Max heap

우선순위 큐 (priority queue)

우선순위 큐 ADT

1. pq.push(elem)
   : 우선순위를 가진 원소를 삽입하면 자동으로 재정렬
2. pq.pop()
   : 우선순위가 가장 큰 원소를 삭제 후 반환
3. pq.top()
   : 우선순위가 가장 큰 원소를 삭제하지 않고 반환

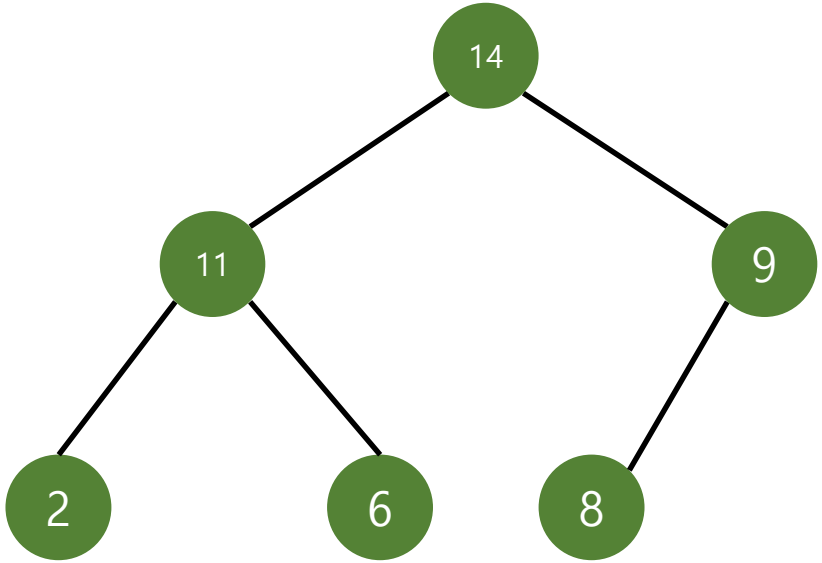
우선순위 큐는 주로 힙을 이용해 구현한다!

최대 힙(max heap)

최대 트리(max tree)
: 어떤 노드의 key가 자식의 key보다 작지 않은 트리

최대 힙(max heap)
: 최대 트리이면서 완전 이진 트리

최대 힙(max heap)

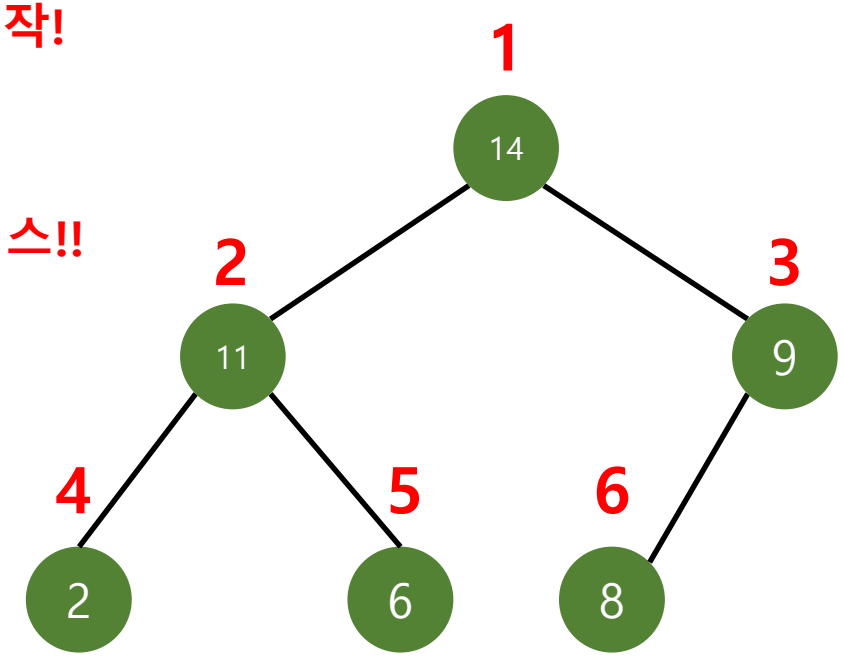


최대 힙(max heap)

Heap은 주로 배열을 이용해 구현

1번 인덱스부터 시작!

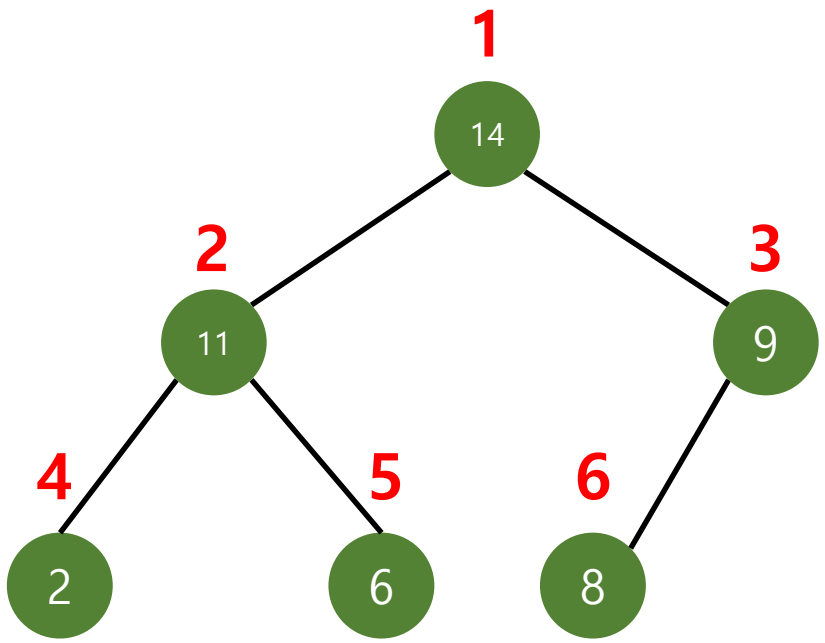
heapsize : 마지막 원소의 인덱스!!



부모 인덱스를 구하는 방법

부모 인덱스 = 인덱스 / 2

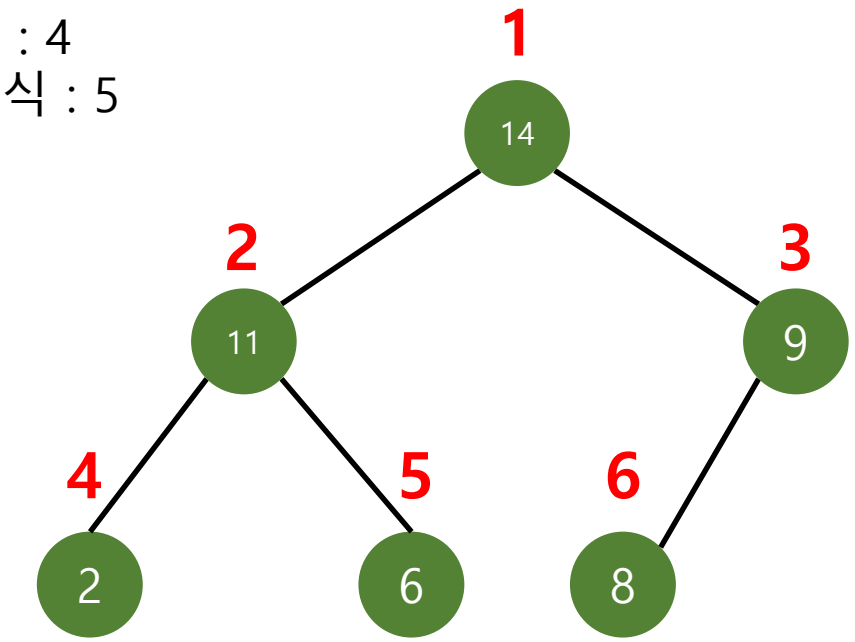
2번 인덱스의 부모 : 1
5번 인덱스의 부모 : 2



자식 인덱스를 구하는 방법

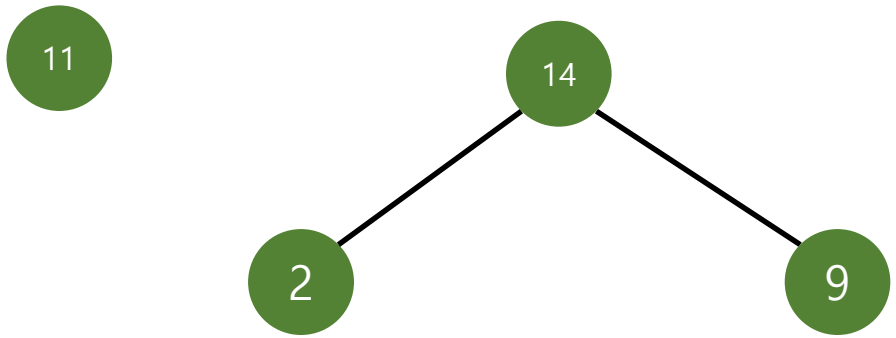
왼쪽 자식 인덱스 = 인덱스 * 2
오른쪽 자식 인덱스 = 인덱스 * 2 + 1

2번 인덱스의 왼쪽 자식 : 4
2번 인덱스의 오른쪽 자식 : 5



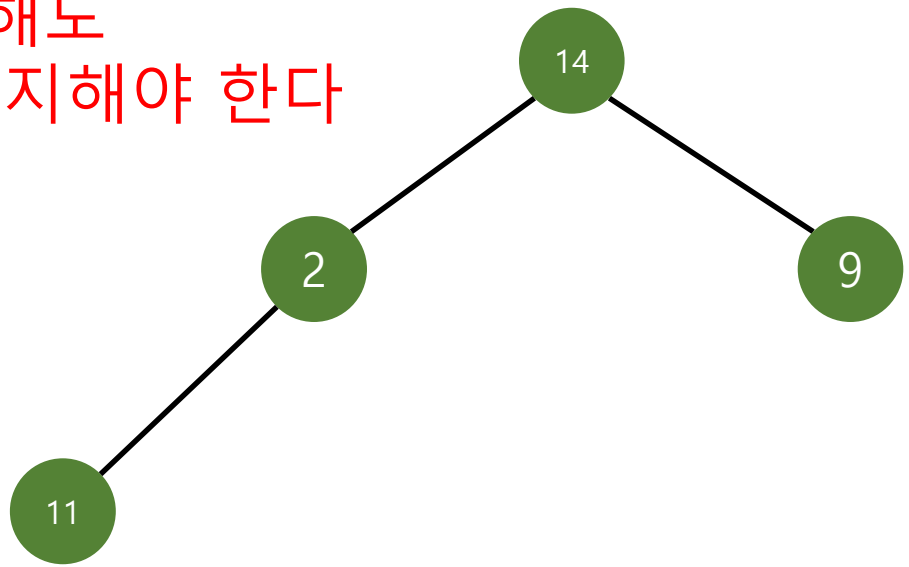
push

item



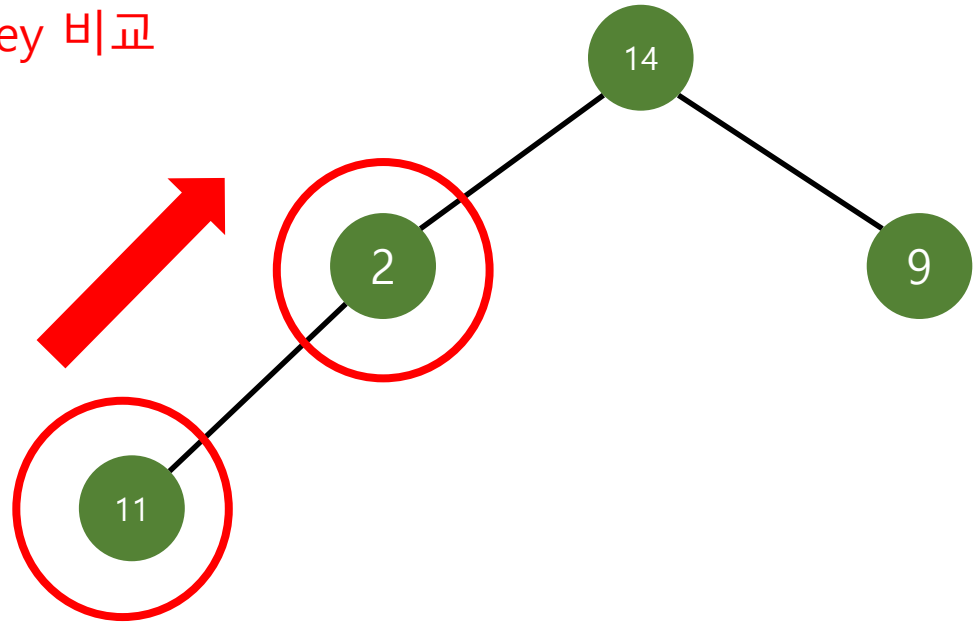
Push 1

새로운 원소를 삽입해도
완전 이진 트리를 유지해야 한다

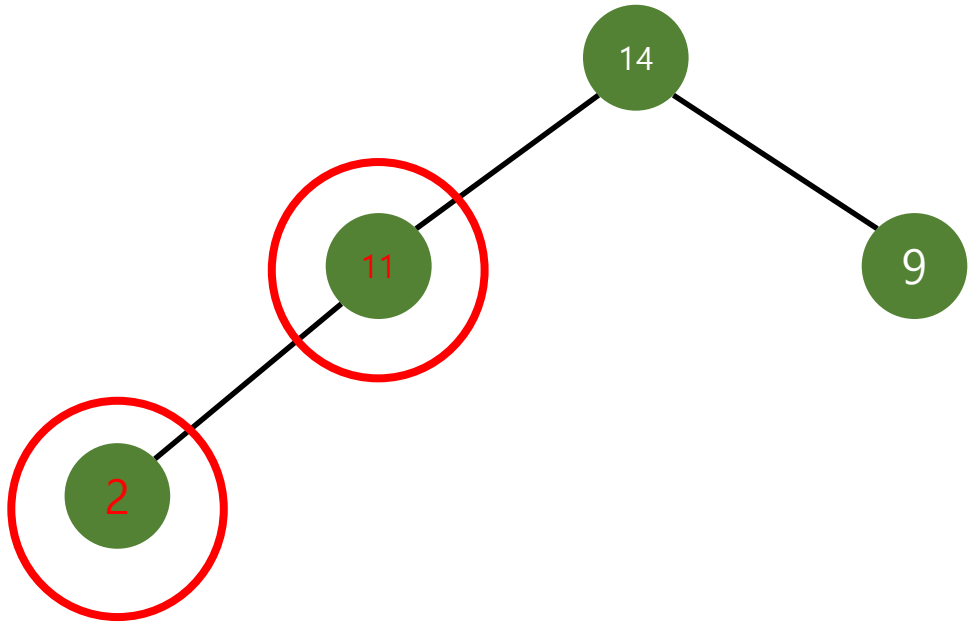


Push 2

삽입된 원소와 부모의 key 비교
→ 부모보다 크면 교환!



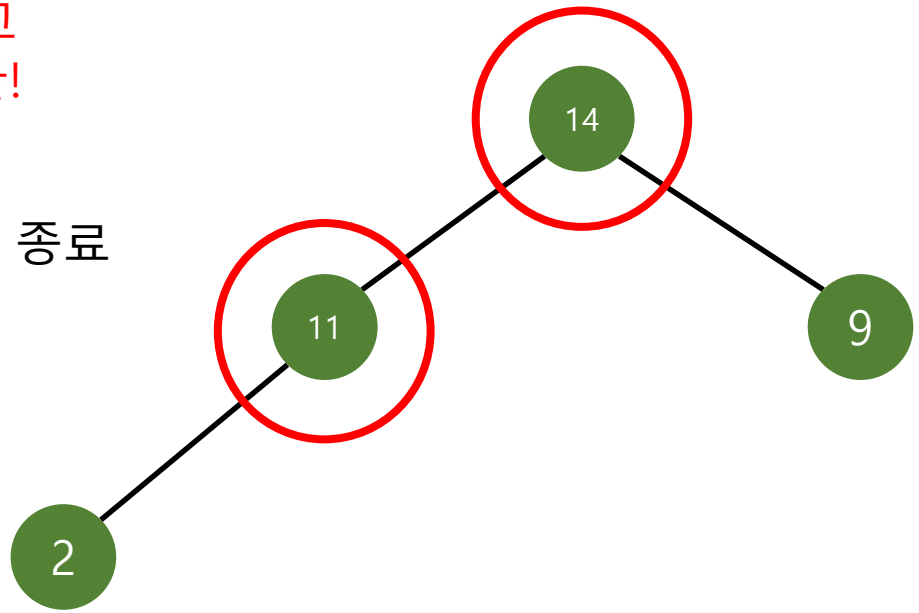
Push 3



Push 4

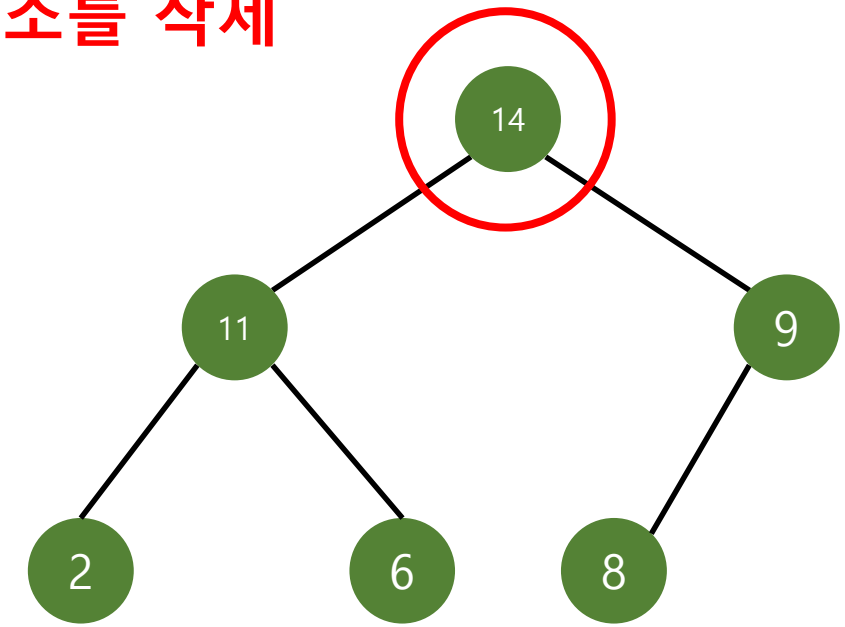
원소와 부모의 key 비교
→ 부모보다 크면 교환!

부모의 key가 더 크므로 종료



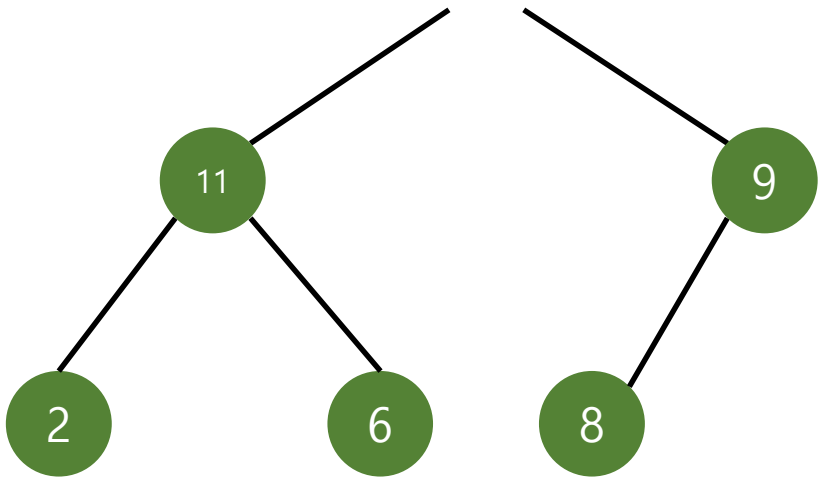
pop

루트에 항상 key가 가장 큰 원소가 위치하므로 루트의 원소를 삭제



Pop 1

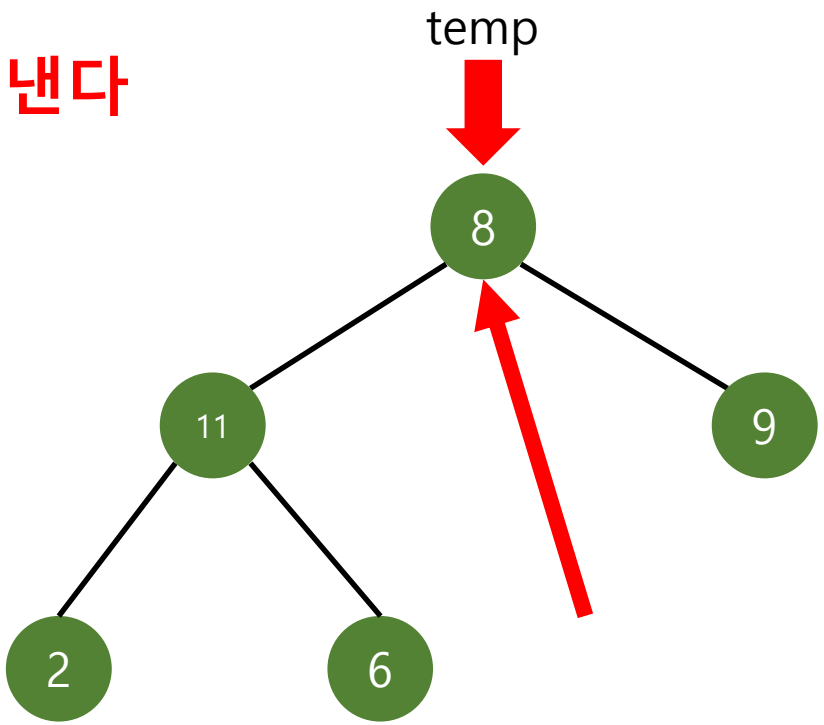
루트가 삭제된 후
힙 프로퍼티를 유지하도록!!



Pop 2

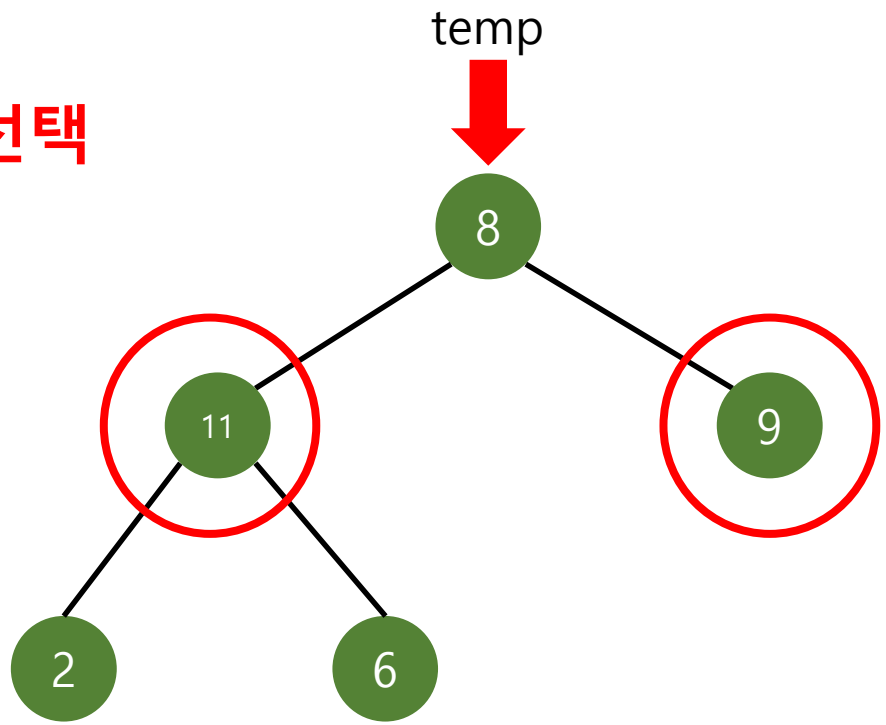
힙의 마지막 원소를
일시적으로 루트로 보낸다

완전 이진 트리 OK!!



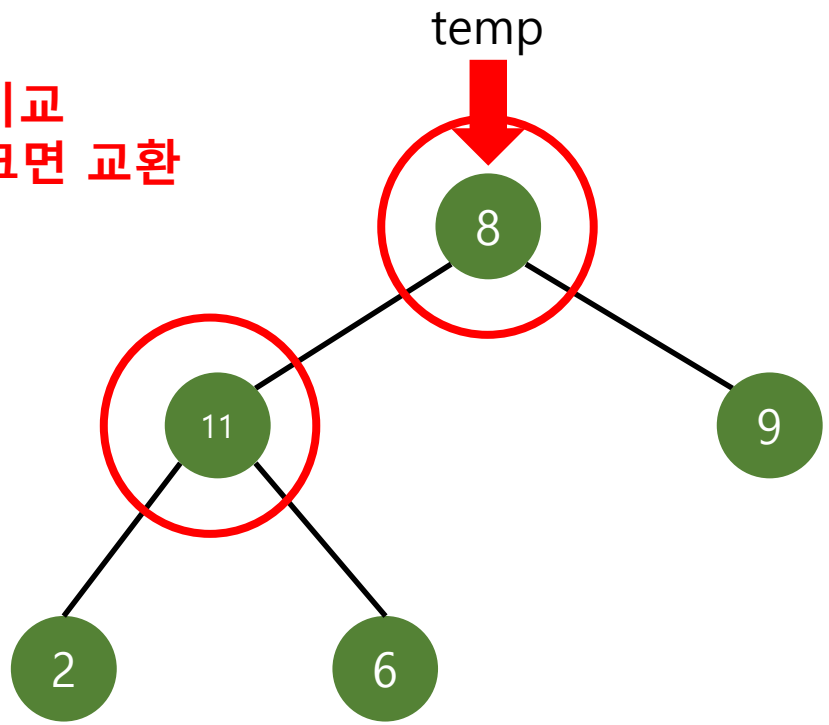
Pop 3

두 자식 원소 중
key가 더 큰 원소를 선택



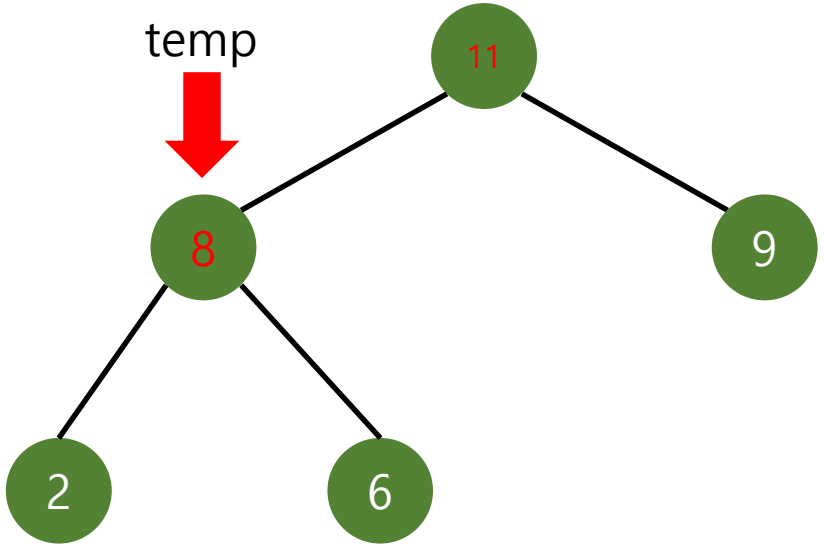
Pop 4

Temp 노드와
Key가 더 큰 자식 원소와 비교
→ 자식 원소의 key가 더 크면 교환



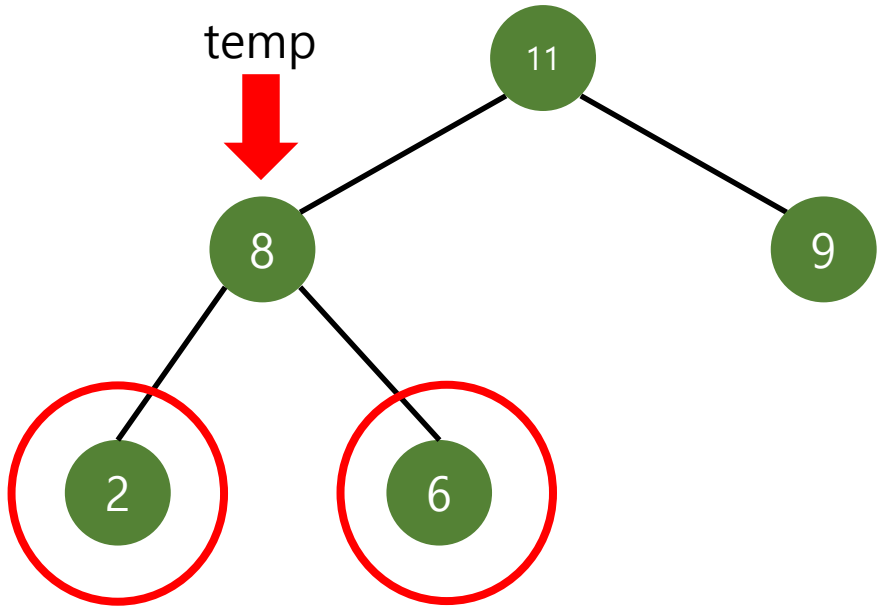
Pop 5

자식 노드의 key가 temp노드의 key보다
더 작을 때까지 반복



Pop 6

두 자식 원소 중
Key가 더 큰 원소 선택



Pop 7

Temp와 key가 더 큰 자식 비교
→ 자식의 key가 더 크면 교환