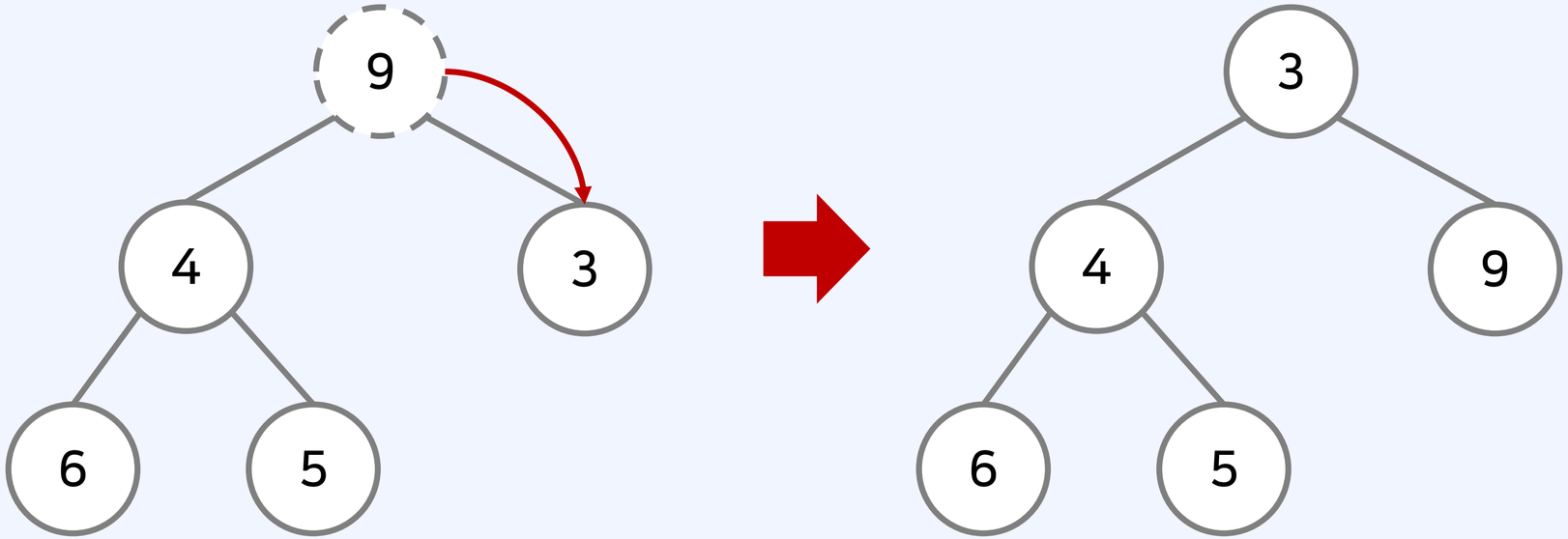
- 원소가 제거되었을 때 $O(logN)$의 시간 복잡도로 힙 성질을 유지하도록 할 수 있다.
- 원소를 제거할 때는 가장 마지막 노드가 루트 노드의 위치에 오도록 한다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

힙에 새로운 원소가 삭제될 때

- 원소가 제거되었을 때 $O(logN)$의 시간 복잡도로 힙 성질을 유지하도록 할 수 있다.
- 이후에 루트 노드에서부터 하향식으로(더 작은 자식 노드로) Heapify()를 진행한다.



fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

힙(Heap)의 특징

- 힙의 삽입과 삭제 연산을 수행할 때를 고려해 보자.
- 직관적으로, 거슬러 갈 때마다 처리해야 하는 범위에 포함된 원소의 개수가 절반씩 줄어든다.
- 따라서 삽입과 삭제에 대한 시간 복잡도는 $O(logN)$이다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

파이썬의 힙(Heap) 라이브러리

- 파이썬에서는 **힙(Heap) 라이브러리**를 제공한다.
- $heapq$ 라이브러리의 삽입 및 삭제에 대한 시간 복잡도는 모두 $O(logN)$이다.

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

heapq 라이브러리 – 힙 초기화

- 단순히 하나의 빈 리스트를 만들면, 그것을 **힙(heap)** 자료구조로 사용할 수 있다.

```
import heapq

heap = []
print(heap)
```

[실행 결과]

```
[]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

heapq 라이브러리 – 삽입(push)과 추출(pop)

- 원소의 삽입: $heappush()$ 메서드
- 원소의 추출: $heappop()$ 메서드

```
import heapq

heap = []

heapq.heappush(heap, 7)
heapq.heappush(heap, 4)
heapq.heappush(heap, 5)
heapq.heappush(heap, 8)

while heap:
    element = heapq.heappop(heap)
    print(element, end=" ")
```

[실행 결과]

```
4 5 7 8
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

heapq 라이브러리 – 최솟값 구하기

- 원소의 삽입: $heappush()$ 메서드
- 원소의 추출: $heappop()$ 메서드

```python
import heapq

heap = []

heapq.heappush(heap, 7)
heapq.heappush(heap, 4)
heapq.heappush(heap, 5)
heapq.heappush(heap, 8)

print(heap[0])
```

[실행 결과]

```
4
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

heapq 라이브러리 – 리스트를 힙으로 변환

- $heappush()$ 메서드를 이용해 하나씩 원소를 삽입하지 않을 수 있다.
- $heapify()$ 메서드는 새로운 리스트를 반환하지 않고, 리스트 내부를 직접 수정한다.

```python
import heapq

heap = [9, 1, 5, 4, 3, 8, 7]
heapq.heapify(heap)

while heap:
    element = heapq.heappop(heap)
    print(element, end=" ")
```

[실행 결과]

```
1 3 4 5 7 8 9
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

heapq 라이브러리 – 최대 힙(Max Heap)

- 파이썬의 heapq 라이브러리는 기본적으로 최소 힙 기능을 제공한다.
- 최대 힙을 위해서는 ① **삽입**과 ② **추출**할 때 키(key)에 음수(-) 부호를 취한다.

```python
import heapq

arr = [9, 1, 5, 4, 3, 8, 7]
heap = []

for x in arr:
    heapq.heappush(heap, -x)

while heap:
    x = -heapq.heappop(heap)
    print(x, end=" ")
```

[실행 결과]

```
9 8 7 5 4 3 1
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

활용 예시 ① 힙 정렬(Heap Sort)

- 단순히 힙에 원소를 넣었다가 꺼내는 것 만으로도 정렬을 수행할 수 있다.

```
import heapq


def heap_sort(arr):
    heap = []
    for x in arr:
        heapq.heappush(heap, x)
    result = []
    while heap:
        x = heapq.heappop(heap)
        result.append(x)
    return result


print(heap_sort([9, 1, 5, 4, 3, 8, 7]))
```

[실행 결과]

```
[1, 3, 4, 5, 7, 8, 9]
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

활용 예시 ② N번째로 작은 값

- 최소 힙이나 최대 힙을 사용하여 n번째로 작은 값이나 n번째로 큰 값을 얻을 수 있다.
- 힙(heap)을 만든 뒤에 추출(pop) 함수를 n번 호출한다.

```python
import heapq


def n_smallest(n, arr):
    heap = []
    for x in arr:
        heapq.heappush(heap, x)
    result = None
    for _ in range(n):
        result = heapq.heappop(heap)
    return result


arr = [9, 1, 5, 4, 3, 8, 7]
print(n_smallest(3, arr))
```

[실행 결과]

```
4
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

활용 예시 ② N번째로 작은 값

- 파이썬에서는 N번째로 작은 값을 구하는 메서드를 제공한다.
- $nsmallest()$ 메서드는 가장 작은 n개의 값을 반환한다.

```
import heapq

arr = [9, 1, 5, 4, 3, 8, 7]
result = heapq.nsmallest(3, arr)
print(result[-1])
```

[실행 결과]

```
4
```

fast campus Copyright FASTCAMPUS Corp.All Rights Reserved

활용 예시 ③ N번째로 큰 값

- 파이썬에서는 N번째로 큰 값을 구하는 메서드를 제공한다.
- $nlargest()$ 메서드는 가장 큰 n개의 값을 반환한다.

```
import heapq

arr = [9, 1, 5, 4, 3, 8, 7]
result = heapq.nlargest(3, arr)
print(result[-1])
```

[실행 결과]

```
7
```