# Heap Sort

# Sorting Algorithm

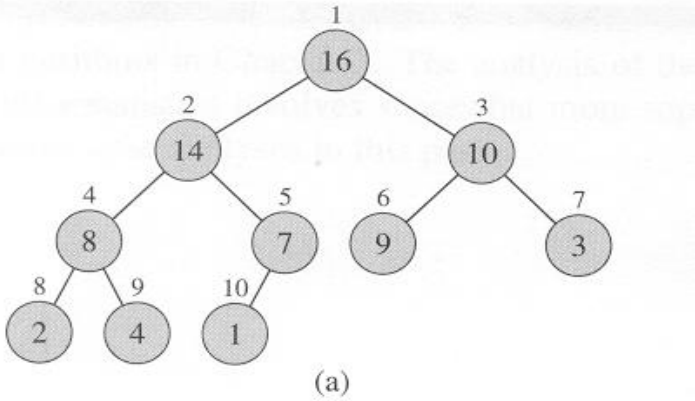
| Algorithm | Running Time | In-Place Sorting | Data structure |
|---|---|---|---|
| Insertion Sort | $O(n^2)$ | yes | array |
| Merge Sort | $\theta(n \lg n)$ | no | array |
| Heap Sort | $O(n \lg n)$ | yes | heap |

# Heaps

- Max-heap
  (1) nearly complete binary tree
  - 최 하단 level 을 제외하고 complete binary tree
  - 최 하단 level 은 왼쪽부터 채워짐

  (2) parent node 의 값 ≥ child node 의 값

(cf) Min-Heap
  (1) nearly complete binary tree
  (2) parent node 의 값 ≤ child node 의 값

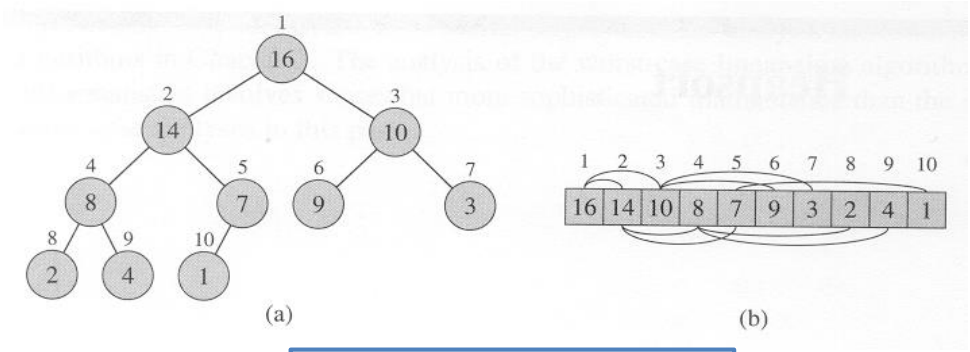


(a)

- Max-heap 의 특징
  – root node 에 항상 최대값이 존재

# Heaps

- Array 로 구현된 Max-heap
  - node 의 index 결정 방법
    (1) root 부터 하위 레벨로 진행
    (2) 같은 레벨에서는 왼쪽에서 오른쪽으로 진행




  - node 식별 함수

i: index of node
A[i]; value of node i
- root: A[1]
- PARENT(i)
  return $\lfloor \frac{i}{2} \rfloor$
- LEFT(i)
  return 2i
- RIGHT(i)
  return 2i+1

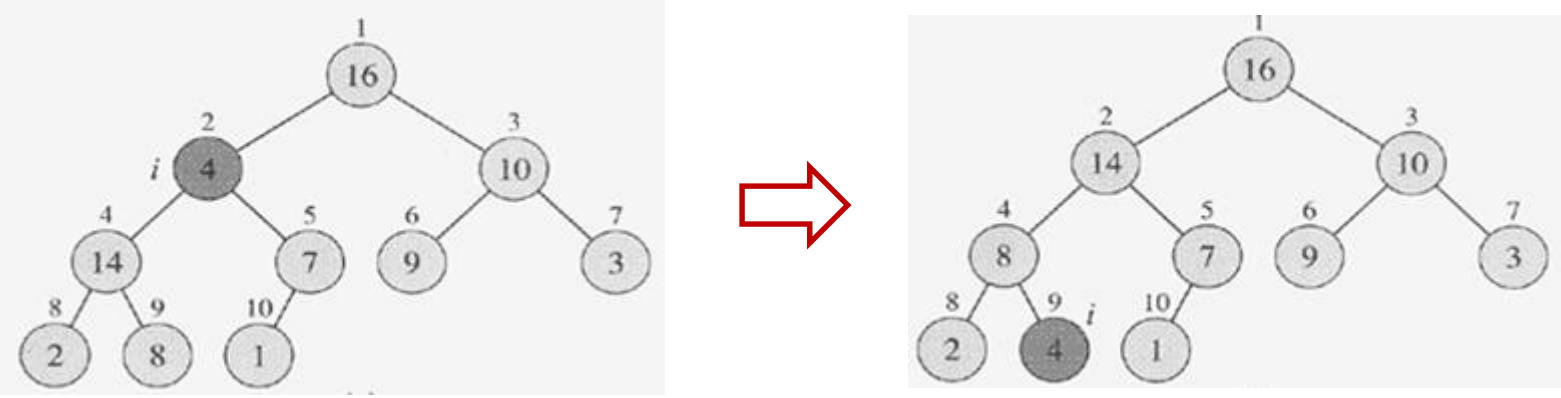
# Heaps

(Q) n-element heap 의 height 는?

(Q) A= <23,17,14,6,13,10,1,5,7,12> 는 max-heap 인가?

# Maintaining the Heap Property

- MAX-HEAPIFY(A, i)
  - 목적: index i 를 root 로 하는 array A 의 subtree 가 max-heap 이 유지 되도록 함.
  - Input: A; array, i; index
    - LEFT[i] 와 RIGHT[i] 를 root 로 하는 tree 는 max-heap
    - A[i] 는 children 값 보다 작음; Max heap 아님
  - Output: A ( i 를 root 로 하는 subtree 가 max-heap)



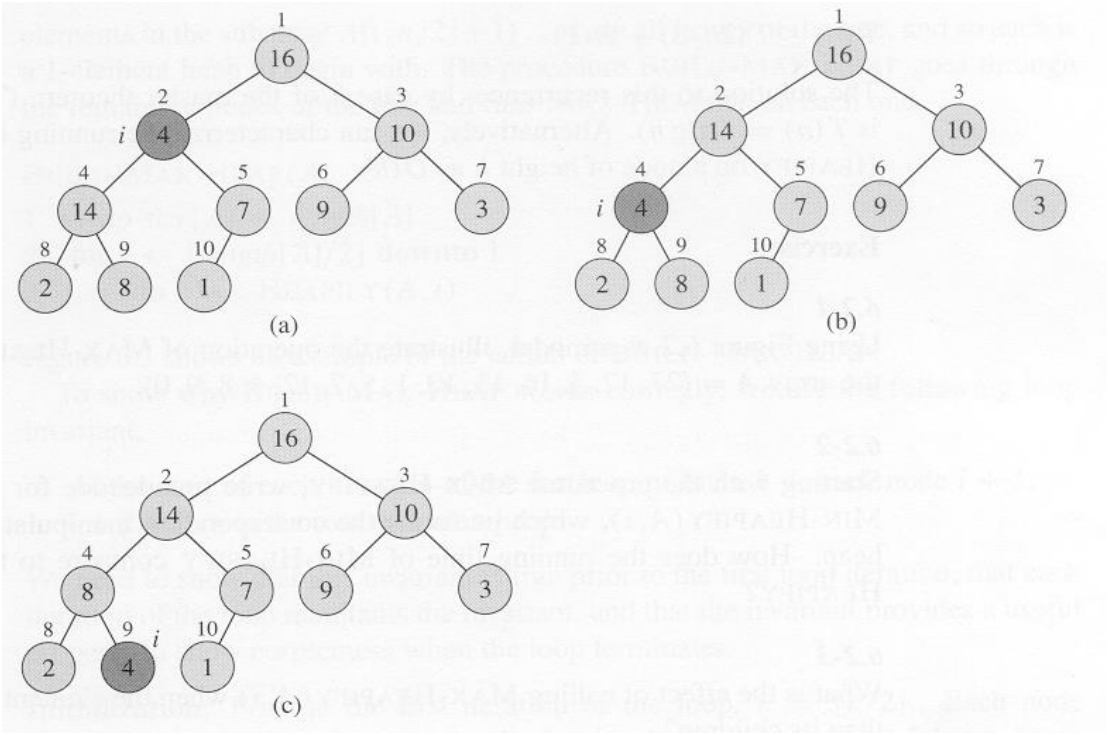
# Maintaining the Heap Property

- MAX-HEAPIFY(A, i)
  - Idea

    parent 와 child 의 값을 비교하여, child 의 값이 parent 의 값보다 크면 교환

  - Operation

MAX-HEAPIFY(A,2)



# Maintaining the Heap Property

- MAX-HEAPIFY(A, i)
  - Algorithm

    S1. index i, LEFT[i], RIGHT[i] 중 value 가 가장 큰 index 를 largest 에 저장

    S2. A[i] 와 A[largest] 값 교환

    S3. MAX_HEAPIFY (A, largest) 실행 ⇒ recursive 동작

  - Running Time
    - Subtree 의 height 에 비례
    - O (lg n)

(Q) A = <27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>

MAX-HEAPIFY(A, 3), A 의 변화과정 ?

```
MAX-HEAPIFY(A, i)
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l ≤ heap-size[A] and A[l] > A[i]
4     then largest ← l
5     else largest ← i
6  if r ≤ heap-size[A] and A[r] > A[largest]
7     then largest ← r
8  if largest ≠ i
9     then exchange A[i] ↔ A[largest]
10          MAX-HEAPIFY(A, largest)
```

# Maintaining the Heap Property

(Q) A = <27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>

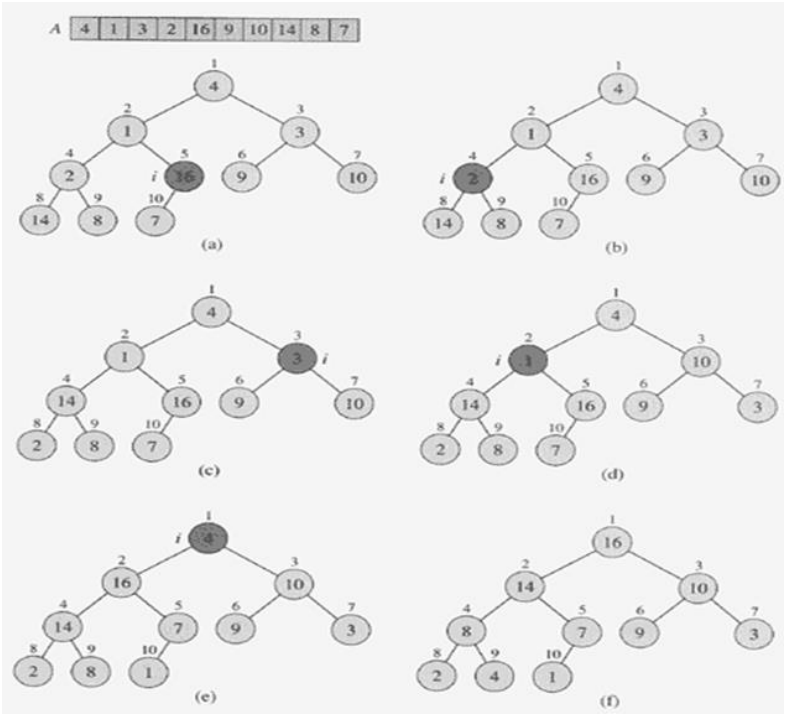
MAX-HEAPIFY(A, 3), A 의 변화과정 ?

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 27 | 17 | 3 | 16 | 13 | 10 | 1 | 5 | 7 | 12 | 4 | 8 | 9 | 0 |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

# Building a Heap

- BUILD-MAX-HEAP(A)
  - 목적: 임의 배열 A 를 max-heap 으로 만듬
  - Input : A
  - Output : A (max heap)
  - Idea:
    - bottom → up 으로 MAX-HEAPIFY 진행
    - non-leaf node 의 마지막 index 부터 index 를 하나씩 감소하며 MAX-HEAPIFY(A,I) 실행
  - Operation

(Q) n-element heap 에서
non-leaf node 의
마지막 index 는 ?



# Building a Heap

- BUILD-MAX-HEAP(A)
  - Algorithm

```
BUILD-MAX-HEAP(A)
1  heap-size[A] ← length[A]
2  for i ← ⌊length[A]/2⌋ downto 1
3      do MAX-HEAPIFY(A, i)
```

  - Running Time

    O (n lg n); MAX-HEAPFY (O(lg n)) 를 n/2 번 수행

    O(n); asymptotically tight bound => See text P.135

# Building a Heap

(Q) A = <5, 3, 17, 10, 84, 19, 6, 22, 9> => BUILD-MAX-HEAP(A)

| 5 | 3 | 17 | 10 | 84 | 19 | 6 | 22 | 9 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

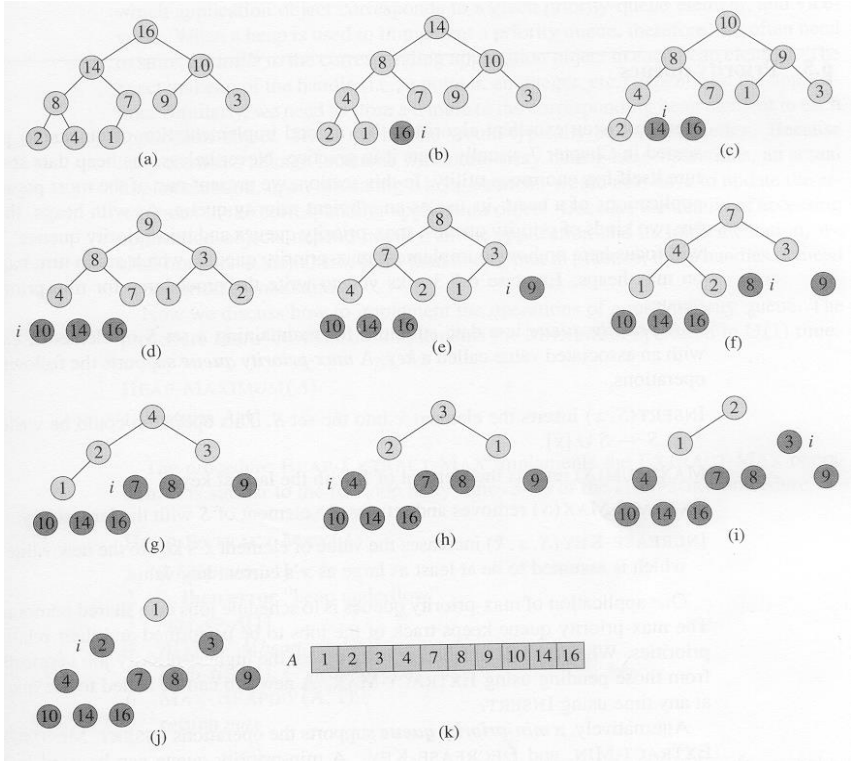
Quiz.

# Heapsort

- HEAPSORT(A)
  - 목적: 임의 배열 A 를 오름차순으로 정열
  - Input : A $< a_1, a_2, \cdots, a_n >$
  - Output: A , $< a_1', a_2', \cdots, a_n' >$ $a_1' \le a_2' \le \cdots \le a_n'$
  - Idea

    A를 max-heap 으로 구성하고,

    root node 의 값 추출 → 반복
  - Operation



# Heapsort

- HEAPSORT(A)
  - Algorithm

    S1. array A 를 max-heap 으로 변환 → BUILD-MAX-HEAP(A)

    S2. A[1] 과 A[heap size] 교환

    S3. heap size ← heap size –1

    S4. MAX-HEAPIFY(A,1) 실행, Go to S2

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i ← length[A] downto 2
3      do exchange A[1] ↔ A[i]
4         heap-size[A] ← heap-size[A] − 1
5         MAX-HEAPIFY(A, 1)
```

  - Running time
    - Build-MAX-HEAP; O(n) (or O (n lg n))
    - MAX-HEAPFY (O(lg n)) 를 n 번 수행; O (n lg n)

    ⇒ O(n) + O(n lg n) = O(n lg n)

  - ➢ In-place sort
  - ➢ Incremental approach

# Heapsort

(Q) A = < 5, 3, 17, 10, 84, 19, 6, 22, 9 > → HEAPSORT(A)

| 5 | 3 | 17 | 10 | 84 | 19 | 6 | 22 | 9 |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

Quiz.