



# 数据结构与算法（Python版）

## 二叉堆的Python实现

陈斌 北京大学 [gischen@pku.edu.cn](mailto:gischen@pku.edu.cn)

# 二叉堆操作的实现

## ❖ 二叉堆初始化

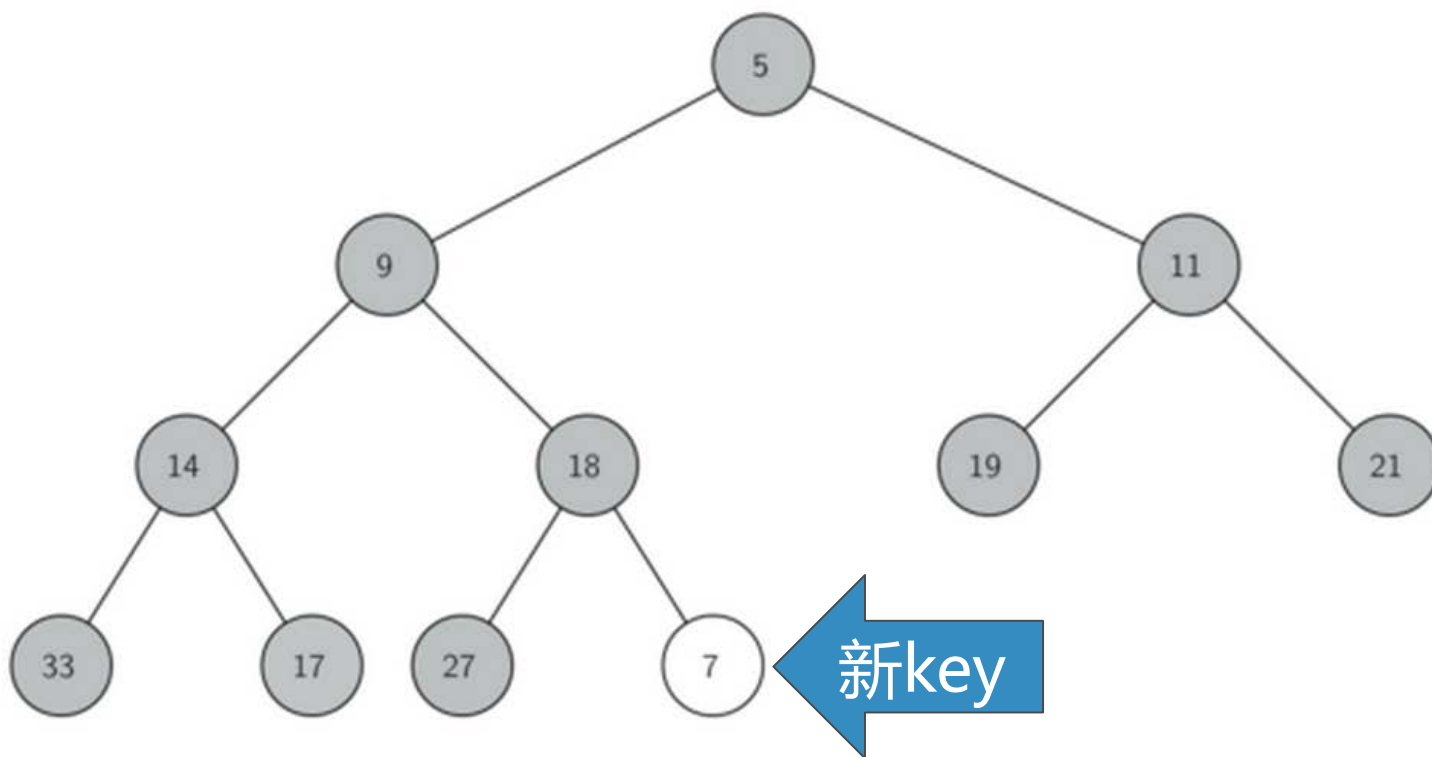
采用一个列表来保存堆数据，其中表首下标为0的项无用，但为了后面代码可以用到简单的整数乘除法，仍保留它。

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

# 二叉堆操作的实现

## ❖ insert(key)方法

首先，为了保持“完全二叉树”的性质，新key应该添加到列表末尾。会有问题吗？

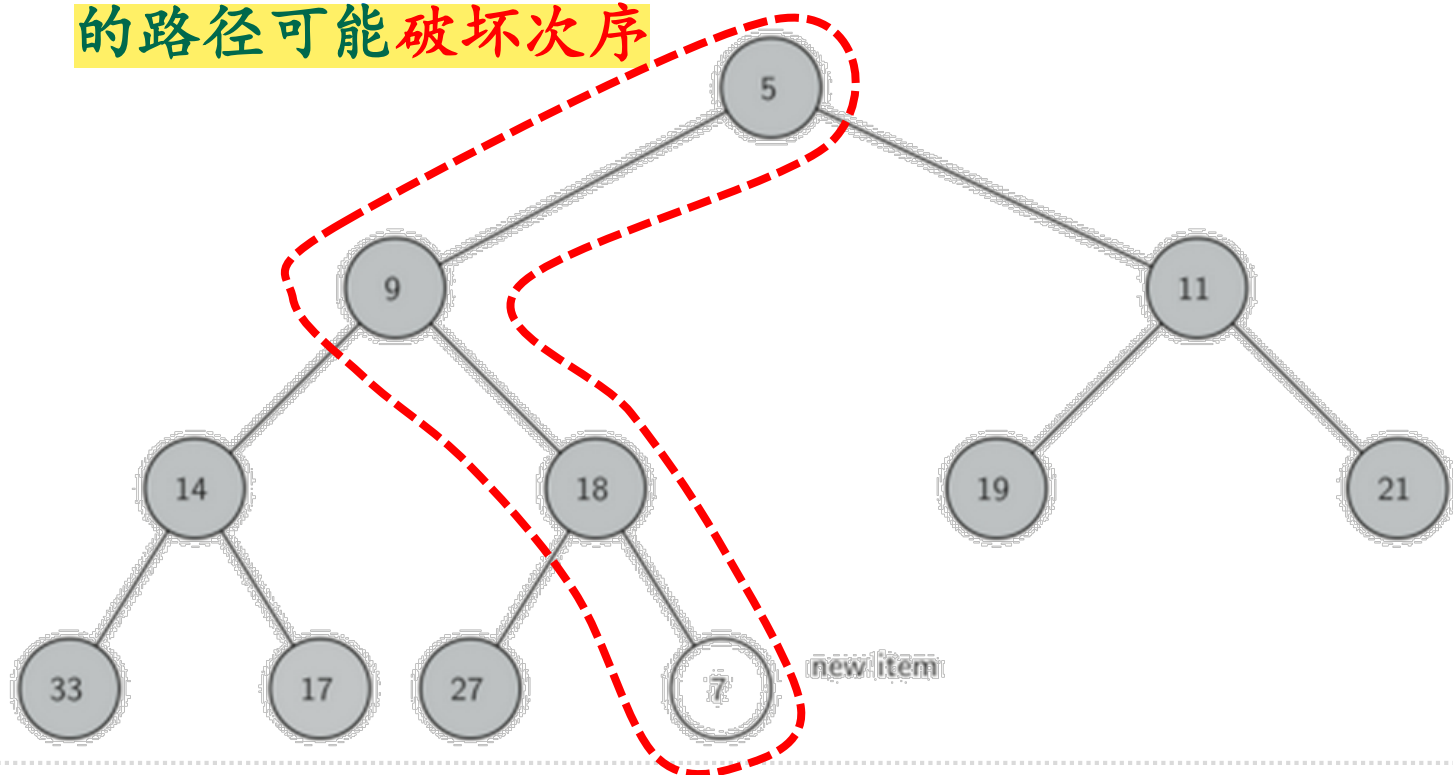


# 二叉堆操作的实现

## ❖ insert(key)方法

新key加在列表末尾，显然无法保持“堆”次序

虽然对其它路径的次序没有影响，但对于其到根的路径可能破坏次序

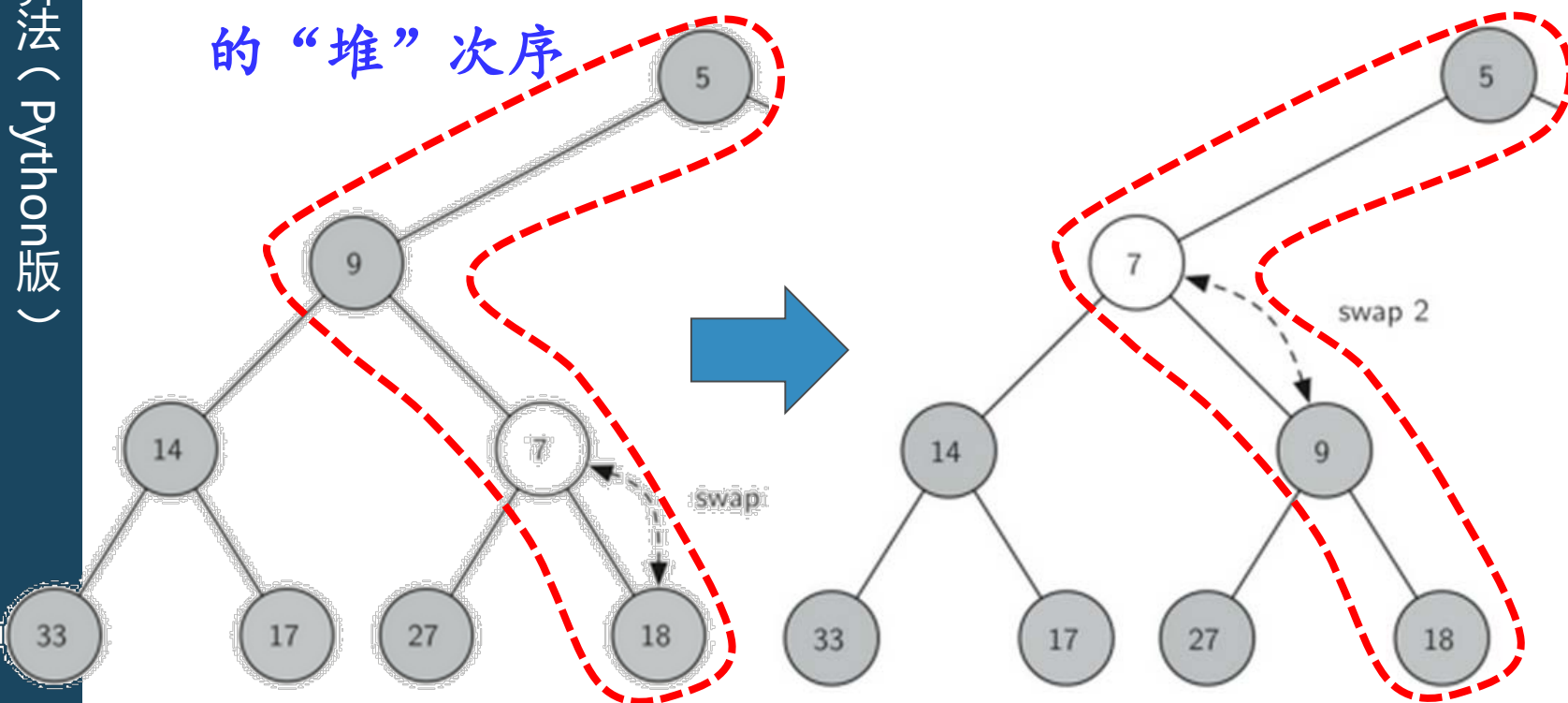


# 二叉堆操作的实现

## ❖ insert(key)方法

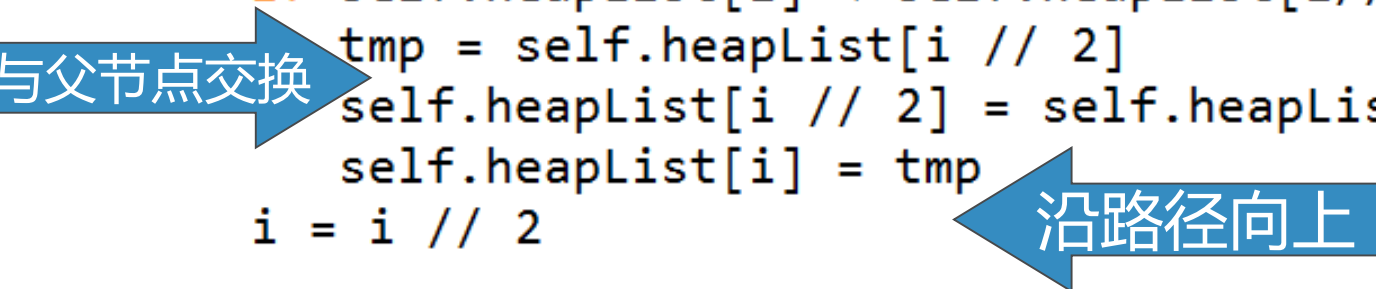
需要将新key沿着路径来“上浮”到其正确位置

注意：新key的“上浮”不会影响其它路径节点的“堆”次序

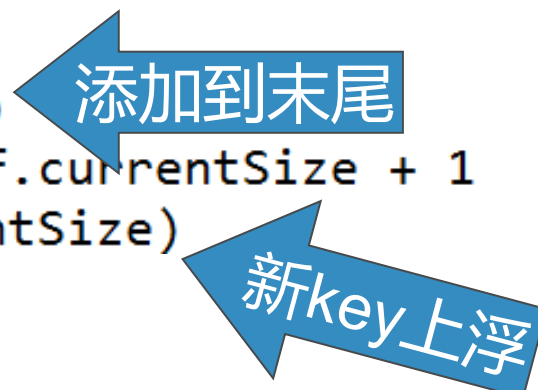


# 二叉堆操作的实现：insert代码

```
def percUp(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.percUp(self.currentSize)
```





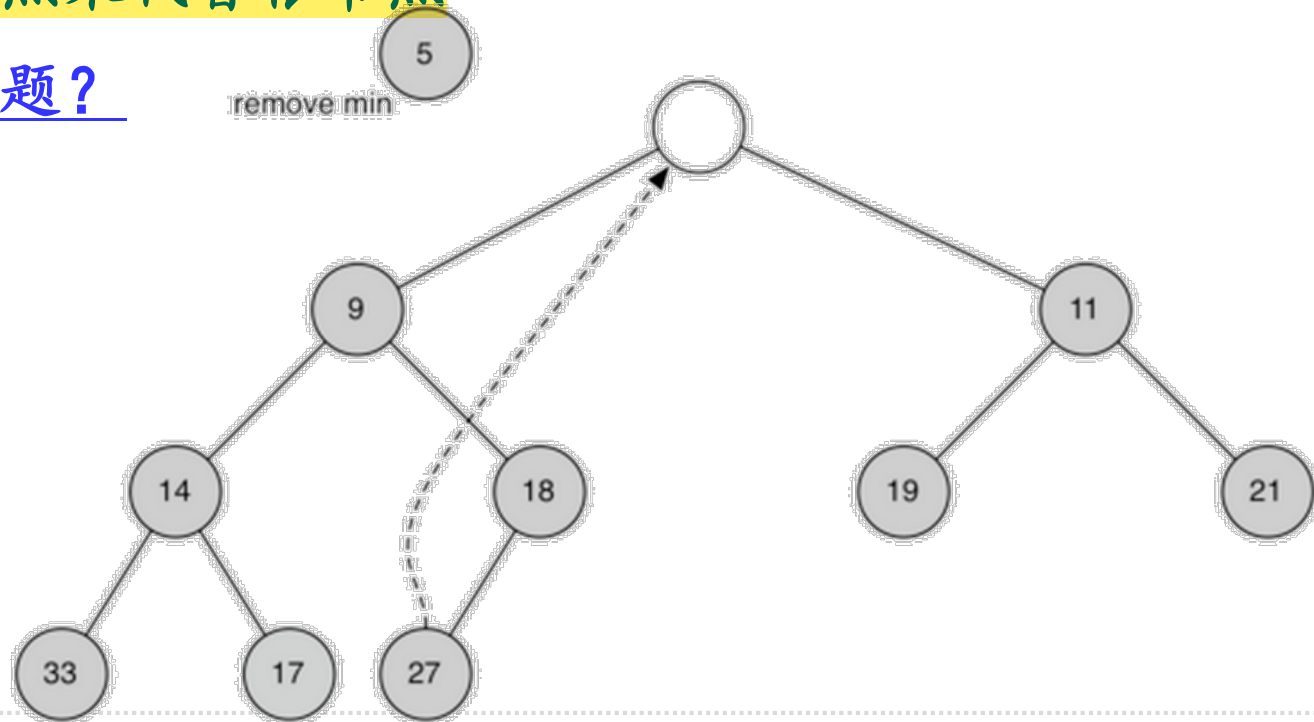
# 二叉堆操作的实现

## ❖ delMin()方法

移走整个堆中最小的key: 根节点heapList[1]

为了保持“完全二叉树”的性质, 只用最后一个节点来代替根节点

问题?

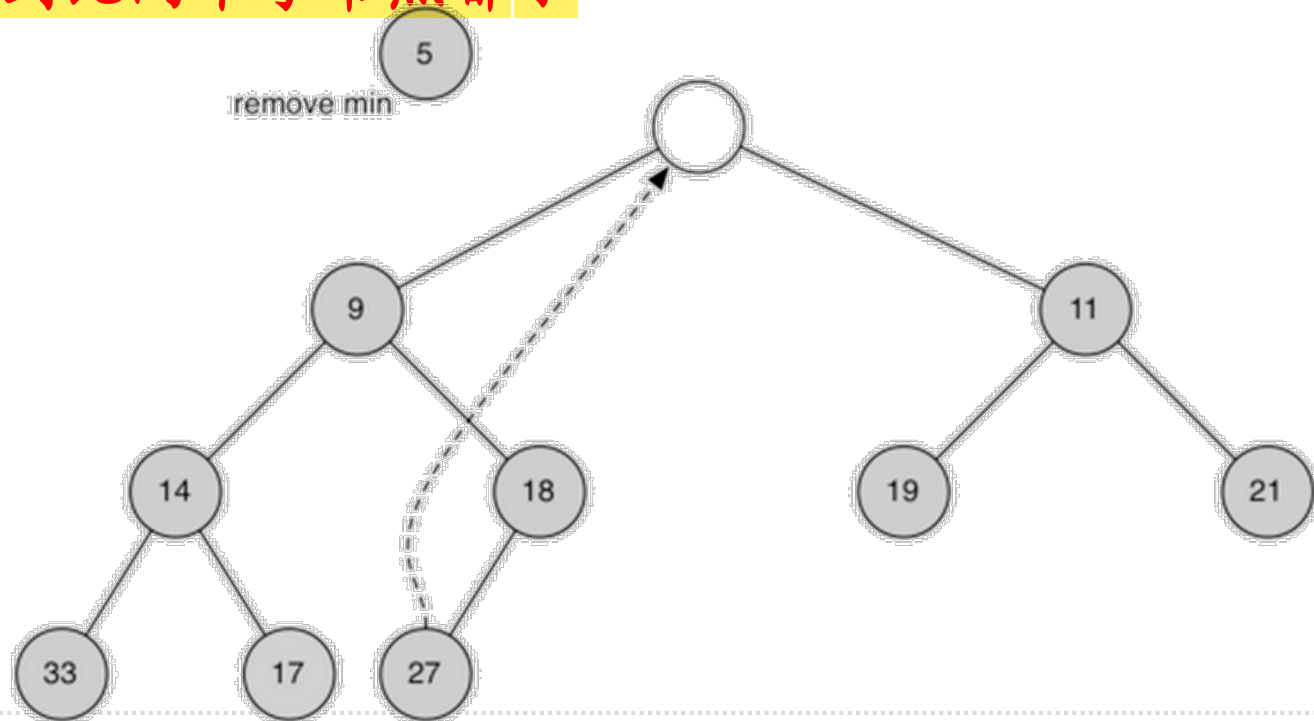


# 二叉堆操作的实现

## ❖ delMin()方法

同样，这么简单的替换，还是破坏了“堆”次序

解决方法：将新的根节点沿着一条路径“下沉”，直到比两个子节点都小

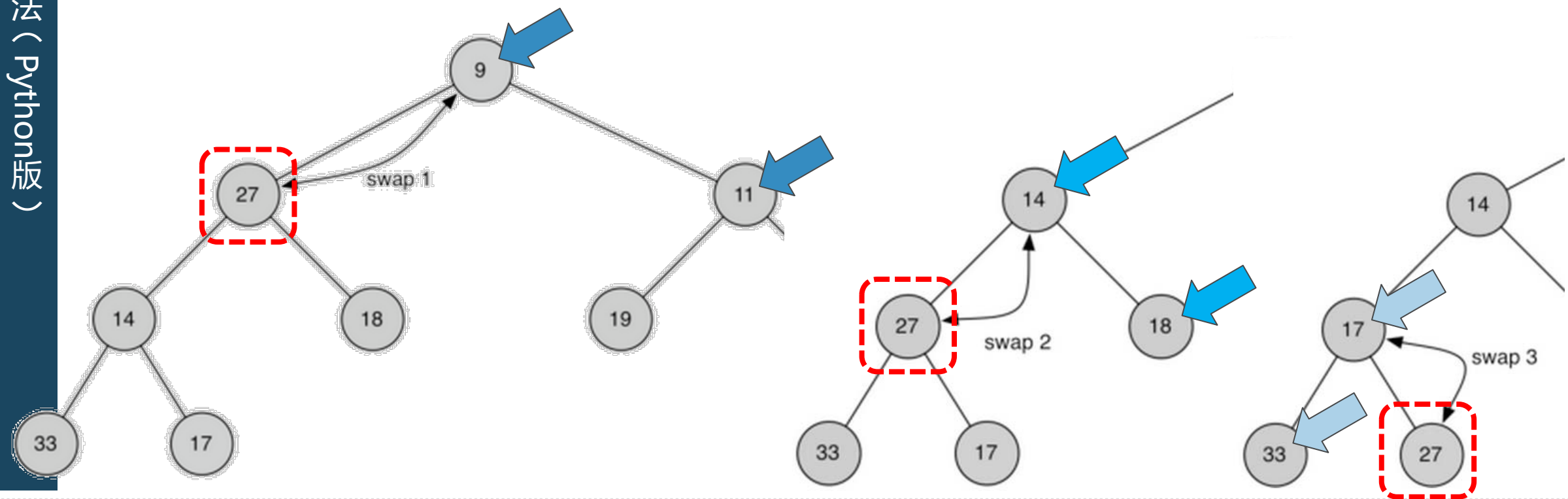




# 二叉堆操作的实现

## ❖ delMin()方法

“下沉”路径的选择：如果比子节点大，那么选择较小的子节点交换下沉



```
def percDown(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
```

交换下沉

沿路径向下

```
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
```

唯一子节点

返回较小的

```
def delMin(self):
    retval = self.heapList[1]
    self.heapList[1] = self.heapList[self.currentSize]
    self.currentSize = self.currentSize - 1
    self.heapList.pop()
    self.percDown(1)
    return retval
```

移走堆顶

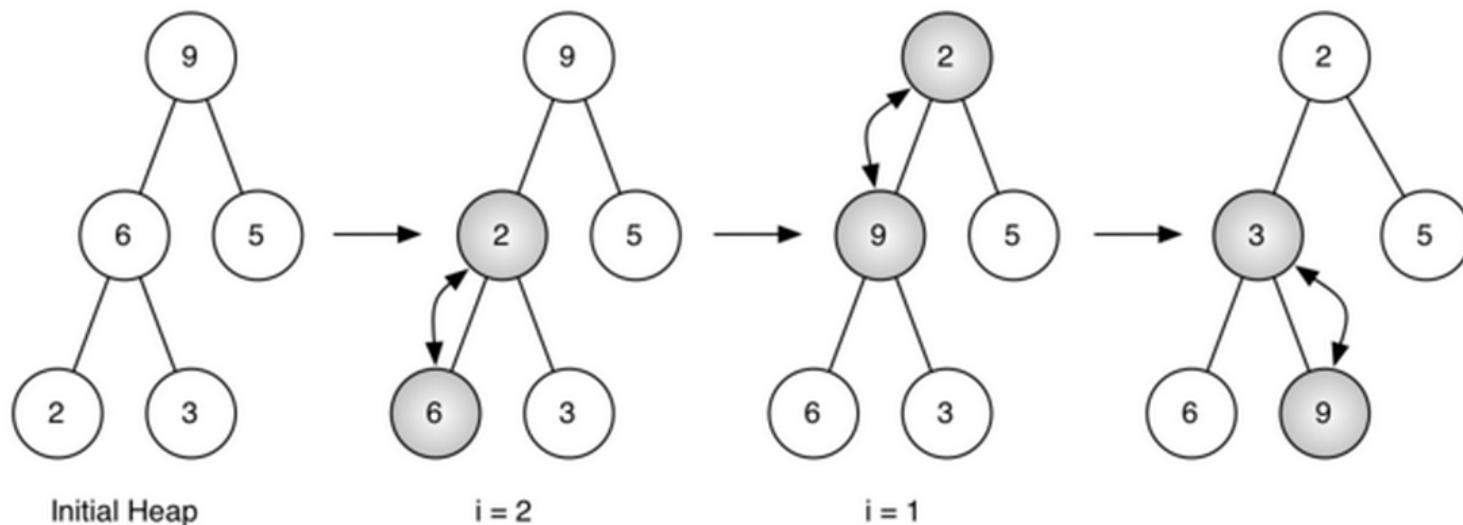
新顶下沉

# 二叉堆操作的实现

## ❖ buildHeap(lst)方法：从无序表生成“堆”

我们最自然的想法是：用insert(key)方法，将无序表中的数据项逐个insert到堆中，但这么做的总代价是 $O(n \log n)$

其实，用“下沉”法，能够将总代价控制在 $O(n)$



# 二叉堆操作的实现

## ❖ buildHeap(lst)方法：从无序表生成“堆”

其实，用“下沉”法，能够将总代价控制在 $O(n)$

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

# 二叉堆操作的实现

❖ 思考：利用二叉堆来进行排序？

“堆排序”算法： $O(n \log n)$

