

## 12B4 批量建堆

#数据结构邓神

### 自上而下的上滤：算法

对于任意给定的n个元素建造一个堆 (heapification)

```
template <typename T> void PQ_ComplHeap(T* A, Rank n){  
    copyFrom(A, 0, n);  
    heapify(n);  
}
```

### 蛮力 Brute

```
template <typename T> void PQ_ComplHeap<T>::heapify(Rank n){  
    for (int i = 1; i < n; ++i) { // 第一个元素没有父节点，可以不做  
        percolateUp(i); // 将所有节点上滤  
    }  
}
```

**自上而下的上滤**

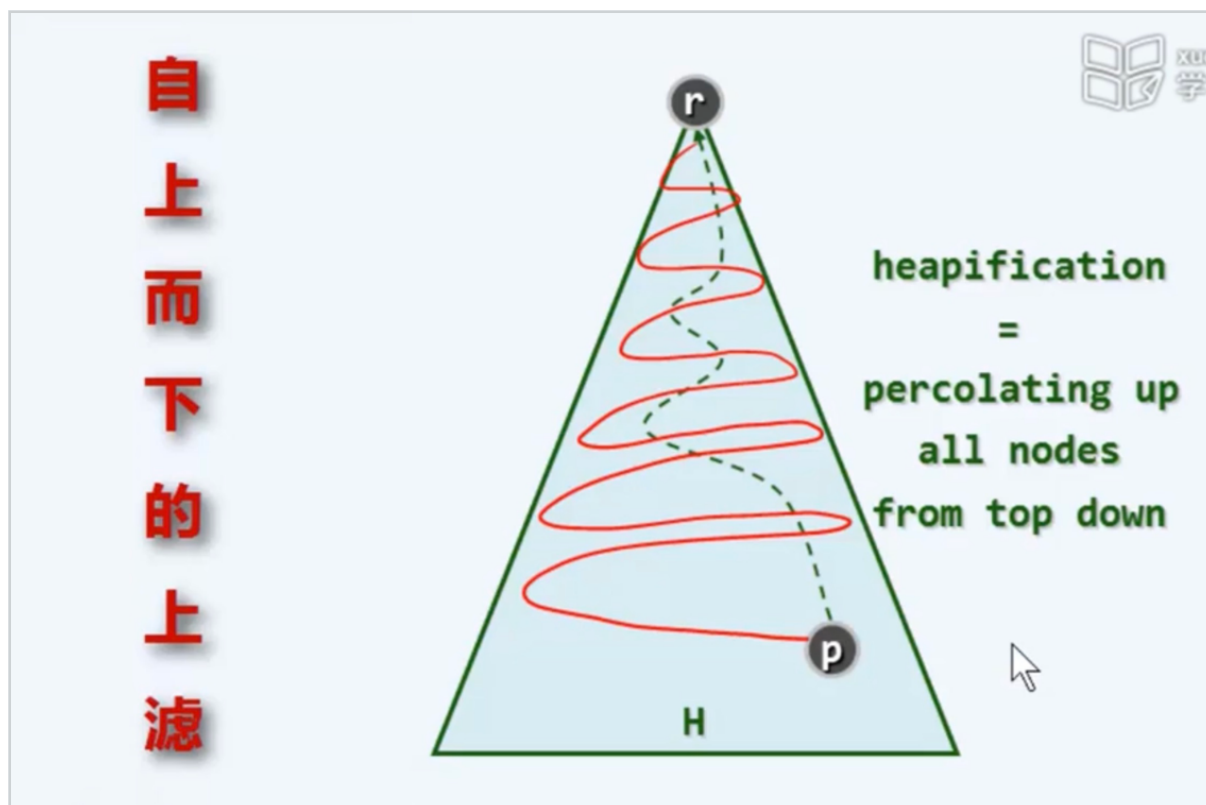
❖ `PQ_ComplHeap( T* A, Rank n ) { copyFrom( A, 0, n ); heapify( n ); }` //如何实现

❖ `template <typename T> void PQ_ComplHeap<T>::heapify ( Rank n ) {` //蛮力

`for ( int i = 1; i < n; i++ )` //按照层次遍历次序逐一 *insert(i)*

`percolateUp ( i );` //经上滤插入各节点

}



## 效率

我们从最坏情况，即每一个新插入的节点都比原来所有的大

也就是每个节点都要上滤到跟节点

计算成本线性正比与深度

xuetangx 学堂

效率

❖ 最坏情况下

每个节点都需上滤至根  $\Leftarrow$

所需成本线性正比于其深度  $\Leftarrow \sum_i \text{depth}(i)$

❖ 即便只考虑底层

$n/2$  个叶节点，深度均为  $O(\log n)$

累计耗时  $O(n \log n)$

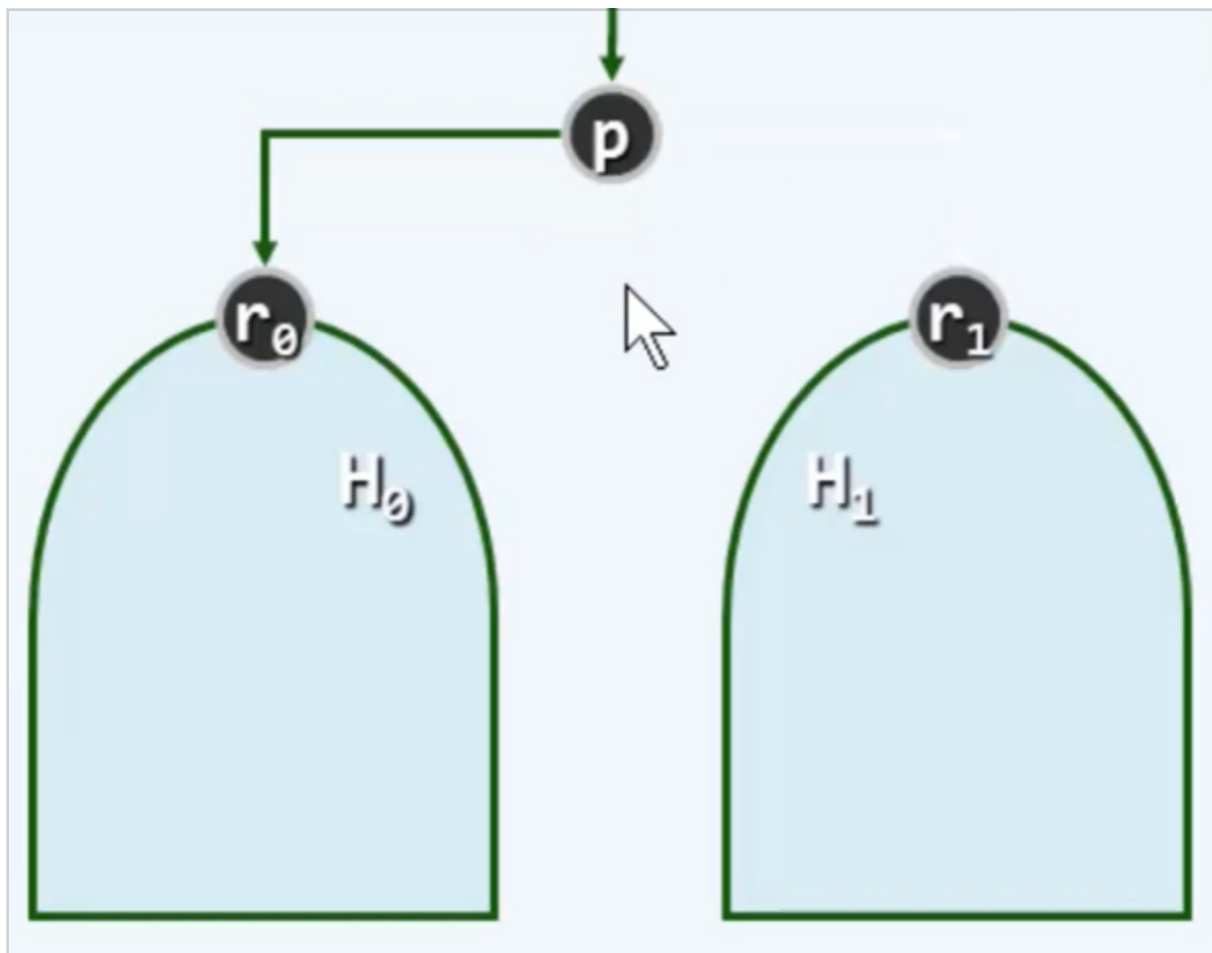
heapification  
=  
percolating up  
all nodes  
from top down

我们会发现时间高达  $O(n \log n)$  为什么说这个是低效的呢？

快速排序的平均时间复杂度也为  $O(n \log n)$  这是不可接受的效率

偏序，所以应该还能再快！

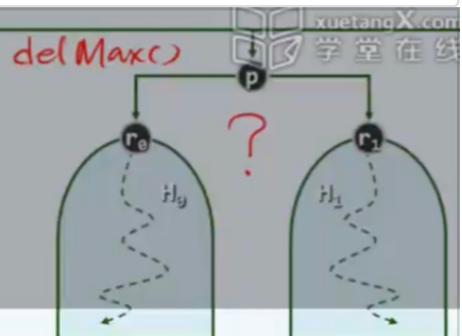
自下而上的下滤：算法



```
template <typename T> void PQ_ComplHeap<T>::heapify(Rank n){
    for (int i = LastInternal(n); i >= 0; i--) { // 最后一个分支节点
        percolateDown(n,i); // 在完全二叉树内部，算法为 (取下整)(N\2) - 1
    }
}
```

### 自下而上的下滤

- ❖ 任意给定堆  $H_0$  和  $H_1$ ，以及节点  $p$
- ❖ 为得到堆  $H_0 \cup \{p\} \cup H_1$ ，只需将  $r_0$  和  $r_1$  当作  $p$  的孩子，对  $p$  下滤



```
❖ template <typename T>
void PQ_ComplHeap<T>::heapify( Rank n ) { //Robert Floyd, 1964

    for ( int i = LastInternal(n); i >= 0; i-- ) //自下而上，依次

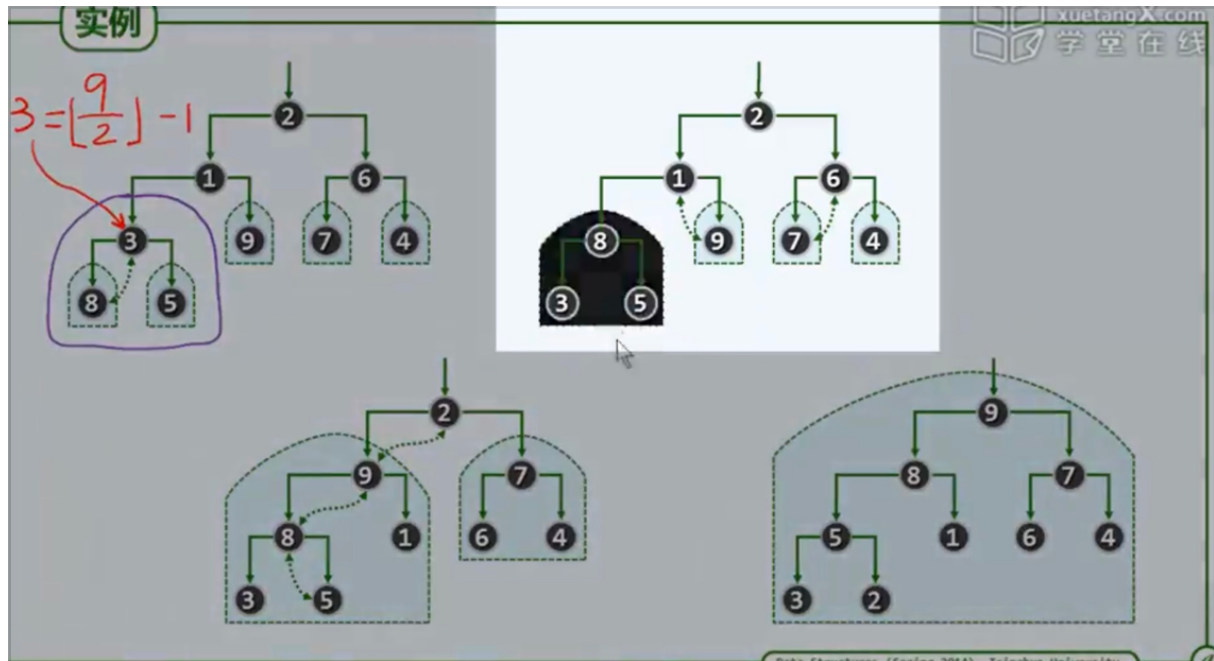
        percolateDown( n, i ); //下滤各内部节点

} //可理解为子堆的逐层合并，——由以上性质，堆序性最终必然在全局恢复
```

处理各个节点的次序与刚才的算法完全相反

Begin with the end in mind // 以终为始

## 实例



这个算法省去大量上滤叶节点的操作

## 效率

计算成本来自于对于每一个节点的下滤。

每个内部节点下降所需的时间，正比与其高度

**Height(i) = O(n)**

**Depth(i) = O(n log n)**

为什么有巨大的差异呢？

越是靠近底层节点越多，越是靠近顶层节点越少