优先级队列

完全二叉堆：批量建堆

12-B4

今世号通人者，务为艰深之文，陈过高之义，以为士大夫劝，而独不为彼什伯千万倍里巷乡闾之子计，则是智益智，愚益愚，智日少，愚日多也。

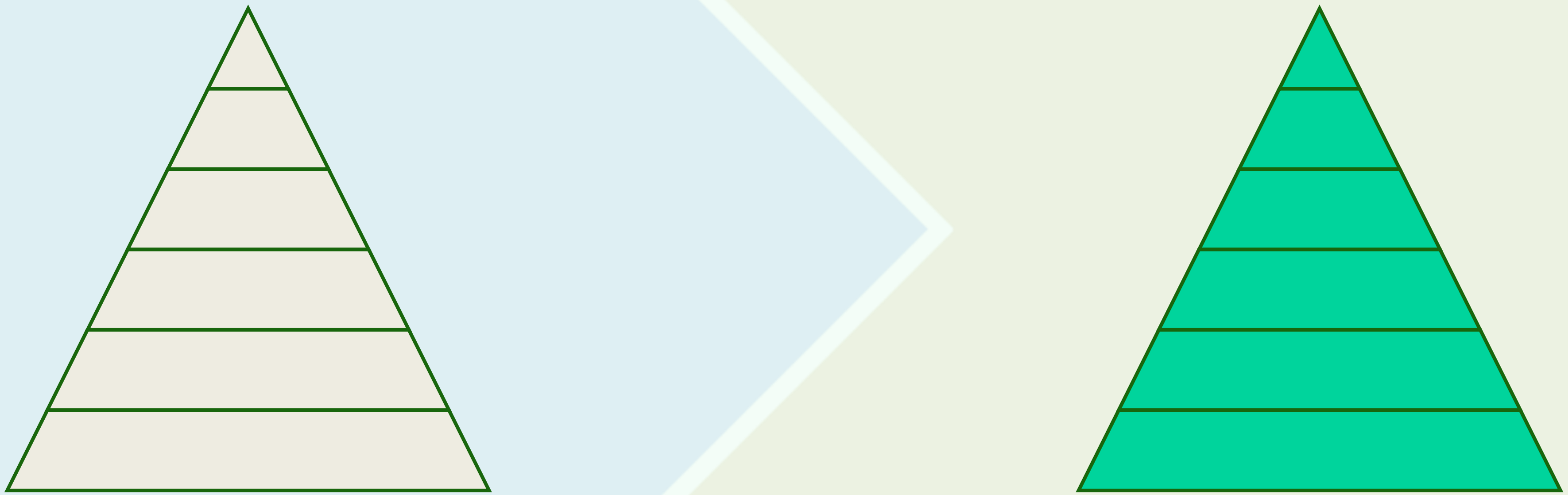
现在，培训富人应对贫穷要比教育穷人获得财富更切合实际，因为从人数比例上来说，富人破产的越来越多，而穷人变富的越来越少。

邓俊辉
deng@tsinghua.edu.cn

自上而下的上滤（1/2）

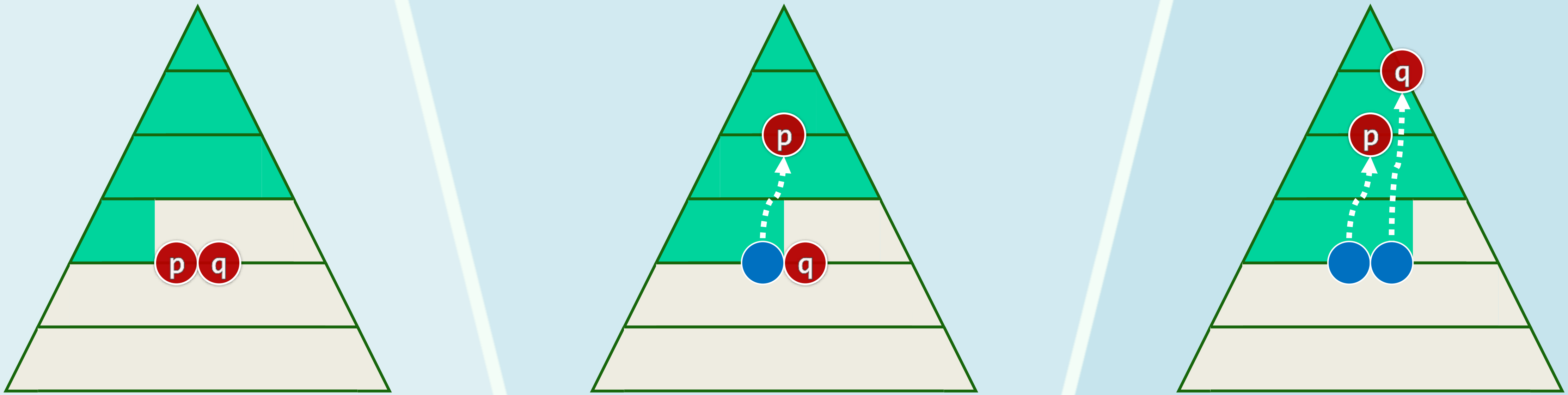
❖ PQ_ComplHeap(T* A, Rank n)

```
{ copyFrom( A, 0, n ); heapify( _elem, n ); }
```



自上而下的上滤 (2/2)

```
❖ template <typename T> void heapify( T* A, const Rank n ) { //蛮力
     for ( int i = 1; i < n; i++ ) //按照逐层遍历次序逐一
        percolateUp( A, i ); //经上滤插入各节点
  }
```



效率

❖ 最坏情况下

- 每个节点都需上滤至**根**
- 所需成本线性正比于其**深度**

❖ 即便只考虑**底层**

- $n/2$ 个叶节点，深度均为 $\mathcal{O}(\log n)$
- 累计耗时 $\mathcal{O}(n \log n)$

❖ 这样长的时间，本足以**全排序**！

应该，能够更快的...

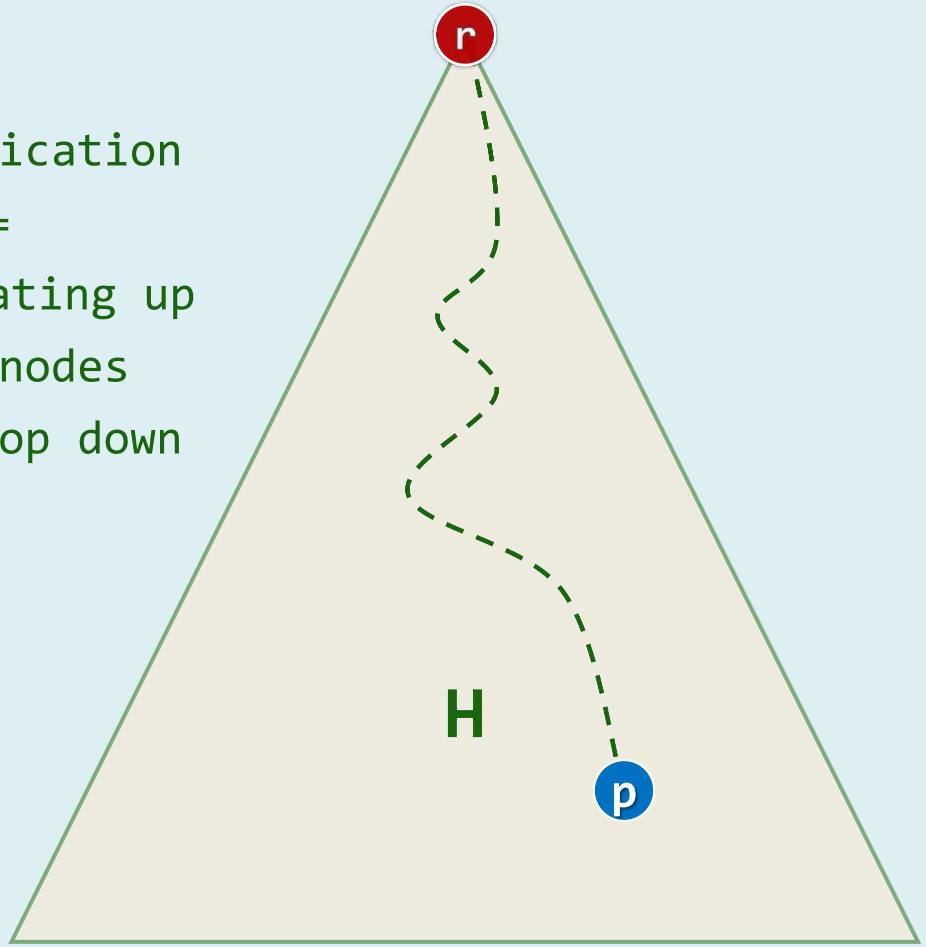
heapification

=

percolating up

all nodes

from top down



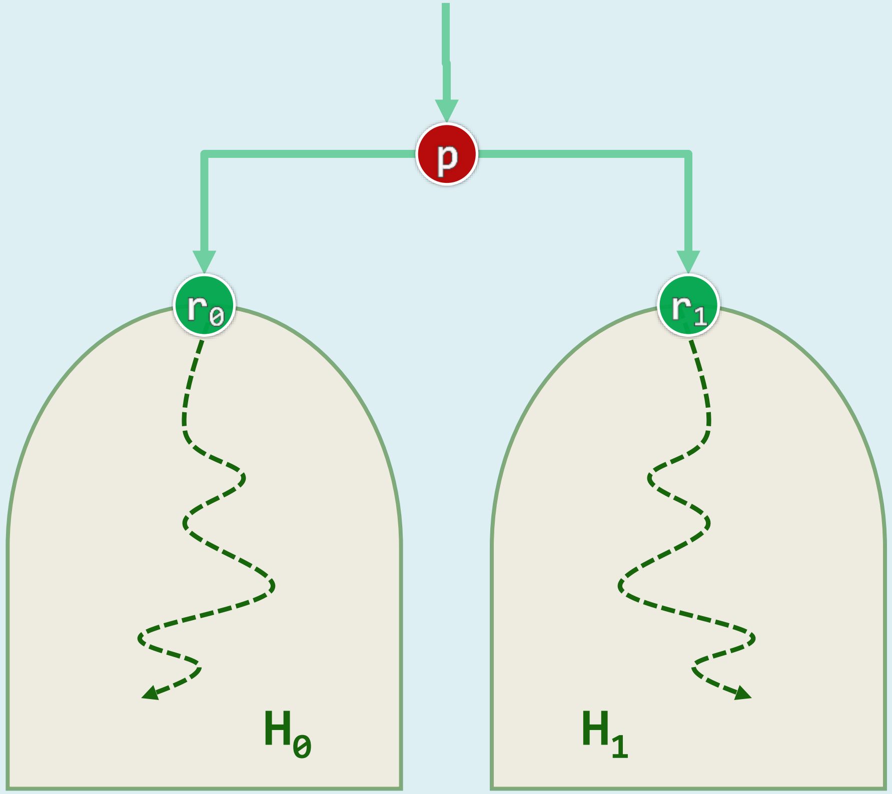
自下而上的下滤

❖ 任意给定堆 $\mathcal{H}_0$ 和 $\mathcal{H}_1$ ，以及节点 p

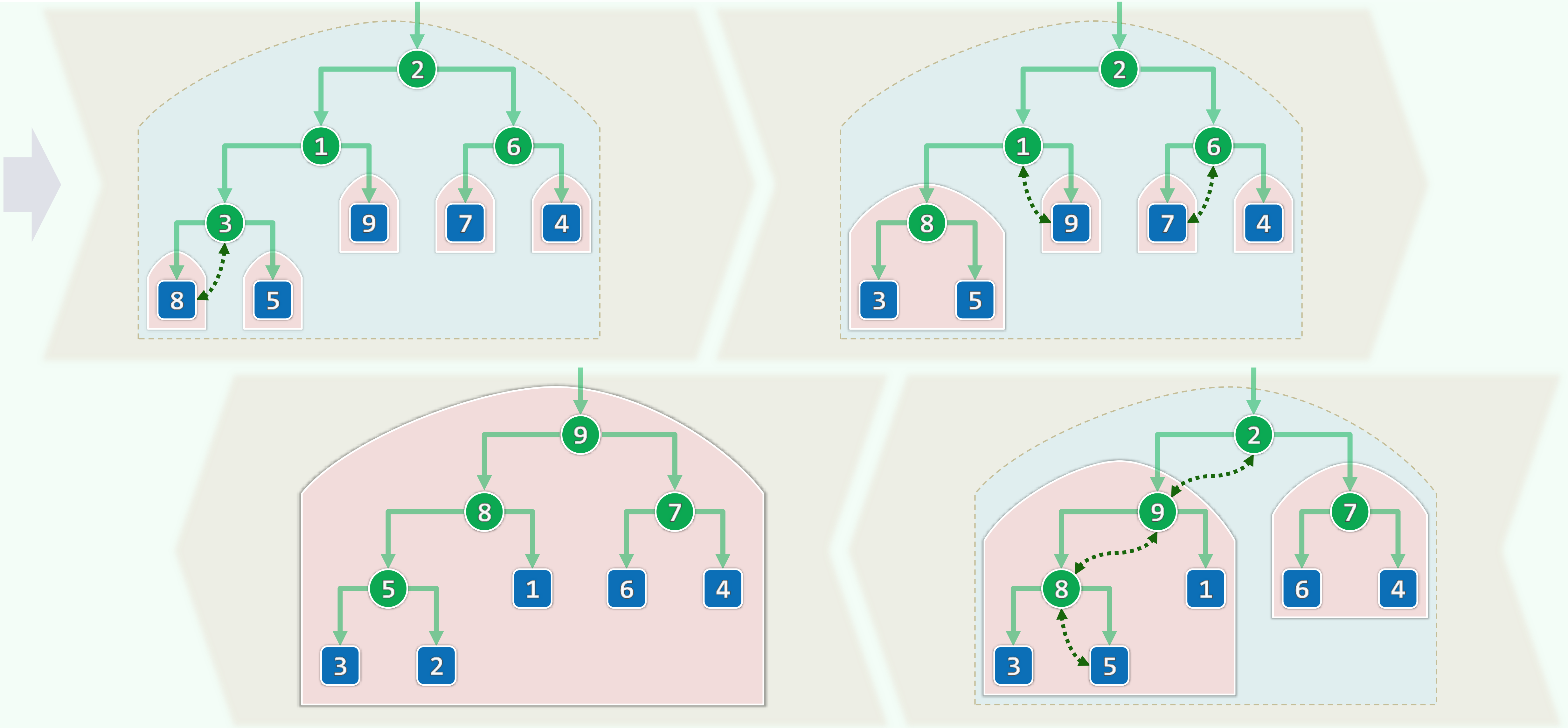
❖ 为得到堆 $\mathcal{H}_0 \cup \{p\} \cup \mathcal{H}_1$，只需

将 r_0 和 r_1 当作 p 的孩子，再对 p 下滤

❖
```
template <typename T> //Robert Floyd, 1964
void heapify( T* A, Rank n ) { //自下而上
   for ( int i = n/2 - 1; 0 <= i; i-- ) //依次
      percolateDown( A, 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
```



实例



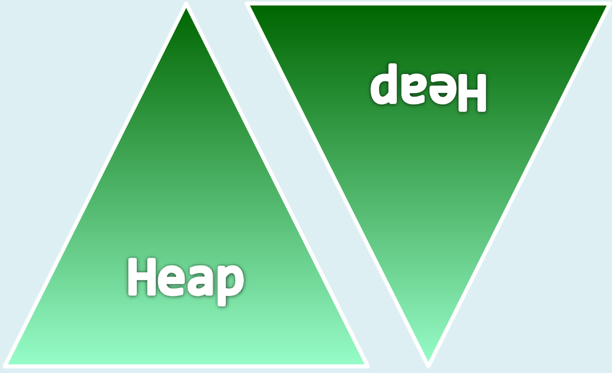
效率

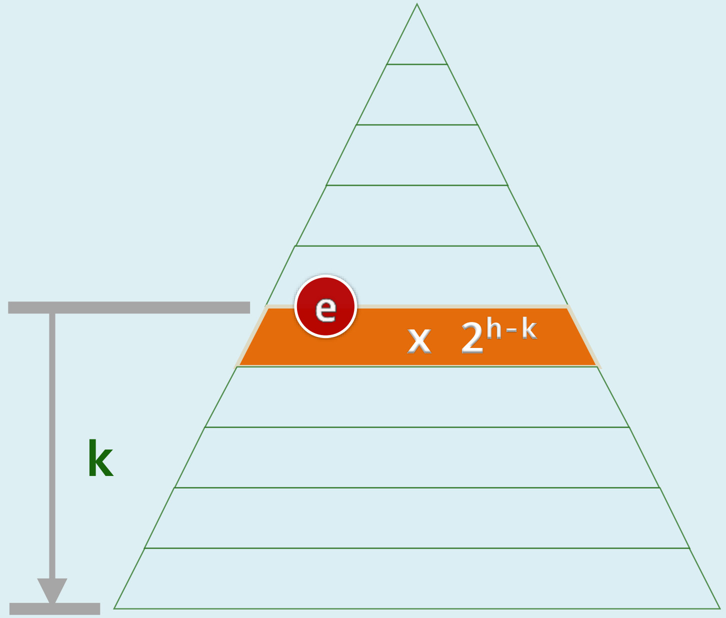
❖ 每个内部节点所需的调整时间，正比于其高度而非深度

❖ 不失一般性，考查满树：$n = 2^{h+1} - 1$

❖ 所有节点的高度总和

$$
\begin{aligned}
S(n) &= \sum_{k=1}^{h} k \cdot 2^{h-k} = \sum_{k=1}^{h}\sum_{i=1}^{k} 2^{h-k} = \sum_{i=1}^{h}\sum_{k=i}^{h} 2^{h-k} \\
&= \sum_{i=1}^{h}\sum_{k=0}^{h-i} 2^{k} = \sum_{i=1}^{h}\{2^{h-i+1} - 1\} = \sum_{i=1}^{h} 2^{h-i+1} - h \\
&= \sum_{i=1}^{h} 2^{i} - h = 2^{h+1} - 2 - h = \mathcal{O}(n)
\end{aligned}
$$





课后

❖ insert()： 最坏情况下效率为O(logn)，**平均**情况呢？

❖ heapify()：构造次序颠倒后，为什么复杂度会有**实质**降低？

这一算法在哪些场合不适用？

❖ 扩充接口：
```
decrease( i, delta ) //任一元素_elem[i]的数值减小delta
increase( i, delta ) //任一元素_elem[i]的数值增加delta
remove( i ) //删除任一元素_elem[i]
```

❖ 借助完全堆，在$\mathcal{O}(n\log n)$时间内构造Huffman树

❖ **大**顶堆的delMin()操作，能否也在$\mathcal{O}(\log n)$时间内完成？

难道，为此需要同时维护一个**小**顶堆？