3.列表

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。

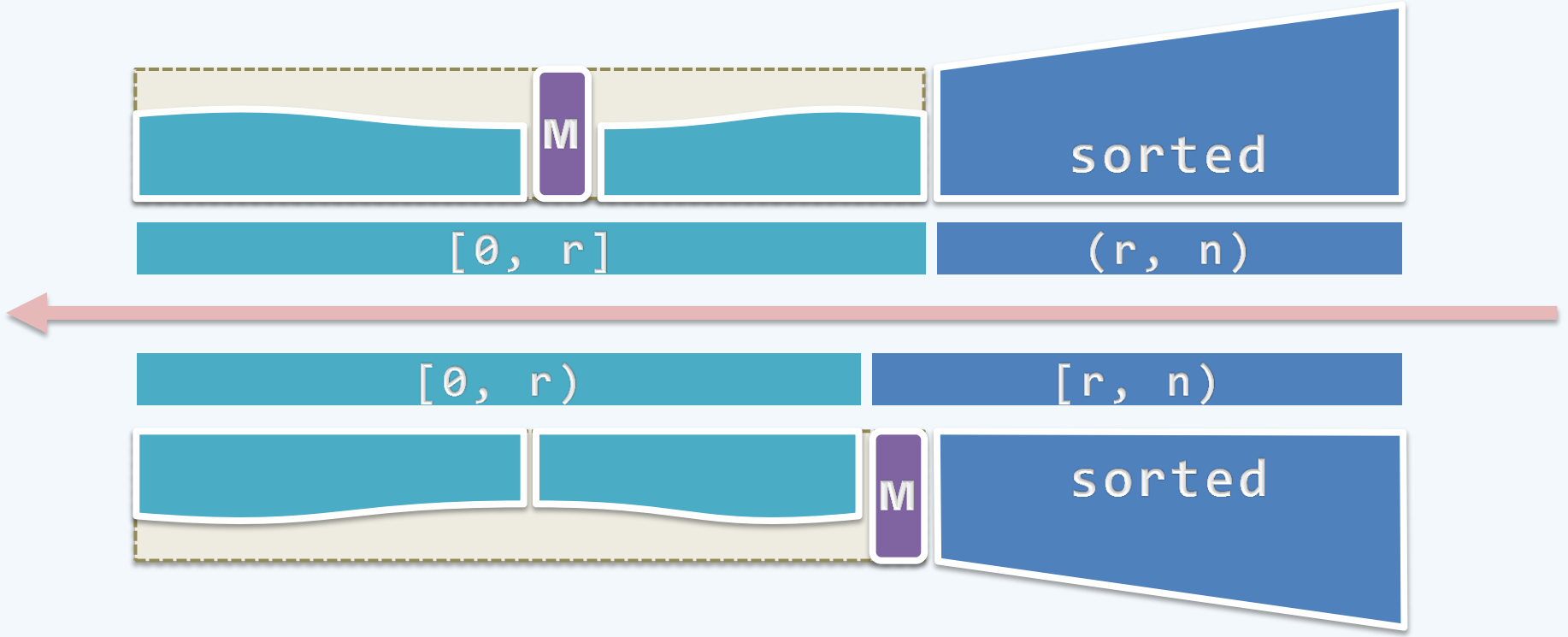
邓俊辉

deng@tsinghua.edu.cn

回忆起泡排序...

❖ 累计需要$O(n^2)$时间，是因为每趟扫描交换都需要$O(n)$时间：$O(n)$次比较 + $O(n)$次交换

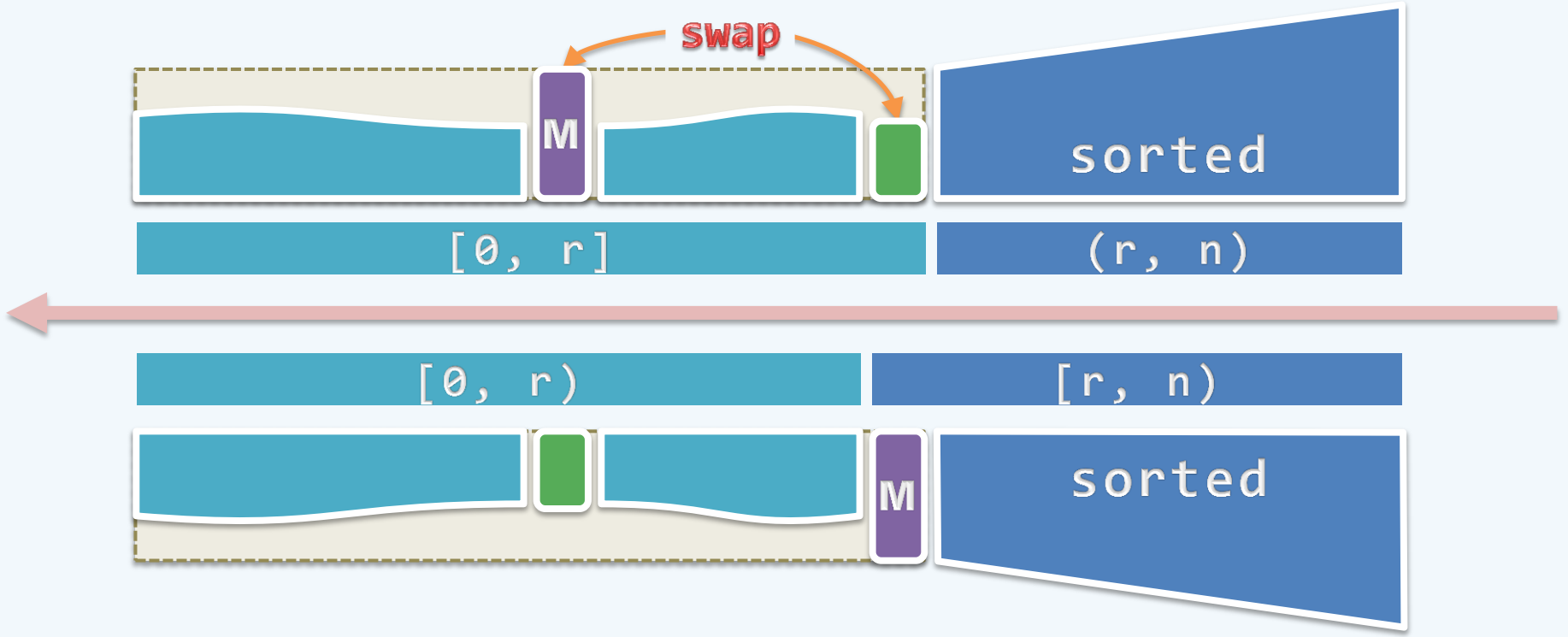
❖ $O(n)$次比较或许无可厚非，但$O(n)$次交换绝对没有必要 //大量无谓的逆向移动



回忆起泡排序...

❖ 每趟扫描交换的实质效果，无非就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O(n)次比较确定M之后，仅需一次交换即足矣

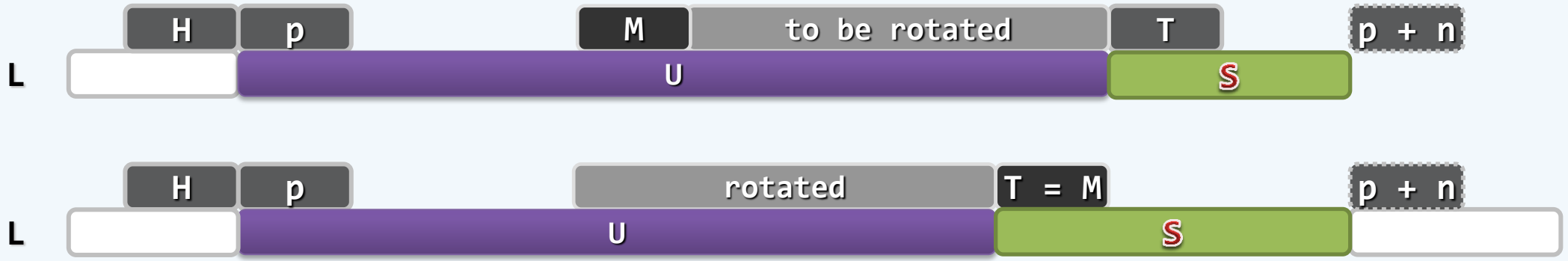


实例

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

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
      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 ) ) //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

- 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   6c  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
```

- 如此即可保证，重复元素**在列表中的次序**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 Θ(n - k)　　//算术级数

  swap() 为 O(1)　　//或 remove() + insertB()

  故总体复杂度应为Θ(n²)

- 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，Θ(n²)主要来自于元素比较操作　　//成本相对更低

- 可否...每轮只做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O(logn)　　//稍后分解

  当然，如此立即可以得到O(nlogn)的排序算法　　//保持兴趣