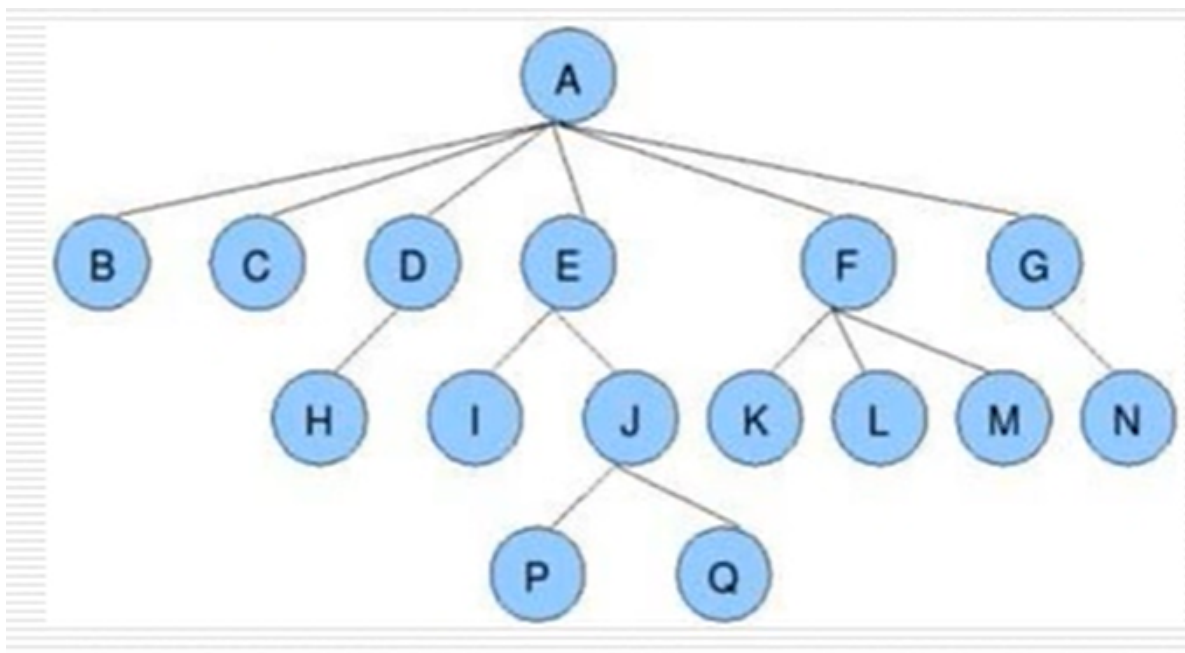


## 4.4 堆排序

### 1. 树的介绍（铺垫知识）

- 树是一种数据结构 比如：目录结构
- 树是一种可以递归定义的数据结构
- 树是由 $n$ 个节点组成的集合：
  - 如果 $n=0$ , 那这是一棵空树;
  - 如果 $n>0$ , 那存在一个节点作为树的根节点, 其他节点可以分为 $m$ 个集合, 每个集合本身又是一棵树。
- 一些概念
  - 根节点 (A), 叶子节点 (BCHIPQKLMN)
  - 树的深度 (高度) 4 (A-E-J-P)
  - 树的度6 (A有BCDEFG, 最广的)
  - 孩子节点/父节点 (B相对于A)
  - 子树 (比如以D为根的子树)

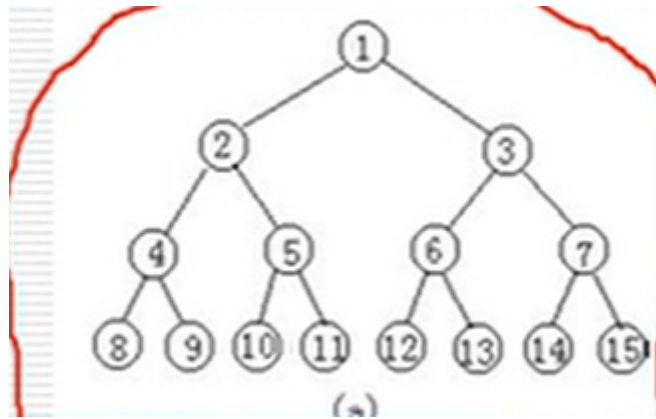


#### 1.1 特殊且常用的树-- 二叉树

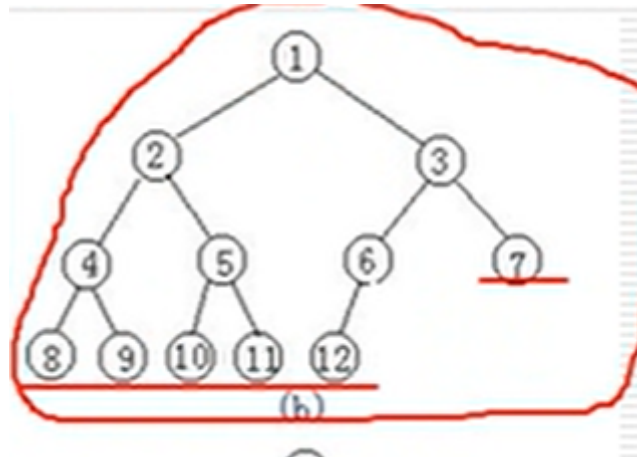
- 二叉树：度不超过2的树（节点最多有两个叉）

##### 两种特殊的二叉树

- 满二叉树：一个二叉树，如果每一个层的节点数都达到最大值，则这个二叉树就是满二叉树。

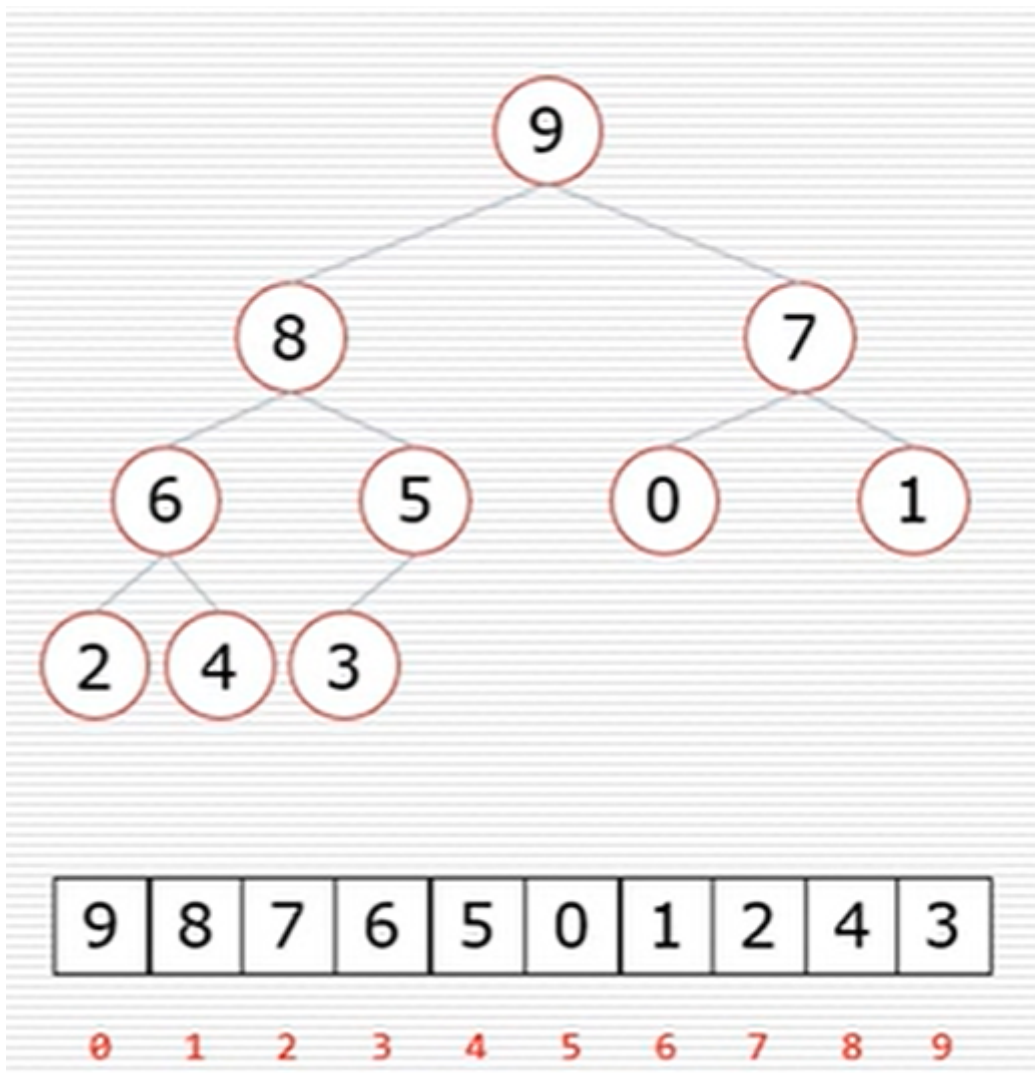


- 完全二叉树：叶节点只能出现在最下层和次下层，并且最下面一层的节点都集中在该层最左边的若干位置的二叉树。



## 1.2 二叉树的存储方式

- 链式存储方式
- 顺序存储方式 (列表)
- 完全二叉树写成列表形式后，父节点位置与孩子节点位置的关系表达式
  - 父节点位置是 $i$ ，左孩子位置是 $2i+1$
  - 父节点位置是 $i$ ，右孩子位置是 $2i+2$
  - 左孩子位置或者右孩子位置是 $i$ ，父节点位置是  $(i-1) // 2$

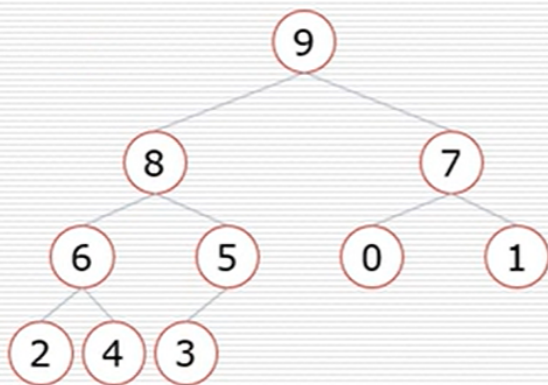


## 2. 堆排序

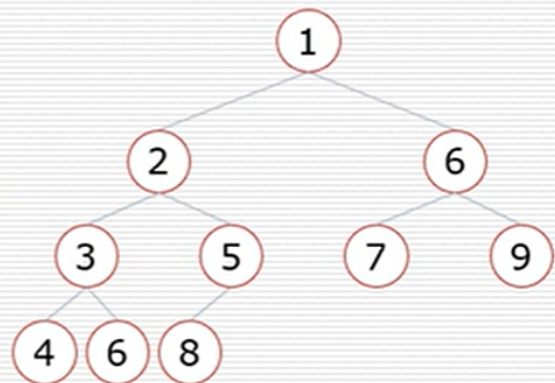
### 2.1 堆（特殊的完全二叉树）

- 大根堆/大顶堆：一棵完全二叉树，满足任一节点都比其孩子节点大
- 小根堆/小顶堆：一棵完全二叉树，满足任一节点都比其孩子节点小

大根堆：



小根堆：

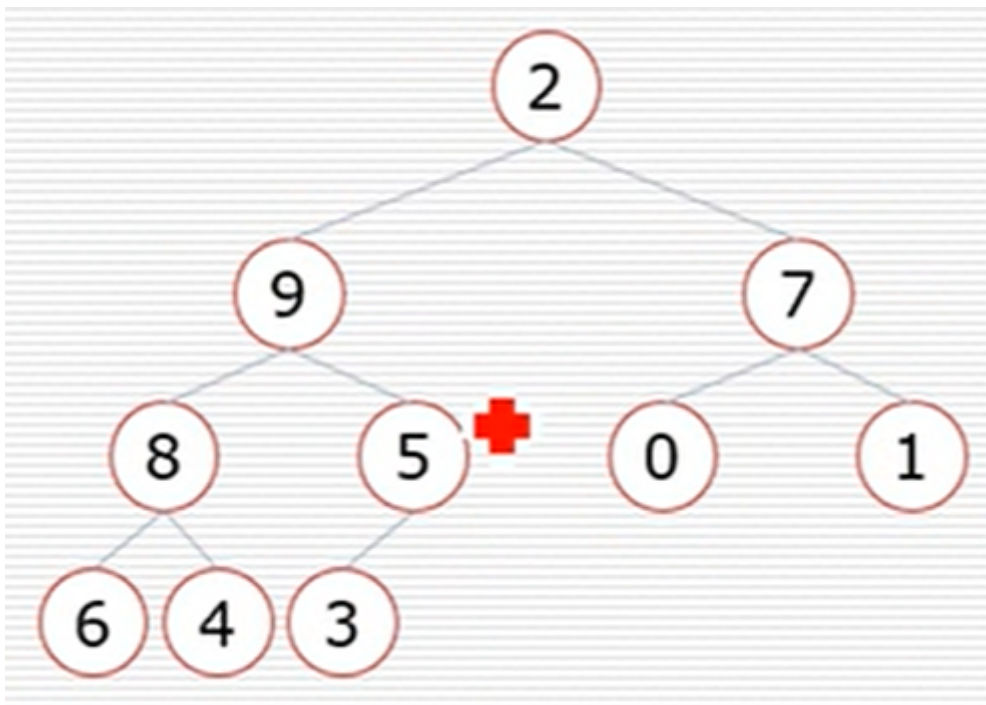


### 2.2 堆的向下调整性质

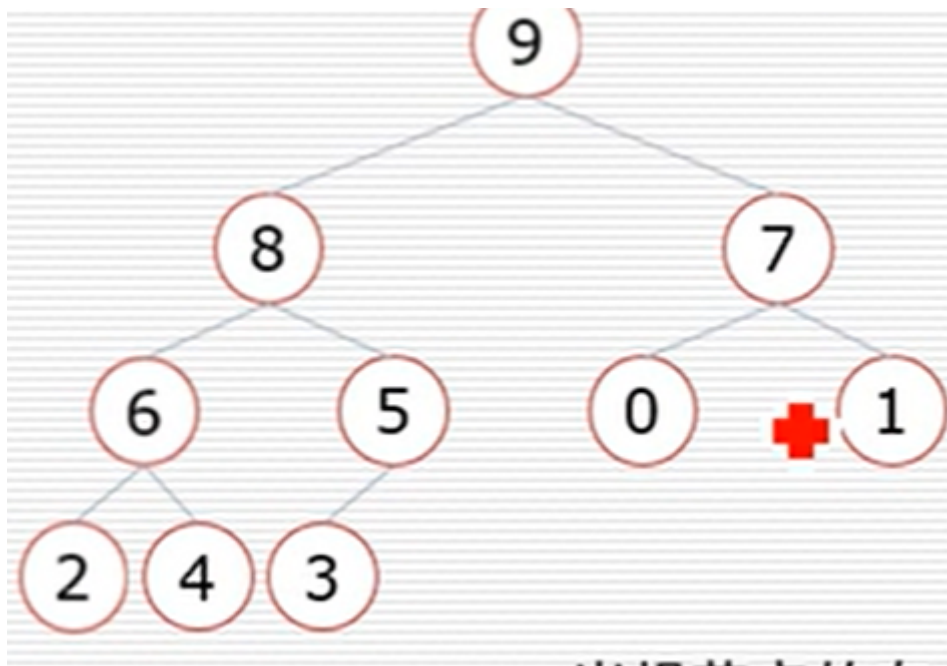
- 假设：节点的左右子树都是堆，但自身不是堆
- 当根节点的左右子树都是堆时，可以通过一次向下的调整来将其变换成一个堆。

- 具体过程，因为2比（9，7）这一层小，所以取下来放一边。用9代替2的位置。2尝试放入最开始9的位置，但是2比（8，5）小，8放在最开始9的位置，以此类推。。。

变换前：



变换后：

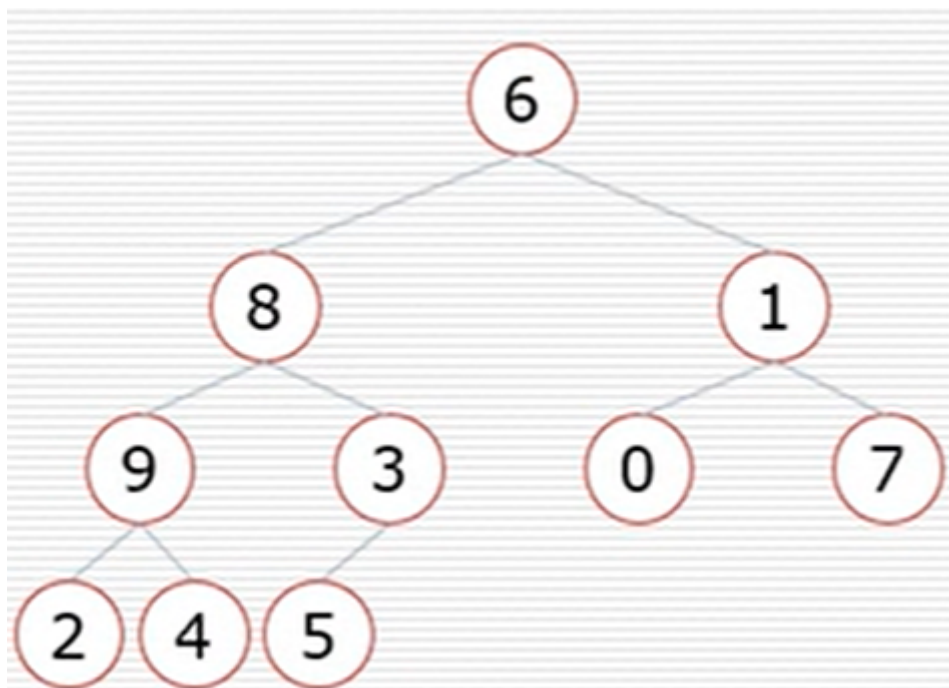


## 2.3 挨个出数

如上图，是一个完整的堆，9先出数，然后找最后一层的最后一个代理9的位置，进行堆的向下调整，调整完后出最顶上的数。以此类推。。。出完所有数。

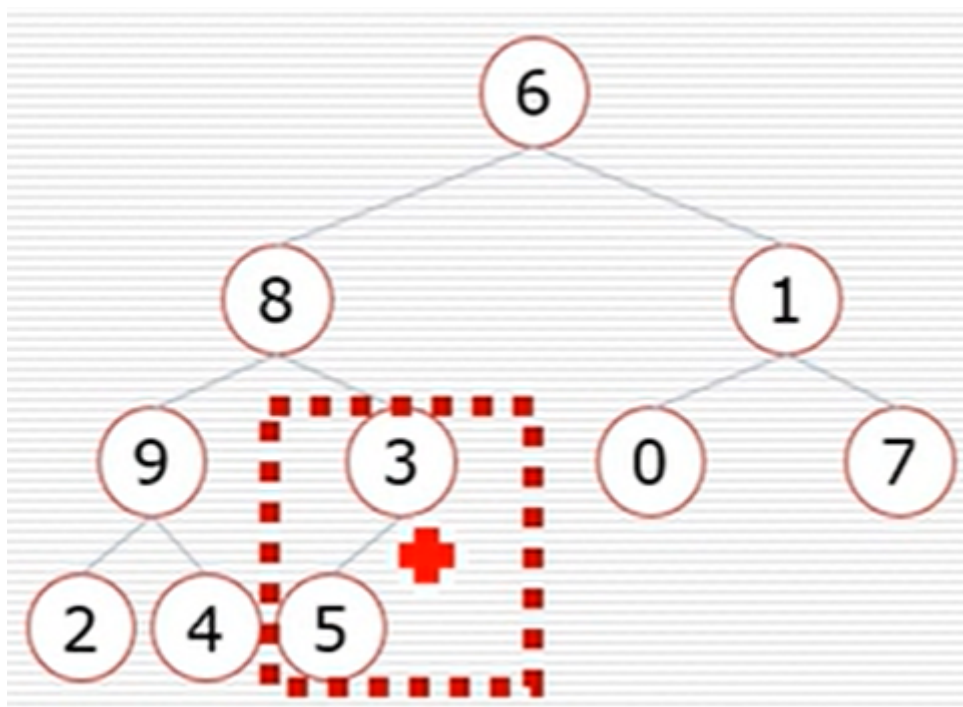
## 2.4 构造堆

最开始的树



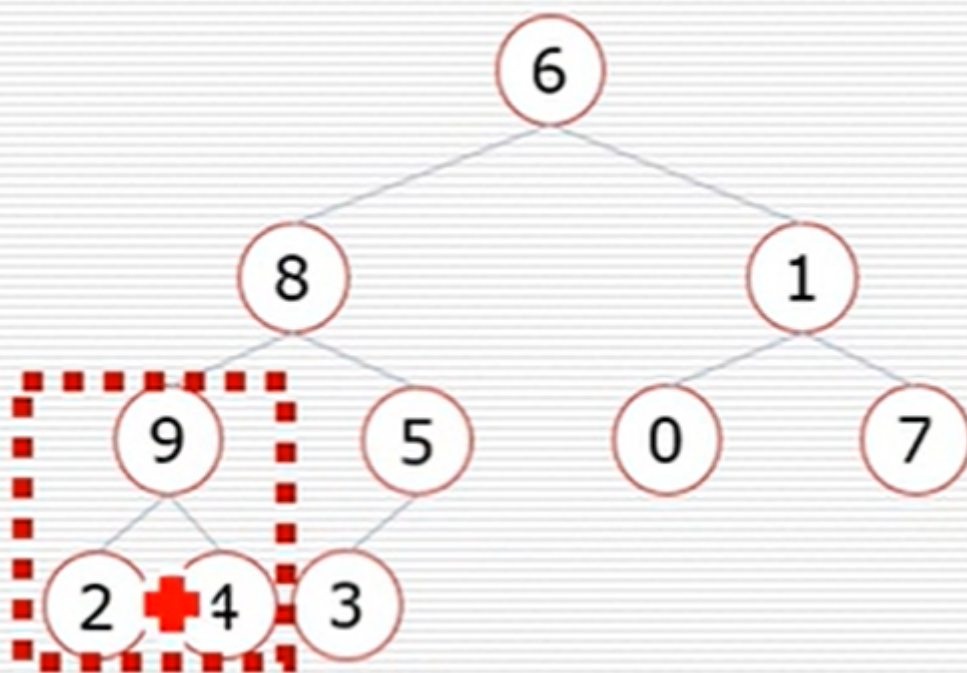
步骤:

1.

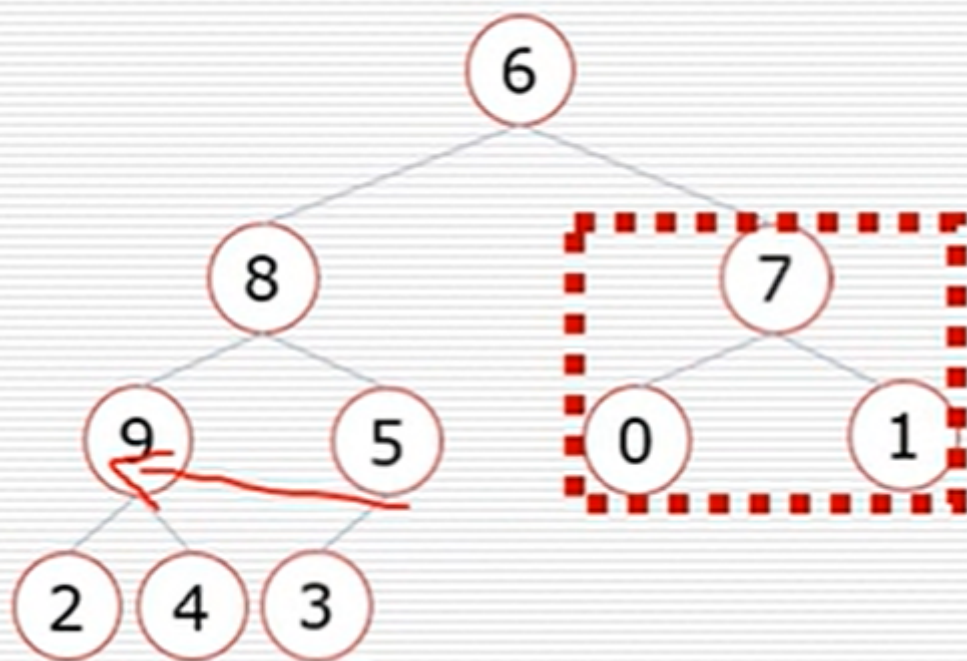




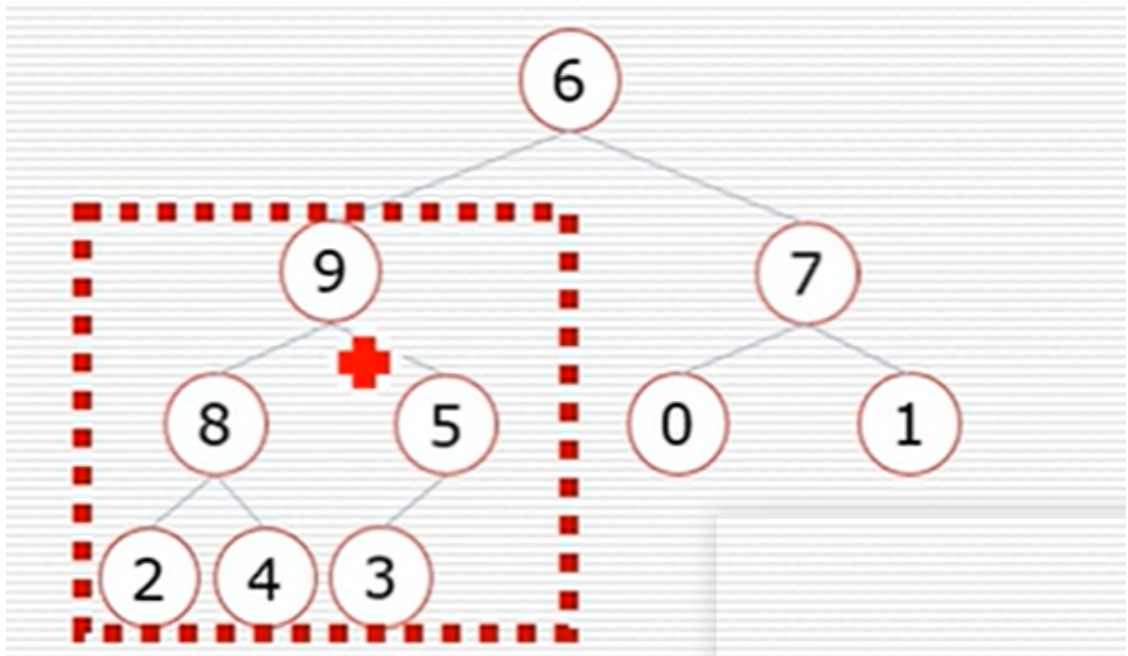
2.



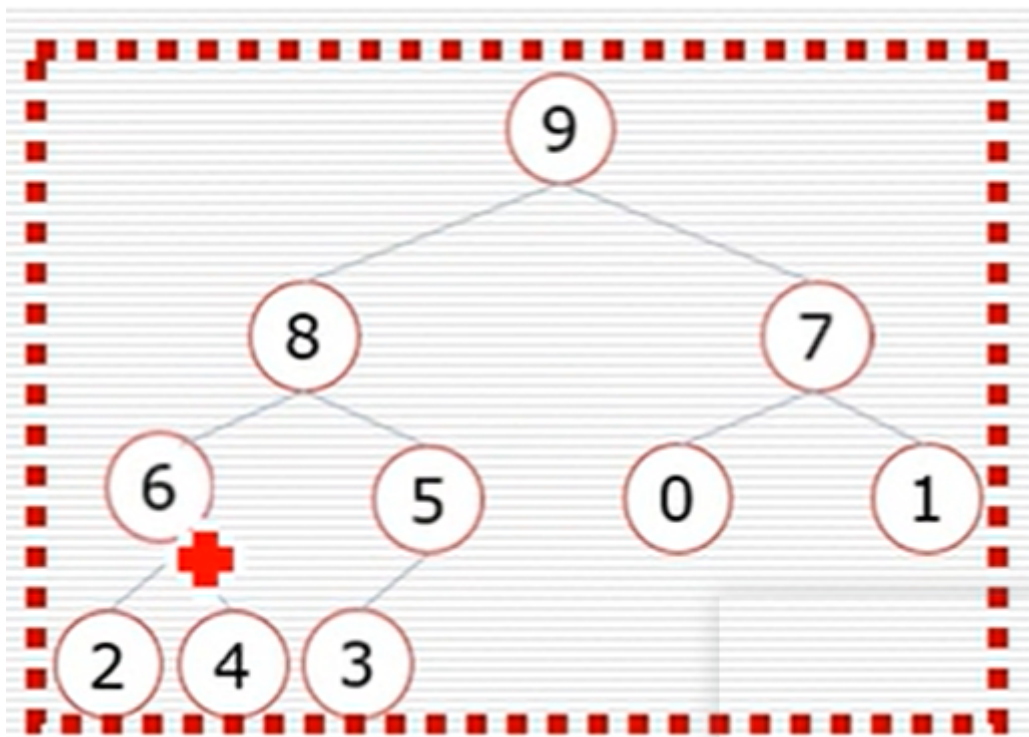
3.



4.



5.



## 2.5 堆排序总结

先运用堆的向下调整构造堆，再运用堆的向下调整挨个出数。

代码中sift () 是堆的向下调整函数，其时间复杂度为 $\log n$ ，heap\_sort()函数的构造堆和挨个出数的时间复杂度都为 $n \log n$ ，所以堆排序的时间复杂度为 $O(n \log n)$

不稳定

最好时间复杂度： $O(n \log n)$

平均时间复杂度： $O(n \log n)$

最坏时间复杂度： $O(n \log n)$

## 2.6 代码

```
import random
```

```

from cal_time import *
def sift(li, low, high):#这是堆的向下调整函数
    # li表示树, low表示树根, high表示树最后一个节点的位置
    tem = li[low]
    i = low
    j = 2*i + 1 #初始j指向空位的左孩子
    # i指向空位, j指向两个孩子
    while j <= high:#循环推退出的第二种情况: j>high,说明空位i是叶子节点
        if j+1 <= high and li[j] < li[j+1]:#如果右孩子存在并且比左孩子大, 指向右孩子
            j += 1
        if li[j] > tem:
            li[i] = li[j]
            i = j
            j = 2*i+1
        else:# 循环退出的第一种情况: j位置的值比tmp小, 说明两个孩子都比tmp小
            break
    li[i] = tem

@cal_time
def heap_sort(li):
    n = len(li)
    # 1. 构造堆
    for low in range(n//2-1, -1, -1):
        sift(li, low, n-1)
    # 2. 挨个出数
    for high in range(n-1, -1, -1):
        li[0], li[high] = li[high], li[0] #退休 棋子
        sift(li, 0, high-1)

li = list(range(100))
random.shuffle(li)
print(li)
heap_sort(li)
print(li)

```

## 2.7 Python内置堆模块

```

import heapq

li = [9,5,7,8,2,6,4,1,3]
heapq.heapify(li)
print(li)
heapq.heappush(li,0)
print(li)
item = heapq.heappop(li)
print(item)
print(li)

```

- 利用heapq内置模块实现堆排序

```

def heapsort(li):
    h = []
    for value in li:#将h构造成堆的列表形式
        heappush(h, value)
    return [heappop(h) for i in range(len(h))]

```



### 3. 堆排序扩展 -- topK问题

---

- 现有 $n$ 个数，设计算法找出前 $K$ 大的数( $k < n$ ).
- 解决方法：（下个笔记）