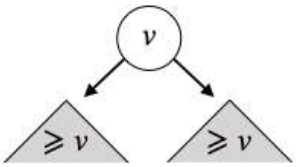
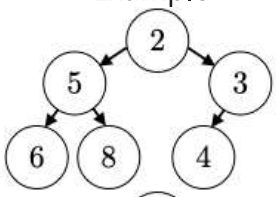
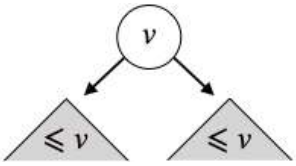
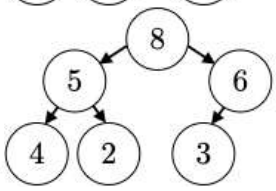
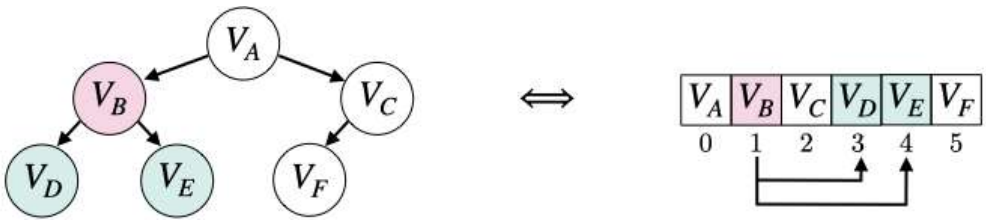
Heaps

Definition

A heap is a complete binary tree with an additional property that makes it either a min-heap or a max-heap:

Type	Description	Illustration	Example
Min-heap	The value of each node is lower than its children's, if any. By definition, the root has the lowest value.		
Max-heap	The value of each node is higher than its children's, if any. By definition, the root has the highest value.		

Since a heap is a complete binary tree, it is convenient to represent it with an array of size n, where n is the number of nodes.



For a given node of index $i \in [\![0, n-1]\!]$,

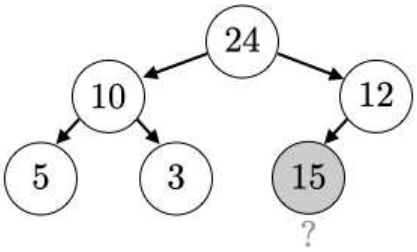
its parent has an index of $\lfloor \frac{i-1}{2} \rfloor$

its left child has an index of $2i+1$ and its right child has an index of $2i+2$

The following parts will use max-heaps for consistency. However, one could also use min-heaps to obtain the same results.

Heapify up

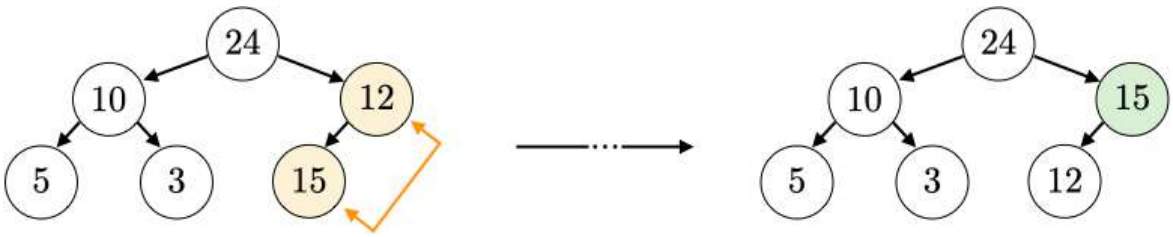
Let's suppose that the last child is potentially not fulfilling the properties of the max-heap. The heapify up operation, also called *bubble up*, aims at finding the correct place for this node.



This operation is done in $\mathrm{O}(\log(n))$ time:

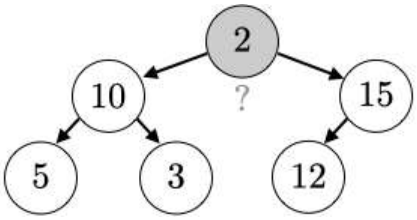
Update step: While the node's parent has a lower value than the node's, swap the two.

Final step: We now have a valid max-heap.



Heapify down

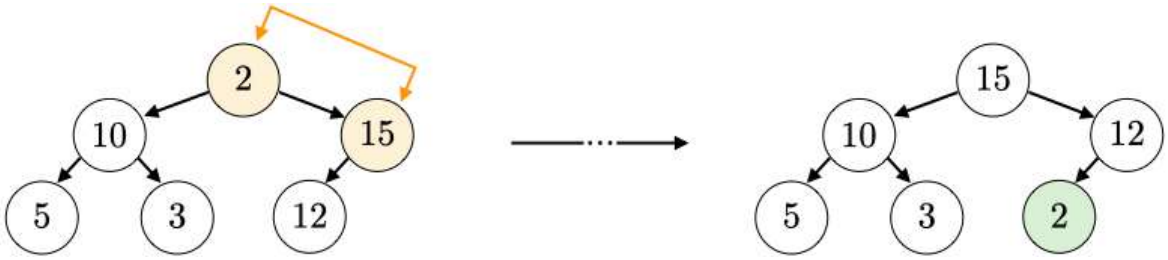
Let's suppose that the root is potentially not fulfilling the properties of the max-heap. The heapify down operation, also called *bubble down, aims at finding the correct place for this node.



This operation is done in $\mathrm{O}(\log(n))$ time:

Update step: While the highest-value child of the node has a higher value than the node's, swap the two.

Final step: We now have a valid max-heap.



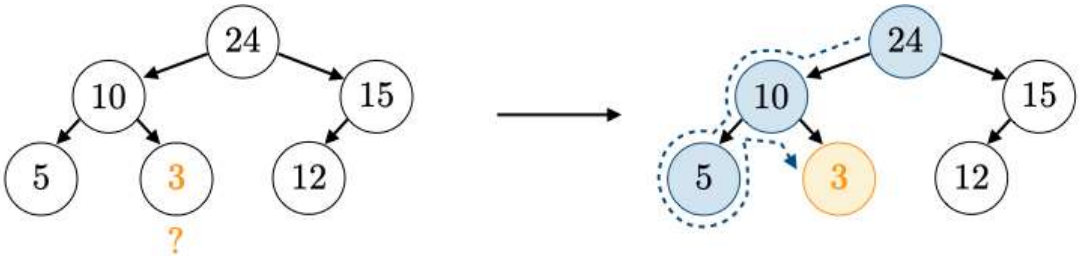
Operations

The main operations that can be performed on a max-heap are explained below:

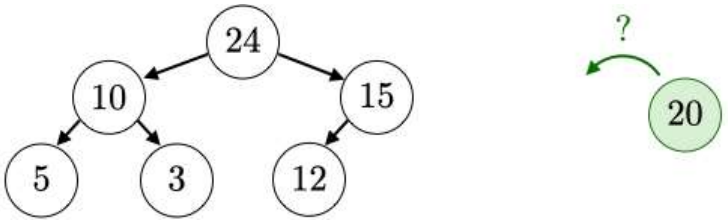
Search We distinguish two cases:

Maximum value: Look at the value corresponding to the root of the heap. It takes $\mathrm{O}(1)$ time.

Any other value: Traverse the tree, given that we have no information as to where each node is. It takes $\mathrm{O}(n)$ time.

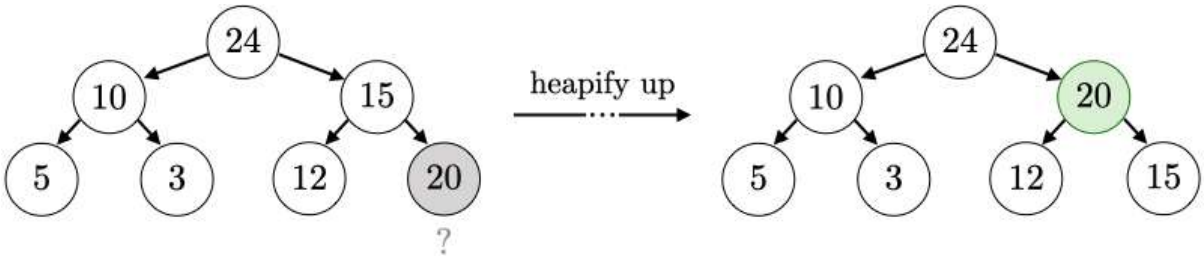


Insertion It takes $\mathrm{O}(\log(n))$ time.



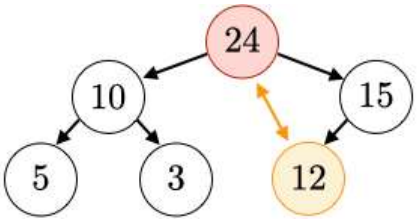
Placeholder step: Add new node as the last child.

Heapify up step: Heapify up the child to its final position.



Deletion It takes $\mathrm{O}(\log(n))$ time.

Swap step: Swap node with last child and remove new last child.



Heapify step: Move the newly-placed node to its final position depending on the situation:

Node's value is higher than parent's: Heapify up to its final position.

Node's value is lower than highest of its children: Heapify down to its final position.

Node's value is lower than parent's and higher than highest of its children: There is nothing to do.