DESIGN AND ANALYSIS OF ALGORITHMS

Surabhi Narayan

Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Heap and Heap Sort

Surabhi Narayan
Department of Computer Science & Engineering

Heap and Heap Sort

Heap is partially ordered data structure that is especially suitable for implementing priority queues.

A ***priority queue*** is a multiset of items with an orderable characteristic called an item's ***priority***, with the following operations:

- finding an item with the highest (i.e., largest) priority
- deleting an item with the highest priority
- adding a new item to the multiset

Heap and Heap Sort

DEFINITION : A ***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.)

Heap and Heap Sort

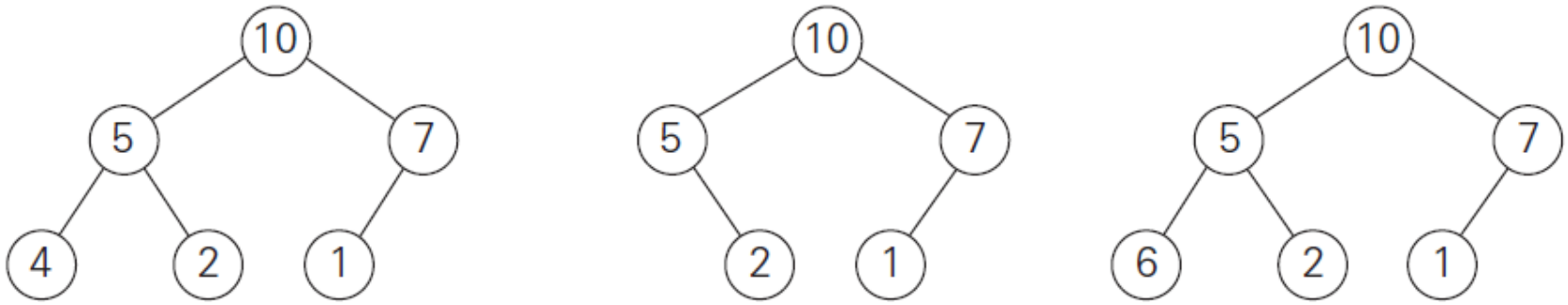

FIGURE 6.9 Illustration of the definition of heap: only the leftmost tree is a heap.

Heap and Heap Sort

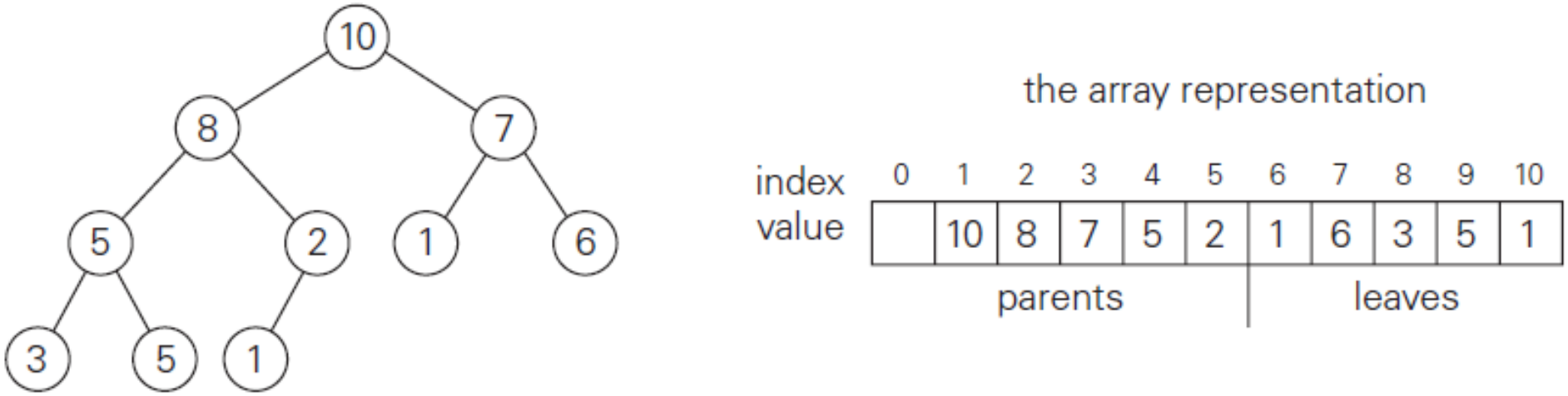

FIGURE 6.10 Heap and its array representation.

Properties of Heap

There exists exactly one essentially complete binary tree with n nodes. Its height is equal to $\log_2 n$.

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 topdown, 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 $n/2$ positions of the array, while the leaf keys will occupy the last $n/2$ positions;

b. the children of a key in the array's parental position i $(1 \le i \le n/2)$ 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 $i/2$.

Heap and Heap Sort

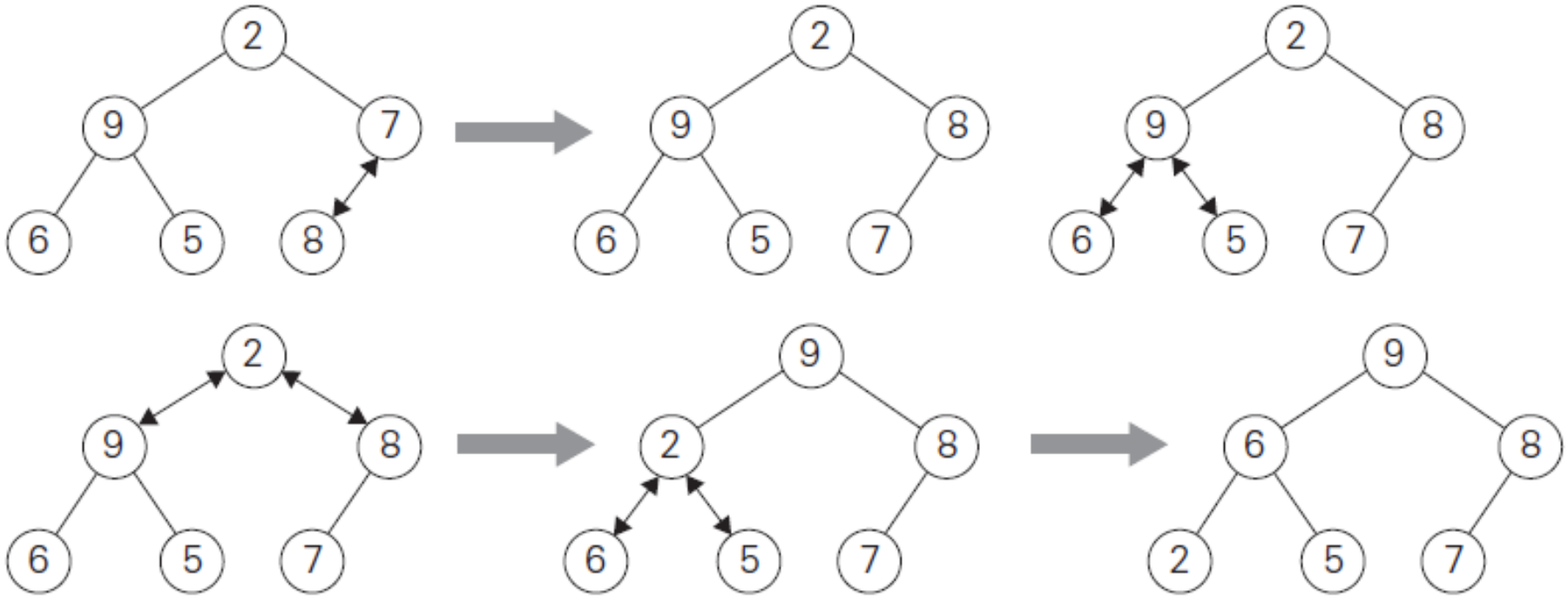

FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8. The double-headed arrows show key comparisons verifying the parental dominance.

Heap and Heap Sort

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
```

Top Down Approach

- Insert a new key *K* into a heap by attaching a new node with key *K* in it after the last leaf of the existing heap.
- Sift *K* up to its appropriate place in the new heap as follows.
  - Compare *K* with its parent's key: if the latter is greater than or equal to *K*, stop (the structure is a heap);
  - otherwise, swap these two keys and compare *K* with its new parent.
- This swapping continues until *K* is not greater than its last parent or it reaches the root
- Insertion operation cannot require more key comparisons than the heap's height.
- The time efficiency of insertion is in $O(\log n)$.

Heap and Heap Sort

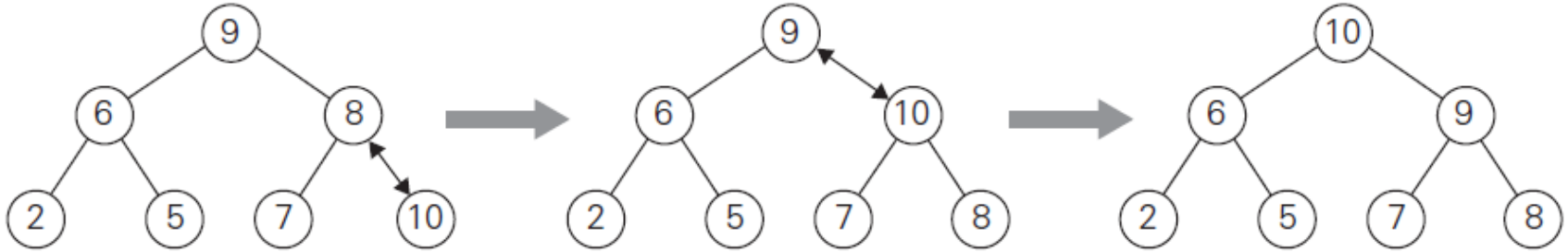

FIGURE 6.12 Inserting a key (10) into the heap constructed in Figure 6.11. The new key is sifted up via a swap with its parent until it is not larger than its parent (or is in the root).

Heap and Heap Sort

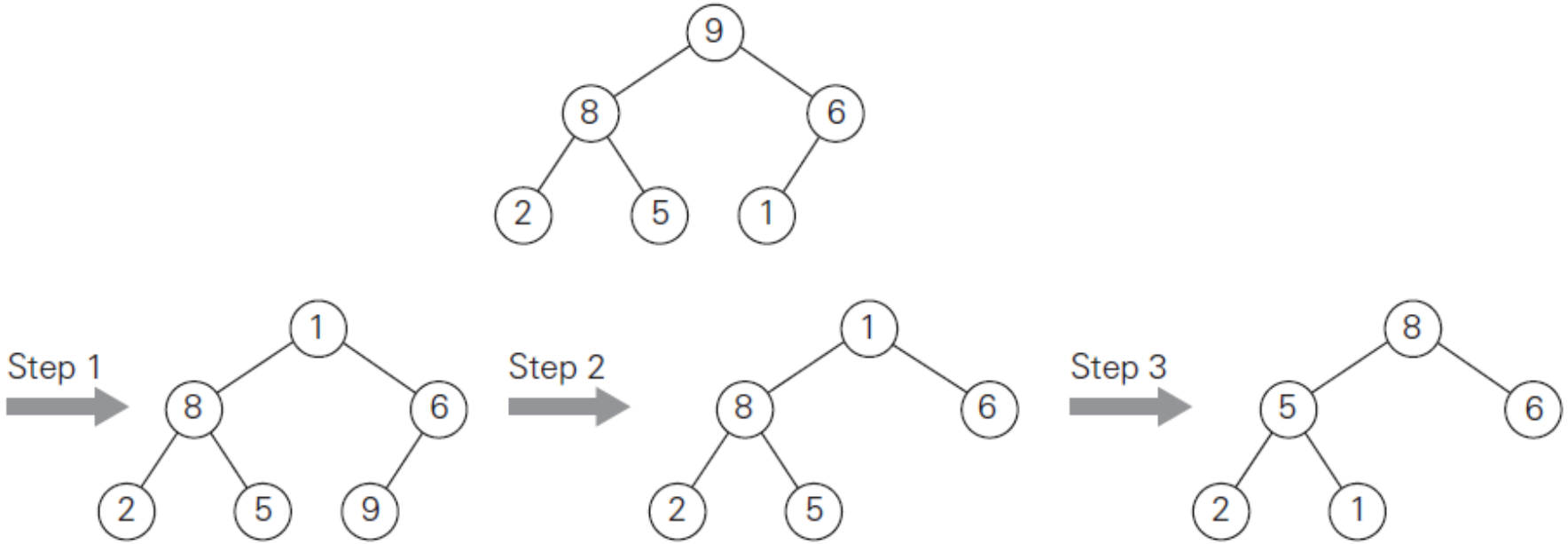

FIGURE 6.13 Deleting the root's key from a heap. The key to be deleted is swapped with the last key after which the smaller tree is "heapified" by exchanging the new key in its root with the larger key in its children until the parental dominance requirement is satisfied.

Heap and Heap Sort

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.

Heap Sort

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.

As a result, the array elements are eliminated in decreasing order. But since under the array implementation of heaps an element being deleted is placed last, the resulting array will be exactly the original array sorted in increasing order.

Heap and Heap Sort

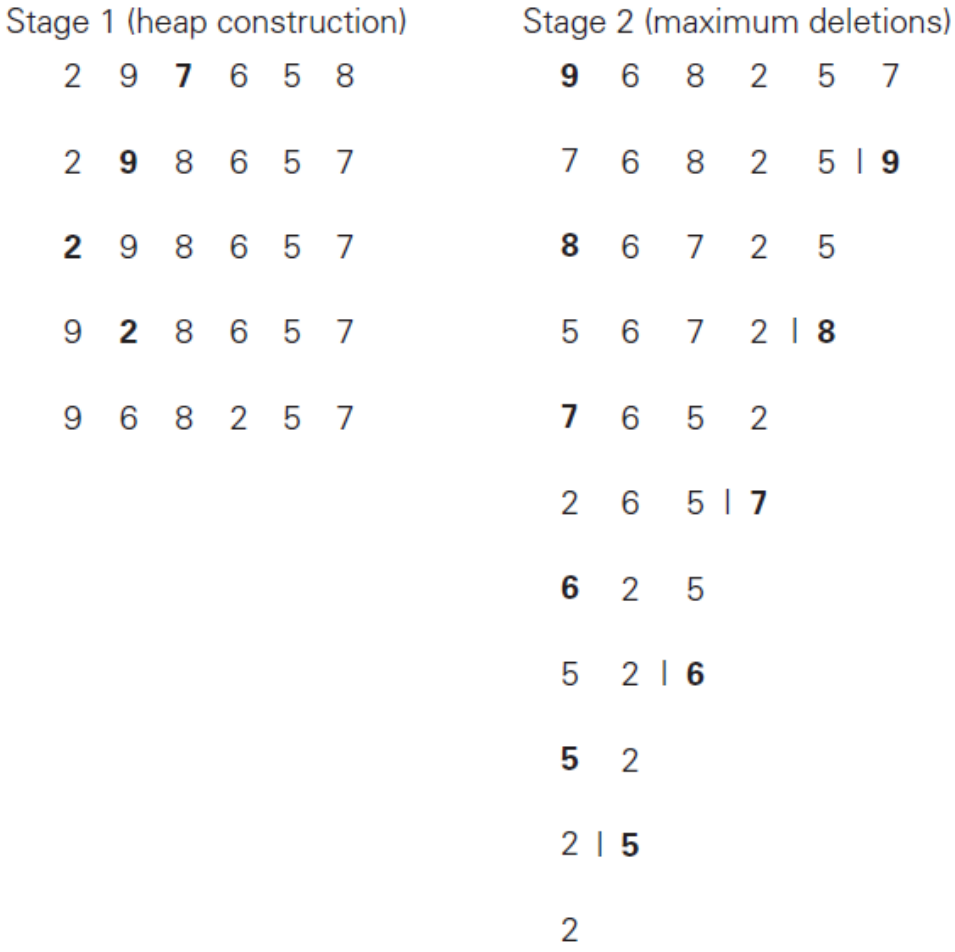

FIGURE 6.14 Sorting the array 2, 9, 7, 6, 5, 8 by heapsort.

Complexity Analysis of Heap Sort

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

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

THANK YOU

Surabhi Narayan
Department of Computer Science &Engineering

surabhinarayan@pes.edu