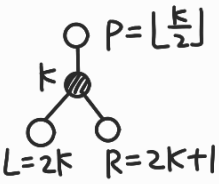
Ch 8 우선순위 큐 (Priority Q)

Heap : 완전이진트리 (Complete BT)

↳ 배열에 저장



5-1, 5-2번

Heap의 성질 (P ≥ L, R) (Max Heap)

≤ (Min Heap)

*모든 subtree의 P가 Max여야 Max Heap이라고 할 수 있음

1. 삽입

① 맨 끝에 data 추가 (size 증가)

② 추가된 원소를 Heap이 되도록 올려보냄

7-3번

O(lgN)

```
HeapAdd (H, N, key)
{   1. H[++N] = key    //data 추가, size 증가
    2. HeapUp(N)       // 올려보냄
}
    HeapUp(k)
    {  if (k>1)  // root 아님
       {  p = ⌊k/2⌋   // 부모
          if (H[k] > H[p])  //Max Heap
          {  swap(H[k], H[p])
             HeapUp(p)   // 재귀적 반복
          }
       }
    }
```

O(lgN)

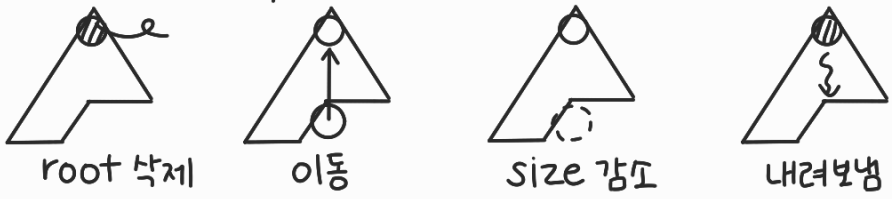
2. 삭제

① root (최댓값)을 삭제

② 맨 끝 원소를 root로 이동 (size 감소)

③ root를 Heap이 될 때까지 내려보냄 (재귀적)

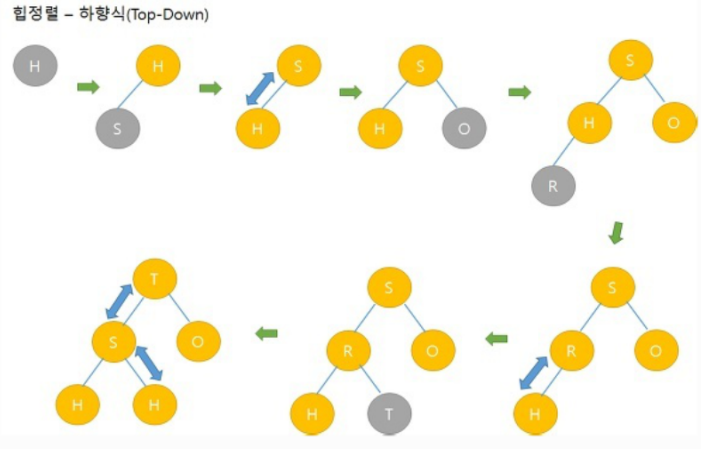
7-2번



```
HeapDelete(H, N)
{  1. Remove root(H[1]) and Service
   2. H[1] ← H[N--]        // 이동, 감소
   3. HeapDown(1, N)
}
```

```
HeapDown(K, N)
{  if(K ≤ ⌊N/2⌋) ← 마지막 비단말 node
   {  L = 2K, R = 2K+1
      M = MaxIndex(H, K, L, R)        O(lg N)
      if(K ≠ M) ← M 자식으로 내려감
      {  swap(H[K], H[M])
         HeapDown(M, N)
      }
   }
}
```

삽입할 때마다 조정



3. 초기 힙 구성 (Heap Building)

배열 H[1..N]에 N개의 data

① Top-down (하향식) 구성

i) H[1]은 자체로 Heap size = 1

ii) 그 후 Heap에 2~N까지 차례로 추가 (HeapUp)

: HeapAdd()를 N번 반복

`for(k=2 to N) HeapUp(N)` ⇒ O(N lg N)

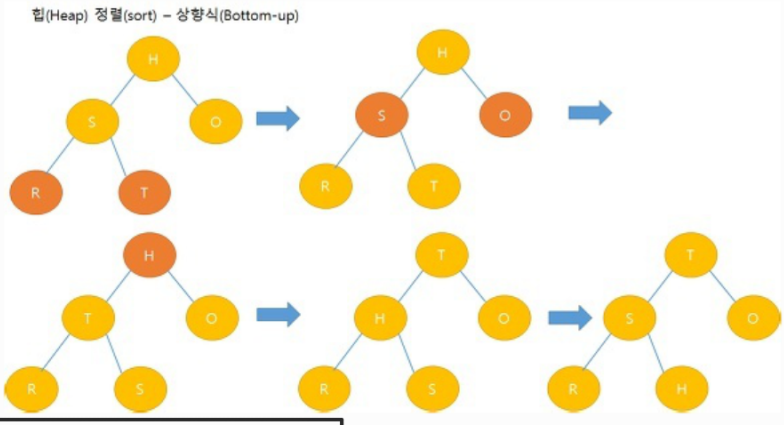
배열 내에서 Heap이 위에서 아래로 확장된다.

6-1번

② Bottom-Up (상향식) 구성

i) 모든 leaf(단말) node는 자체로 Heap

ii) 그 후 모든 internal node (비단말)에 대해 index가 감소하는 순으로 HeapDown 시킨다.



트리 구성 후에 재조정 (아래 → 위, 오른쪽 → 왼쪽)

해당 노드, 해당 노드의 아래쪽만 검사

```
for (K = ⌊N/2⌋ down to 1) HeapDown(K, N)
```

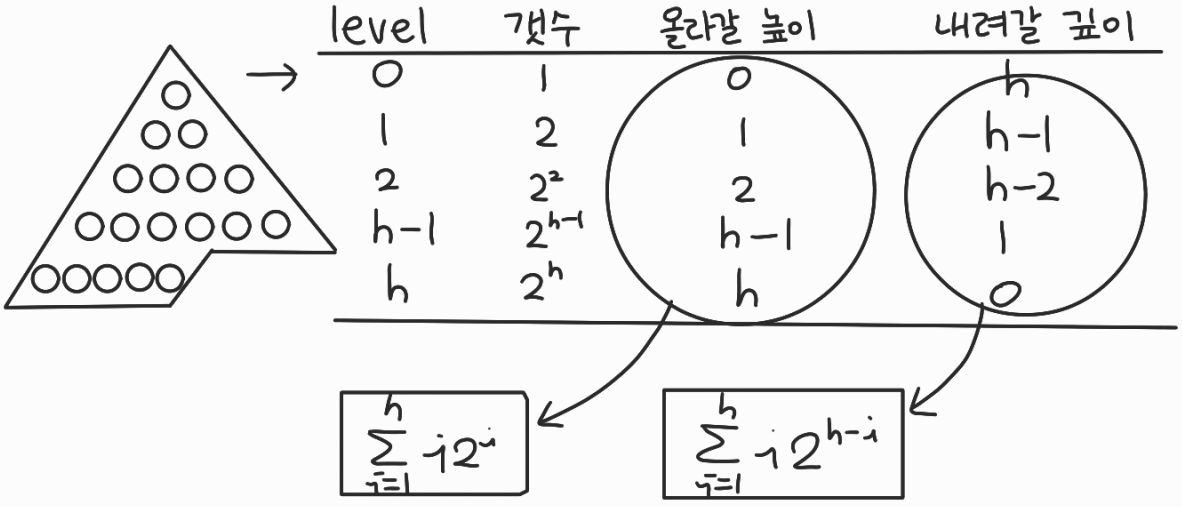
배열 내에서 Heap이 아래에서 위로 확장된다.

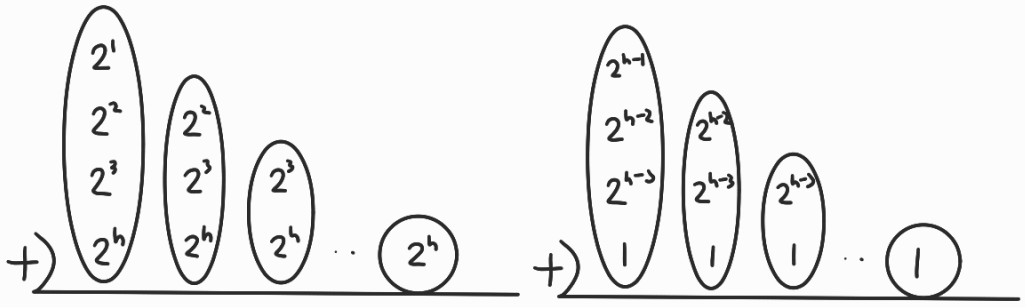
O(N)

why?

6-2번

4. Why Bottom-Up faster than Top-down?





$$\sum_{i=1}^{n}\left(\sum_{j=i}^{h} 2^j\right) = O(N \lg N)$$

$$\sum_{i=1}^{h}\left(\sum_{j=0}^{i-1} 2^j\right) = O(N)$$

5. Heap Sorting

: Heap을 구성한 후 root를 N번 삭제하여 나열함으로 Sorting 한다.

⇒ O(N lg N)

A[1..N]

1. Build_Heap // Bottom-Up으로 MaxHeap 구성 ⇒ O(N)
2. for(k=N down to 2)

```
{ swap(A[i], A[k])   // 삭제
  HeapDown(1, k-1)   // 재정비
}
```

) O(N lg N)