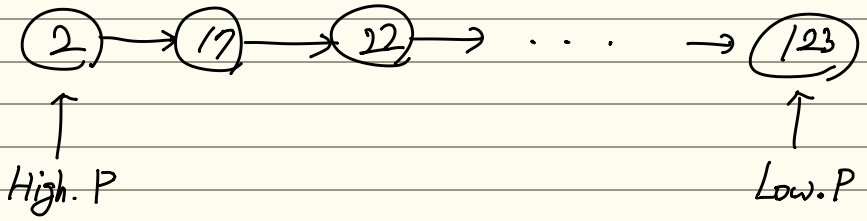
7.1 우선순위 큐

Queue ⇒ FIFO 자료구조

Queue + Priority ⇒ 우선순위 큐

⇒ 우선순위에 따라 출력 순서 결정



우선순위가 20인 Node 삽입

⇒ → 17 → 20 → 22 → ...

Linked List로 구현한 경우 O(n) 삽입시간

제거의 경우는 High. P 제거하면 되므로 O(1)

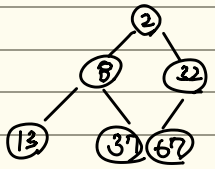
7.2 Heap

→ Heap Order Property를 만족하는 완전이진트리

⇓

Min Heap의 경우

⇒ 트리 구성하는 모든 노드는 자신의 부모보다 큰 수를 갖는다.



Heap을 사용하는 이유 ⇒ 가장 작은 값 or 가장 큰 값을 O(1) 시간으로 찾을 수 있다

7.3 Heap 기반 Priority Queue

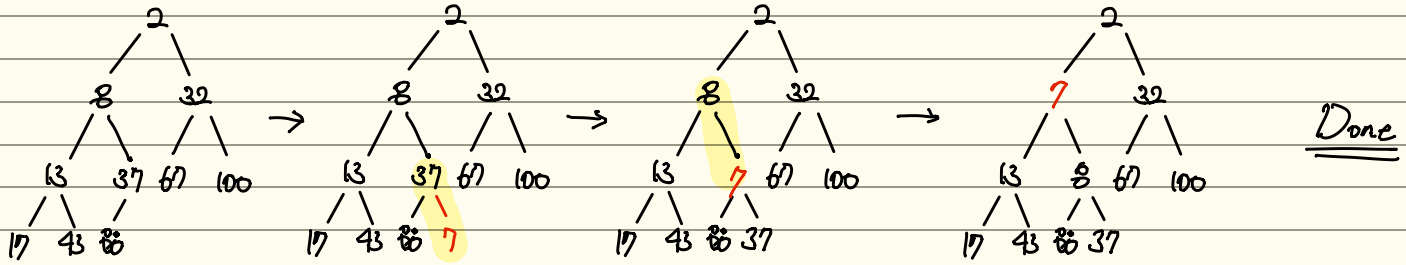
⇒ 초기를 Priority로 생각하면 된다.

항상 Priority가 (Min Heap 기준) 큰 값부터 노드를 뽑아낼 수 있다.

7.2.1 Heap Insert

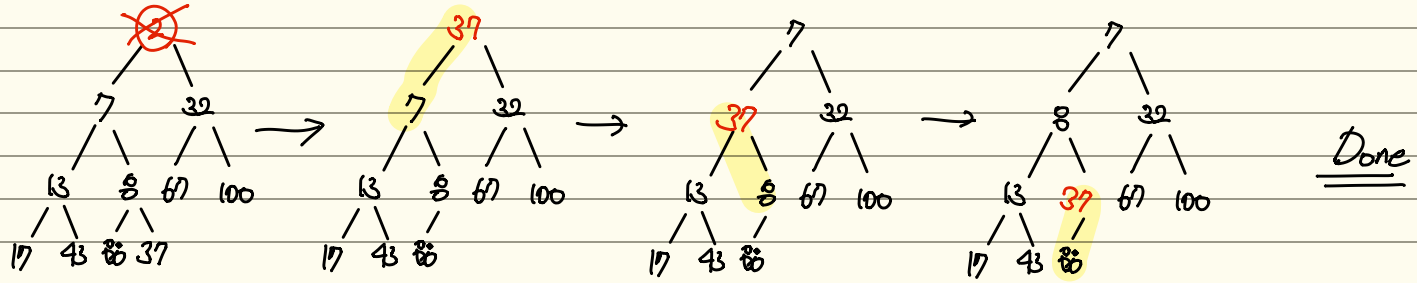
① Tree의 최고 깊이의 가장 우측에 새 노드 추가
② Heap 조건 만족하는지 체크
③ 만족하지 않으면 부모와 자리 교체 후 ②번 다시 수행

ex) ⑦ 삽입 (Min Heap)



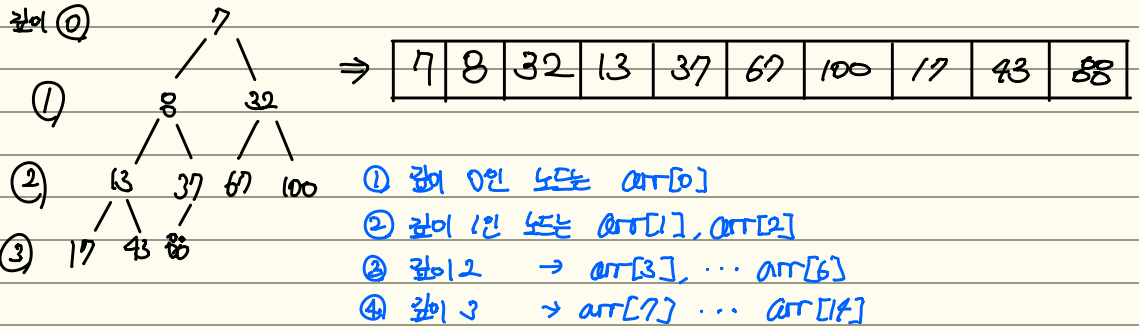
7.2.2 Heap Deletion

① Root 삭제
② 가장 우측의 Leaf Node를 Root로 설정
③ Heap 조건 파악
④ 조건에 맞지 않다면 두 자식 중 더 작은값과 교환 후, ③ 다시 수행



7.2.3 Implementing Heap

⇒ 배열로 구현한다.



① 깊이 0인 노드는 arr[0]
② 깊이 1인 노드는 arr[1], arr[2]
③ 깊이 2 → arr[3], ··· arr[6]
④ 깊이 3 → arr[7] ··· arr[14]
∴ 깊이 n → $arr[2^n-1]$ ··· $arr[2^{n+1}-2]$

* k번 인덱스에 위치한 노드의 자식 인덱스
- Left Child : 2k+1
- Right Child : 2k+2

* k번 인덱스 노드의 부모 노드 인덱스
- (k-1)//2