기술면접 – 힙 정렬(Heap Sort)

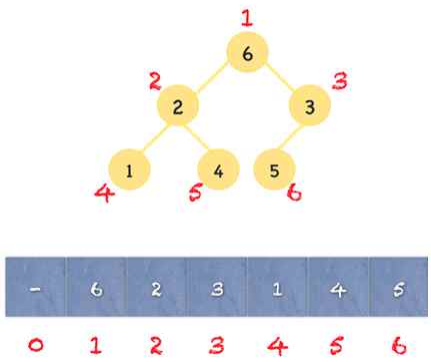
힙 특징

- Heap은 Tree의 구조로 이루어진 자료구조로, 완전이진트리 자료구조입니다. 대표적으로 최소힙과 최대힙이 존재하며, 최소힙은 각 Node에 해당하는 값이 자식 Node의 값보다 작거나 같은 Binary Tree입니다.
- 이때, root Node는 최소값을 가지기 때문에, 최소값을 찾는데는 항상 O(1) 시간복잡도를 보장합니다. 반면, 추가나 삭제에서는 O(log n)의 시간복잡도를 가집니다.
- 최댓값, 최솟값을 쉽게 추출할 수 있는 자료구조

- 1. 구조조건 - 완전이진트리
- 2. 순서조건 - Partial Order를 만족한다.

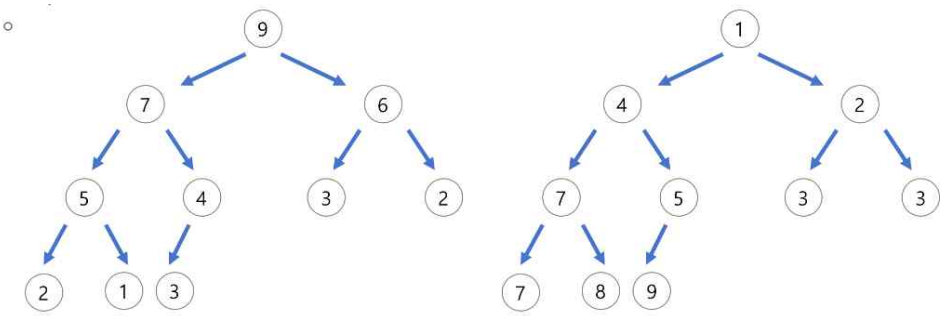
구조조건 - 완전이진트리?

- 완전 이진트리는 높이가 h 일 때 레벨 h-1까지는 Full BT이고, 레벨 h에서는 왼쪽부터 노드가 순서대로 채워진 이진트리



순서 조건

- Total Order : 정렬된 시퀀스(sorted sequence)라 할 수 있어요. 순서조건이 모든 원소에 대해 만족
- Partial Order : Partial Order는 데이터의 일부분에 대해서 어떠한 특성을 만족하는 것
- 힙의 순서조건은 부모노드의 키값이 자식노드의 키값보다 항상 큰 힙을 '최대 힙' 부모노드의 키값이 자식노드의 키값보다 항상 작은 힙을 '최소 힙'이라고 부릅니다.
- 하지만 total Order를 만족하지 못하는 이유는 위 조건이 "직계관계"에서만 만족한다는 것입니다.
- 힙 구조에서 어떤 키 값에 대해 레벨은 위지만(depth가 작지만) 현재 이 노드의 키값보다 키값이 작을 수 있다는 것입니다. 그래서 Partial Order입니다.
- 힙은 정렬된 구조가 아니다. 최소 힙의 경우 부모 노드가 항상 작다는 조건만 만족할 뿐, 서로 정렬되어 있지 않다.



-최대 힙(max heap)- -최소 힙(min heap)-

힙정렬 알고리즘 장점, 시간복잡도

- 최소 힙은 부모가 항상 자식보다 작기 때문에, 루트가 결국 가장 작은 값을 갖게 되며, 우선순위 큐에서 가장 작은 값을 추출하는 것은 매번 힙의 루트를 가져오는 형태로 구현된다. O(1). 대신 삽입할 때마다 정렬해야 하므로, 삽입할 때의 시간복잡도는 O(NlogN)이다.
- 우선순위 큐에서 활용
- 다익스트라 알고리즘에서도 힙을 사용하면 시간복잡도를 $O(V^2)$에서 O(ElogV)로 줄일 수 있다.
- 적은 시간복잡도

평균	최선	최악
Θ(nlogn)	Ω(nlogn)	O(nlogn)

- 빠른 탐색: 탐색 시 시간 복잡도가 O(logN)

→ 1억개의 아이템에서 27번만에 원하는 값 찾음
→ 100개의 아이템에서 7번만에 원하는 값 찾음

- 삽입 할 때 시간복잡도 역시 O(logN)
- But, 이진 탐색 트리의 균형이 깨지면, 탐색 시 시간 복잡도가 O(logN)이 아니라 O(N)에 근접한 시간이 소요될 수 있다.

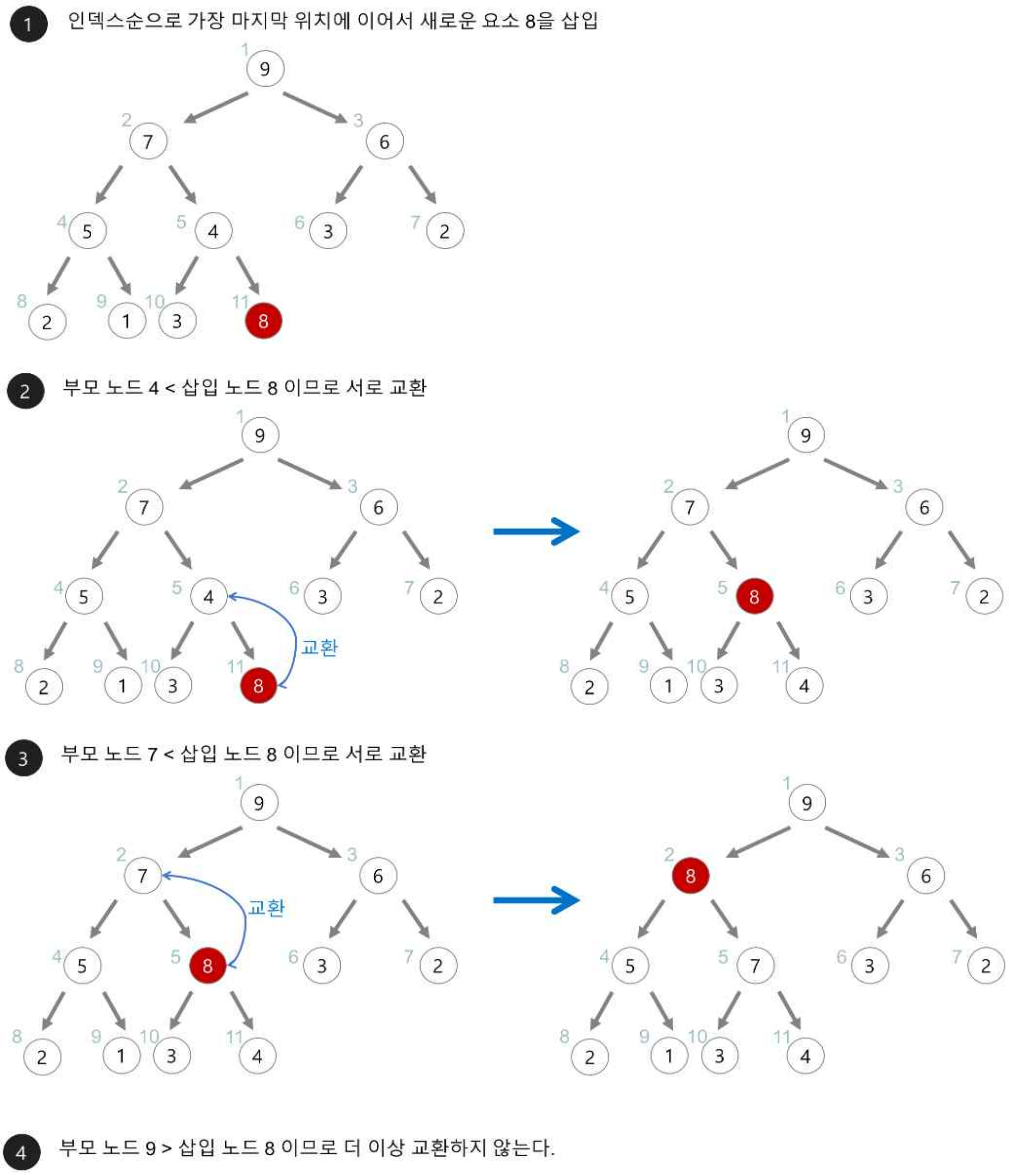
힙 정렬(Heap Sort) 알고리즘

- 최대 힙 트리나 최소 힙 트리를 구성해 정렬을 하는 방법
- 내림차순 정렬을 위해서는 최대 힙을 구성하고 오름차순 정렬을 위해서는 최소 힙을 구성하면 된다.
- 과정 설명 (최대힙 정렬)
  1. 최대 힙을 구성 - 정렬해야 할 n개의 요소들로 최대 힙(완전 이진 트리 형태)을 만든다. (내림차순을 기준으로 정렬)
  2. 현재 힙 루트는 가장 큰 값이 존재함. 그 다음으로 한 번에 하나씩 요소를 힙에서 꺼내서 배열의 뒤부터 저장하면 된다.
  3. 삭제되는 요소들(최댓값부터 삭제)은 값이 감소되는 순서로 정렬되게 된다
  4. 힙의 사이즈가 1보다 크면 위 과정 반복

[추가 알고리즘]

- 힙에 새로운 요소가 들어오면, 일단 새로운 노드를 힙의 마지막 노드에 이어서 삽입한다.
- 힙의 성질을 만족시킬 때까지 새로운 노드를 부모 노드들과 교환해서 승격작업을 반복해 자리를 찾아 간다.

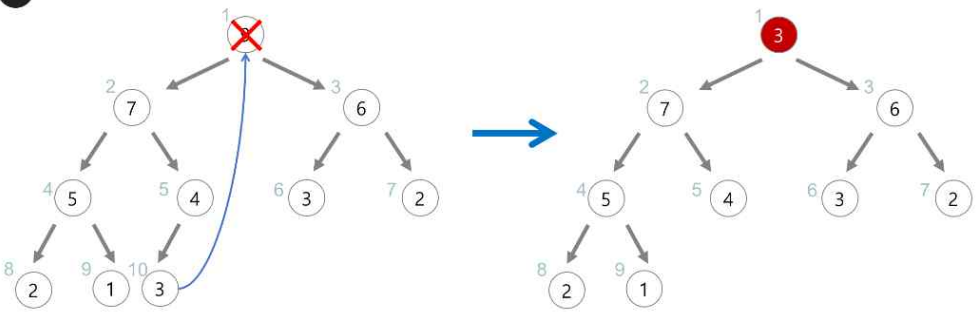
아래의 최대 힙(max heap)에 새로운 요소 8을 삽입해보자.



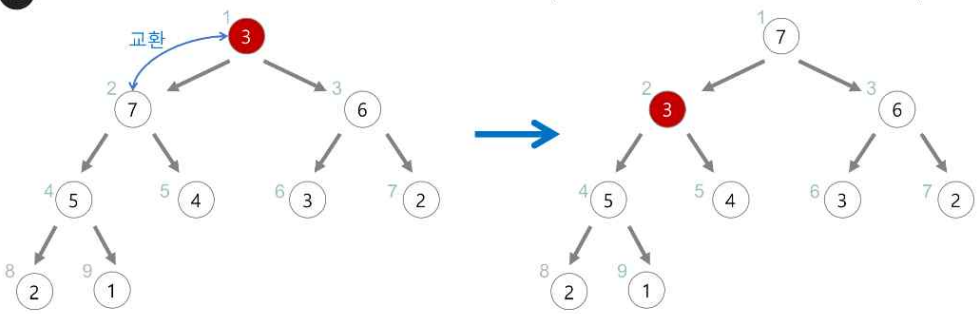
[삭제 알고리즘]

- 최대 힙에서 최댓값은 루트 노드이므로 루트 노드가 삭제된다.
- 최대 힙(max heap)에서 삭제 연산은 최댓값을 가진 요소를 삭제하는 것이다.
- 삭제된 루트 노드에는 힙의 마지막 노드를 가져온다.
- 힙의 성질을 만족시킬 때까지 루트 노드를 자식 노드들과 교환해서 강등작업을 반복해 자리를 찾아 간다.

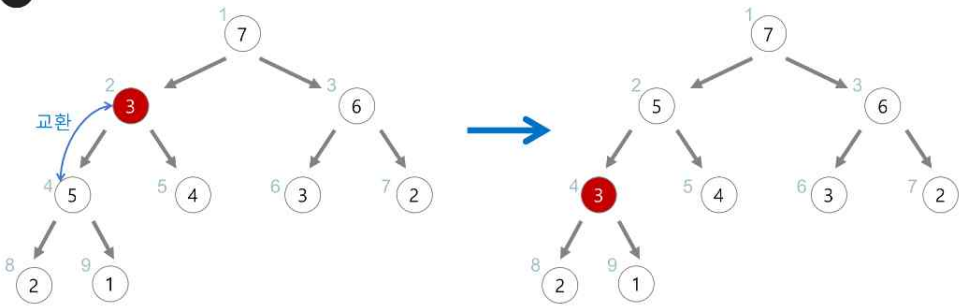
1 최댓값인 루트 노드 9를 삭제. (빈자리에는 최대 힙의 마지막 노드를 가져온다.)



2 삽입 노드와 자식 노드를 비교. 자식 노드 중 더 큰 값과 교환. (자식 노드 7 > 삽입 노드 3 이므로 서로 교환)



3 삽입 노드와 더 큰 값의 자식 노드를 비교. 자식 노드 5 > 삽입 노드 3 이므로 서로 교환



4 자식 노드 1, 2 < 삽입 노드 3 이므로 더 이상 교환하지 않는다.

여기서 부모노드와 자식노드의 인덱스를 구하는 방법은 이렇다

* 0번 인덱스부터 시작하는 트리
부모 노드 인덱스의 왼쪽 자식의 노드 인덱스 : 부모노드 인덱스 * 2 + 1
부모 노드 인덱스의 오른쪽 자식의 노드 인덱스 : 부모노드 인덱스 * 2 + 2
자식 노드 인덱스의 부모 노드 인덱스 : (자식의 인덱스 – 1) / 2
* 1번 인덱스부터 시작하는 트리
부모 노드 인덱스의 왼쪽 자식의 노드 인덱스 : 부모노드 인덱스 * 2
부모 노드 인덱스의 오른쪽 자식의 노드 인덱스 : 부모노드 인덱스 * 2 + 1
자식 노드 인덱스의 부모 노드 인덱스 : 자식의 인덱스 / 2

힙 정렬(Heap Sort)의 시간복잡도 정리

- 힙 트리의 전체 높이가 거의 $\log_2 n$(완전 이진 트리이므로)이므로 하나의 요소를 힙에 삽입하거나 삭제할 때 힙을 재정비하는 시간이 $\log_2 n$만큼 소요된다.
- 요소의 개수가 n개 이므로 전체적으로 $O(n\log_2 n)$의 시간이 걸린다.
- $T(n) = O(n\log_2 n)$

정렬 알고리즘 시간복잡도 비교

- 단순(구현 간단)하지만 비효율적인 방법 : 삽입 정렬, 선택 정렬, 버블 정렬
- 복잡하지만 효율적인 방법 : 퀵 정렬, 힙 정렬, 합병 정렬, 기수 정렬

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셸 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n\log_2 n$	$n\log_2 n$	n^2	0.014
힙 정렬	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	0.034
병합정렬	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	0.026

이진트리와의 차이점

- 힙은 상/하 관계를 보장하고
- 이진 탐색 트리는 좌/우 관계를 보장한다.