Heap: nearly complete binary tree filled on all levels. Except possibly the lowest, which is filled from the left up to a point.

Two representations: Tree. Array.
See slide 2.

A.length # elements in array

A.heap-size ≤ A.length
elements stored in heap

A[1] root

PARENT(i)

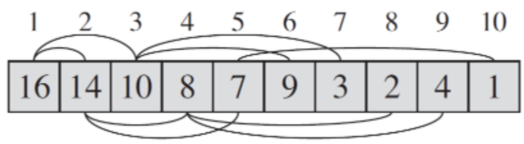
return $\lfloor i/2 \rfloor$

LEFT(i)

return $2i$

RIGHT(i)

return $2i + 1$



(b)

max-heap: $\forall i > 1 \quad A[\text{PARENT}(i)] \geq A[i]$

min-heap: $\quad \ldots \quad \leq A[i]$

height of node = # Kanten in Pfad von Wurzel zu Knoten

height of heap = maximum aller height of node

n Elemente $\Rightarrow$ height of heap $= \Theta(\lg n)$

MAX-HEAPIFY

Input: A, i mit der Eigenschaft, dass Unterbäume mit Wurzeln LEFT(i), RIGHT(i) sind MAX-HEAPS. A[i] kann aber kleiner als seine Kinder sein.

Output: A: Teilbaum mit Wurzel i ist Max-heap und hat dieselbe Knotenmenge wie vorher. Der Rest von A ändert sich nicht

```
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)
```

Laufzeit: $T(n) \leq T(2n/3) + \Theta(1)$

Weil die Größe des Teilbaumes höchstens 2/3 Größe des urspr. Baumes ist.

MAX

Laufzeit HEAPIFY $a = 1$, $b = 3/2$

$f(n) = C$

Vergleiche $f(n)$ mit $n^0 = 1$

Fall 2 des Master Theorems

$$T(n) = O(\lg n) = O(h)$$

h height of Heap.

BUILD-MAX-HEAP(A)

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

At the start of each iteration of the **for** loop of lines 2–3, each node $i+1, i+2, \ldots, n$ is the root of a max-heap.

Initialization: Prior to the first iteration of the loop, $i = \lfloor n/2 \rfloor$. Each node $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$ is a leaf and is thus the root of a trivial max-heap.

Maintenance: To see that each iteration maintains the loop invariant, observe that the children of node i are numbered higher than i. By the loop invariant, therefore, they are both roots of max-heaps. This is precisely the condition required for the call MAX-HEAPIFY(A, i) to make node i a max-heap root. Moreover, the MAX-HEAPIFY call preserves the property that nodes $i+1, i+2, \ldots, n$ are all roots of max-heaps. Decrementing i in the **for** loop update reestablishes the loop invariant for the next iteration.

Termination: At termination, $i = 0$. By the loop invariant, each node $1, 2, \ldots, n$ is the root of a max-heap. In particular, node 1 is.

Heap with n elements has

height $\lfloor \lg n \rfloor$

and at most $\lfloor n / 2^{h+1} \rfloor$
nodes of height h.

Running time of build max heap:

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

$$= O\Big(n \underbrace{\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}}_{\leq 2}\Big)$$

$|x| < 1$

$$\sum_{k=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2}$$

$x = 1/2$

$$= O(n)$$

HEAPSORT(A)

1 BUILD-MAX-HEAP(A)
2 **for** $i = A.length$ **downto** 2
3   exchange $A[1]$ with $A[i]$
4     $A.heap\text{-}size = A.heap\text{-}size - 1$
5   MAX-HEAPIFY(A, 1)

Laufzeit HEAP-SORT: $O(n \log n)$