10.优先级队列

（xa）左式堆

左之左之，君子宜之

右之右之，君子有之

邓俊辉

deng@tsinghua.edu.cn

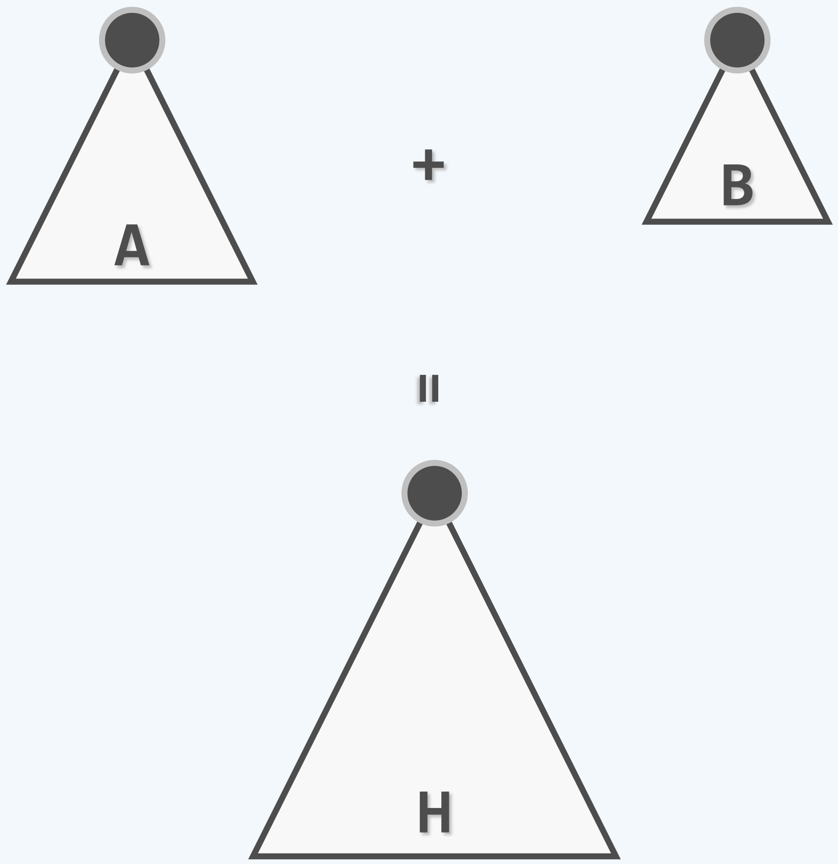
堆合并

- ❖ H = merge(A, B)：将堆A和B合二为一 //不妨设 $|A| = n \ge m = |B|$
- ❖ 方法一：A.insert(B.delMax())

  $\mathcal{O}(\ m * (\ \log m + \log(n + m)\ )\ )$

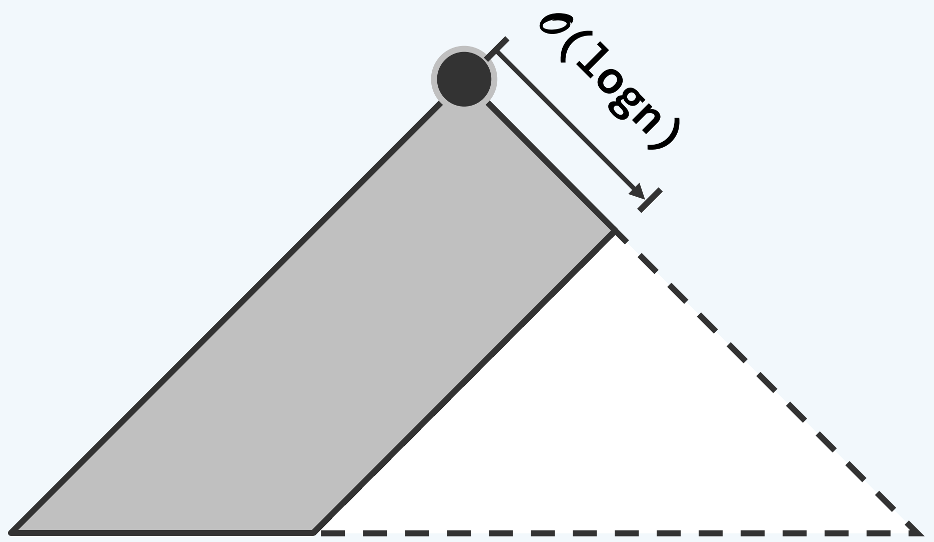
  $= \mathcal{O}(\ m * \log(n + m)\ )$
- ❖ 方法二：union(A, B).heapify(n + m)

  $\mathcal{O}(\ m + n\ )$
- ❖ 有没有更好的办法？比如...
- ❖ 可否奢望在...$\mathcal{O}(\log n)$...时间内实现merge()？



单侧倾斜

- C. A. Crane, 1972： 保持$\boxed{\text{堆序性}}$，附加$\boxed{\text{新条件}}$，使得
  在堆合并过程中，只需调整$\boxed{\text{很少}}$部分的节点 $\boxed{\mathcal{O}(\log n)}$
- 新条件 = $\boxed{\text{单侧倾斜}}$： $\boxed{\text{节点分布}}$偏向于$\boxed{\text{左}}$侧
  $\boxed{\text{合并操作}}$只涉及$\boxed{\text{右}}$侧
- 可是，果真如此，则...
- 拓扑上不见得是$\boxed{\text{完全}}$二叉树，$\boxed{\text{结构性}}$无法保证！？
- 是的，实际上，$\boxed{\text{结构性}}$并非堆结构的$\boxed{\text{本质}}$要求

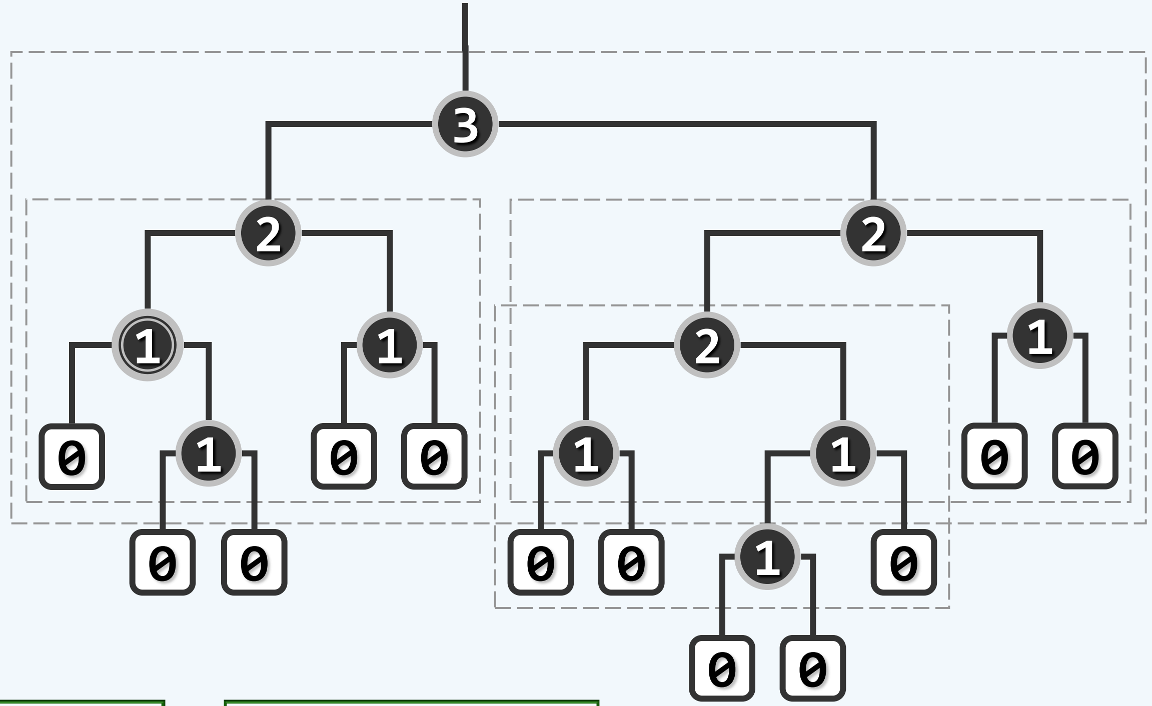


空节点路径长度

❖ 引入所有的外部节点

消除一度节点

转为真二叉树



❖ Null Path Length

0) npl(NULL) = 0

1) npl(x) = 1 + min(npl(lc(x)), npl(rc(x)))

❖ 验证： npl(x) = x到外部节点的最近距离

npl(x) = 以x为根的最大满子树的高度

左倾性 & 左式堆

❖ 左倾：对任何内节点x，都有 npl($\boxed{lc(x)}$) $\geq$ npl($\boxed{rc(x)}$)

推论：对任何内节点x，都有 npl($\boxed{x}$) = $\boxed{1}$ + npl($\boxed{rc(x)}$)

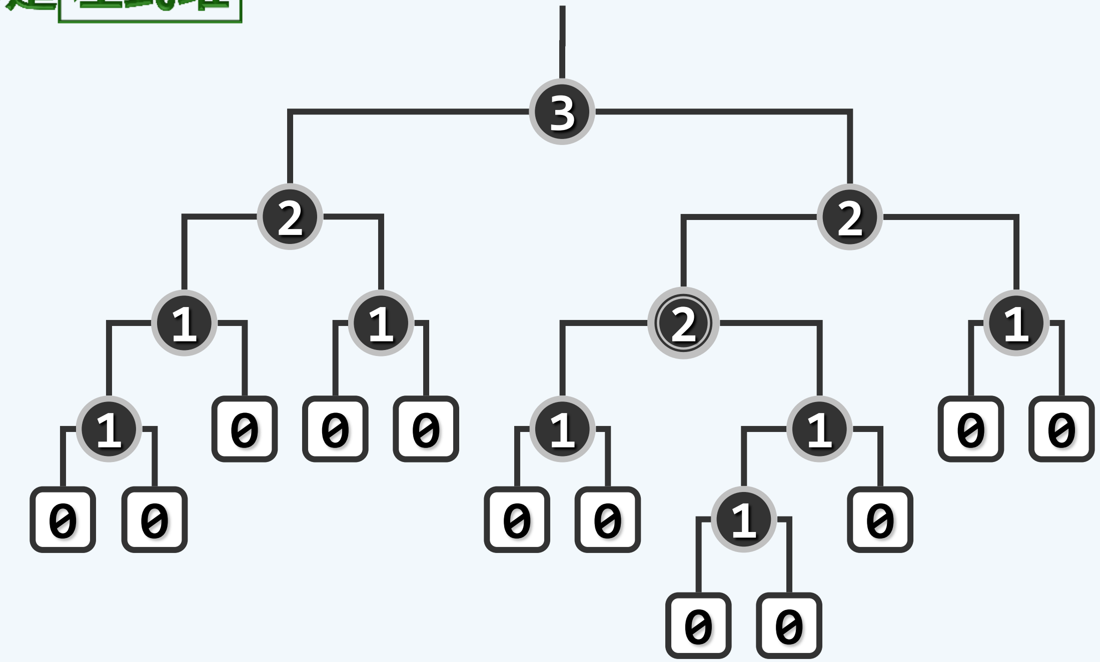
❖ 满足左倾性 leftist property 的堆，即是 左式堆

❖ 左倾性与堆序性，相容而不矛盾

❖ 左式堆的子堆，必是左式堆

❖ 左式堆倾向于

更多节点分布于左侧分支



❖ 这是否意味着，左子堆的规模和高度必然大于右子堆？

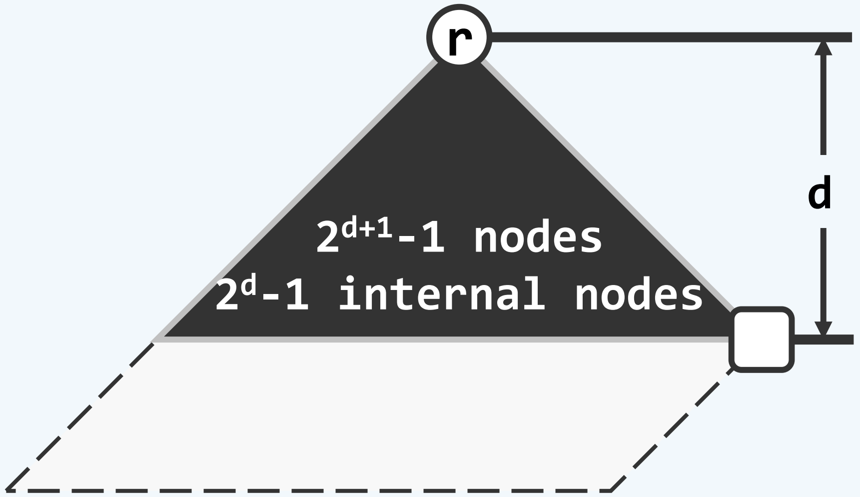
右侧链

- 右侧链 $\boxed{\text{rChain(x)}}$：从节点x出发，一直沿 $\boxed{\text{右分支}}$ 前进

- 特别地，rChain(root)的终点，必为全堆中最 $\boxed{\text{浅}}$ 的外部节点

  $npl(r) \equiv |\, rChain(r) \,| = d$

  存在一棵以r为根、高度为d的满子树



- 右侧链长为d的左式堆，$\boxed{\text{至少}}$ 包含

  $\boxed{2^d - 1}$ 个内部节点、$\boxed{2^{d+1} - 1}$ 个节点

- 反之，在包含n个节点的左式堆中

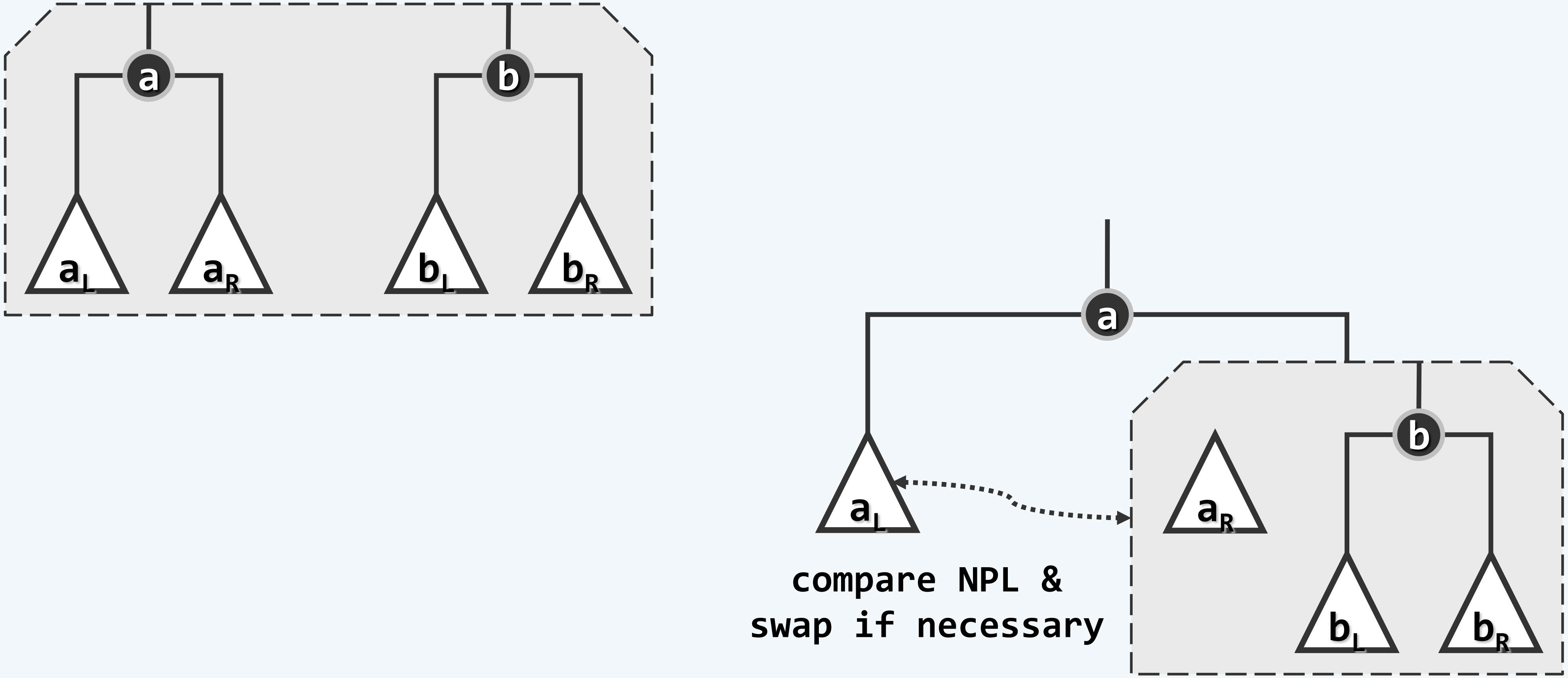
  右侧链的长度 $d \le \lfloor \log_2(n+1) \rfloor - 1 = \mathcal{O}(\log n)$

LeftHeap

❖ template <typename T> //基于二叉树，以左式堆形式实现的优先级队列

```
class PQ_LeftHeap : public PQ<T>, public BinTree<T> {
public:
    void insert(T); //（按比较器确定的优先级次序）插入元素
    T getMax() { return _root->data; } //取出优先级最高的元素
    T delMax(); //删除优先级最高的元素
}; //主要接口，均基于统一的合并操作实现...
```

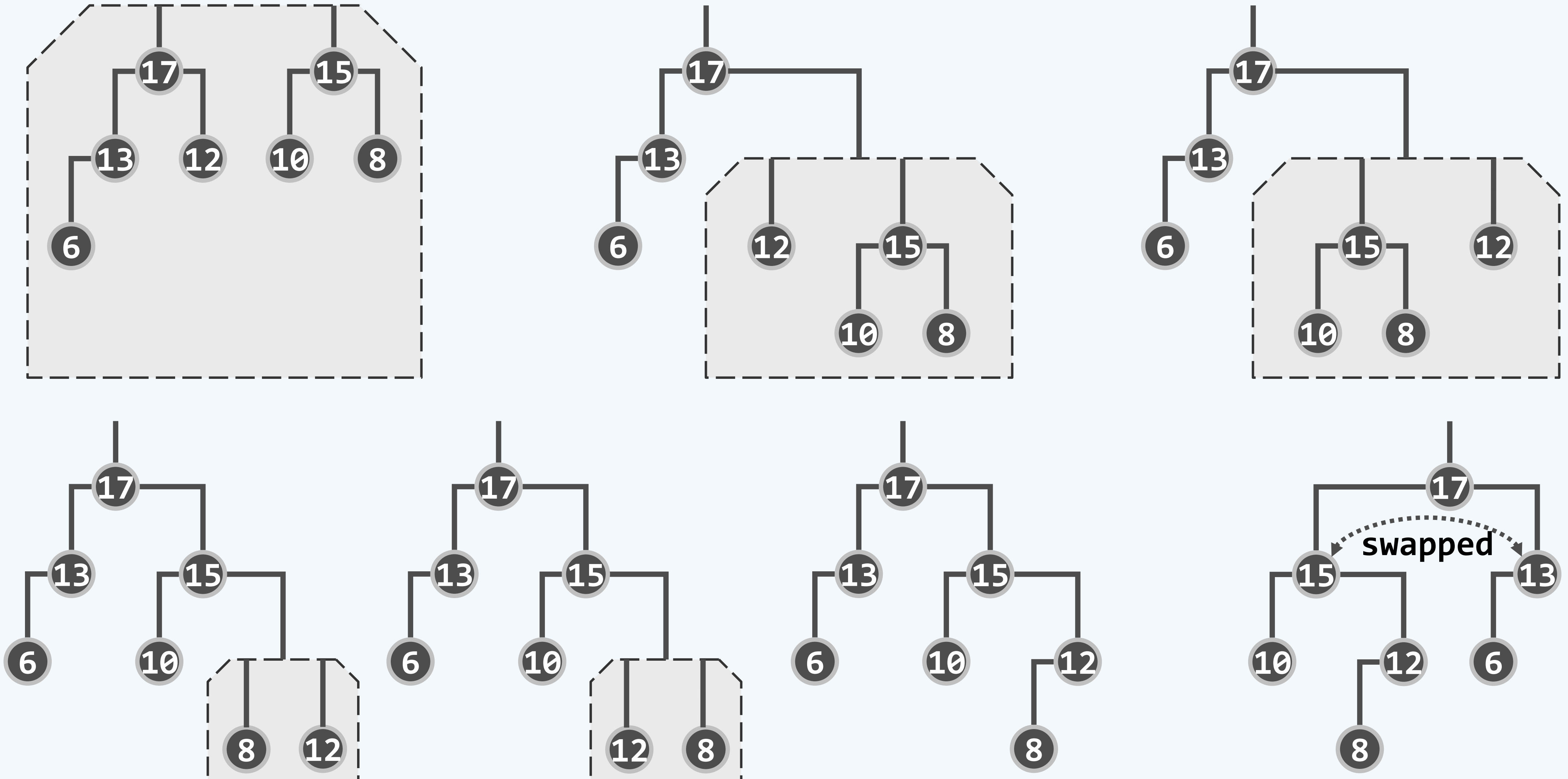
merge()：算法



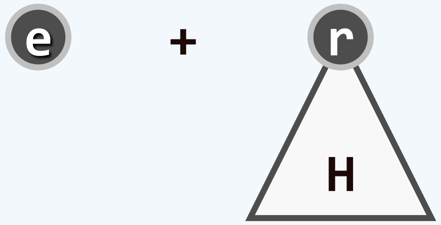
merge()：实现

```
template <typename T>
static BinNodePosi(T) merge( BinNodePosi(T) a, BinNodePosi(T) b ) {
   if ( ! a ) return b; //递归基
   if ( ! b ) return a; //递归基
   if ( lt( a->data, b->data ) ) swap( b , a ); //一般情况：首先确保b不大
   a->rc = merge( a->rc, b ); //将a的右子堆，与b合并
   a->rc->parent = a; //并更新父子关系
   if ( ! a->lc || a->lc->npl < a->rc->npl ) //若有必要
      swap( a->lc, a->rc ); //交换a的左、右子堆，以确保右子堆的npl不大
   a->npl = a->rc ? a->rc->npl + 1 : 1 ; //更新a的npl
   return a; //返回合并后的堆顶
}
```

merge()：实例



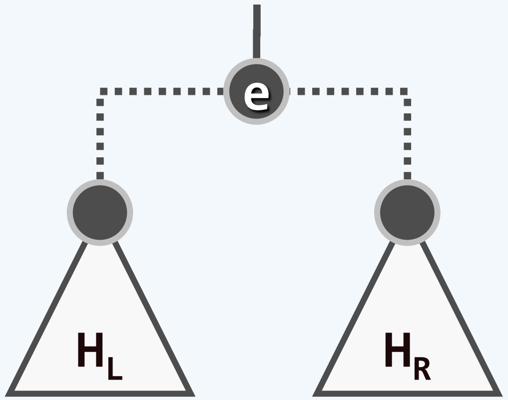
insert()




```
template <typename T>
void PQ_LeftHeap<T>::insert( T e ) { //O(logn)
   BinNodePosi(T) v = new BinNode<T>( e ); //为e创建一个二叉树节点
   _root = merge( _root, v ); //通过合并完成新节点的插入
   _root->parent = NULL; //既然此时堆非空，还需相应设置父子链接
   _size++; //更新规模
}
```

delMax()

```
❖ template <typename T> T PQ_LeftHeap<T>::delMax() { //O(logn)
    BinNodePosi(T) lHeap = _root->lc; //左子堆
    BinNodePosi(T) rHeap = _root->rc; //右子堆
    T e = _root->data; //备份堆顶处的最大元素
    delete _root; _size--; //删除根节点
    _root = merge( lHeap, rHeap ); //原左、右子堆合并
    if ( _root ) _root->parent = NULL; //更新父子链接
    return e; //返回原根节点的数据项
}
```

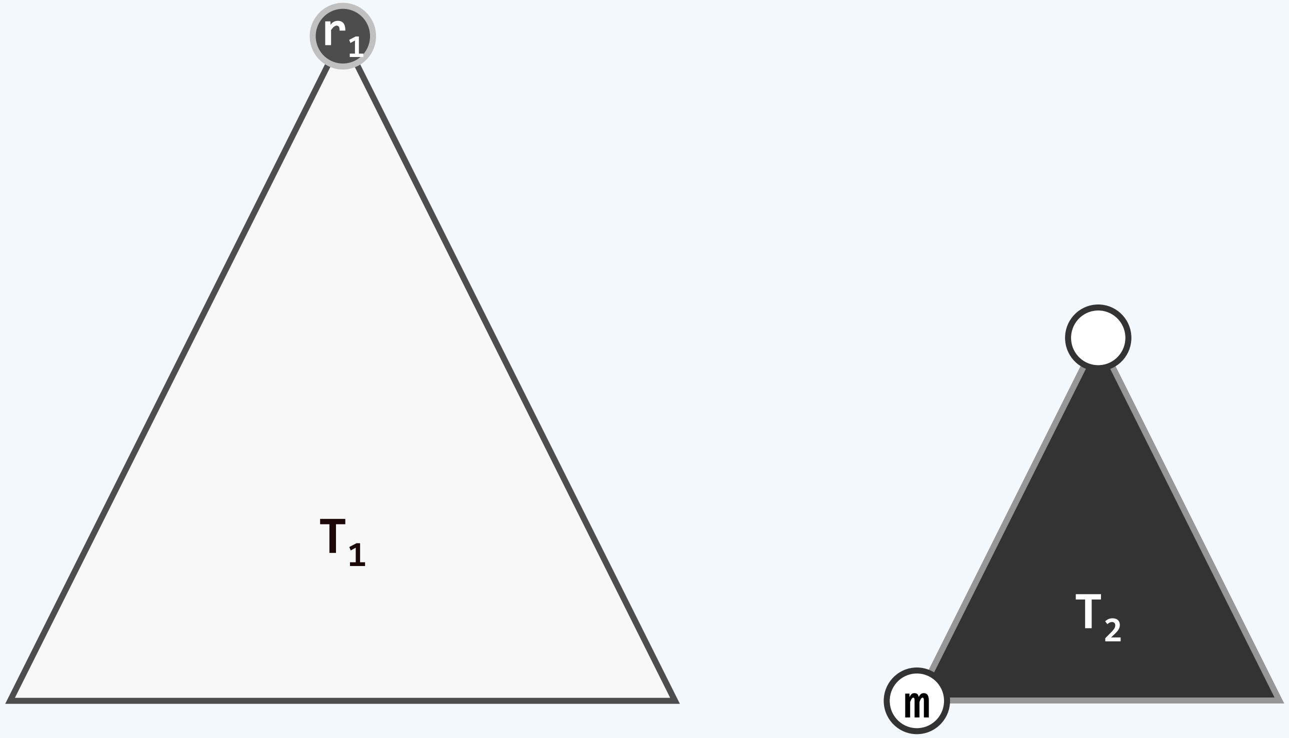


AVL::merge()

- 设T_1和T_2为两棵AVL树，且 $\boxed{\max(T_1) < m = \min(T_2)}$

  如何 $\boxed{\text{尽快}}$ 地将其合并为一棵AVL树？

- WLOG，$\boxed{\text{height}(T_1) \geq \text{height}(T_2)}$



AVL::merge()