CST 311
Algorithm Analysis & Design

Al Lake
Oregon Institute of Technology

Chapter 6
Heapsort

Heapsort

- The heapsort data structure is an array object that can be viewed as a nearly complete binary tree.
- Each node of the tree corresponds to an element of the array that stores the value in the node.
- The tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point.
- An array A that represents a heap is an object with 2 attributes:
  - length[A], the number of elements in the array
  - heap-size[A], number of element in the heap

Max-heap

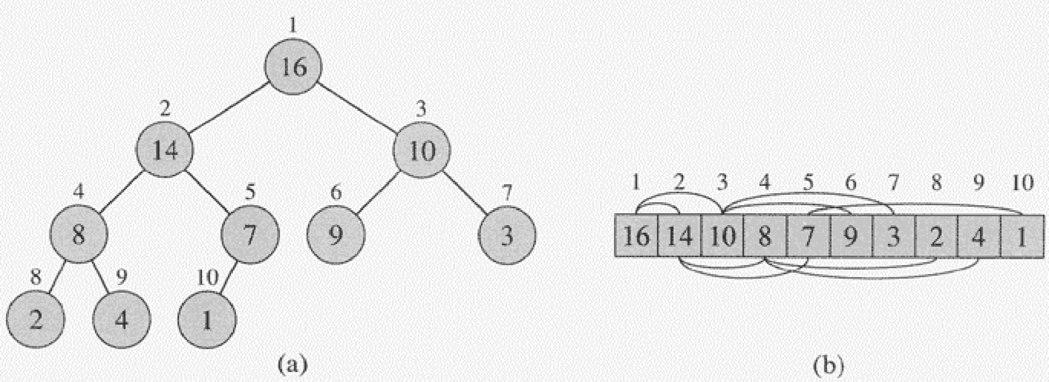

Figure 6.1 A max-heap viewed as **(a)** a binary tree and **(b)** an array. The number within the circle at each node in the tree is the value stored at that node. The number above a node is the corresponding index in the array. Above and below the array are lines showing parent-child relationships; parents are always to the left of their children. The tree has height three; the node at index 4 (with value 8) has height one.

Heap properties

- **There are 2 kinds of binary heaps:**
  - **max-heaps**
  - **min-heaps**
- **Max-heap: every node i other than the root:**
  - **A[Parent(i)] ≥ A[i]**
  - **Thus the largest element in a max-heap is stored at the root, and the subtree rootted at a noed contains values no larger than that contained at the node itself.**
- **Min-heap: every node i other than the root:**
  - **A[Parent(i)] ≥ A[i]**

Maintaining a heap

```
MAX-HEAPIFY(A, i)
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 ≠ i
   then exchange A[i] ↔ A[largest]
        MAX-HEAPIFY(A, largest)
```

Maintaining a heap

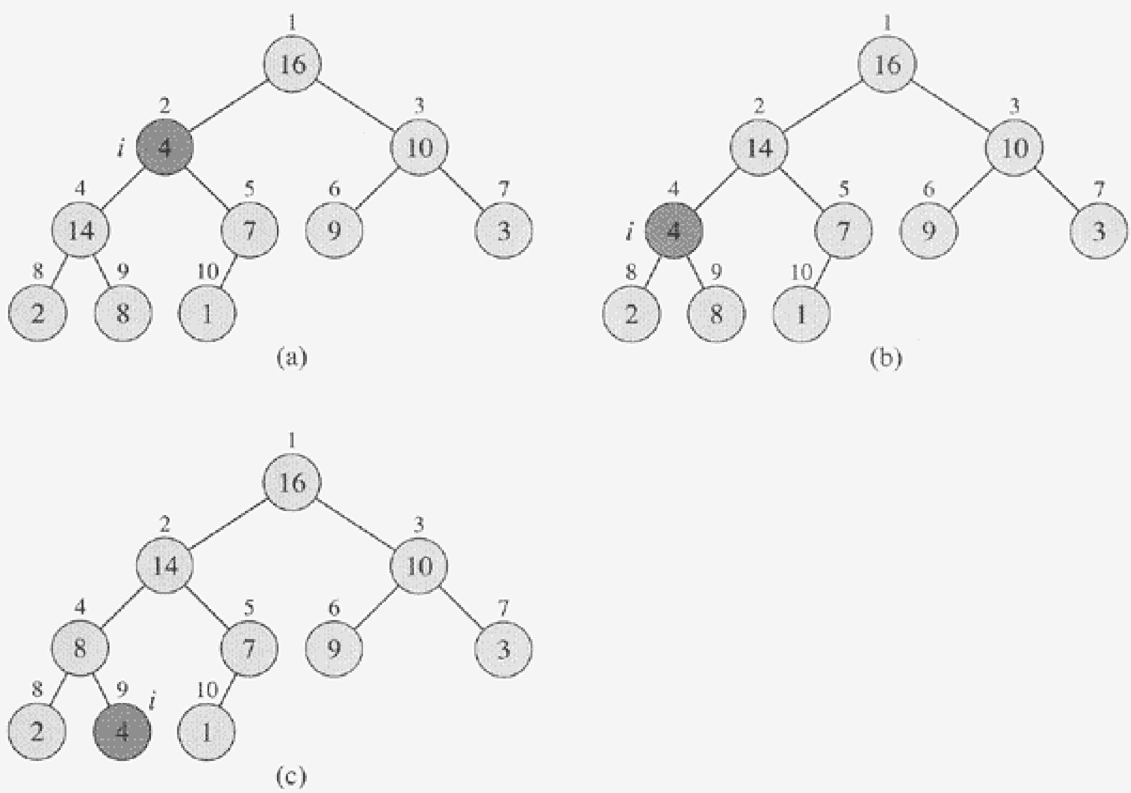

Figure 6.2 The action of MAX-HEAPIFY(A, 2), where *heap-size*[A] = 10. **(a)** The initial configuration, with $A[2]$ at node $i = 2$ violating the max-heap property since it is not larger than both children. The max-heap property is restored for node 2 in **(b)** by exchanging $A[2]$ with $A[4]$, which destroys the max-heap property for node 4. The recursive call MAX-HEAPIFY(A, 4) now has $i = 4$. After swapping $A[4]$ with $A[9]$, as shown in **(c)**, node 4 is fixed up, and the recursive call MAX-HEAPIFY(A, 9) yields no further change to the data structure.

Running time

- **The running time of MAX-HEAPIFY on a subtree of size n rooted at a given node i is the $\Theta(1)$ time to fix up the relationships among the elements A[i], A[LEFT(i)], and A[RIGHT](i), plus the time to run MAX-HEAPIFY on a subtree rooted at one of the children of node i.**
- **MAX-HEAPIFY can be described by the recurrence:**
  - **$T(n) \leq T(2n/3) + \Theta(1)$**
  - **Which is: $T(n) = O(\lg n)$**

Build a heap

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

Build a heap

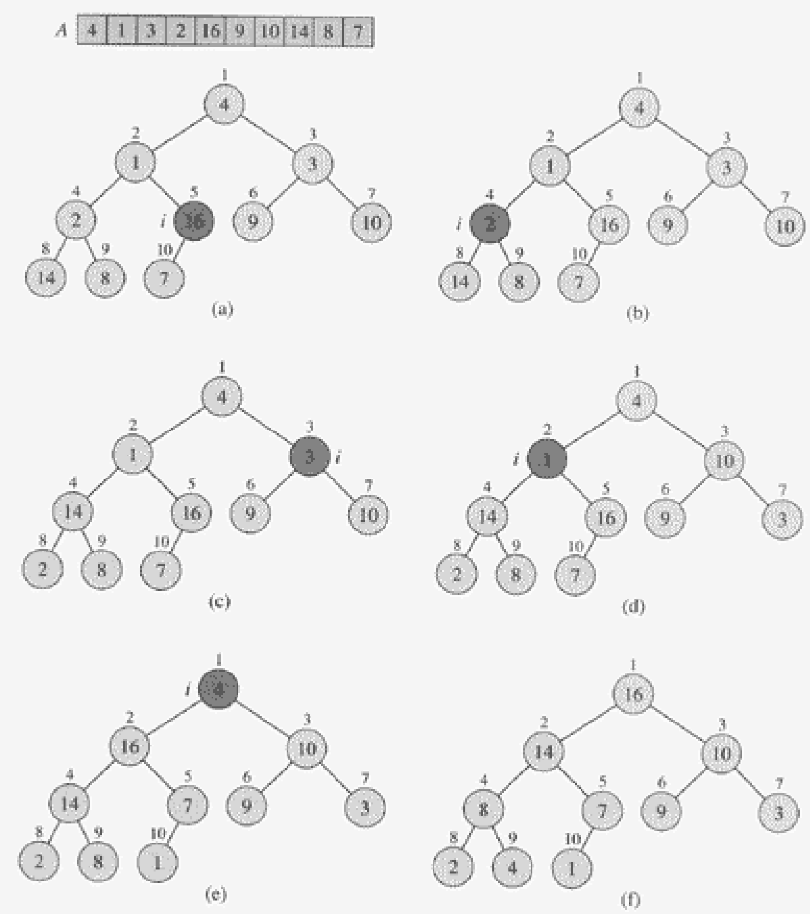

Figure 6.3 The operation of BUILD-MAX-HEAP, showing the data structure before the call to MAX-HEAPIFY in line 3 of BUILD-MAX-HEAP. **(a)** A 10-element input array A and the binary tree it represents. The figure shows that the loop index i refers to node 5 before the call MAX-HEAPIFY(A, i). **(b)** The data structure that results. The loop index i for the next iteration refers to node 4. **(c)–(e)** Subsequent iterations of the **for** loop in BUILD-MAX-HEAP. Observe that whenever MAX-HEAPIFY is called on a node, the two subtrees of that node are both max-heaps. **(f)** The max-heap after BUILD-MAX-HEAP finishes.

Priority queue

- A priority queue is a data structure for maintaining a set *S* of elements, each with an associated value called ***key***.
- A max-priority queue has the following operations:
  - INSERT(S,x) inserts the element x into the set S
  - MAXIMUM(S) returns the element of S with the largest key.
  - EXTRACT-MAX(S) removes and returns the elemtn of S with the largest key.
  - INCREASE-KEY(S,x,k) increases the value of element x's key to the new value k.