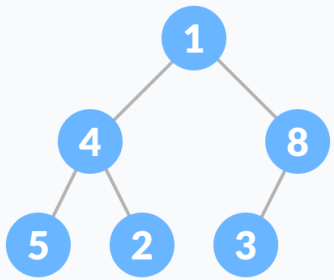
Heap Data Structure

In this tutorial, you will learn what heap data structure is. Also, you will find working examples of heap operations in C, C++, Java and Python.

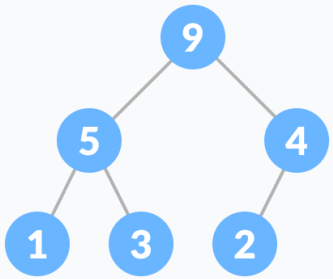
Heap data structure is a complete binary tree that satisfies **the heap property**. It is also called as **a binary heap**.

A complete binary tree is a special binary tree in which

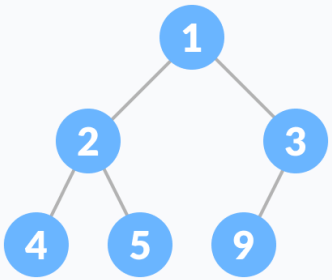
- every level, except possibly the last, is filled
- all the nodes are as far left as possible



Heap Property is the property of a node in which



- (for min heap) key of each node is always smaller than the child node/s and the key of the root node is the smallest among all other nodes.



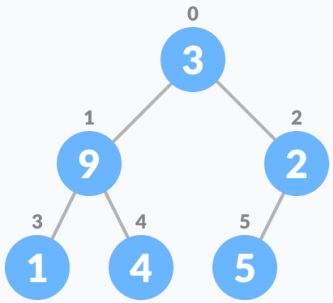
Heap Operations

Some of the important operations performed on a heap are described below along with their algorithms.

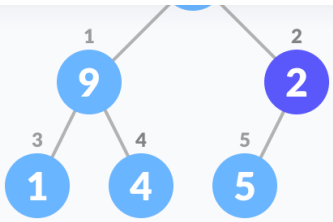
Heapify is the process of creating a heap data structure from a binary tree. It is used to create a Min-Heap or a Max-Heap.

1. Let the input array be

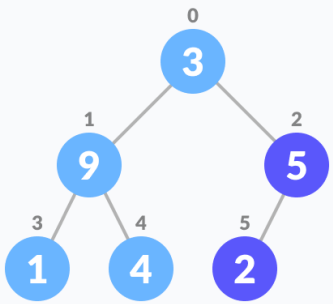
2. Create a complete binary tree from the array



3. Start from the first index of non-leaf node whose index is given by `n/2 - 1`.



4. Set current element `i` as `largest`.
5. The index of left child is given by `2i + 1` and the right child is given by `2i + 2`.
   If `leftChild` is greater than `currentElement` (i.e. element at `ith` index), set `leftChildIndex` as largest.
   If `rightChild` is greater than element in `largest`, set `rightChildIndex` as `largest`.
6. Swap `largest` with `currentElement`



7. Repeat steps 3-7 until the subtrees are also heapified.

```
Heapify(array, size, i)
  set i as largest
  leftChild = 2i + 1
  rightChild = 2i + 2

  if leftChild &gt; array[largest]
    set leftChildIndex as largest
  if rightChild &gt; array[largest]
    set rightChildIndex as largest

  swap array[i] and array[largest]
```

To create a Max-Heap:

```
MaxHeap(array, size)
  loop from the first index of non-leaf node down to zero
    call heapify
```

For Min-Heap, both `leftChild` and `rightChild` must be smaller than the parent for all nodes.

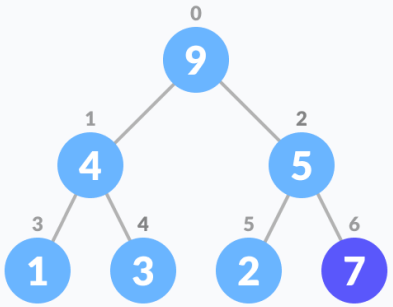
Insert Element into Heap

Algorithm for insertion in Max Heap

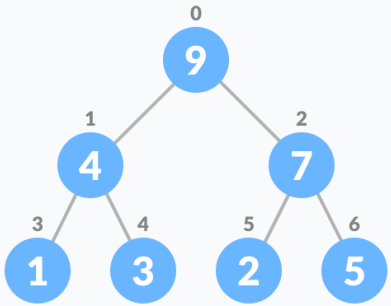
```
else (a node is already present)
  insert the newNode at the end (last node from left to right.)

heapify the array
```

1. Insert the new element at the end of the tree.



2. Heapify the tree.



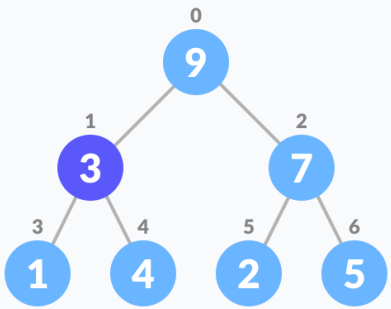
Delete Element from Heap

Algorithm for deletion in Max Heap

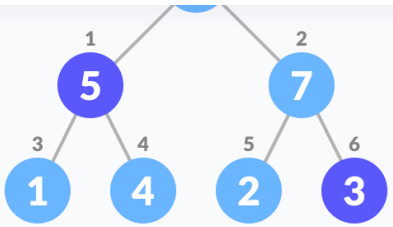
```
If nodeToBeDeleted is the leafNode
  remove the node
Else swap nodeToBeDeleted with the lastLeafNode
  remove noteToBeDeleted

heapify the array
```

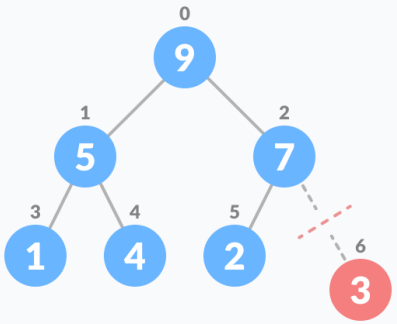
1. Select the element to be deleted.



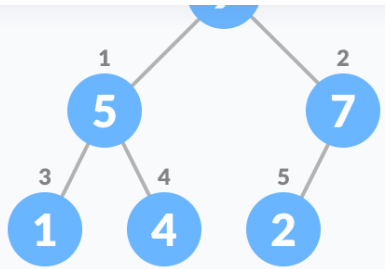
2. Swap it with the last element.



3. Remove the last element.



4. Heapify the tree.



For Min Heap, above algorithm is modified so that both `childNodes` are greater smaller than `currentNode`.

Peek (Find max/min)

Peek operation returns the maximum element from Max Heap or minimum element from Min Heap without deleting the node.

For both Max heap and Min Heap

```
return rootNode
```

Extract-Max/Min

Min Heap.

Python, Java, C/C++ Examples

Python | Java | C | C++

```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[i] < arr[l]:
        largest = l

    if r < n and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]
        heapify(arr, n, largest)

def insert(array, newNum):
    size = len(array)
    if size == 0:
        array.append(newNum)
    else:
        array.append(newNum);
        for i in range((size//2)-1, -1, -1):
            heapify(array, size, i)

def deleteNode(array, num):
    size = len(array)
    i = 0
```

Heap Data Structure Applications

- Heap is used while implementing a priority queue.

Heap Sort

Previous Tutorial: Hash Table (/dsa/hash-table)

Next Tutorial: Fibonacci Heap (/dsa/fibonacci-heap)

Related Tutorials

DS & Algorithms

Linear Search

(/dsa/linear-search)

DS & Algorithms

Backtracking Algorithm

(/dsa/backtracking-algorithm)

DS & Algorithms

DS & Algorithms

(/dsa/greedy-algorithm)

(/dsa/types-of-queue)