------------------------------ 6 ------------------------------

Heapsort

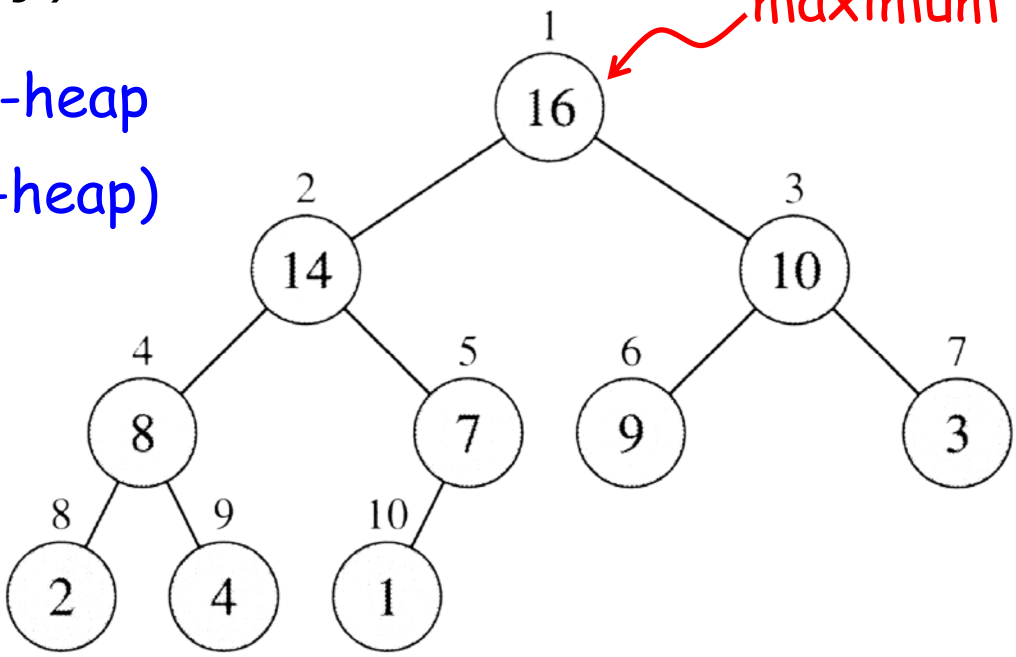
(binary) ①

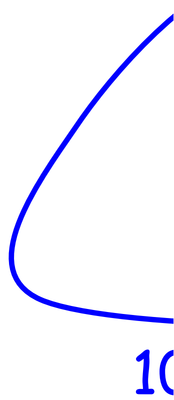
6.1 ***Heap***: an array that can be viewed as a complete binary tree in which each node has a key not smaller than those of its children ② (***heap property***).

maximum

max-heap

(min-heap)





(a)

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

(b)

- $A[1]$ is the root
- $Parent(i)=\lfloor i/2 \rfloor$ (shifting i right one bit)
- $Left(i)=2i$ (shifting i left one bit)
- $Right(i)=2i+1$
- A heap of n nodes has height $\Theta(\lg n)$ $= \lfloor \lg n \rfloor$

max # of edges from root to a leaf

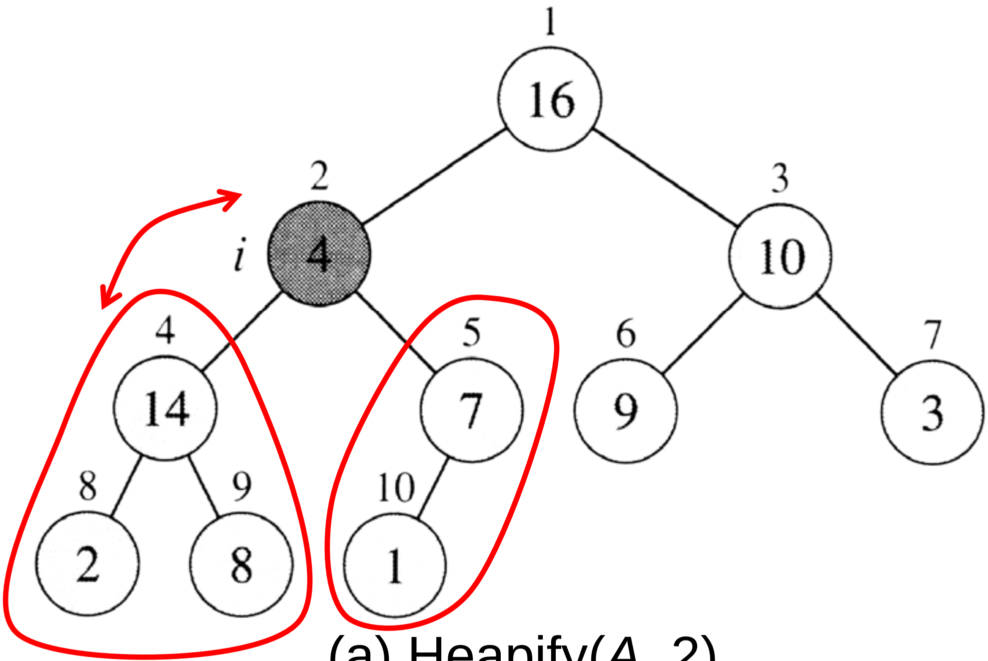
6.2 Maintaining the heap property

***Heapify*(*A*, *i*)**: Binary trees rooted at *Left*(*i*) and *Right*(*i*) are heaps, but *A*[*i*] may be smaller than its children.

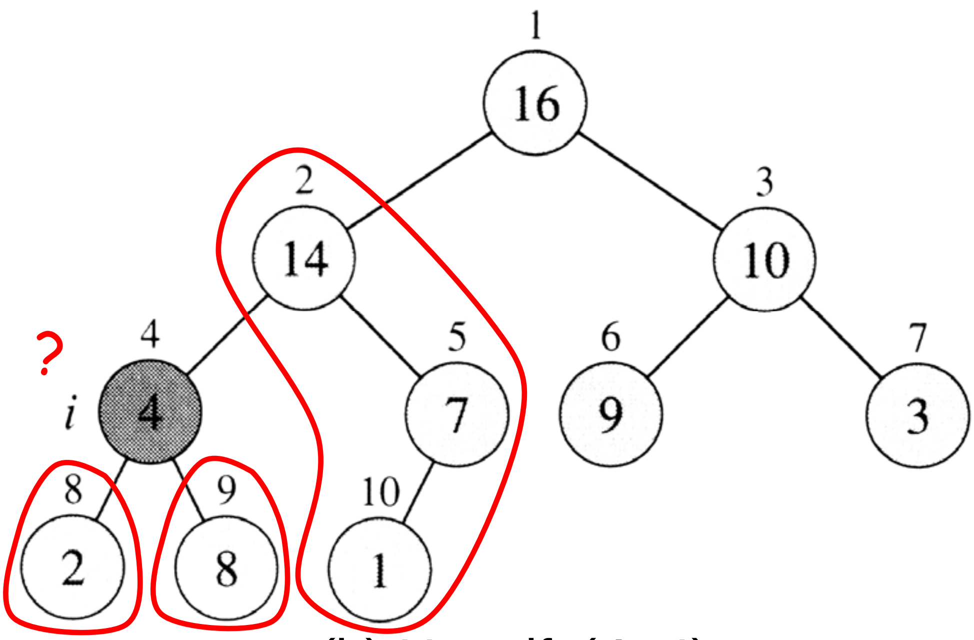
6-2a

* merge two smaller heap into one

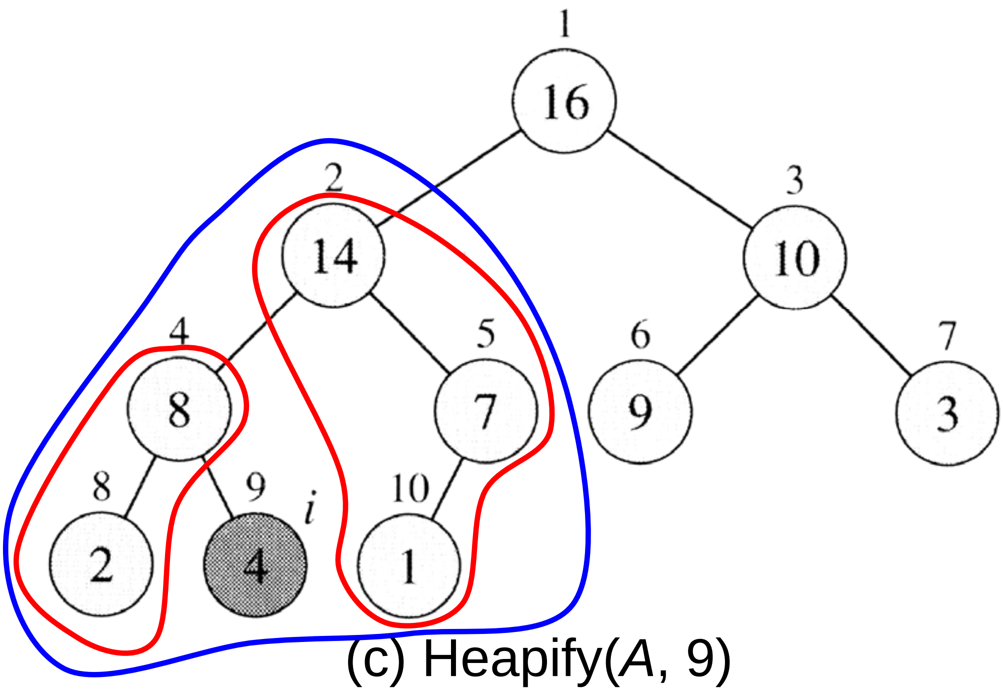
Example:



(a) Heapify(*A*, 2)



(b) Heapify(*A*, 4)



(c) Heapify(A, 9)

6-3x

① 用 recurrence　　subtree i 中 node #　　i 的 高 度

- $T(n) \le T(2n/3) + \Theta(1) = O(\lg n) = O(h)$, where n is the number of nodes in the subtree rooted at $A[i]$ and h is the height of $A[i]$.

② 用 高 度 說 明

6.3 Building a heap

***Build-Heap*(*A*)**

```
heap-size[A] ← length[A]
for i ← ⌊length[A]/2⌋ downto 1 do
    Heapify(A, i)
```

$$T(n) \le \sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \times \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) = O(n \times 2)$$

6-3a

$$= O(n).$$

(Ex. 6.3-3)

(at most # nodes at h)

$$\sum_{k=1}^{\infty} kx^k = \frac{x}{(1-x)^2} \quad (x < 1)$$

OR: $T(n)=2T(n/2)+\lg n$ (Assume $n = 2^{h+1} - 1$.)

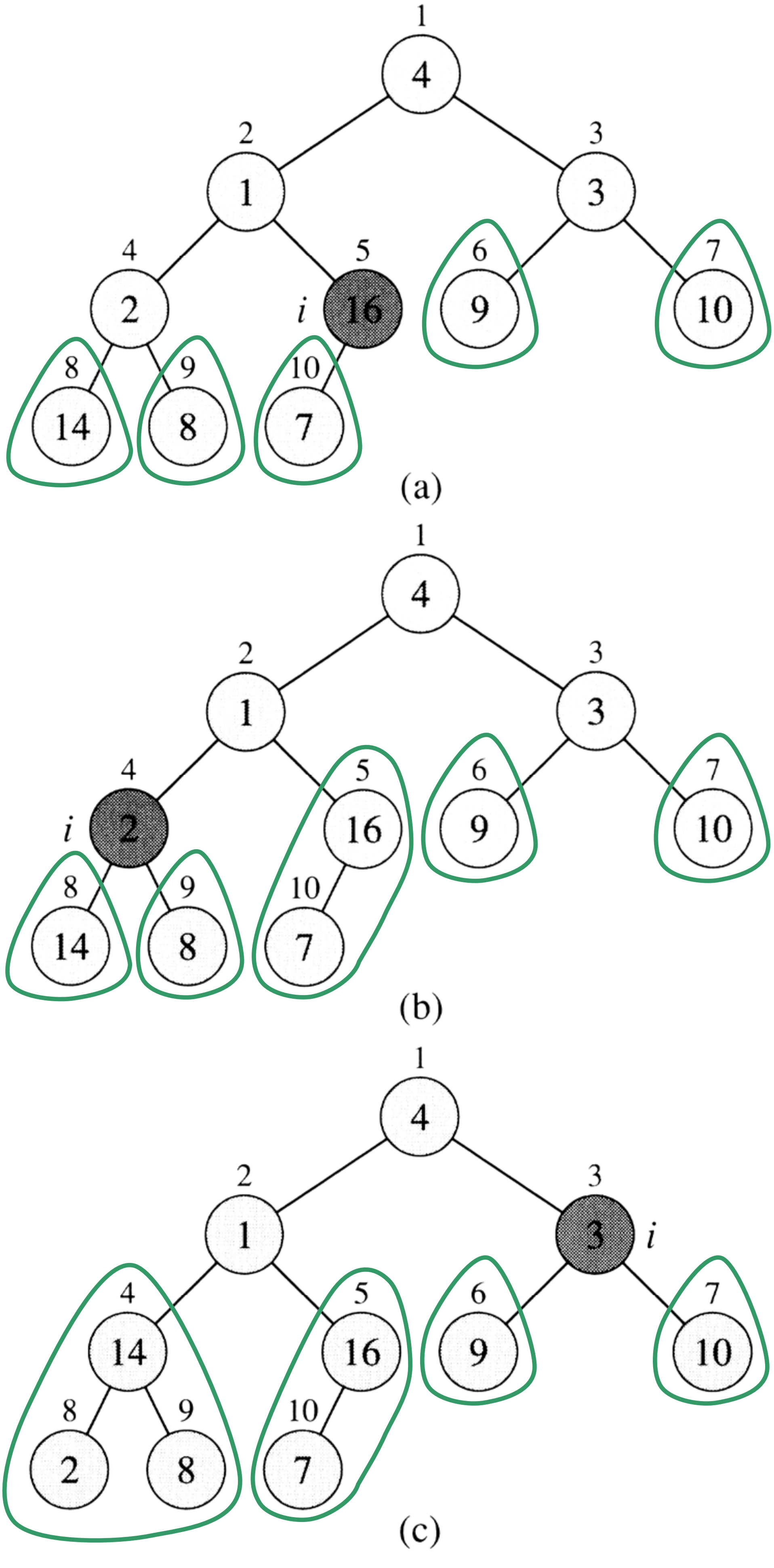
$\lfloor \ \rfloor$

6-3y

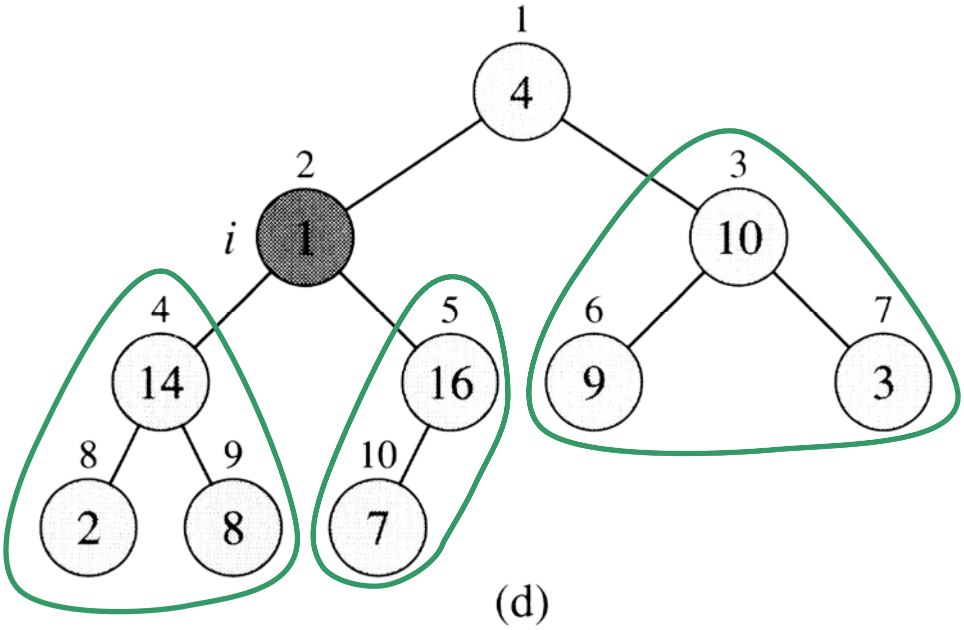
Append dummy nodes

Example:

A									
4	1	3	2	16	9	10	14	8	7

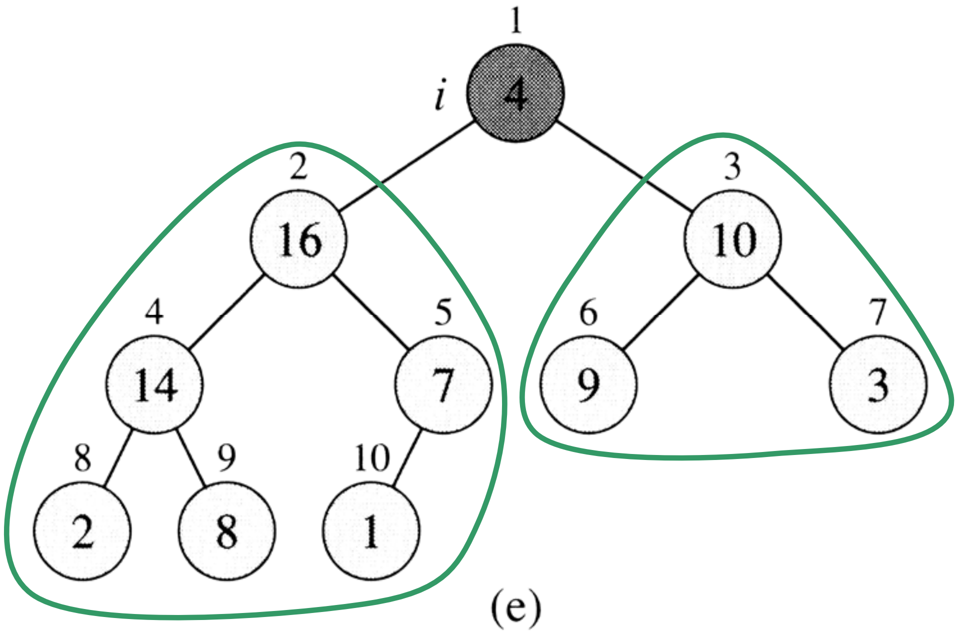


(a) (b) (c)



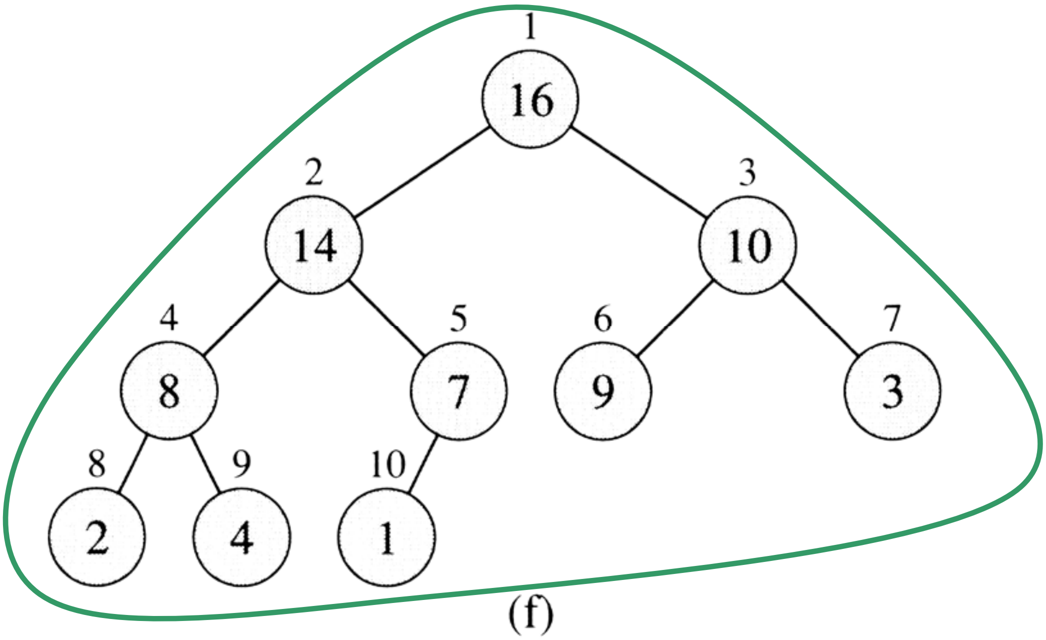
1
4
2
i
1
3
10
4
14
5
16
6
9
7
3
8
2
9
8
10
7

(d)



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

(e)



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

(f)

6.4 Heapsort

Stage 1: Build a heap

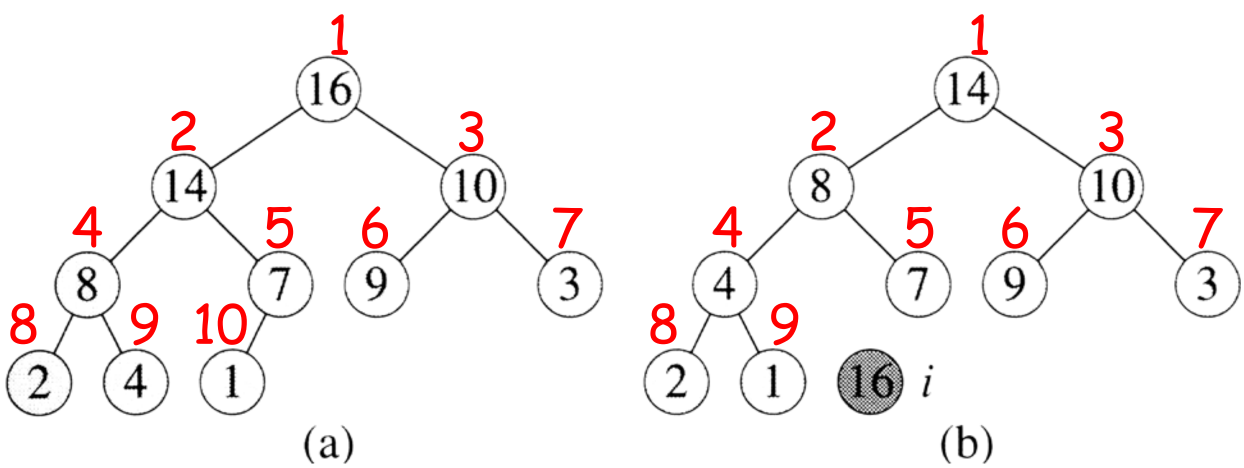
Stage 2: Repeatedly delete the root of the heap. 6-6x

HeapSort(*A*)

```
Build-Heap(A)
for i ← length[A] downto 2
    do  exchange A[1] ↔ A[i]
        heap-size[A] ← heap-size[A]–1
        Heapify(A, 1)
```

- $T(n) = O(n) + O((n\text{-}1) \times \lg n)$ 6-6y
  $= O(n\lg n)$

Example: sort *A*={4, 1, 3, 2, 16, 9, 10, 14, 8, 7}.



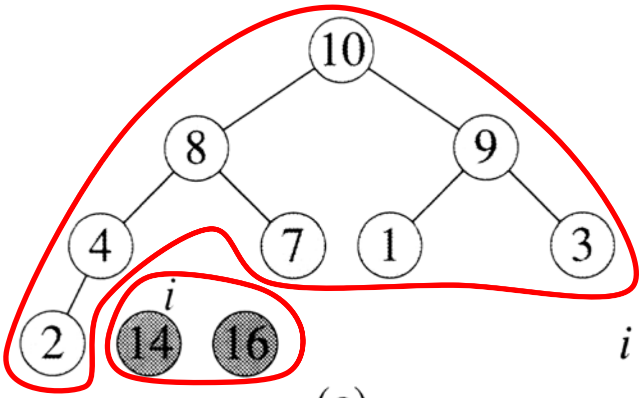
n = 10

16	14	10	8	7	9	3	2	4	1

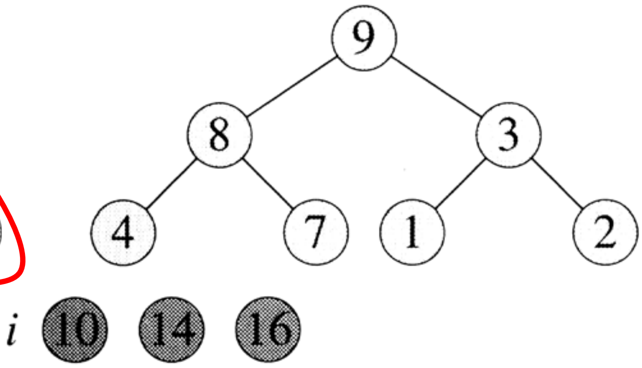
n = 9

1	14	10	8	7	9	3	2	4	16

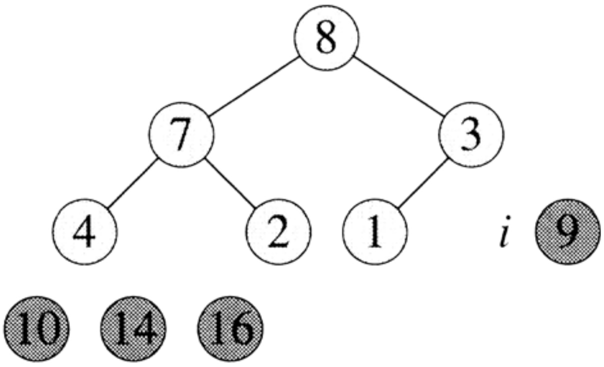
Heapify(A, 1)


10
8 9
4 7 1 3
i
2 14 16

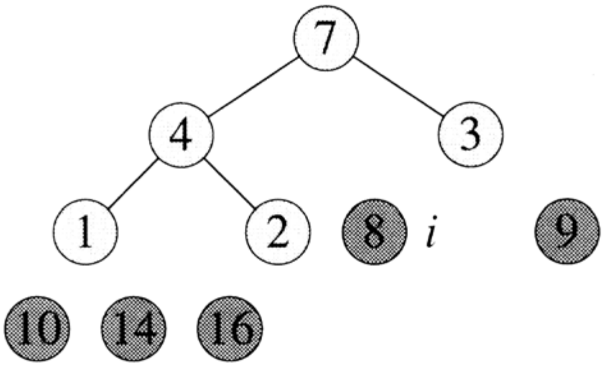
(c)


9
8 3
4 7 1 2
i 10 14 16

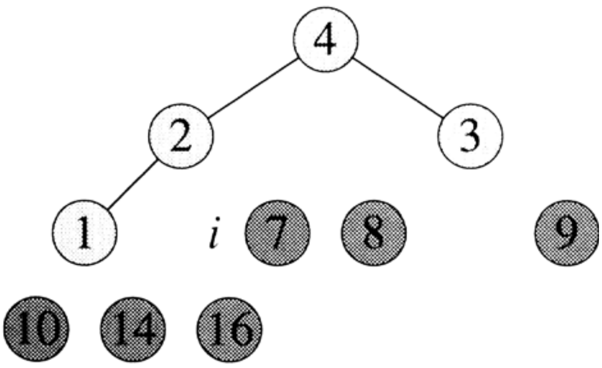
(d)


8
7 3
4 2 1 i 9
10 14 16

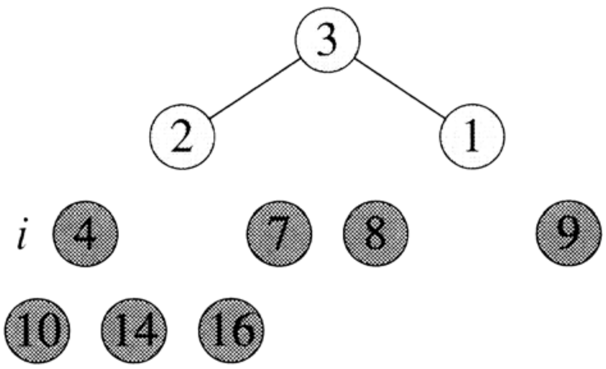
(e)


7
4 3
1 2 8 i 9
10 14 16

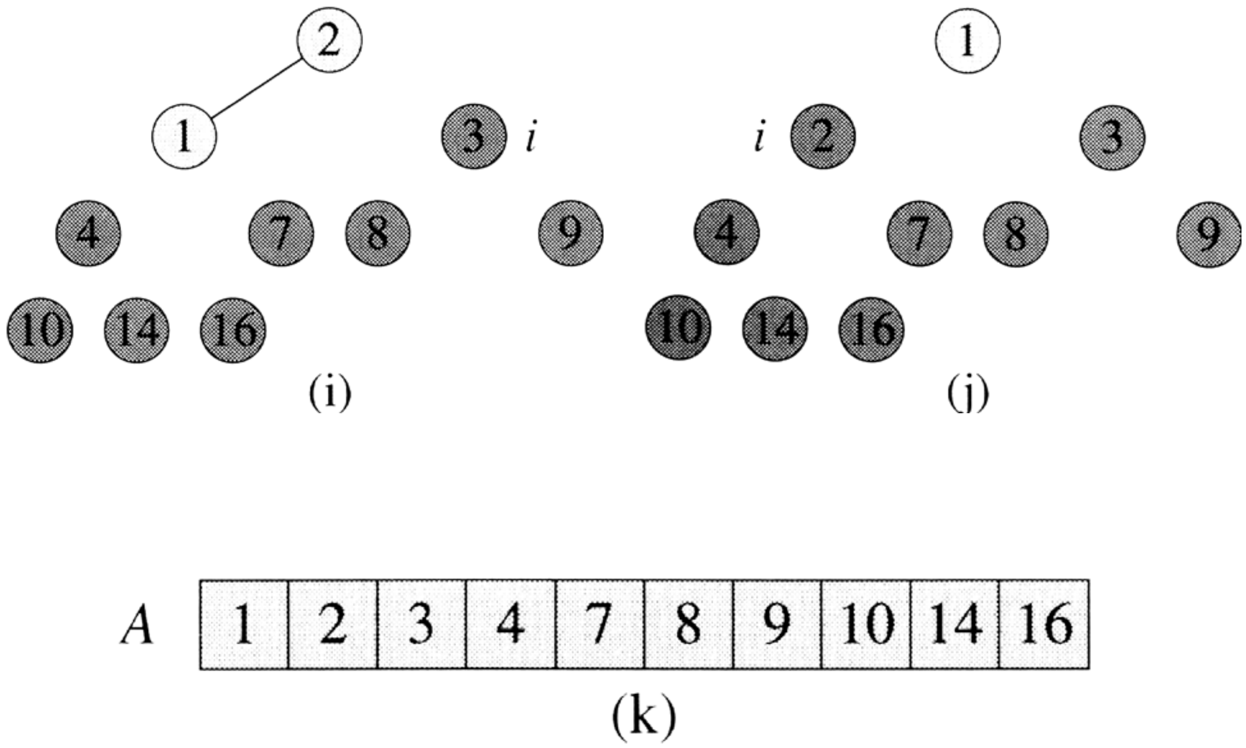
(f)


4
2 3
1 i 7 8 9
10 14 16

(g)


3
2 1
i 4 7 8 9
10 14 16

(h)



6.5 Priority queues

Priority queue*:** a data structure for maintaining a set *A* of elements, each has a value called a ***key. It should support the following operations.

*for convenience, assume "element = key"

Insert(*A*, *x*): *x has a key

Maximum(*A*): (return)

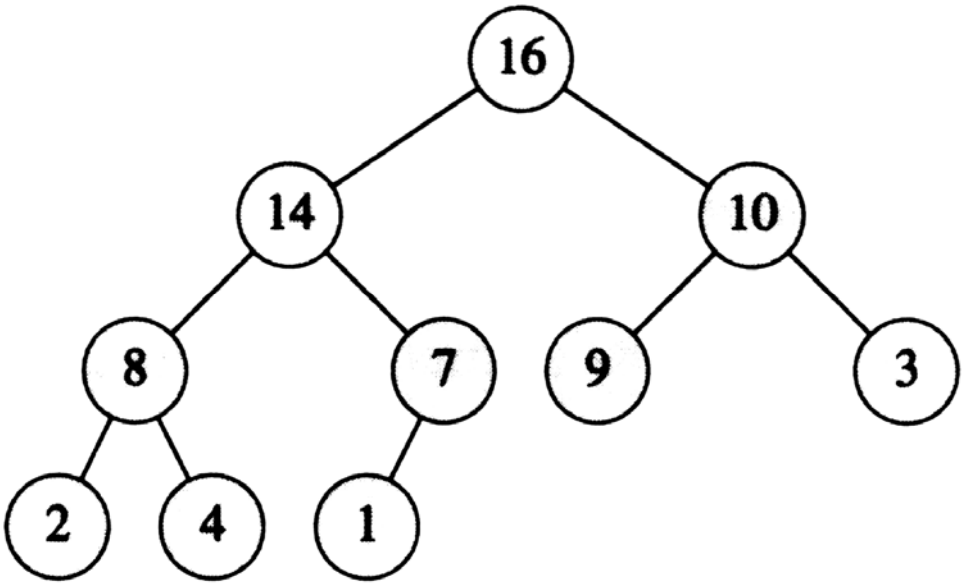
Extract-Max(*A*): (return and remove)

Increase-Key(A, a, k): increase *a*'s key to larger key *k*, where *a* is an element of *A*

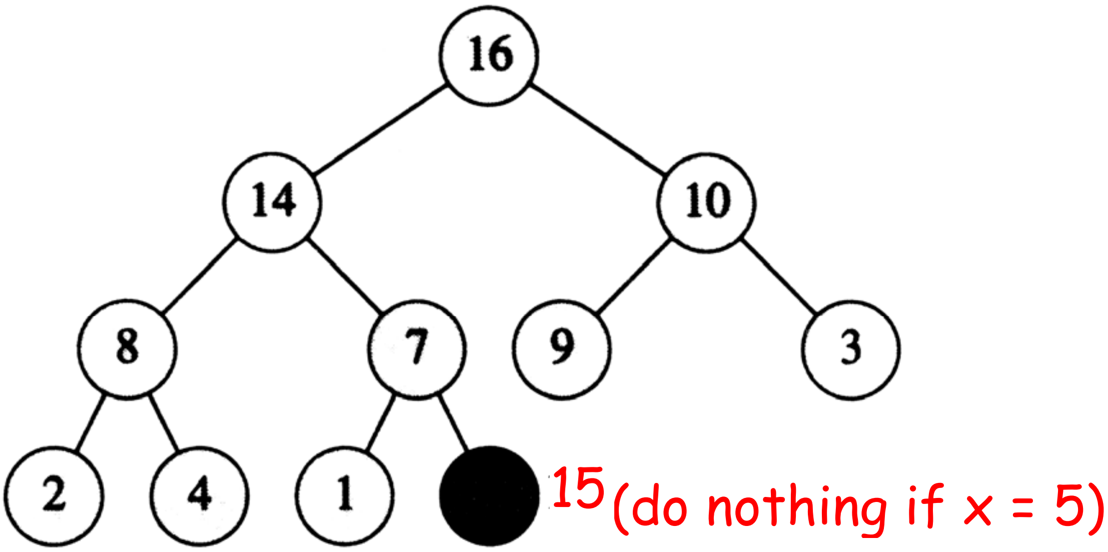
Applications: Job scheduling on a shared computer based upon “priorities.”

Implement a priority queue by a heap:

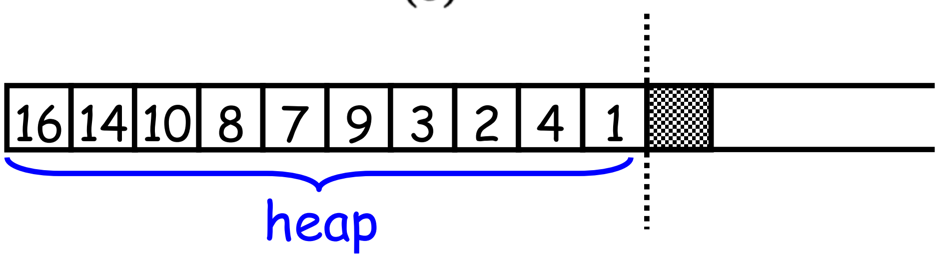
Insert(*A*, *x*): $O(\lg n)$ time

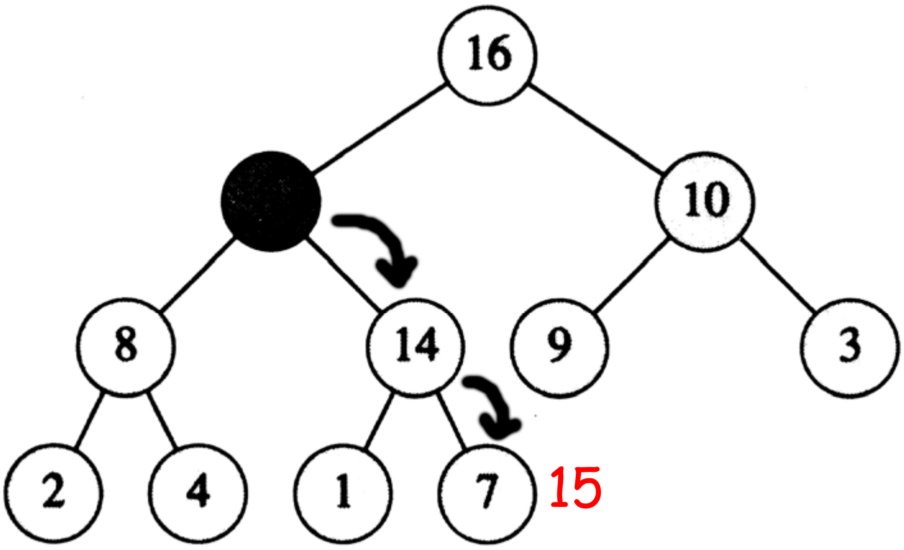


(a) *Insert*(*A*, 15)

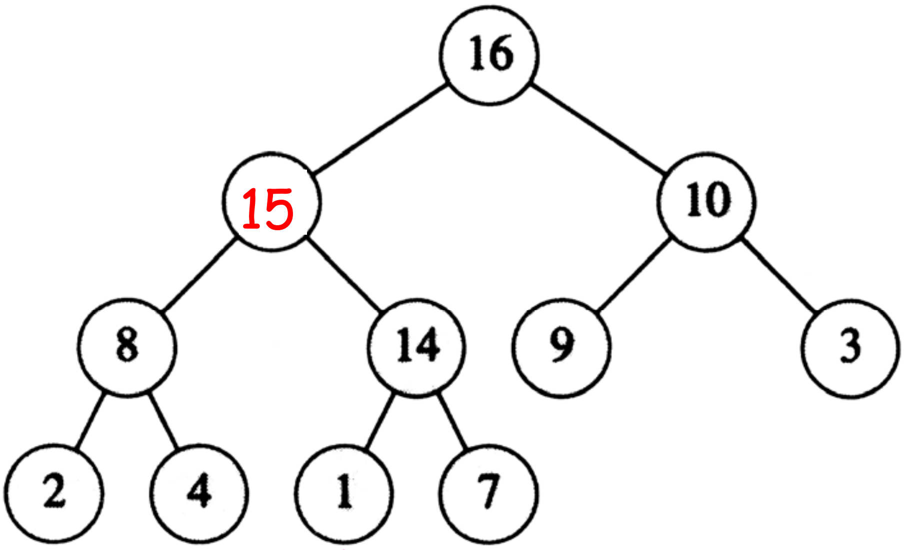


(b)





(c)



(d)

increase to 15.5

Insert(A, x)

```
heap-size(A) ← heap-size(A)+1
i ← heap-size(A)
while i>1 and A[Parent(i)]<x
        do  A(i) ← A[Parent(i)]   往 下 拉
            i ← Parent(i)         向 上 check
A[i] ← x
```

6-11x

Increase-Key(A, i, k): $O(\lg n)$ time
(similar to *Insert*)

Maximum(*A*): $O(1)$ time

Extract-Max(*A*): $O(\lg n)$ time

Step 1: Exchange $A[1]$ and $A[\textit{heap-size}]$ n = 10
Step 2: $\textit{heap-size} \leftarrow \textit{heap-size} - 1$ n = 9
Step 3: *Heapify*(*A*, 1)
Step 4: return $A[\textit{heap-size} + 1]$ n+1 = 10

Homework: Ex. 6.2-5, 6.5-9, Prob. 6-2, 6-3

* array implementation 6-11y

* handles: pointers to the objects in a data structure 6-11z