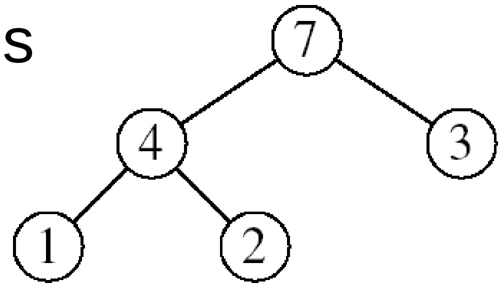
Heapsort

- Goal:
  - Sort an array using heap representations



- Idea:
  - Build a **max-heap** from the array
  - Swap the root (the maximum element) with the last element in the array
  - “Discard” this last node by decreasing the heap size
  - Call MAX-HEAPIFY on the new root
  - Repeat this process until only one node remains

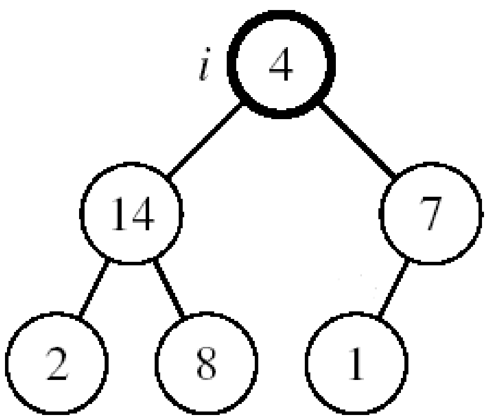
Alg: HEAPSORT(*A*)

```
BUILD-MAX-HEAP(A)                          O(n)
for i ← length[A] downto 2
    do exchange A[1] ↔ A[i]                n-1 times
        MAX-HEAPIFY(A, 1, i - 1)           O(lgn)
```

- Running time: $O(n\lg n)$ --- Can be shown to be $\Theta(n\lg n)$

Maintaining the Heap Property

- Assumptions:
  - Left and Right subtrees of i are max-heaps
  - $A[i]$ may be smaller than its children



Alg: MAX-HEAPIFY(A, i, n)

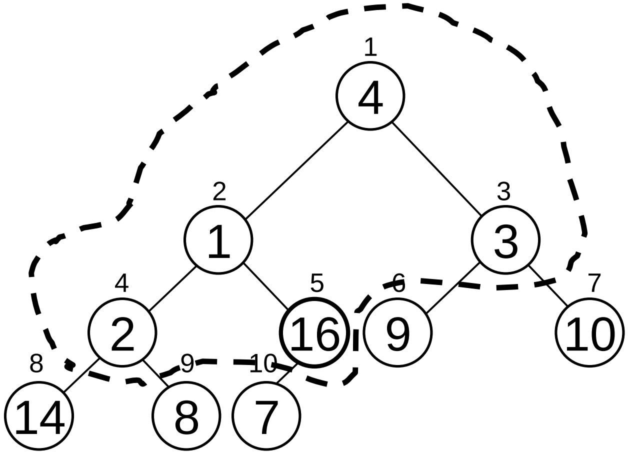
$l \leftarrow$ LEFT(i)
$r \leftarrow$ RIGHT(i)
if $l \le n$ and $A[l] > A[i]$
    then $largest \leftarrow l$
    else $largest \leftarrow i$
if $r \le n$ and $A[r] > A[largest]$
    then $largest \leftarrow r$
if $largest \ne i$
    then exchange $A[i] \leftrightarrow A[largest]$
        MAX-HEAPIFY(A, $largest$, n)

Building a Heap

- Convert an array $A[1 \ldots n]$ into a max-heap ($n = length[A]$)
- The elements in the subarray $A[(\lfloor n/2 \rfloor+1) .. n]$ are leaves
- Apply MAX-HEAPIFY on elements between 1 and $\lfloor n/2 \rfloor$

Alg: BUILD-MAX-HEAP(A)

1. n = length[A]
2. **for** $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1
3. **do** MAX-HEAPIFY(A, i, n)



A:

4	1	3	2	16	9	10	14	8	7

Running Time of BUILD MAX HEAP

Alg: BUILD-MAX-HEAP(A)

```
n = length[A]
 for i ← ⌊n/2⌋ downto 1
      do MAX-HEAPIFY(A, i, n)        O(lgn)
```

$O(n)$

$\Rightarrow$ Running time: $O(n\lg n)$

- This is not an asymptotically tight upper bound