자료구조

L07 Priority Queues

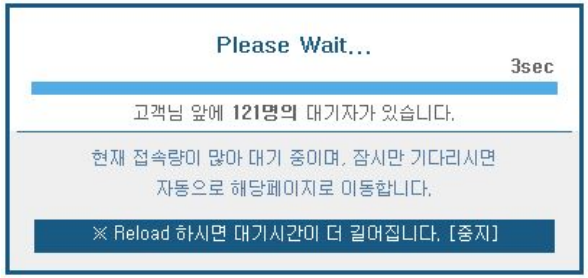
2022년 1학기

국민대학교 소프트웨어학부

In this lecture

- Priority Queue 자료형의 필요성
- Heap 자료구조의 핵심 아이디어 및 구현
- Heap 자료구조 분석

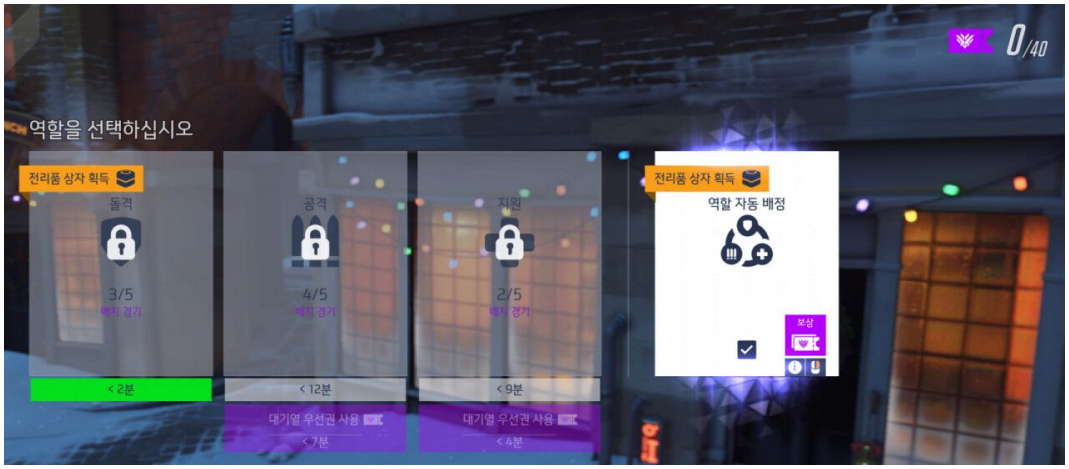
Priority Queues



모두에게 평등한
수강신청 대기열

모두에게 평등한 게임 대기열 →

우선순위를 조절하는 게임 대기열
↓↓↓↓



Priority Queues

문제상황: 들어오는대로 레코드를 저장하고 (insert), 우선순위를 고려하여 값을 꺼내는(removemax) 그런 자료구조는 없을까?

예)

- 멀티태스킹이 가능한 운영체제에서 작업 스케쥴링하기
- VIP 고객은 우선하여 들여보내기

Priority Queues

- 어떻게 구현하지?
  - **Unsorted** array or linked list?
  - **Sorted** array or linked list?
  - or something others?

Priority Queues

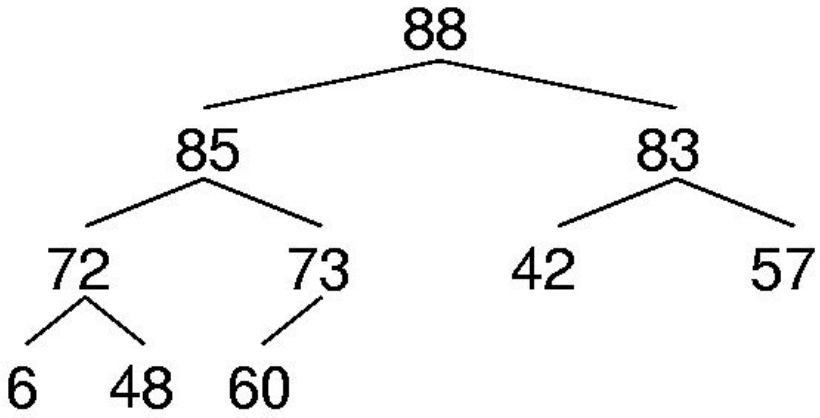
- 어떻게 구현하지?
  - **정렬되지 않은** array 혹은 linked list의 한쪽 끝에 넣고 O(1), 전체 레코드를 스캔하여 최대값을 찾아 꺼낸다 O(n)
  - **정렬된** array 혹은 linked list를 사용; 적절한 위치에 넣고 O(n), 한쪽 끝에 있는 최대값을 꺼낸다 O(1).
  - ***BST* 사용** - 레코드를 넣고, 최대값을 꺼내는게 모두 O(log n)
  - ***heap* 사용** - 레코드를 넣고, 최대값을 꺼내는게 모두 O(log n)

Heap vs BST

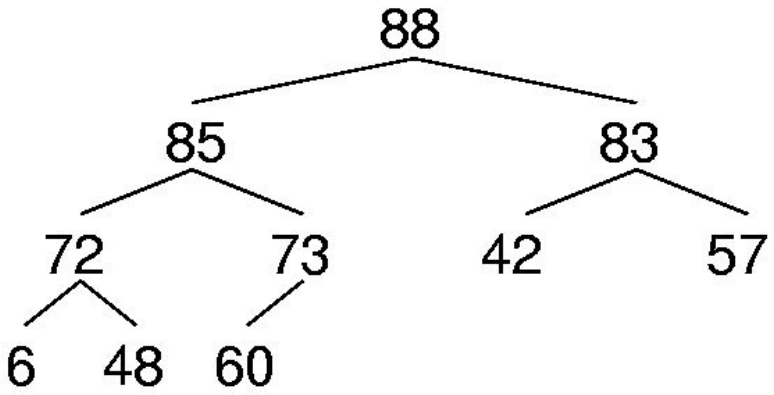
- Priority Queue를 구현할 때,
  BST보다 heap이 더 선호되는 이유?
  - heap은 배열로 구현 → 추가 메모리 사용 X
  - Build heap 이 O(n)에 처리됨!
    - BST는 insert를 매번 해야해서 O(n log n)
  - 삽입 삭제 복잡도가 같지만, 실제로는 heap이 더 빠름!
    - heap의 constant factor가 더 작음

Heaps

- Heap: heap property를 만족하는 완전이진트리:
  - Min-heap: 모든 노드는 자손 노드보다 **작은 값**을 가짐
  - Max-heap: 모든 노드는 자손 노드보다 **큰 값**을 가짐
- 값들이 **부분적으로 정렬**Partially ordered됨 (parent - child)
  - ⇔ Binary Search Tree
- Heap의 구현: 보통



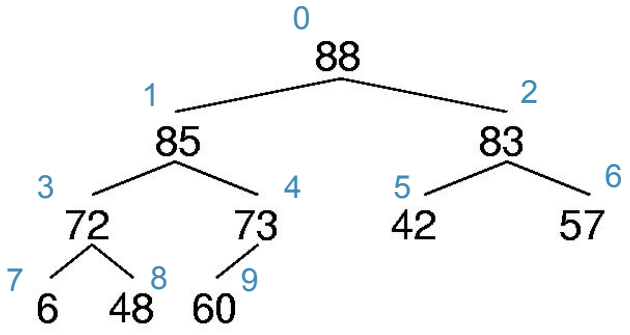
Max Heap Example

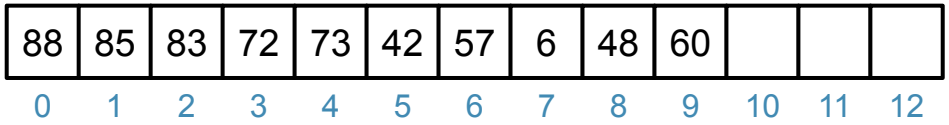


88	85	83	72	73	42	57	6	48	60			
0	1	2	3	4	5	6	7	8	9	10	11	12

Max Heap implementation (1)

- parent(r) = ⌊(r-1)/2⌋ (if r > 0)
- leftchild(r) = 2r+1 (if 2r+1 < n)
- rightchild(r) = 2r+2 (if 2r+2 < n)
- Node(r) is a leaf if ⌊n/2⌋ ≤ r < n
- Node(r) is internal if 0 ≤ r < ⌊n/2⌋

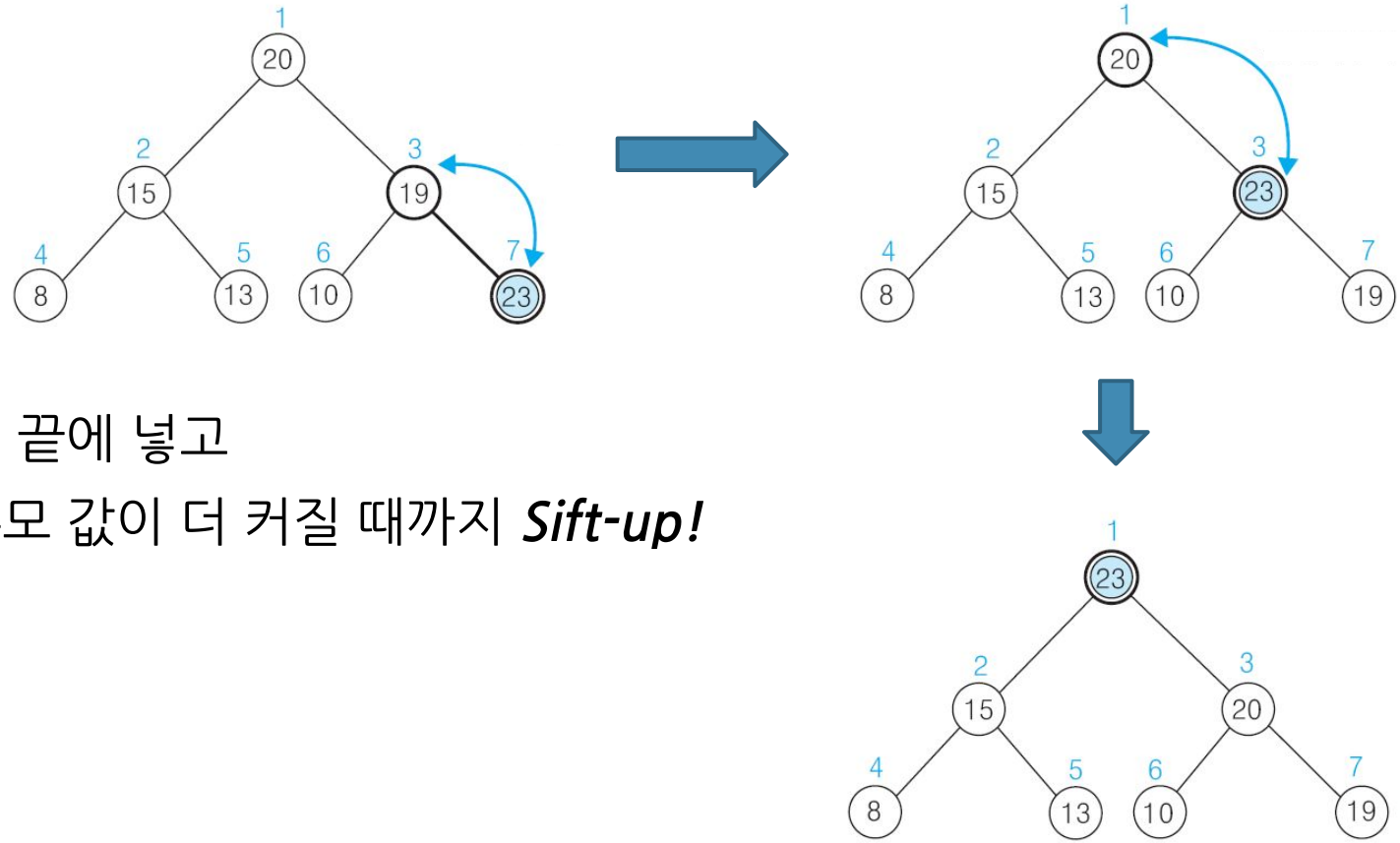




Max Heap implementation (2)

```
public class MaxHeap<E extends Comparable<? super E>> {
      private E[] Heap; // Pointer to heap array
      private int size; // Maximum size of heap
      private int n; // # of things in heap

      public MaxHeap(E[] h, int num, int max) {
            Heap = h; n = num; size = max; buildheap();
      }

      public int heapsize() { return n; }

      public boolean isLeaf(int pos) // Is pos a leaf position?
      { return (pos >= n / 2) && (pos < n); }

      public int leftchild(int pos) { // Leftchild position
            assert pos < n / 2 : "Position has no left child";
            return 2 * pos + 1;
      }

      public int rightchild(int pos) { // Rightchild position
            assert pos < (n - 1) / 2 : "Position has no right child";
            return 2 * pos + 2;
      }

      public int parent(int pos) {
            assert pos > 0 : "Position has no parent";
            return (pos - 1) / 2;
      }
```

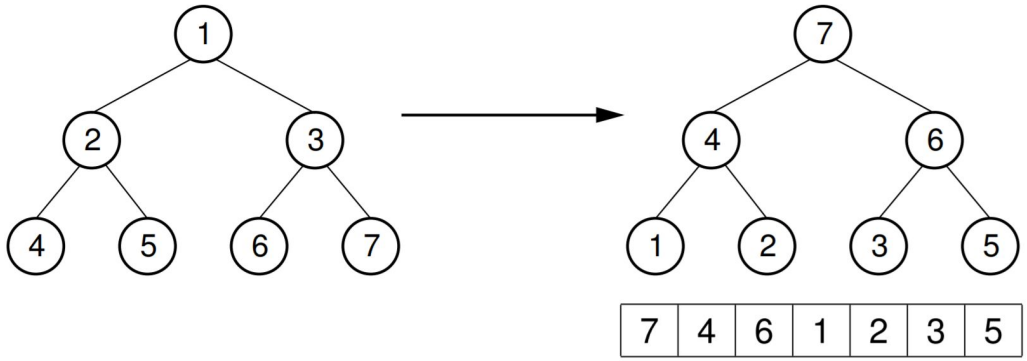
Insert



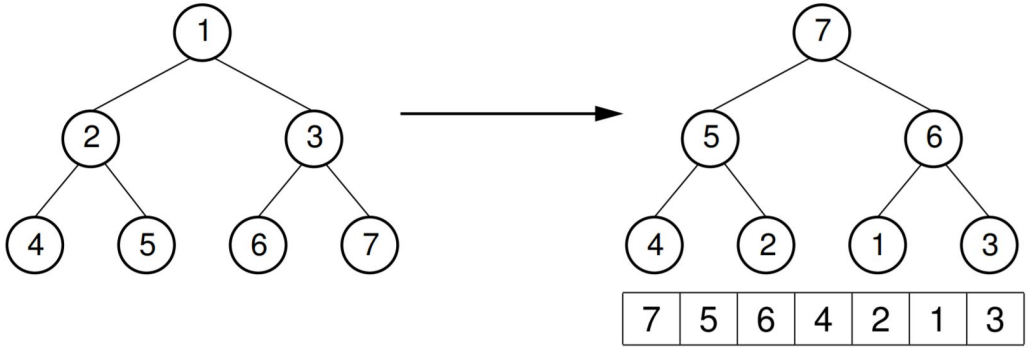
- 맨 끝에 넣고
- 부모 값이 더 커질 때까지 ***Sift-up!***

Building Heaps

- A Complete Binary tree (or an array) to a heap



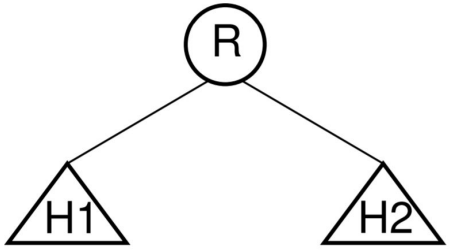
(4-2), (4-1), (2-1), (5-2), (5-4), (6-3), (6-5), (7-5), (7-6)



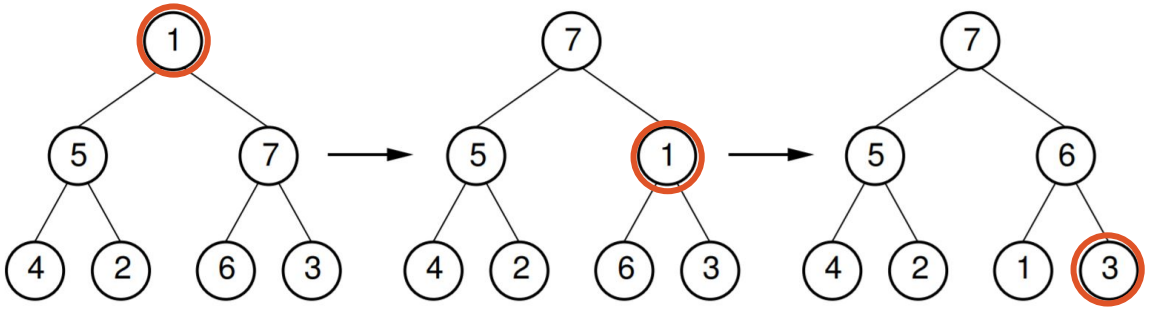
(5-2), (7-3), (7-1), (6-1)

Sift down

- How to build a heap?
  - **Sift down**: 작는 노드를 밑으로 내리기

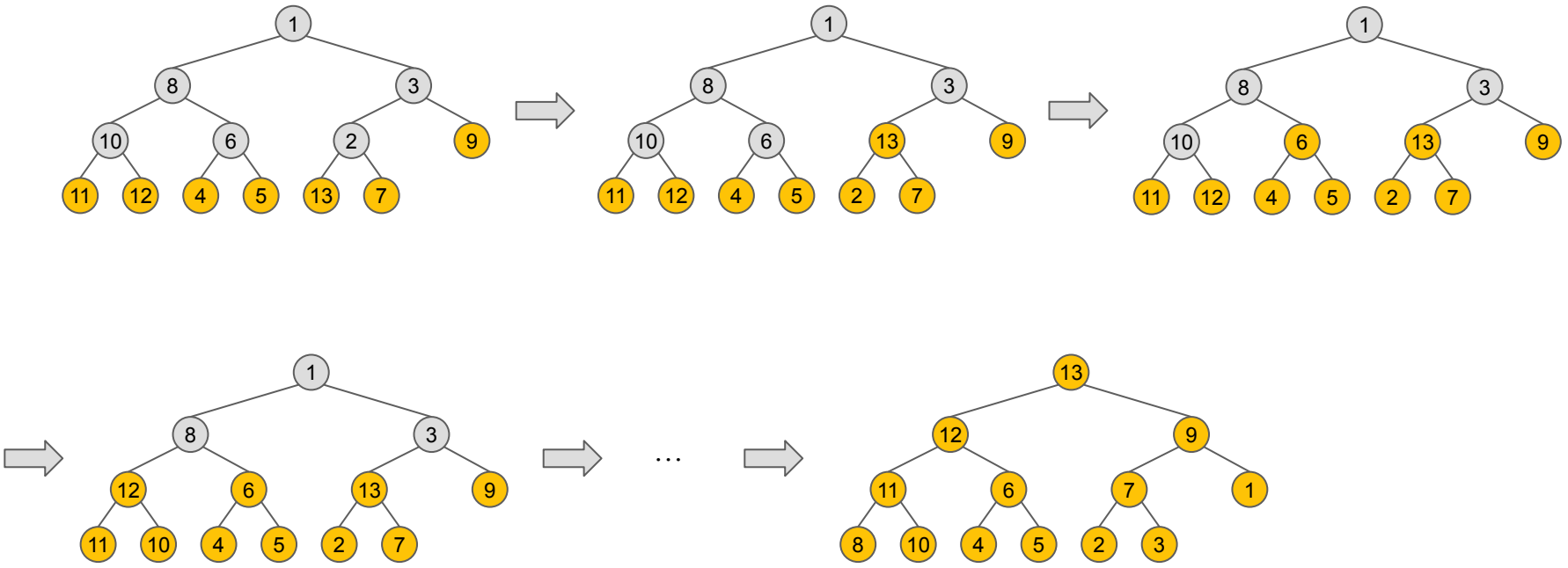


  - H1과 H2는 모두 Heap이다. R을 적절한 위치까지 내린다. 내려도 여전히 힙 구조를 따른다.



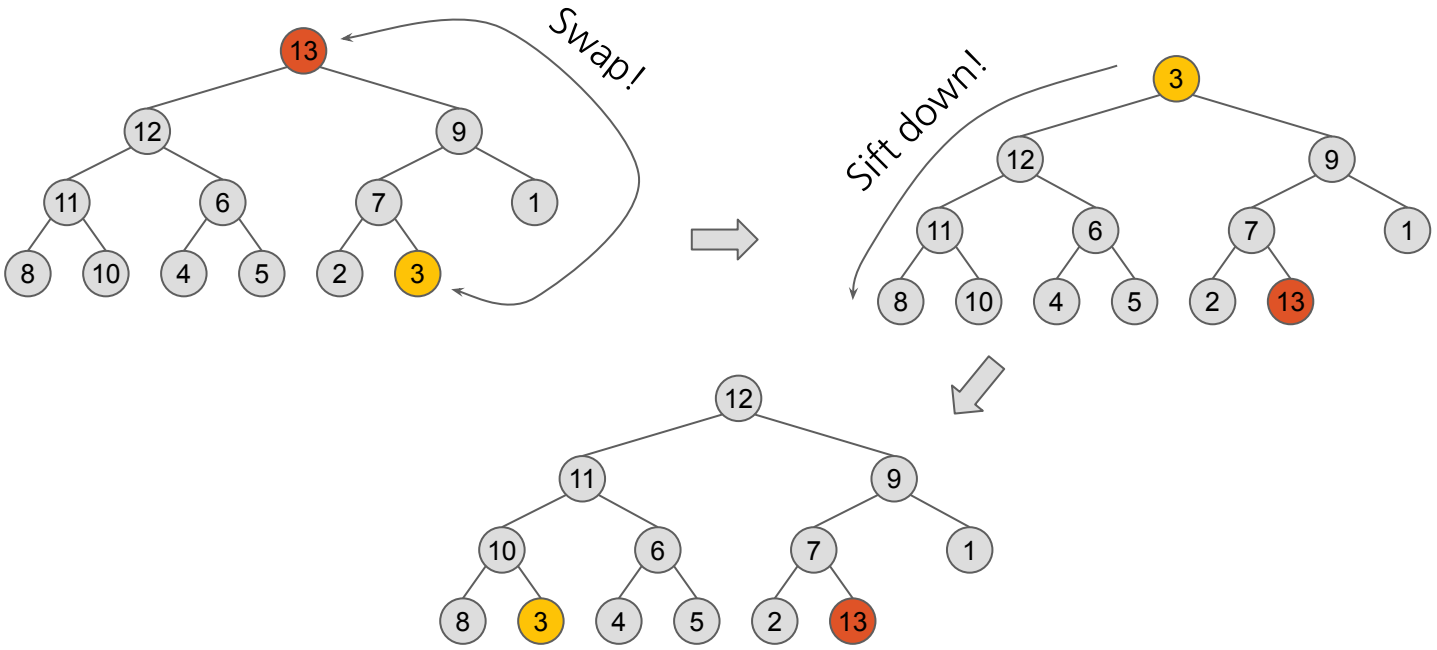
Build Heap using Sift Down

```
public void buildheap() // Heapify contents
{ for (int i = n / 2 - 1; i >= 0; i--) siftdown(i); }
```



RemoveMax

- 마지막 노드와 루트 노드의 위치를 바꾼다.
- 마지막 노드를 지운다
- 루트노드에서부터 **Sift down!**



Heap Building Analysis

- 값을 하나씩 넣어서 힙을 구축하는 경우
  - 빈 Heap에 값을 하나씩 입력(Sift-up) 하면…

$$\sum_{i=1}^{n} \log i = O(n \log n)$$

- 꽉찬 array에서 시작해서, 밑에서부터 작업한다
  - 끝에서부터, 각 노드를 Sift-down
  - 대부분의 노드는 아래쪽에 있다는 점에 주목
  - i를 tree의 바닥에서부터 시작하는 level이라고 하면…

$$\sum_{i=1}^{\log n} \frac{(i-1)n}{2^i} = \frac{n}{2} \sum_{i=1}^{\log n} \frac{i-1}{2^{i-1}} = \Theta(n)$$

$$\sum_{i=1}^{n} \frac{i}{2^i} = 2 - \frac{n+2}{2^n}$$

Questions?

Insert

```
public void insert(E val) {
     assert n < size : "Heap is full";
     int curr = n++;
     Heap[curr] = val;
     // Siftup until curr parent's key > curr key
     while ((curr != 0) && (Heap[curr].compareTo(Heap[parent(curr)]) > 0)) {
          swap(curr, parent(curr));
          curr = parent(curr);
     }
}
```

Sift Down

```
public void buildheap() // Heapify contents
{ for (int i = n / 2 - 1; i >= 0; i--) siftdown(i); }

private void siftdown(int pos) {
     assert (pos >= 0) && (pos < n) : "Illegal heap position";
     while (!isLeaf(pos)) {
          int j = leftchild(pos);
          if ((j < (n - 1)) && (Heap[j].compareTo(Heap[j + 1]) < 0))
               j++; // index of child w/ greater value
          if (Heap[pos].compareTo(Heap[j]) >= 0)
               return;
          swap(pos, j);
          pos = j; // Move down
     }
}

private void swap(int i, int j) {
     E tmp = Heap[i];
     Heap[i] = Heap[j];
     Heap[j] = tmp;
}
```

RemoveMax

- 마지막 노드와 루트 노드의 위치를 바꾼다.
- 마지막 노드를 지운다
- 루트노드에서부터 **Sift down!**

```
public E removemax() {
    assert n > 0 : "Removing from empty heap";
    swap(0, --n);
    if (n != 0)
        siftdown(0);
    return Heap[n];
}
```