Selected Solutions for Chapter 6: Heapsort

Solution to Exercise 6.1-1

Since a heap is an almost-complete binary tree (complete at all levels except possibly the lowest), it has at most $2^{h+1} - 1$ elements (if it is complete) and at least $2^h - 1 + 1 = 2^h$ elements (if the lowest level has just 1 element and the other levels are complete).

Solution to Exercise 6.1-2

Given an n-element heap of height h, we know from Exercise 6.1-1 that

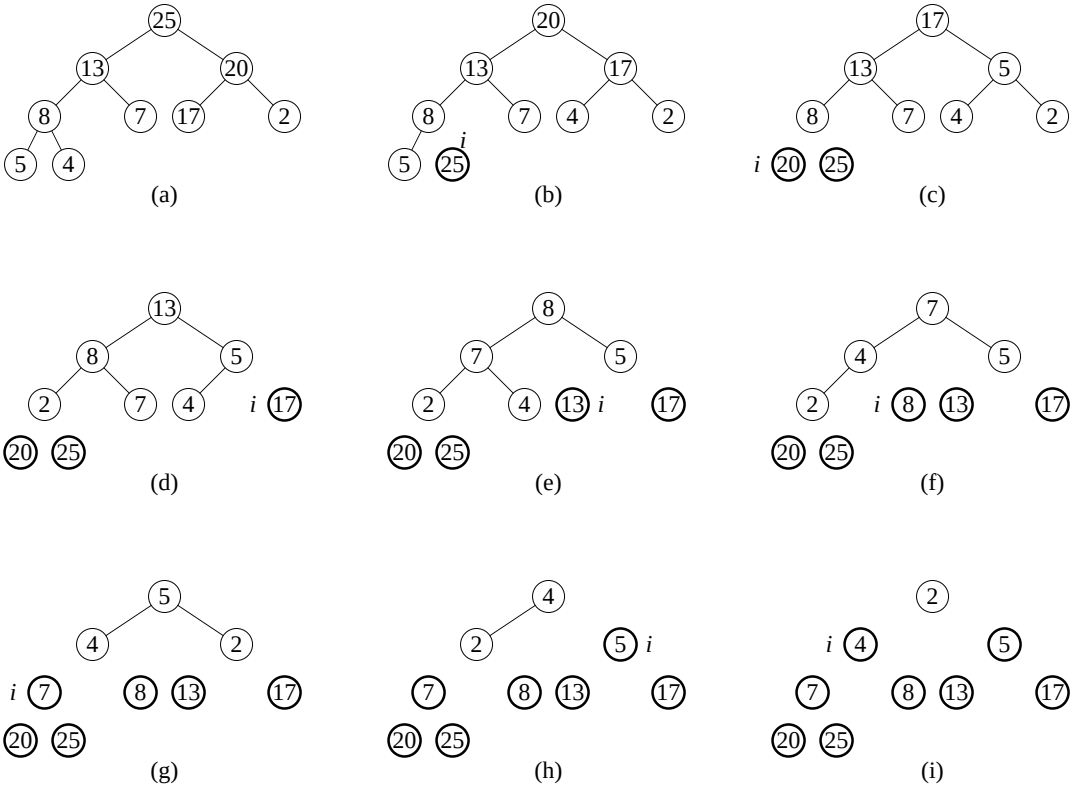
$$2^h \le n \le 2^{h+1} - 1 < 2^{h+1} .$$

Thus, $h \le \lg n < h + 1$. Since h is an integer, $h = \lfloor \lg n \rfloor$ (by definition of $\lfloor\ \rfloor$).

Solution to Exercise 6.2-6

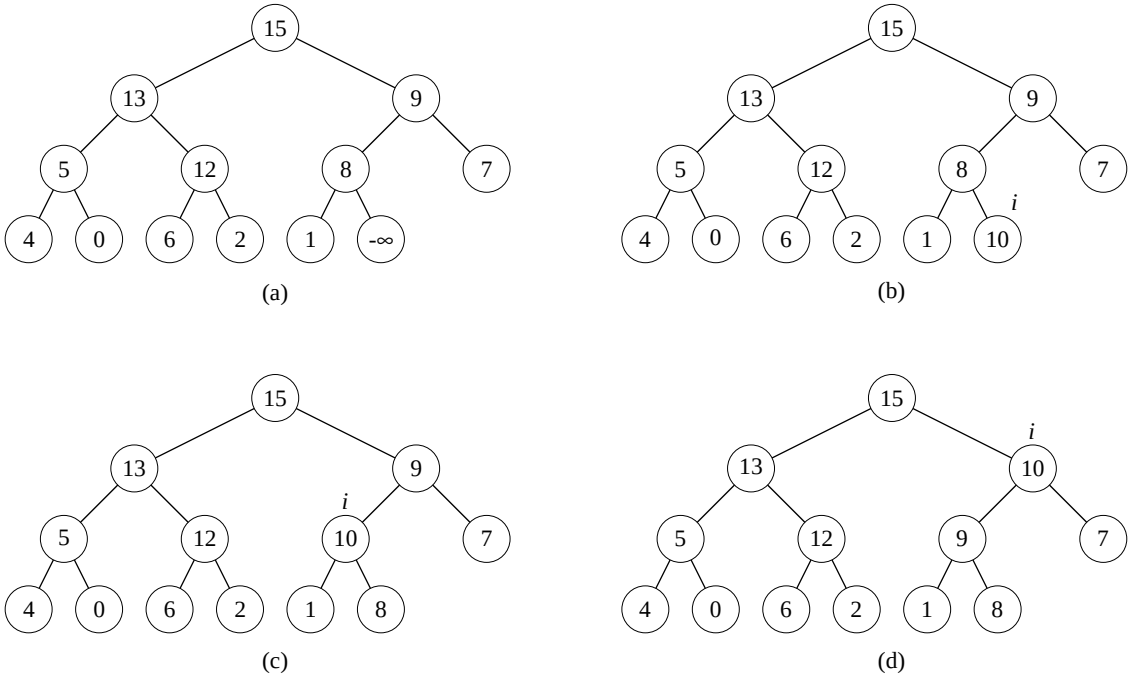
If you put a value at the root that is less than every value in the left and right subtrees, then MAX-HEAPIFY will be called recursively until a leaf is reached. To make the recursive calls traverse the longest path to a leaf, choose values that make MAX-HEAPIFY always recurse on the left child. It follows the left branch when the left child is greater than or equal to the right child, so putting 0 at the root and 1 at all the other nodes, for example, will accomplish that. With such values, MAX-HEAPIFY will be called h times (where h is the heap height, which is the number of edges in the longest path from the root to a leaf), so its running time will be $\Theta(h)$ (since each call does $\Theta(1)$ work), which is $\Theta(\lg n)$. Since we have a case in which MAX-HEAPIFY's running time is $\Theta(\lg n)$, its worst-case running time is $\Omega(\lg n)$.

Solution to Exercise 6.4-1



A	2	4	5	7	8	13	17	20	25

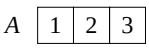
Solution to Exercise 6.5-2



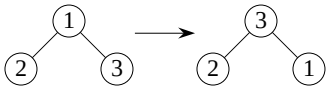
Solution to Problem 6-1

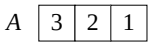
a. The procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP′ do not always create the same heap when run on the same input array. Consider the following counterexample.

Input array A:

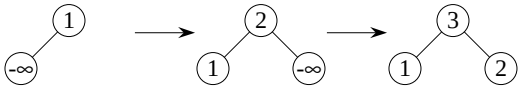


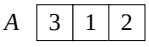
BUILD-MAX-HEAP(A):





BUILD-MAX-HEAP′(A):





b. An upper bound of $O(n \lg n)$ time follows immediately from there being $n - 1$ calls to MAX-HEAP-INSERT, each taking $O(\lg n)$ time. For a lower bound

of $\Omega(n \lg n)$, consider the case in which the input array is given in strictly increasing order. Each call to MAX-HEAP-INSERT causes HEAP-INCREASE-KEY to go all the way up to the root. Since the depth of node i is $\lfloor \lg i \rfloor$, the total time is

$$\begin{aligned}
\sum_{i=1}^{n} \Theta(\lfloor \lg i \rfloor) &\geq \sum_{i=\lceil n/2 \rceil}^{n} \Theta(\lfloor \lg \lceil n/2 \rceil \rfloor) \\
&\geq \sum_{i=\lceil n/2 \rceil}^{n} \Theta(\lfloor \lg(n/2) \rfloor) \\
&= \sum_{i=\lceil n/2 \rceil}^{n} \Theta(\lfloor \lg n - 1 \rfloor) \\
&\geq n/2 \cdot \Theta(\lg n) \\
&= \Omega(n \lg n) \; .
\end{aligned}$$

In the worst case, therefore, BUILD-MAX-HEAP′ requires $\Theta(n \lg n)$ time to build an n-element heap.