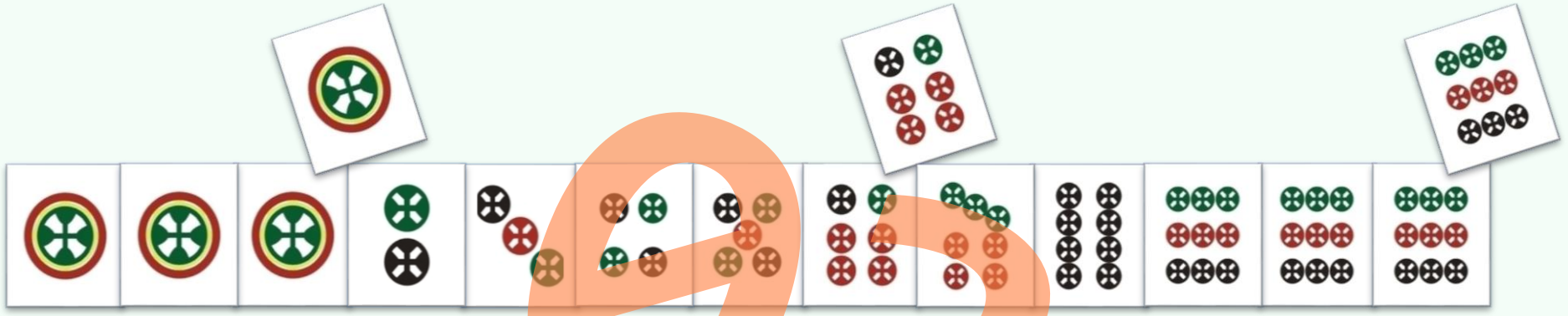


列表

插入排序

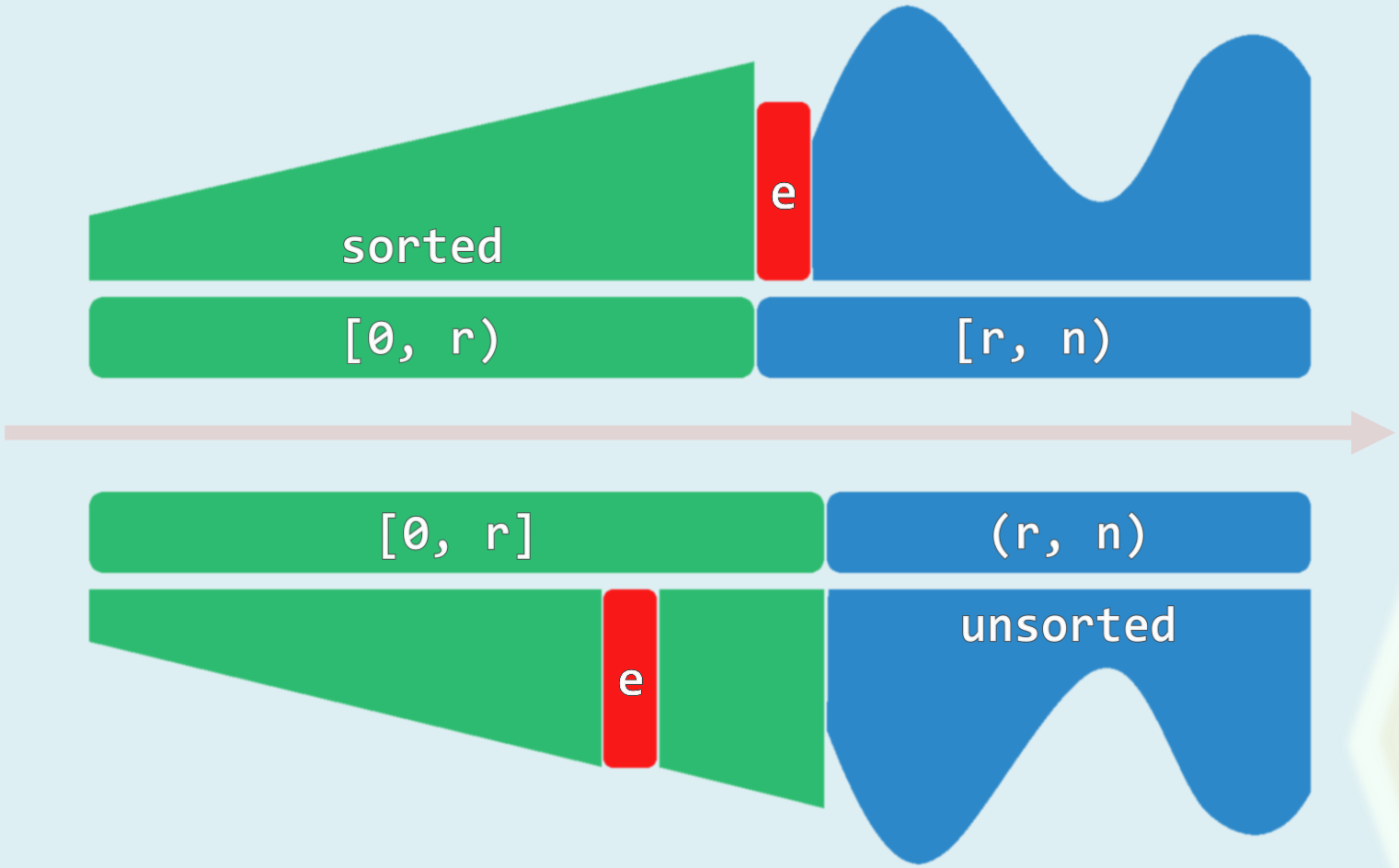
03-G

一语未了，只见宝玉笑嘻嘻的掮了一枝红梅进来，众丫鬟忙已接过，插入瓶内。

邓俊辉

deng@tsinghua.edu.cn

减而治之



❖ 不变性

序列总能视作两部分：

S[0, r) + U[r, n)

❖ 初始化

|S| = r = 0

❖ 反复地，针对e = A[r]

在S中查找适当位置，以插入e

❖ 二分查找？顺序查找？

实例

迭代轮次	前缀有序子序列	当前元素	后缀无序子序列
-1	^	^	5 2 7 4 6 3 1
0	^	5	2 7 4 6 3 1
1	(5)	2	7 4 6 3 1
2	(2) 5	7	4 6 3 1
3	2 5 (7)	4	6 3 1
4	2 (4) 5 7	6	3 1
5	2 4 5 (6) 7	3	1
6	2 (3) 4 5 6 7	1	^
7	(1) 2 3 4 5 6 7	^	^

实现

```
//对列表中起始于位置p、宽度为n的区间做插入排序，valid(p) && rank(p) + n <= size
  template <typename T> void List<T>::insertionSort( ListNodePosi<T> p, int n ) {
     for ( int r = 0; r < n; r++ ) { //逐一引入各节点，由Sr得到Sr+1
        insert( search( p->data, r, p ), p->data ); //查找 + 插入
        p = p->succ; remove( p->pred ); //转向下一节点
     } //n次迭代，每次O(r + 1)
  } //仅使用O(1)辅助空间，属于就地算法
```

- 紧邻于search()接口返回的位置之后插入当前节点，总是保持有序
- 验证各种情况下的正确性，体会哨兵节点的作用：

  S_r中含有/不含与p相等的元素；S_r中的元素均严格小于/大于p

性能分析

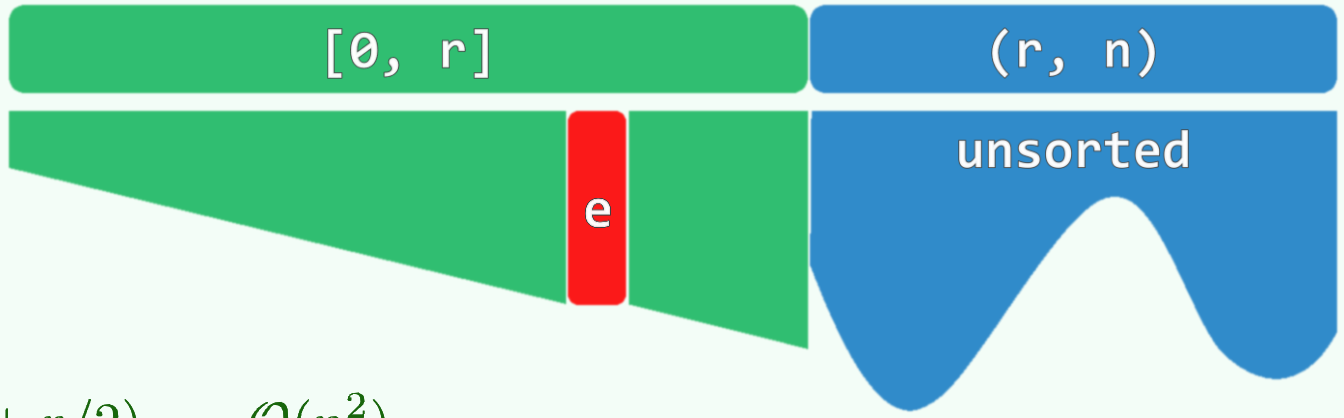
- 属于就地算法（in-place）
- 属于在线算法（online）：牌戏中不二的选择
- 具有输入敏感性（input sensitivity）
  - 后面将会看到：Shellsort之类算法的高效性，完全依赖于insertionsort的这一特性
- 最好情况：完全（或几乎）有序
  - 每次迭代，只需1次比较，0次交换：累计$O(n)$时间！
- 最坏情况：完全（或几乎）逆序
  - 第k次迭代，需$O(k)$次比较，1次交换：累计$O(n^2)$时间！
- “优化”的可能：在有序前缀中的查找定位，为何采用了顺序查找，而不是二分查找？

平均性能：后向分析

- 若各元素的取值系独立均匀分布，平均要做多少次元素比较？
- 考查e=[r]刚插入完成的那一时刻...此时的有序前缀[0,r]中，谁是e？
- 观察：其中的r+1个元素均有可能，且概率均为1/(r+1)
- 因此，在刚完成的这次迭代中

  为引入S[r]所花费时间的数学期望为

$$1 + \sum_{k=0}^{r} k/(r+1) \;=\; 1 + r/2$$



- 于是，总体时间的数学期望为 $\sum_{r=0}^{n-1} (1 + r/2) \;=\; \mathcal{O}(n^2)$
- 再问：在n次迭代中，平均有多少次无需交换呢？ //习题[3-10]