COMP2421—DATA STRUCTURES AND ALGORITHMS

Heaps (Priority Queue)

Dr. Radi Jarrar
Department of Computer Science
Birzeit University

جامعة بيرزيت
BIRZEIT UNIVERSITY

Heaps (Priority Queue)

- Queues are implemented under the FIFO principle.
- The first element to come in, the first element to be served.
- However, there has to be a scheme for cases in which there are some priorities.

Heaps (Priority Queue)

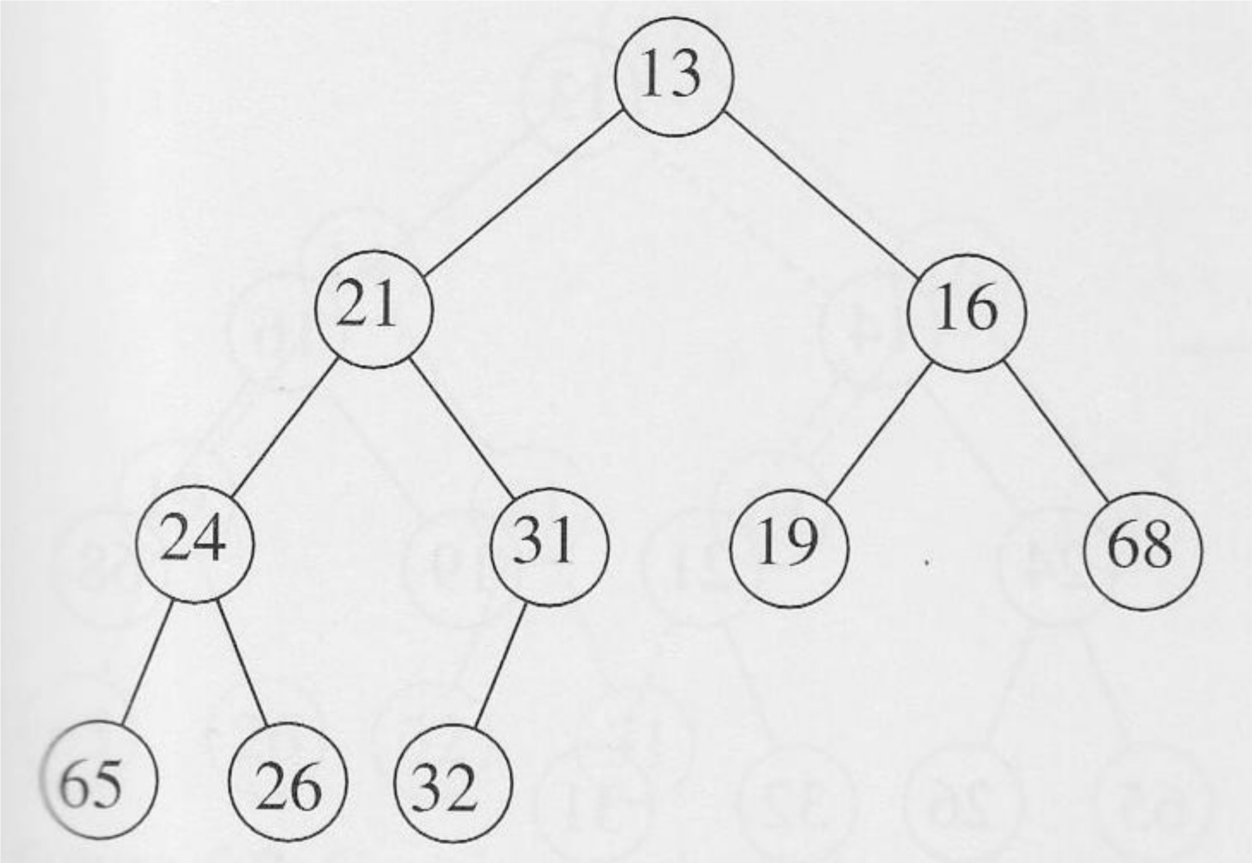
- Take an example a queuing system for printers.
- The first print job to come in is the first to be printed.
- Imagine you are printing a 2-paper report but unfortunately there are a few jobs that have been enqueued before. Each of these jobs of thousands of pages!
- This means you have to wait until all jobs have been finished.

Heaps (Priority Queue)

- There has to be a method so that short jobs to finish as fast as possible. These jobs have higher precedence over jobs that have already been running.
- However, some (not short) jobs are still important and they should have higher precedence in some cases!

Heaps (Priority Queue)

- The Heaps data structures is a special kind of queue (called Priority Queue).
- Heaps use Binary Tree implementation. This gives O(log n) average running time for both operations.

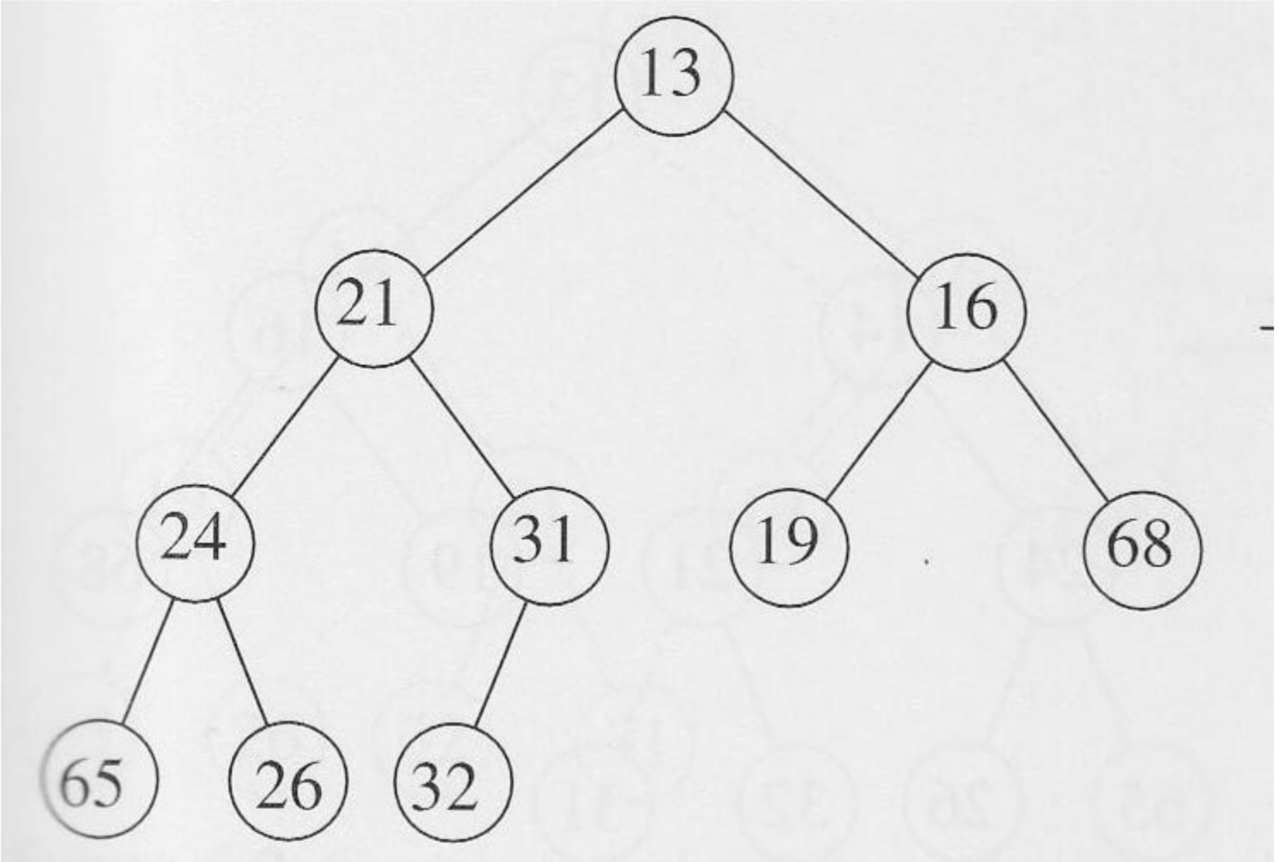


MinHeap / MaxHeap

- MinHeap: a Binary Tree such that the data in each node is less than or equal to the data in the nodes' children.
- In MinHeap, the root is the minimum value in the tree.
- In the MaxHeap, the root is the maximum value in the tree.

Properties of Heaps

- Minimum/maximum element is in the root.
- A heap with N elements has height = $log_2 N$.



Heaps (Priority Queue)

- There are two basic operations in Heaps:

  1. Insert (enqueue)

  2. DeleteMin (or max) (dequeue): finds, return, and delete the minimum element in the priority queue

- Heapify: create a heap from an input array

Heaps (Priority Queue)

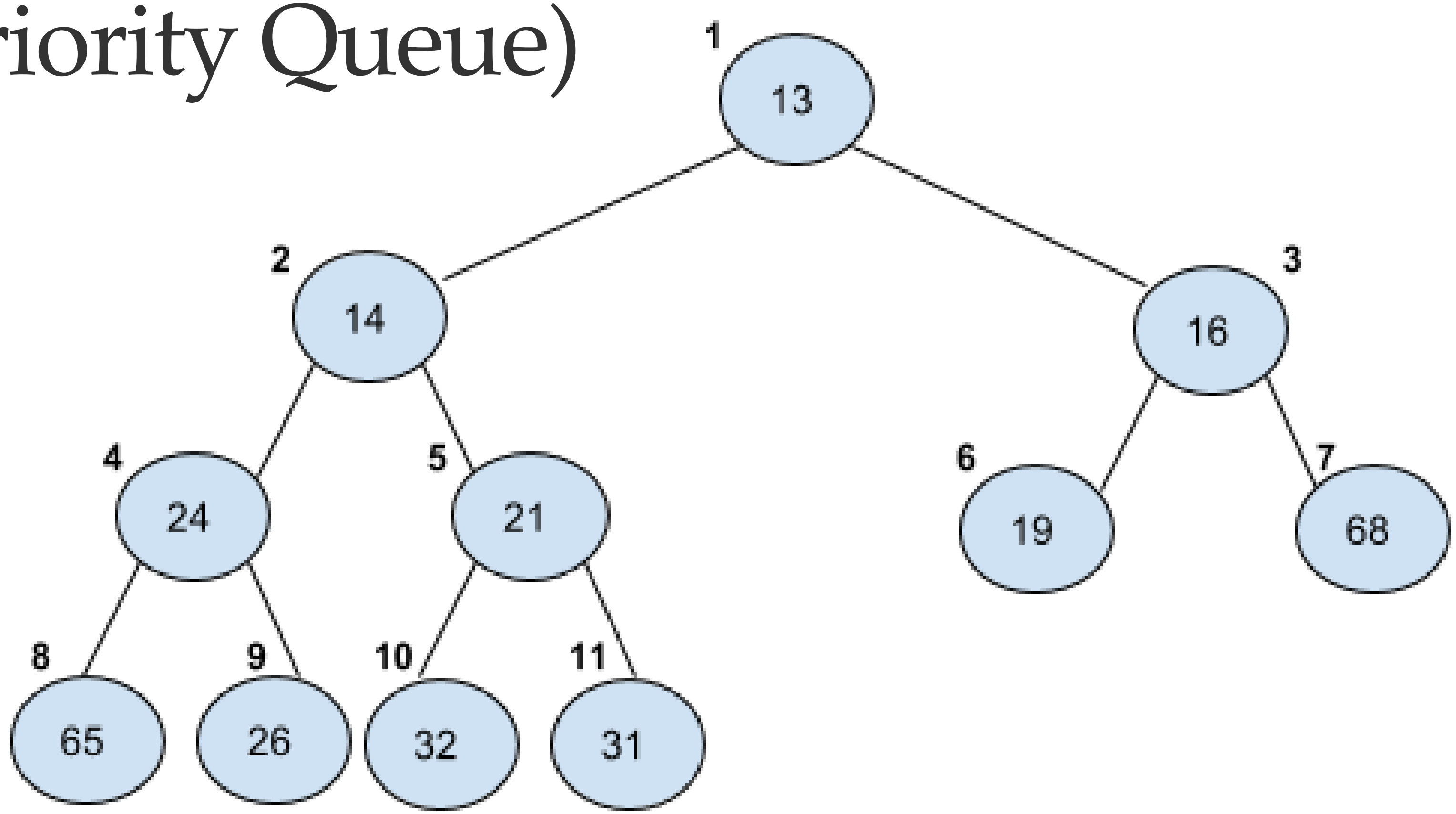
- Heap Data Structure: it is an array that can be viewed as Binary Tree. It may also be called Binary Heaps.
- For any element at position i:

Parent(i) = i/2

Child(i) = 2i left

= 2i + 1 right

Heaps (Priority Queue)



Array		13	14	16	24	21	19	68	65	26	32	31
Index	0	1	2	3	4	5	6	7	8	9	10	11

Notice that there is no need for pointers!

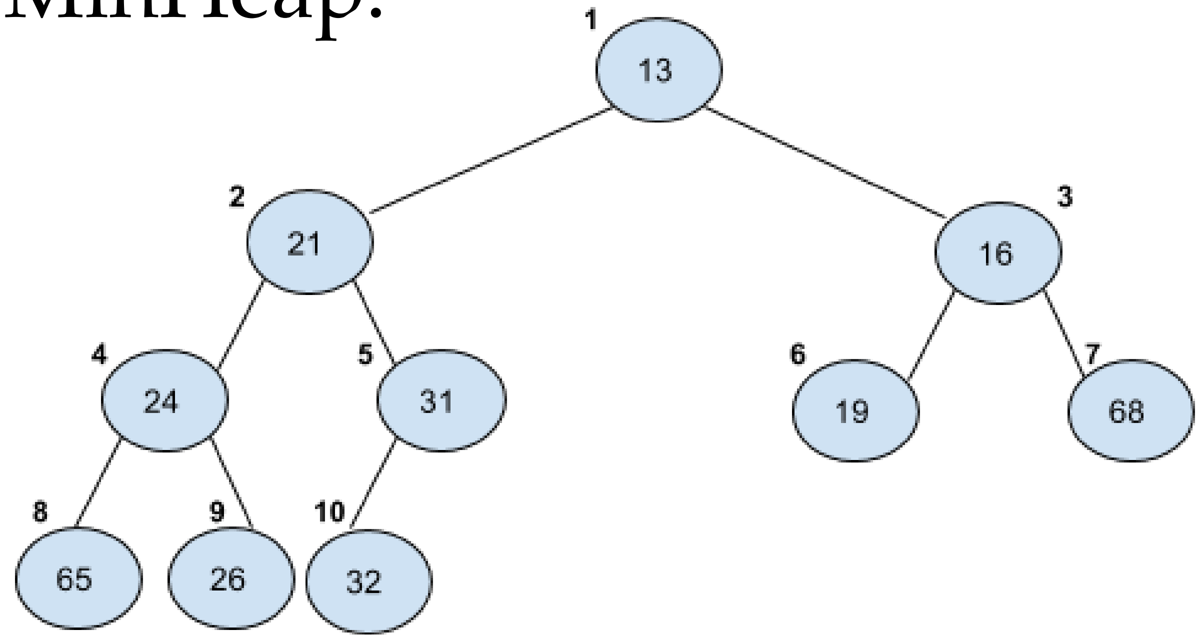
Heap Order Property

- Goal: find the min value quickly.
- It makes sense to have the minimum value at the root as we always delete min.
- Any node should be smaller than any of its descendants.
- For every node X, the key in the parent of X is smaller than or equal to the key in X.

Heaps - Example

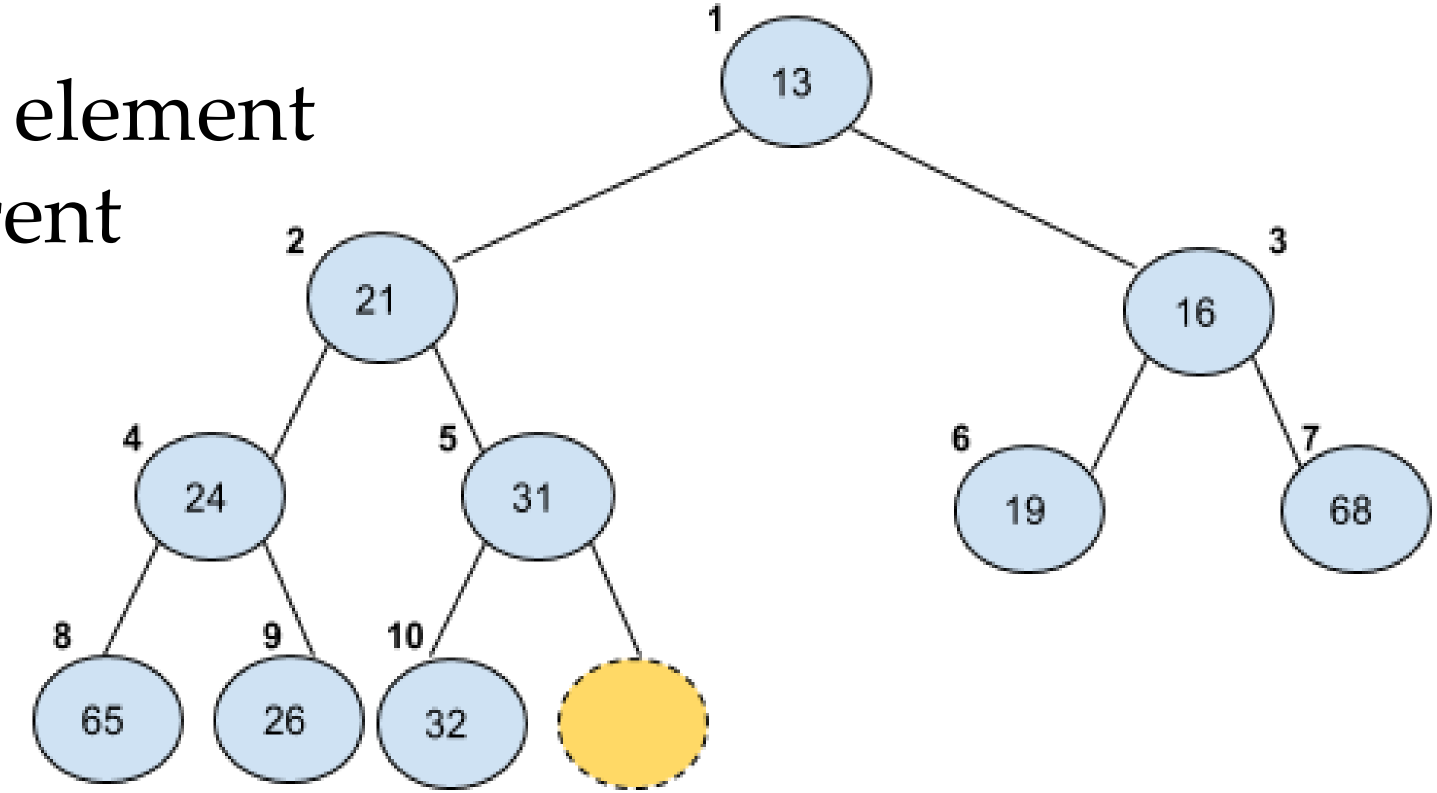
- Insert(14) to the following MinHeap.

1. Find the next available spot in the tree to insert a new node

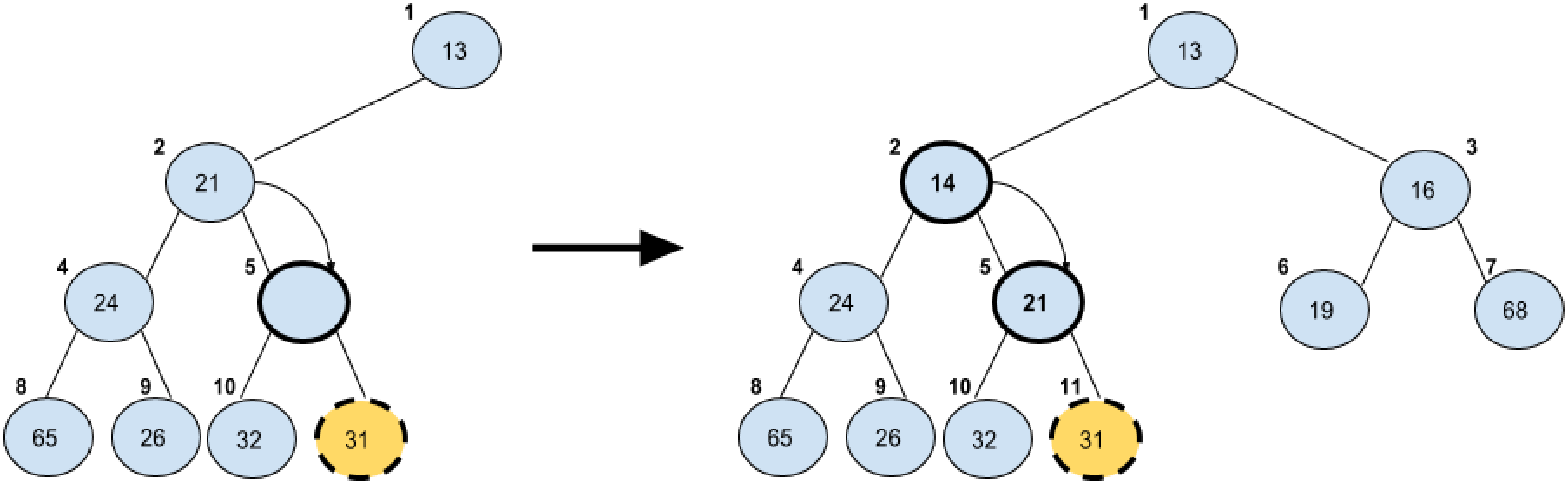


Heaps - Example

2. If X can be placed without violating the rule of heaps, then done.
3. Otherwise, slide the element that is in the hole's parent node into the hole.

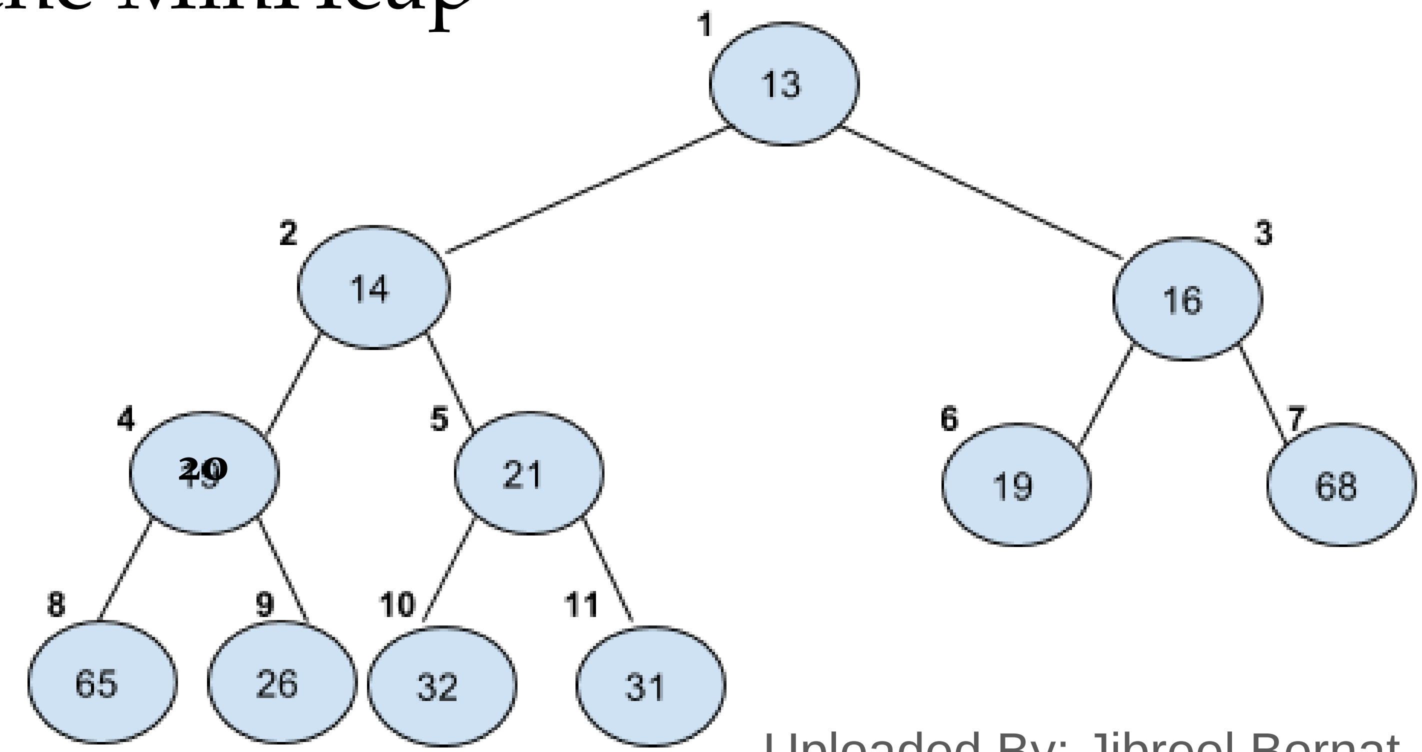


Heaps - Example



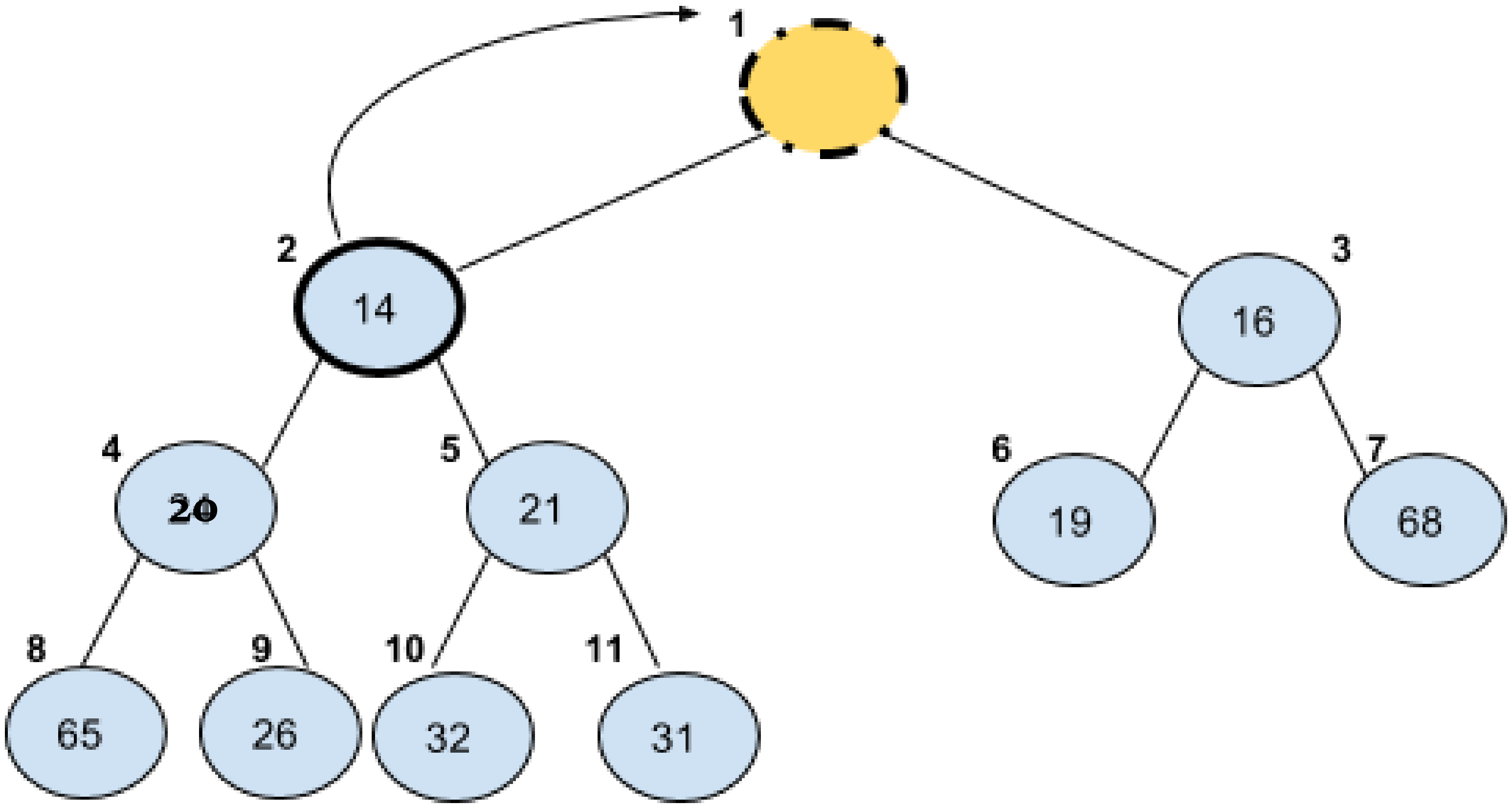
Heaps - Example

- Delete: We can delete either the min or max values. In our case, it is the min as we are operating on MinHeap.
- Example: Delete() from the MinHeap in the following heap.



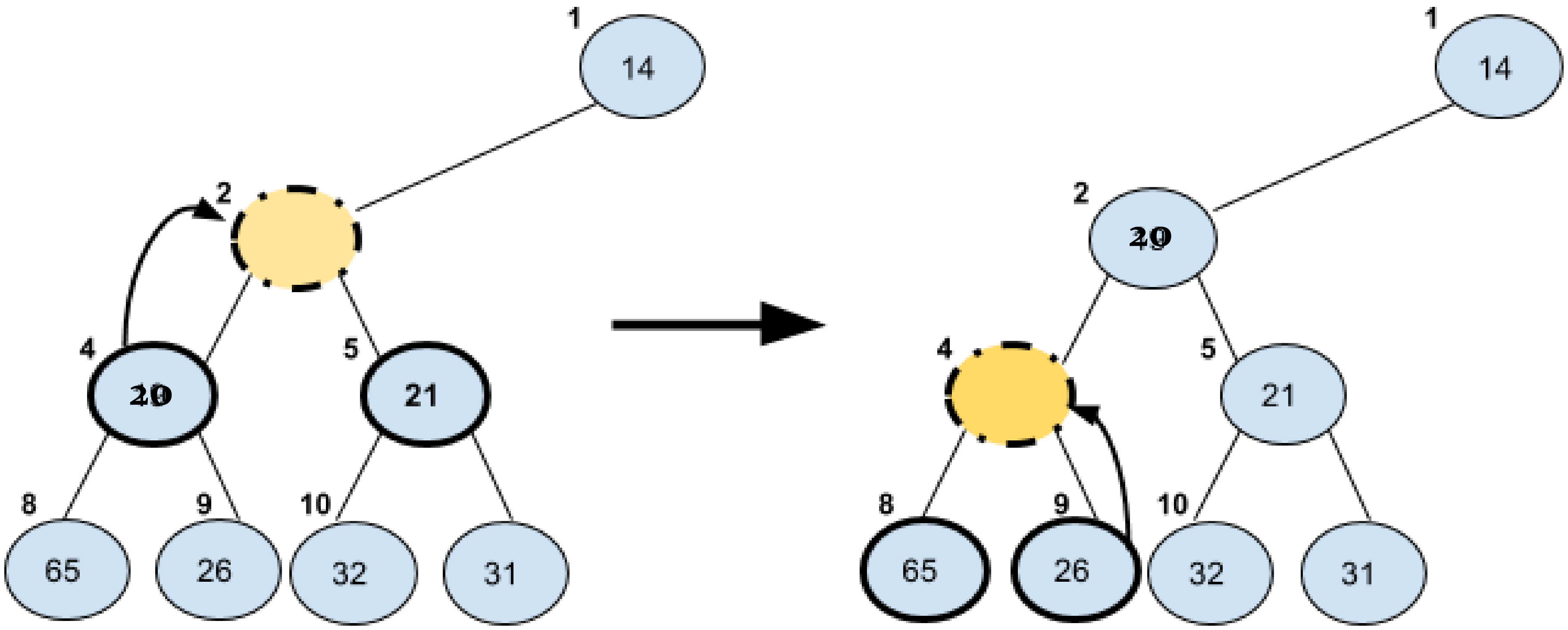
Heaps - Example

1. Find min and remove it. Keep a hole in its place (root).



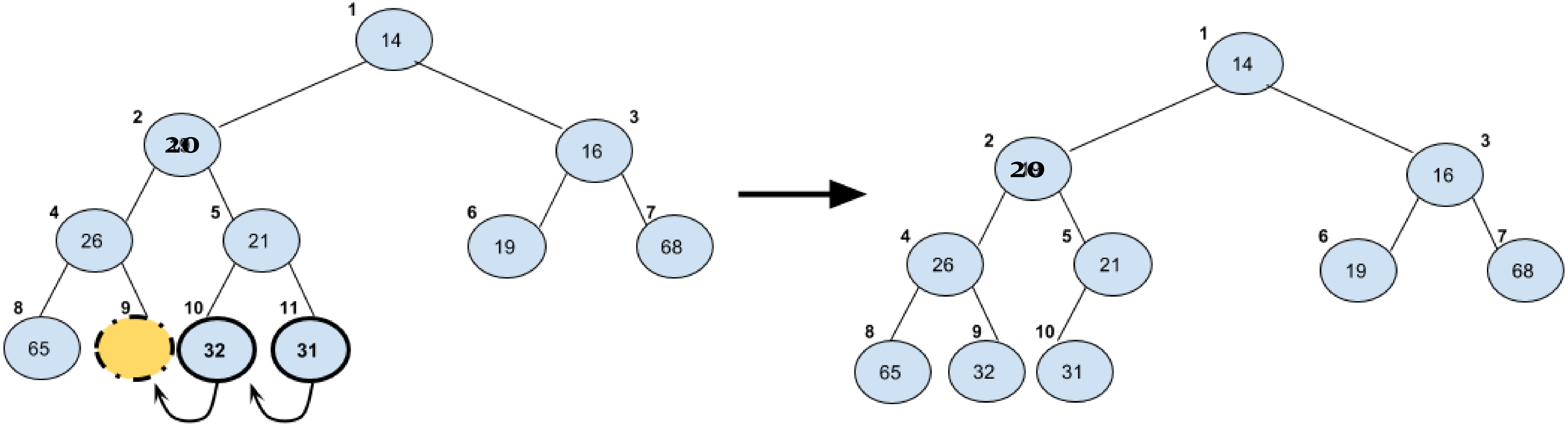
Heaps - Example

2. Push the smaller of the children up.



Heaps - Example

2. Slide the elements on the right to the left to fill the empty spot.



Heapify

Is the process of creating a heap from a binary tree (creating either a min-heap or max-heap)

1. The input is the following array

Array		4	10	3	2	5	6
Index	0	1	2	3	4	5	6

Heapify

2. Create a binary tree (complete tree)

Array		**4**	**10**	**3**	**2**	**5**	**6**
Index	0	1	2	3	4	5	6

Heapify

Array		10	20	25	6	12	15	4	16
Index	0	1	2	3	4	5	6	7	8

Heapify

Algorithm 1: Max-Heapify Pseudocode

Data: B: input array; s: an index of the node

Result: Heap tree that obeys max-heap property

```
Procedure Max-Heapify(B, s)
    left = 2s;
    right = 2s + 1;
    if left ≤ B.length and B[left] > B[s] then
        largest = left;
    else
        largest = s;
    end
    if right ≤ B.length and B[right] > B[largest] then
        largest = right;
    end
    if largest ≠ s then
        swap(B[s], B[largest]);
        Max-Heapify(B, largest);
    end
end
```

Applications

- Priority queue.
- Heap sort: sorting data using heaps.
- Selection of min/max from a set of data in O(1) time.
- Graph algorithms: heaps are used in graph algorithms like implementing Dijkstra's algorithm.

Running time

	Insert	DeleteMin	FindMin	Build the heap
Binary Heap	O(log n)	O(log n)	O(1)	O(n)