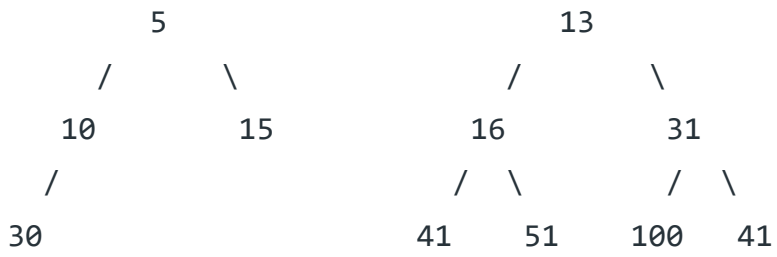
# Min Heap in Python

Difficulty Level : Medium • Last Updated : 19 Feb, 2022

A Min-Heap is a complete binary tree in which the value in each internal node is smaller 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**.

**Example of Min Heap :**




**How is Min Heap represented ?**
A Min Heap is a Complete Binary Tree. A Min heap is typically represented as an array. The root element will be at **Arr[0]**. For any ith node, i.e., **Arr[i]**:

- **Arr[(i -1) / 2]** returns its parent node.
- **Arr[(2 * i) + 1]** returns its left child node.
- **Arr[(2 * i) + 2]** returns its right child node.

**Operations on Min Heap :**

1. **getMin()**: It returns the root element of Min Heap. Time

Complexity of this operation is **O(1)**.

2. **extractMin()**: Removes the minimum element from MinHeap. 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 larger than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

Below is the implementation of Min Heap in Python –

---

## Python3

```
# Python3 implementation of Min Heap

import sys

class MinHeap:

    def __init__(self, maxsize):
        self.maxsize = maxsize
        self.size = 0
        self.Heap = [0]*(self.maxsize + 1)
        self.Heap[0] = -1 * 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):
        return pos*2 > self.size

    # Function to swap two nodes of the heap
    def swap(self, fpos, spos):
        self.Heap[fpos], self.Heap[spos] = self.Heap[spos], self.Heap[

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

        # If the node is a non-leaf node and greater
        # 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.rig
                    self.swap(pos, self.leftChild(pos))
                    self.minHeapify(self.leftChild(pos))

                # Swap with the right child and heapify
                # the right child
                else:
                    self.swap(pos, self.rightChild(pos))
                    self.minHeapify(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 build the min heap using
    # the minHeapify function
    def minHeap(self):

        for pos in range(self.size//2, 0, -1):
            self.minHeapify(pos)

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

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

# Driver Code
if __name__ == "__main__":

    print('The minHeap is ')
    minHeap = MinHeap(15)
    minHeap.insert(5)
    minHeap.insert(3)
    minHeap.insert(17)
    minHeap.insert(10)
    minHeap.insert(84)
    minHeap.insert(19)
    minHeap.insert(6)
    minHeap.insert(22)
    minHeap.insert(9)
    minHeap.minHeap()

    minHeap.Print()
    print("The Min val is " + str(minHeap.remove()))
```

## Output :

```
The Min Heap is
 PARENT : 3 LEFT CHILD : 5 RIGHT CHILD :6
 PARENT : 5 LEFT CHILD : 9 RIGHT CHILD :84
 PARENT : 6 LEFT CHILD : 19 RIGHT CHILD :17
```

```
  PARENT : 9 LEFT CHILD : 22 RIGHT CHILD :10
 The Min val is 3
```

## Using Library functions :

We use heapq class to implement Heaps in Python. By default Min Heap is implemented by this class.

---

## Python3

```
# Python3 program to demonstrate working of heapq

from heapq import heapify, heappush, heappop

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

# Adding items to the heap using heappush function
heappush(heap, 10)
heappush(heap, 30)
heappush(heap, 20)
heappush(heap, 400)

# printing the value of minimum element
print("Head value of heap : "+str(heap[0]))

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

element = heappop(heap)

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

**Output :**

```
Head value of heap : 10
The heap elements :
10 30 20 400

The heap elements :
20 30 400
```





Previous
Max Heap in Python

Next
Merge Sort

RECOMMENDED ARTICLES Page : 1 2 3

01 Difference between Binary Heap, Binomial Heap and Fibonacci Heap
02, Nov 20

05 Given level order traversal of a Binary Tree, check if the Tree is a Min-Heap
08, May 17