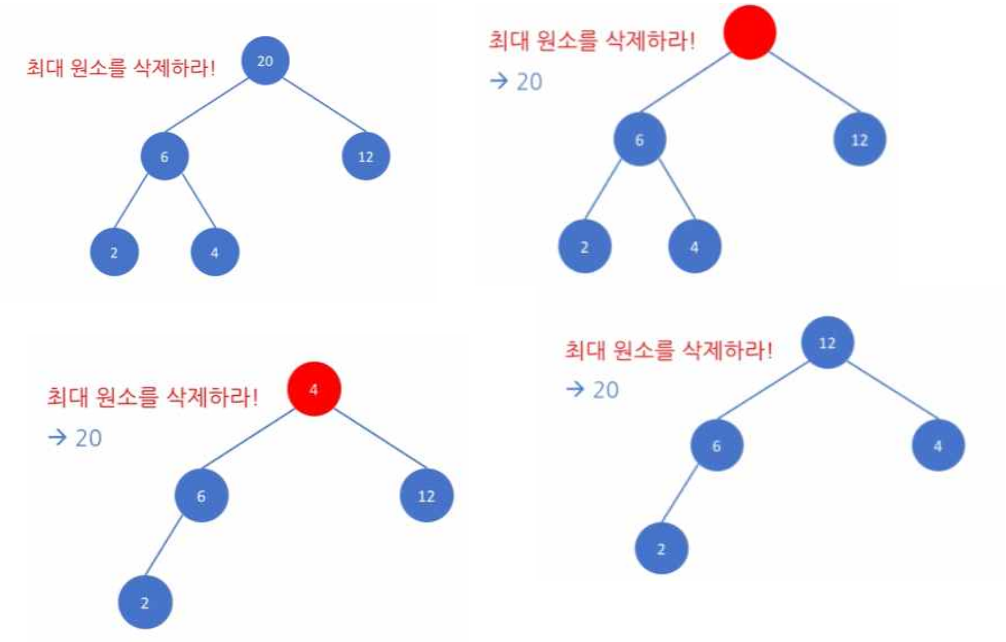
23, 힙(Heaps) (2)

1, 최대 힙에서 원소의 삭제

1) 루트 노드의 제거 : 이것이 원소들 중 최댓값
2) 트리 마지막 자리 노드를 임시로 루트 노드의 자리에 배치
3) 자식 노드들과의 값 비교와 아래로, 아래로 이동
: 자식이 둘이라면 더 큰 키를 가지는 쪽으로 이동!



2, 최대 힙으로부터 원소 삭제: 복잡도

원소의 개수가 n인 최대 힙에서 최대 원소 삭제
-> 자식 노드들과의 대소 비교 최대 회수: 2 * log(2)n
최악 복잡도 O(logn) 의 삭제 연산

3, 최대/최소 힙의 응용

1) 우선 순위 큐 (priority queue)
: Enqueue 할 때, “느슨한 정렬”을 이루고 있도록 함: O(logn)
: Dequeue 할 때, 최댓값을 순서대로 추출 : O(logn)
: 제 16강에서의 양방향 연결 리스트 이용 구현과 효율성 비교

2) 힙 정렬 (heap sort)

: 정렬되지 않은 원소들을 아무 순서로나 최대 힙에 삽입: O(logn)
: 삽입이 끝나면, 힙이 비게 될 때까지 하나씩 삭제: O(logn)
: 원소들이 삭제된 순서가 원소들의 정렬 순서
: 정렬 알고리즘의 복잡도: O(nlogn)

4, 힙 정렬 (heap sort)의 코드 구현

```
def heapsort(unsorted):
    H = MaxHeap()
    for item in unsorted:
        H.insert(item)
    sorted = []
    d = H.remove()
    while d:
        sorted.append(d)
        d = H.remove()
    return sorted
```

5) 문제

초기 코드에 여기 저기 포함된 빈 칸을 채움으로써 class MaxHeap 의 메서드인 maxHeapify() 의 구현을 완성하세요. 이것은 이미 주어져 있는 remove() 메서드와 연결되어 최대 힙에서의 원소 삭제 연산을 구성합니다.
[참고 1] remove() 메서드의 내용은 이미 주어져 있으므로 수정하지 않는 쪽이 좋습니다.
[참고 2] solution() 함수의 구현은 그대로 두세요. 이것을 없애면 테스트가 되지 않습니다.
[참고 3] 코드 실행 을 눌렀을 때 통과하는 것은 아무런 의미가 없습니다.

```
class MaxHeap:

    def __init__(self):
        self.data = [None]

    def remove(self):
        if len(self.data) > 1:
            self.data[1], self.data[-1] = self.data[-1], self.data[1]
            data = self.data.pop(-1)
            self.maxHeapify(1)
        else:
            data = None
        return data

    def maxHeapify(self, i):
        # 왼쪽 자식 (left child) 의 인덱스를 계산합니다.
        left = 2*i

        # 오른쪽 자식 (right child) 의 인덱스를 계산합니다.
        right = 2*i+1
```

```
        smallest = i
        # 왼쪽 자식이 존재하는지, 그리고 왼쪽 자식의 (키) 값이 (무엇보다?) 더 큰지를 판단합니다.
        if left<len(self.data) and self.data[left]>self.data[smallest]:
            # 조건이 만족하는 경우, smallest 는 왼쪽 자식의 인덱스를 가집니다.
            smallest=left

        # 오른쪽 자식이 존재하는지, 오른쪽 자식의 (키) 값이 (무엇보다?) 더 큰지를 판단합니다.
        if right<len(self.data) and self.data[right]>self.data[smallest]:
            # 조건이 만족하는 경우, smallest 는 오른쪽 자식의 인덱스를 가집니다.
            smallest=right

        if smallest != i:
            # 현재 노드 (인덱스 i) 와 최댓값 노드 (왼쪽 아니면 오른쪽 자식) 를 교체합니다.
            self.data[smallest],self.data[i]=self.data[i],self.data[smallest]

            # 재귀적 호출을 이용하여 최대 힙의 성질을 만족할 때까지 트리를 정리합니다.
            self.maxHeapify(smallest)


def solution(x):
    return 0
```