Heaps

Adapted from the CLRS book slides

SORTING ALGORITHMS

Goals

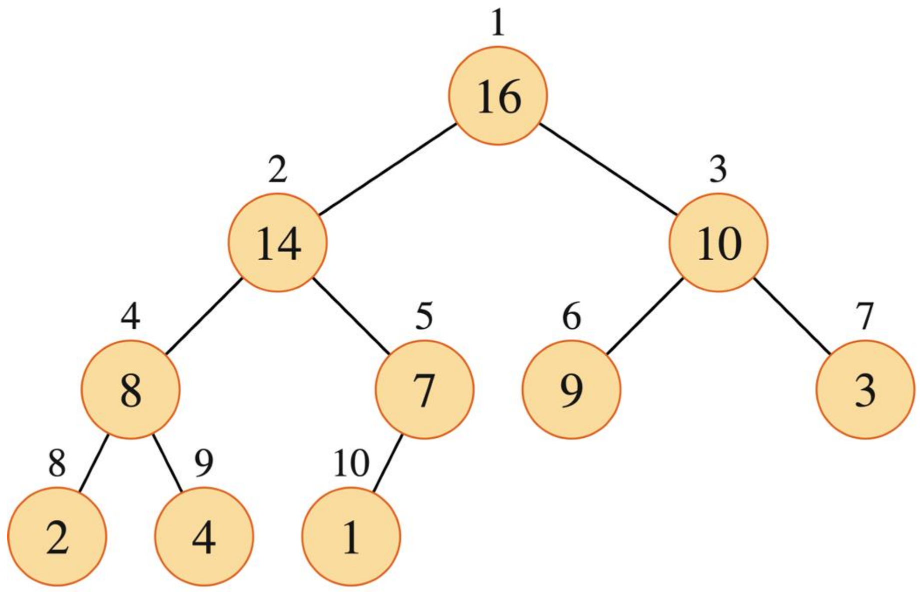
- $O(n \lg n)$ worst case—like merge sort.
- Sorts in place—like insertion sort.
- Heapsort combines the best of both algorithms.

To understand heapsort, we'll cover heaps and heap operations, and then we'll take a look at priority queues.

HEAP DATA STRUCTURE

A heap (***not*** garbage-collected storage) is a nearly complete binary tree.

- ***Height*** of node = # of edges on a longest simple path from the node down to a leaf.
- ***Height*** of heap = height of root = $\Theta(\lg n)$.



HEAP DATA STRUCTURES (continued)

A heap can be stored as an array A.

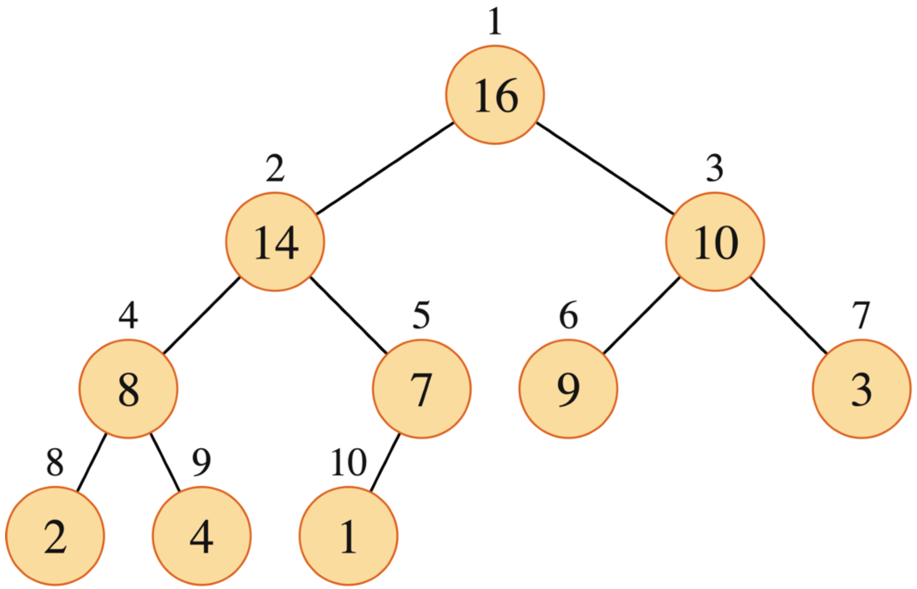
- Root of tree is $A[1]$.
- Parent of $A[i] = A[\lfloor i/2 \rfloor]$.
- Left child of $A[i] = A[2i]$.
- Right child of $A[i] = A[2i+1]$.

PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

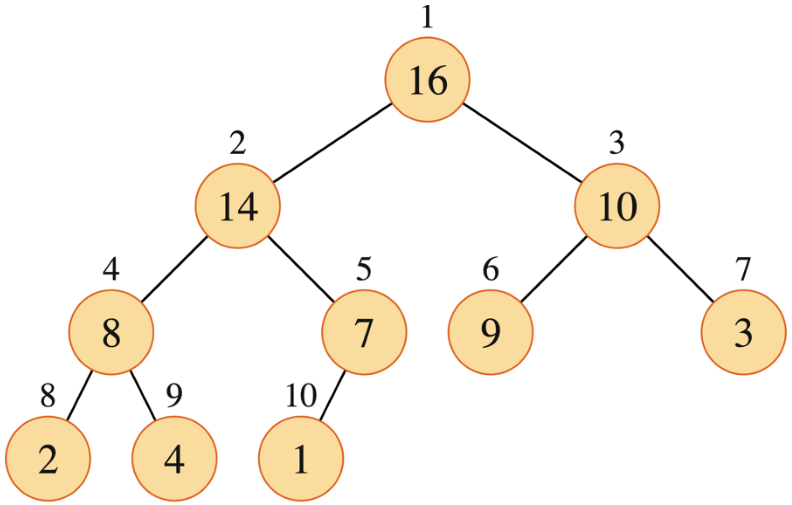
- Attribute $A.heap\text{-}size$ says how many elements are stored in A. Only the elements in $A[1 : A.heap\text{-}size]$ are in the heap.

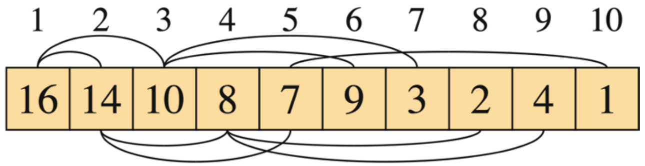


EXAMPLE

Of a max-heap in array with *heap-size* $= 10$.

- For max-heaps (largest element at root), ***max-heap property:*** for all nodes i, excluding the root, $A[\text{PARENT}(i)] \geq A[i]$.
- For min-heaps (smallest element at root), ***min-heap property:*** for all nodes i, excluding the root, $A[\text{PARENT}(i)] \leq A[i]$.





MAINTAINING THE HEAP PROPERTY

MAX-HEAPIFY is important for manipulating max-heaps. It is used to maintain the max-heap property.

Before MAX-HEAPIFY, $A[i]$ may be smaller than its children.

Assume that left and right subtrees of i are max-heaps. (No violations of maxheap property within the left and right subtrees. The only violation within the subtree rooted at i could be between i and its children.)

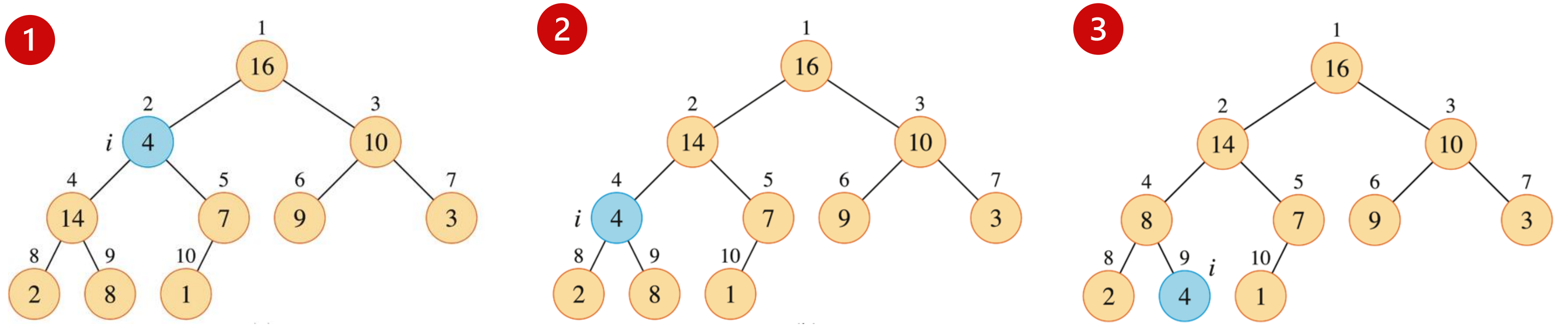
After MAX-HEAPIFY, subtree rooted at i is a max-heap.

PSEUDOCODE

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

EXAMPLE

Run MAX-HEAPIFY on the following heap example.



Time: $O(\lg n)$.

BUILDING A HEAP

The following procedure, given an unordered array $A[1{:}n]$, will produce a max-heap of the n elements in A.

```
BUILD-MAX-HEAP(A, n)
  A.heap-size = n
  for i = ⌊n/2⌋ downto 1
      MAX-HEAPIFY(A, i)
```

EXAMPLE

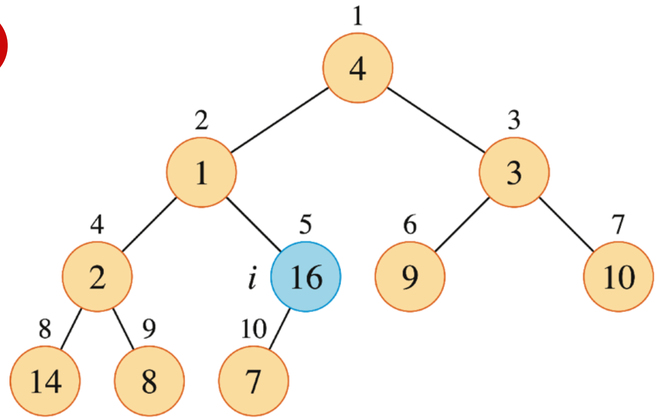
Building a max-heap by calling BUILD-MAX-HEAP$(A, 10)$ on the following unsorted array array $A[1{:}10]$ results in the first heap example.

- $A.heap\text{-}size$ is set to 10.
- i starts off as 5.
- MAX-HEAPIFY is applied to subtrees rooted at nodes (in order): $A[5]$, $A[4]$, $A[3]$, $A[2]$, $A[1]$.

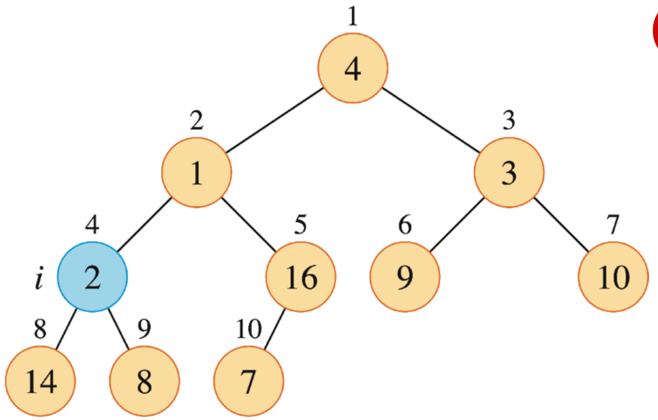
EXAMPLE (continued)

A	4	1	3	2	16	9	10	14	8	7

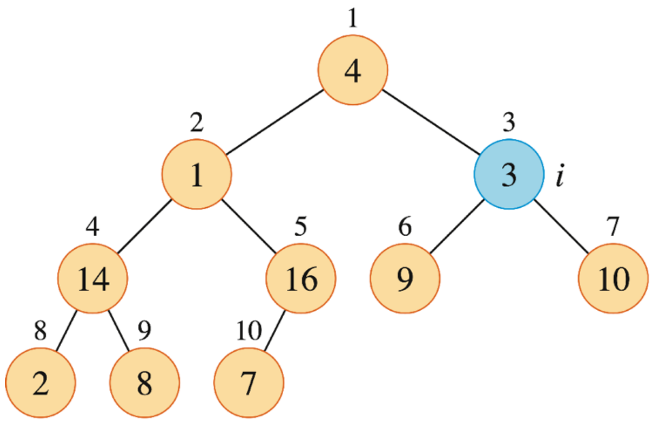
1



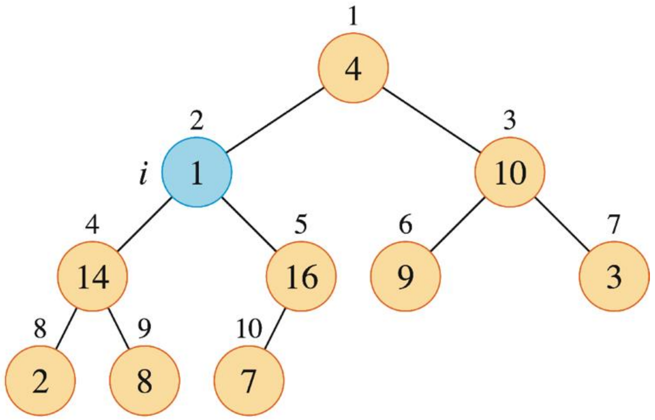
2



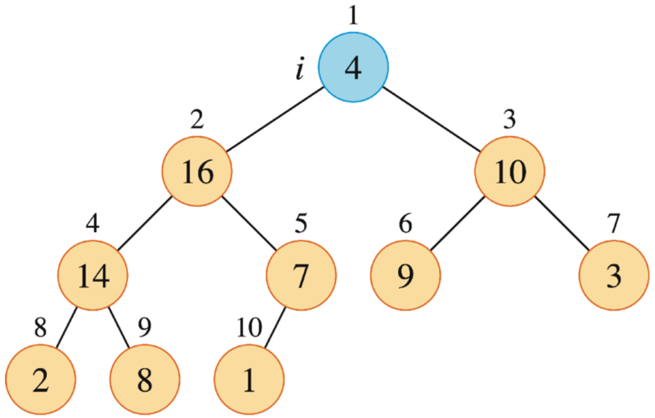
3



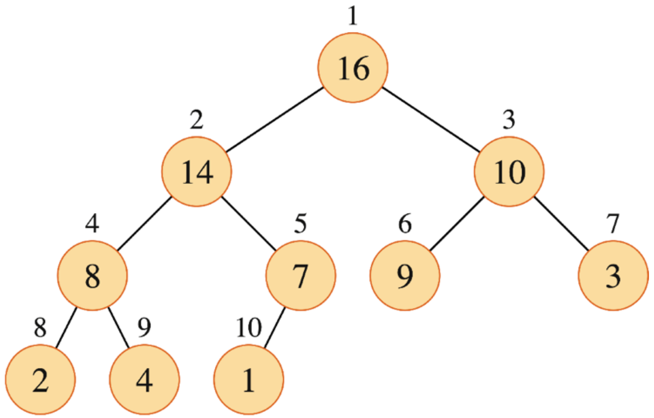
4



5



6



ANALYSIS

Simple bound:

$O(n)$ calls to MAX-HEAPIFY, each of which takes $O(\lg n)$ time $\Longrightarrow O(n \lg n)$.

Tighter analysis:

Observation: Time to run MAX-HEAPIFY is linear in the height of the node it's run on, and most nodes have small heights. Have $\leq \lceil n/2^{h+1} \rceil$ nodes of height h (see Exercise 6.3-4), and height of heap is $\lfloor \lg n \rfloor$ (Exercise 6.1-2).

ANALYSIS (continued)

The time required by MAX-HEAPIFY when called on a node of height h is $O(h)$, so the total cost of BUILD-MAX-HEAP is

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) .$$

Evaluate the last summation by substituting $x = 1/2$ in the formula (A.11) $(\sum_{k=0}^{\infty} k x^k)$, which yields

$$\begin{aligned} \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} &< \sum_{h=0}^{\infty} \frac{h}{2^h} \\ &= \frac{1/2}{(1-1/2)^2} \\ &= 2 . \end{aligned}$$

Thus, the running time of BUILD-MAX-HEAP is $O(n)$.