UNIVERSITY OF CALIFORNIA
UC RIVERSIDE

Heaps

Chapter 6

Objectives

- Understand what a priority queue is
- Identify the applications where a priority queue can be used
- Analyze the running time and space overhead of the heap data structure
- Use heaps to solve the top-k problem

Top-K

- Given n numbers, find the k largest numbers
- k=1
- k=2
- k=n/100 (Top 1%)

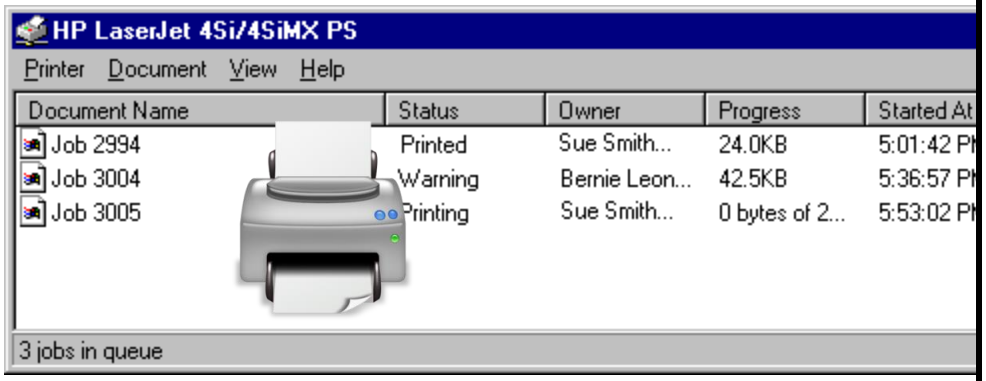
- Sort and select
- Use a BST (or AVL tree) to keep track of the top-k items
- Is there something more efficient?

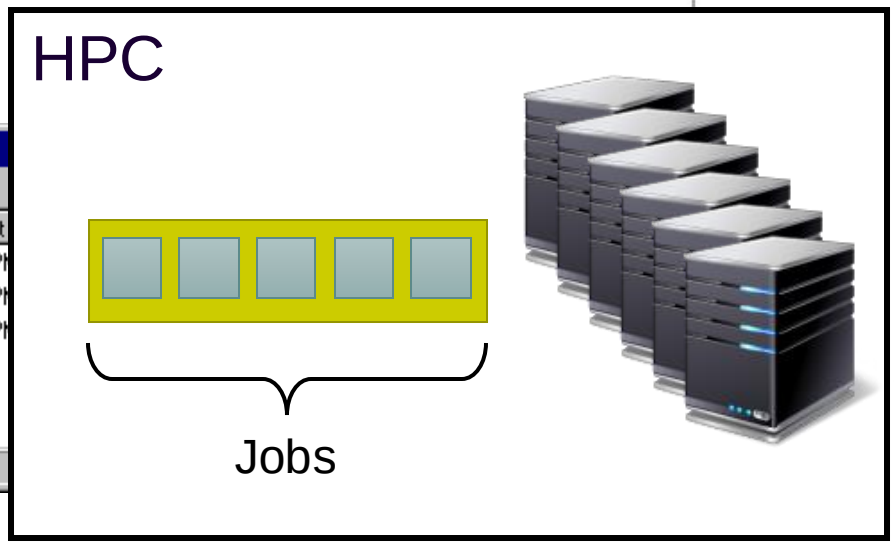
Top-K

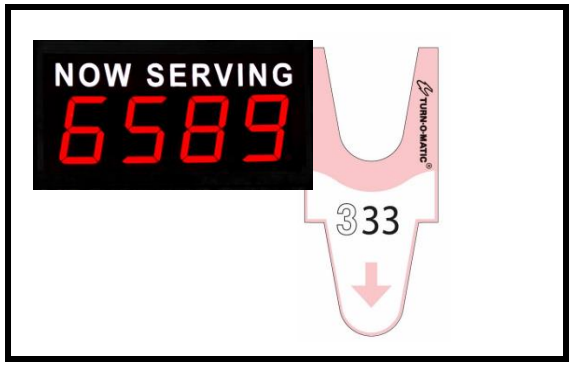
- Given n numbers, find the k largest numbers
- k=1
- k=2
- k=n/100 (Top 1%)

- Sort and select O(n log n)
- Use a BST (or AVL tree) to keep track of the top-k items O(n log k)
- Is there something more efficient?

Regular Queues

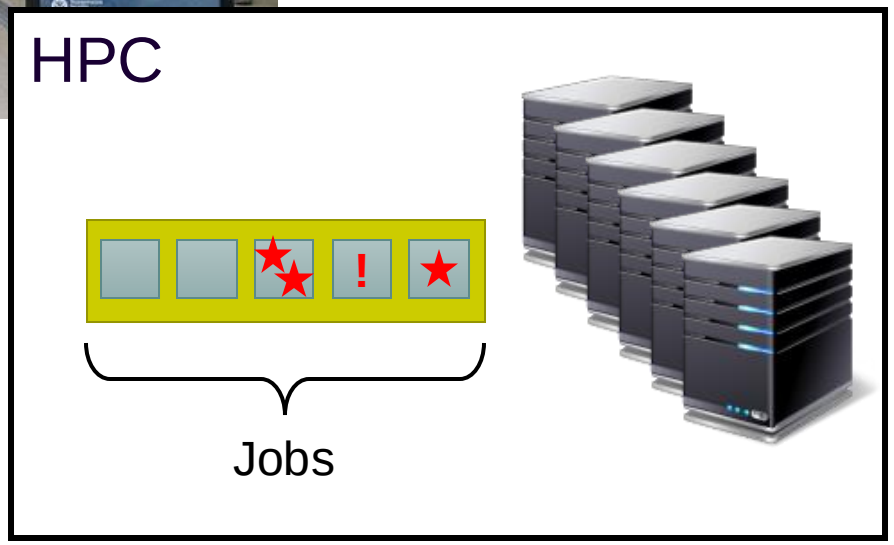






Priority Queue

- Airport check-in



Priority Queue ADT

- Queue
  - Enqueue
  - Dequeue
  - Front
  - Empty?
- Priority Queue
  - Insert
  - DeleteMin/Max
  - PeekMin/Max
  - Empty?
  - IncreaseKey
  - DecreaseKey
  - Remove

Priority Queue Implementation

- Straight-forward implementations?

- Unsorted list
- Sorted
- BST/AV

	Unsorted list	Sorted list	BST/AVL
Insert			
PeekMin			
DeleteMin			
IncreaseKey			
DecreaseKey			

Priority Queue Implementation

- Straight-forward implementations

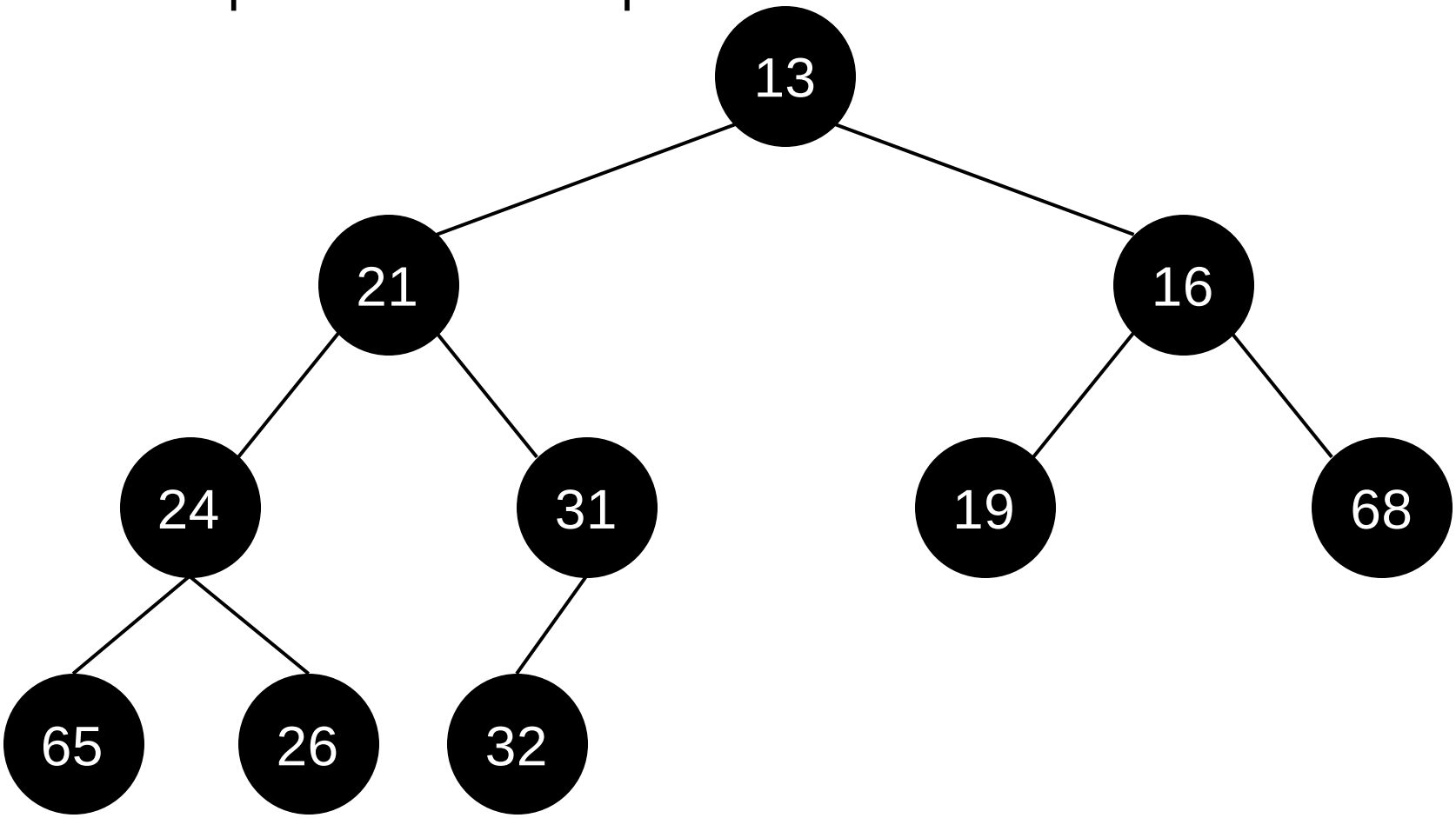
- Unsorted list
- Sorted
- BST/AV

	Unsorted list	Sorted list	BST/AVL
Insert	O(1)	O(n)	O(log n)
PeekMin	O(n)	O(1)	O(log n)
DeleteMin	O(n)	O(1)	O(log n)
IncreaseKey	O(1)*	O(n)	O(log n)
DecreaseKey	O(1)*	O(n)	O(log n)

* Assuming the record is already located

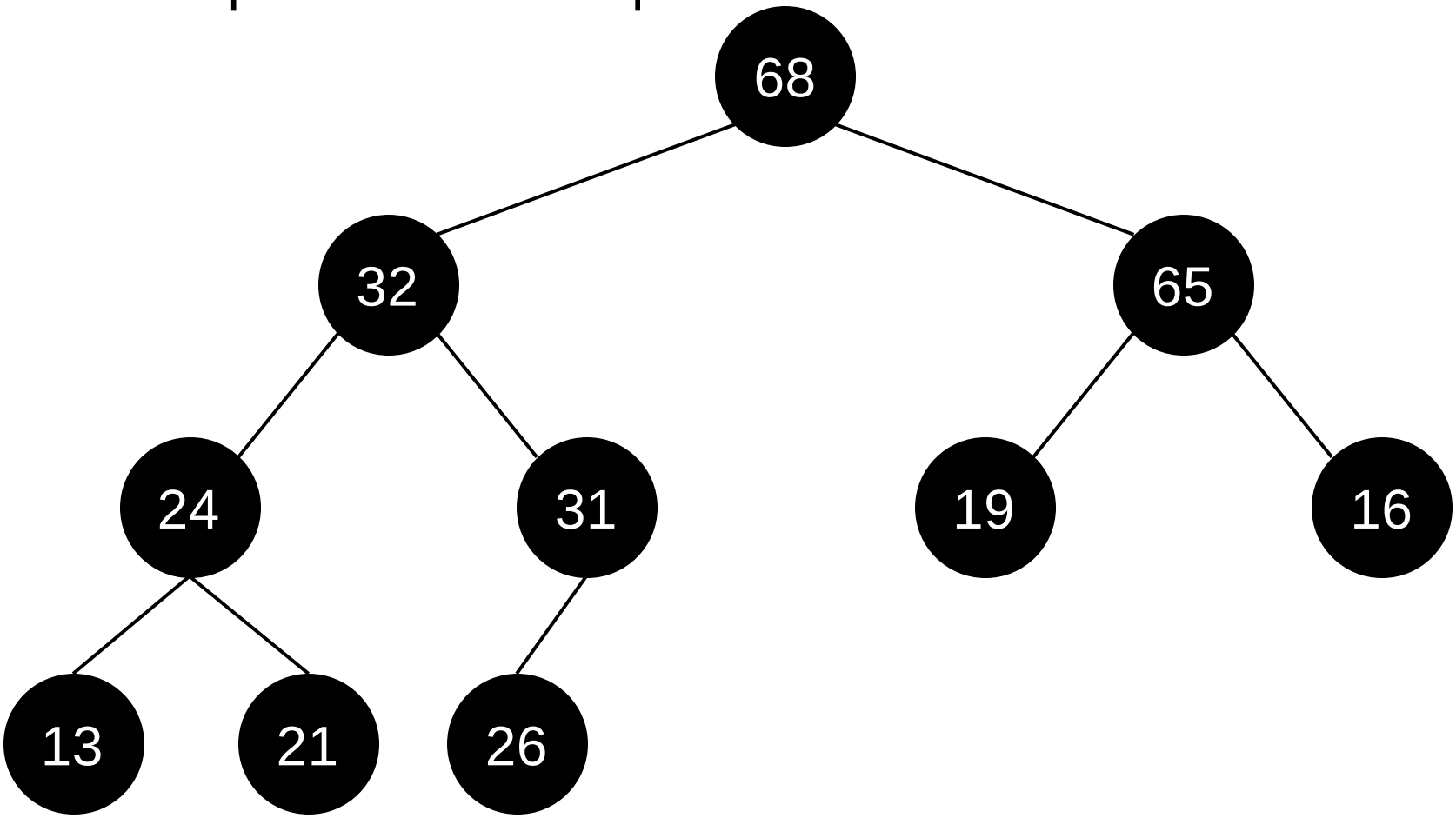
Heap Implementation

Example of a MinHeap



Heap Implementation

Example of a MaxHeap

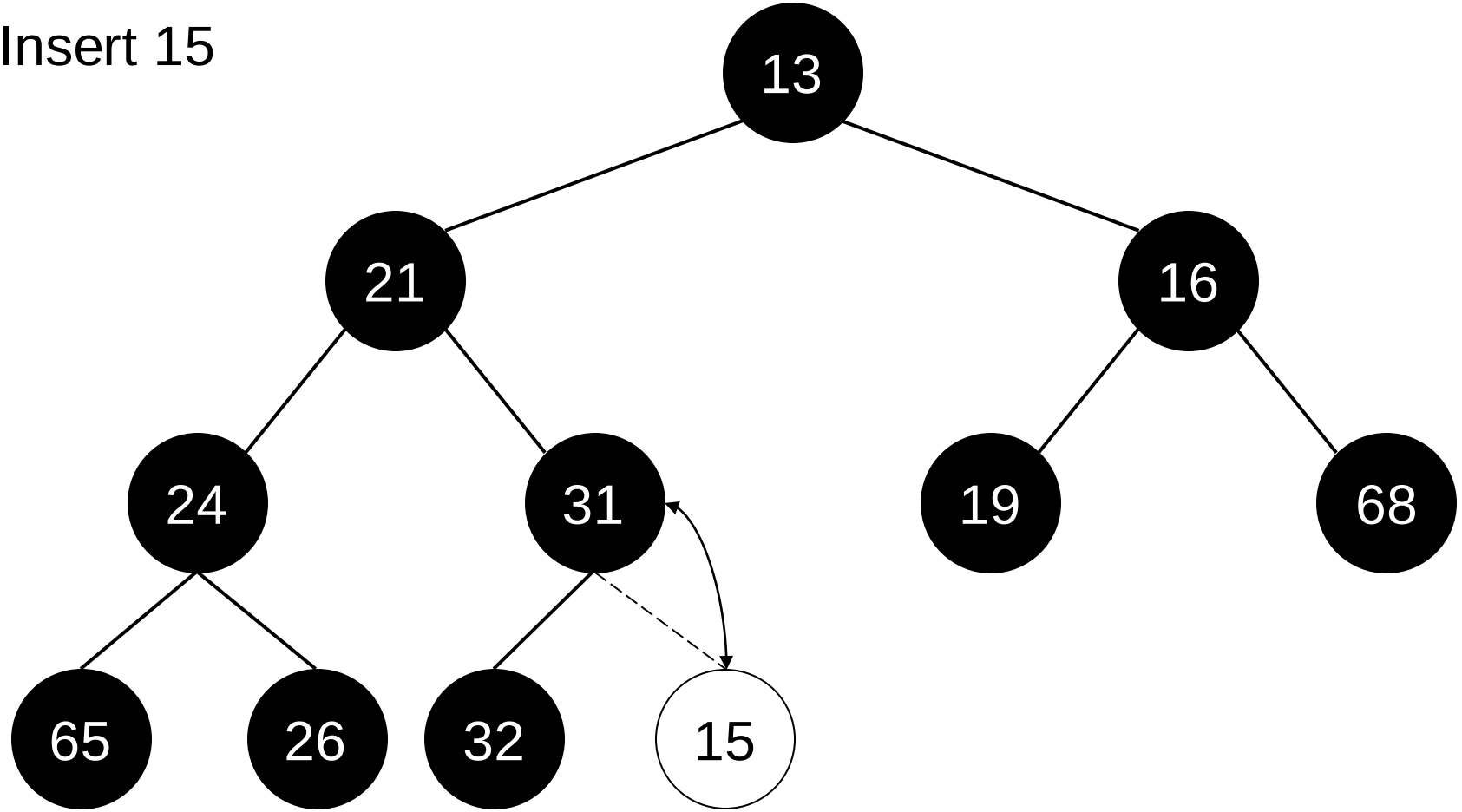


Heap Properties

- A complete binary tree
- For any internal node **n**
  - **n**.Key ≤ **n**.child.Key (Min Heap)
  - **n**.Key ≥ **n**.child.Key (Max Heap)

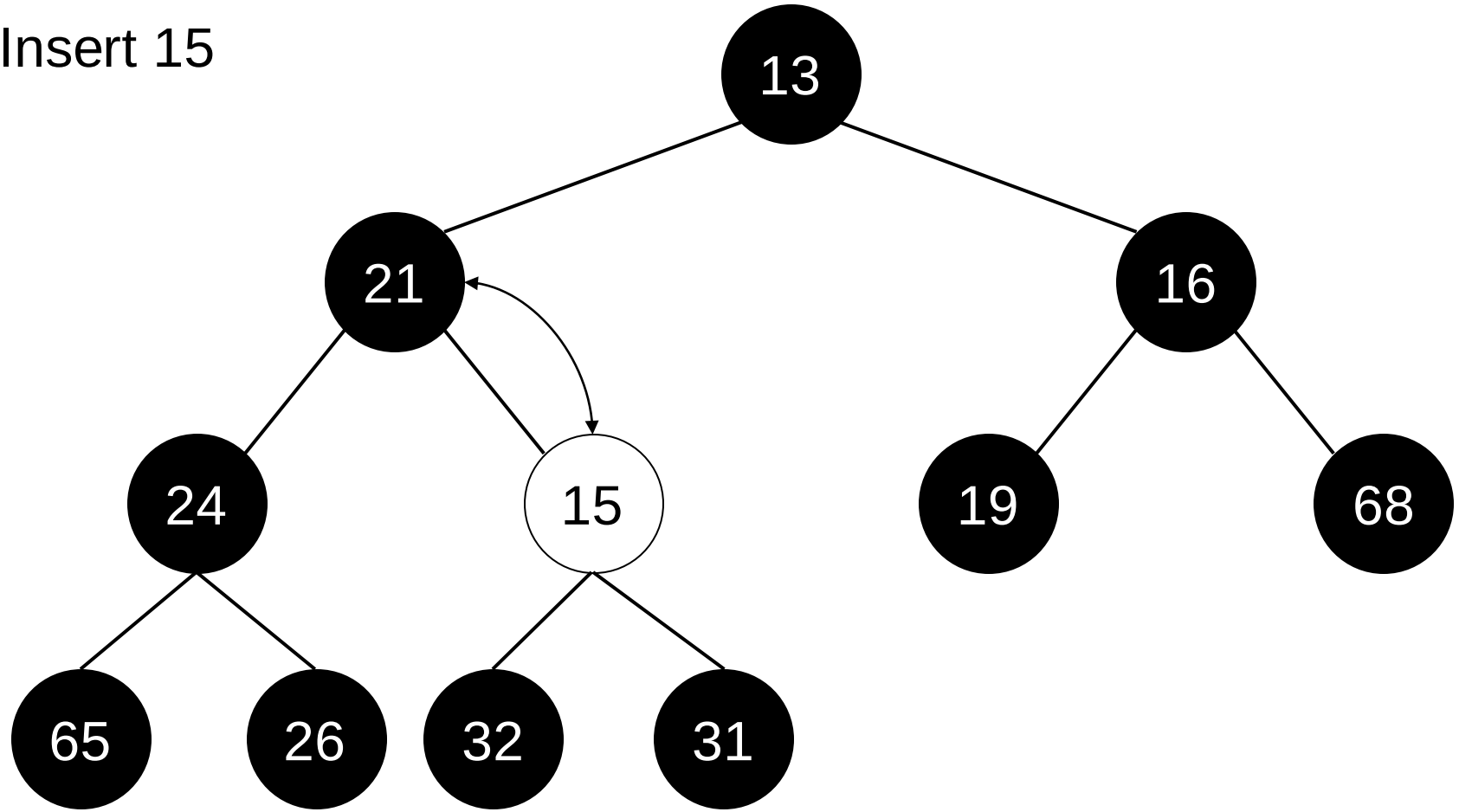
MinHeap Insertion

Insert 15



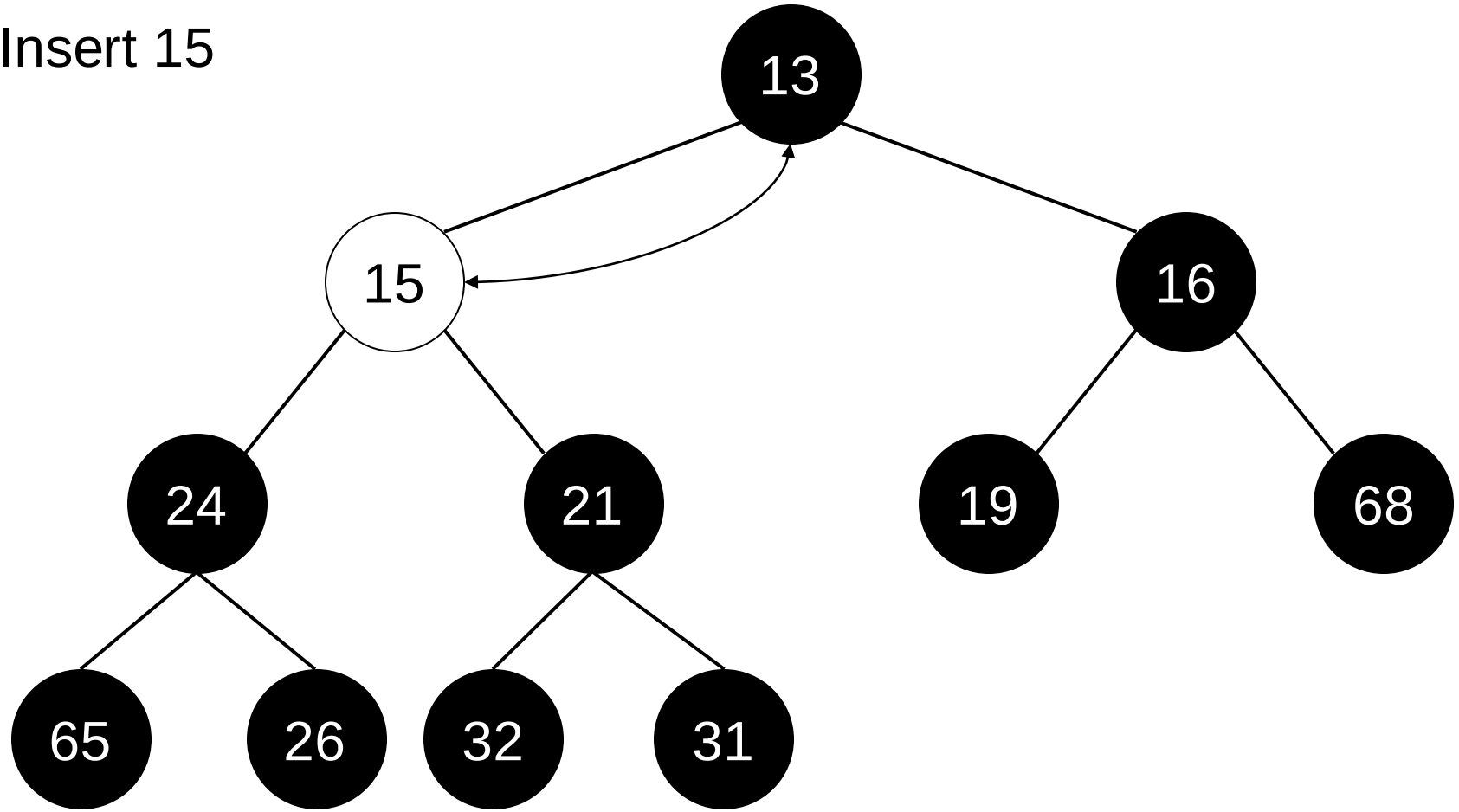
MinHeap Insertion

Insert 15



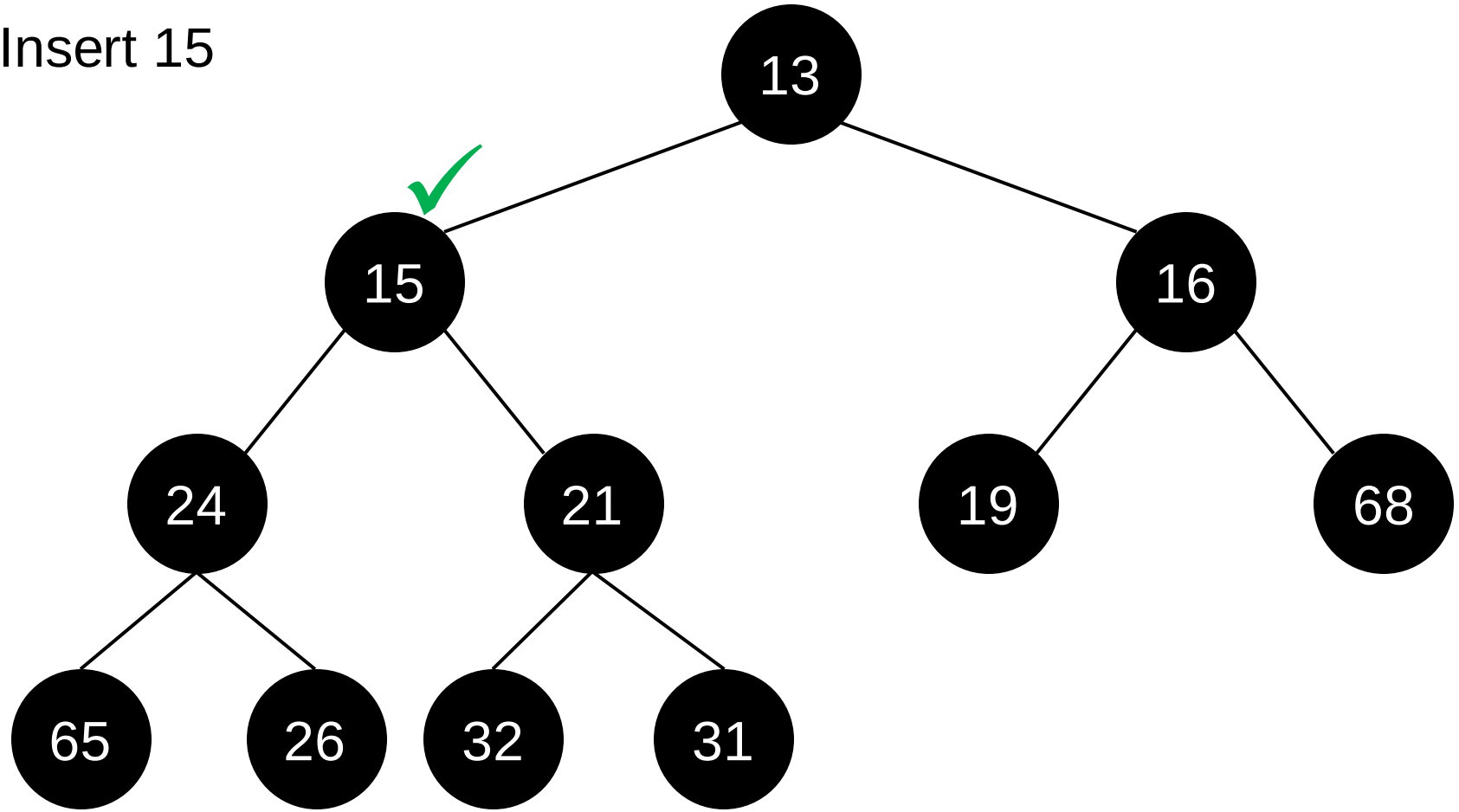
MinHeap Insertion

Insert 15



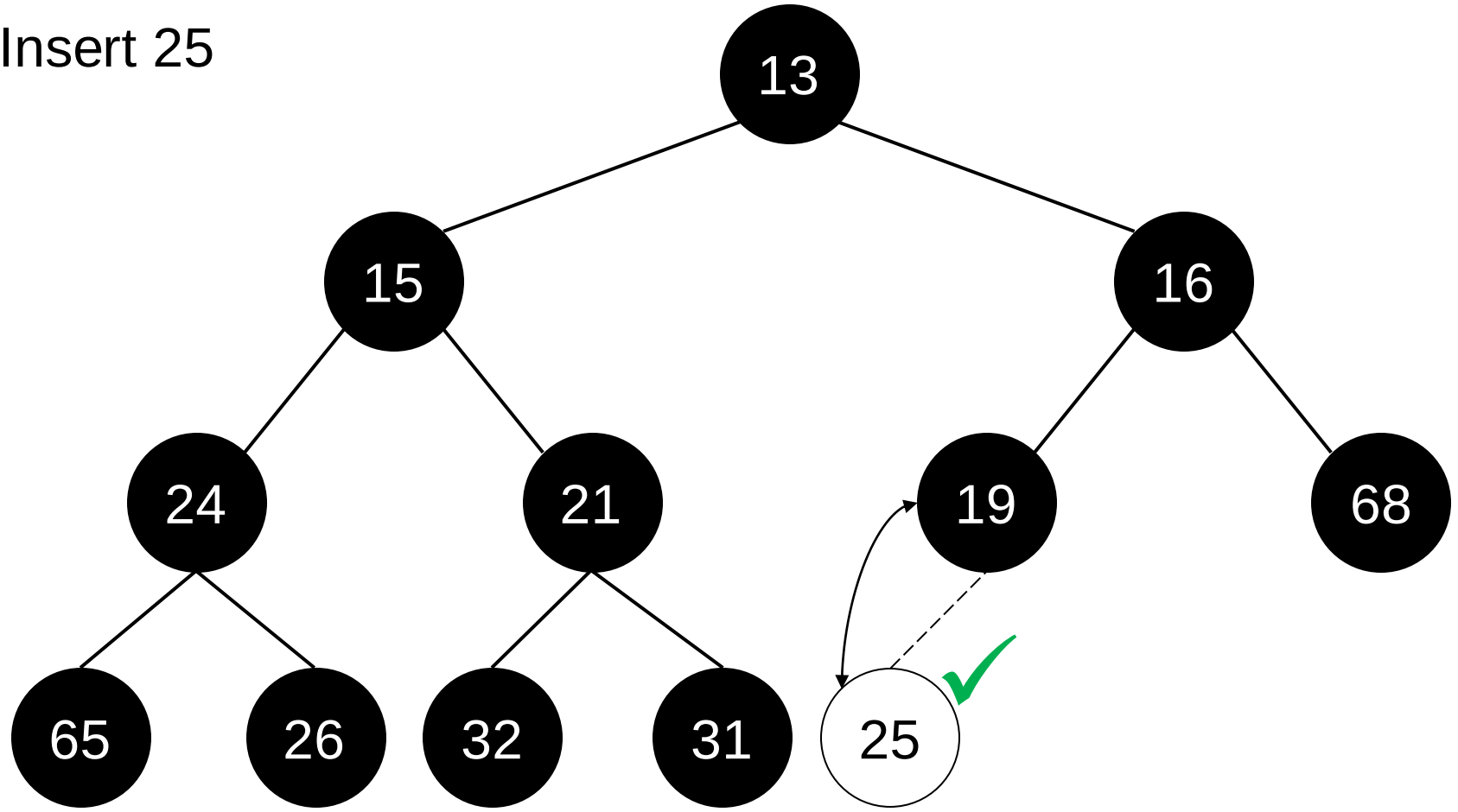
MinHeap Insertion

Insert 15



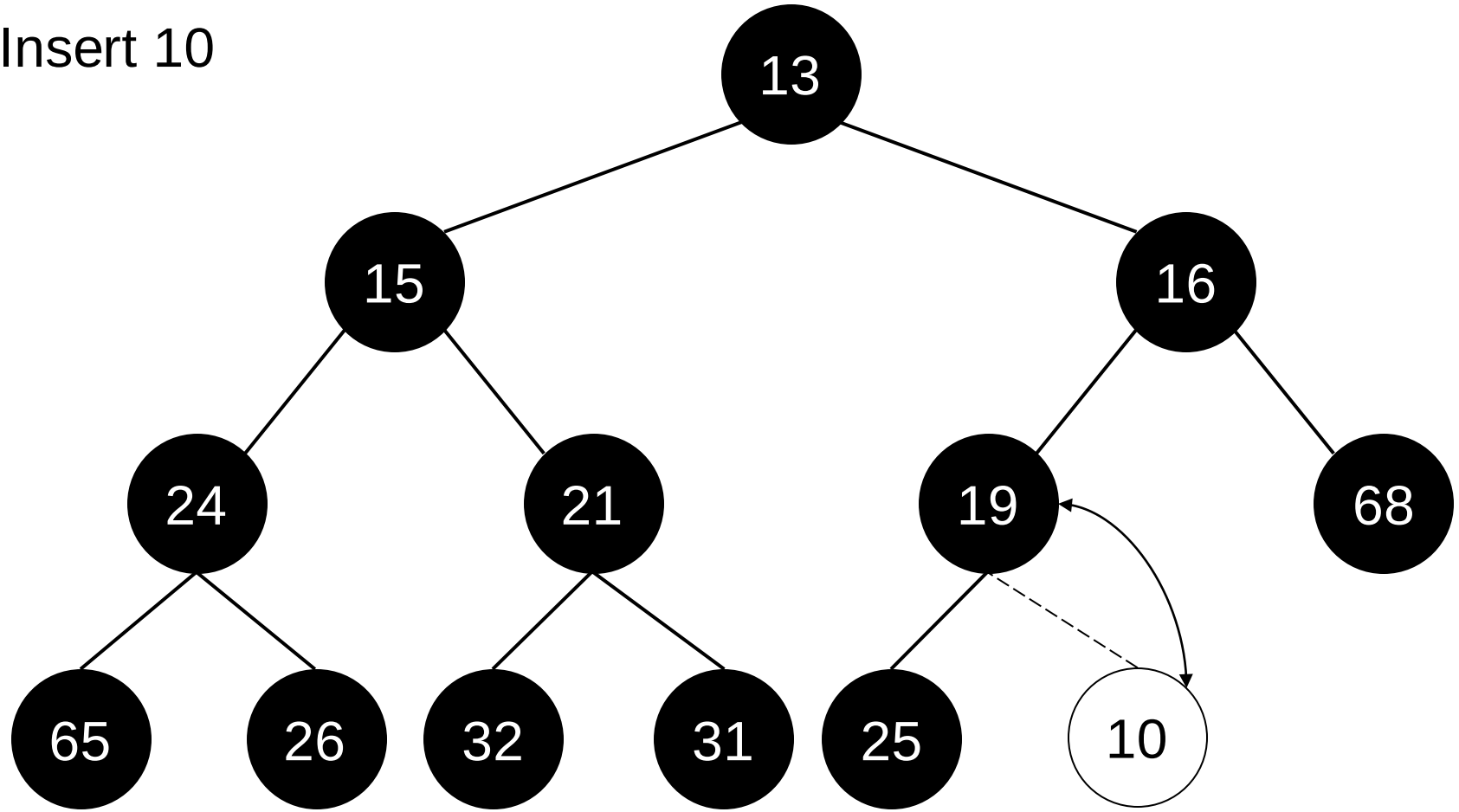
MinHeap Insertion

Insert 25



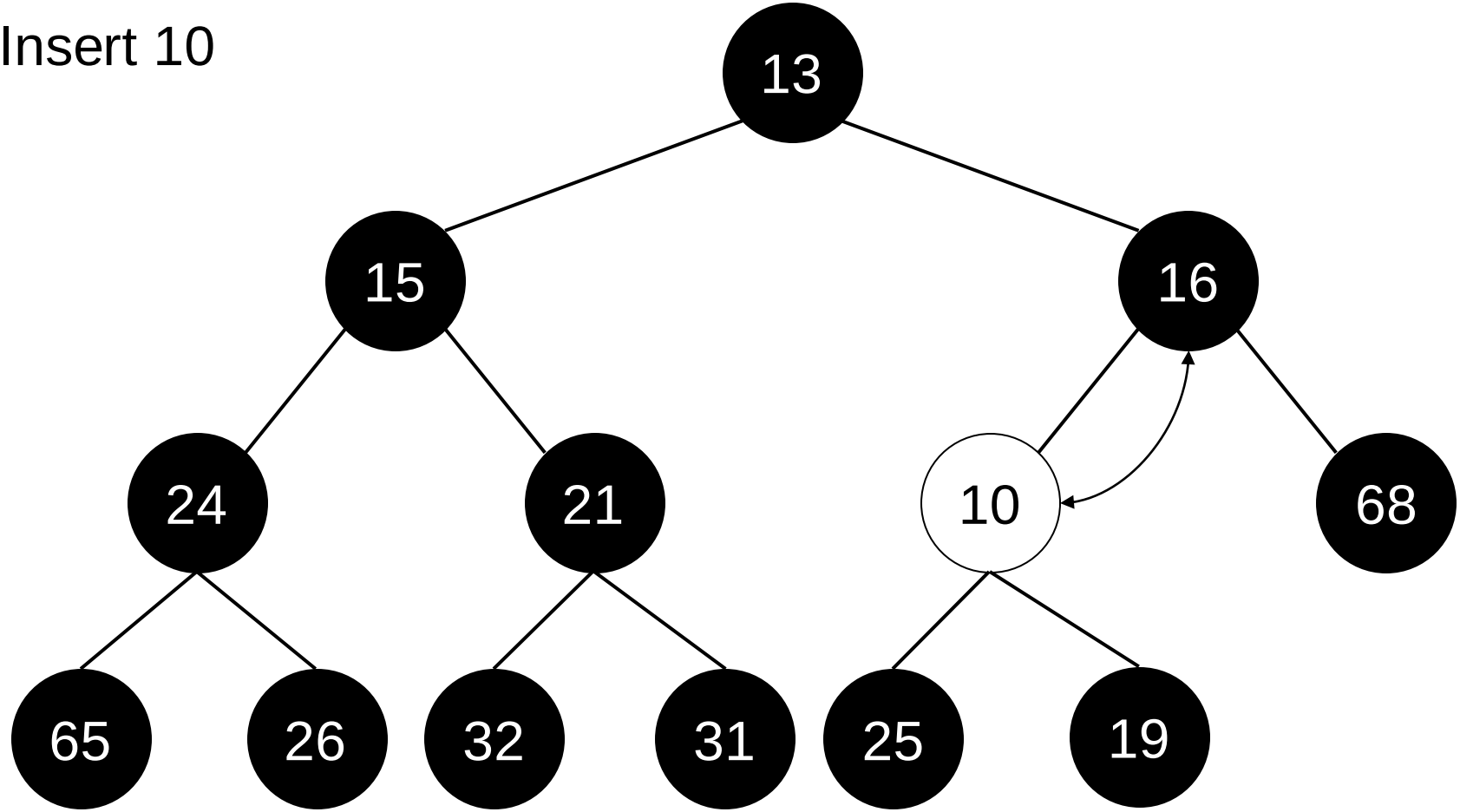
MinHeap Insertion

Insert 10



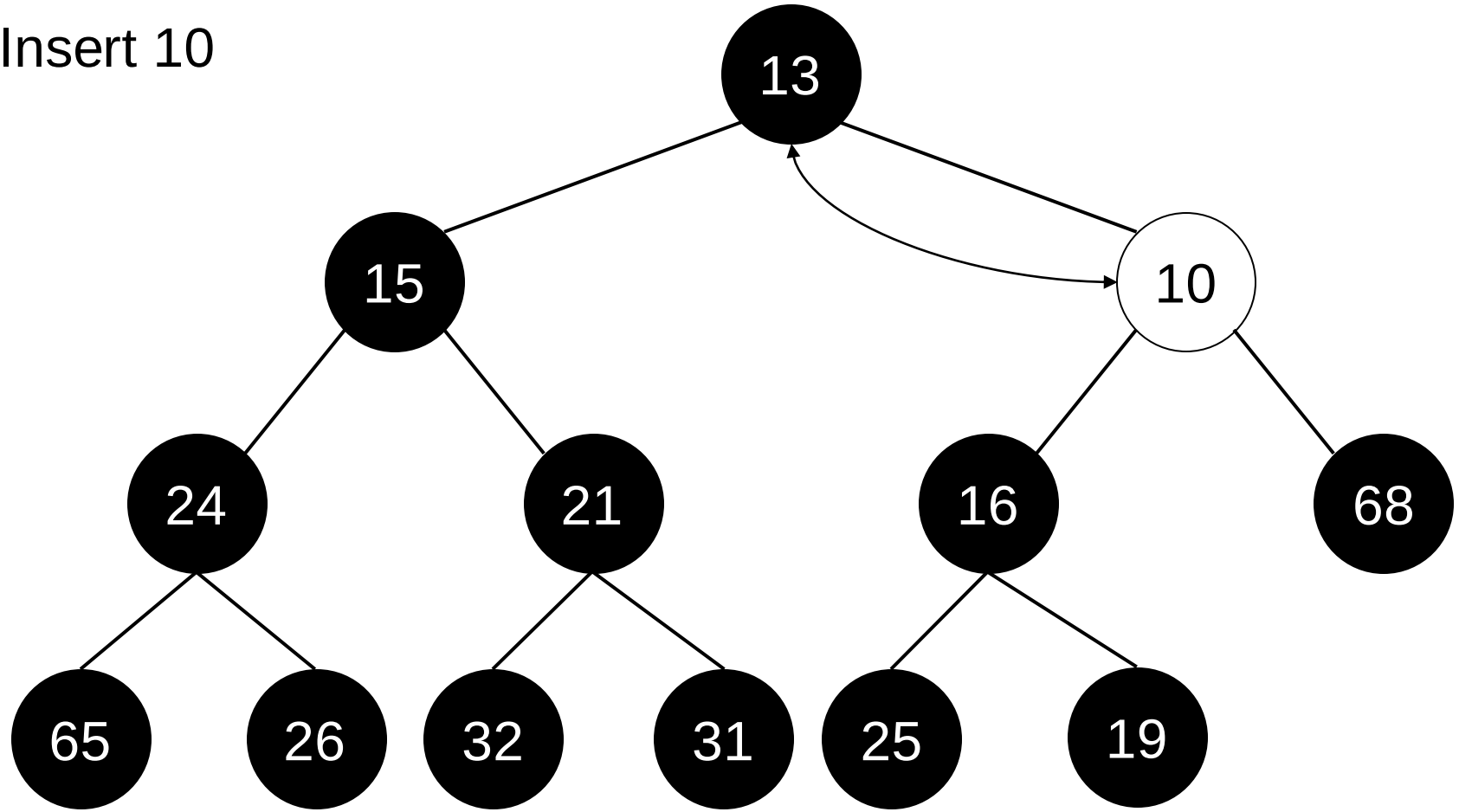
MinHeap Insertion

Insert 10



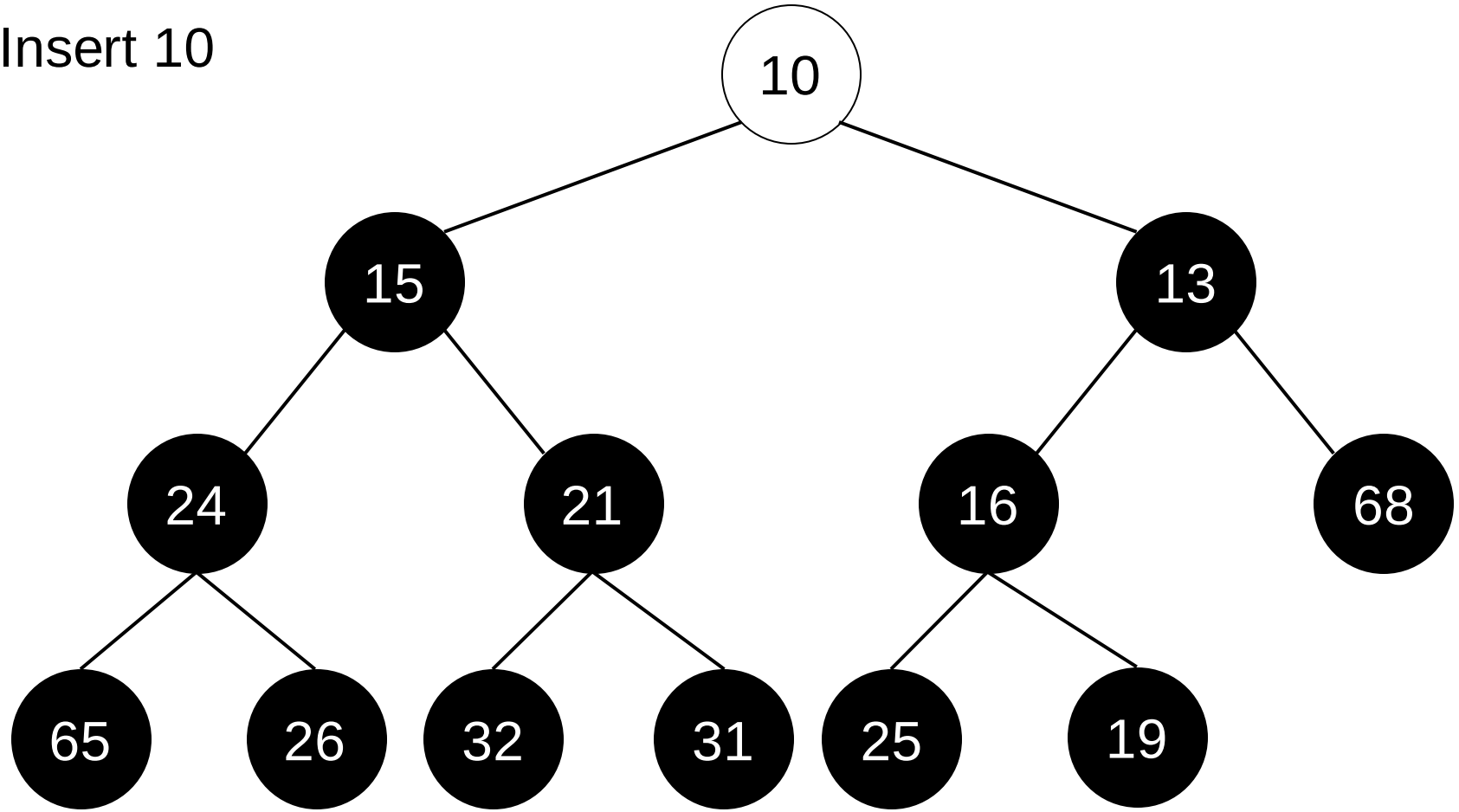
MinHeap Insertion

Insert 10



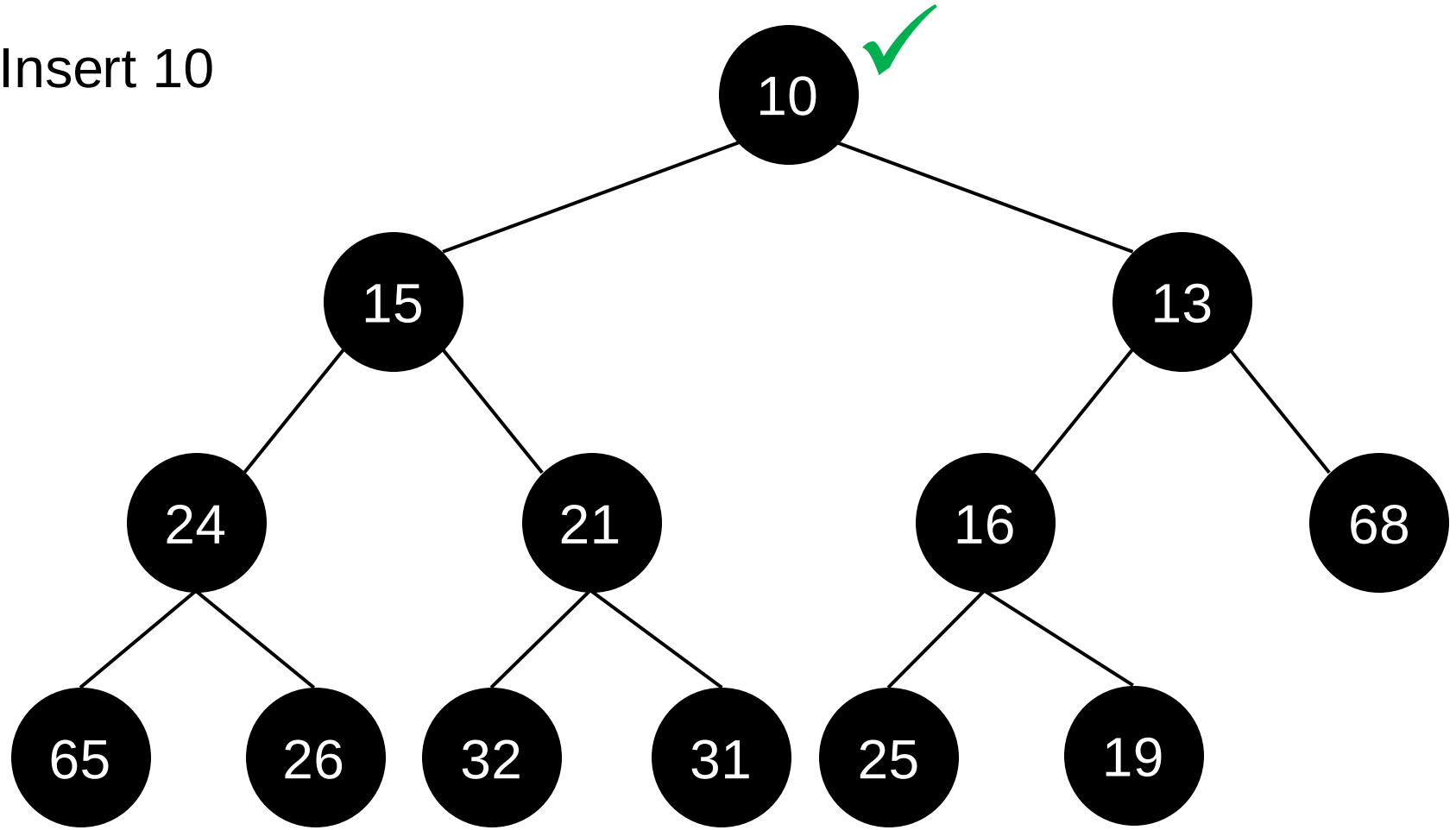
MinHeap Insertion

Insert 10



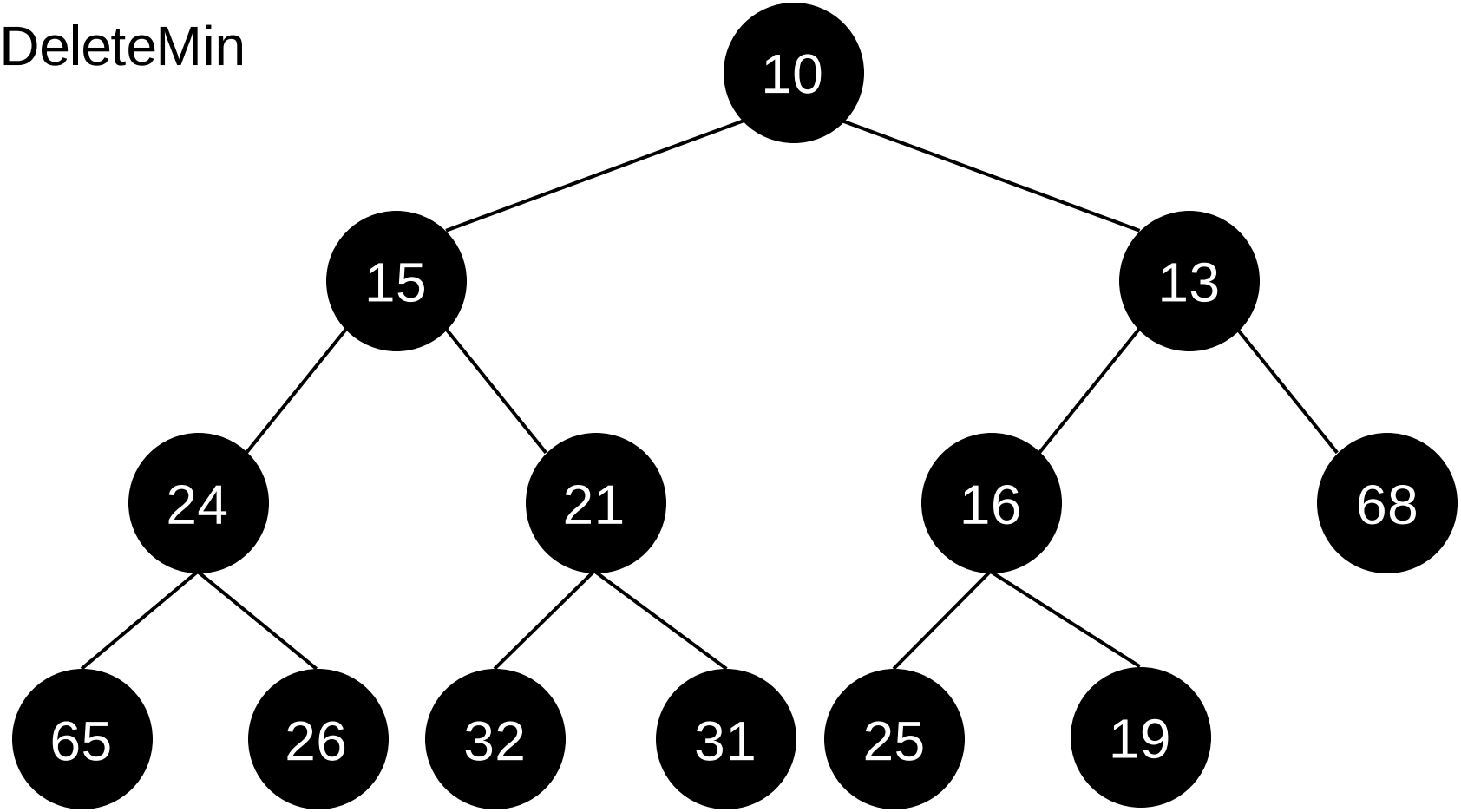
MinHeap Insertion

Insert 10



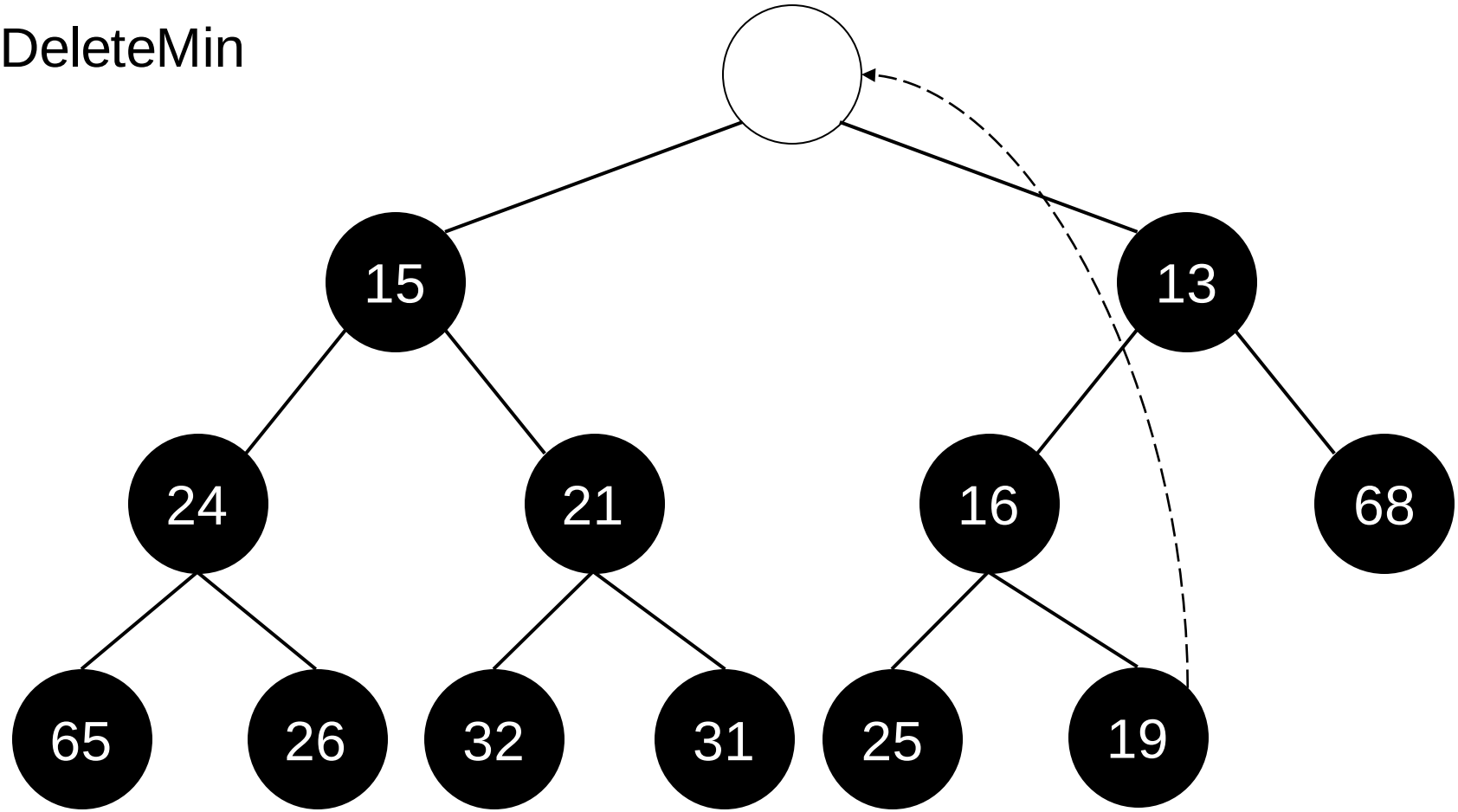
MinHeap Deletion

DeleteMin



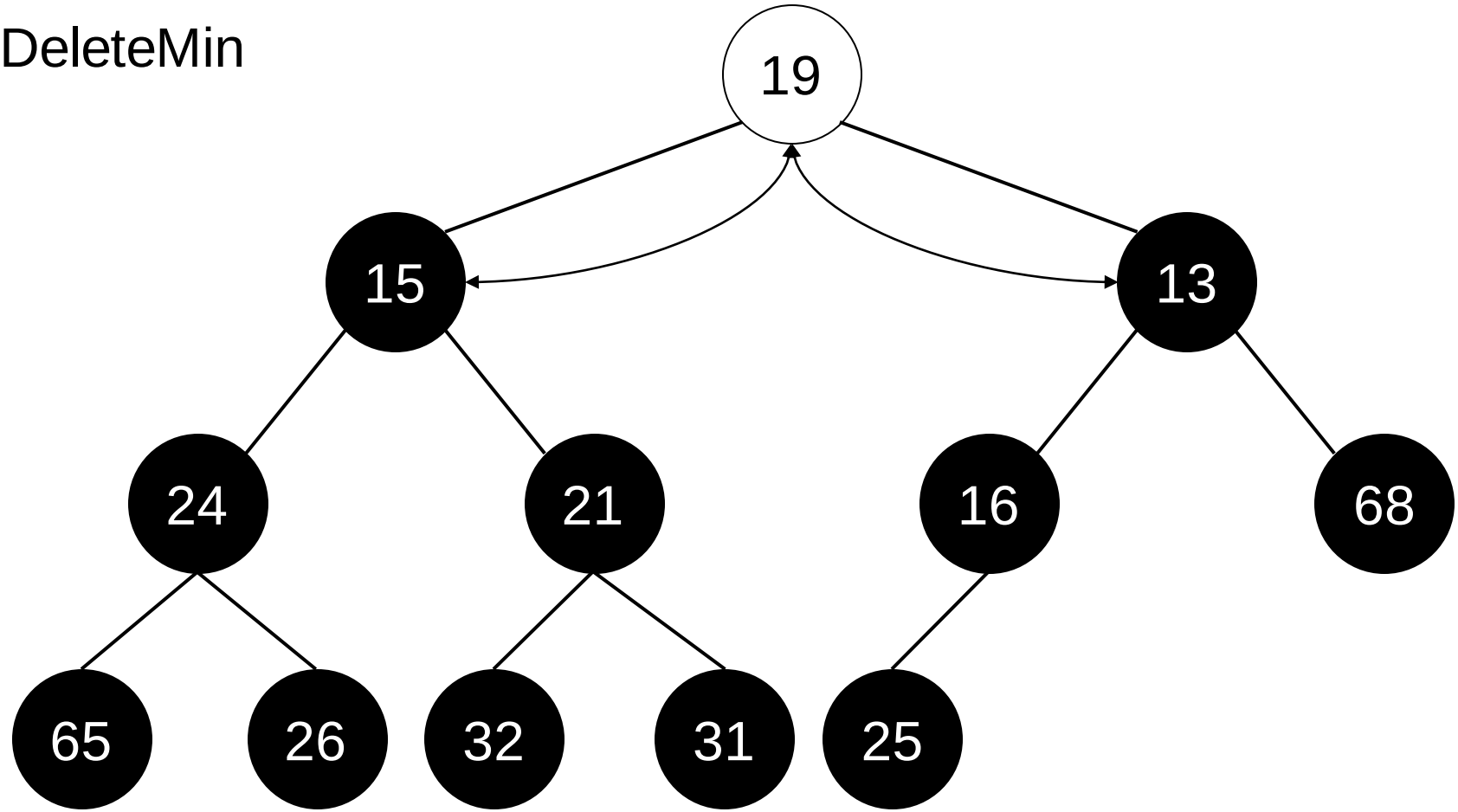
MinHeap Deletion

DeleteMin



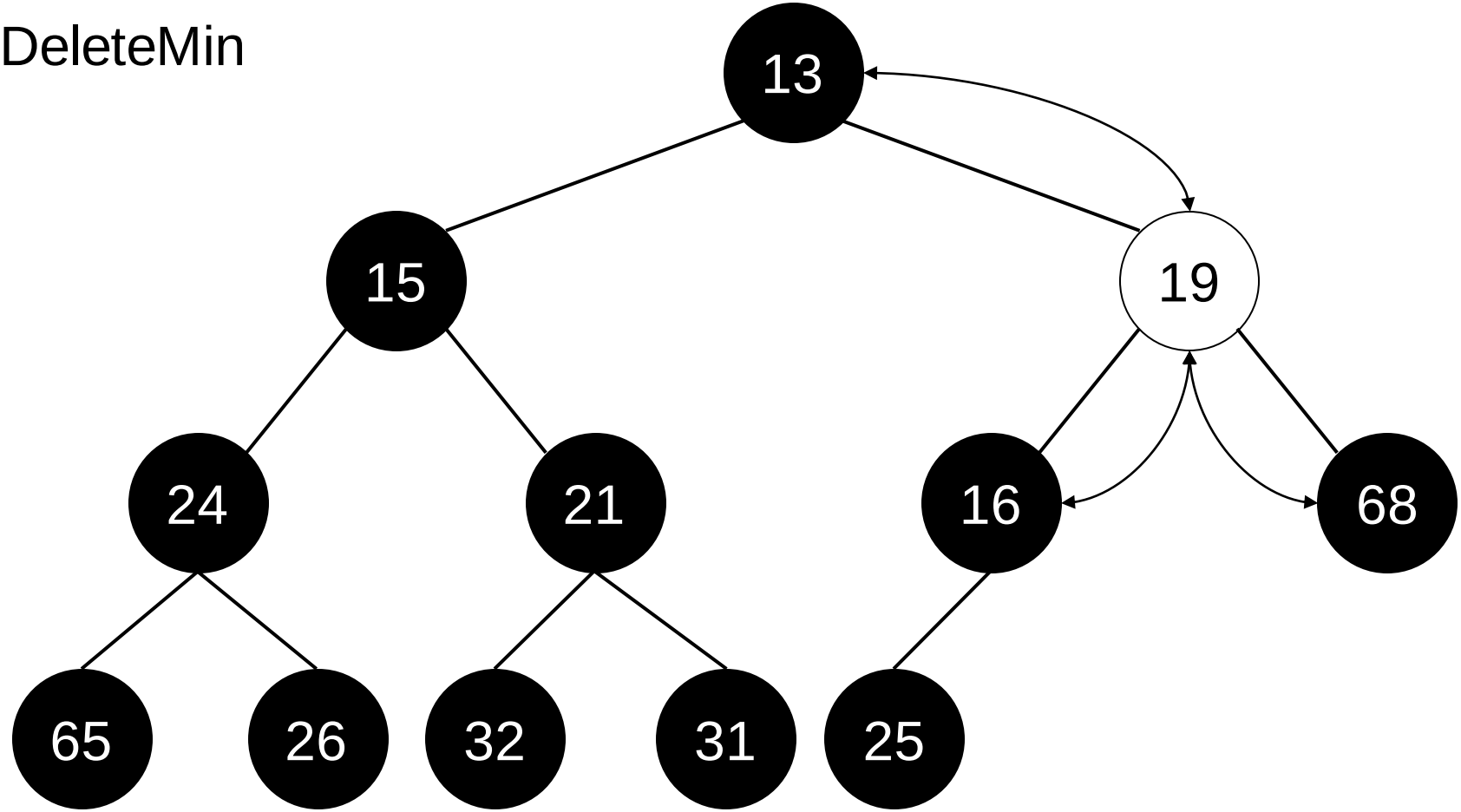
MinHeap Deletion

DeleteMin



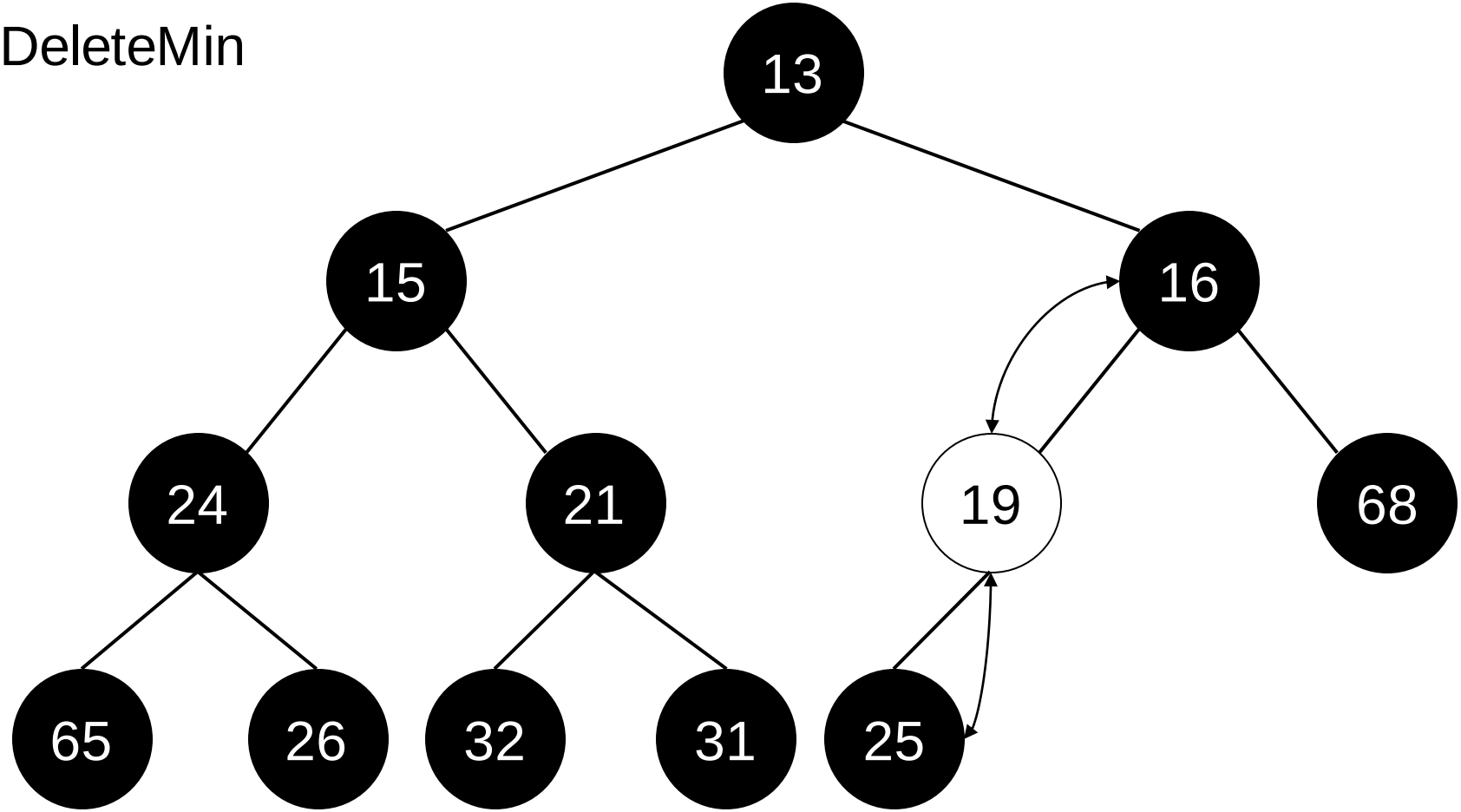
MinHeap Deletion

DeleteMin



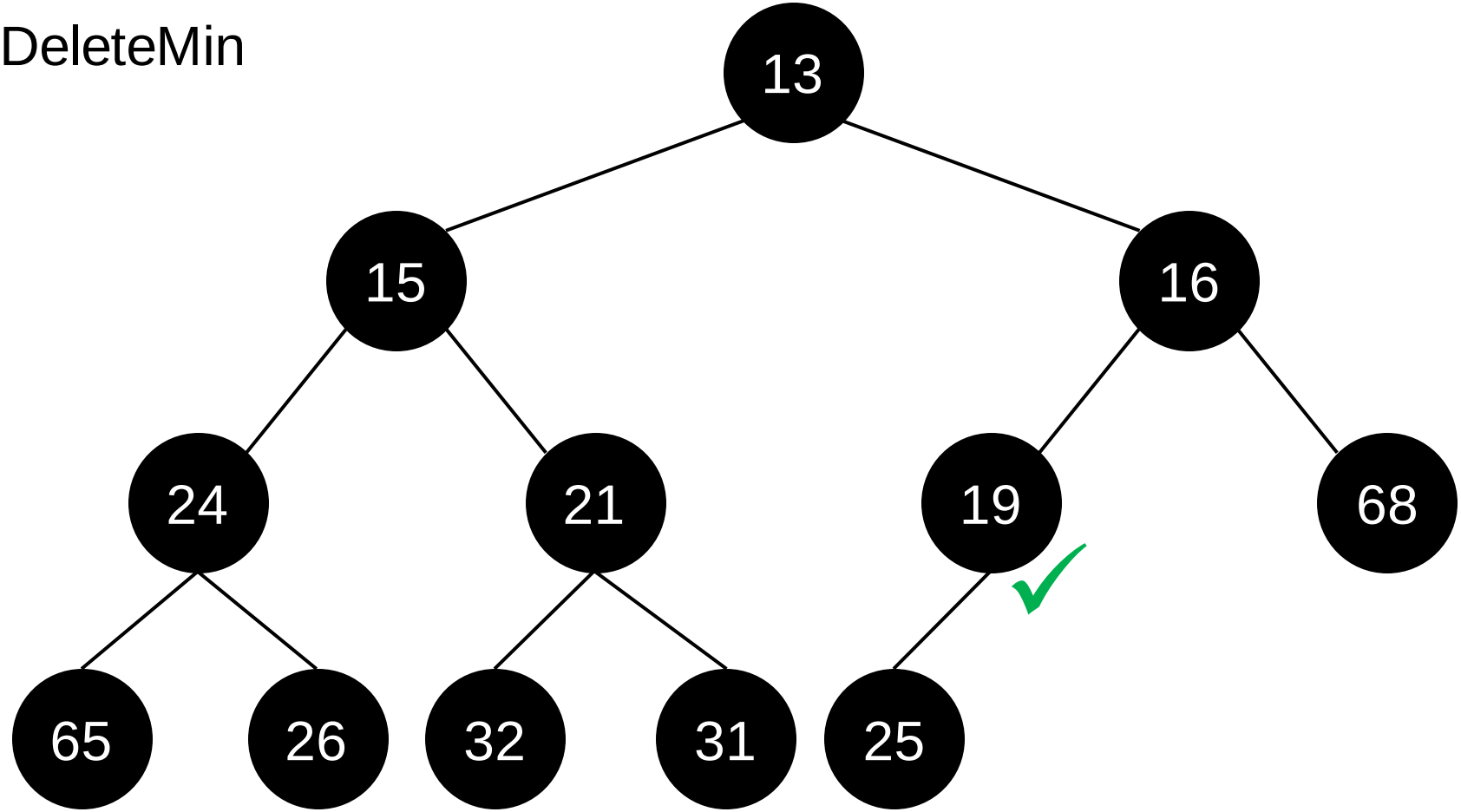
MinHeap Deletion

DeleteMin



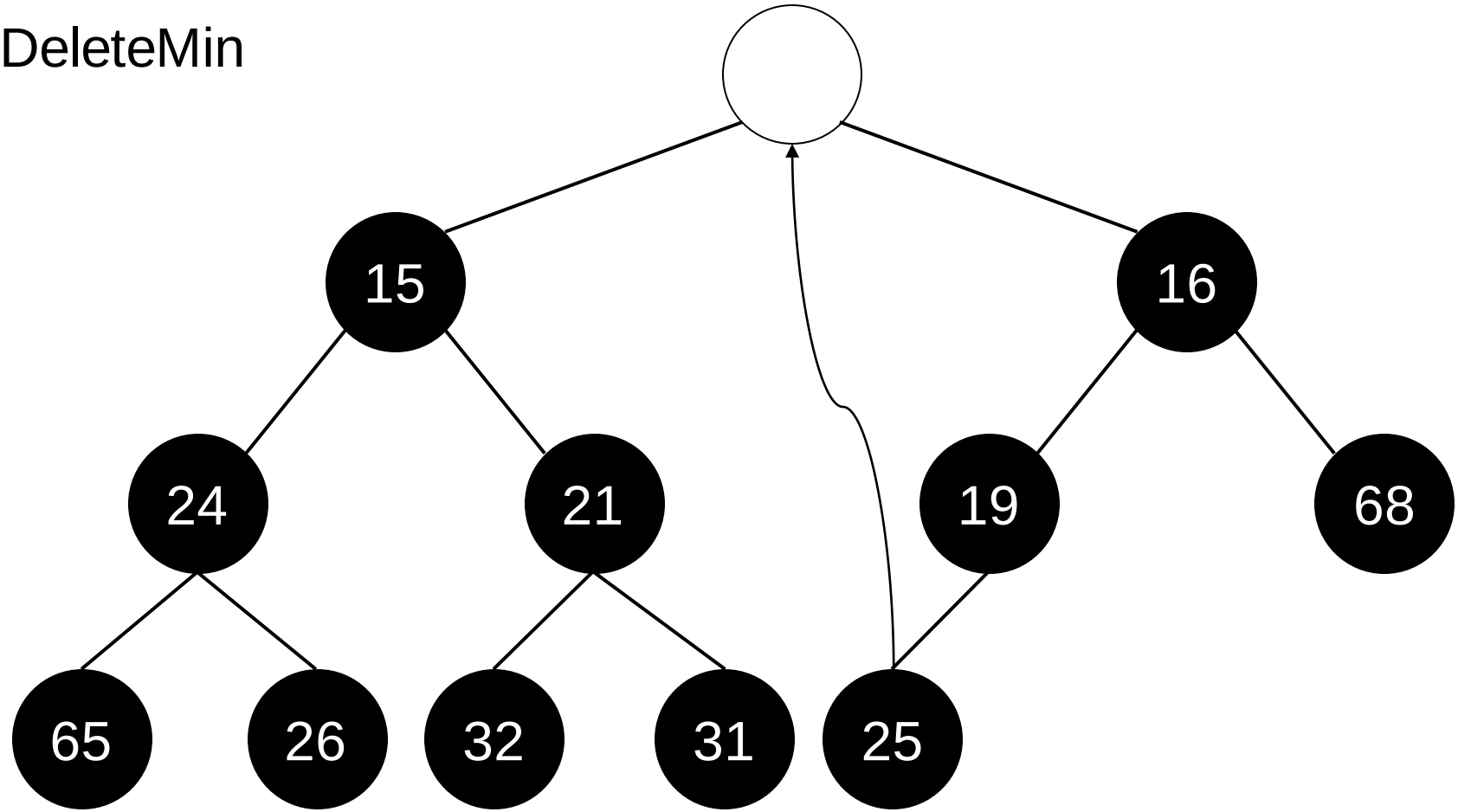
MinHeap Deletion

DeleteMin



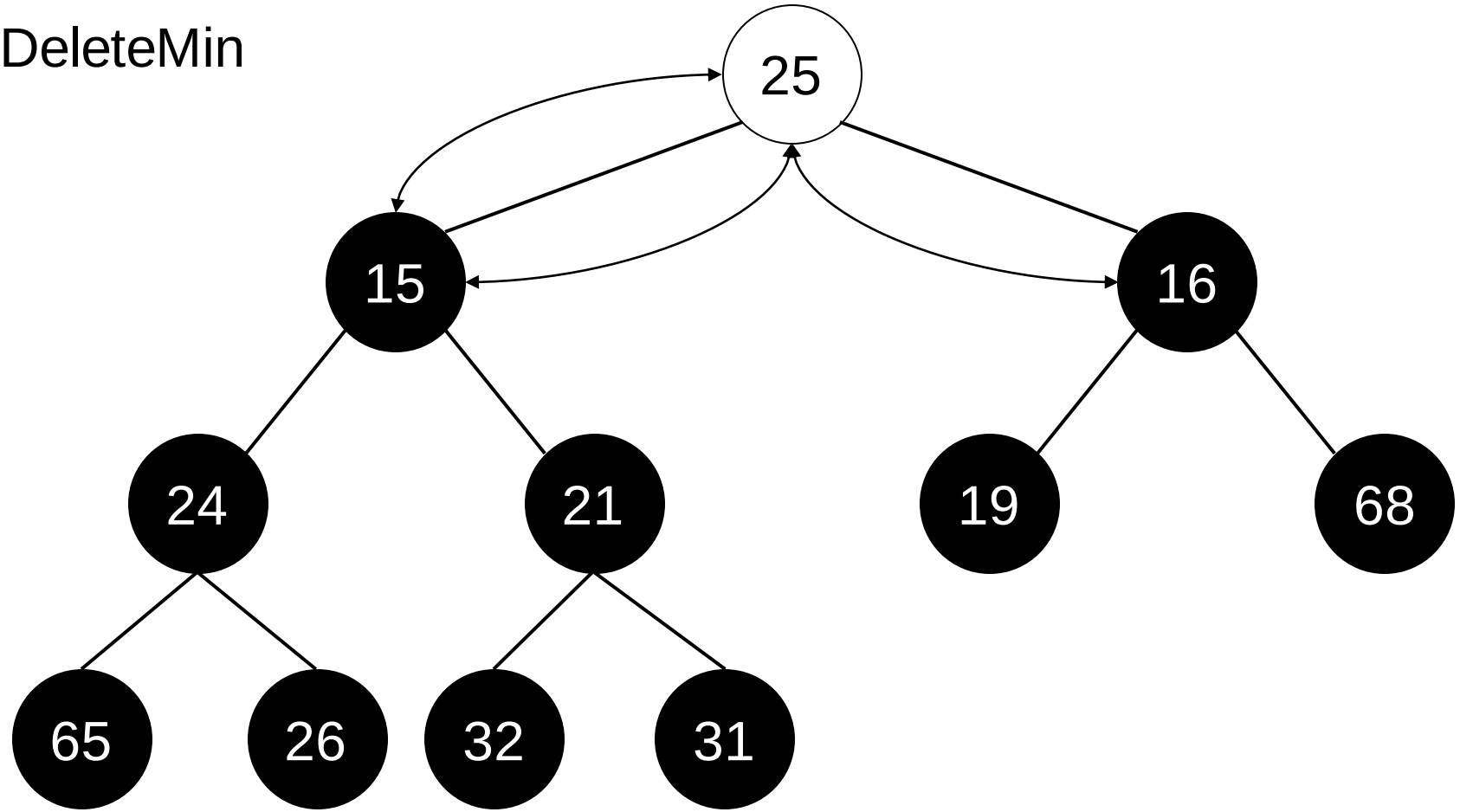
MinHeap Deletion

DeleteMin



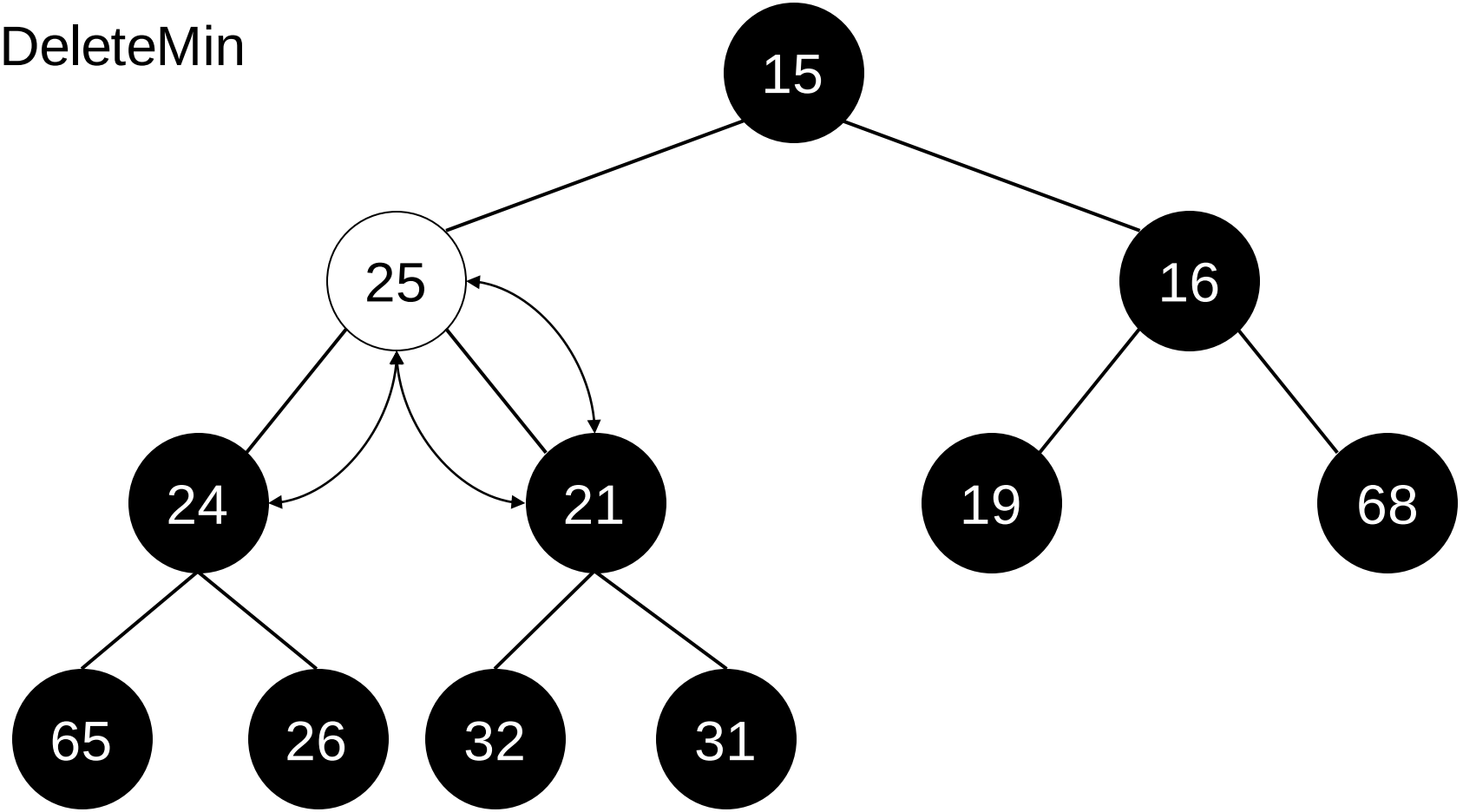
MinHeap Deletion

DeleteMin



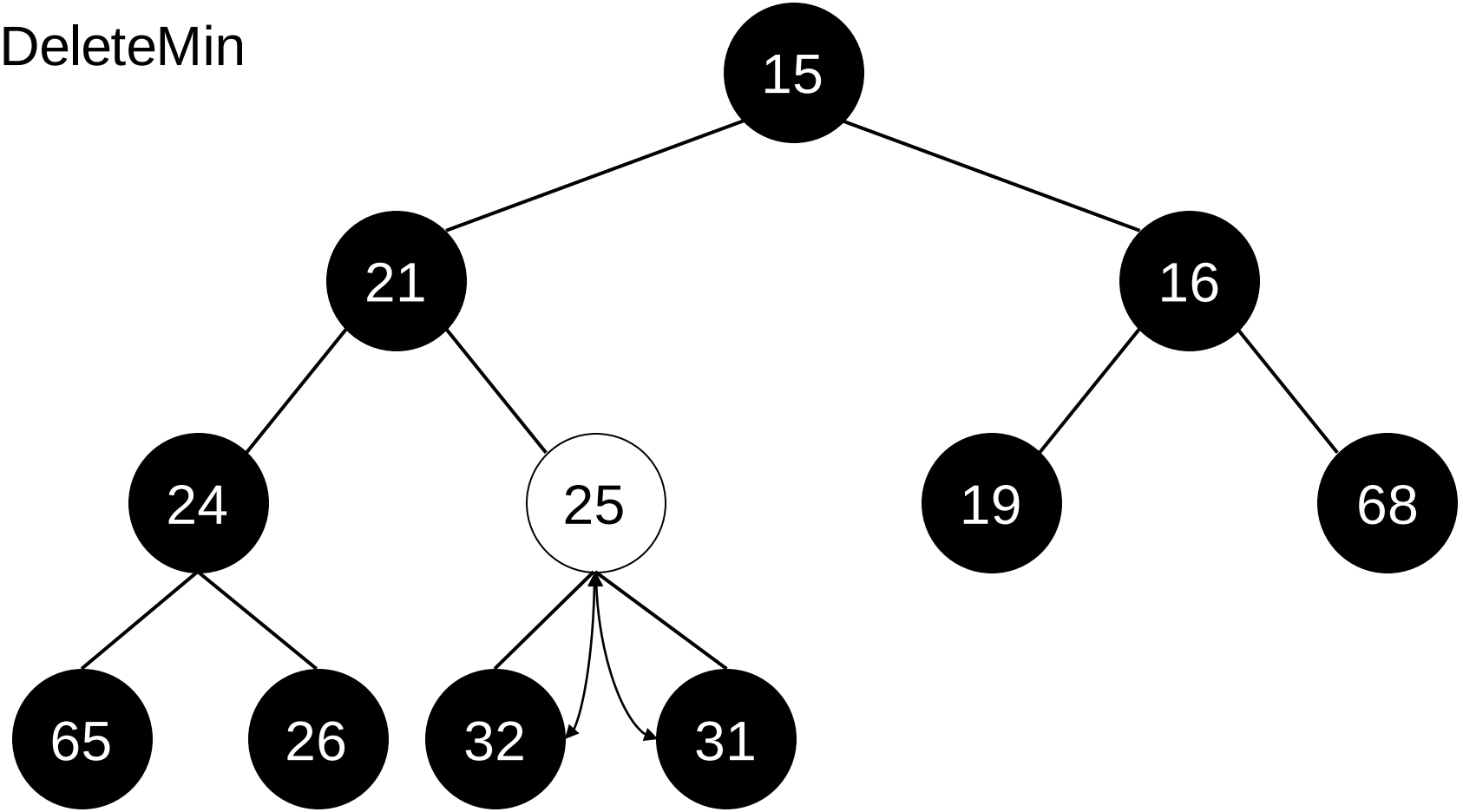
MinHeap Deletion

DeleteMin



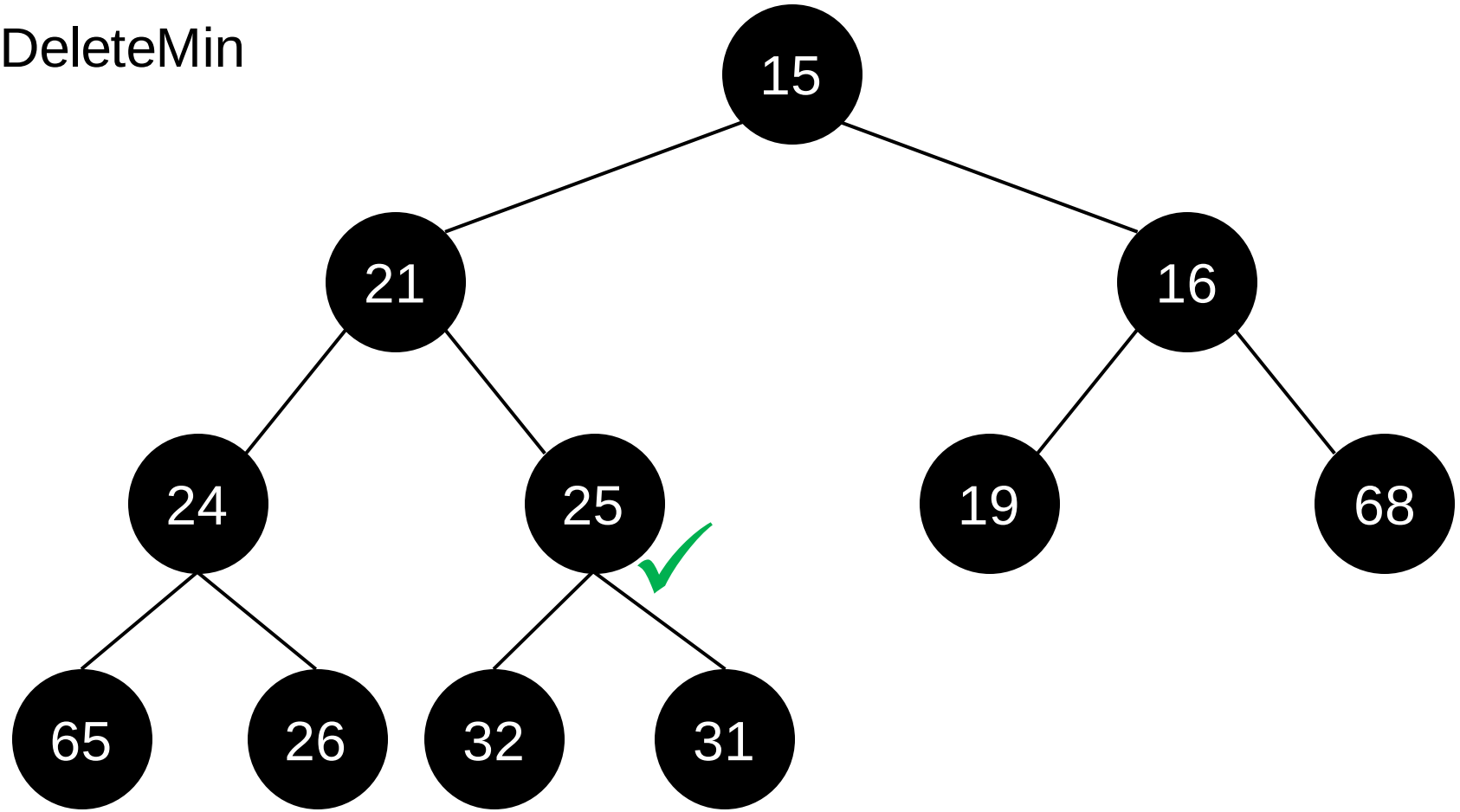
MinHeap Deletion

DeleteMin



MinHeap Deletion

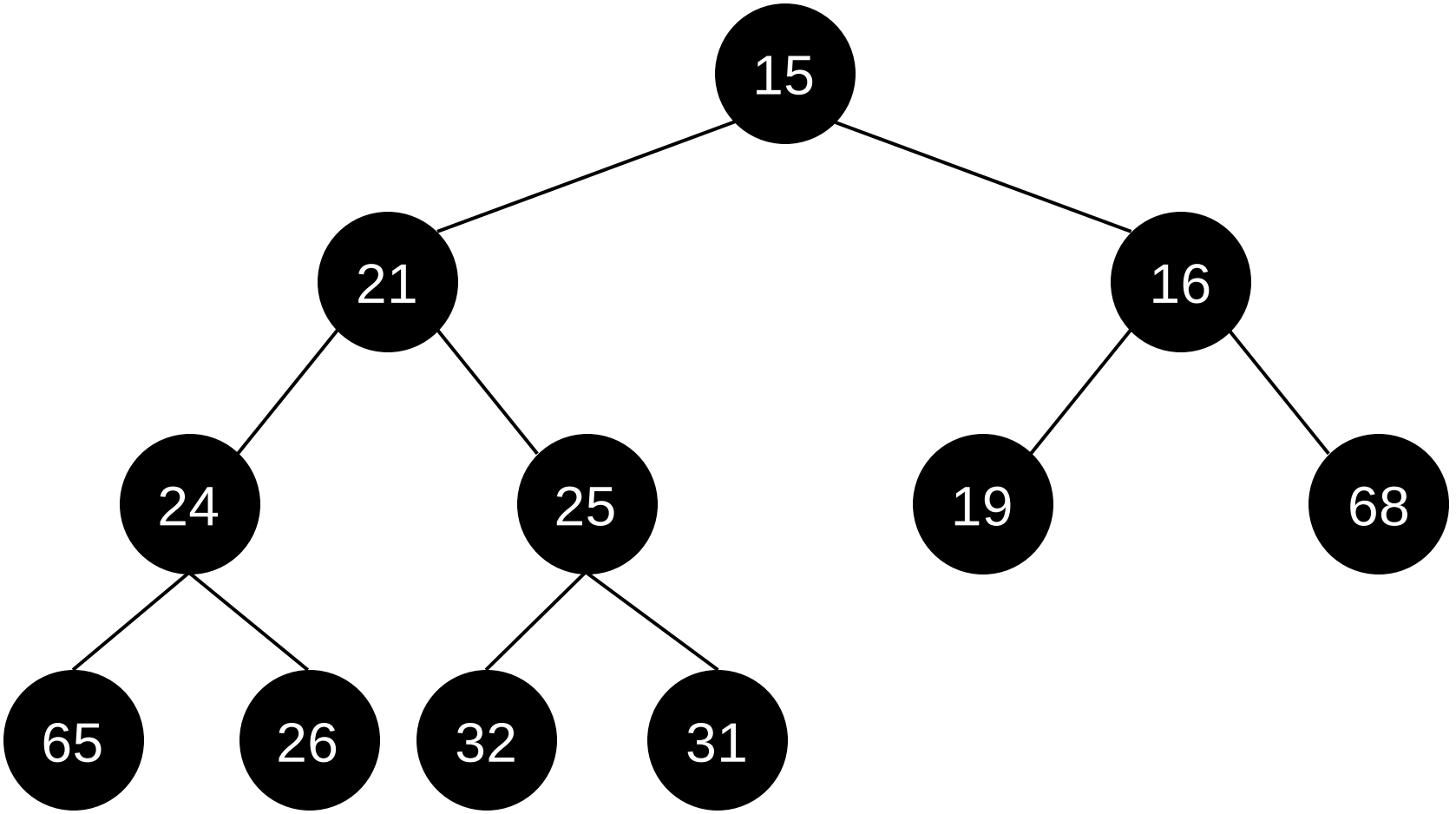
DeleteMin



Array Representation

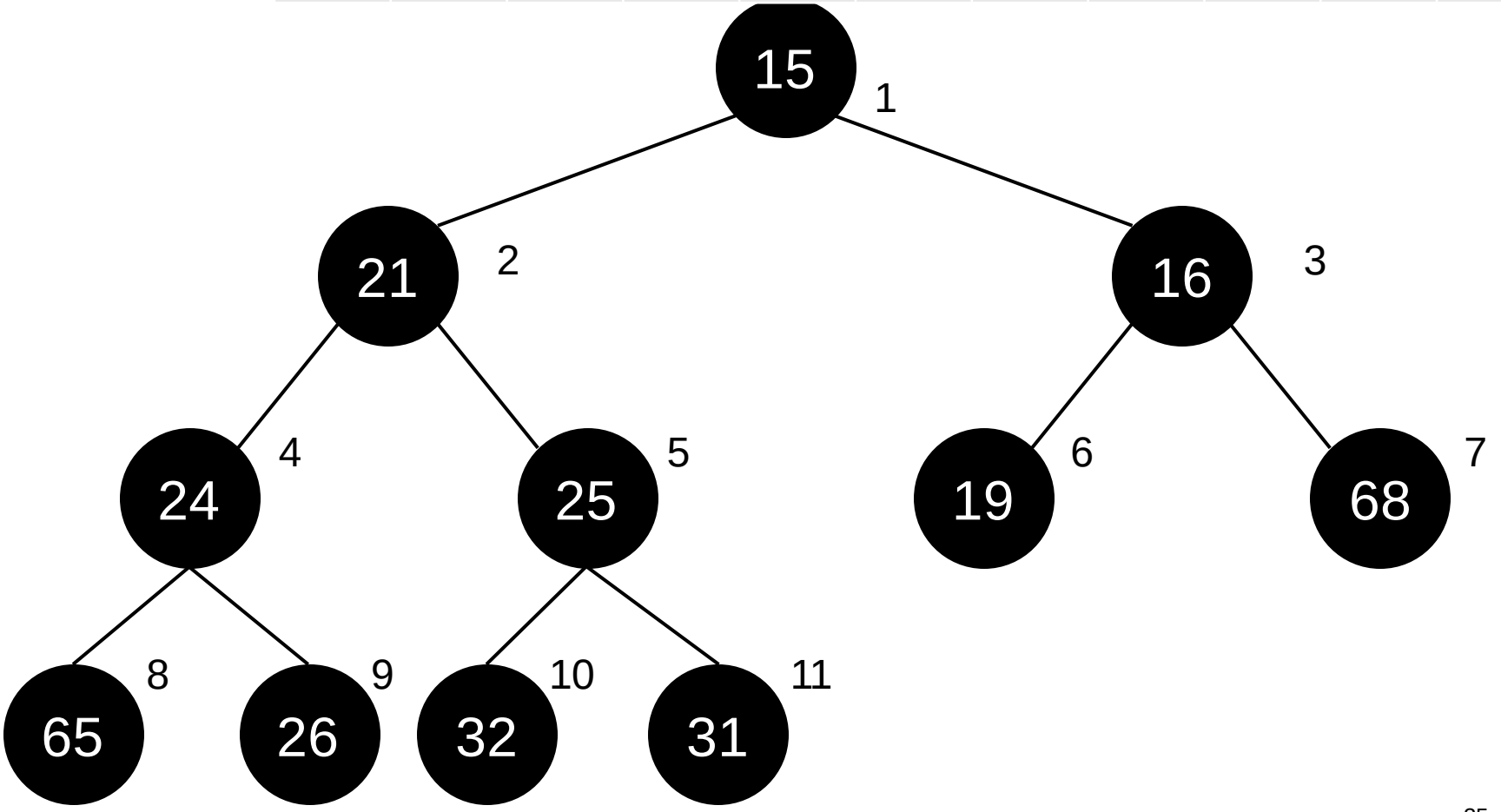
Level-order Traversal

15	21	16	24	25	19	68	65	26	32	31



Array Representation

Level-order	15	21	16	24	25	19	68	65	26	32	31
Traversal	1	2	3	4	5	6	7	8	9	10	11



Array Implementation

```
template <typename T>
class MinHeap {
  T[] keys;
  int size;
  int capacity;
};

#define ROOT 1
#define LEFT_CHILD(p) p*2
#define RIGHT_CHILD(p) p*2+1
#define PARENT(p) p/2
int treePointer = ROOT;
```

Do you spot any mistakes or bad practices?

Array Implementation

```
template <typename T>
class MinHeap {
  T[] keys;
  int size;
  int capacity;
};

#define ROOT 1
#define LEFT_CHILD(p) ((p)*2)
#define RIGHT_CHILD(p) ((p)*2+1)
#define PARENT(p) ((p)/2)
int treePointer = ROOT;
```

Top-K Revisit

- Given an unordered list of n values, and an integer K, find the K largest elements

- Initialize a MinHeap of size K
- Insert the first K values in the list into the Heap
- For each additional value x
  - If x is smaller than the top of the heap (PeekMin)
    - Discard x
  - Else
    - DeleteMin
    - Insert(x)
- Return all the values in the MinHeap

O(n log k)