CSC 225 - Summer 2019

Priority Queues I

Bill Bird

Department of Computer Science
University of Victoria

June 10, 2019

Priority Queues (1)

A **Priority Queue** data structure stores a collection of elements, each with an associated priority (also called a 'key'). Priority Queues have two core operations:

1. INSERT(e): Add the element e to the collection.
2. REMOVEMIN(): Remove (and return) the element in the collection with the lowest priority.

In this course, the keys used for priority queues tend to be integers. However, any type which supports pairwise comparisons (of the form '$a < b$') can be used as a key.

Priority Queues (2)

The priority queue operations can be implemented with several different underlying data structures:

Data Structure	INSERT	REMOVEMIN
Unsorted Array	$O(1)$	$O(n)$
Sorted Array	$O(n)$	$O(1)$
Heap	$O(\log n)$	$O(\log n)$

Priority Queues (3)

Applications of Priority Queues

- Job Scheduling: Each element represents a job which requires a shared resource (e.g. CPU time). When the resource is available, jobs are selected in priority order.
- Graph Algorithms: Priority queues are a component of Dijkstra's algorithm and Kruskal's Algorithm (covered in CSC 226).

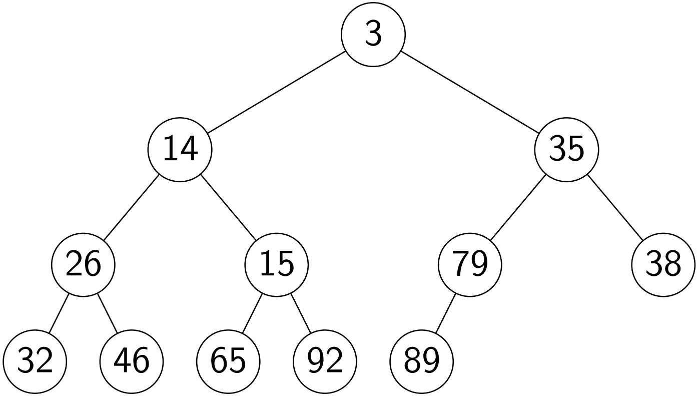
Sorting with a Priority Queue

Any priority queue Q can be used to sort an array A of size n with a simple algorithm:

- Add each element $A[i]$ to Q with priority $A[i]$.
- Call $\textsc{RemoveMin}$ n times and store the resulting sequence back to A.

When a heap is used to implement the priority queue, the resulting algorithm is Heap Sort, which is $O(n \log n)$.

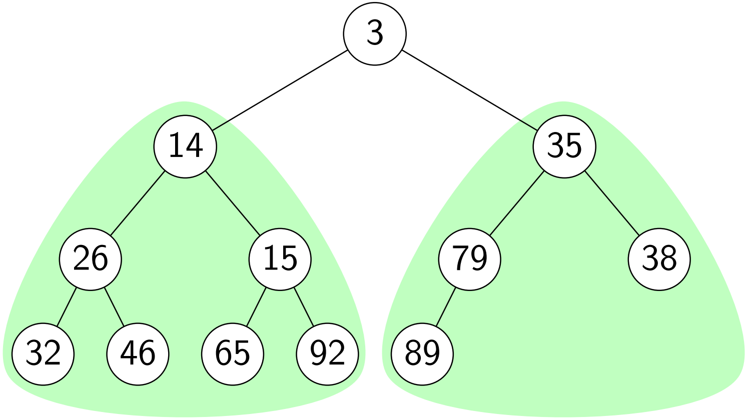
Heaps (1)



A **heap** is a specialized binary tree data structure with two extra properties:

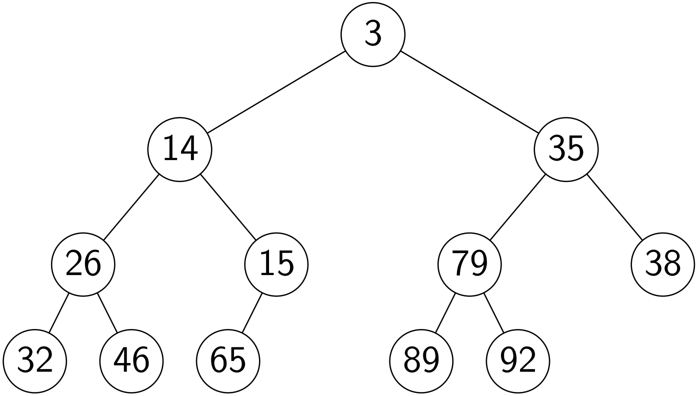
- **Heap Property**: The value at a node must be less than or equal to the values of its children.
- **Shape Property**: A heap must be a complete binary tree.

Heaps (2)



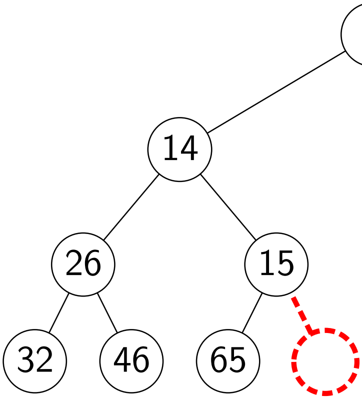
- The root of a heap is always the minimum element.
- The subtree rooted at any node of a heap is also a heap. For example, the element 14 is the root (and minimum) of the left subtree in the diagram above.

Heaps (3)



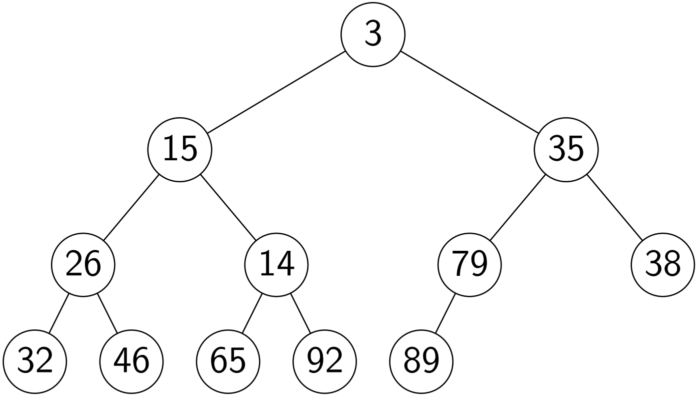
- Is this a heap?

Heaps (4)



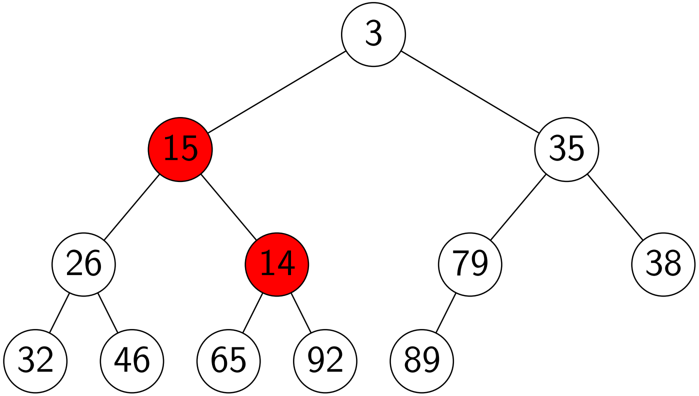
- Is this a heap?
- **No**. The shape property is violated by the missing node in the last level.

Heaps (5)



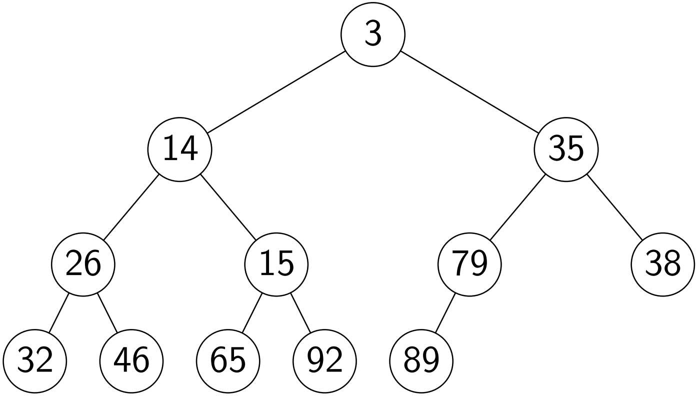
- Is this a heap?

Heaps (6)



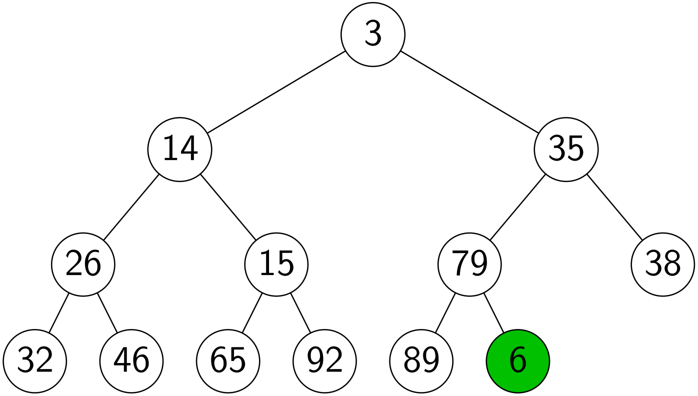
- Is this a heap?
- **No**. The heap property is violated by the pair of highlighted nodes.

Heap Insertion (1)



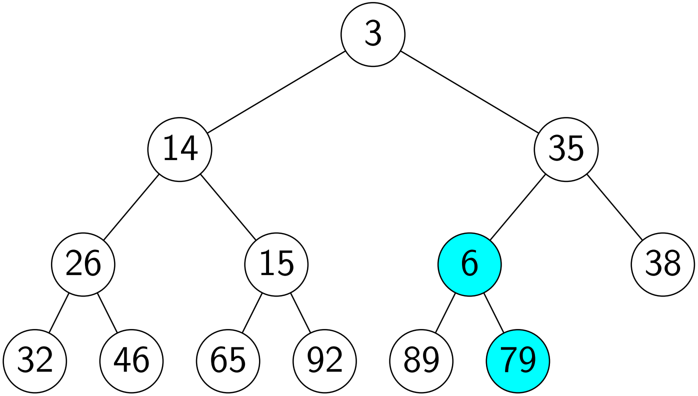
- Task: Insert the element 6 into the heap above.

Heap Insertion (2)



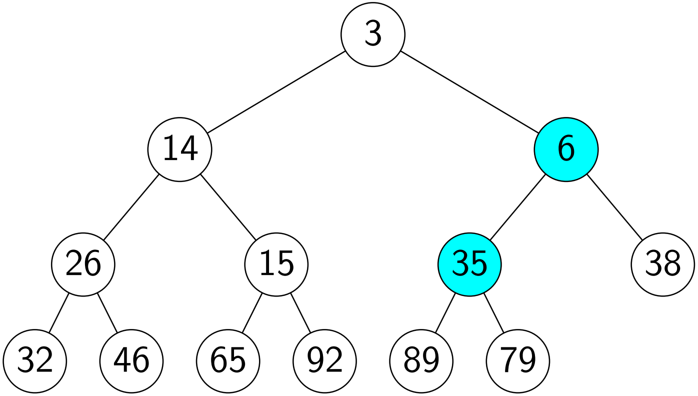
- First, a new node for element 6 is allocated in the next available position on the last level.
- Adding elements this way ensures that the shape property is maintained.

Heap Insertion (3)



- After the element is added to the tree, the heap property is restored.
- At each step, the value of the node is compared to the value of the parent. If the parent is larger, the values are swapped.

Heap Insertion (4)



- The process continues until the parent has a smaller value or the node becomes the root.
- The number of steps is at most the height of the tree.
- This operation is often called ‘bubble-up’.

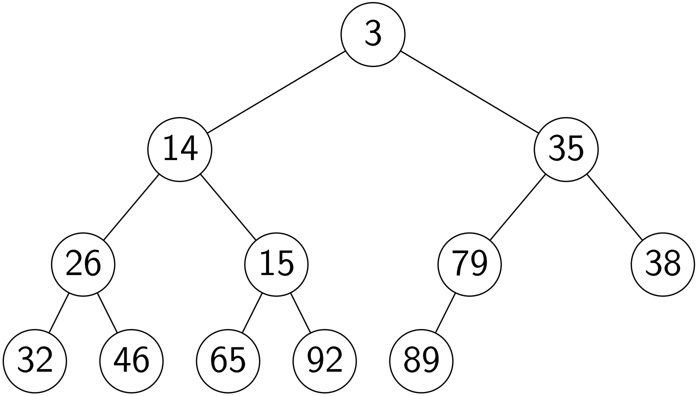
Bubble Up - Iterative

```
procedure BubbleUp(node)
    while node $\neq$ root and node.parent.value $>$ node.value do
        Swap node.value and node.parent.value
        node $\leftarrow$ node.parent
    end while
end procedure
```

Bubble Up - Recursive

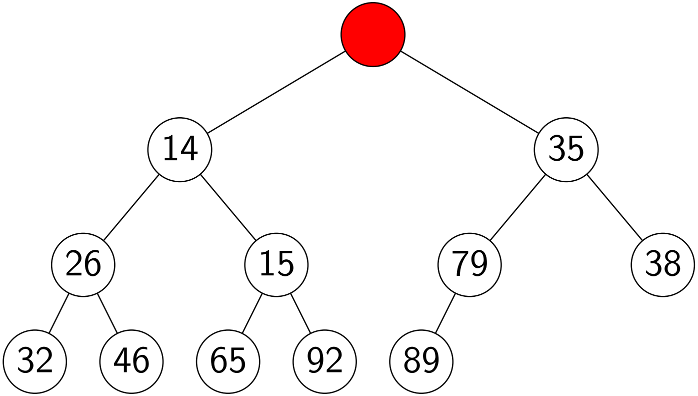
```
procedure BubbleUp(node)
    if node = root then
        return
    end if
    if node.parent.value > node.value then
        Swap node.value and node.parent.value
        BubbleUp(node.parent)
    end if
end procedure
```

Heap Removal (1)



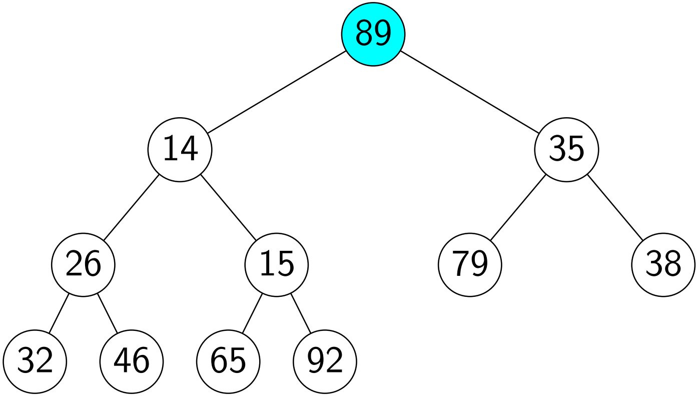
- Task: Remove the minimum element from the heap above.
- When used as a priority queue, the minimum is the only element ever removed.
- In general, any element can be removed from a heap.

Heap Removal (2)



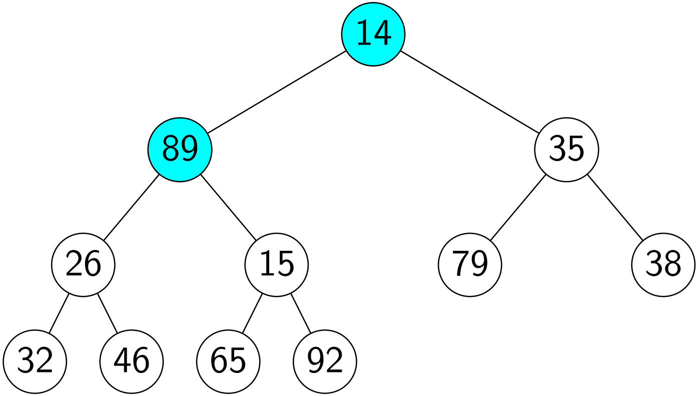
- To remove the minimum, the element at the root is deleted from the tree. Normally, the value is returned after the heap is restored.

Heap Removal (3)



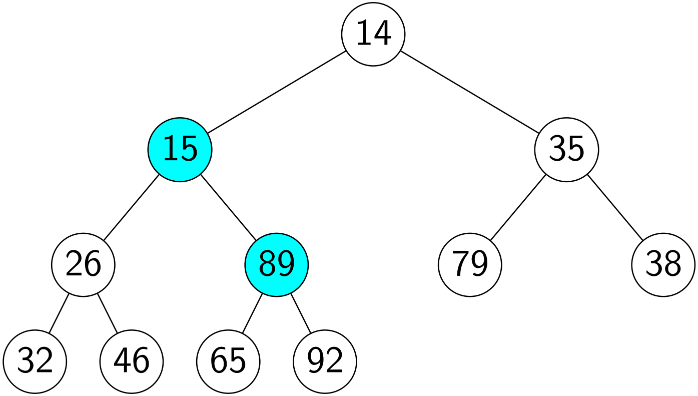
- To fill the ‘hole’ left by the deleted element, the last element in the last row is moved into the empty position.

Heap Removal (4)



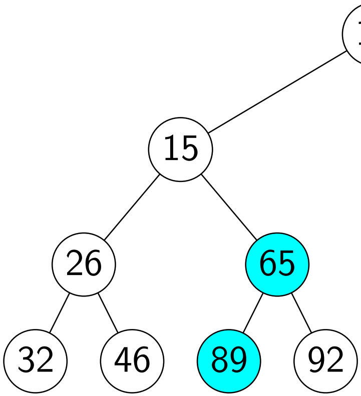
- After the element is moved, the heap property is restored with the ‘bubble-down’ operation.

Heap Removal (5)



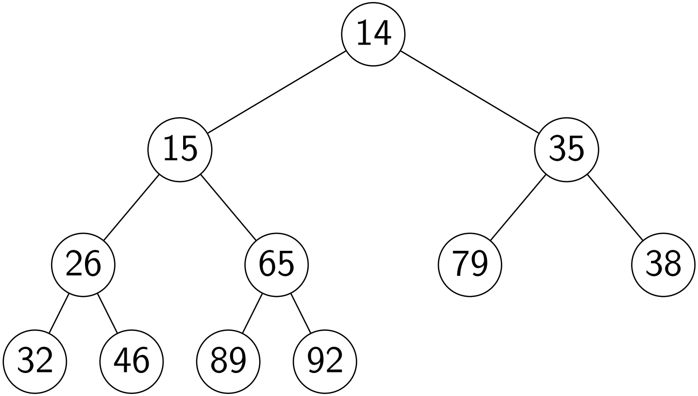
- If the node has a larger value than one of its children, it is swapped with the smaller child.

Heap Removal (6)



- The process continues until the node has a smaller value than both children, or it has become a leaf.

Heap Removal (7)



- The number of steps is at most the height of the tree.

Bubble Down - Iterative

```
procedure BubbleDown(node)
    while node is not a leaf do
        smallestChild $\leftarrow$ Child of node with smaller value.
        if node.value $>$ smallestChild.value then
            Swap node.value and smallestChild.value
            node $\leftarrow$ smallestChild
        else
            Break
        end if
    end while
end procedure
```

Bubble Down - Recursive

```
procedure BubbleDown(node)
    if node is a leaf then
        return
    end if
    smallestChild $\leftarrow$ Child of node with smaller value.
    if node.value $>$ smallestChild.value then
        Swap node.value and smallestChild.value
        BubbleDown(smallestChild)
    end if
end procedure
```