10.优先级队列

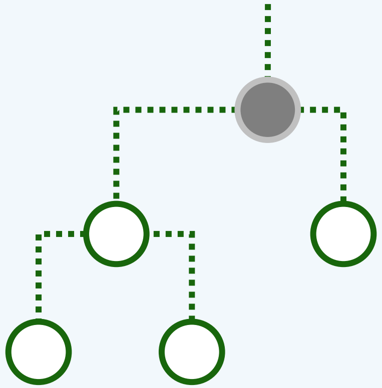
(b4) 完全二叉堆：批量建堆

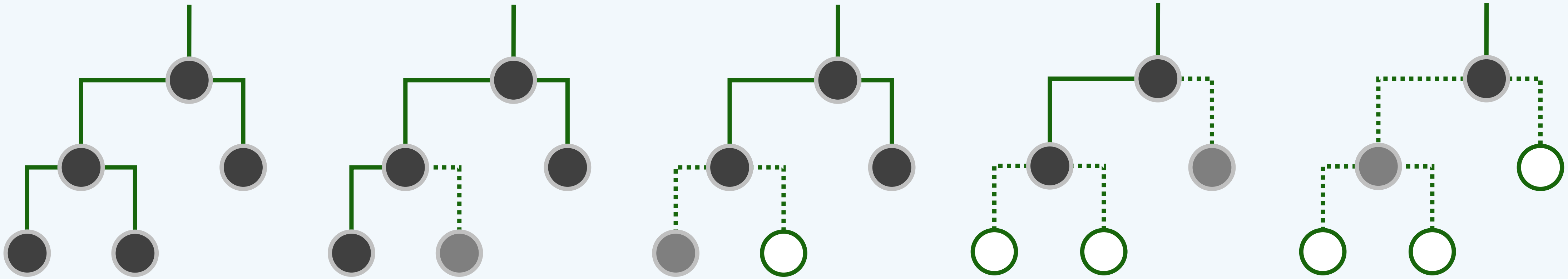
邓俊辉

deng@tsinghua.edu.cn

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

         percolateUp ( 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 $\boxed{\text{根}}$

所需成本线性正比于其 $\boxed{\text{深度}}$

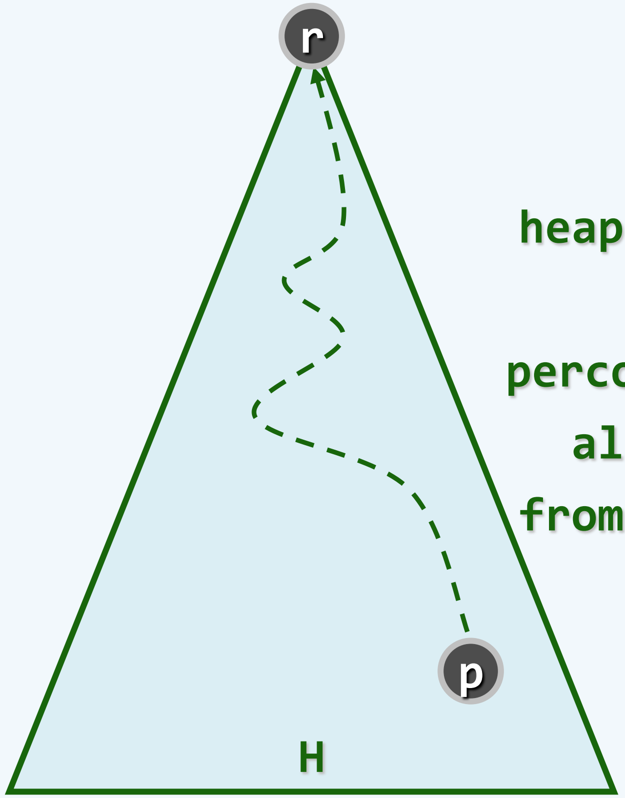
❖ 即便只考虑 $\boxed{\text{底层}}$

$\boxed{n/2}$ 个叶节点，深度均为 $\boxed{\mathcal{O}(\log n)}$

累计耗时 $\boxed{\mathcal{O}(n\log n)}$

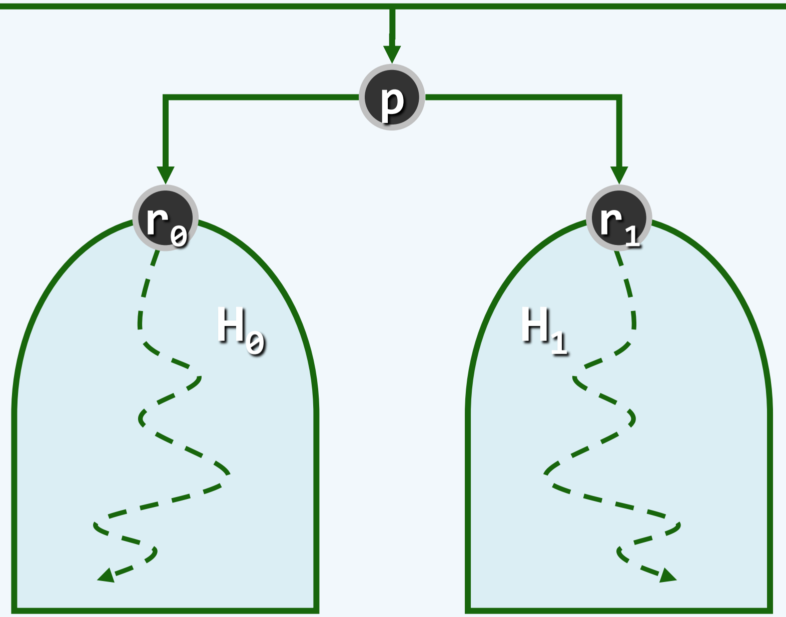
❖ 这样长的时间，本足以 $\boxed{\text{全排序}}$ ！

应该，能够更快的...



heapification
=
percolating up
all nodes
from top down

自下而上的下滤



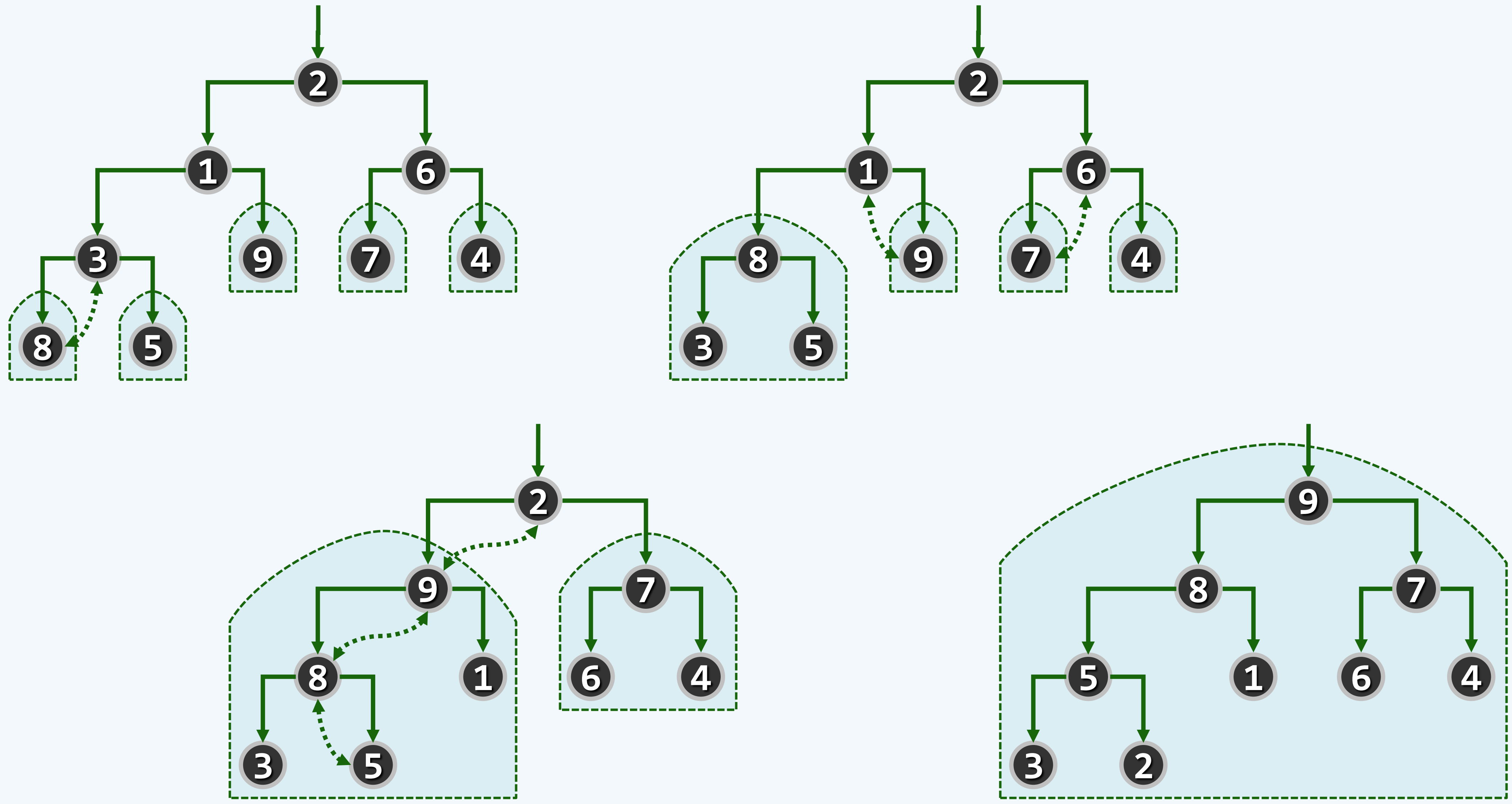
❖ 任意给定堆 H_0 和 H_1，以及节点 p

❖ 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
```

实例



效率

❖ 每个内部节点所需的调整时间，正比于其 $\boxed{\text{高度}}$ 而非 $\boxed{\text{深度}}$

❖ 不失一般性，考查满树：$n = 2^{d+1} - 1$

❖ $S(n)$ = 所有节点的 $\boxed{\text{高度}}$ 总和

$= \Sigma_{i=0..d}(\ (d - i) \times 2^i\ )$

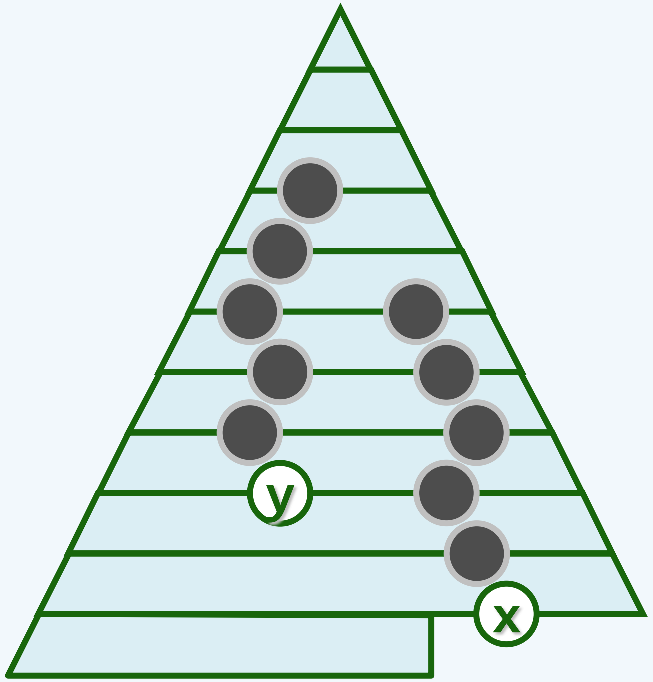
$= d \times \Sigma_{i=0..d}\ 2^i - T(n)$

$= d \times (\ 2^{d+1} - 1\ ) - [\ (\ d - 1\ ) \times 2^{d+1} + 2\ ]$

$= 2^{d+1} - (\ d + 2\ )$

$= n - \log_2(\ n + 1\ )$

$= \mathcal{O}(n)$



课后

- insert()：最坏情况下效率为$\mathcal{O}(\log n)$，平均情况呢？
- heapify()：构造次序颠倒后，为什么复杂度会实质性地降低？
  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
- 在大顶堆中，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？