Priority Queue (Heap)

(chapter 9 in our book)

- A kind of queue
- Dequeue gets element with the highest priority
- Priority is based on a comparable value (key) of each object (smaller value higher priority, or higher value higher priority)
- Example Applications:
  - printer **->** print (dequeue) the shortest document first
  - operating system **->** run (dequeue) the shortest job first
  - normal queue **->** dequeue the first enqueued element first

Priority Queue (Heap) Operations

- insert (enqueue)
- deleteMin (dequeue)
  - smaller value higher priority
  - Find / save the minimum element, delete it from structure and return it

Implementation using Linked List

- Unsorted linked list
  - insert takes O(1) time
  - deleteMin takes O(N) time
- Sorted linked list
  - insert takes O(N) time
  - deleteMin takes O(1) time

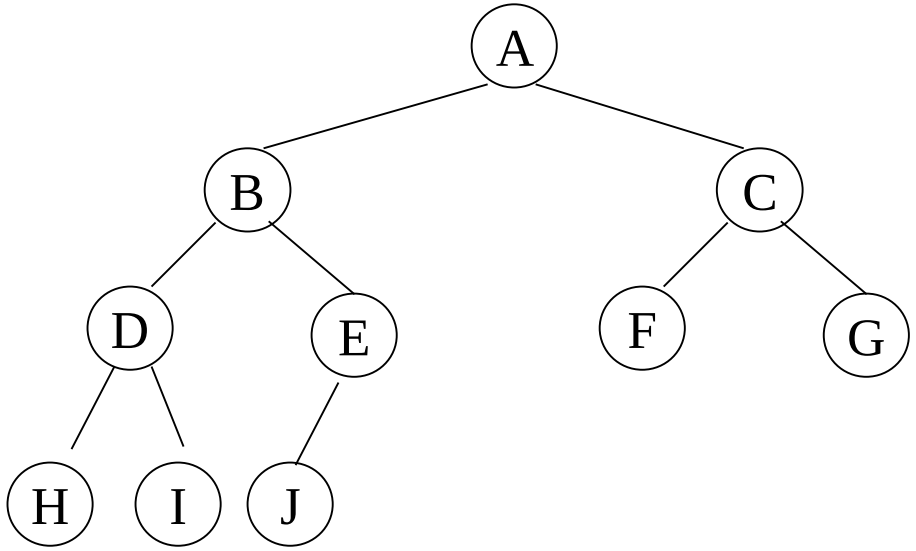
Implementation using Binary Search Tree

- insert takes O(log N) time
- deleteMin takes O(log N) time

- support other operations that are not required by priority queue (for example, findMax)
- deleteMin operations make the tree unbalanced

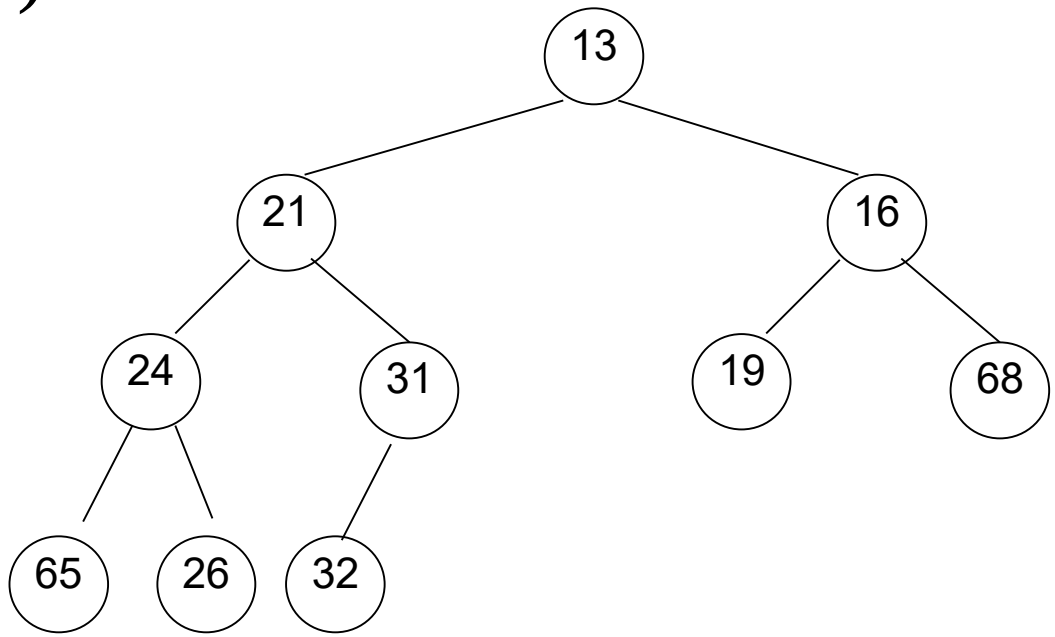
Terminology: complete tree

- completely filled (bottom level is filled from left to right
- size between 2^h (bottom level has only one node) and $2^{h+1}-1$



Heap: Order Property (for Minimum Heap)

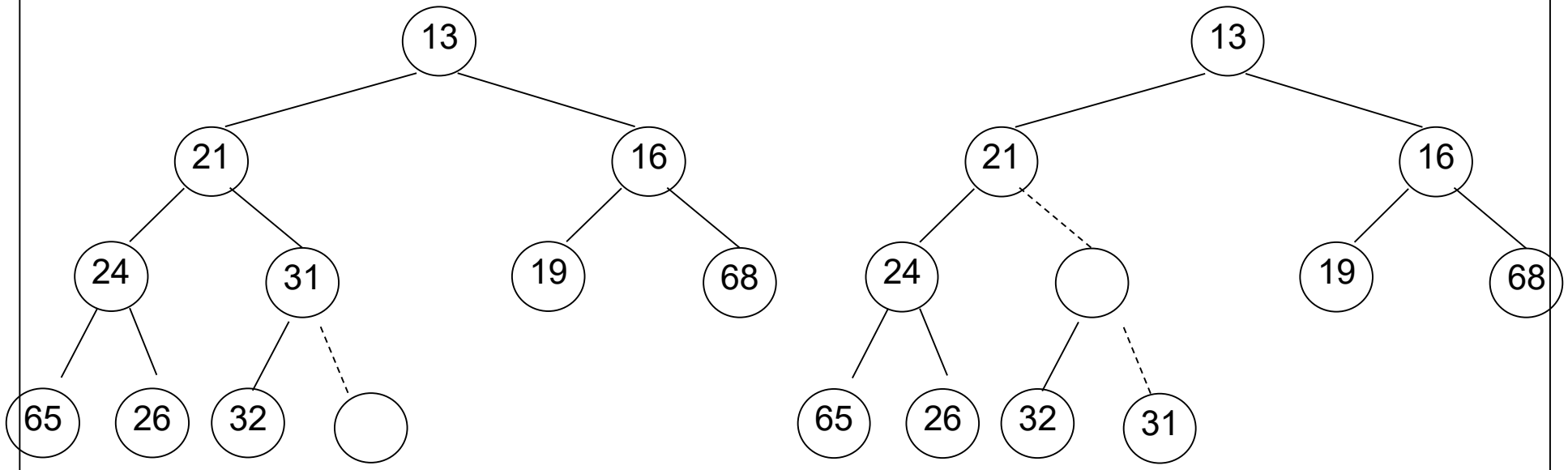
- Any node is smaller than (or equal to) all of its children (any subtree is a heap)
- Smallest element is at the root (findMin takes O(1) time)



Source: Muangsin / Weiss

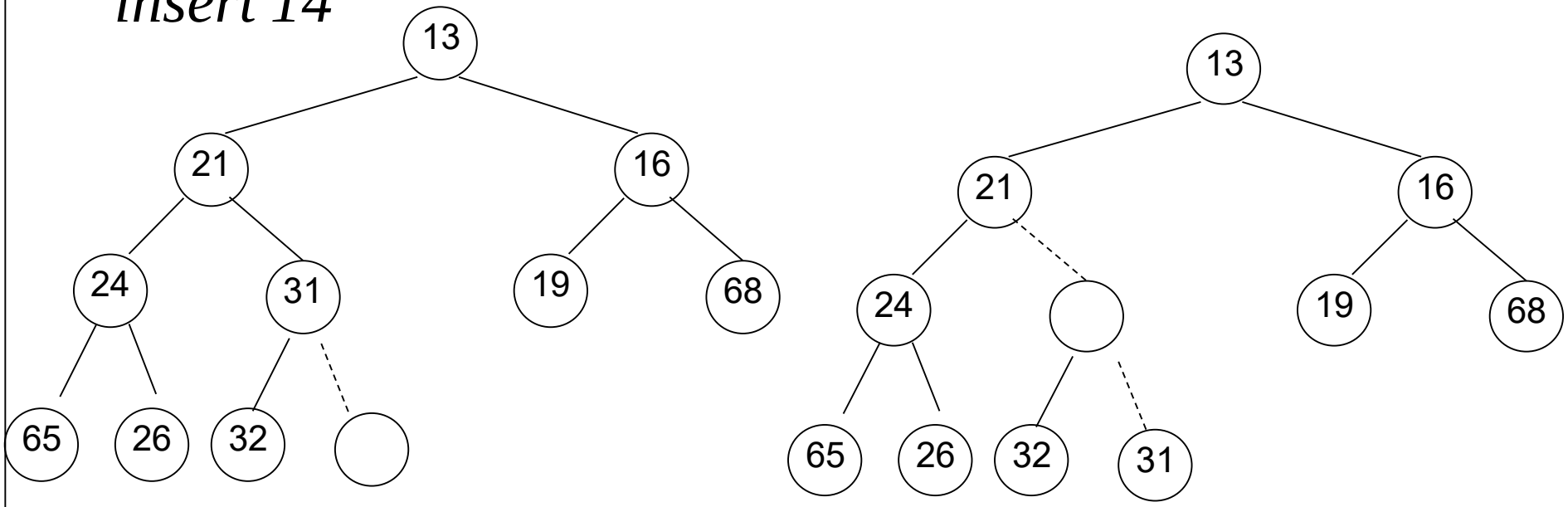
Insert

- Create a hole in the next available location
- Move the hole up (swap with its parent) until data can be placed in the hole without violating the heap order property (called ***percolate up***)



Insert

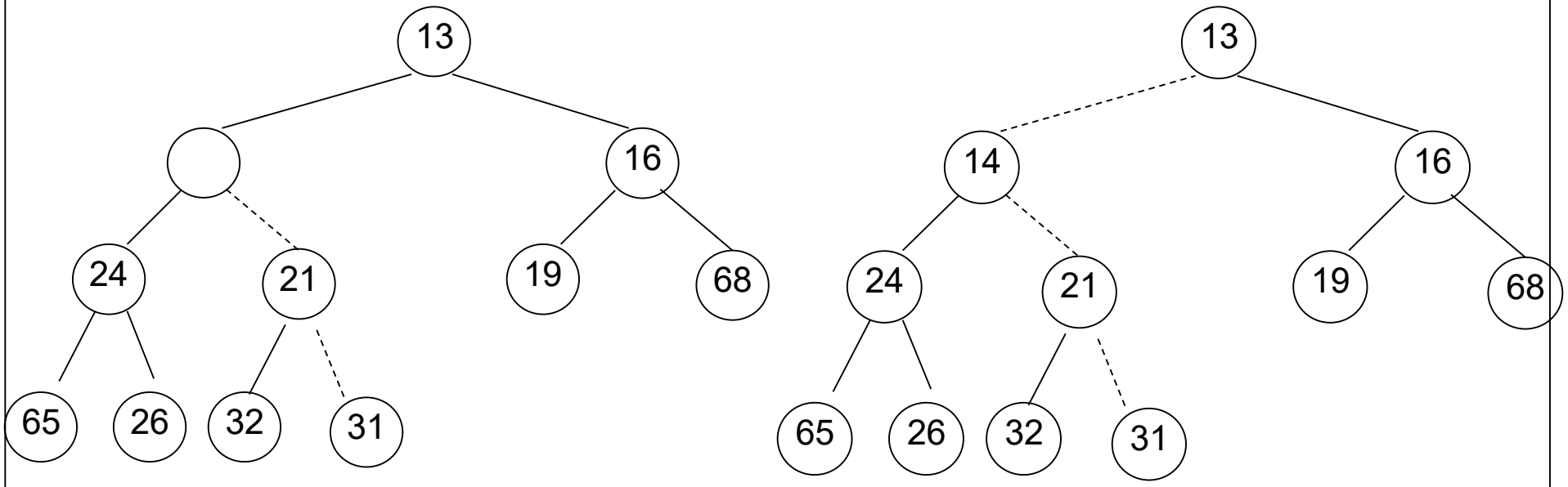
insert 14



Percolate Up -> move the place to put *14* up

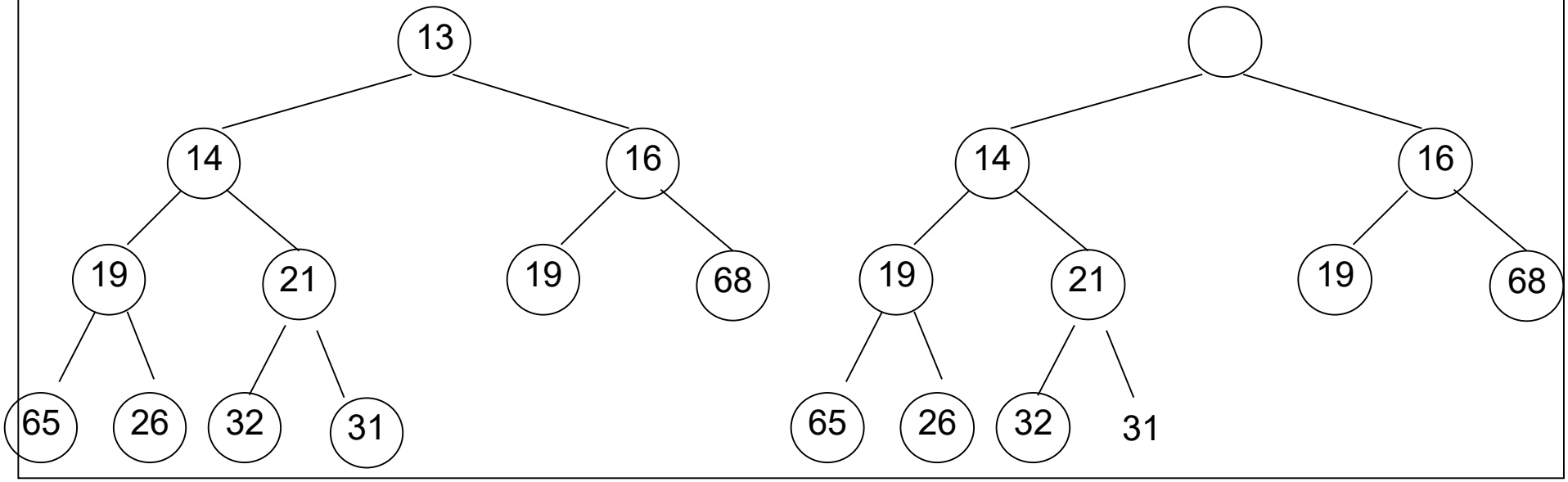
(move its parent down) until its parent <= 14

Insert

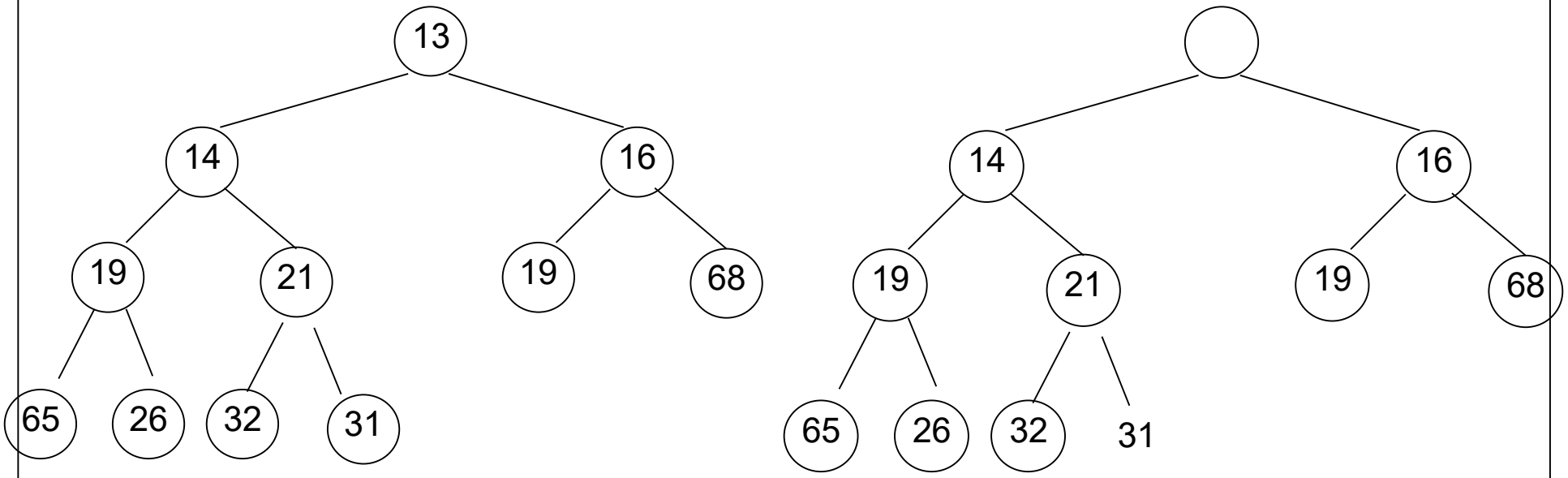


deleteMin

- Create a hole at the root
- Move the hole down until the last element of the heap can be placed in the hole without violating the heap order property (called ***percolate down***)

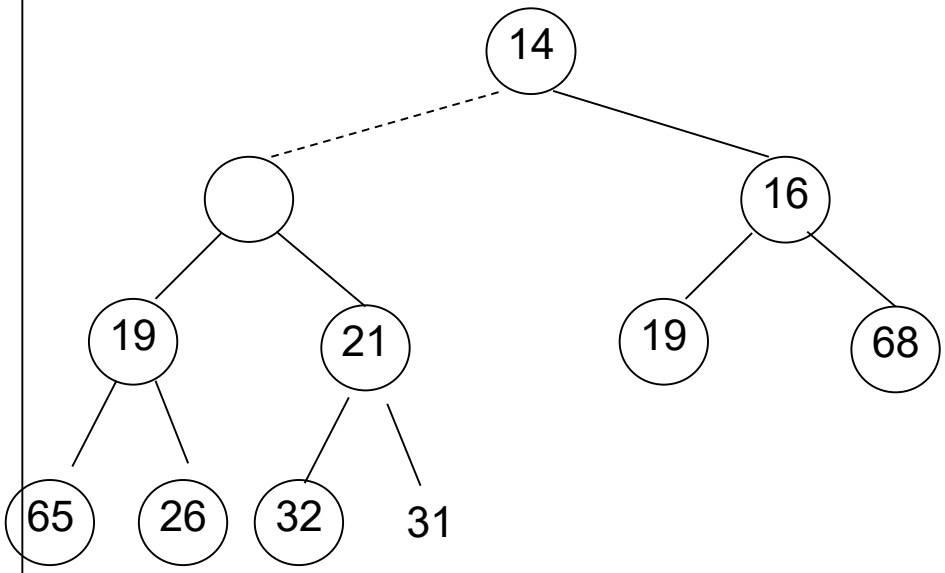


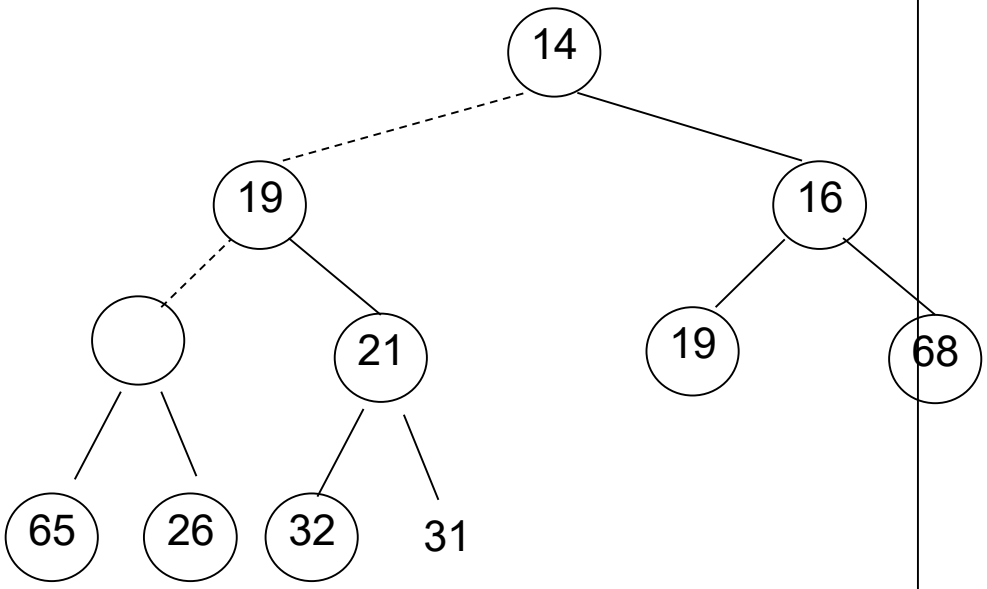
deleteMin



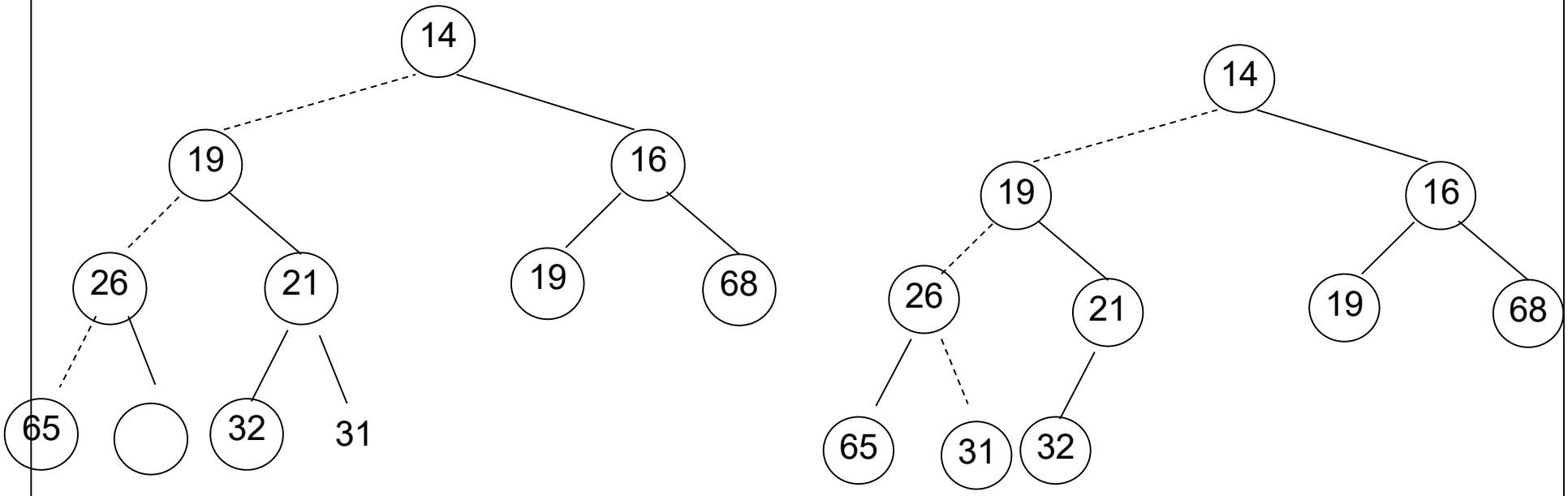
Percolate Down -> move the place to put *31* down (move its smaller child up) until its children >= 31

deleteMin





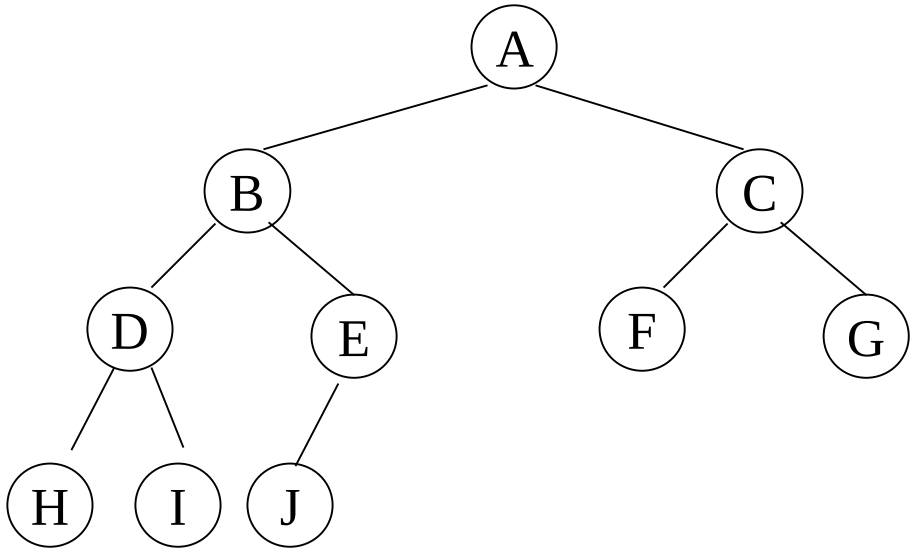
deleteMin



Running Time

- *insert*
  - worst case: takes O(log N) time, moves an element from the bottom to the top
  - on average: takes a constant time (2.607 comparisons), moves an element up 1.607 levels
- deleteMin
  - worst case: takes O(log N) time
  - on average: takes O(log N) time (element that is placed at the root is large, so it is percolated almost to the bottom)

Array Implementation of Binary Heap



left child is in position 2i

right child is in position (2i+1)

parent is in position i/2

	A	B	C	D	E	F	G	H	I	J			
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Visualization

https://www.cs.usfca.edu/~galles/JavascriptVisual/Heap.html