

3. 列表

(d) 选择排序

当下又选了几样果菜与凤姐送去，
凤姐儿也送了几样来。

- 红楼梦·第六十二回

邓俊辉

deng@tsinghua.edu.cn

构思

❖ 回忆起泡排序...

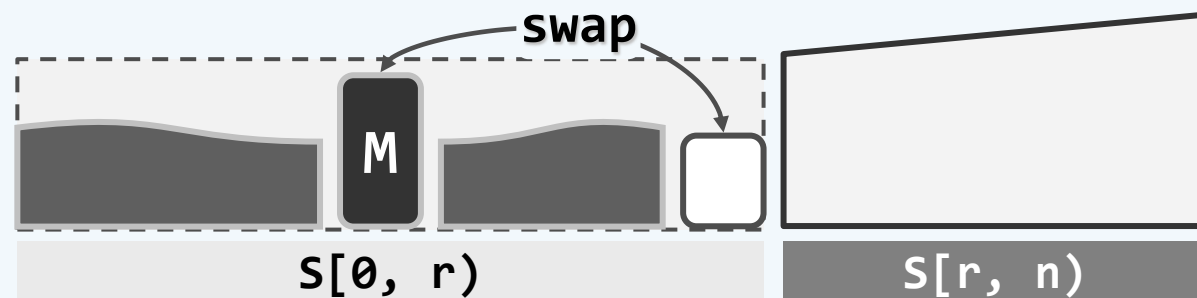
❖ 每经一趟扫描交换，当前的最大元素必然就位

//实质效果等同于...

从未排序的元素中**挑选**出最大者，并使之就位

//选择排序！不过...

❖ 起泡排序之所以需要 $O(n^2)$ 时间，是因为
为挑选每个最大元素 M ，需做
 $O(n)$ 次比较和 $O(n)$ 次交换



❖ $O(n)$ 次**比较**或许无可厚非，但
 $O(n)$ 次**交换**绝对没有必要



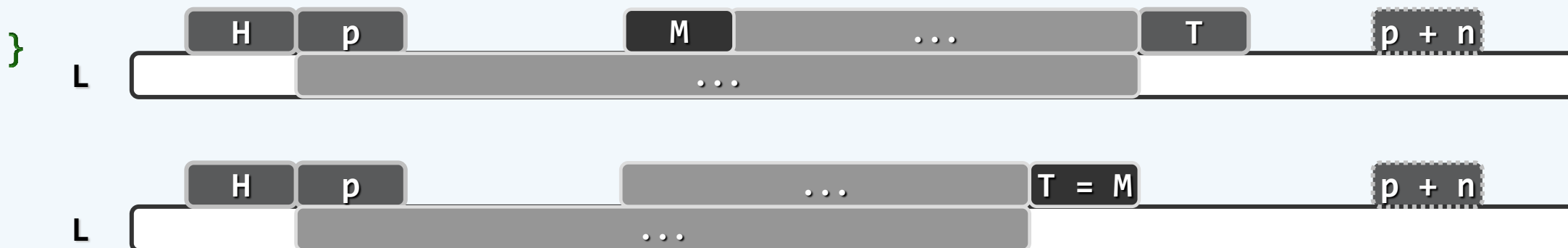
❖ 经 $O(n)$ 次比较确定 M 之后，**一次**交换足矣

实例

迭代轮次	前缀无序子序列							后缀有序子序列
0	5	2	7	4	6	3	1	^
1	5	2	4	6	3	1		7
2	5	2	4	3	1			6 7
3	2	4	3	1				5 6 7
4	2	3	1					4 5 6 7
5	2	1						3 4 5 6 7
6	1							2 3 4 5 6 7
7	^							1 2 3 4 5 6 7

实现：selectionSort()

```
//对列表中起始于位置p的连续n个元素做选择排序, valid(p) && rank(p) + n <= size
template <typename T> void List<T>::selectionSort( Posi(T) p, int n ) {
    Posi(T) head = p->pred; Posi(T) tail = p; //待排序区间(head, tail)
    for ( int i = 0; i < n; i++ ) tail = tail->succ; //head/tail可能是头/尾哨兵
    while ( 1 < n ) { //反复从 ( 非平凡 ) 待排序区间内找出最大者, 并移至有序区间前端
        insertBefore( tail, remove( selectMax( head->succ, n ) ) );
        tail = tail->pred; n--; //待排序区间、有序区间的范围, 均同步更新
    }
}
```

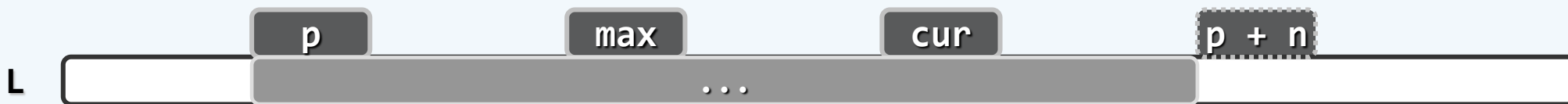


实现：selectMax()

```

❖ template <typename T> //从起始于位置p的n个元素中选出最大者, 1 < n
    Posi(T) List<T>::selectMax( Posi(T) p, int n ) { //Θ(n)
        Posi(T) max = p; //最大者暂定为p
        for ( Posi(T) cur = p; 1 < n; n-- ) //后续节点逐一与max比较
            if ( !lt( ( cur = cur->succ )->data, max->data ) ) //若 >= max
                max = cur; //则更新最大元素位置记录
        return max; //返回最大节点位置
    }

```



❖ 稳定性：在有重复元素时，采用比较器 `!lt()` 或 `ge()` 等效于 后者优先

2	6a	4	6b	3	0	6c	1	5	7	8	9
2	6a	4	6b	3	0	1	5	6c	7	8	9

性能

❖ 共迭代 n 次，在第 k 次迭代中

selectMax()为 $\Theta(n - k)$

remove()和insertBefore()均为 $\mathcal{O}(1)$

故总体复杂度应为 $\Theta(n^2)$

❖ 尽管如此，元素移动操作远远少于起泡排序

//实际更为费时

也就是说， $\Theta(n^2)$ 主要来自于元素比较操作

//成本相对更低

❖ 可否...每轮只做 $\mathcal{O}(n)$ 次比较，即找出当前的最大元素？

❖ 可以！...利用高级数据结构，selectMax()可改进至 $\mathcal{O}(\log n)$

//稍后分解

当然，如此立即可以得到 $\mathcal{O}(n \log n)$ 的排序算法

//保持兴趣