



数据结构和算法 (Python描述)

郭 炜

微信公众号



微博: <http://weibo.com/guoweiofpku>

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄



堆



北京大学
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

堆的定义、性质 和用途

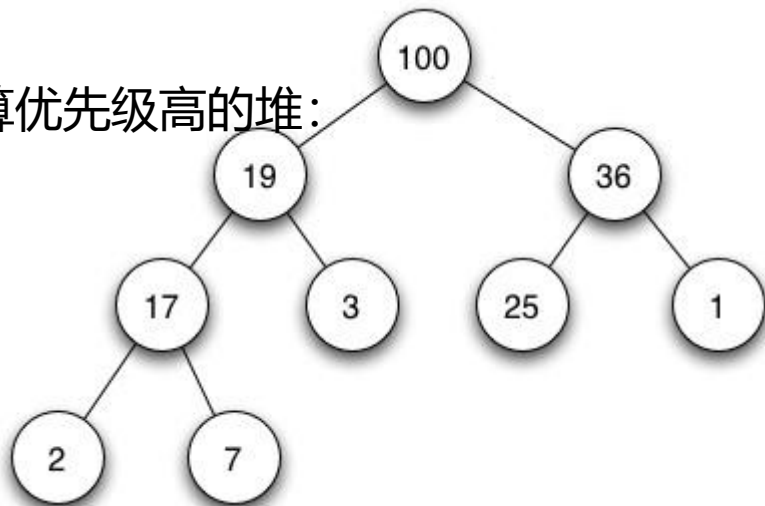


美国鹅颈湾

堆的定义

- 1) 堆(二叉堆)是一个**完全二叉树**
- 2) 堆中任何结点优先级都高于或等于其两个子结点 (什么叫优先级高可以自己定义)

一个“大” 就算优先级高的堆：
(即大根堆)



- 3) 一般将堆顶元素最大的堆称为大根堆 (大顶堆), 堆顶元素最小的堆称为小根堆 (小顶堆)

堆的存储

用列表存放堆。**堆顶元素下标是0**。下标为 i 的结点，其左右子结点下标分别为 $i*2+1$, $i*2 + 2$ 。

堆的性质

- 1) 堆顶元素是优先级最高的(啥叫优先级高可自定义)
- 2) 堆中的任何一棵子树都是堆
- 3) 往堆中添加一个元素, 并维持堆性质, 复杂度 $O(\log(n))$
- 4) 删除堆顶元素, 剩余元素依然维持堆性质, 复杂度 $O(\log(n))$
- 5) 在无序列表中原地建堆, 复杂度 $O(n)$

堆的作用

- 堆用于需要经常从一个集合中**取走**(即删除)优先级最高元素, 而且还要经常往集合中添加元素的场合(堆可以用来实现优先队列)
- 可以用堆进行排序, 复杂度 $O(n\log(n))$, 且只需要 $O(1)$ 的额外空间, 称为“**堆排序**”。递归写法需要 $o(\log(n))$ 额外空间, 非递归写法需要 $O(1)$ 额外空间。



北京大学
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

堆的操作

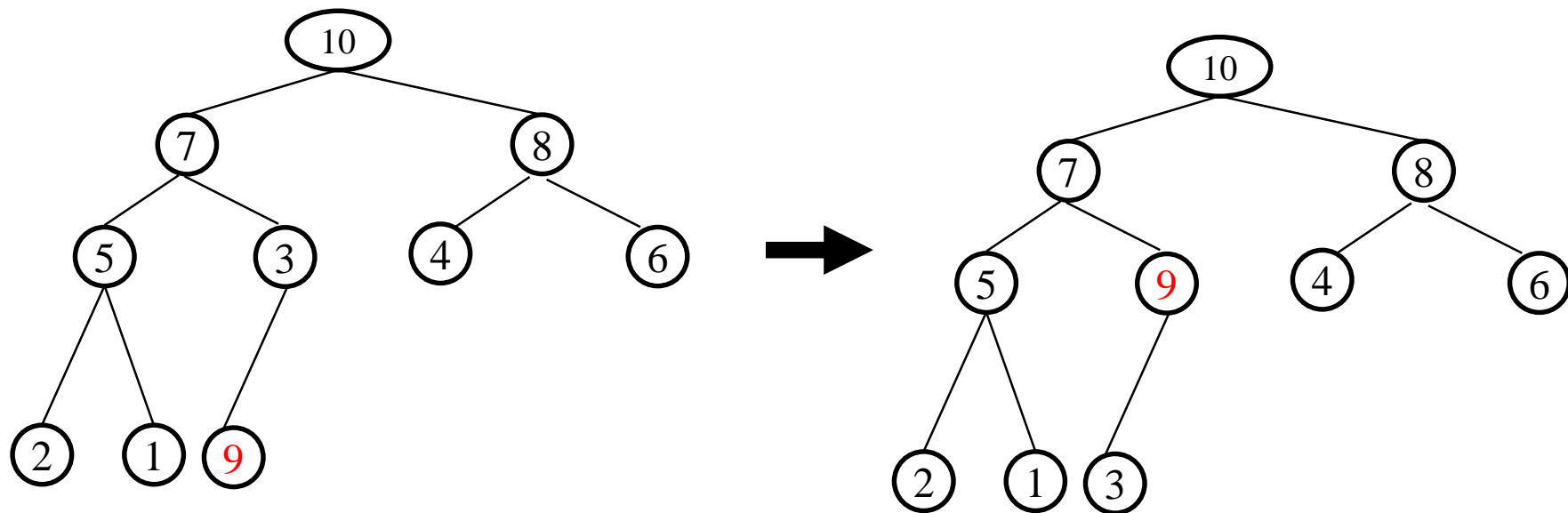


泰国普吉岛最南端

堆的操作：添加一个元素

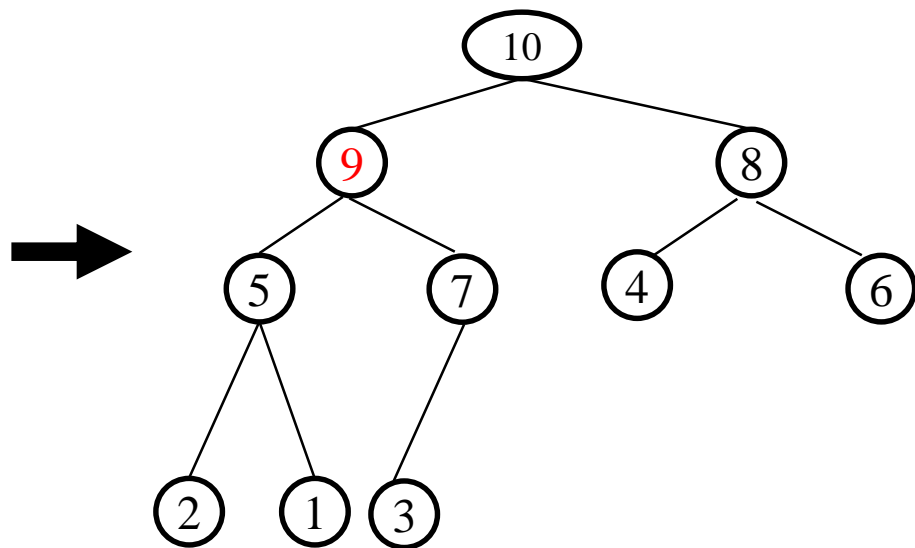
- 1) 假设堆存放在列表a中，长度为n
- 2) 添加元素x到列表a尾部，使其成为a[n]
- 3) 若x优先级高于其父结点，则令其和父结点交换，直到x优先级不高于其父结点，或x被交换到a[0]，变成堆顶为止。此过程称为将x"上移"
- 4) x停止交换后，新的堆形成，长度为n+1

堆的操作：添加一个元素



在堆中添加新元素9

堆的操作：添加一个元素



堆的操作：添加一个元素

显然，交换过程中，以 x 为根的子树，一直都是个堆

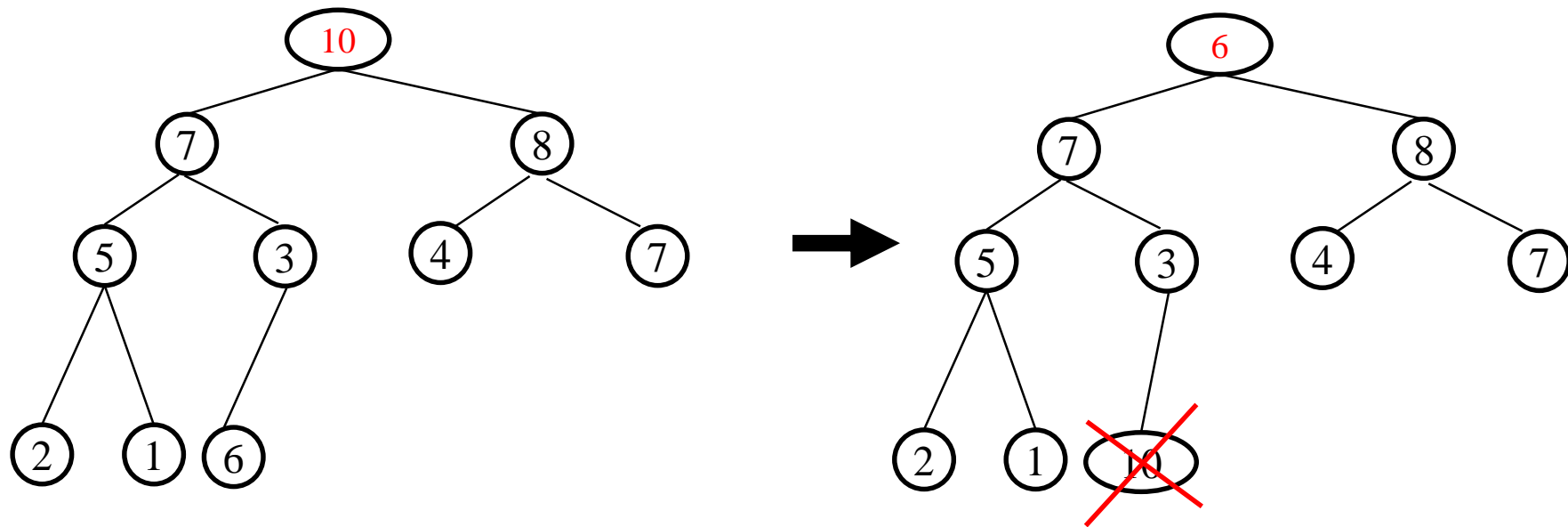
由于 n 个元素的完全二叉树高度为 $\lceil \log_2(n) \rceil$ ，每交换一次 x 就上升一层，因此上移操作复杂度 $O(\log(n))$ ，即添加元素复杂度 $O(\log(n))$

堆的操作：删除堆顶元素

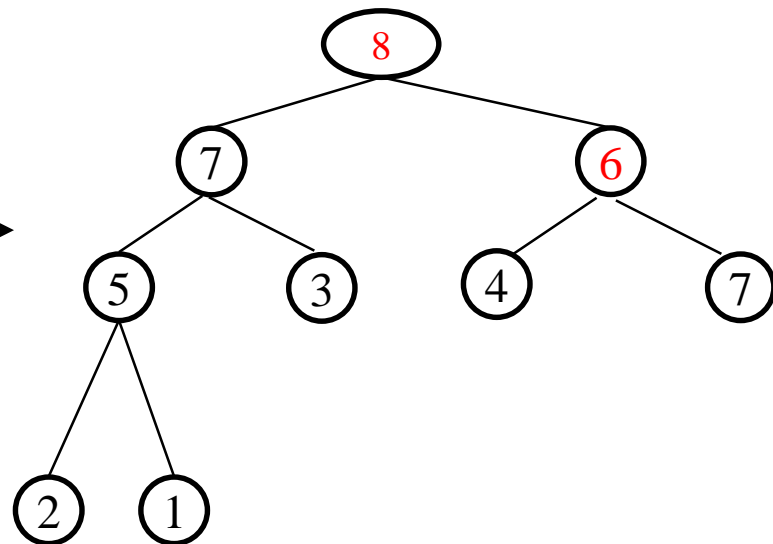
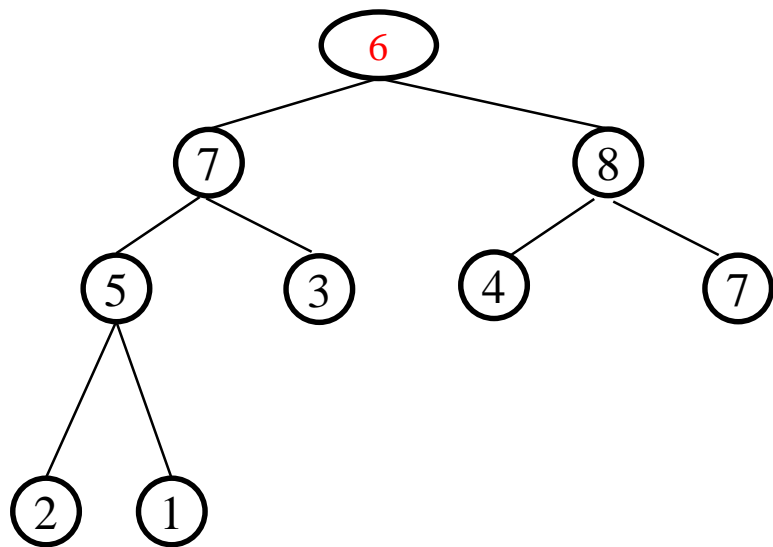
- 1) 假设堆存放在列表a中，长度为n
- 2) 将a[0]和a[n-1]交换
- 3) 将a[n-1]删除(pop)
- 4) 记此时的a[0]为x，则将x和它两个儿子中优先级较高的，且优先级高于x的那个交换，直到x变成叶子结点，或者x的儿子优先级都不高于x为止。将此整个过程称为将x"下移"
- 5) x停止交换后，新的堆形成，长度为n-1

下移过程复杂度为 $O(\log(n))$ ，因此删除堆顶元素复杂度 $O(\log(n))$

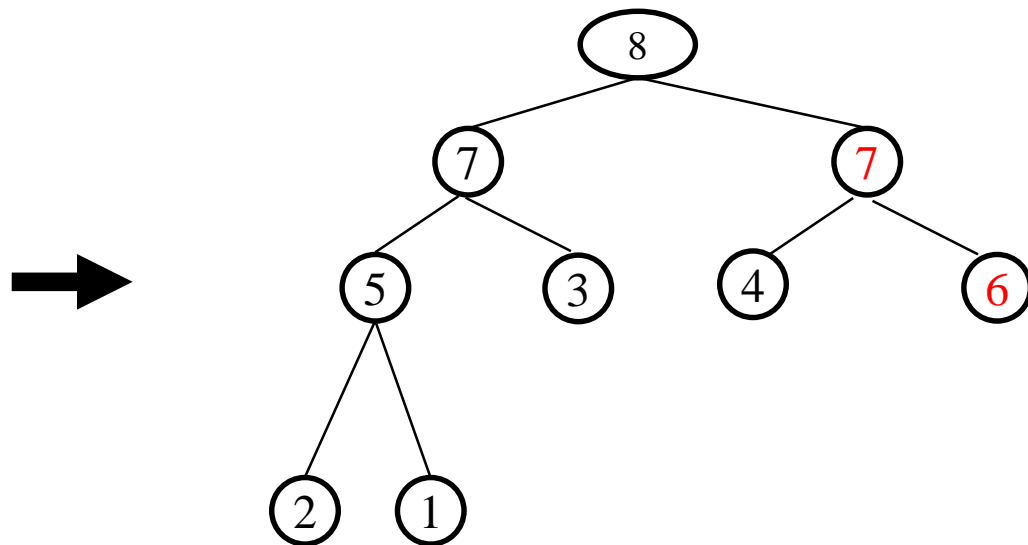
堆的操作：删除堆顶元素



堆的操作：删除堆顶元素



堆的操作：删除堆顶元素



堆的操作：删除堆顶元素

重要结论：

如果 $a[i]$ 的两棵子树都是堆，则对 $a[i]$ 的下移操作完成后，以新 $a[i]$ 为根的子树会形成堆。

堆的操作：建堆

一个长度为 n 的列表 a ,要原地将 a 变成一个堆

方法：

将 a 看作一个完全二叉树。假设有 H 层。根在第0层，第 $H-1$ 层都是叶子

对第 $H-2$ 层的每个元素执行下移操作

对第 $H-3$ 层的每个元素执行下移操作

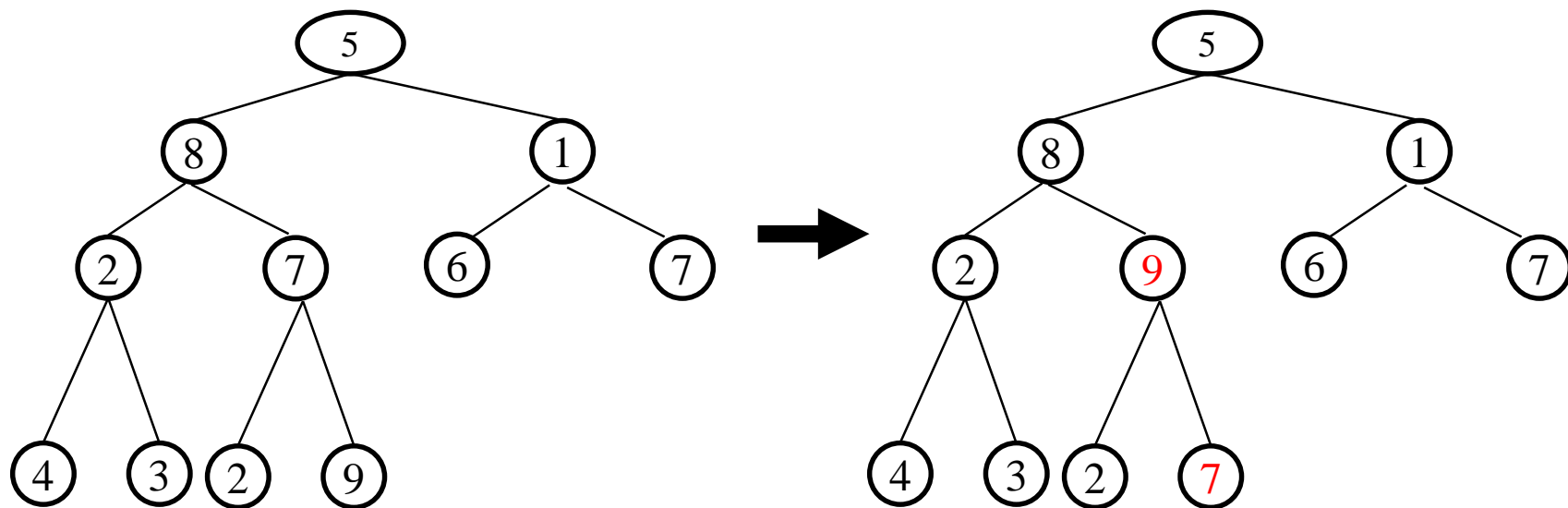
.....

对第0层的元素执行下移操作

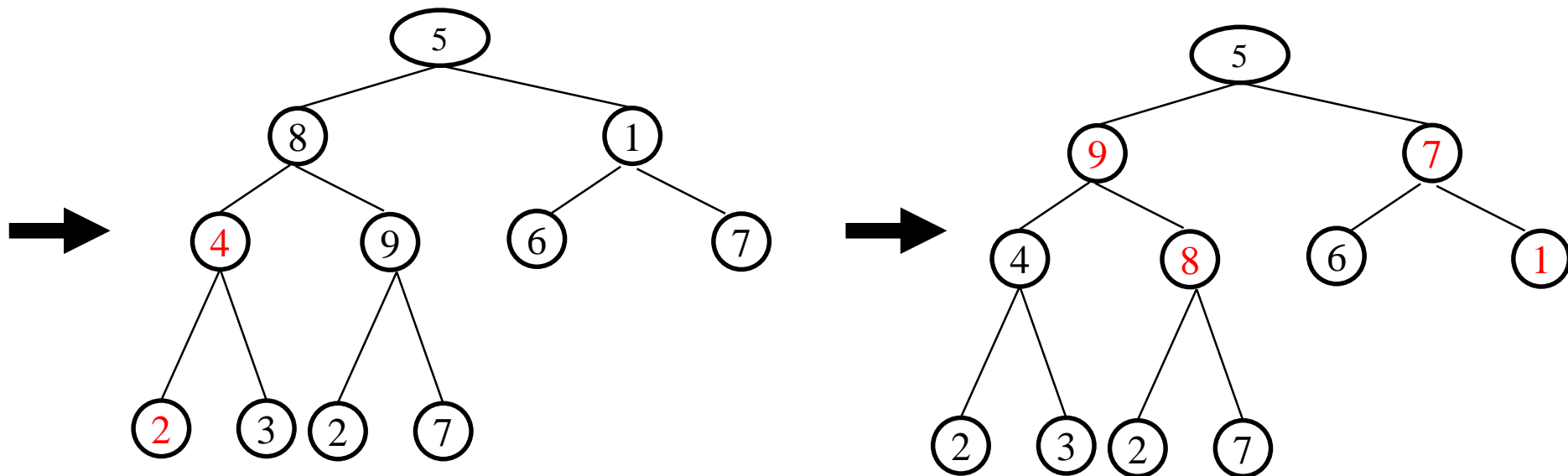
堆即建好。复杂度 $O(n)$ 。证明较难，略

堆的操作：建堆

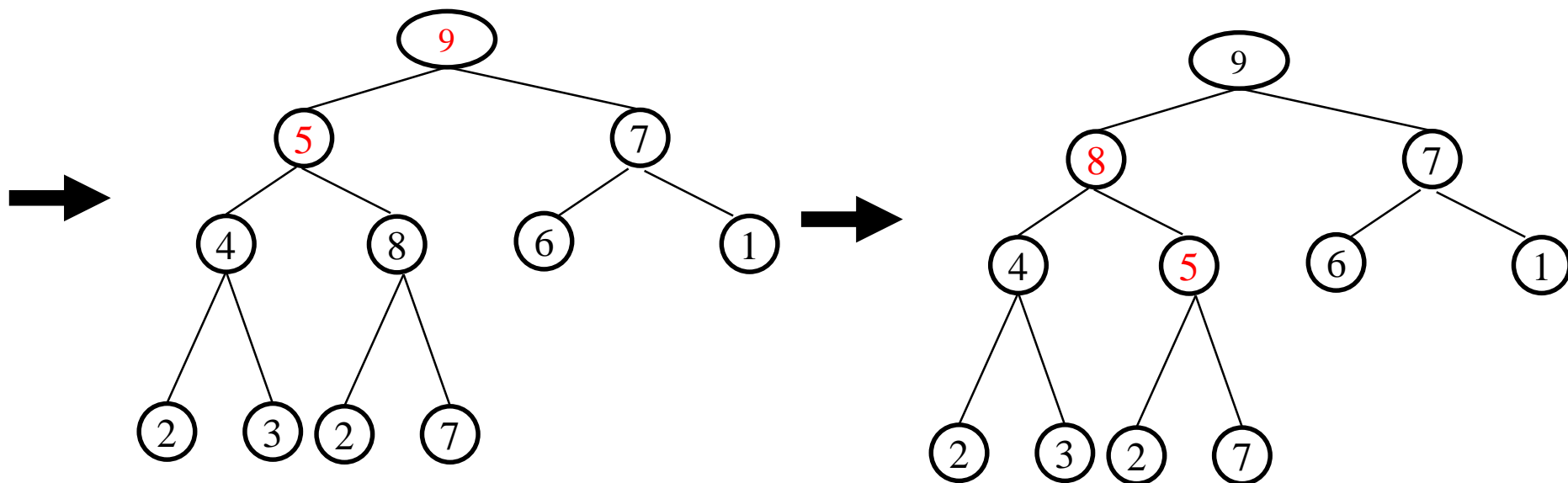
无序列表



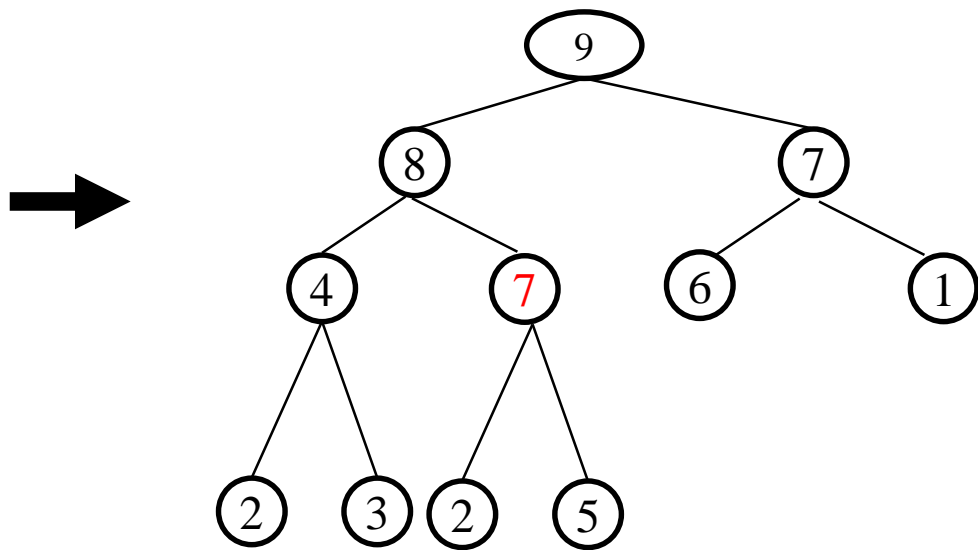
堆的操作：建堆



堆的操作：建堆



堆的操作：建堆





北京大学
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

堆的应用



香港维多利亚湾

堆的应用：哈夫曼编码树的构造

Initial leaves

{ (A 8) (B 3) (C 1) (D 1) (E 1) (F 1) (G 1) (H 1) } Merge

{ (A 8) (B 3) ({C D} 2) (E 1) (F 1) (G 1) (H 1) } Merge

{ (A 8) (B 3) ({C D} 2) ({E F} 2) (G 1) (H 1) } Merge

{ (A 8) (B 3) ({C D} 2) ({E F} 2) ({G H} 2) } Merge

{ (A 8) (B 3) ({C D} 2) ({E F G H} 4) } Merge

{ (A 8) ({B C D} 5) ({E F G H} 4) } Merge

{ (A 8) ({B C D E F G H} 9) } Final merge

{ ({A B C D E F G H} 17) }

哈夫曼编码树不唯一

用"堆"存放结点集合，便于快速取出最小权值的两个结点，以及加入合并后的新结点。

堆的应用：堆排序

- 1) 将待排序列表a变成一个堆($O(n)$)
- 2) 将 $a[0]$ 和 $a[n-1]$ 交换，然后对新 $a[0]$ 做下移，维持前 $n-1$ 个元素依然是堆。此时优先级最高的元素就是 $a[n-1]$
- 3) 将 $a[0]$ 和 $a[n-2]$ 交换，然后对新 $a[0]$ 做下移，维持前 $n-2$ 个元素依然是堆。此时优先级次高的元素就是 $a[n-2]$

.....

直到堆的长度变为1，列表a就按照优先级从低到高排好序了。

整个过程相当不断删除堆顶元素放到a的后部。堆顶元素依次是优先级最高的、次高的....

一共要做 n 次下移，每次下移 $O(\log(n))$ ，因此总复杂度 $O(n\log(n))$

堆排序

如果用递归实现，需要 $O(\log(n))$ 额外栈空间(递归要进行 $\log(n)$ 层)。

如果不用递归实现，需要 $O(1)$ 额外空间。



北京大学
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

堆的实现



美国胡佛水坝

堆的实现

```
class Heap:
```

```
    def __init__(self, array = [], less = lambda x, y : x < y):
```

```
        #若x为堆顶元素, y为堆中元素, 则 less(x,y)为True
```

```
        #默认情况下, 小的算优先级高 #i的儿子是2*i+1和2*i+2
```

```
        self._a = array[:] #array是列表
```

```
        self._size = len(array)
```

```
        self._less = less      #less是比较函数
```

```
        self.makeHeap()
```

```
    def top(self):
```

```
        return self._a[0]
```

```
    def pop(self): #删除堆顶元素
```

```
        tmp = self._a[0]
```

```
        self._a[0] = self._a[-1]
```

```
        self._a.pop()
```

```
        self._size -= 1
```

```
        self._goDown(0)    #_goDown是下移操作, 将a[0]下移
```

```
        return tmp
```

堆的实现

```
def append(self,x):    #往堆中添加x
    self._size += 1
    self._a.append(x)
    self._goUp(self._size-1) #_goUp是上移
def _goUp(self,i): #将a[i]上移
    #只在append的时候调用，不能直接调用或在别处调用
    #被调用时，以a[i]为根的子树，已经是个堆
    if i == 0:
        return
    f = ( i -1 )// 2 #父结点下标
    if self._less(self._a[i],self._a[f]):
        self._a[i],self._a[f] = self._a[f],self._a[i]
        self._goUp(f) #a[f]上移
```

堆的实现

```
def _goDown(self,i): #a[i]下移
    #前提: 在a[i]的两个子树都是堆的情况下, 下移
    if i * 2 + 1 >= self._size: #a[i]没有儿子
        return
    L,R = i * 2 + 1, i * 2 + 2
    if R >= self._size or self._less(self._a[L],self._a[R]):
        s = L
    else:
        s = R
    #上面选择小的儿子
    if self._less(self._a[s],self._a[i]):
        self._a[i],self._a[s] = self._a[s],self._a[i]
        self._goDown(s)
```

堆的实现

```
def makeHeap(self): #建堆
    i = (self._size - 1 - 1) // 2 #i是最后一个叶子的父亲
    for k in range(i, -1, -1):
        self._goDown(k)

def heapSort(self): #建好堆之后调用, 进行堆排序
    for i in range(self._size-1, -1, -1):
        self._a[i], self._a[0] = self._a[0], self._a[i]
        self._size -= 1
        self._goDown(0)
    self._a.reverse()
    return self._a
```

堆的实现

#下面是堆的用法，不是堆内部的代码

```
import random
def heapSort(a,less): #对列表a进行堆排序,哪个less哪个排在前面
    hp = Heap(a,less)
    return hp.heapSort()

s = [i for i in range(17)]
random.shuffle(s)
print(s)
h = heapSort(s,lambda x,y : x < y)
print(h)
```




北京大学
PEKING UNIVERSITY

信息科学技术学院

北京大学信息学院 郭炜

Python中的堆



新疆安集海大峡谷

python中的堆: heapq

要 `import heapq`

heapq中的函数:

<code>heapq.heapify(s)</code>	将列表s变成一个堆
<code>heapq.heappush(s, item)</code>	往已经是堆的列表s里面添加元素item
<code>heapq.heappop(s)</code>	取出并返回堆顶元素。s必须已经是个堆(注意: 会减少s长度)
<code>heapq.heapreplace(s, item)</code>	取出并返回堆顶元素, 并将元素item加入堆(s还是个堆, 长度不变)
<code>heapq.nlargest(n, s, key)</code>	返回序列s中的最大n个元素构成的列表。key是关键字函数
<code>heapq.nsmallest(n, s, key)</code>	返回序列s中的最小n个元素构成的列表
<code>#key(x) < key(y)</code>	x就比y小

python中的堆: heapq

```
import heapq
```

```
heapq.heapify(s)
```

将列表s变成一个堆

```
heapq.heappush(s,item)
```

往已经是堆的s里面添加元素item

```
heapq.heappop(s)
```

弹出堆顶元素(会减少s长度)

```
.....
```

```
def heapsort(iterable): #iterable是个序列
```

```
#函数返回一个列表, 内容是iterable中元素排序的结果
```

```
    h = []
```

```
    for value in iterable:
```

```
        h.append(value)
```

```
    heapq.heapify(h)
```

```
    return [heapq.heappop(h) for i in range(len(h))]
```

不便之处: 没有设定排序规则的机会。如果要形成大元素在顶的整数堆, 只能取相反数进堆。出来的时候再取相反数

python中的堆: heapq

```
import heapq
def heapSorted(iterable): #iterable是个序列
#函数返回一个列表, 内容是iterable中元素排序的结果, 不会改变iterable
    h = []
    for value in iterable:
        h.append(value)
    heapq.heapify(h)
    return [heapq.heappop(h) for i in range(len(h))]

a = (2,13,56,31,5)
print(heapSorted(a)) #>>[2, 5, 13, 31, 56]
print(heapq.nlargest(3,a)) #>>[56, 31, 13]
print(heapq.nlargest(3,a,lambda x:x%10)) #>>[56, 5, 13]
#取个位数最大的三个
print(heapq.nsmallest(3,a,lambda x:x%10)) #>>[31, 2, 13]
#取个位数最小的三个
```