

3. 列表 插入排序

一语未了，只见宝玉笑嘻嘻的掇了一枝红
梅进来，众丫鬟忙已接过，插入瓶内。

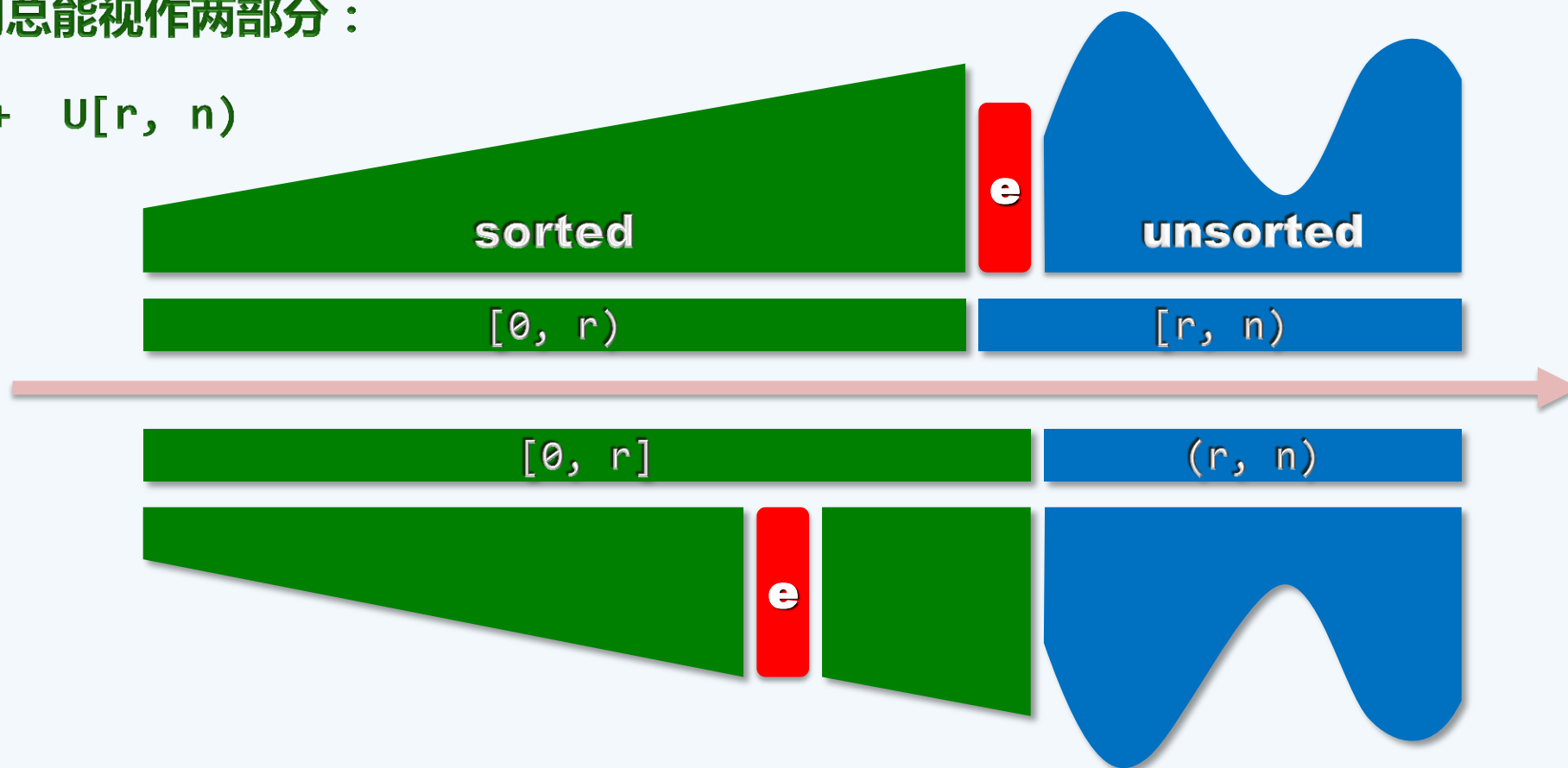
邓俊辉

deng@tsinghua.edu.cn

构思

❖ 【不变性】 序列总能视作两部分：

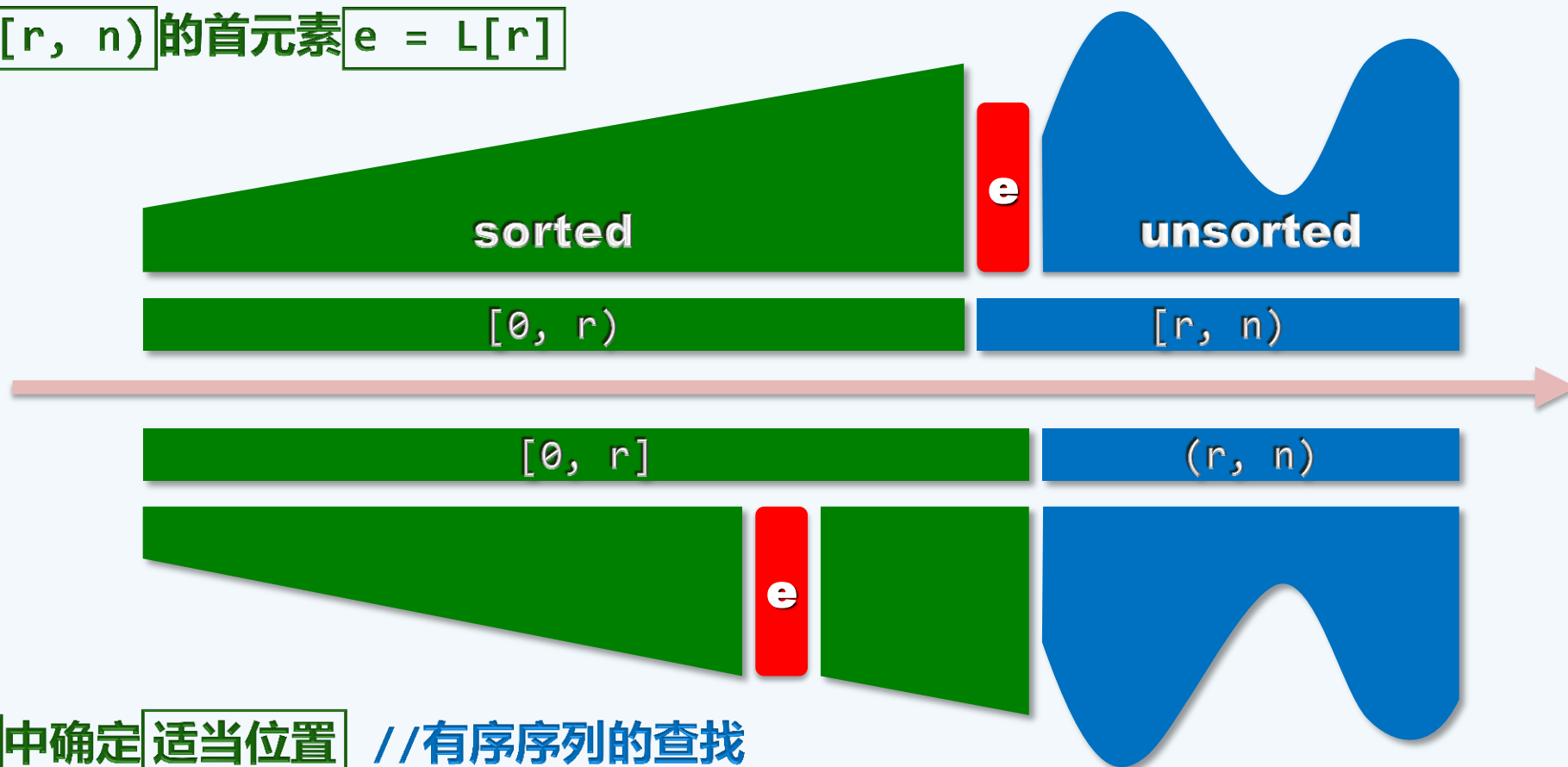
$$S[0, r) + U[r, n)$$



❖ 【初始化】： $|S| = r = 0$ //空序列无所谓有序或无序

减而治之

❖ 反复地，针对 $U[r, n)$ 的首元素 $e = L[r]$



- 在 $S[0, r)$ 中确定 **适当位置** //有序序列的查找
- 插入 e , 得到 $S[0, r]$ //有序序列的插入

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的连续n个元素做插入排序, $\text{valid}(p) \ \&\& \ \text{rank}(p) + n \leq \text{size}$

```
template <typename T> void List<T>::insertionSort( Posi(T) p, int n ) {
```

```
    for ( int r = 0; r < n; r++ ) { //逐一引入各节点, 由 $s_r$ 得到 $s_{r+1}$ 
```

```
        insertA( 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:

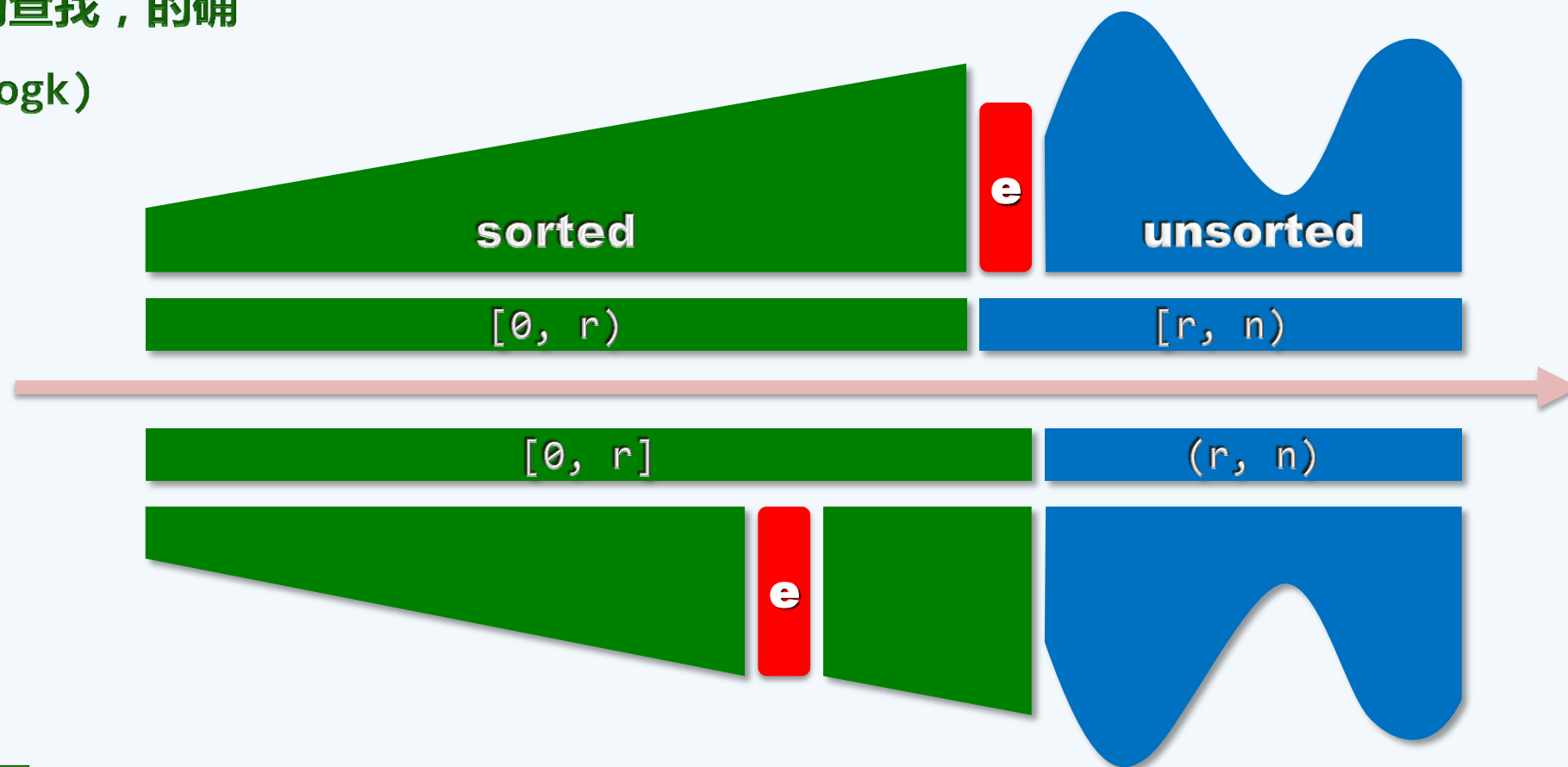
s_r 中含有/不含与p相等的元素; s_r 中的元素均严格小于/大于p

性能分析

- ❖ 属于就地算法 (in-place)
- ❖ 属于在线算法 (online)
- ❖ 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？

❖ 在有序前缀中的查找，的确
不难优化至 $O(\log k)$



❖ 然而，**总体性能**会否因此
相应地提高？

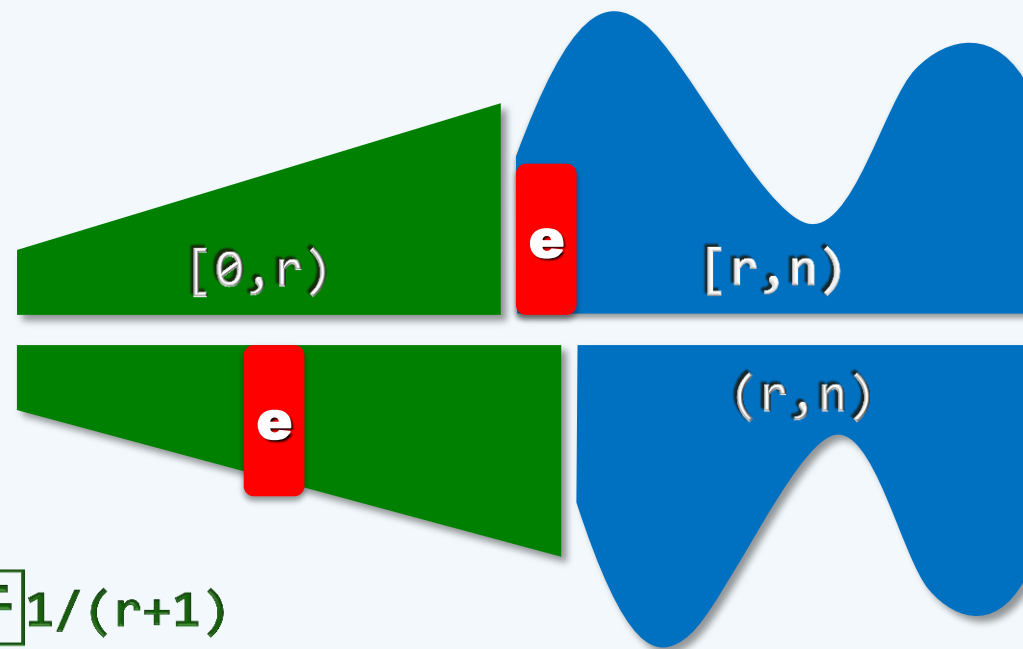
平均性能：后向分析

❖ 假定：各元素的取值遵守均匀、独立分布

于是：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（穿越？！）

试问：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，且概率均等于 $1/(r+1)$

❖ 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

$$[r + (r-1) + \dots + 3 + 2 + 1 + 0] / (r+1) + 1 = r/2 + 1$$

❖ 于是，总体时间的数学期望 = $[0 + 1 + \dots + (n-1)] / 2 + n = O(n^2)$

❖ 再问：在 n 次迭代中，平均有多少次无需交换呢？

//习题[3-10]