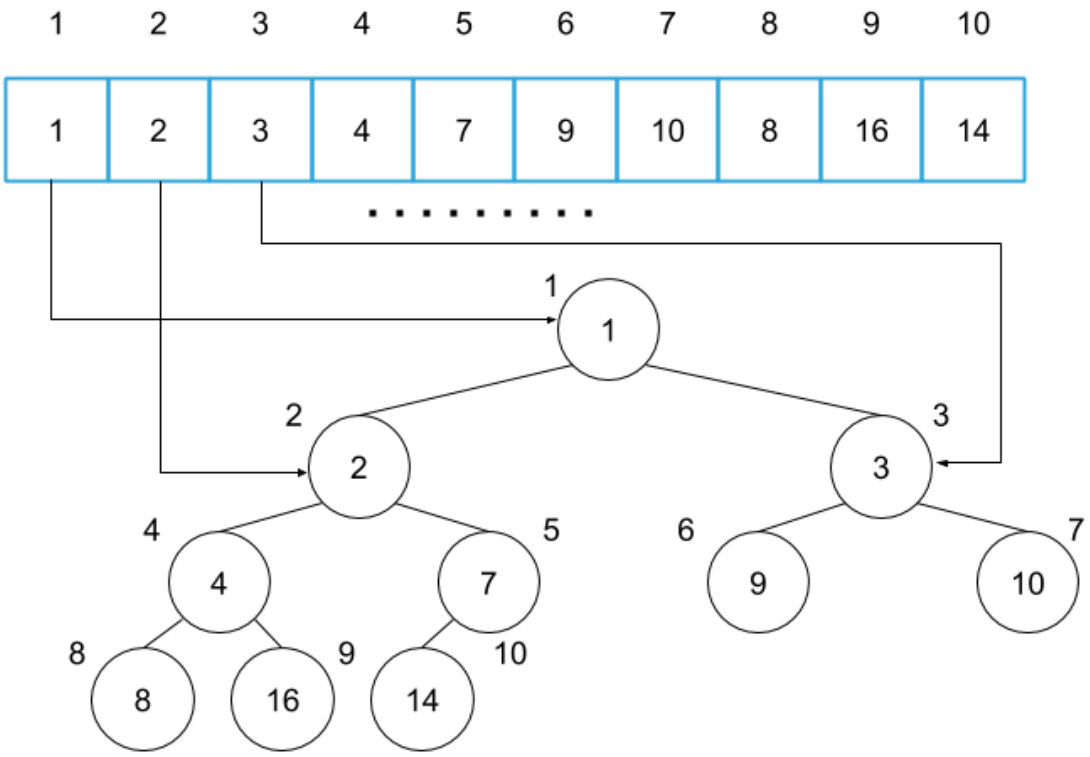
Heap

April 1, 2021

1 HEAP

1.1 A heap is one of the tree structures and represented as a binary tree.



A root node | $i = 1$, the first item of the array

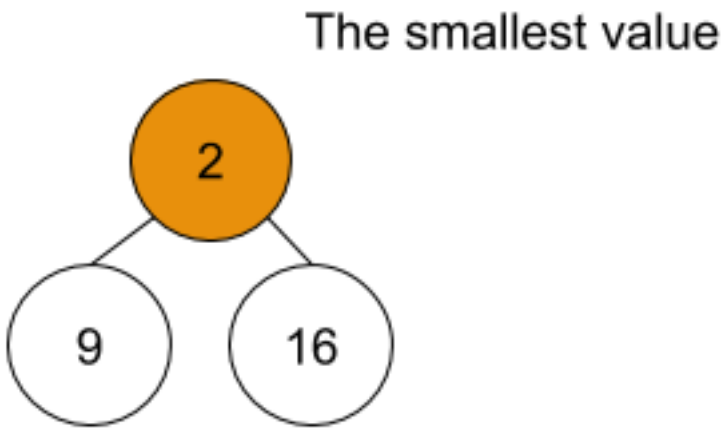
A parent node | $\text{parent}(i) = i / 2$

A left child node | $\text{left}(i) = 2i$

A right child node | $\text{right}(i) = 2i + 1$

Array Representation

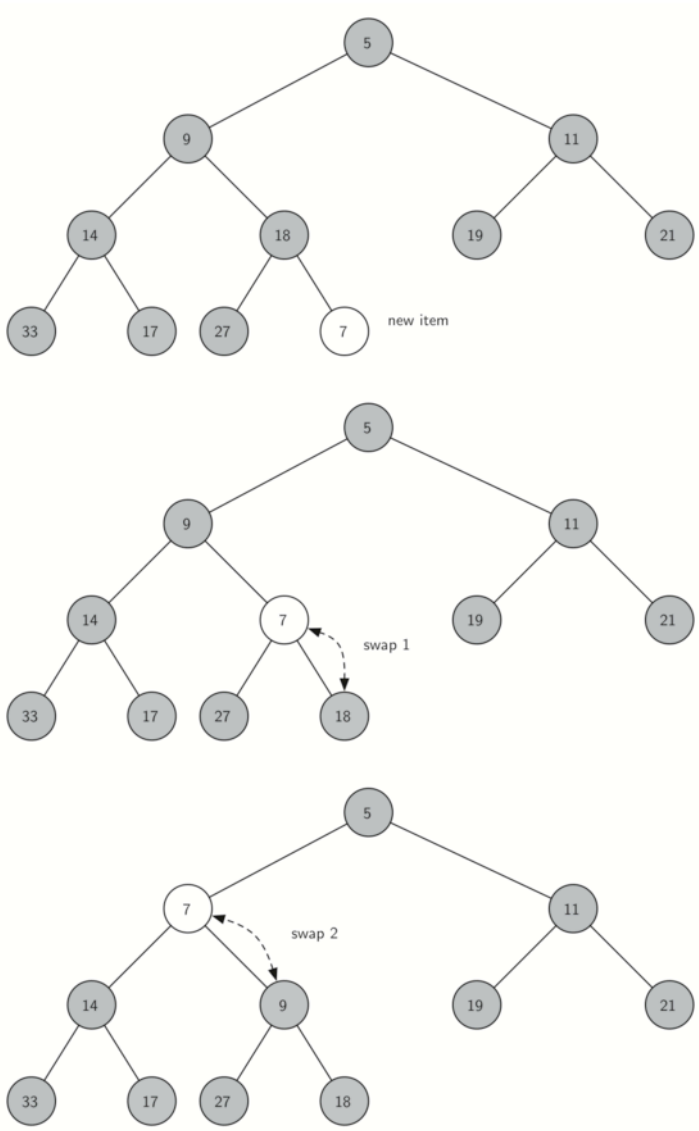
Min Heap each value of nodes is less than or equal to the value of child nodes.



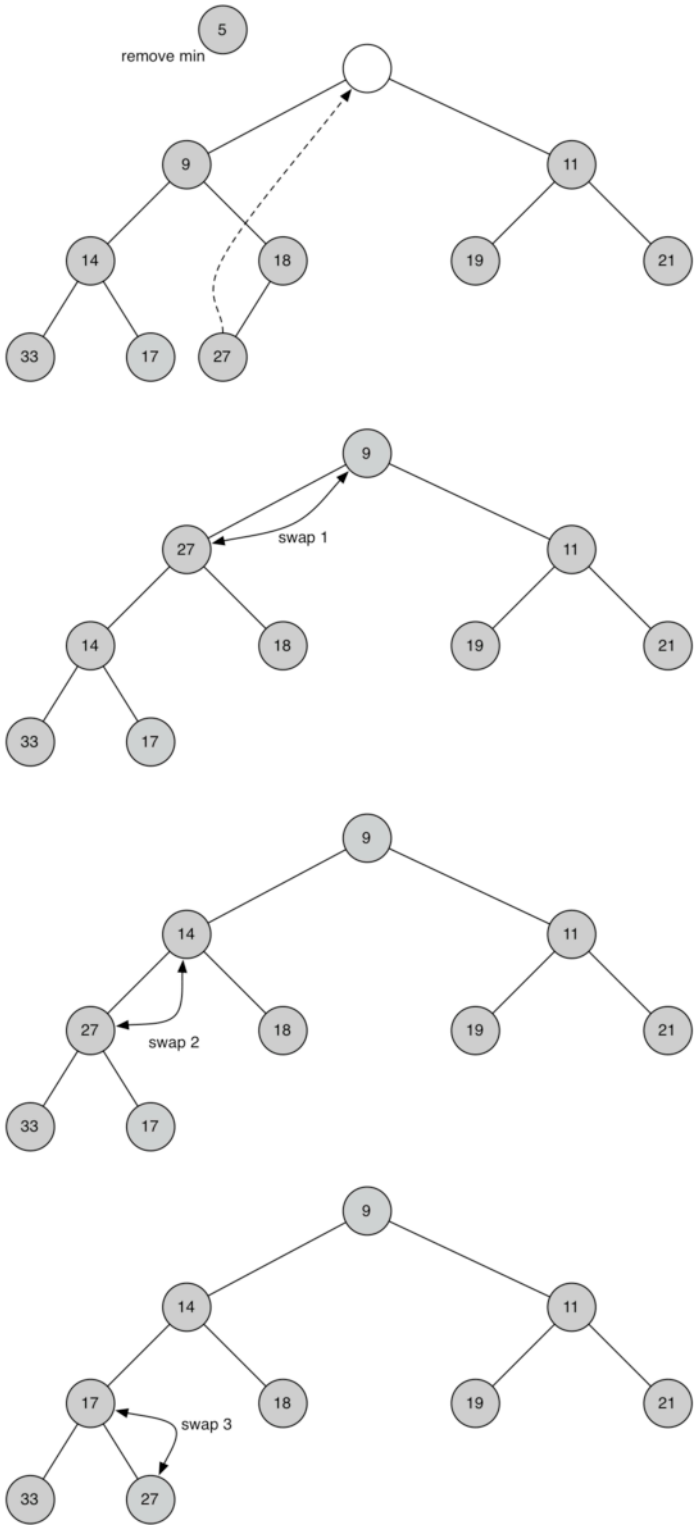
Basic Operations on Heap

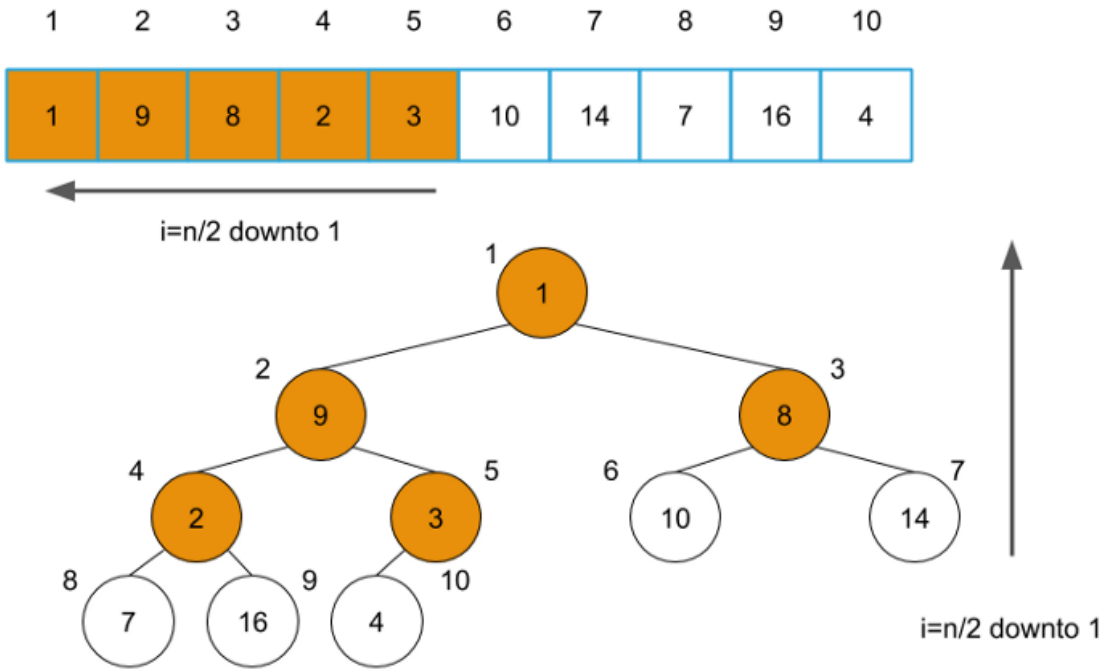
- Heapify/build_heap: create a heap out of given array of elements.
- insert adding a new key to the heap
- delete delete the root.
- siftup Move a node up in the tree to restore heap property.
- siftdown Move a node down in the tree to restore heap property.
- size, is_empty, find_max, find_min

1.2 Insert Operation



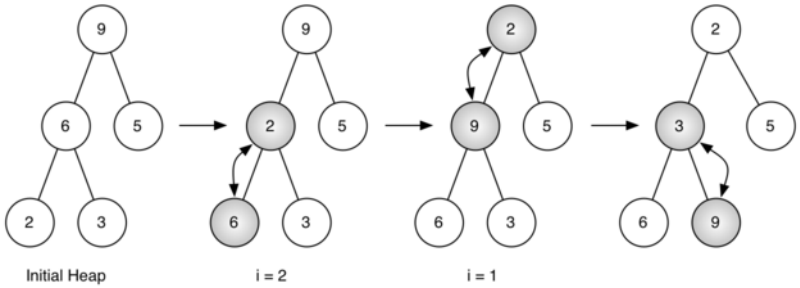
1.3 Delete Operation -Min heap





N/2 logic

1.4 Build heap



[ ]:
```
class Heap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0


    def siftUp(self,i):
        while i // 2 > 0:
            if self.heapList[i] < self.heapList[i // 2]:
                tmp = self.heapList[i // 2]
                self.heapList[i // 2] = self.heapList[i]
                self.heapList[i] = tmp
            i = i // 2
```

```
def insert(self,k):
  self.heapList.append(k)
  self.currentSize = self.currentSize + 1
  self.siftUp(self.currentSize)

def siftDown(self,i):
  while (i * 2) <= self.currentSize:
      mc = self.minChild(i)
      if self.heapList[i] > self.heapList[mc]:
          tmp = self.heapList[i]
          self.heapList[i] = self.heapList[mc]
          self.heapList[mc] = tmp
      i = mc

def minChild(self,i):
  if i * 2 + 1 > self.currentSize:
      return i * 2
  else:
      if self.heapList[i*2] < self.heapList[i*2+1]:
          return i * 2
      else:
          return i * 2 + 1

def delMin(self):
    if len(self.heapList) == 1:
        return 'Empty heap'

    retval = self.heapList[1]
    self.heapList[1] = self.heapList[self.currentSize]
    self.currentSize = self.currentSize - 1
    self.heapList.pop()
    self.siftDown(1)
    return retval

def buildHeap(self,alist):
  i = len(alist) // 2
  self.currentSize = len(alist)
  self.heapList = [0] + alist[:]
  while (i > 0):
      self.siftDown(i)
      i = i - 1

def printMinHeap(self):
    for val in range(1,len(self.heapList)):
        print(self.heapList[val])
```

```
```

[ ]:
```
h = Heap()
h.buildHeap([9,5,6,2,3])
```

[ ]:
```
print(h.delMin())
```

[ ]:
```
print(h.delMin())
```

[ ]:
```
h.printMinHeap()
```

Reference