Heap and Heap Sort

Heap can be defined as a binary tree with keys assigned to its nodes, one key per node, provided the following two conditions are met:
1. The ***shape property***—the binary tree is ***essentially complete*** (or simply ***complete***), i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing.
2. The ***parental dominance*** or ***heap property***—the key in each node is greater than or equal to the keys in its children. (This condition is considered automatically satisfied for all leaves.)

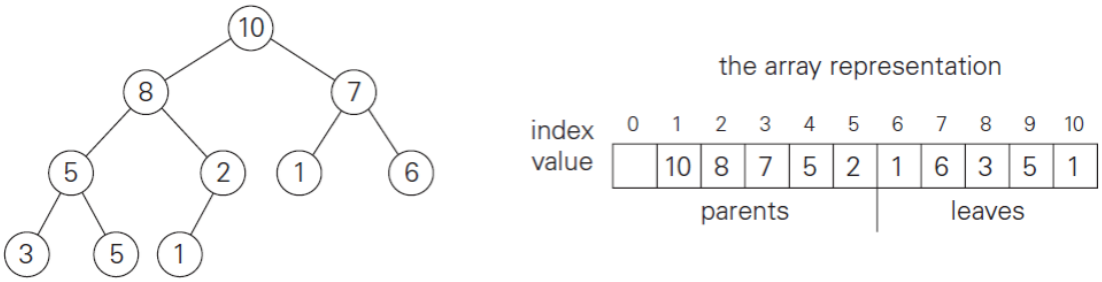

FIGURE 6.10 Heap and its array representation.

properties of heaps

1. There exists exactly one essentially complete binary tree with n nodes. Its height is equal to $\lfloor \log_2 n \rfloor$.
2. The root of a heap always contains its largest element.
3. A node of a heap considered with all its descendants is also a heap.
4. A heap can be implemented as an array by recording its elements in the top-down, left-to-right fashion. It is convenient to store the heap's elements in positions 1 through n of such an array, leaving $H[0]$ either unused or putting there a sentinel whose value is greater than every element in the heap. In such a representation,
   a. the parental node keys will be in the first $\lfloor n/2 \rfloor$ positions of the array, while the leaf keys will occupy the last $\lceil n/2 \rceil$ positions;
   b. the children of a key in the array's parental position i $(1 \le i \le \lfloor n/2 \rfloor)$ will be in positions $2i$ and $2i + 1$, and, correspondingly, the parent of a key in position i $(2 \le i \le n)$ will be in position $\lfloor i/2 \rfloor$.

Thus, we could also define a heap as an array $H[1..n]$ in which every element in position i in the first half of the array is greater than or equal to the elements in positions $2i$ and $2i + 1$, i.e.,

$$H[i] \ge \max\{H[2i], H[2i + 1]\} \quad \text{for } i = 1, \ldots, \lfloor n/2 \rfloor.$$

Bottom Up Construction

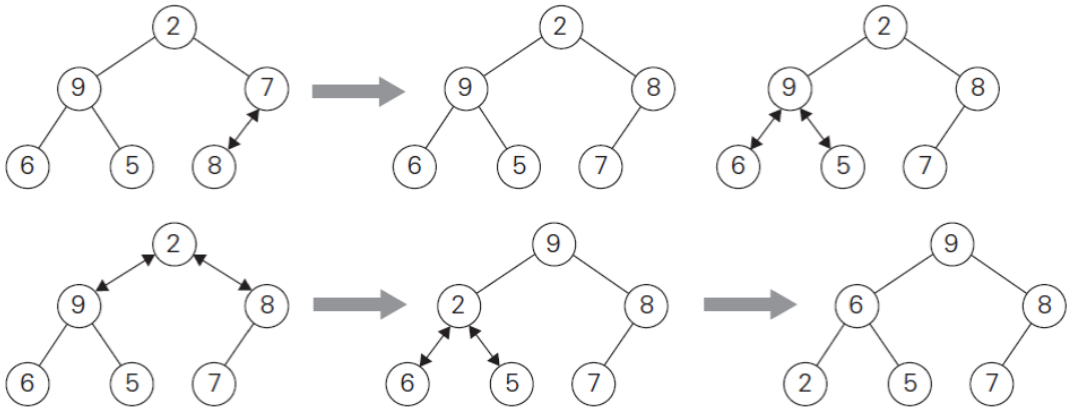

FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8. The double-headed arrows show key comparisons verifying the parental dominance.

ALGORITHM *HeapBottomUp*$(H[1..n])$

```
//Constructs a heap from elements of a given array
// by the bottom-up algorithm
//Input: An array H[1..n] of orderable items
//Output: A heap H[1..n]
for i ← ⌊n/2⌋ downto 1 do
    k ← i;   v ← H[k]
    heap ← false
    while not heap and 2 * k ≤ n do
        j ← 2 * k
        if j < n    //there are two children
            if H[j] < H[j + 1]  j ← j + 1
        if v ≥ H[j]
            heap ← true
        else H[k] ← H[j];    k ← j
    H[k] ← v
```

$$C_{worst}(n) = \sum_{i=0}^{h-1} \sum_{\text{level } i \text{ keys}} 2(h-i) = \sum_{i=0}^{h-1} 2(h-i)2^i = 2(n - \log_2(n+1)),$$

Top Down Construction

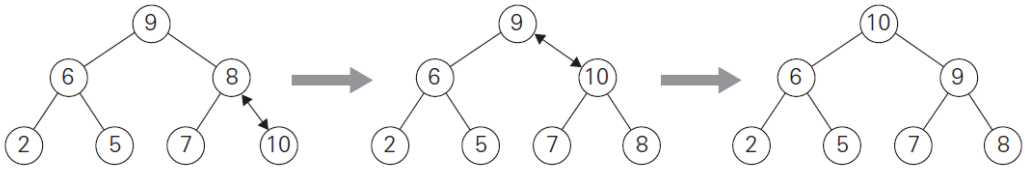

FIGURE 6.12 Inserting a key (10) into the heap constructed in Figure 6.11. The new key is sifted up via a swap with its parent until it is not larger than its parent (or is in the root).

Maximum Key Deletion from a heap

Step 1 Exchange the root's key with the last key K of the heap.

Step 2 Decrease the heap's size by 1.

Step 3 "Heapify" the smaller tree by sifting K down the tree exactly in the same way we did it in the bottom-up heap construction algorithm. That is, verify the parental dominance for K: if it holds, we are done; if not, swap K with the larger of its children and repeat this operation until the parental dominance condition holds for K in its new position.

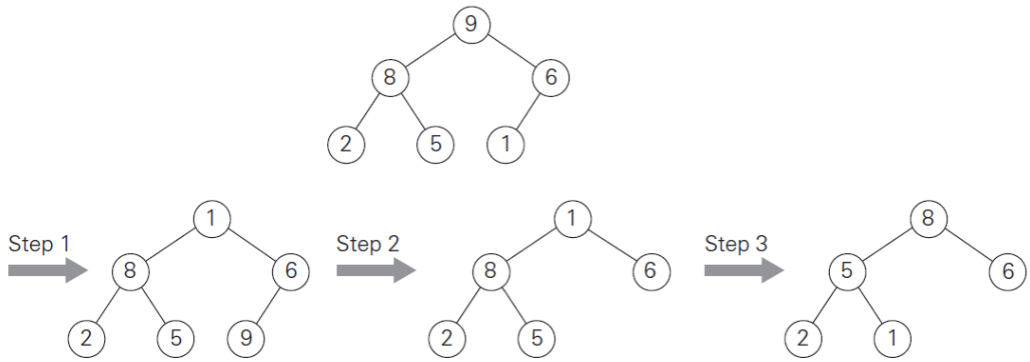

FIGURE 6.13 Deleting the root's key from a heap. The key to be deleted is swapped with the last key after which the smaller tree is "heapified" by exchanging the new key in its root with the larger key in its children until the parental dominance requirement is satisfied.

Heapsort

Now we can describe ***heapsort***—an interesting sorting algorithm discovered by J. W. J. Williams [Wil64]. This is a two-stage algorithm that works as follows.

Stage 1 (heap construction): Construct a heap for a given array.

Stage 2 (maximum deletions): Apply the root-deletion operation $n - 1$ times to the remaining heap.

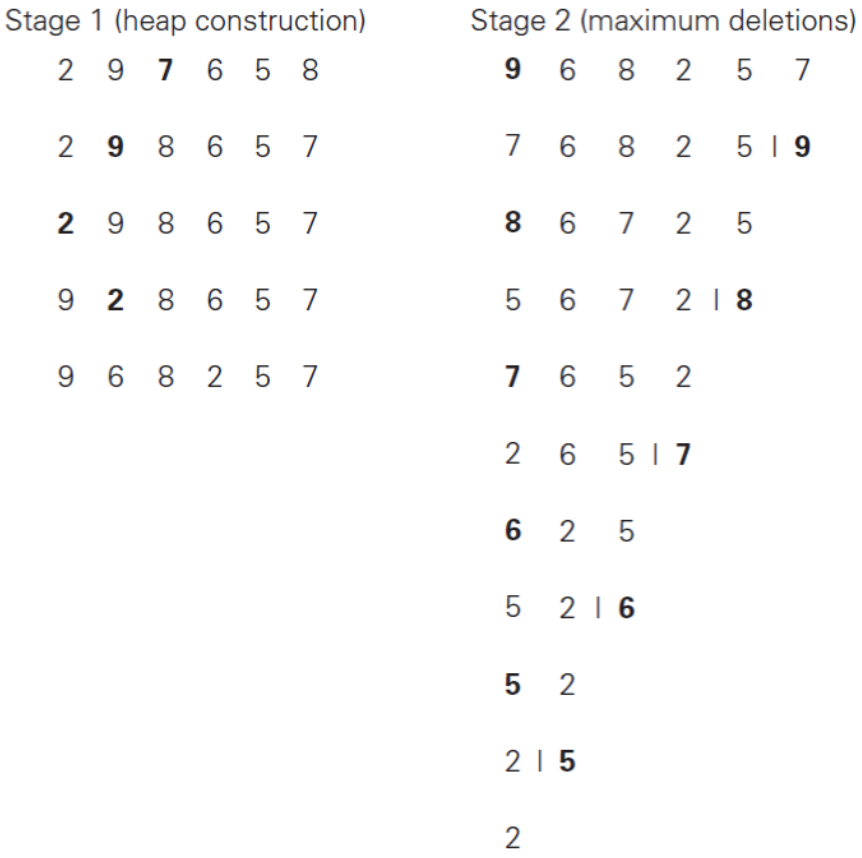

FIGURE 6.14 Sorting the array 2, 9, 7, 6, 5, 8 by heapsort.

$$C(n) \le 2\lfloor \log_2(n-1) \rfloor + 2\lfloor \log_2(n-2) \rfloor + \cdots + 2\lfloor \log_2 1 \rfloor \le 2 \sum_{i=1}^{n-1} \log_2 i$$

$$\le 2 \sum_{i=1}^{n-1} \log_2(n-1) = 2(n-1) \log_2(n-1) \le 2n \log_2 n.$$