

[Array](#) [Matrix](#) [Strings](#) [Hashing](#) [Linked List](#) [Stack](#) [Queue](#) [Binary Tree](#) [Binary Search](#)

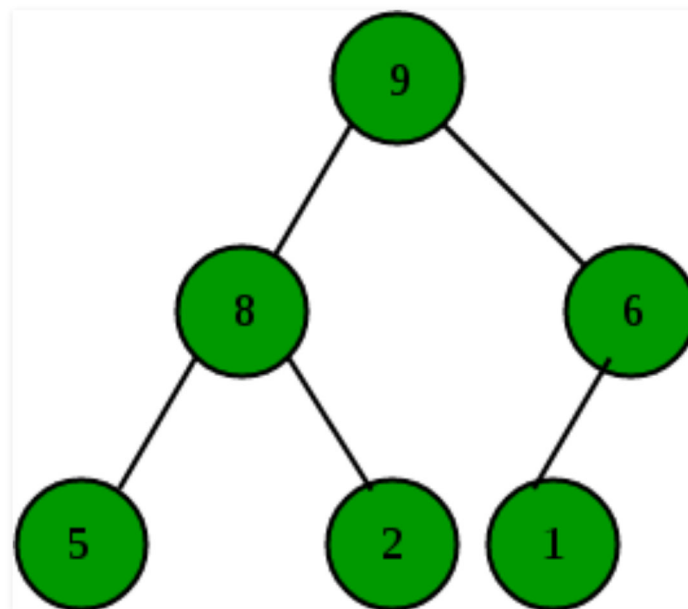
# Max Heap in Python

Difficulty Level : Medium • Last Updated : 10 Sep, 2020

A [Max-Heap](#) is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node.

Mapping the elements of a heap into an array is trivial: if a node is stored at index  $k$ , then its left child is stored at index  $2k + 1$  and its right child at index  $2k + 2$ .

## Examples of Max Heap :



## How is Max Heap is represented ?

A Max Heap is a Complete Binary Tree. A Max heap is typically represented as an array. The root element will be at `Arr[0]`. Below table shows indexes of other nodes for the  $i$ th node, i.e., `Arr[i]`:



`Arr[(i-1)/2]` Returns the parent node.

`Arr[(2*i)+1]` Returns the left child node.

`Arr[(2*i)+2]` Returns the right child node.

## Operations on Max Heap:

1. **getMax()**: It returns the root element of Max Heap. Time Complexity of this operation is  **$O(1)$** .
2. **extractMax()**: Removes the maximum element from MaxHeap. Time Complexity of this Operation is  **$O(\log n)$**  as this operation needs to maintain the heap property (by calling `heapify()`) after removing root.
3. **insert()**: Inserting a new key takes  **$O(\log n)$**  time. We add a new key at the end of the tree. If new key is smaller than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

**Note :** In below implementation, we do indexing from index 1 to simplify the implementation.

```
# Python3 implementation of Max Heap
import sys

class MaxHeap:

    def __init__(self, maxsize):

        self.maxsize = maxsize
        self.size = 0
        self.Heap = [0] * (self.maxsize + 1)
        self.Heap[0] = sys.maxsize
        self.FRONT = 1

    # Function to return the position of
    # parent for the node currently
    # at pos
    def parent(self, pos):

        return pos // 2

    # Function to return the position of
    # the left child for the node currently
```



```
# at pos
def leftChild(self, pos):

    return 2 * pos

# Function to return the position of
# the right child for the node currently
# at pos
def rightChild(self, pos):

    return (2 * pos) + 1

# Function that returns true if the passed
# node is a leaf node
def isLeaf(self, pos):

    if pos >= (self.size//2) and pos <= self.size:
        return True
    return False

# Function to swap two nodes of the heap
def swap(self, fpos, spos):

    self.Heap[fpos], self.Heap[spos] = (self.Heap[spos],
                                         self.Heap[fpos])

# Function to heapify the node at pos
def maxHeapify(self, pos):

    # If the node is a non-leaf node and smaller
    # than any of its child
    if not self.isLeaf(pos):
        if (self.Heap[pos] < self.Heap[self.leftChild(pos)] or
            self.Heap[pos] < self.Heap[self.rightChild(pos)]):

            # Swap with the left child and heapify
            # the left child
            if (self.Heap[self.leftChild(pos)] >
                self.Heap[self.rightChild(pos)]):
                self.swap(pos, self.leftChild(pos))
                self.maxHeapify(self.leftChild(pos))

            # Swap with the right child and heapify
            # the right child
            else:
                self.swap(pos, self.rightChild(pos))
                self.maxHeapify(self.rightChild(pos))
```



```
# Function to insert a node into the heap
def insert(self, element):

    if self.size >= self.maxsize:
        return
    self.size += 1
    self.Heap[self.size] = element

    current = self.size

    while (self.Heap[current] >
           self.Heap[self.parent(current)]):
        self.swap(current, self.parent(current))
        current = self.parent(current)

# Function to print the contents of the heap
def Print(self):

    for i in range(1, (self.size // 2) + 1):
        print(" PARENT : " + str(self.Heap[i]) +
              " LEFT CHILD : " + str(self.Heap[2 * i]) +
              " RIGHT CHILD : " + str(self.Heap[2 * i + 1]))

# Function to remove and return the maximum
# element from the heap
def extractMax(self):

    popped = self.Heap[self.FRONT]
    self.Heap[self.FRONT] = self.Heap[self.size]
    self.size -= 1
    self.maxHeapify(self.FRONT)

    return popped

# Driver Code
if __name__ == "__main__":

    print('The maxHeap is ')

    maxHeap = MaxHeap(15)
    maxHeap.insert(5)
    maxHeap.insert(3)
    maxHeap.insert(17)
    maxHeap.insert(10)
    maxHeap.insert(84)
    maxHeap.insert(19)
    maxHeap.insert(6)
    maxHeap.insert(22)
```



```

maxHeap.insert(9)

maxHeap.Print()

print("The Max val is " + str(maxHeap.extractMax()))

```

## Output :

```

The maxHeap is
PARENT : 84 LEFT CHILD : 22 RIGHT CHILD : 19
PARENT : 22 LEFT CHILD : 17 RIGHT CHILD : 10
PARENT : 19 LEFT CHILD : 5 RIGHT CHILD : 6
PARENT : 17 LEFT CHILD : 3 RIGHT CHILD : 9
The Max val is 84

```

## Using Library functions :

We use [heapq](#) class to implement Heaps in Python. By default Min Heap is implemented by this class. But we multiply each value by -1 so that we can use it as MaxHeap.

```

# Python3 program to demonstrate working of heapq

from heapq import heappop, heappush, heapify

# Creating empty heap
heap = []
heapify(heap)

# Adding items to the heap using heappush
# function by multiplying them with -1
heappush(heap, -1 * 10)
heappush(heap, -1 * 30)
heappush(heap, -1 * 20)
heappush(heap, -1 * 400)

# printing the value of maximum element
print("Head value of heap : "+str(-1 * heap[0]))

# printing the elements of the heap
print("The heap elements : ")
for i in heap:
    print(-1 * i, end = ' ')
print("\n")

```



```
element = heappop(heap)

# printing the elements of the heap
print("The heap elements : ")
for i in heap:
    print(-1 * i, end = ' ')
```

### Output :

Head value of heap : 400

The heap elements :

400 30 20 10

The heap elements :

30 10 20

Take the First Byte Of *Python* &  
**Master The Language**

*Beginner Friendly | Self-Paced*

[Learn now](#)



Like 18

Previous

Heap queue (or heapq) in  
Python

Next

Min Heap in Python

