Chapter 9

Lecture - 8
Heaps

Objectives

Upon completion you will be able to:

- **Define and implement heap structures**
- **Design and implement selection applications using a heap**
- **Design and implement priority queues using a heap**

9-1 Basic Concepts

A heap is a binary tree whose left and right subtrees have values less than their parents. We begin with a discussion of the basic heap structure and its two primary operations, reheap up and reheap down.

- **Definition**
- **Maintenance Operations**

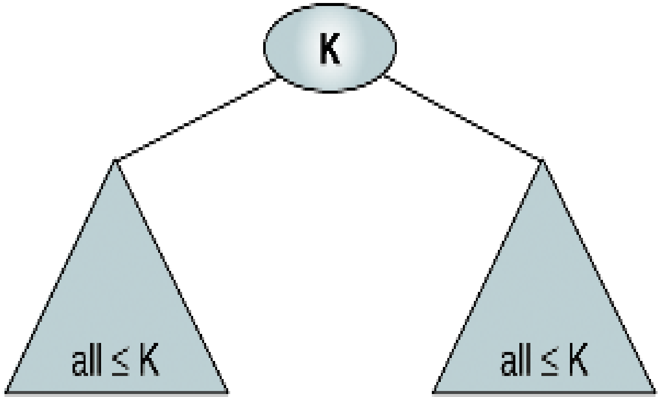


FIGURE 9-1 Heap

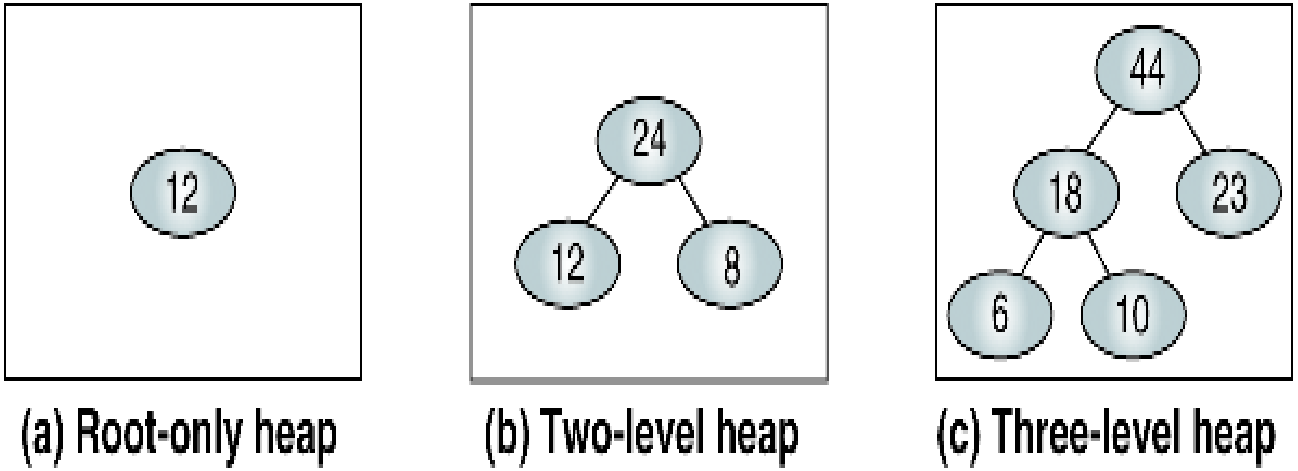
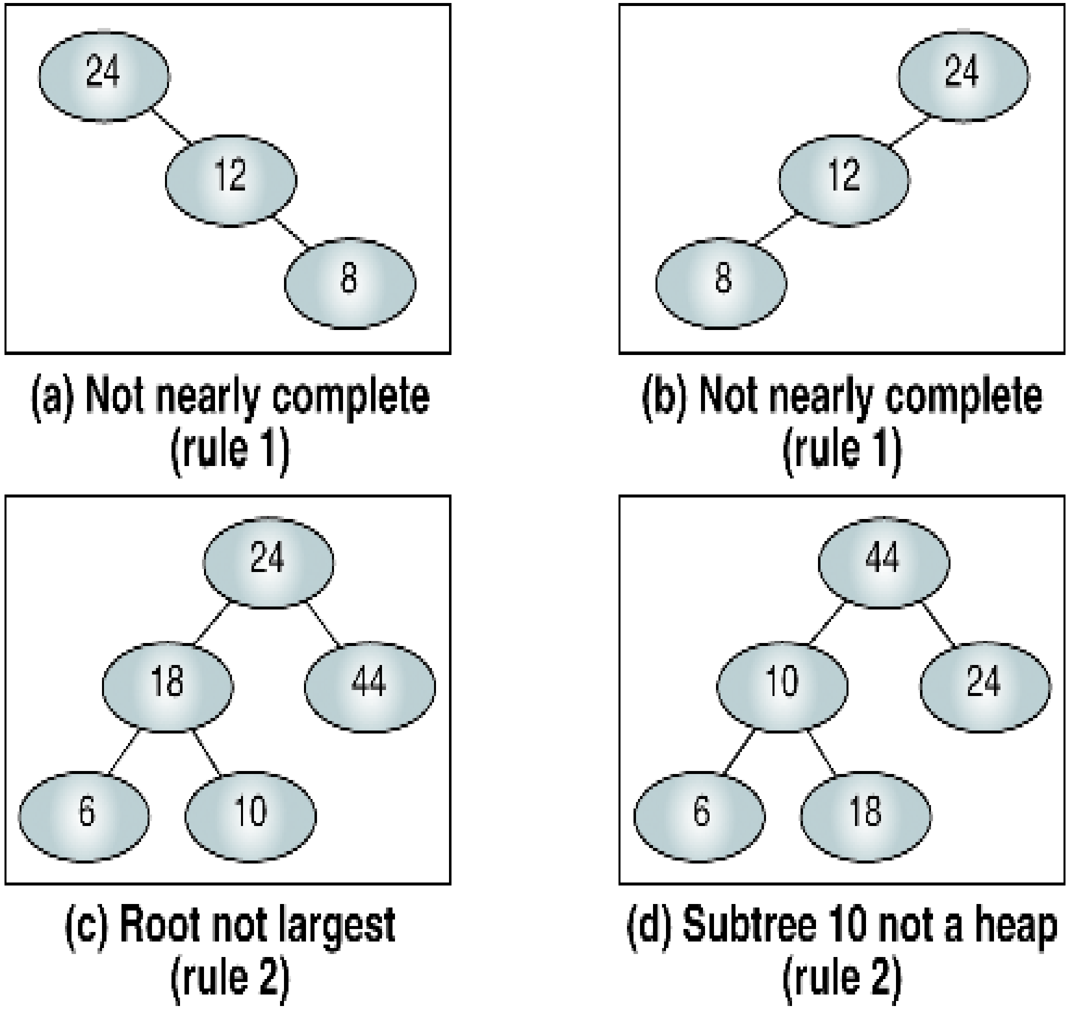


FIGURE 9-2 Heap Trees



(a) Not nearly complete (rule 1)

(b) Not nearly complete (rule 1)

(c) Root not largest (rule 2)

(d) Subtree 10 not a heap (rule 2)

FIGURE 9-3 Invalid Heaps

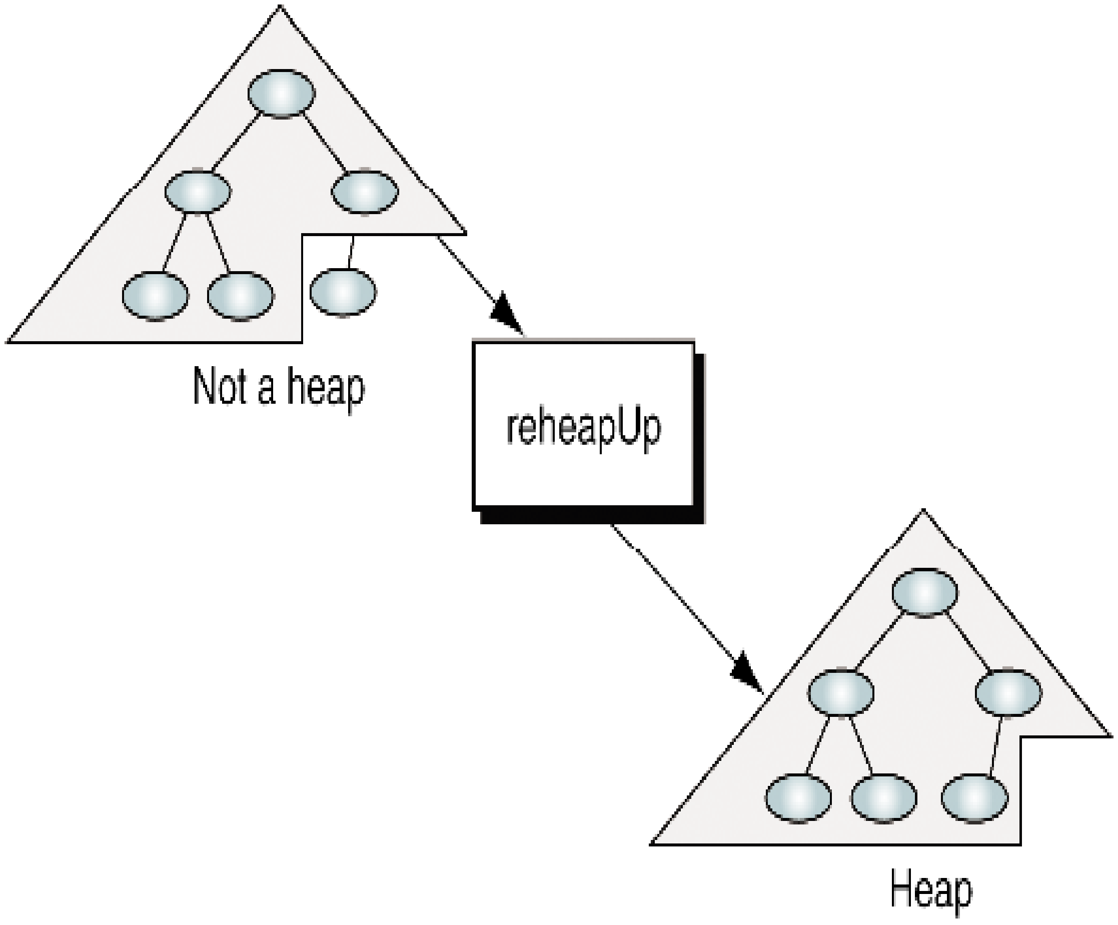
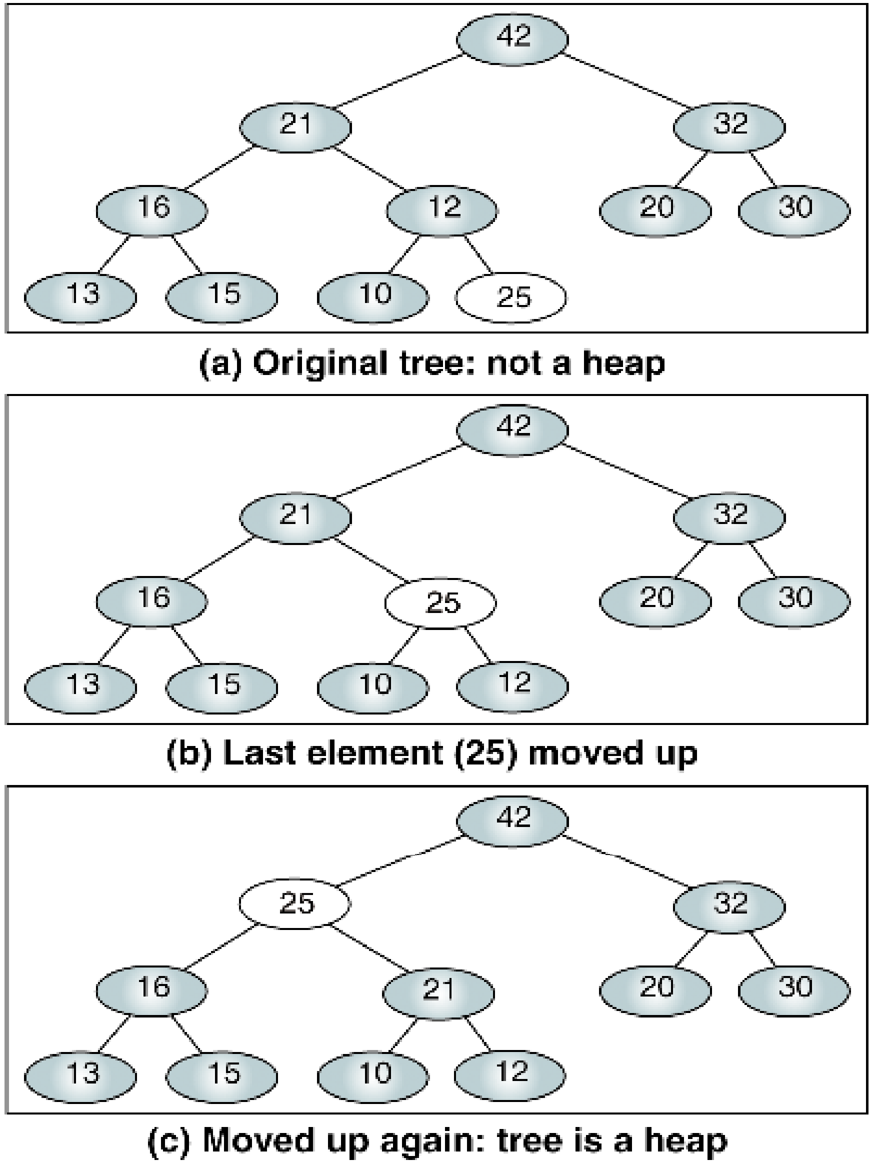


FIGURE 9-4 Reheap Up Operation



(a) Original tree: not a heap

(b) Last element (25) moved up

(c) Moved up again: tree is a heap

FIGURE 9-5 Reheap Up Example

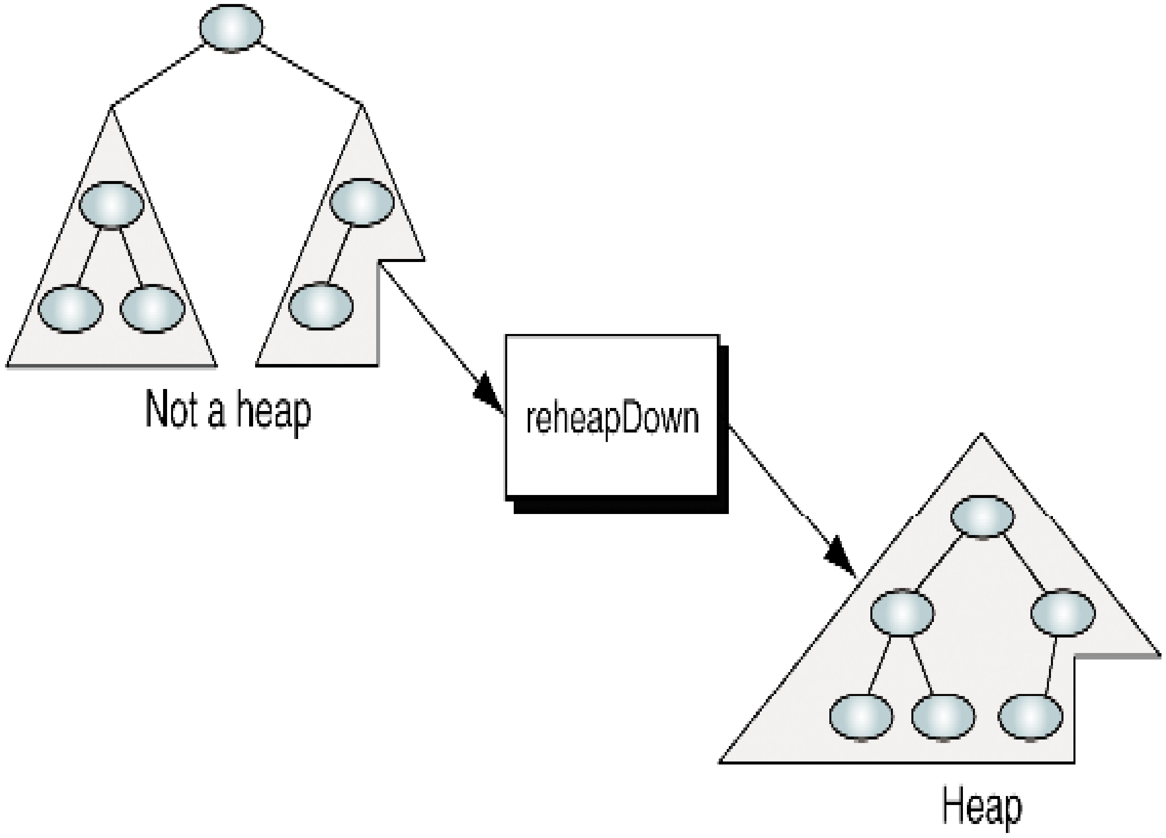
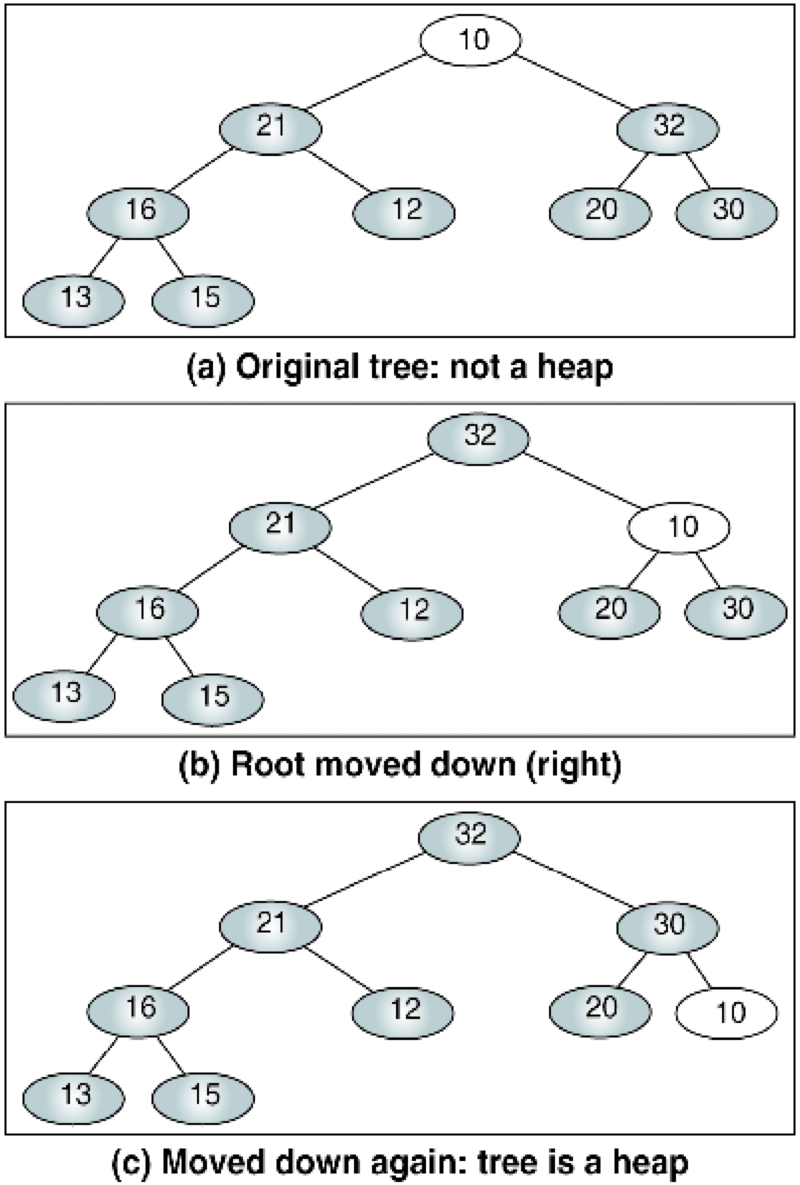


FIGURE 9-6 Reheap Down Operation



(a) Original tree: not a heap

(b) Root moved down (right)

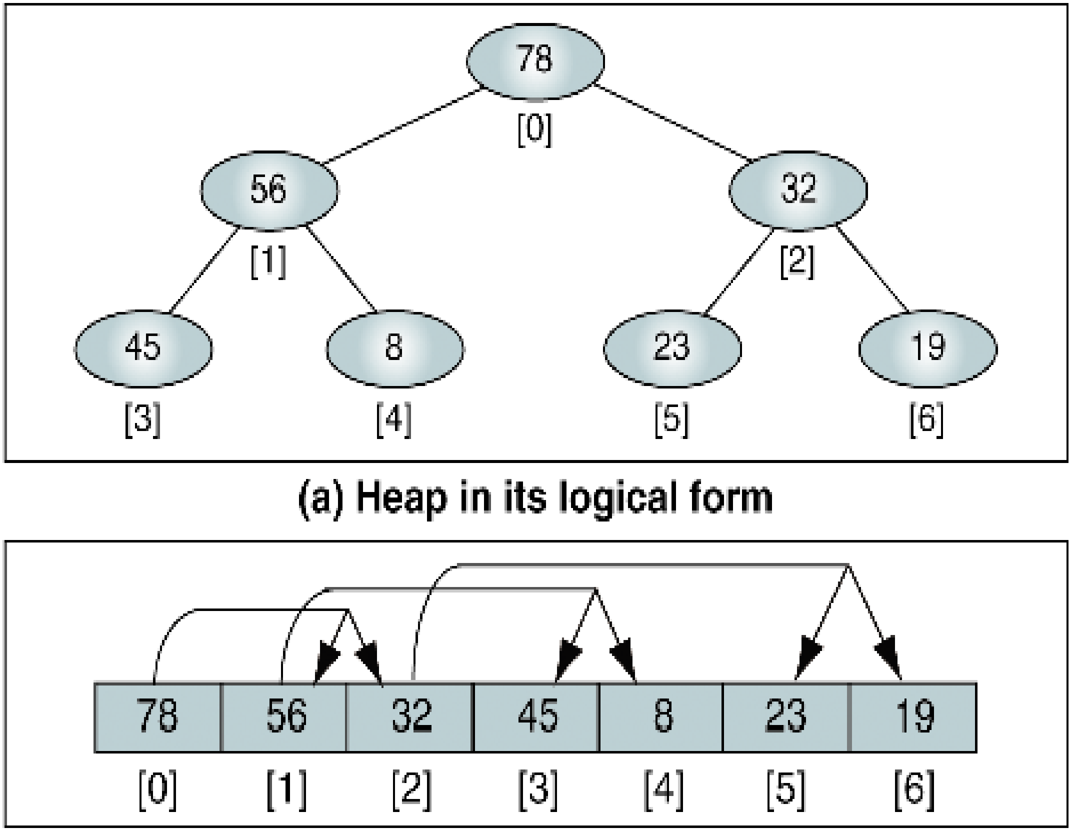
(c) Moved down again: tree is a heap

FIGURE 9-7 Reheap Down Example

9-2 Heap Implementation

Heaps are usually implemented in an array structure. In this section we discuss and develop five heap algorithms.

- **Reheap Up**
- **Reheap Down**
- **Build a Heap**
- **Insert a Node into a Heap**
- **Delete a Node from a Heap**

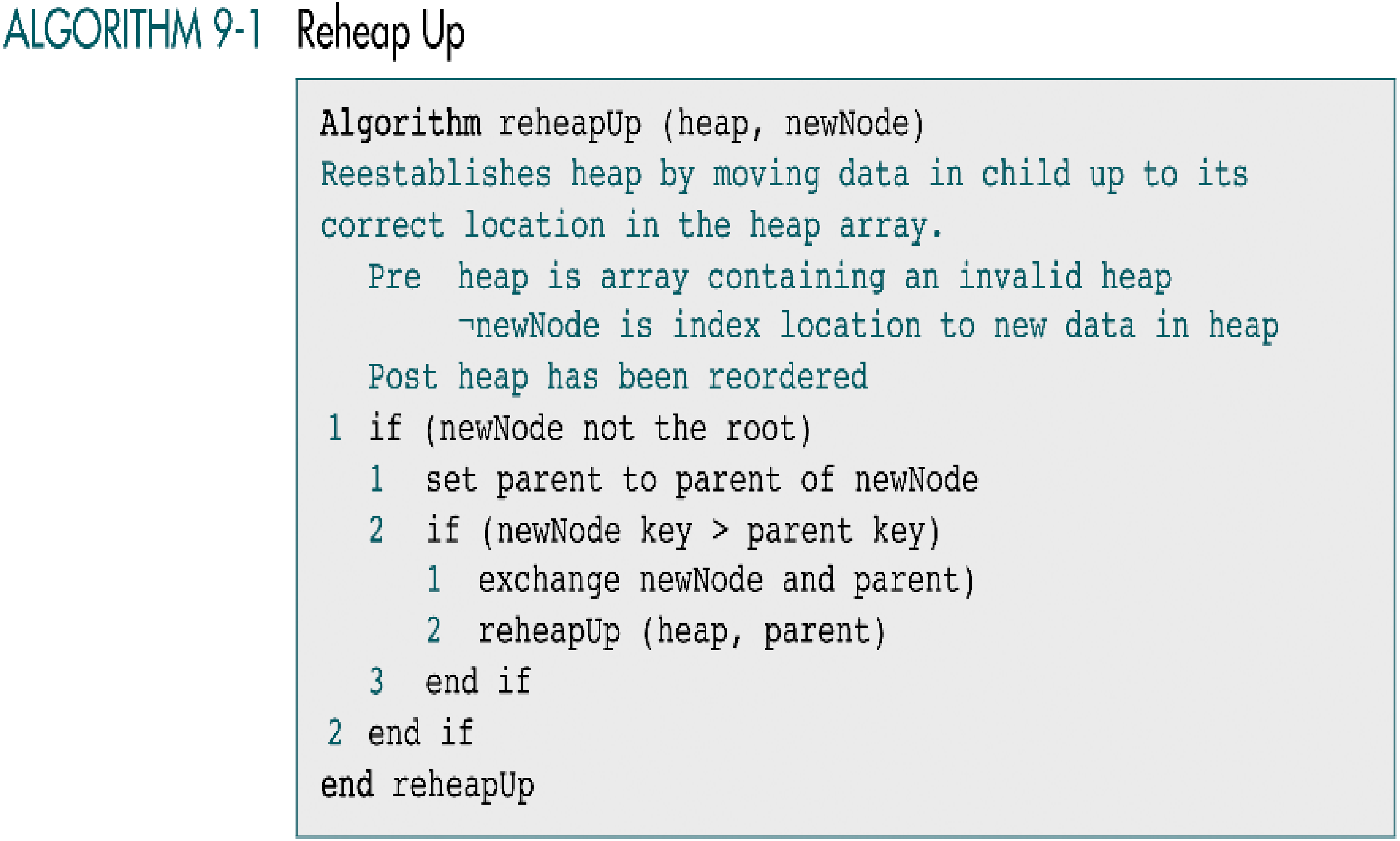


(a) Heap in its logical form

78	56	32	45	8	23	19
[0]	[1]	[2]	[3]	[4]	[5]	[6]

(b) Heap in an array

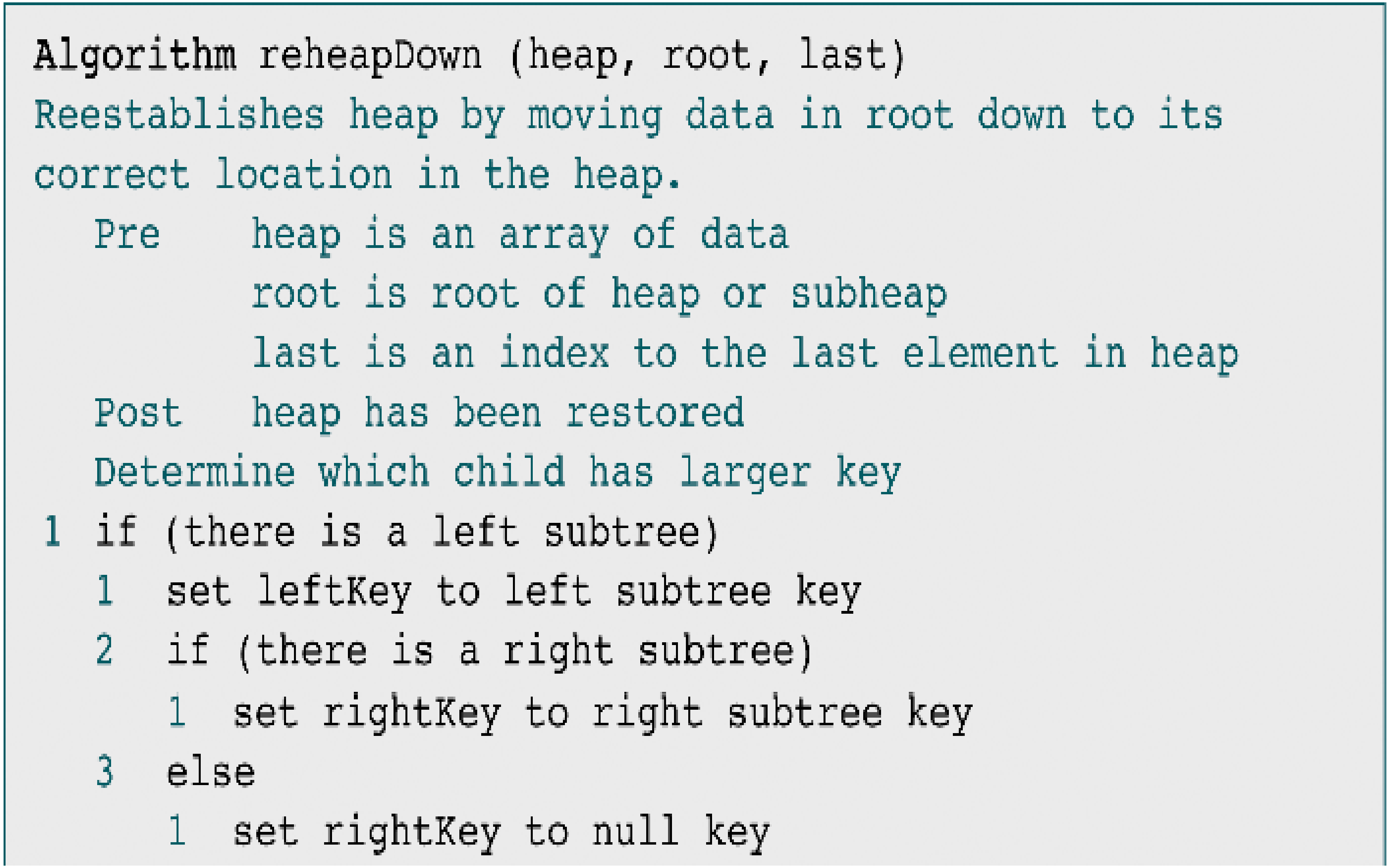
FIGURE 9-8 Heaps in Arrays



ALGORITHM 9-1 Reheap Up

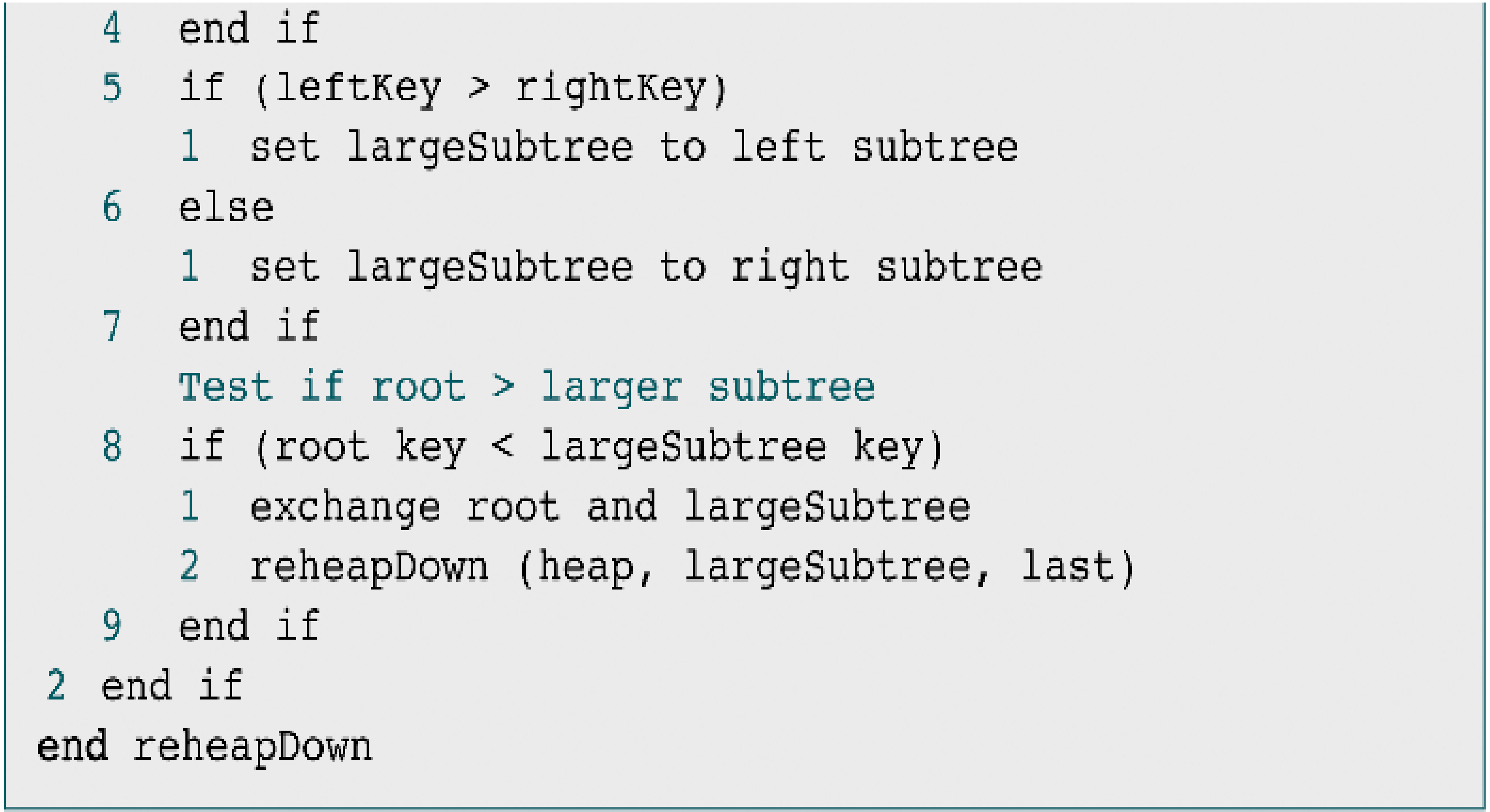
```
Algorithm reheapUp (heap, newNode)
Reestablishes heap by moving data in child up to its
correct location in the heap array.
   Pre  heap is array containing an invalid heap
        ¬newNode is index location to new data in heap
   Post heap has been reordered
1 if (newNode not the root)
   1  set parent to parent of newNode
   2  if (newNode key > parent key)
      1  exchange newNode and parent)
      2  reheapUp (heap, parent)
   3  end if
2 end if
end reheapUp
```

ALGORITHM 9-2 Reheap Down



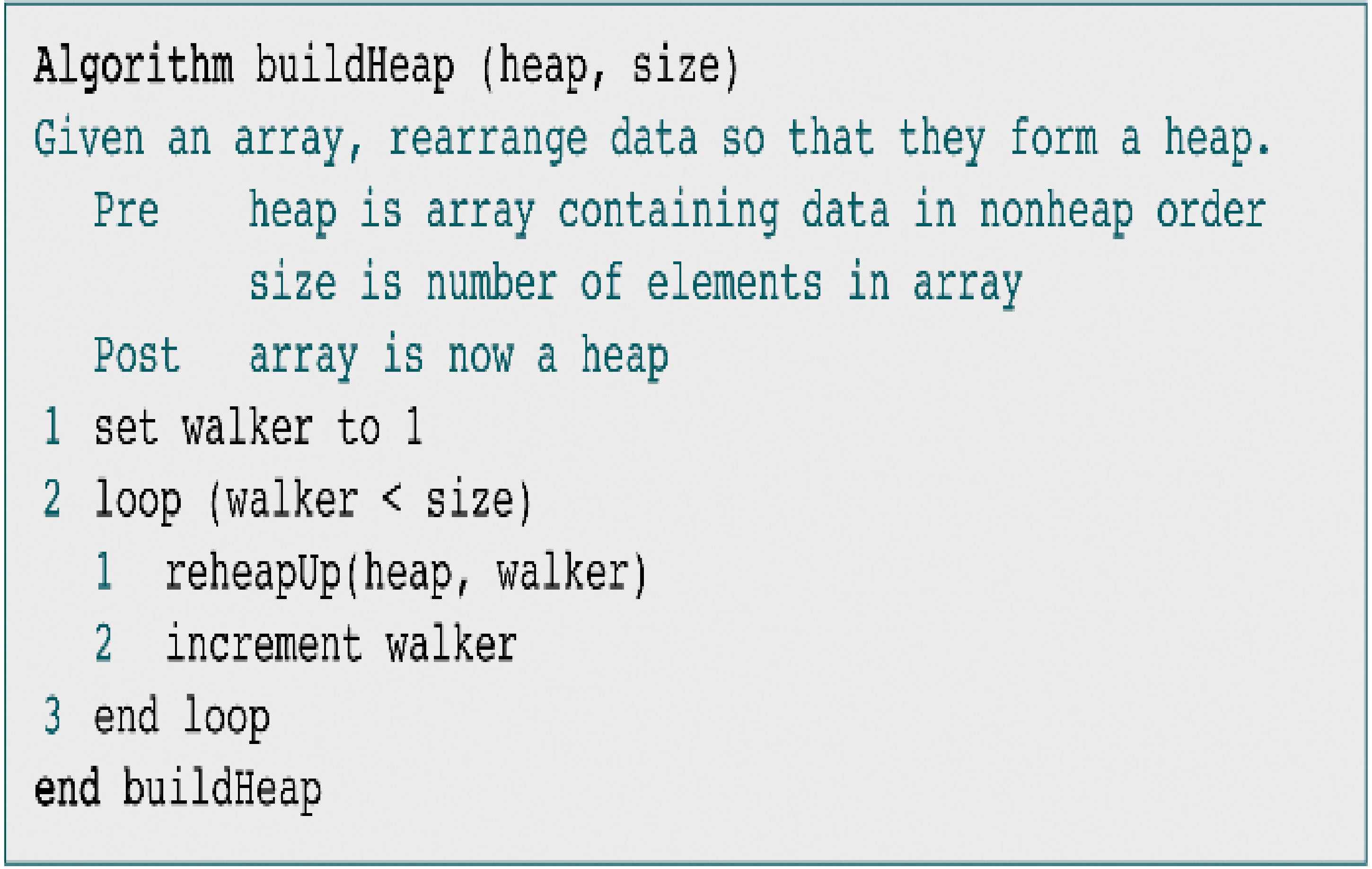
```
Algorithm reheapDown (heap, root, last)
Reestablishes heap by moving data in root down to its
correct location in the heap.
   Pre    heap is an array of data
          root is root of heap or subheap
          last is an index to the last element in heap
   Post   heap has been restored
   Determine which child has larger key
1  if (there is a left subtree)
   1  set leftKey to left subtree key
   2  if (there is a right subtree)
      1  set rightKey to right subtree key
   3  else
      1  set rightKey to null key
```

ALGORITHM 9-2 Reheap Down (continued)



```
      4  end if
      5  if (leftKey > rightKey)
         1  set largeSubtree to left subtree
      6  else
         1  set largeSubtree to right subtree
      7  end if
         Test if root > larger subtree
      8  if (root key < largeSubtree key)
         1  exchange root and largeSubtree
         2  reheapDown (heap, largeSubtree, last)
      9  end if
   2  end if
end reheapDown
```

ALGORITHM 9-3 Build Heap



```
Algorithm buildHeap (heap, size)
Given an array, rearrange data so that they form a heap.
   Pre    heap is array containing data in nonheap order
          size is number of elements in array
   Post   array is now a heap
1  set walker to 1
2  loop (walker < size)
   1  reheapUp(heap, walker)
   2  increment walker
3  end loop
end buildHeap
```

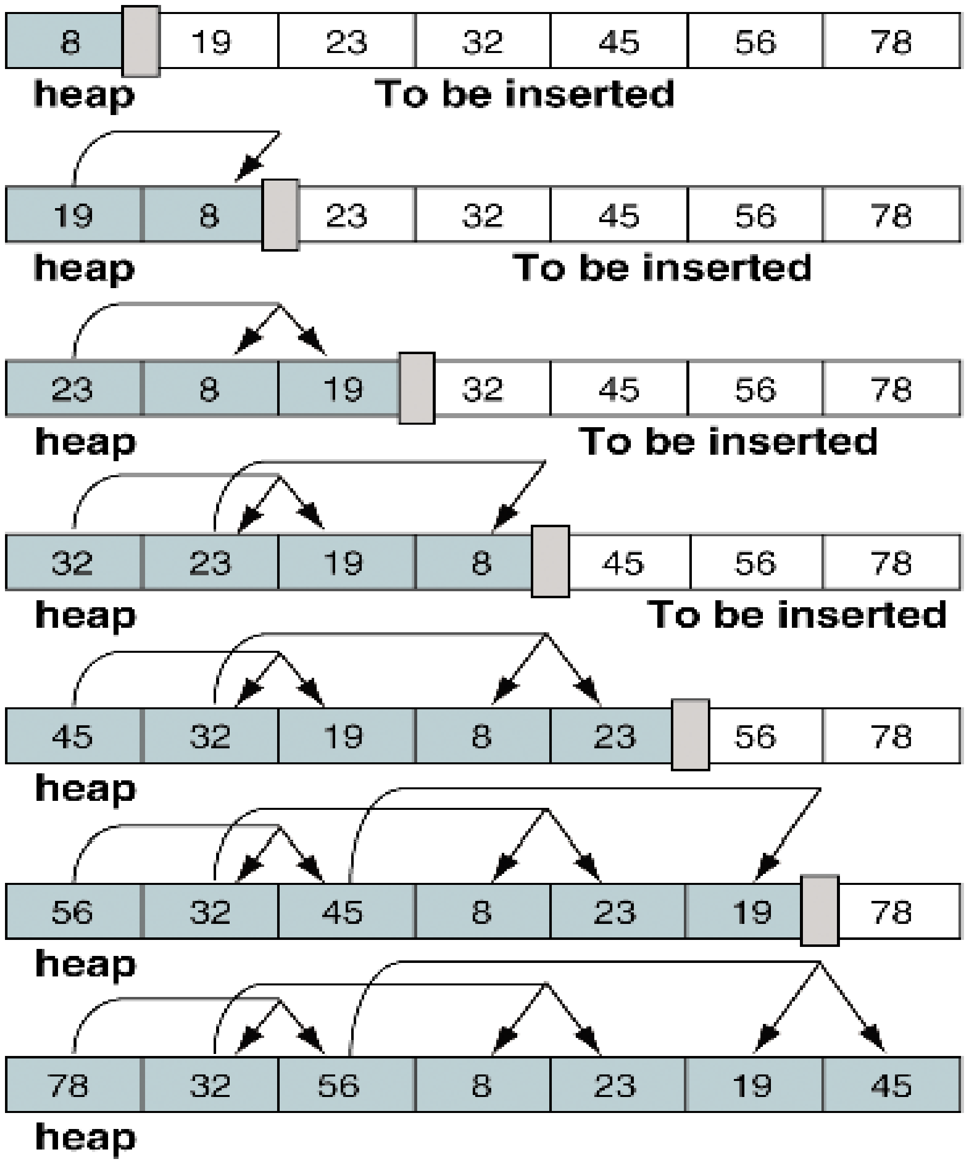
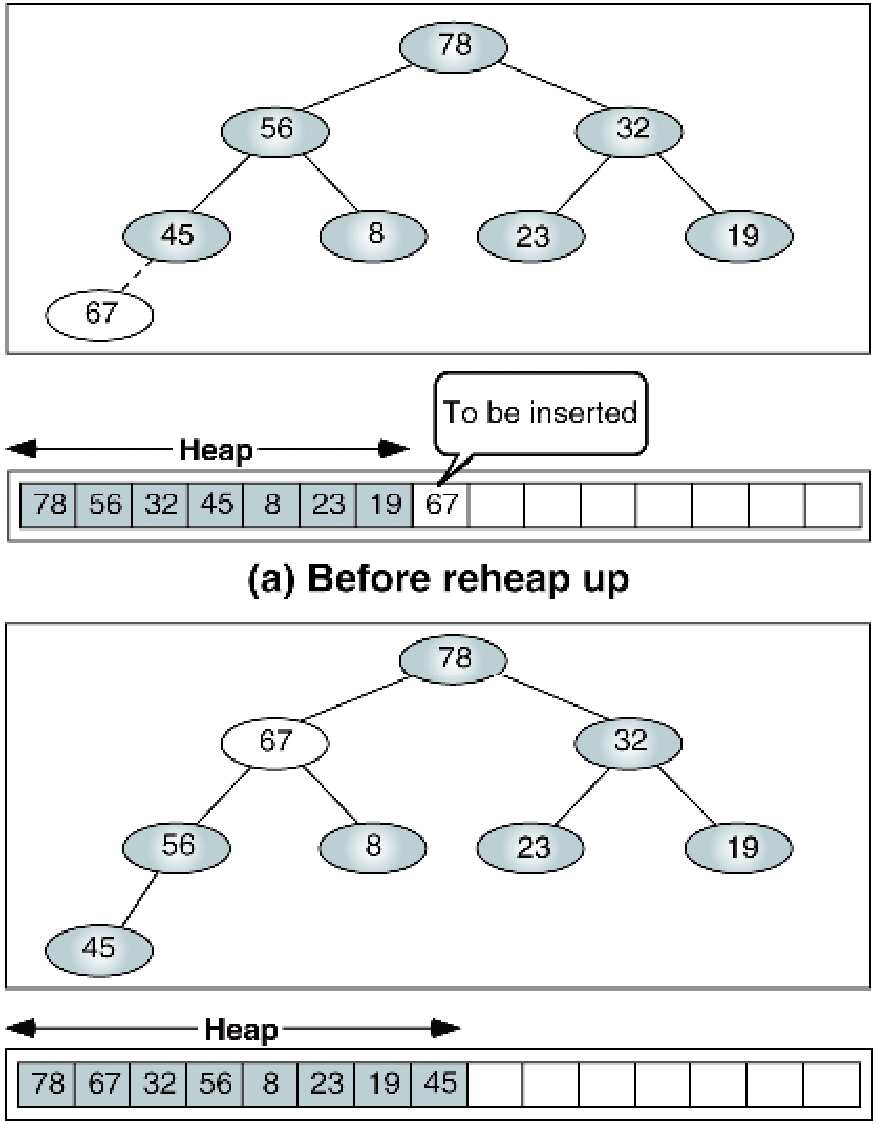


FIGURE 9-9 Building a Heap

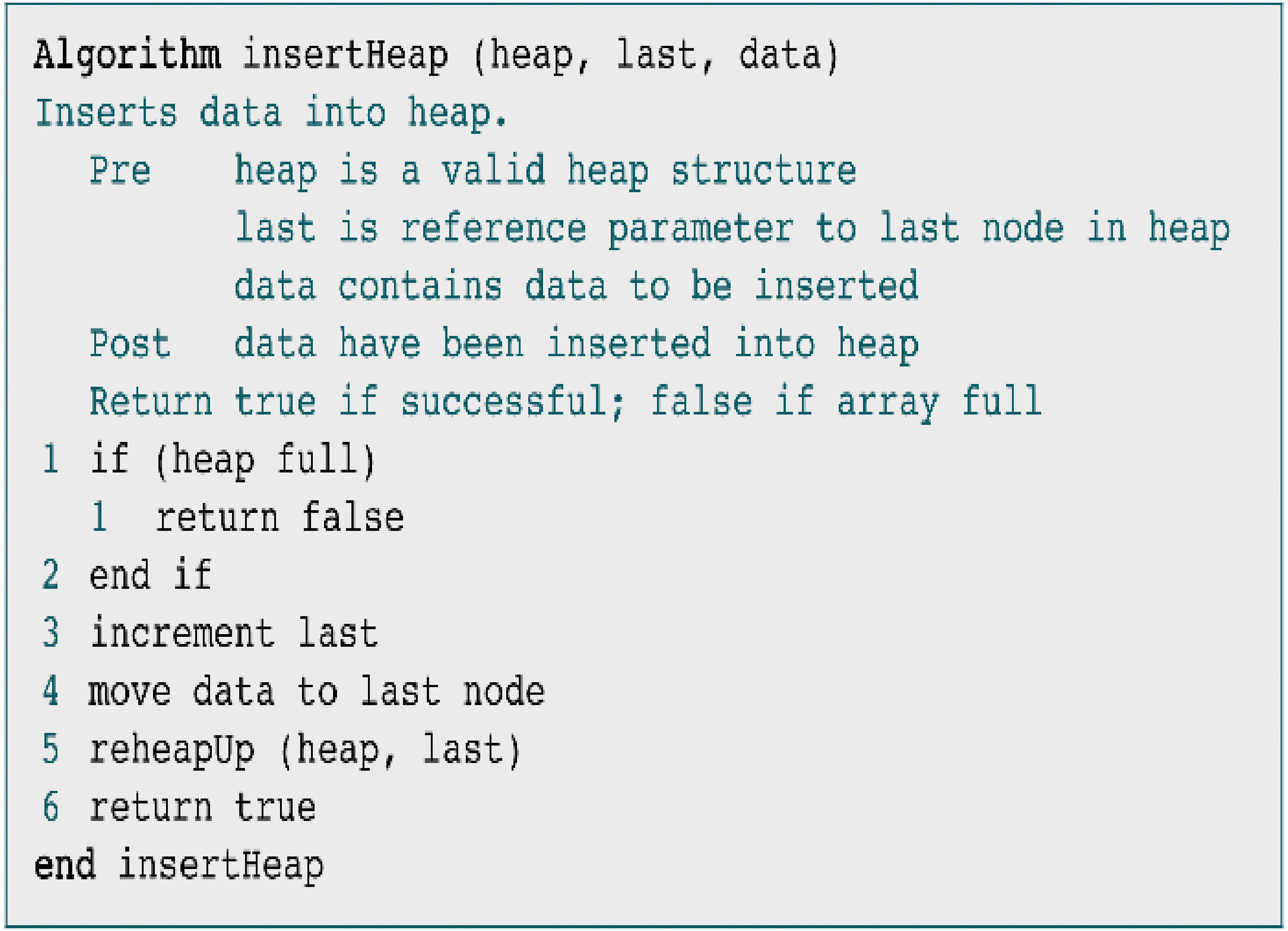


(a) Before reheap up

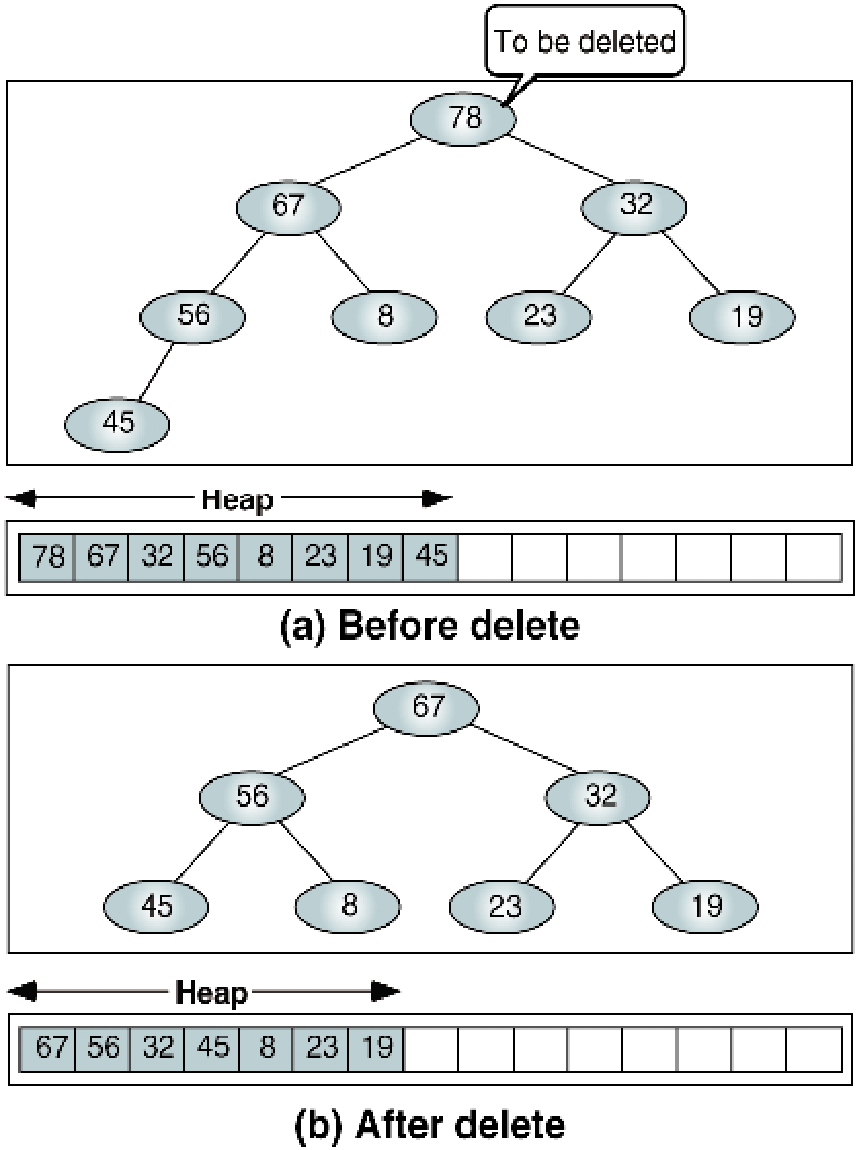
(b) After reheap up

FIGURE 9-10 Insert Node

ALGORITHM 9-4 Insert Heap



```
Algorithm insertHeap (heap, last, data)
Inserts data into heap.
   Pre    heap is a valid heap structure
          last is reference parameter to last node in heap
          data contains data to be inserted
   Post   data have been inserted into heap
   Return true if successful; false if array full
   if (heap full)
      return false
   end if
   increment last
   move data to last node
   reheapUp (heap, last)
   return true
end insertHeap
```

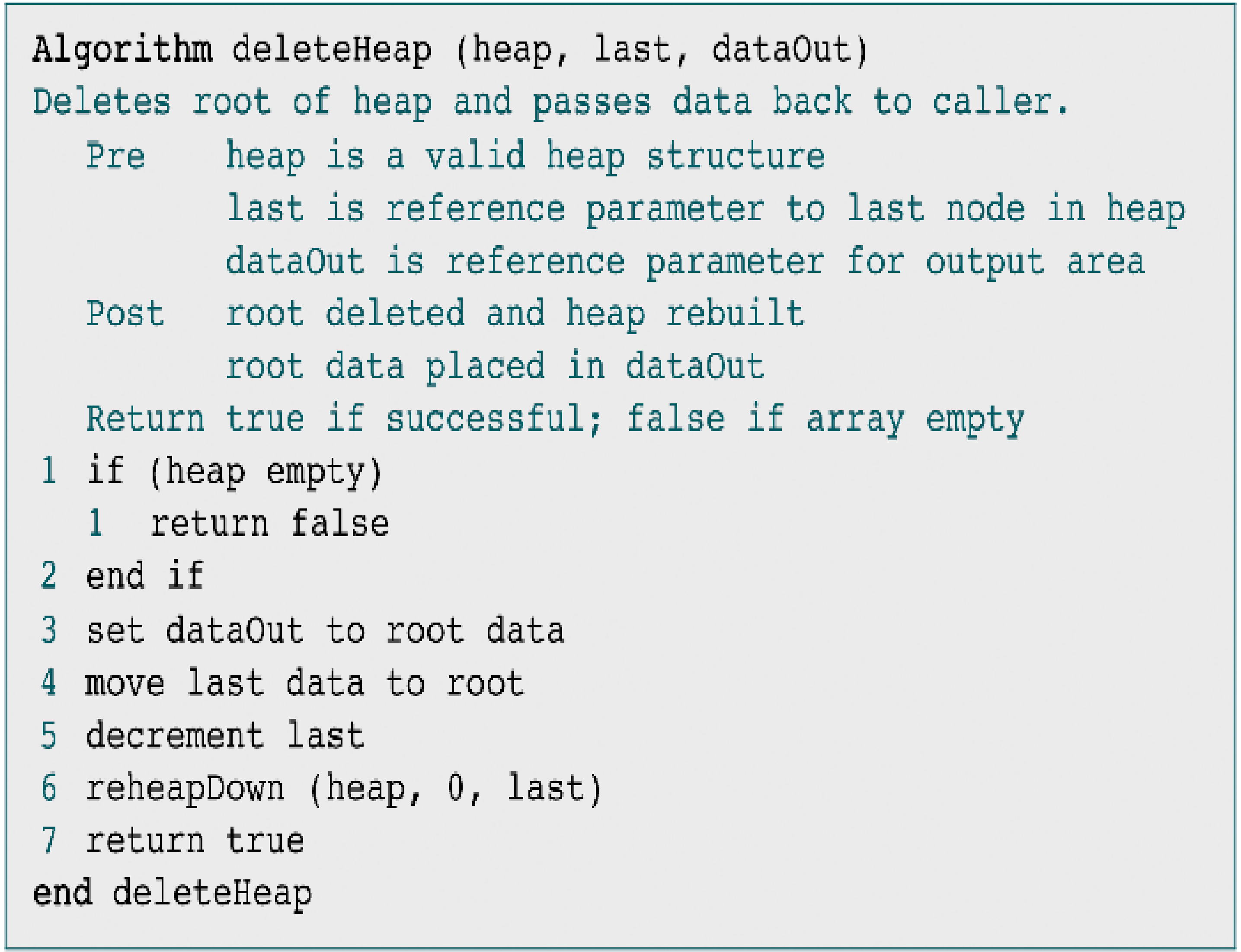


(a) Before delete

(b) After delete

FIGURE 9-11 `deleteHeap` Node

ALGORITHM 9-5 Delete Heap Node

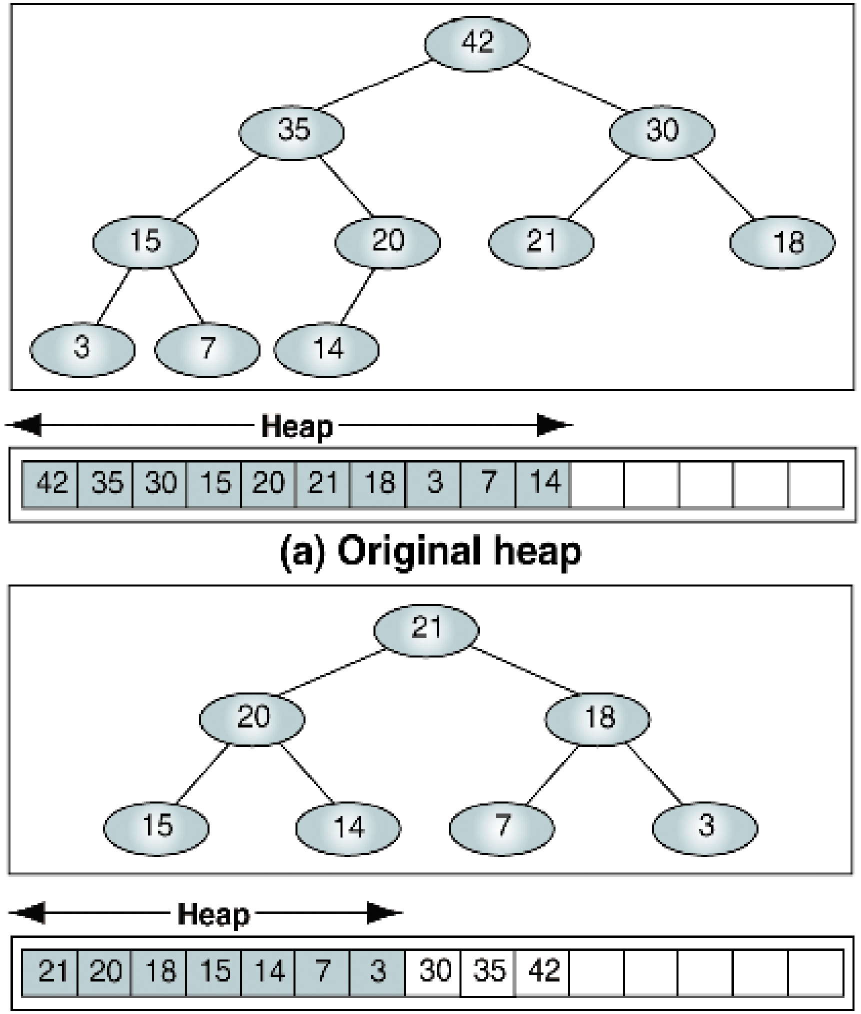


```
Algorithm deleteHeap (heap, last, dataOut)
Deletes root of heap and passes data back to caller.
   Pre    heap is a valid heap structure
          last is reference parameter to last node in heap
          dataOut is reference parameter for output area
   Post   root deleted and heap rebuilt
          root data placed in dataOut
   Return true if successful; false if array empty
   if (heap empty)
      return false
   end if
   set dataOut to root data
   move last data to root
   decrement last
   reheapDown (heap, 0, last)
   return true
end deleteHeap
```

9-4 Heap Applications

We discuss two of the three common heap applications-selection and priority queues. For selection applications, we develop a high-level algorithm. For priorityh queues, we develop the C code.

- **Selection Algorithms**
- **Priority Queues**



42	35	30	15	20	21	18	3	7	14					

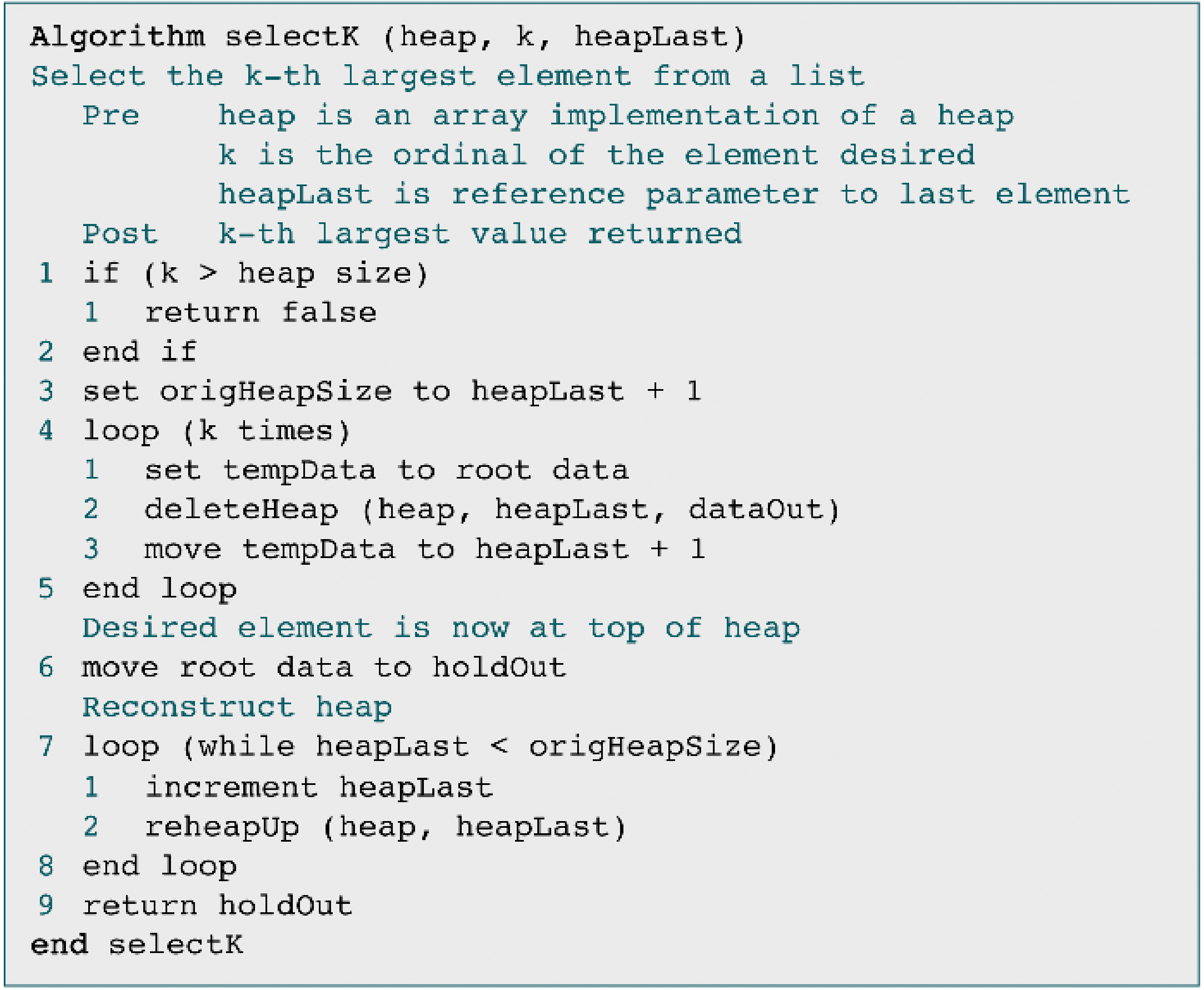
(a) Original heap

21	20	18	15	14	7	3	30	35	42					

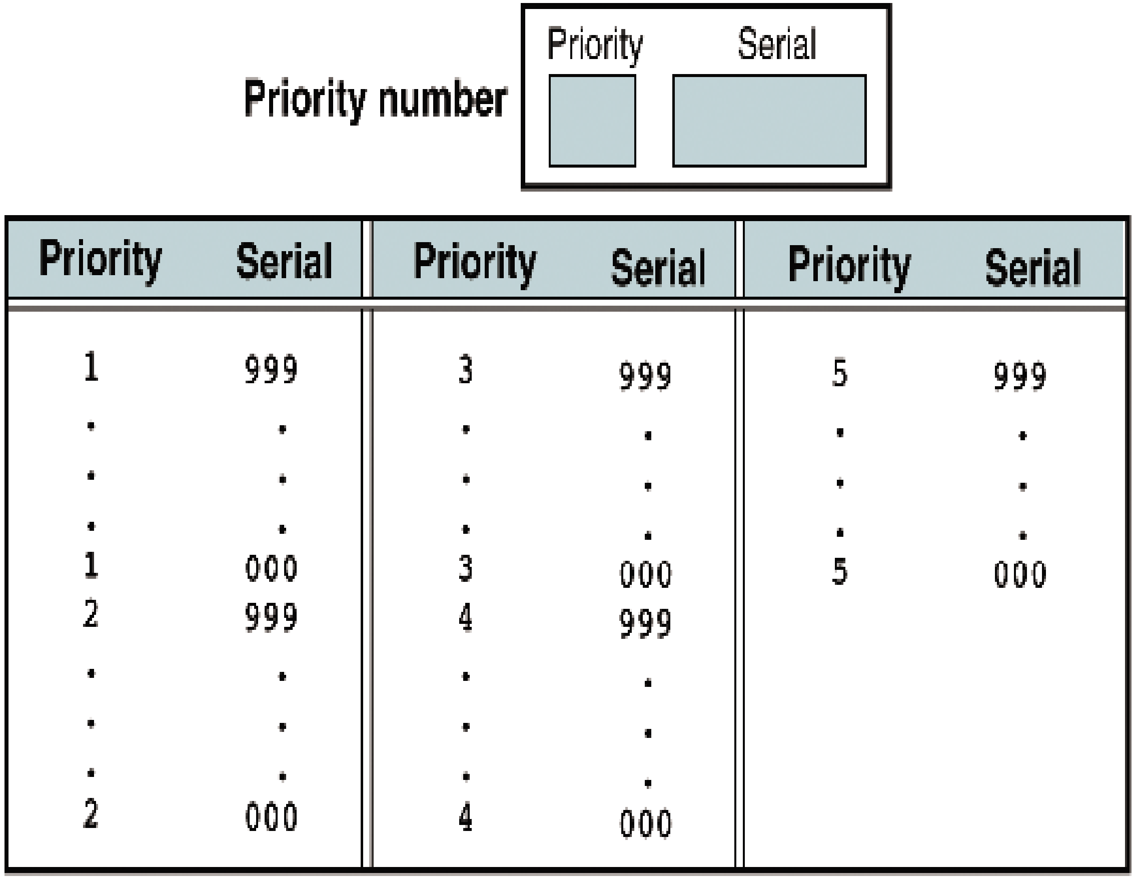
(b) After three deletions

FIGURE 9-14 Heap Selection

ALGORITHM 9-6 Heap Selection

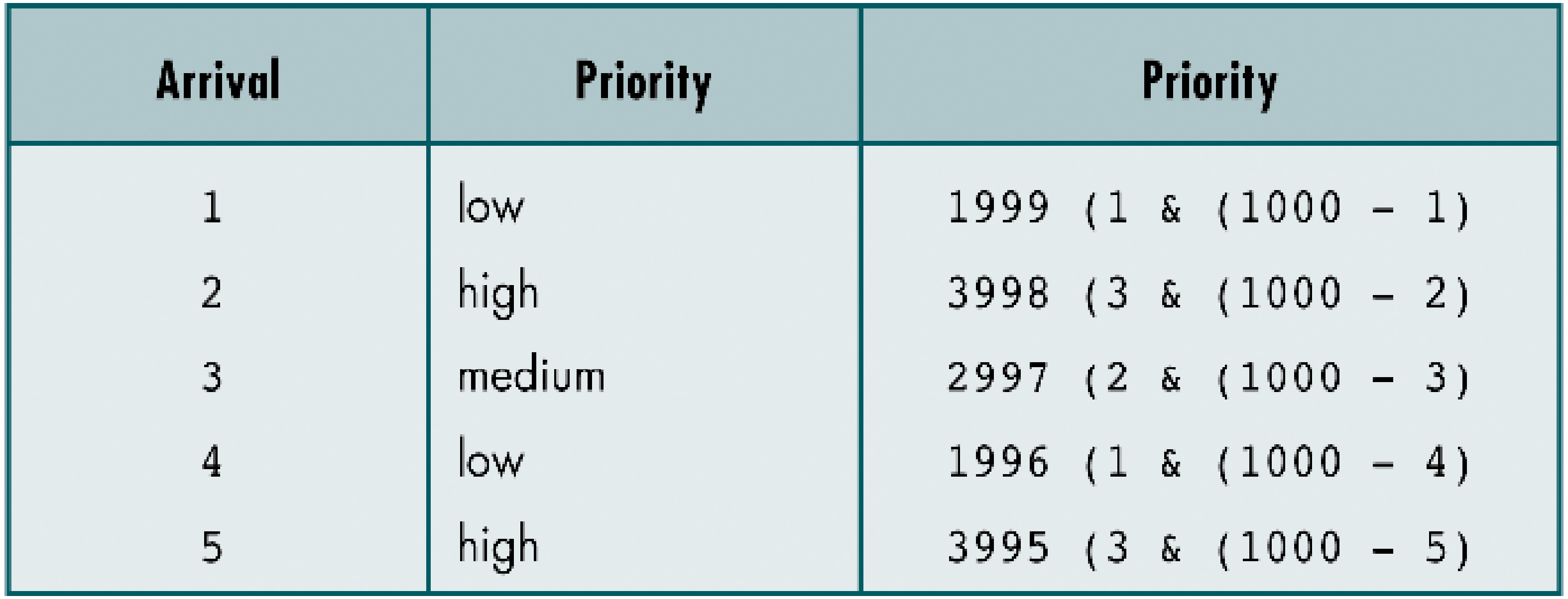


```
Algorithm selectK (heap, k, heapLast)
Select the k-th largest element from a list
   Pre     heap is an array implementation of a heap
           k is the ordinal of the element desired
           heapLast is reference parameter to last element
   Post    k-th largest value returned
   if (k > heap size)
      return false
   end if
   set origHeapSize to heapLast + 1
   loop (k times)
      set tempData to root data
      deleteHeap (heap, heapLast, dataOut)
      move tempData to heapLast + 1
   end loop
   Desired element is now at top of heap
   move root data to holdOut
   Reconstruct heap
   loop (while heapLast < origHeapSize)
      increment heapLast
      reheapUp (heap, heapLast)
   end loop
   return holdOut
end selectK
```



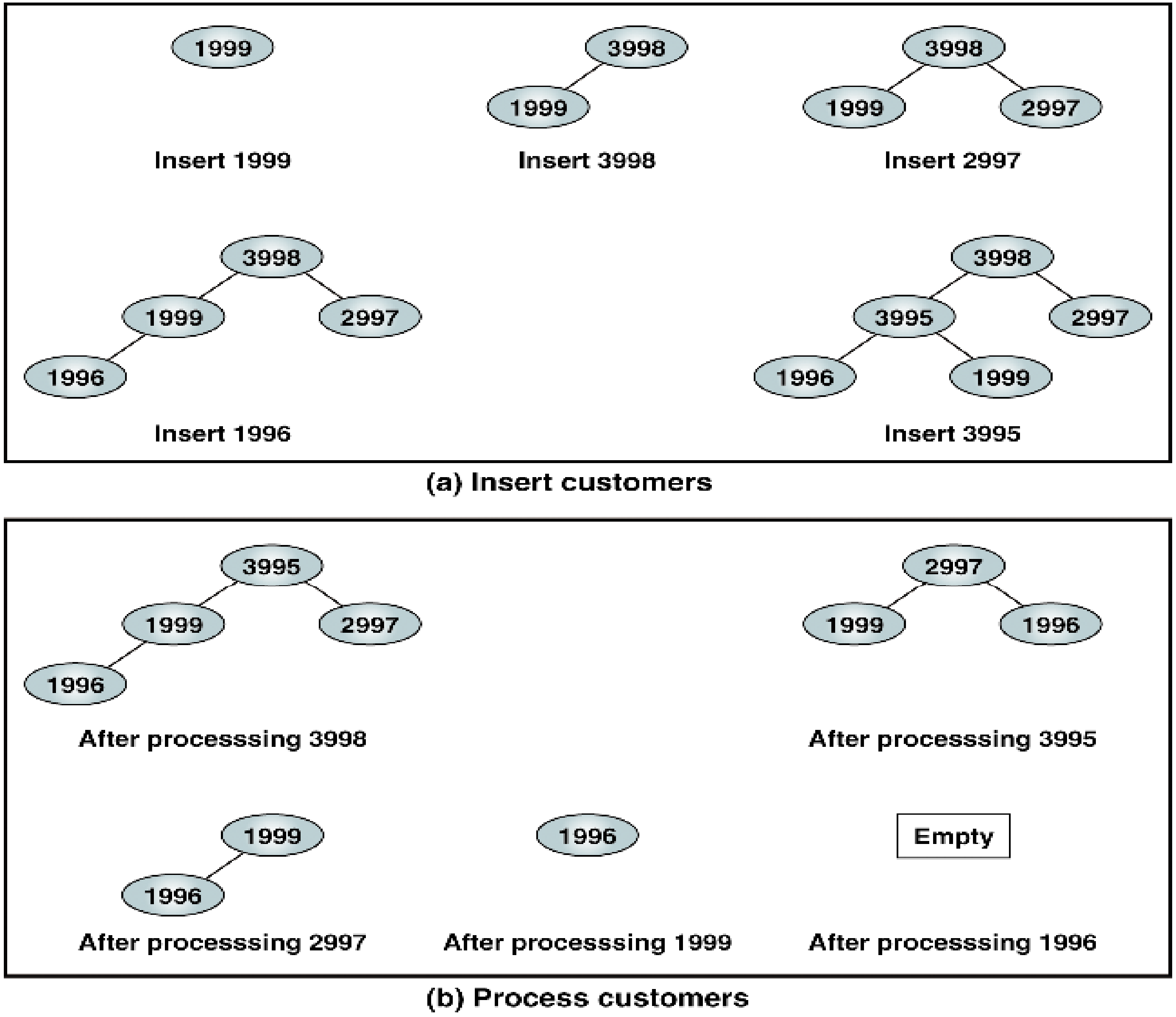
Priority	Serial	Priority	Serial	Priority	Serial
1	999	3	999	5	999
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
1	000	3	000	5	000
2	999	4	999		
.	.	.	.		
.	.	.	.		
.	.	.	.		
2	000	4	000		

FIGURE 9-15 Priority Queue Priority Numbers



Arrival	Priority	Priority
1	low	1999 (1 & (1000 – 1)
2	high	3998 (3 & (1000 – 2)
3	medium	2997 (2 & (1000 – 3)
4	low	1996 (1 & (1000 – 4)
5	high	3995 (3 & (1000 – 5)

TABLE 9-1 Priority Number Assignments



(a) Insert customers

(b) Process customers

FIGURE 9-16 Priority Queue Example