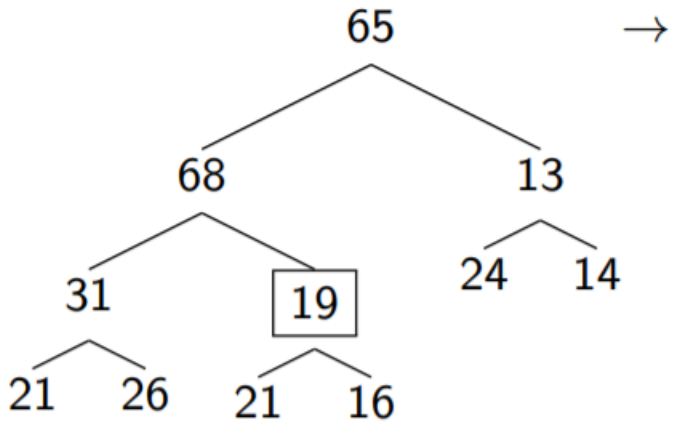
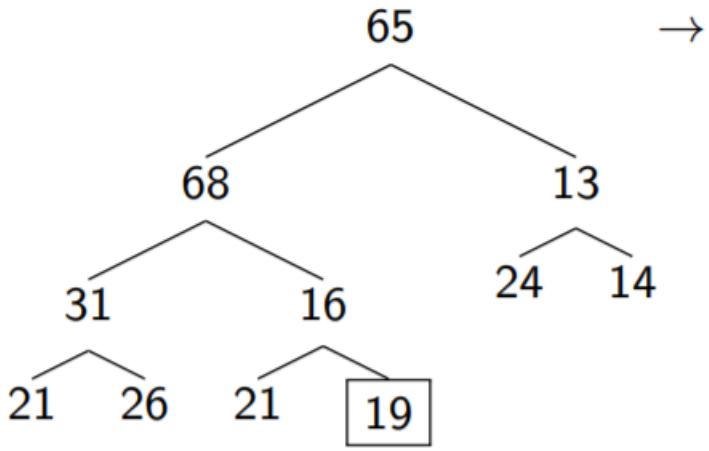
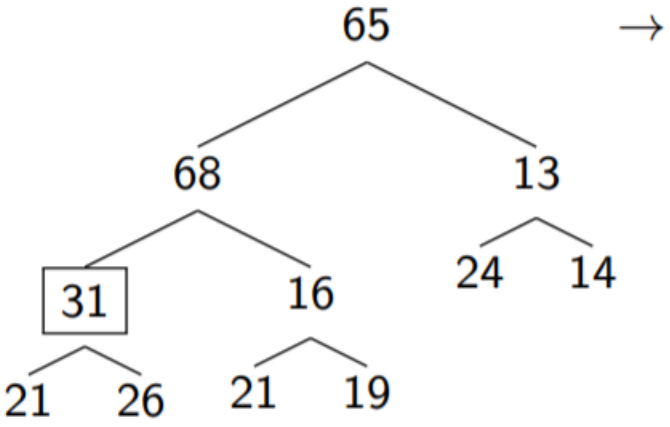
Heapify and Heapsort

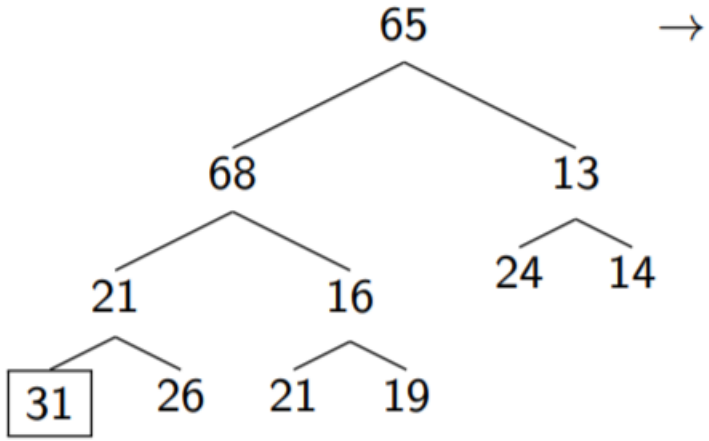
Building a binary heap

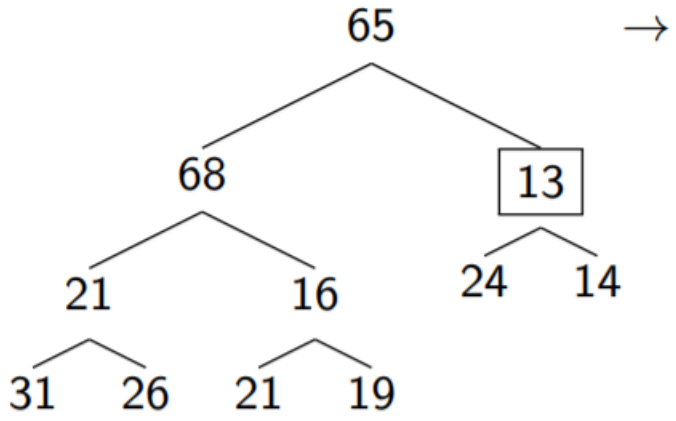
- To build a heap from a list of elements:
  1. Copy the elements into the array, in any order
  2. Percolate each element down the heap, starting with the second last level, moving right to left along levels

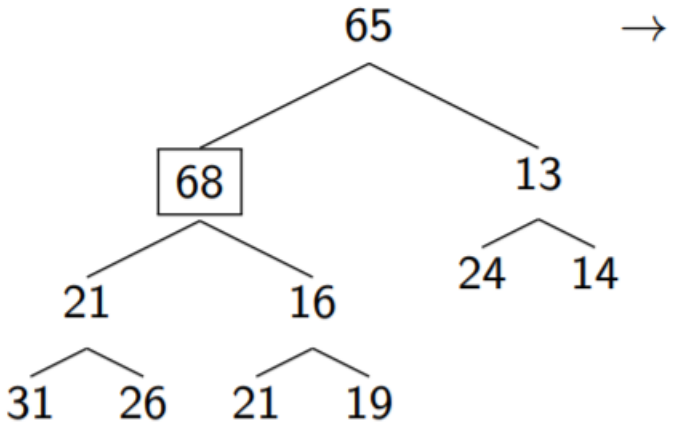

65
68
13
31
19
24
14
21
26
21
16

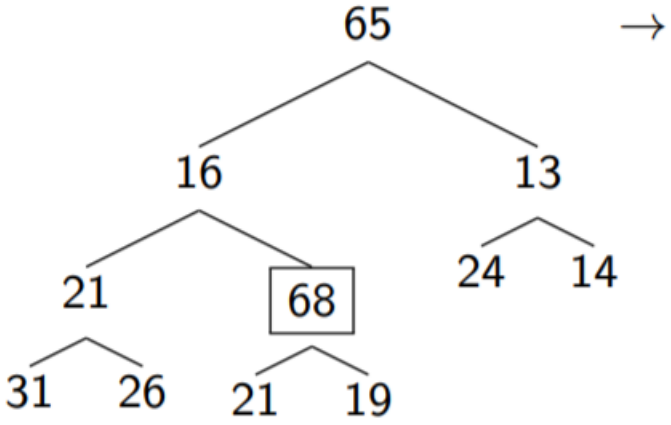

65
68
13
31
16
24
14
21
26
21
19

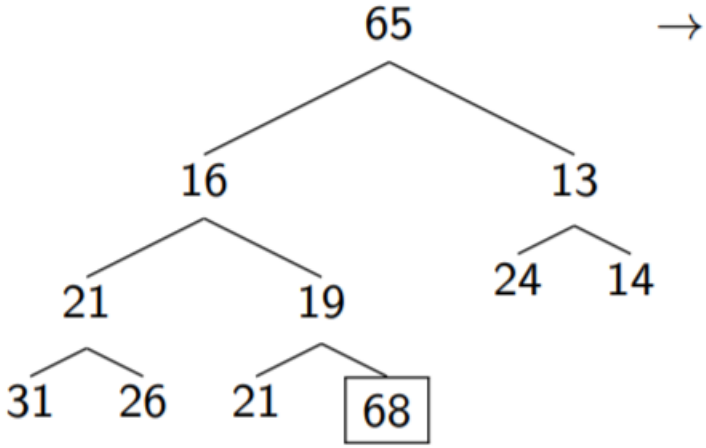

65
68
13
31
16
24
14
21
26
21
19

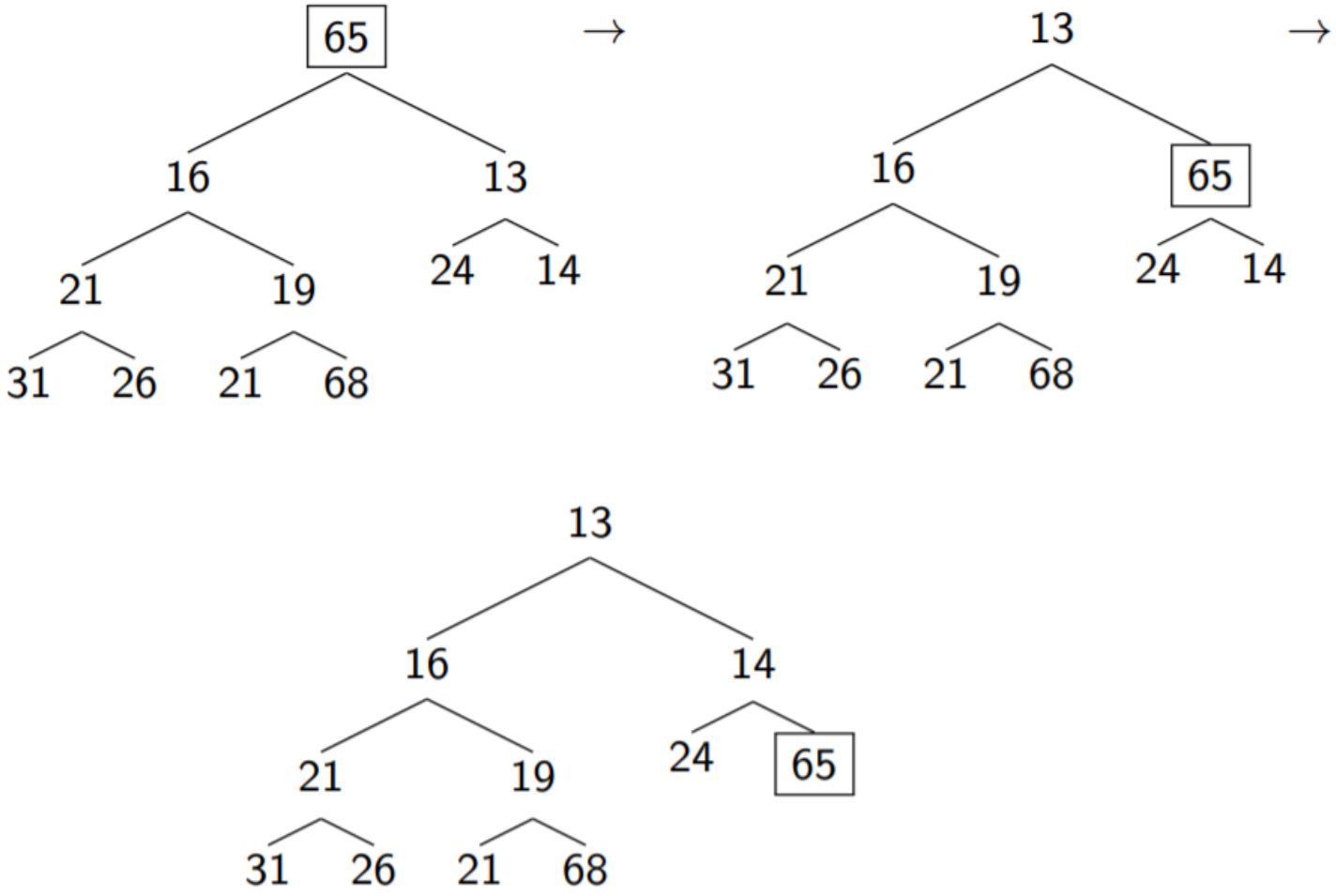

65
68
13
21
16
24
14
31
26
21
19


65
68
13
21
16
24
14
31
26
21
19


65
68
13
21
16
24
14
31
26
21
19


65
16
13
21
68
24
14
31
26
21
19


65
16
13
21
19
24
14
31
26
21
68


65
16
13
21
19
24
14
31
26
21
68
13
16
65
21
19
24
14
31
26
21
68
13
16
14
21
19
24
65
31
26
21
68

Why heapify works

- Every value in the second half of the array is a leaf
- Because we work level by level, for every node we encounter, each of its children has already been "heapified"
  - Percolating the node down doesn't change that
- By the time we reach the root, the entire array is heapified

Implementation

```
heapify(arr, size)
 heap[1:(size+1)] = arr // need to store arr
                        // starting index 1
 for (node = PARENT(size); node >= 1; node--)
    percolate_down(heap, node, size) // similar to
                                     //extract_min

 return heap
```

Runtime of heapify

- n/2 calls to percolate_down
- percolate_down is O(log(n)) in the worst case
- Upper bound: O((n/2)log(n))
- Can get a tighter upper bound

- There are (at most) n/2 leaves in the heap
- There are n/4 nodes at height 1
- There n/8 nodes at height 2
- …
- There is 1 node at height $\log_2 n$ (the root)
- The runtime of percolate_down for any node is at most $c \times (height\ of\ the\ node)$

- Worst case for heapify:
- At height h, at most $c \times h \times \frac{n}{2^{h+1}}$ operations
- $c \sum_{h=1}^{\log_2 n} h \times \frac{n}{2^{h+1}} = \frac{n}{2} \sum_{h=1}^{\log_2 n} \frac{h}{2^h}$
- $\sum_{h=1}^{\infty} \frac{h}{2^h} = 2$, so the total runtime is O(n)

Some algebra

- $S_n(t) = t + 2t^2 + 3t^3 + \cdots + nt^n$
- $tS_n(t) = t^2 + 2t^3 + 3t^4 + \cdots + nt^{n+1}$
- $(1-t)S_n(t) = t^2 + t^3 + t^4 + \cdots + t^n - nt^{n+1}$
- $S_n(t) = \frac{1}{1-t}\left(t\frac{1-t^n}{1-t} - nt^{n+1}\right)$
- Substitute $t = \frac{1}{2}$:
- $S_n(t) = 2\left(1 - \left(\frac{1}{2}\right)^n - n1/2^{n+1}\right) \rightarrow_{n\rightarrow\infty} 2$

Heapsort

- Create a heap from a list using heapify (O(n))
- Call extract_min n times, and place the elements back into the original list in order
- Total time: O(n + nlog(n)) = O(nlog(n))