

12C 堆排序

#数据结构邓神

算法

选取 $\Leftarrow \text{getMax}()$

❖ 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** : $H \leq S$

Selection sort

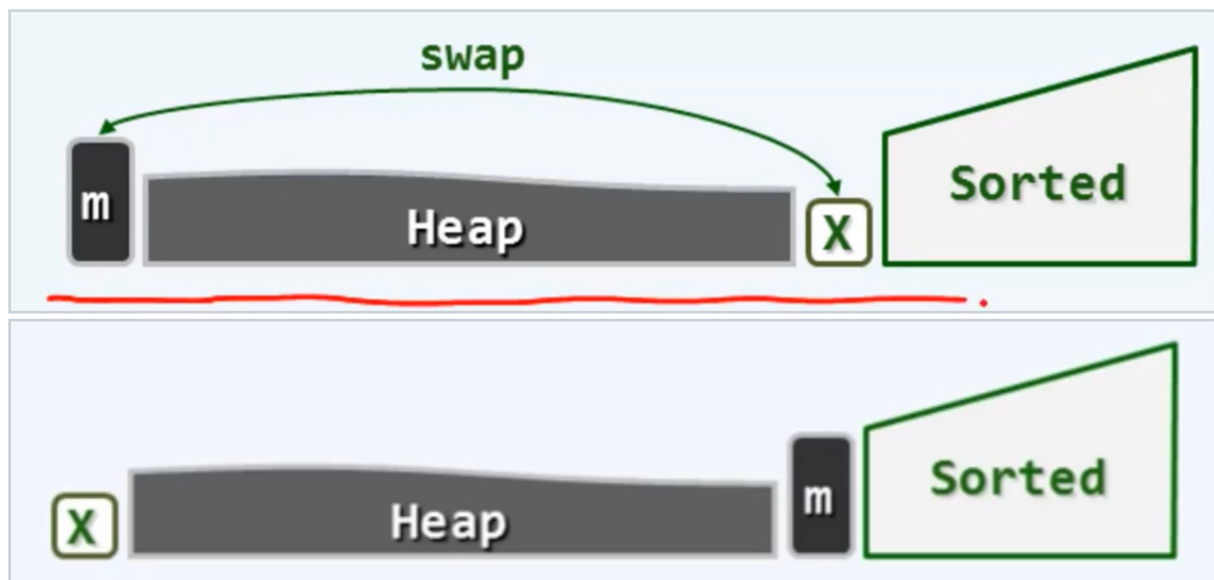
The diagram illustrates the Selection Sort algorithm. It shows an array divided into two parts: 'Unsorted' and 'Sorted'. In the 'Unsorted' part, a vertical bar labeled 'm' indicates the current element being compared. In the 'Sorted' part, a trapezoid represents the sorted elements. Below this, a 'Heap' is shown as a triangle with a circle labeled 'm' at its peak, representing the root of the heap. The 'Sorted' part is again shown as a trapezoid. The complexity $O(n^2)$ is written in red.

$O(n \log n)$ 时间

等效于常规选择排序，正确无疑

就地 只使用 $O(1)$ 的辅助空间

在物理上，完全二叉树就是向量



就地 $\leftarrow O(1)$

❖ 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：

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

第一步交换，第二部下滤，反复操作就可以。

实现

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

}

```
template <typename T> void heapSort(Rank lo, Rank hi){
    PQ_complHeap<T> H(_elem + lo, hi-lo);
    while (H.empty()) {
        _elem[--hi] = H.delMax();
    }
}
```