

## 10. 优先级队列

### (c) 堆排序

邓俊辉

deng@tsinghua.edu.cn

## 选取

❖ 在 `selectionSort()` 中...

❖ 将 `U` 替换为 `H`

❖ J. Williams, 1964

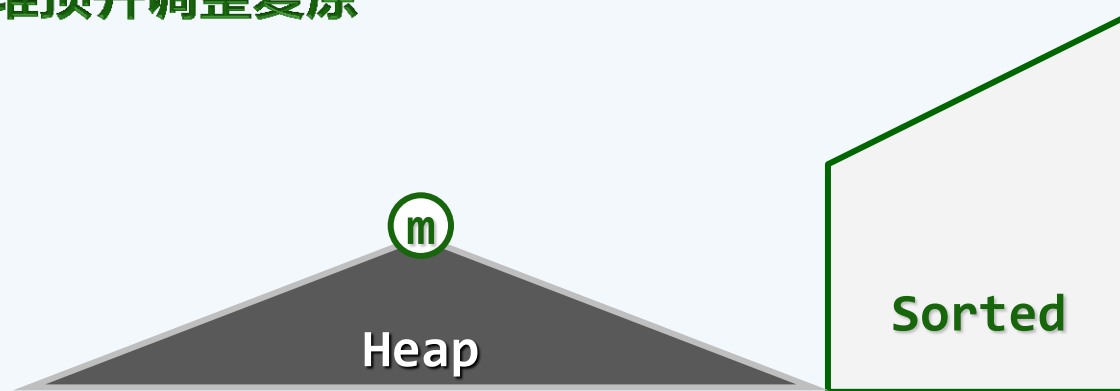
**初始化** : `heapify()`,  $O(n)$ , 建堆

**迭代** : `delMax()`,  $O(\log n)$ , 取出堆顶并调整复原

**不变性** :  $H \leq S$

❖ 等效于常规选择排序, 正确无疑

❖  $O(n) + n \times O(\log n) = O(n \log n)$



## 就地

❖ 在物理上

完全二叉堆 即是 向量

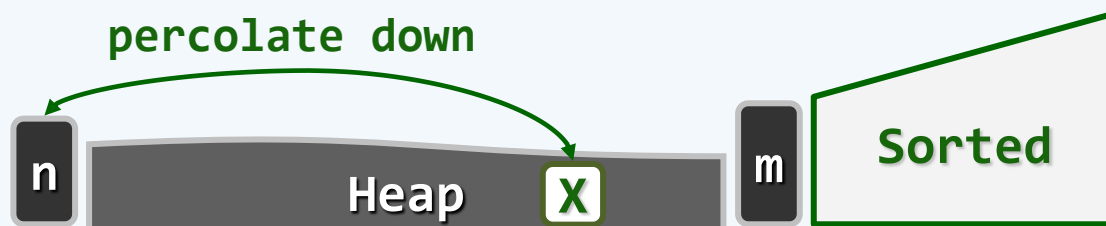
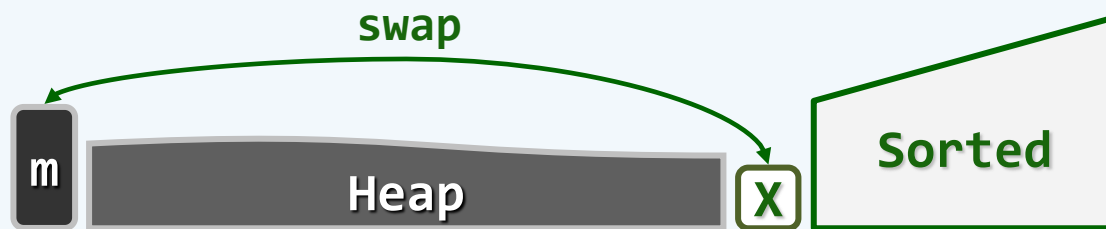
❖ 既然此前有：

$$m = H[0]$$

$$x = S[-1]$$

不妨随即就：

$$\text{swap}(m, x) = H.\text{insert}(x) + S.\text{insert}(m)$$



## 实现

❖ template <typename T> //对向量区间[lo, hi)做就地堆排序

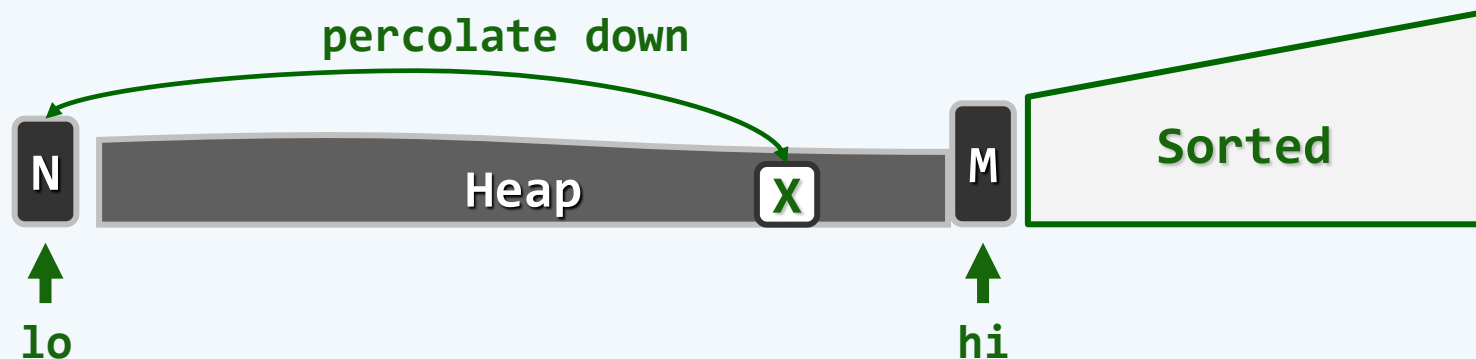
```
void Vector<T>::heapSort( Rank lo, Rank hi ) {
```

```
    PQ_ComplHeap<T> H( _elem + lo , hi - lo ); //待排序区间建堆, O(n)
```

```
    while ( ! H.empty() ) //反复地摘除最大元并归入已排序的后缀, 直至堆空
```

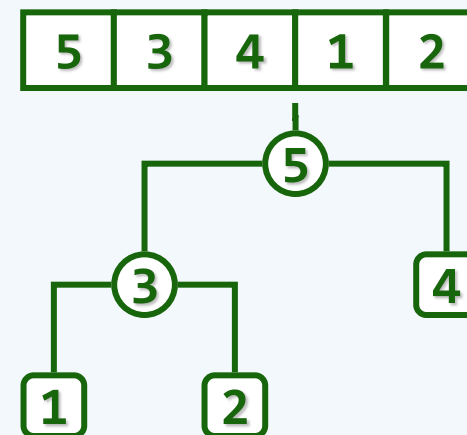
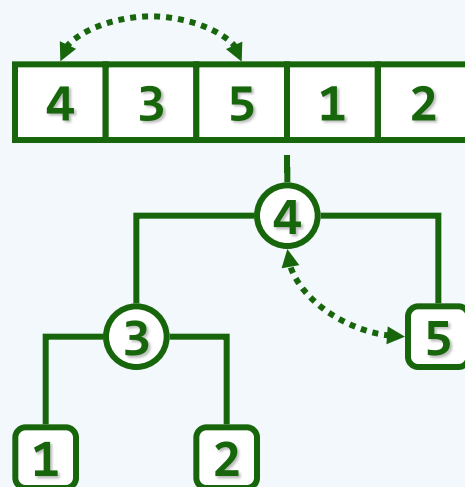
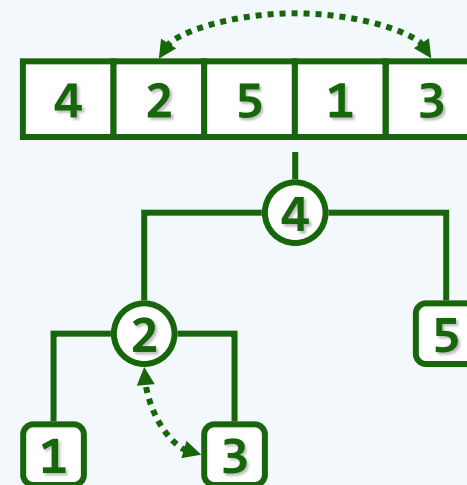
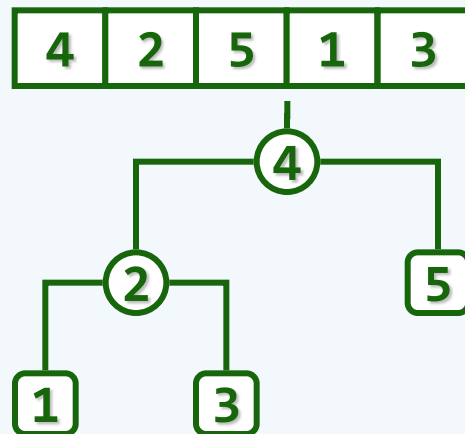
```
        _elem[ --hi ] = H.delMax(); //等效于堆顶与末元素对换后下滤
```

```
}
```

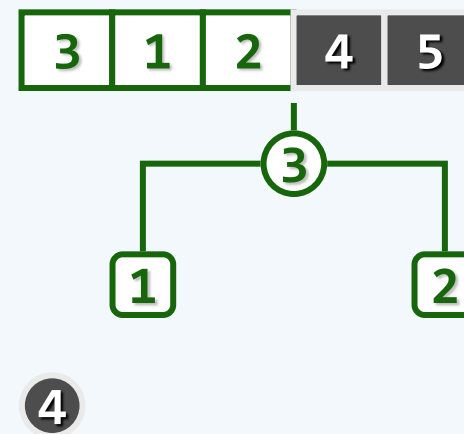
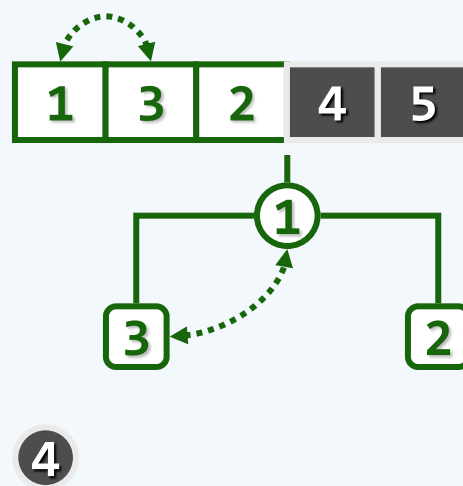
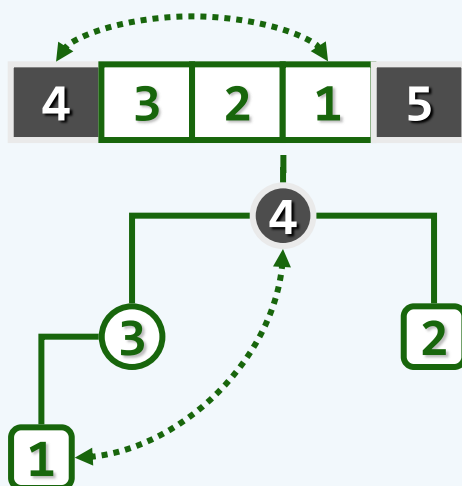
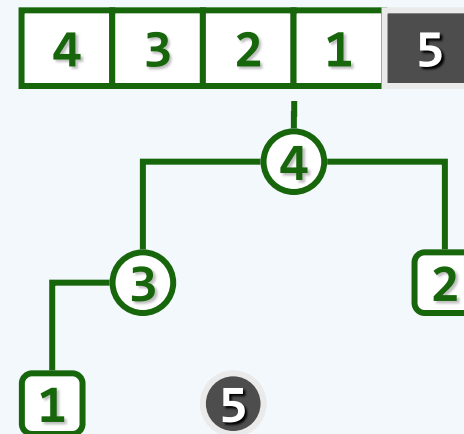
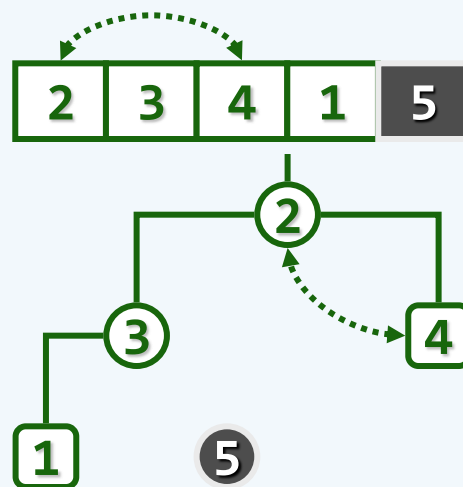
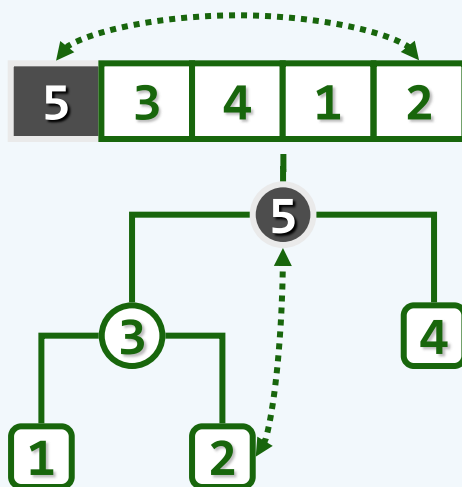


## 实例：建堆

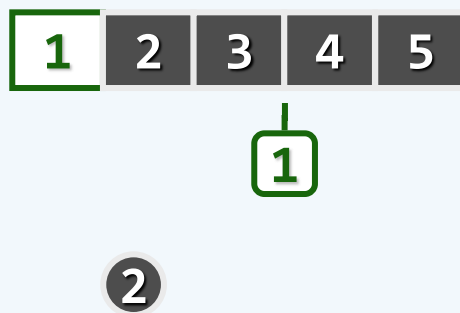
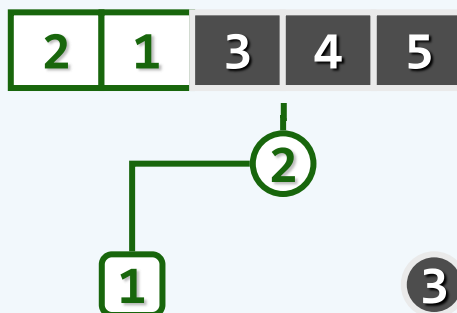
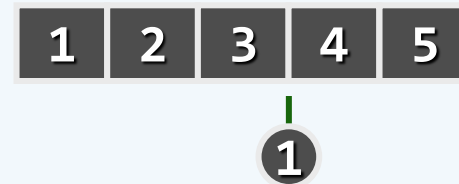
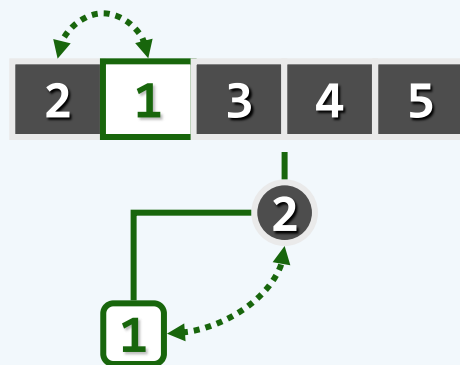
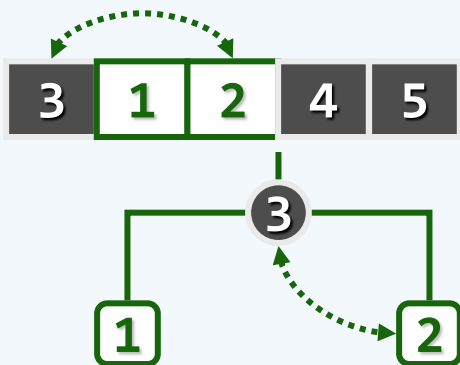
4 2 5 1 3



# 实例：选取 + 调整



## 实例：选取 + 调整



## 综合评价

❖ 易于理解，便于实现 //完全基于二叉堆结构及其操作接口

快速高效 //尤其适用于大规模数据

可就地运转

不需全排序即可找出前 $k$ 个词条 //  $O(k \log n)$ 的selection算法

❖ 不稳定 //为什么？可否克服？

❖ 权衡： 采用就地策略，是否值得？

固然可以节省一定的空间

但对换操作因此须涉及两个完整的词条，操作的单位成本增加