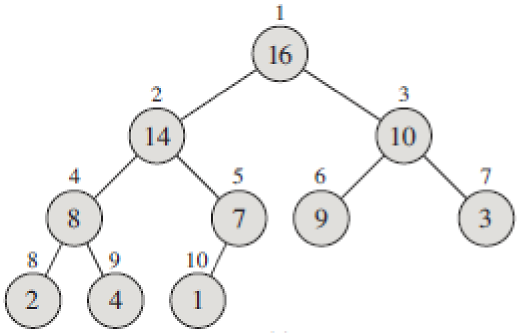
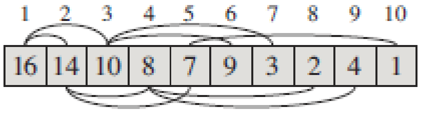
MAX-HEAP



(a)

Array representation of MAX-HEAP



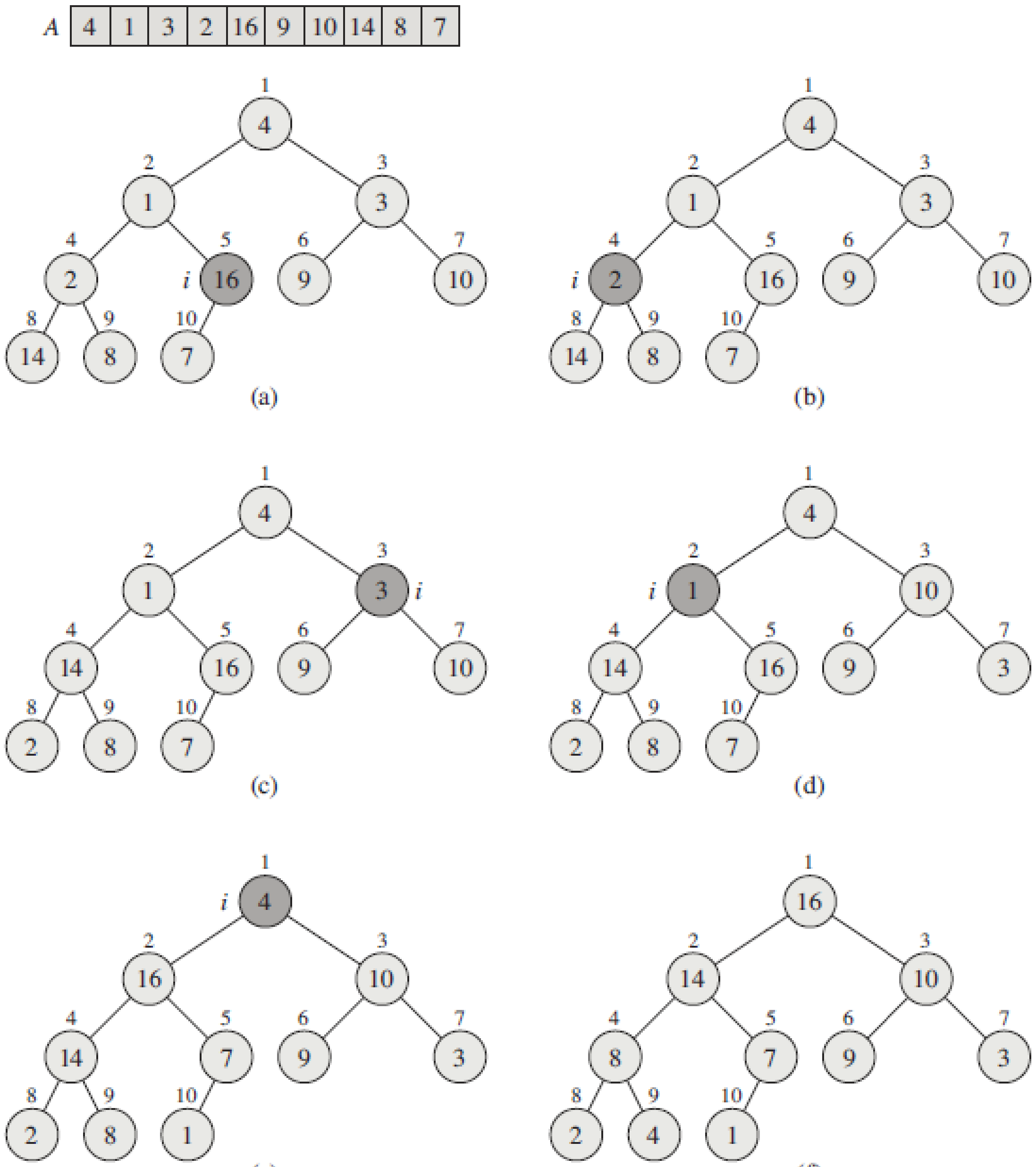
(b)

```
PARENT($i$)    //index of parent
    return $\lfloor i/2 \rfloor$

LEFT($i$)    // index of left child
    return $2i$

RIGHT($i$)    //index of right child
    return $2i + 1$

BUILD-MAX-HEAP($A$)    // builds the heap
    $A.heap\text{-}size = A.length$
    for $i = \lfloor A.length/2 \rfloor$ downto 1
        MAX-HEAPIFY($A, i$)    // places an element with index i at proper position in heap.
```

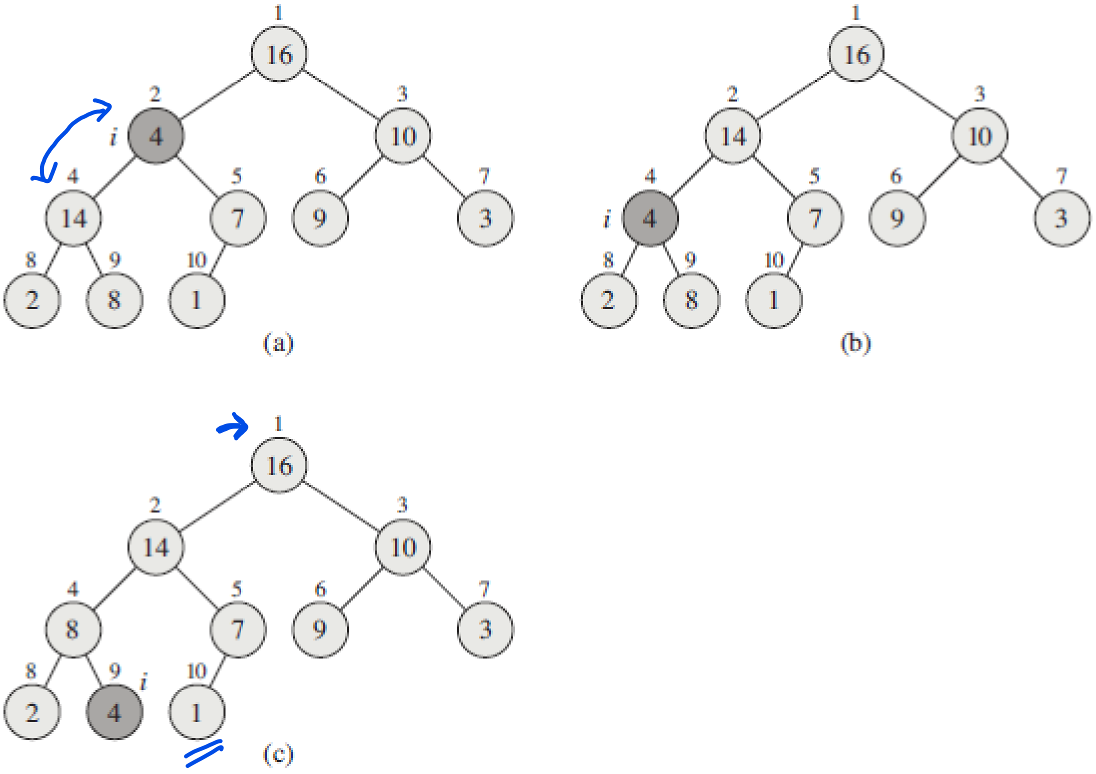


A									
4	1	3	2	16	9	10	14	8	7

(a) (b) (c) (d) (e) (f)

MAX-HEAP

```
MAX-HEAPIFY(A, i)                                  i (A, 2)        heapsize = 10
    l = LEFT(i)                                      l = 4
    r = RIGHT(i)                                     r = 5
    if l ≤ A.heap-size and A[l] > A[i]                  14 > 4
        largest = l                                    largest = 4
    else largest = i
    if r ≤ A.heap-size and A[r] > A[largest]           7 > 14 ×
        largest = r
    if largest ≠ i
        exchange A[i] with A[largest]
        MAX-HEAPIFY(A, largest)                   (A, 4)
```



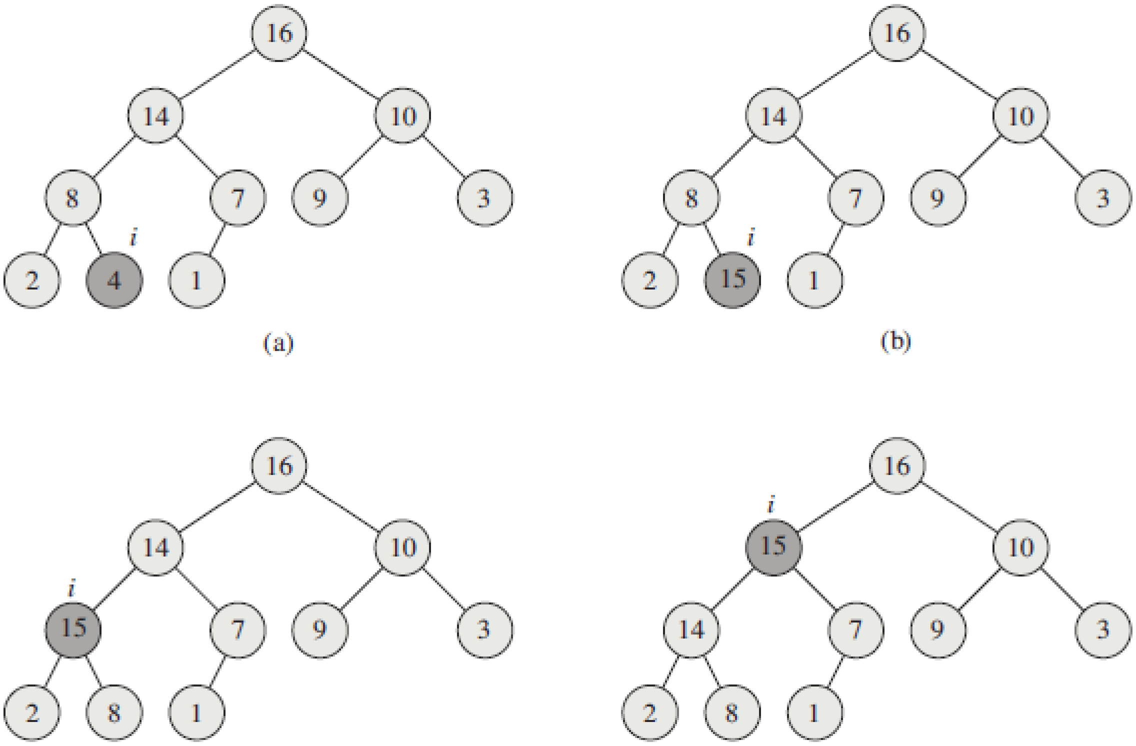
(a)

(b)

(c)

```
HEAP-EXTRACT-MAX(A)     // Returns the maximum element
                                 from the heap.
    if A.heap-size < 1
        error "heap underflow"
    max = A[1]
    A[1] = A[A.heap-size]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
    return max
```

```
HEAP-INCREASE-KEY$(A, i, key)$
if $key < A[i]$
    error "new key is smaller than current key"
$A[i] = key$
while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
    exchange $A[i]$ with $A[\text{PARENT}(i)]$
    $i = \text{PARENT}(i)$
```



(a)

(b)

(c)

(d)