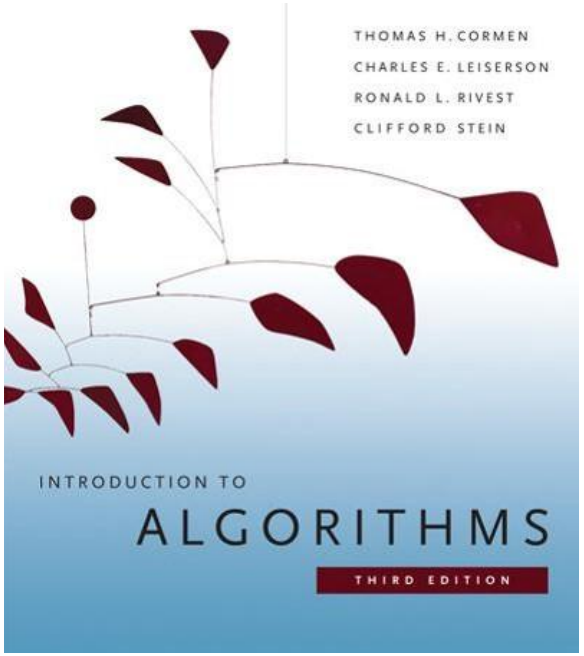
6.006- 알고리즘 개론



Courtesy of MIT Press. Used with permission.

4강

목차

- 우선순위 큐
- 힙
- 힙 정렬

우선순위 큐

요소들 집합 S를 구현하는 자료구조
각 요소들은 키와 관련이 있고, 다음 작업들을 지원한다.

insert(S, x) : 요소 x 를 집합 S에 삽입

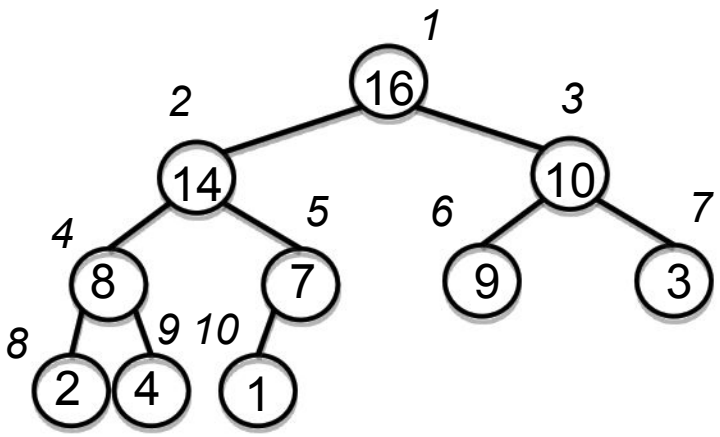
max(S) : 최대 키인 S의 요소를 반환

extract_max(S) : 최대 키인 S의 요소를 반환하고 집합 S에서 제거

increase_key(S, x, k) : 요소 x의 키를 새로운 값 k로 증가시킴
(k가 현재 값만큼 크다고 가정)

힙

- 우선순위 큐의 구현
- 완전 이진 트리로 시각화된 배열
- 최대 힙 특성 : 노드의 키가 자식 값들의 키보다 크다.
  (최소 힙도 마찬가지로 동일하게
  정의된다.)



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

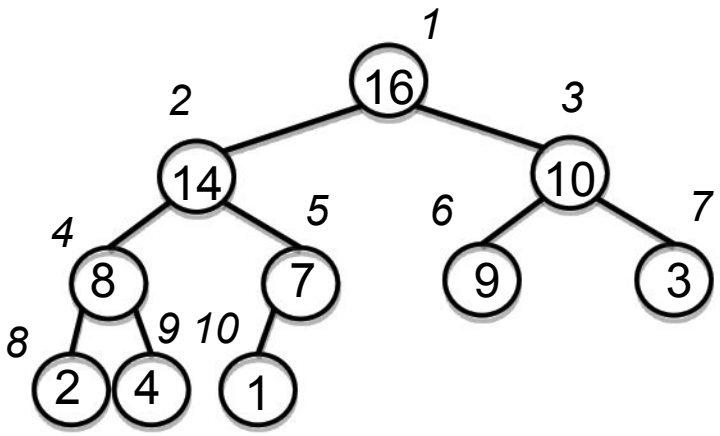
트리로서의 힙

트리의 루트: 배열의 첫 요소, i = 1 이다.

parent(i) =i/2: 노드의 부모의 인덱스를 반환

left(i)=2i: 노드의 왼쪽 자식의 인덱스를 반환

right(i)=2i+1: 노드의 오른쪽 자식의 인덱스를 반환



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

포인터가 필요 없음! 이진 힙의 높이는 O(lg n)

힙에서의 연산

build_max_heap : 정렬 되지 않은 배열로부터 최대-힙을 만든다.

max_heapify : 그 루트의 서브 트리에서 힙 특성을 위반한 걸 한 가지 고친다.

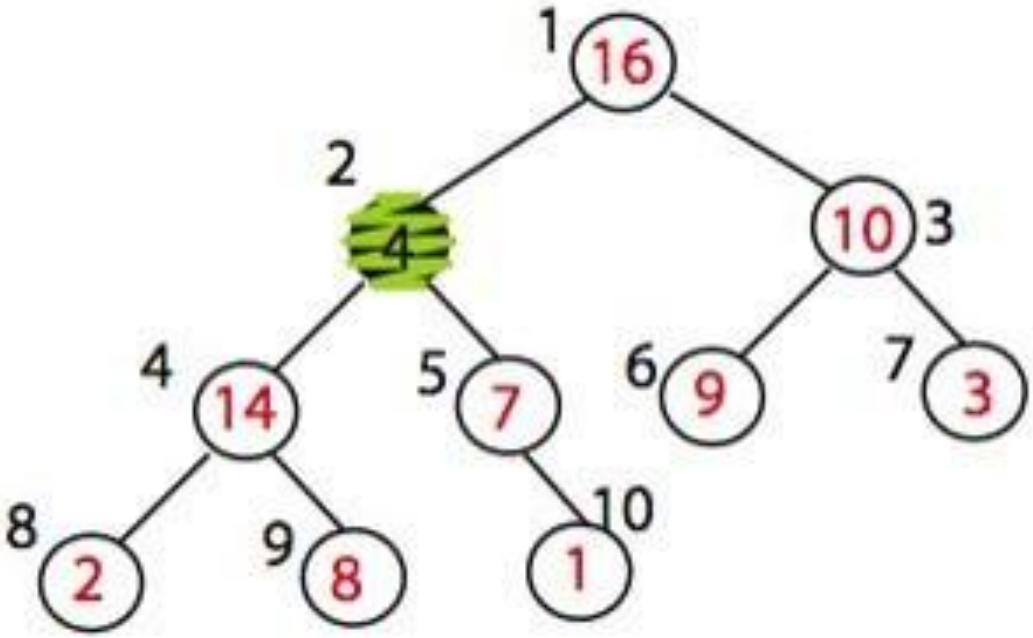
insert, extract_max, heapsort

최대-힙화

- **left(i)와 right(i) 서브 트리들이 최대-힙이라고 가정**

- 만약 요소 A[*i*]가 최대-힙 특성을 위반한다면, 요소 A[*i*]를 아래로 흘러 내려가도록 하면서 위반을 완전히 해결한다. 이 때 i의 서브 트리들을 최대-힙으로 만든다.

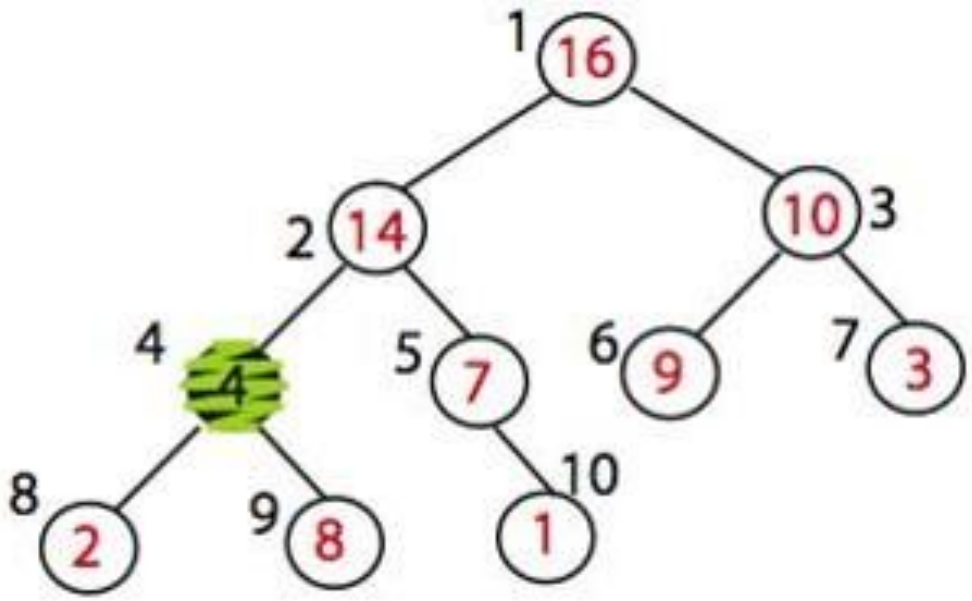
최대-힙화 (예시)



MAX_HEAPIFY (A,2)
heap_size[A] = 10

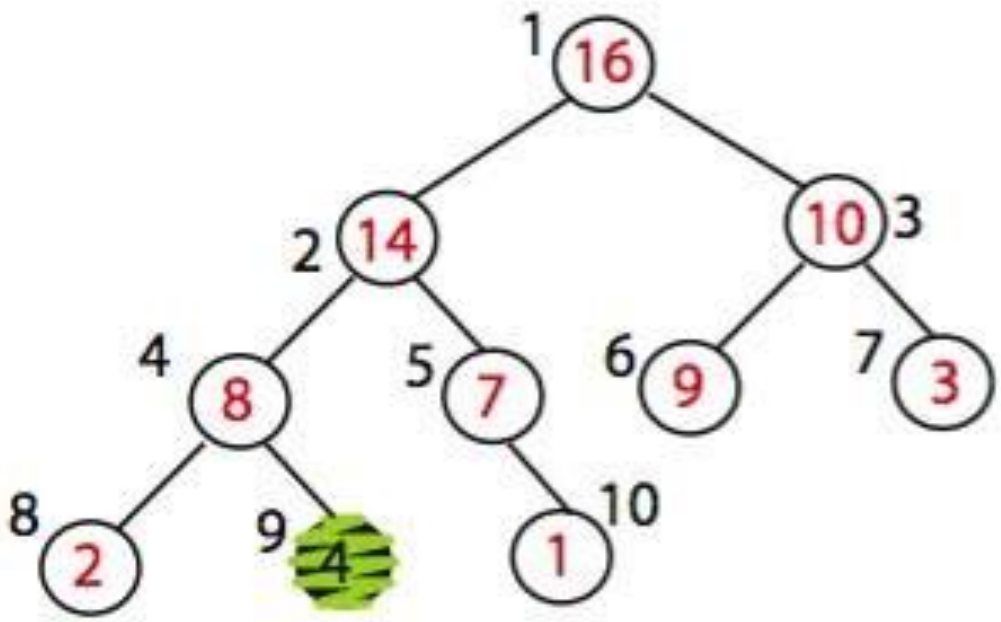
노드 10은 노드 5의 왼쪽 자식이지만 편의를 위해 오른쪽에 그렸다.

최대-힙화 (예시)



Exchange A[2] with A[4]
Call MAX_HEAPIFY(A,4)
because max_heap property
is violated

최대-힙화 (예시)



Exchange A[4] with A[9]
No more calls

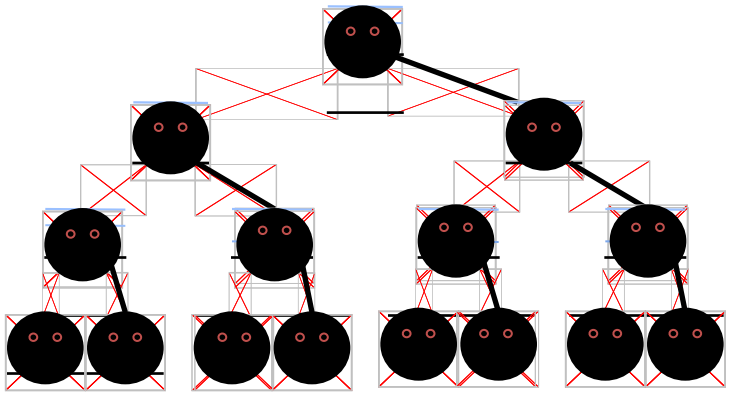
시간=? O(log n)

Max_Heapify 의사코드

l = left(i)
r = right(i)
if (l <= heap-size(A) and A[l] > A[i])
    then largest = l else largest = i
if (r <= heap-size(A) and A[r] > A[largest])
    then largest = r
if largest $\neq i$
    then exchange A[i] and A[largest]
        Max_Heapify(A, largest)

Build_Max_Heap(A)

A[1…n]를 최대-힙으로 변환



Build_Max_Heap(A):

for i=n/2 downto 1

do Max_Heapify(A, i)

왜 n/2 부터 시작할까?

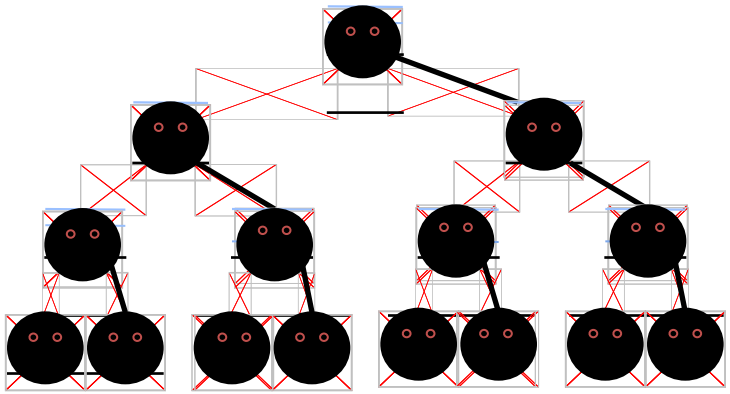
왜냐하면 요소 A[n/2 + 1 … n] 들이 트리의 모든 단말 노드들이라서이다.

2i > n, for i > n/2 이다.

시간=? 간단한 분석을 통하면 O(n log n)

Build_Max_Heap(A) 분석

A[1…n]를 최대-힙으로 변환



Build_Max_Heap(A):

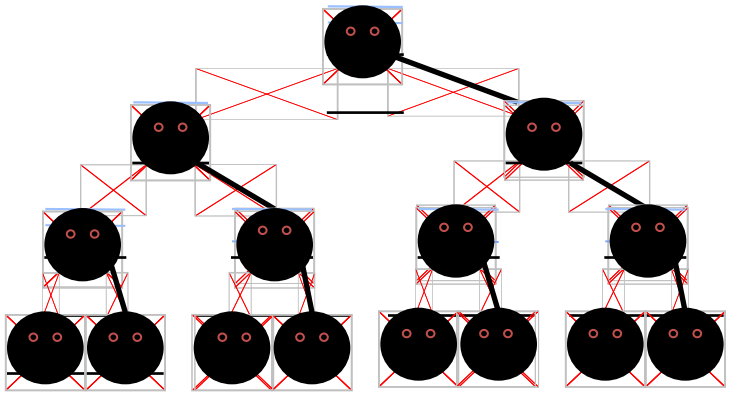
for i=n/2 downto 1

do Max_Heapify(A, i)

하지만 단말 노드보다 1단계 위에 있는 노드들은 Max_Heapify하는 데 O(1) 시간이 걸린다. 일반적으로 단말 노드보다 l 단계 위에 있는 노드들은 O(l) 시간이 걸린다. 1단계에서 n/4 노드들을 가지고, 2단계에서 n/8을 가지고, 한 개의 루트 노드만 남을 때까지 계속 이어진다. 마지막은 단말 노드로부터 log n 단계이다

Build_Max_Heap(A) 분석

A[1…n]를 최대-힙으로 변환



Build_Max_Heap(A):

for i=n/2 downto 1

do Max_Heapify(A, i)

for문에서 하는 전체 일의 양은 다음과 같이 계산된다.

: n/4 (1 c) + n/8 (2 c) + n/16 (3 c) + … + 1 (lg n c)

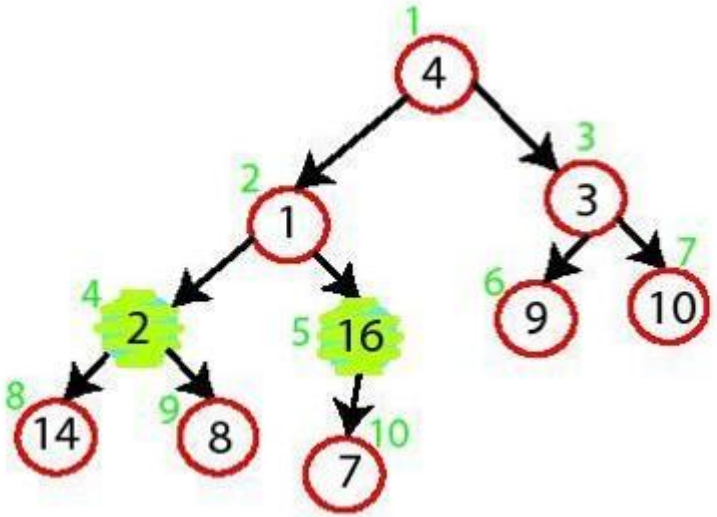
n/4 = 2^k 라고 가정하면 간단하게 바꿀 수 있다. :

$c\ 2^k(1/2^0 + 2/2^1 + 3/2^2 + \ldots (k+1)/2^k)$

괄호 안의 각 항은 상수로 한정됨!

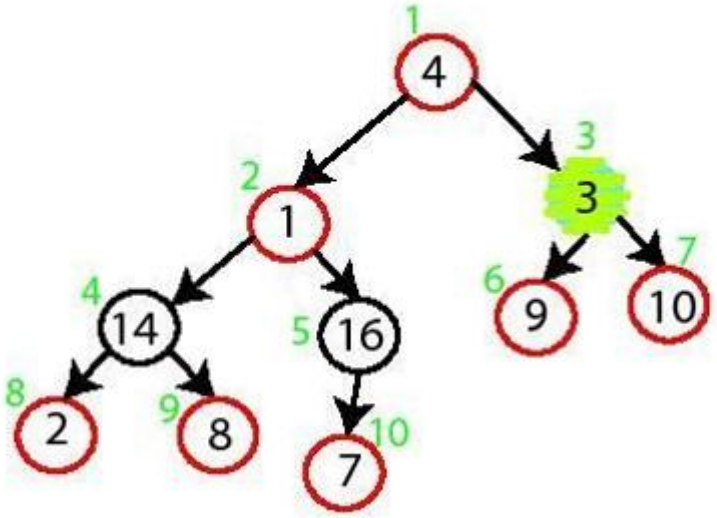
Build_Max_Heap 의 복잡도는 O(n)이다

Build-Max-Heap 예시



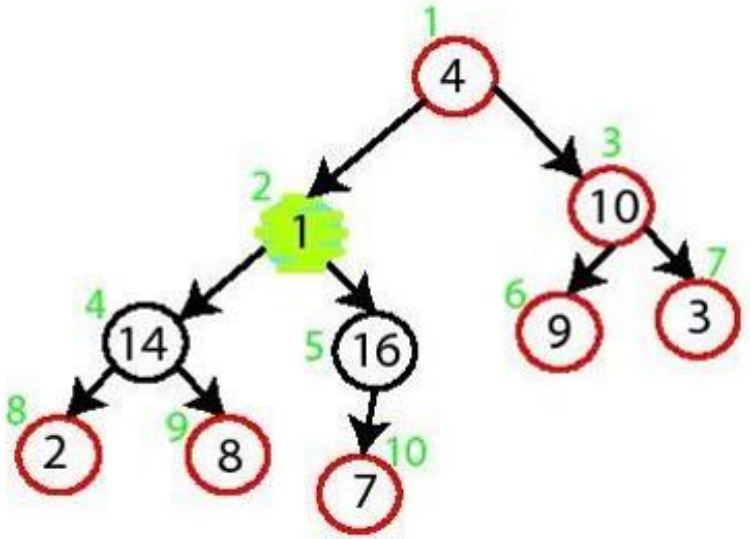
A	4	1	3	2	16	9	10	14	8	7

MAX-HEAPIFY (A,5)
no change
MAX-HEAPIFY (A,4)
Swap A[4] and A[8]

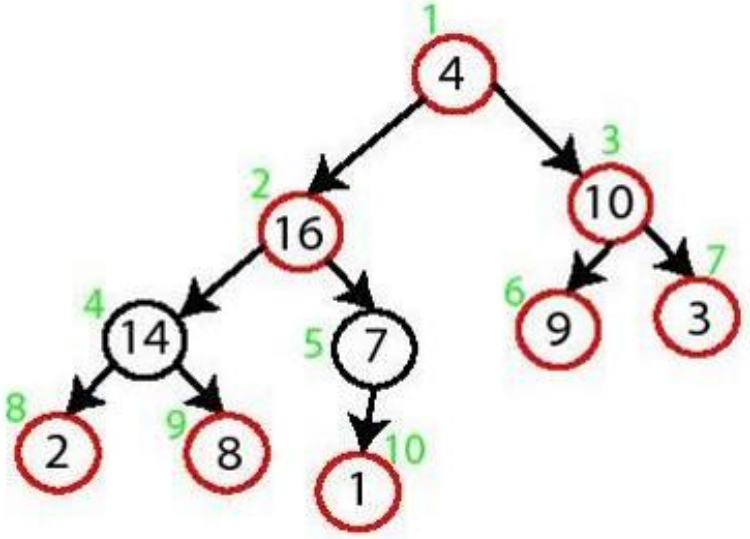


MAX-HEAPIFY (A,3)
Swap A[3] and A[7]

Build-Max-Heap 예시



MAX-HEAPIFY (A,2)
Swap A[2] and A[5]
Swap A[5] and A[10]



MAX-HEAPIFY (A,1)
Swap A[1] with A[2]
Swap A[2] with A[4]
Swap A[4] with A[9]

Build-Max-Heap

A	4	1	3	2	16	9	10	14	8	7

힙 정렬

정렬 전략:

1. 정렬되지 않은 배열에서 최대-힙을 만든다.

힙 정렬

정렬 전략:

1. 정렬된 배열에서 최대-힙을 만든다.

2. 최대 요소 A[1] 을 찾는다.

3. 요소 A[n]와 A[1]을 스왑한다. :
   이제 최대 요소는 배열의 끝에 위치한다!

힙 정렬

정렬 전략:

1. 정렬된 배열에서 최대-힙을 만든다.

2. 최대 요소 A[1] 을 찾는다.

3. 요소 A[n]와 A[1]을 스왑한다. :
   이제 최대 요소는 배열의 끝에 위치한다!

4. 힙에서 노드 n 을 제거한다.
   (힙 크기 변수를 줄이는 방법을 통해서)

힙 정렬

정렬 전략:

1. 정렬된 배열에서 최대-힙을 만든다.

2. 최대 요소 A[1] 을 찾는다.

3. 요소 A[n]와 A[1]을 스왑한다. :
   이제 최대 요소는 배열의 끝에 위치한다!

4. 힙에서 노드 n 을 제거한다.
   (힙 크기 변수를 줄이는 방법을 통해서)

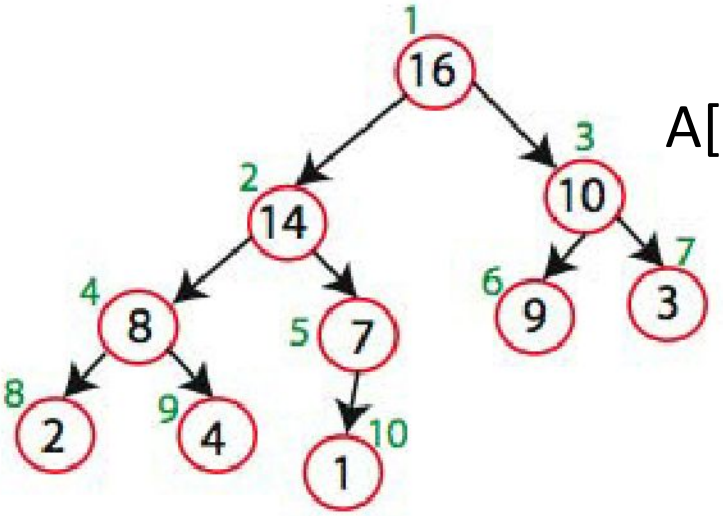
5. 새로운 루트는 아마 최대-힙 특성을 위반할 것이다. 하지만 그 자식 값들이 최대-힙이다.
   max_heapify로 이걸 해결한다.

힙 정렬

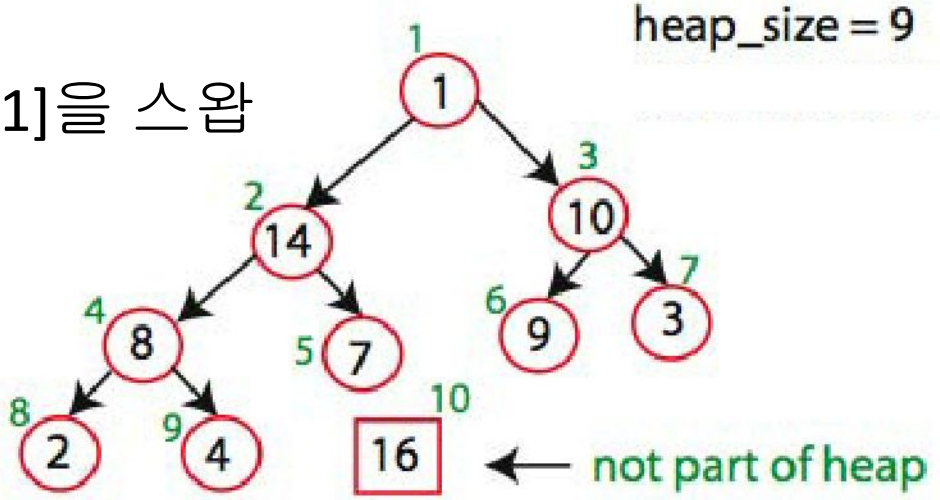
정렬 전략:

1. 정렬된 배열에서 최대-힙을 만든다. O(n)
2. 최대 요소 A[1] 을 찾는다. O(1)
3. 요소 A[n]와 A[1]을 스왑한다. :
   이제 최대 요소는 배열의 끝에 위치한다! O(1)
4. 힙에서 노드 n 을 제거한다.
   (힙 크기 변수를 줄이는 방법을 통해서)
5. 새로운 루트는 아마 최대-힙 특성을 위반할 것이다. 하지만 그 자식 값들이 최대-힙이다.
   max_heapify로 이걸 해결한다.
6. 힙이 비어있지 않다면 2단계로 돌아간다.

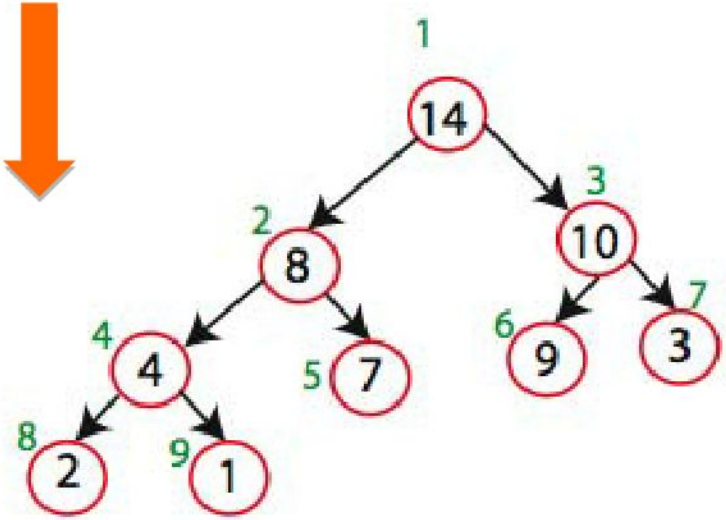
힙 정렬 예시



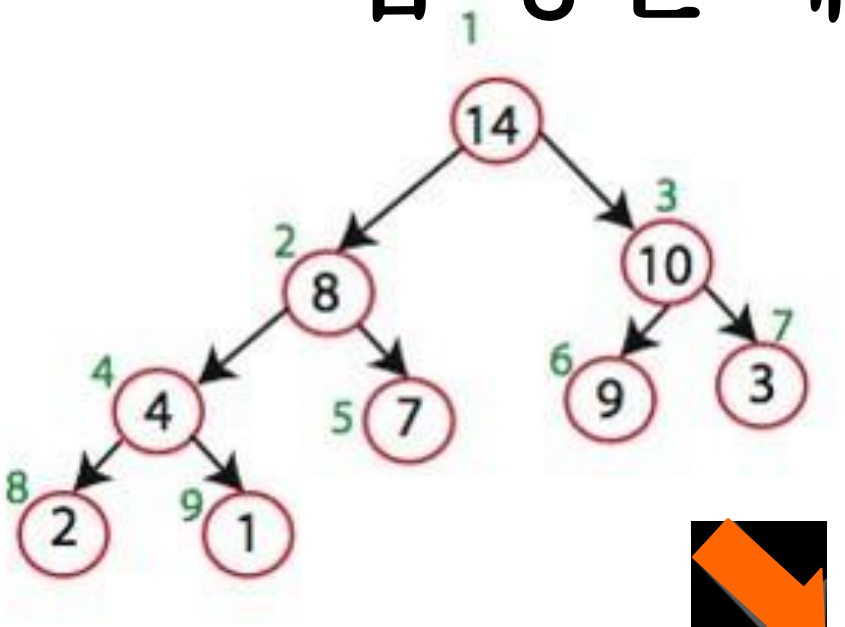
A[10]과 A[1]을 스왑



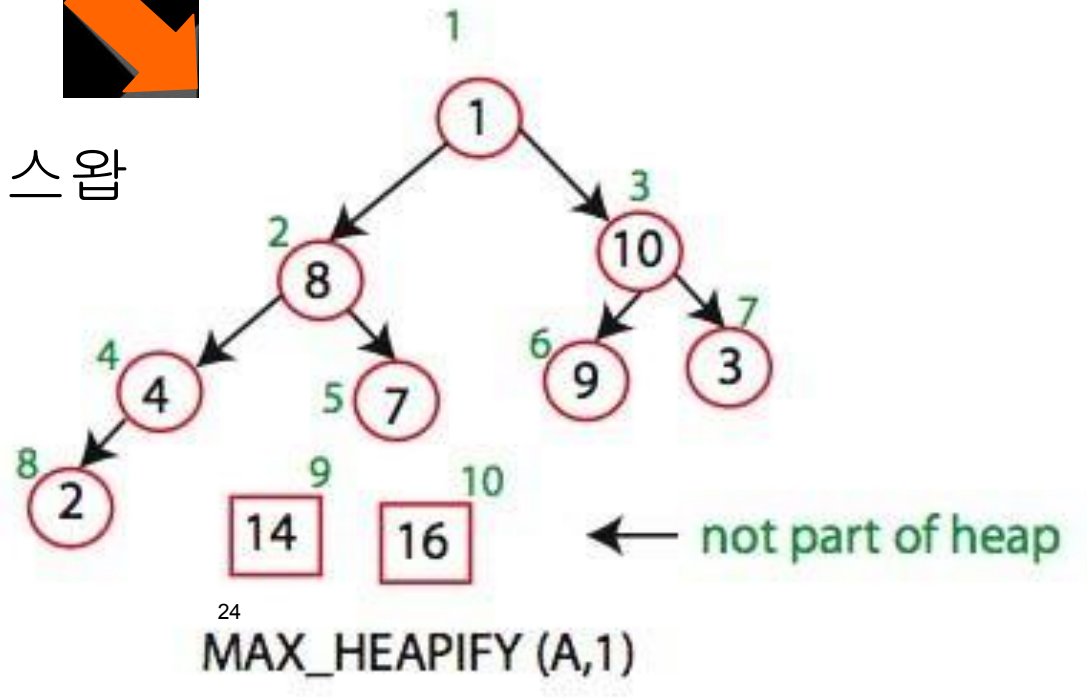
Max_heapify(A,1)



힙 정렬 예시

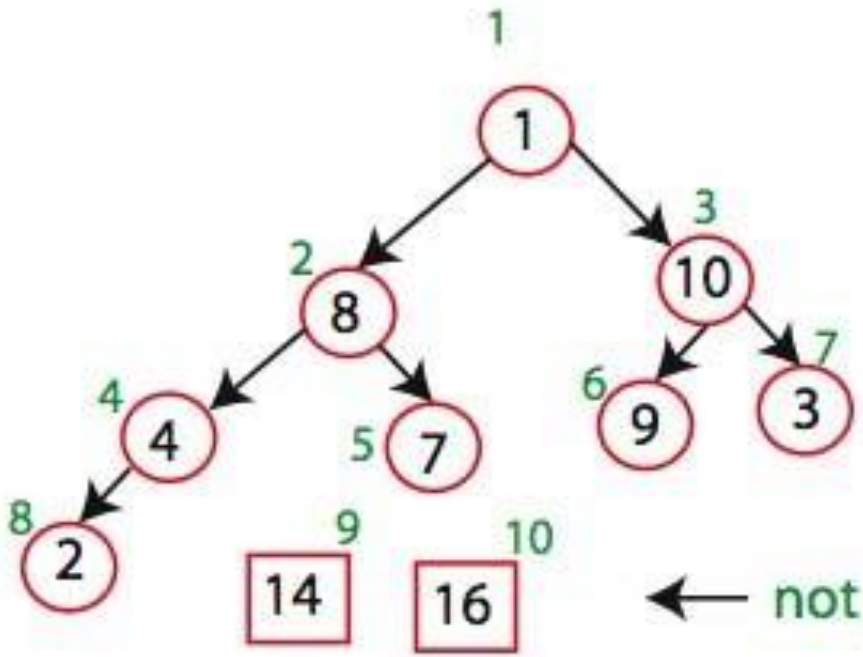


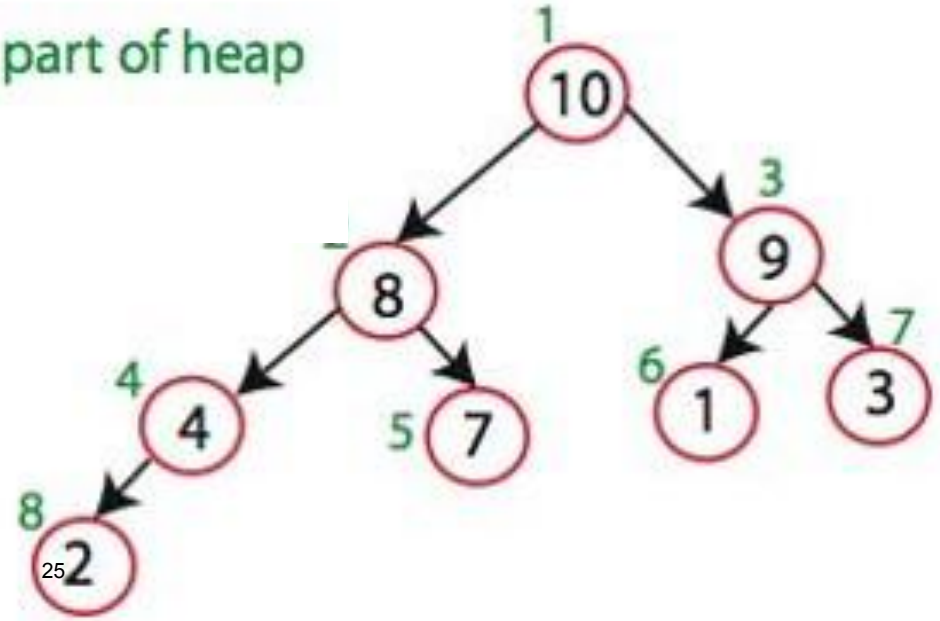
A[9]와 A[1]을 스왑



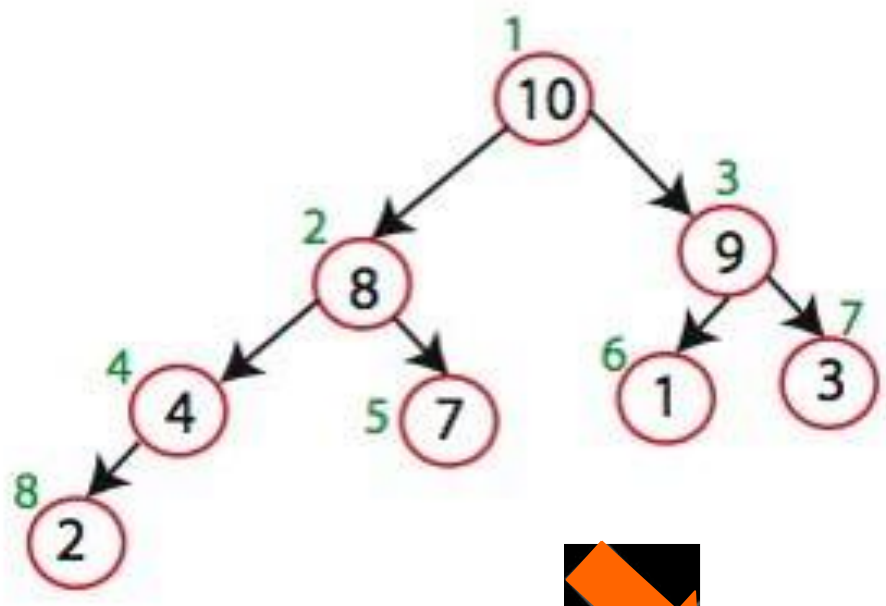
MAX_HEAPIFY (A,1)

힙 정렬 예시

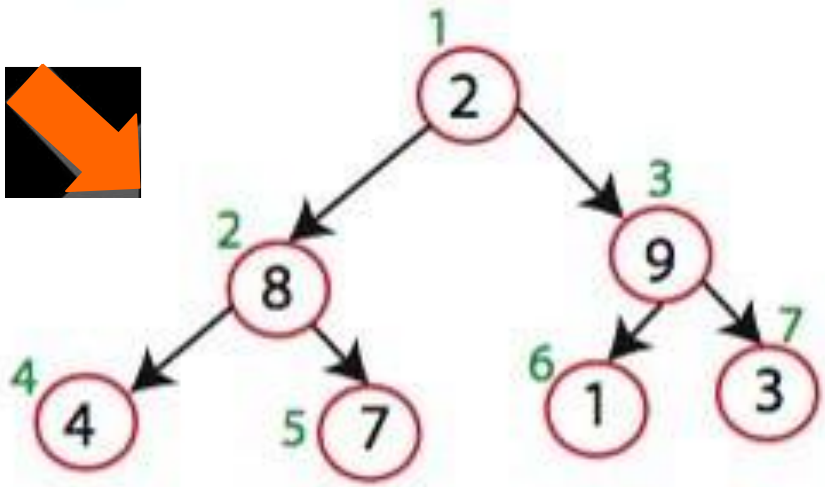

1
1
2
8
3
10
4
4
5
7
6
9
7
3
8
2
9
14
10
16
not part of heap
MAX_HEAPIFY (A,1)

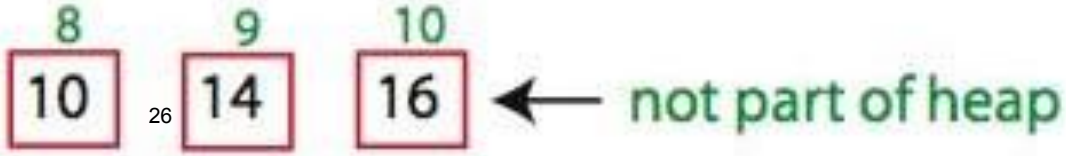

1
10
3
9
8
4
5
7
6
1
7
3
8
2

힙 정렬 예시



A[8]와 A[1]을 스왑





힙 정렬

실행 시간:

n번 반복 후 힙이 비게 된다.
각 반복은 스왑과 max_heapify 작업을 포함한다.
따라서 O(log n) 시간이 걸린다.

전체 실행 시간 O(n log n)

MIT OpenCourseWare
http://ocw.mit.edu

6.006 알고리즘의 기초
가을 2011

본 자료 이용 또는 이용 약관에 대한 정보를 확인하려면 다음의 사이트를 방문하십시오: http://ocw.mit.edu/terms.