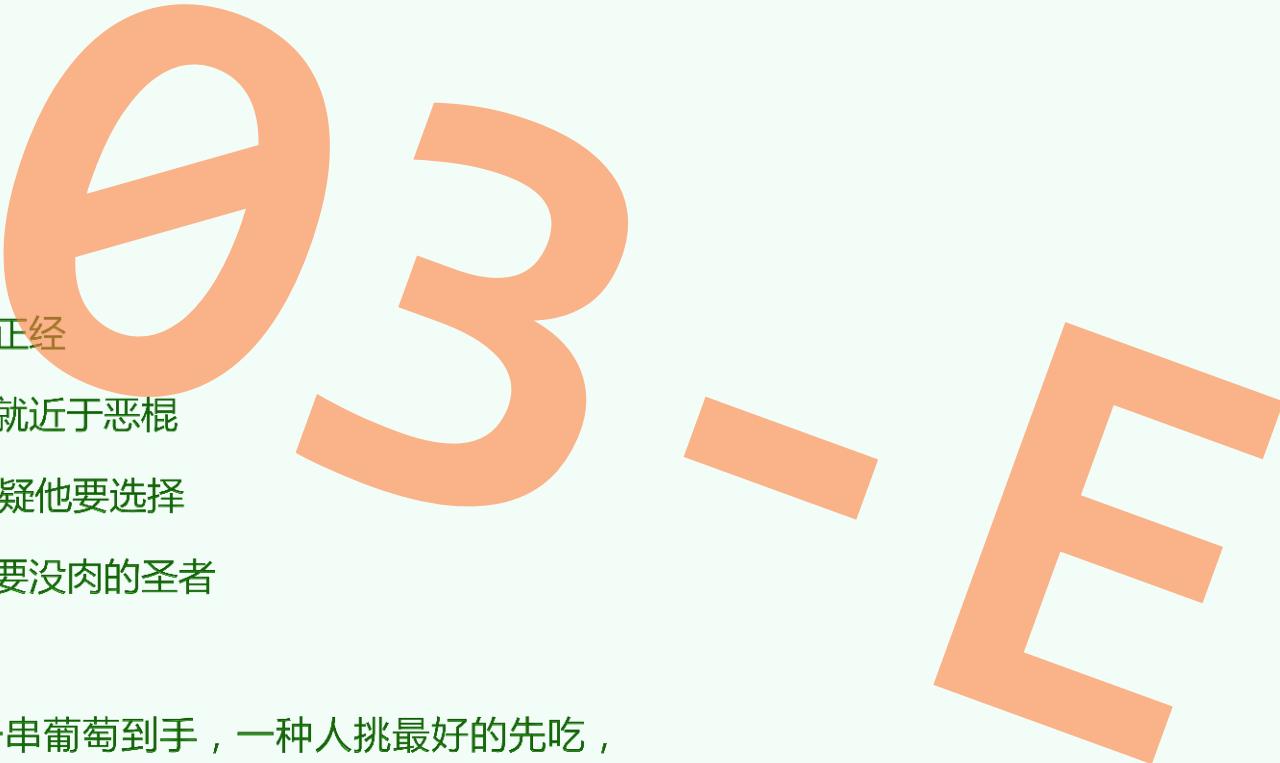


列表

选择排序

邓俊辉

deng@tsinghua.edu.cn



卡修斯永远讲道德，永远正经

他认为容忍恶棍的人自己就近于恶棍

只有在吃饭的时候——无疑他要选择

一个有鹿肉的坏蛋，而不要没肉的圣者

天下只有两种人。譬如一串葡萄到手，一种人挑最好的先吃，

另一种人把最好的留在最后吃。

起泡排序：温故知新

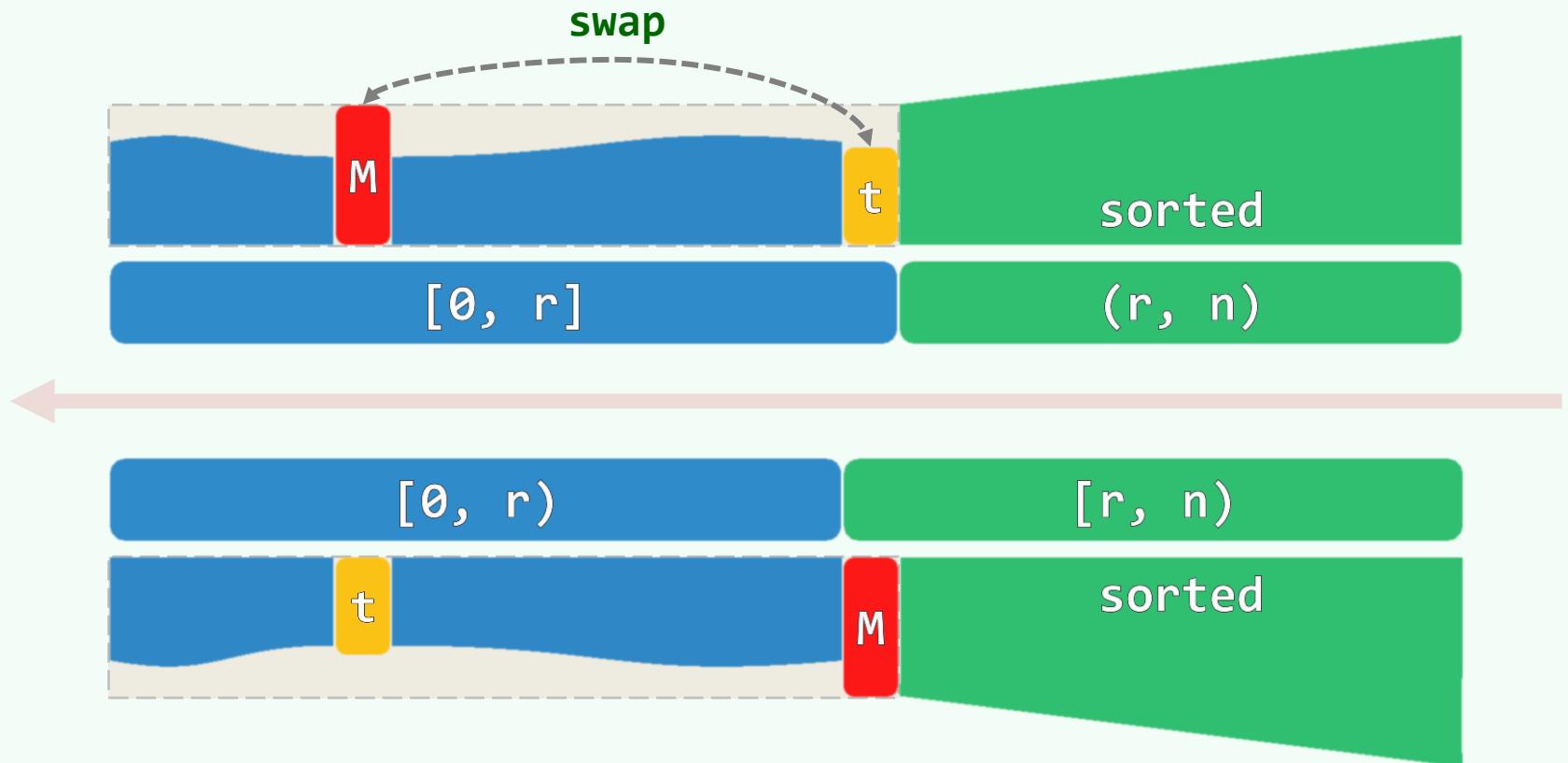
每趟扫描交换都需 $\Theta(n)$ 次比较、 $\Theta(n)$ 次交换；然而其中， $\Theta(n)$ 次交换完全没有必要

扫描交换的实质效果无非是

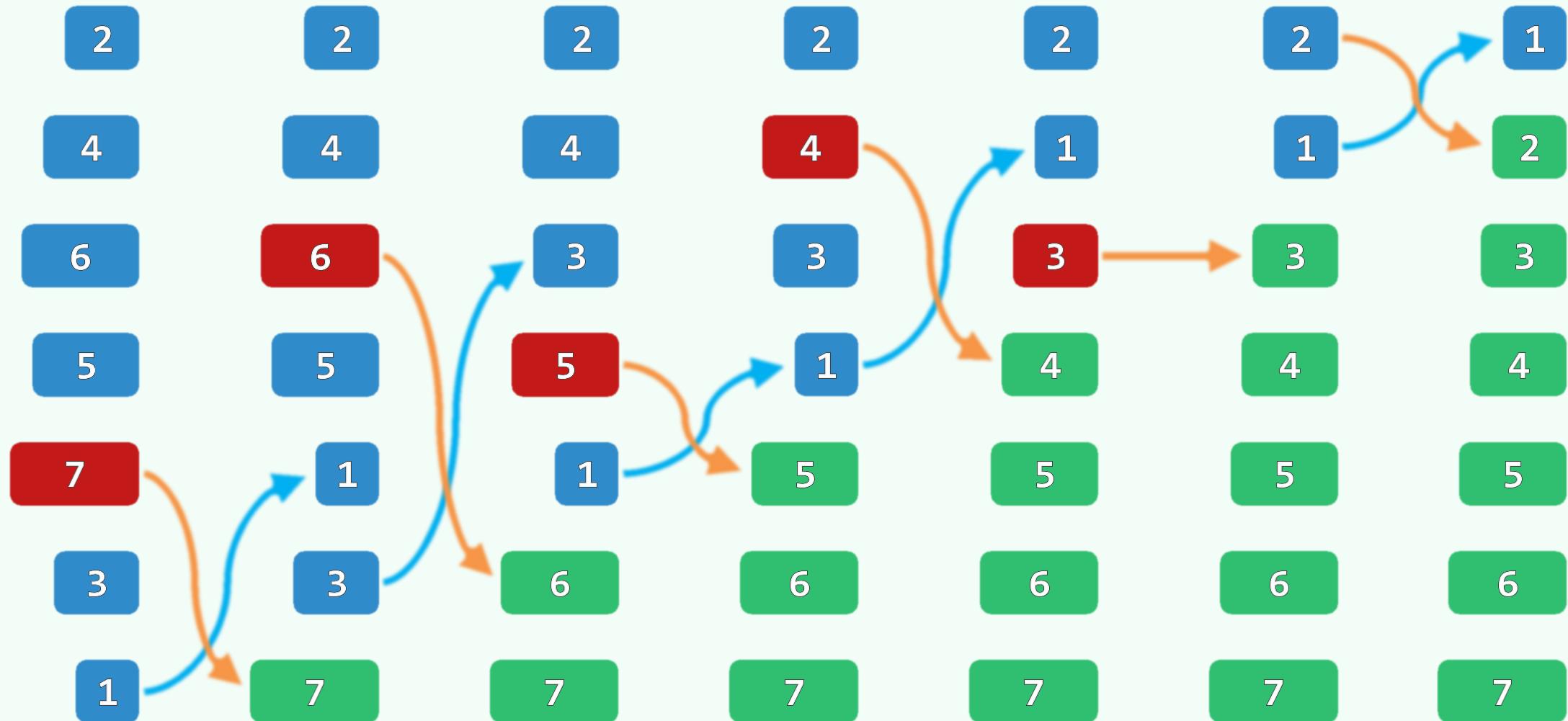
- 通过比较找到当前的最大元素M，并
- 通过交换使之就位

如此看来

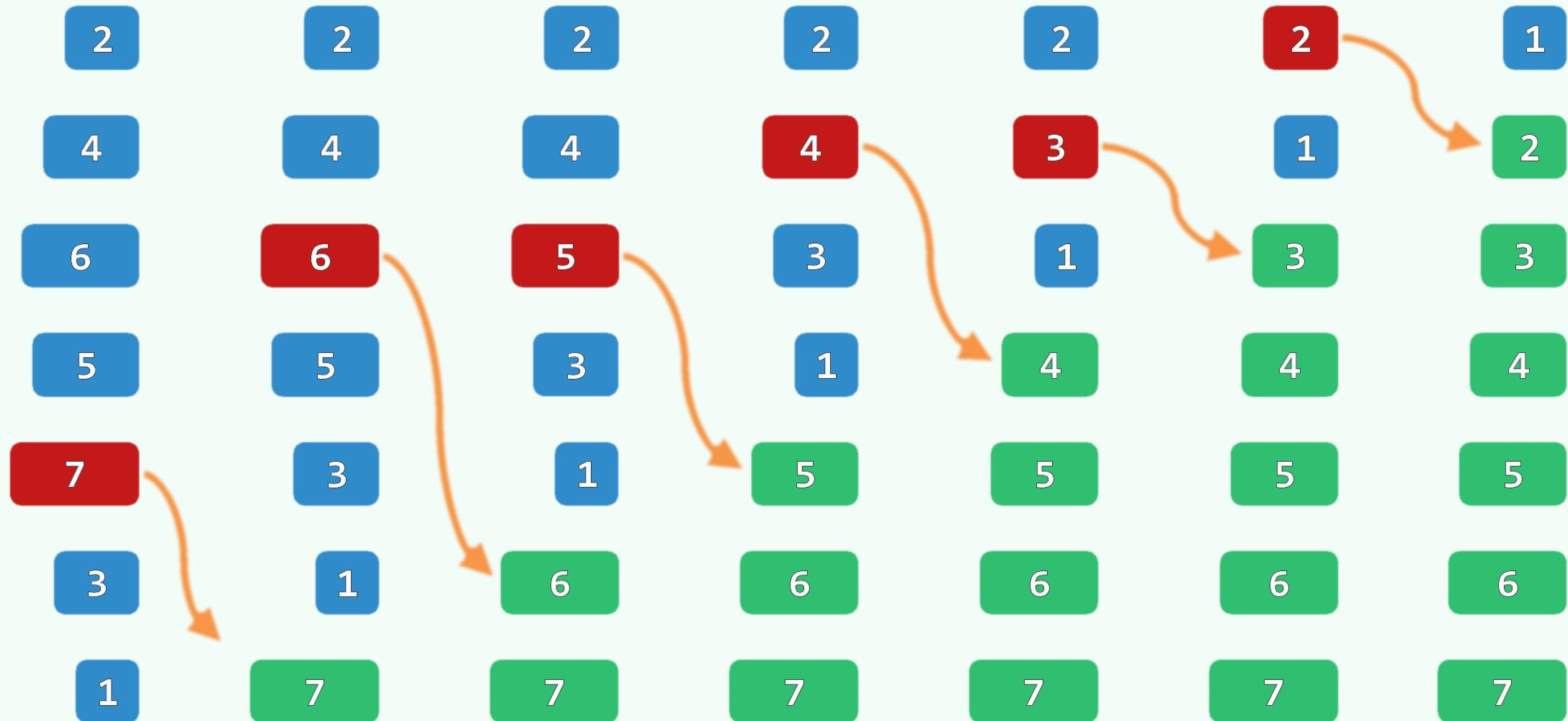
在经 $\Theta(n)$ 次比较确定M之后，仅需一次交换即足矣



交换法

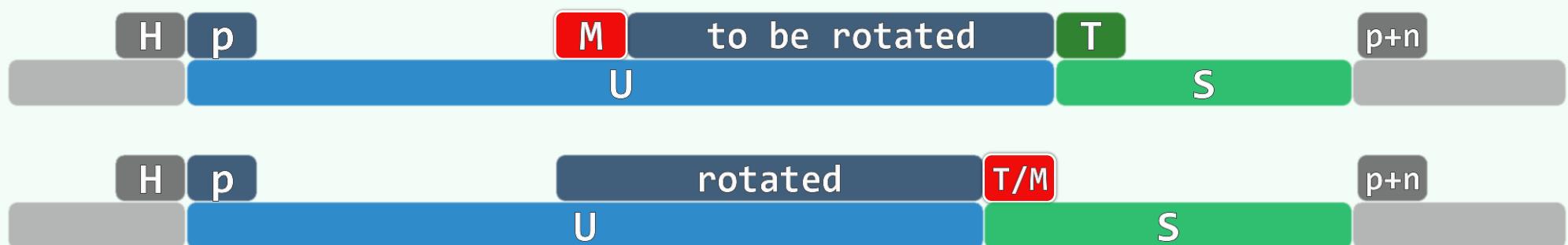


平移法



selectionSort()

```
//对列表中起始于位置p、宽度为n的区间做选择排序，valid(p) && rank(p) + n <= size  
template <typename T> void List<T>::selectionSort( ListNodePosi(T) p, int n ) {  
    ListNodePosi(T) head = p->pred; ListNodePosi(T) tail = p;  
    for ( int i = 0; i < n; i++ ) tail = tail->succ; //待排序区间为(head, tail)  
    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  
        insertB( tail, remove( selectMax( head->succ, n ) ) ); //改进...  
        tail = tail->pred; n--; //待排序区间、有序区间的范围，均同步更新  
    }  
}
```



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 ) ) //data ≥ 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

- ◆ 有多个重复元素同时命中时，往往需要按照某种**附加的约定**，返回其中**特定的某一个**
- ◆ 比如，通常都约定“**靠后者优先返回**”
- ◆ 为此，必须采用**比较器!lt()**或**ge()**，即等效于**后者优先**

2	6a	4	6b	3	0	<u>6c</u>	1	5	7	8	9
2	6a	4	6b	3	0	1	5	6c	7	8	9
2	6a	4	3	0	1	5	6b	6c	7	8	9
2	4	3	0	1	5	6a	6b	6c	7	8	9

- ◆ 若采用平移法，如此即可保证，重复元素在列表中的相对次序，与其插入次序一致



性能分析

◆ 共迭代 n 次，在第 k 次迭代中

- selectMax() 为 $\Theta(n - k)$ //算术级数
- swap() 为 $\mathcal{O}(1)$ //或 remove() + insertB()

故总体复杂度应为 $\Theta(n^2)$

◆ 尽管如此，元素的移动操作远远少于起泡排序 //实际更为费时

也就是说， $\Theta(n^2)$ 主要来自于元素的比较操作 //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