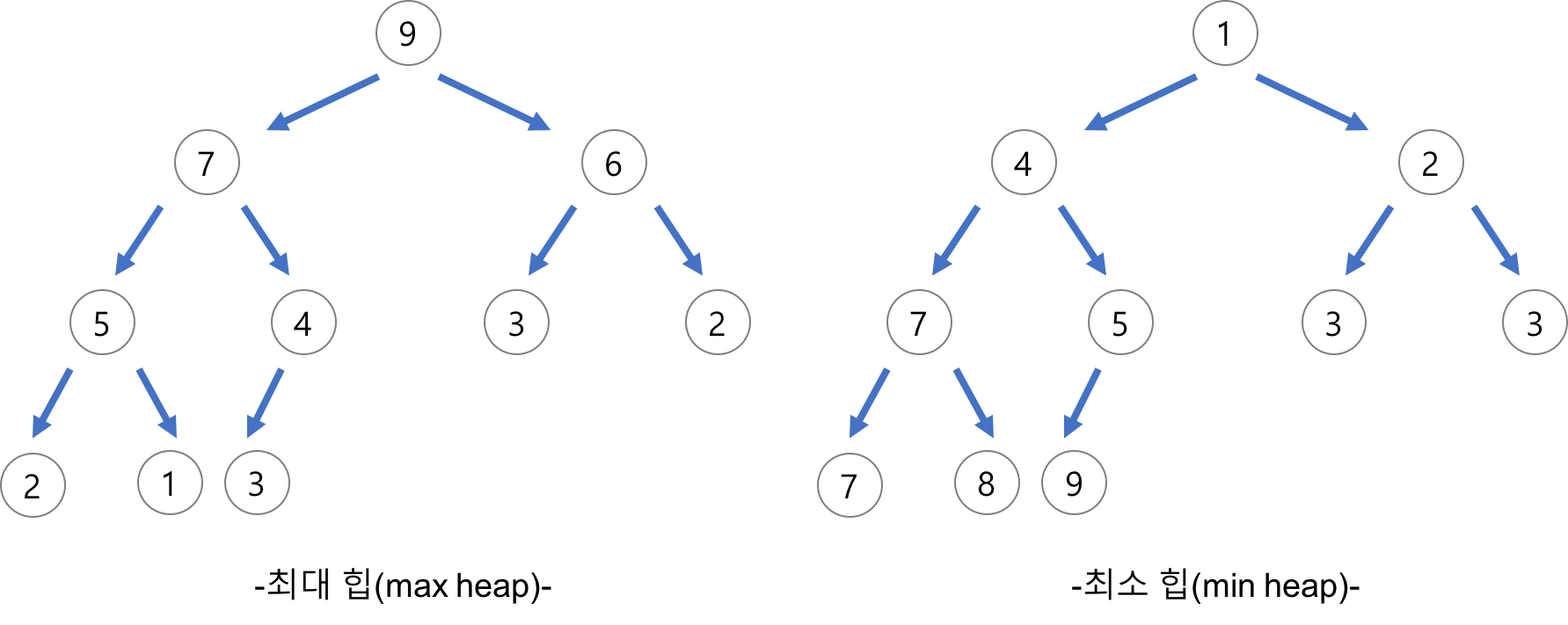
**Heap**

1. Heap이란?

우선순위 큐(가장 우선순위가 높은 데이터를 pop)를 구현하기 위한 자료구조. 완전 이진 트리의 일종. 최대값, 최솟값을 빠르게 검색할 수 있다.

반 정렬상태를 유지한다. 정렬 상태를 자세히 설명하면 단지 부모, 자식 노드 사이의 정렬 상태만을 고려한다.

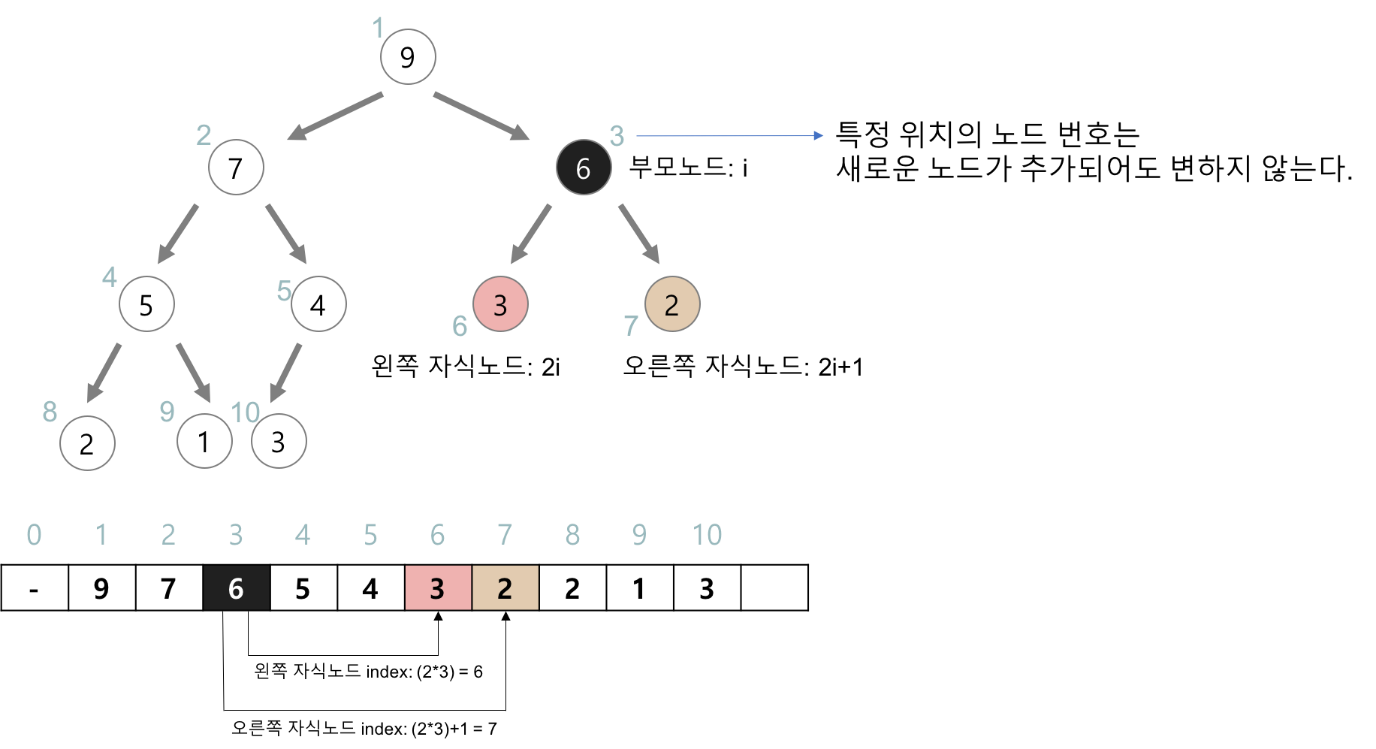
중복된 값이 허용된다. 이는 이진 탐색 트리와의 차이점이다.

1. Heap의 종류

* 최대 Heap: 자식 노드의 Key값 <= 부모 노드의 Key값.
* 최소 Heap: 부모 노드의 Key값 <= 자식 노드의 Key값.

1. Heap의 구현

Heap은 일반적으로 배열 혹은 List형 자료구조를 이용하여 구현한다. 경우에 따라서 구현을 편하게 하기 위해 0번 인덱스를 사용하지 않기도 한다.



* 0번 인덱스를 사용하지 않고 구현한 경우

왼쪽 자식 노드의 인덱스 = 부모의 인덱스 \* 2

오른쪽 자식 노드의 인덱스 = 부모의 인덱스 \* 2 + 1

* 0번 인덱스를 사용하고 구현한 경우

Root의 왼쪽 자식 노드 = 1

Root의 오른쪽 자식 노드 = 2

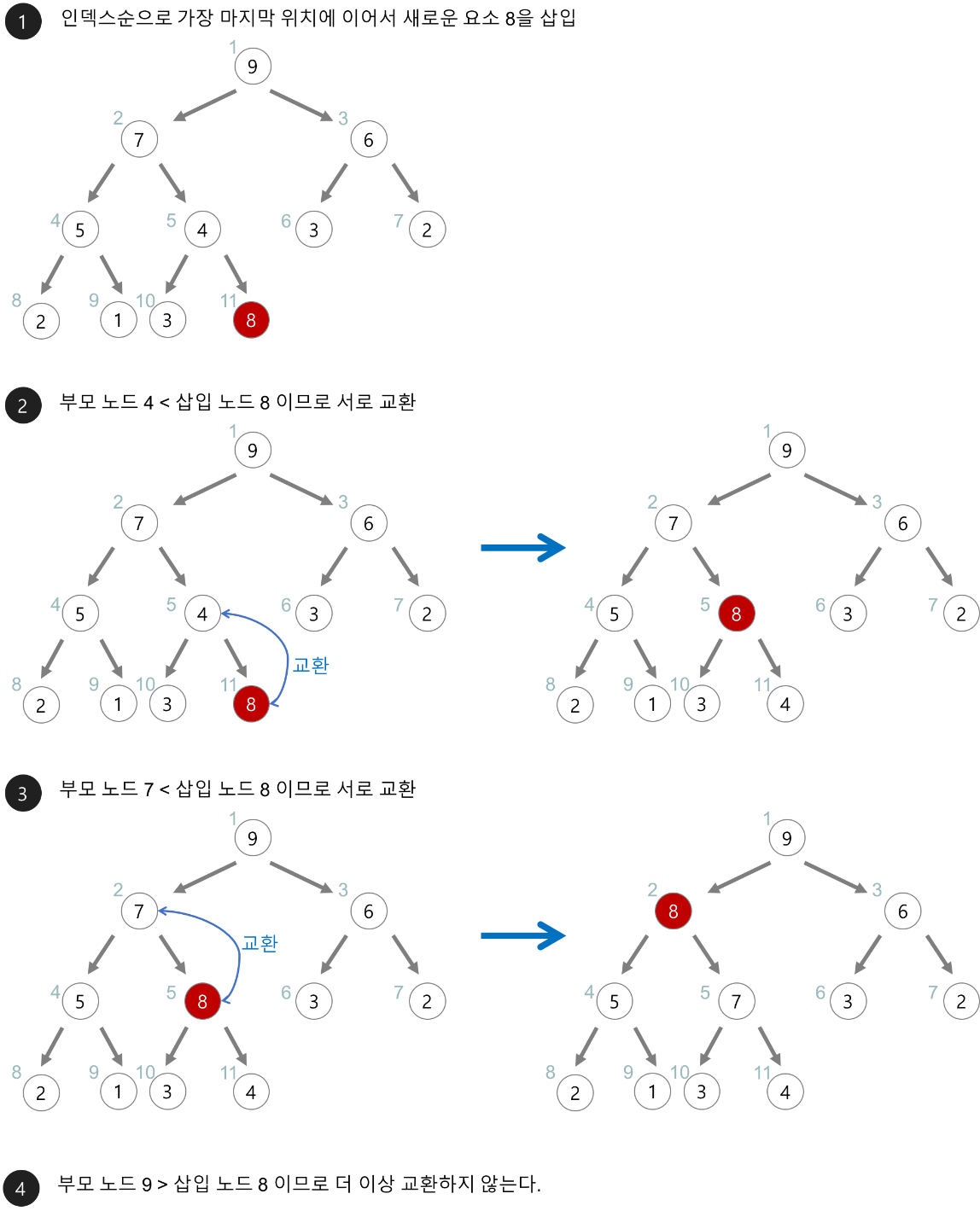
왼쪽 자식 노드의 인덱스 = 부모의 인덱스 \* 2 - 1

오른쪽 자식 노드의 인덱스 = 부모의 인덱스 \* 2

1. Heap의 삽입, 삭제

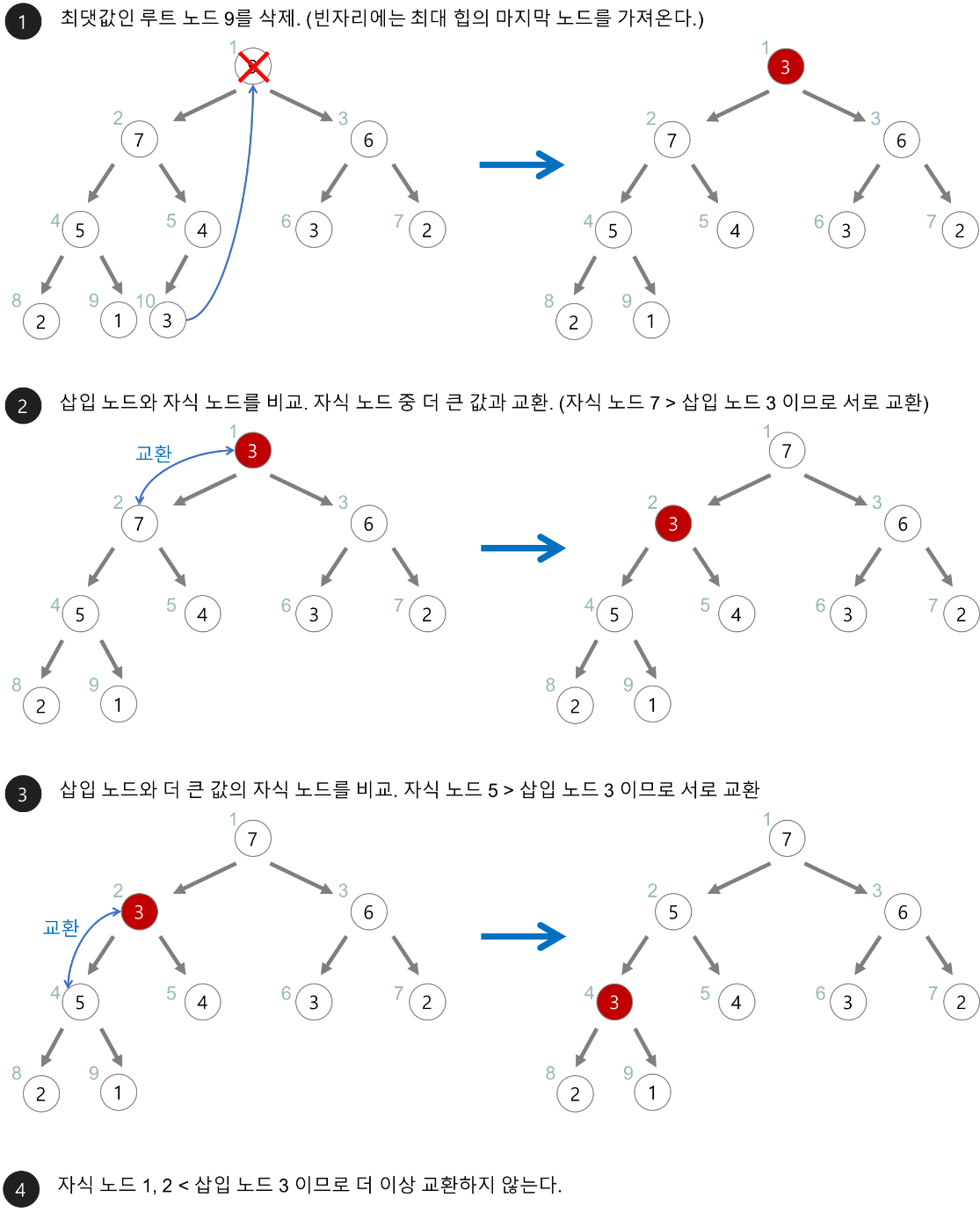
* 삽입

1. 제일 마지막 노드에 새로 들어온 Key를 삽입.
2. 부모 노드와 새로 들어온 노드의 Key값을 비교.
3. 만약 자리를 바꿔야 된다면 서로의 Key값을 교환한 뒤 다시 그 부모와 비교.
4. 위의 과정을 Root노드까지 반복.



* 삭제

1. 최대/최소 Heap 모두 Root노드가 최대/최소 값이기 때문에 Root노드가 Pop된다.
2. 가장 마지막 노드를 Root로 이동시킨다.
3. 자식 노드와 비교하여 더 큰/작은 값과 Key값을 교환한 뒤 다시 자식 노드와 비교.
4. 위의 과정을 말단 노드까지 반복.



1. Heap을 이용하는 사례
2. 시뮬레이션 시스템
3. 네트워크 트래픽 제어
4. 운영 체제에서의 작업 스케쥴링
5. 수치 해석적인 계산