

2. 向量

(d1) 有序向量

- 唯一化

贾政道：“我要你另换个主意，不许雷同了前人，只做个破题也使得。” 宝玉只得答应着，低头搜索枯肠

邓俊辉

deng@tsinghua.edu.cn

有序性及其甄别

❖ 与起泡排序算法的理解相同：**有序**/**无序**序列中，**任意**/**总有**一对相邻元素**顺序**/**逆序**

❖ 因此，相邻逆序对的数目，可用以**度量**向量的**逆序**程度

❖ `template <typename T>`

```
int Vector<T>::disordered() const { //统计向量中的逆序相邻元素对
```

```
    int n = 0; //计数器
```

```
    for ( int i = 1; i < _size; i++ ) //逐一检查各对相邻元素
```

```
        n += ( _elem[i - 1] > _elem[i] ); //逆序则计数
```

```
    return n; //向量有序当且仅当n = 0
```

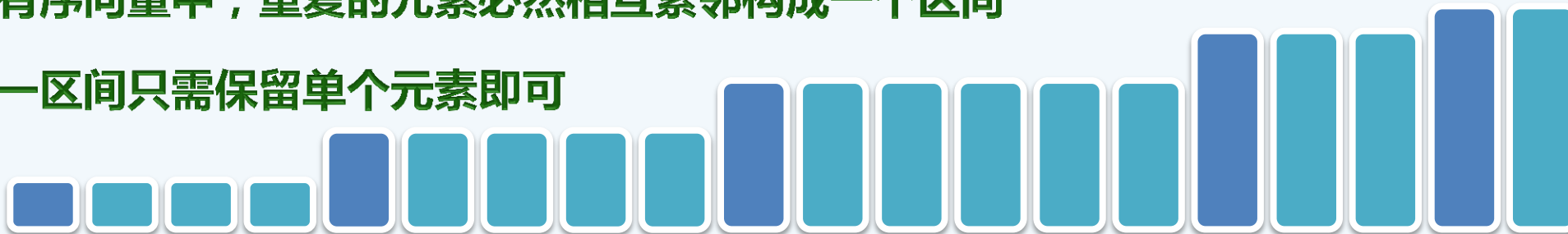
```
} //若只需判断是否有序，则首次遇到逆序对之后，即可立即终止
```

❖ 无序向量经预处理转换为有序向量之后，相关算法多可优化

低效算法

❖ 观察：在有序向量中，重复的元素必然相互紧邻构成一个区间

因此，每一区间只需保留单个元素即可



```
❖ template <typename T> int Vector<T>::uniquify() {  
    int oldSize = _size; int i = 1; //从首元素开始  
    while ( i < _size ) //从前向后，逐一比对各对相邻元素  
        //若雷同，则删除后者；否则，转至后一元素
```

```
    _elem[i - 1] == _elem[i] ? remove( i ) : i++;
```

```
    return oldSize - _size; //向量规模变化量，即删除元素总数
```

```
} //注意：其中_size的减小，由remove()隐式地完成
```



复杂度

❖ 运行时间主要取决于while循环次数

共计： $_size - 1 = n - 1$

❖ 最坏情况下，每次都需调用remove()

- 各耗时 $O(n-1) \sim O(1)$

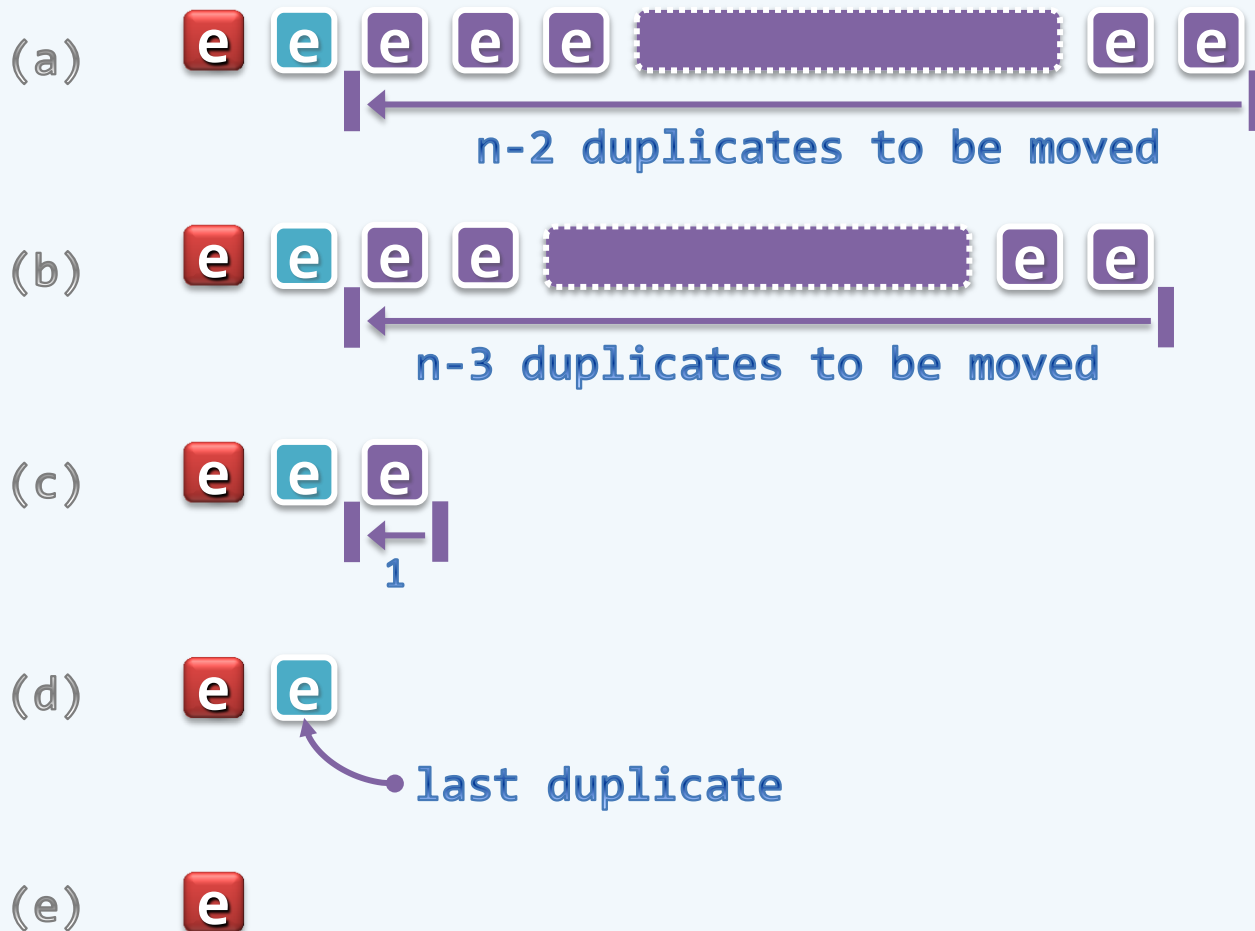
- 累计 $O(n^2)$

- 尽管省去find()，总体竟与

无序向量的deduplicate()相同

❖ 反思：低效的根源在于，同一元素可作为被删除元素的后继多次前移

启示：若能以重复区间为单位，**成批**删除雷同元素，性能必将改进...



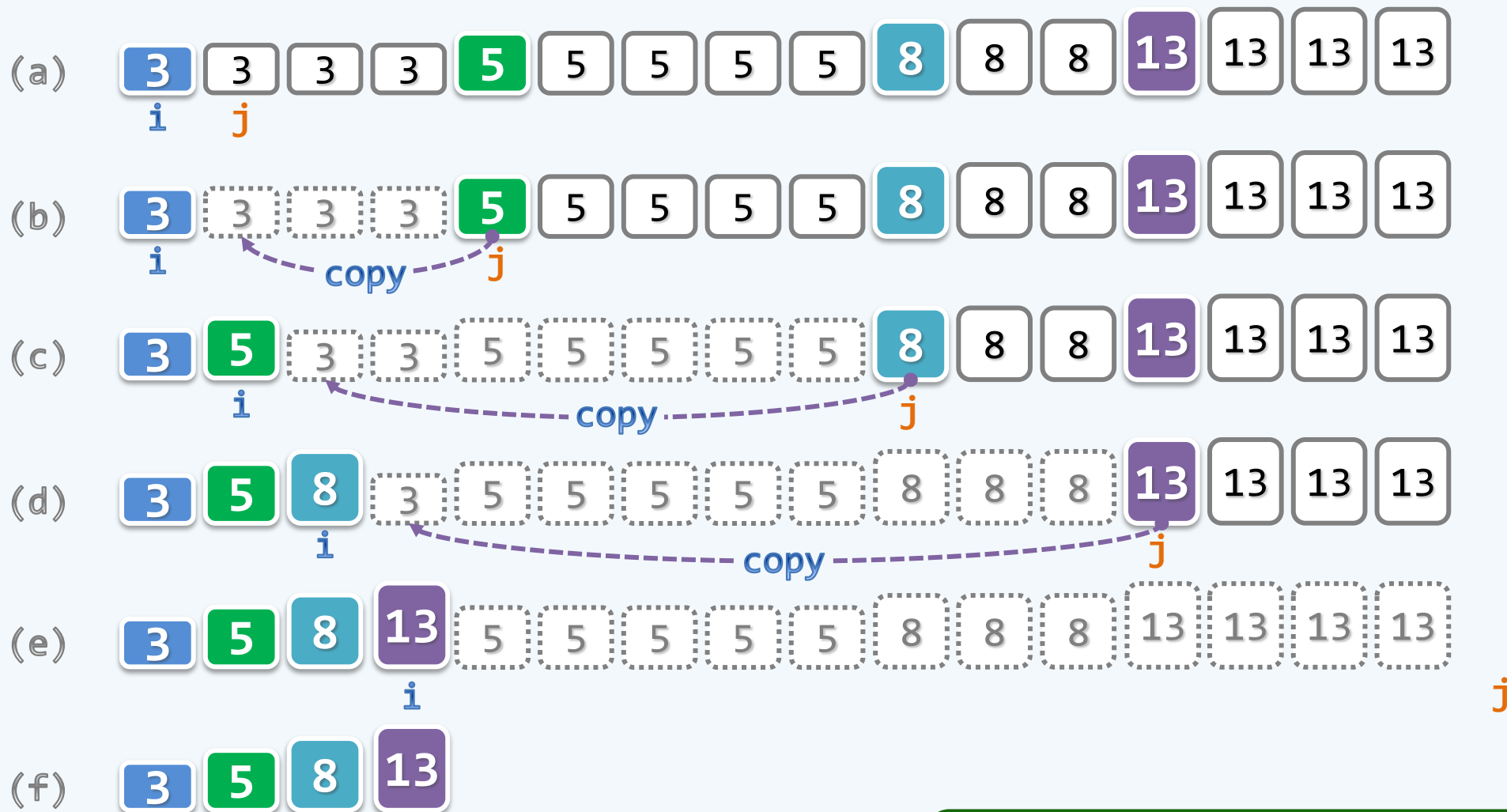
高效算法

```
❖ template <typename T> int Vector<T>::uniquify() {  
    Rank i = 0, j = 0; //各对互异 “相邻” 元素的秩  
    while ( ++ j < _size ) //逐一扫描，直至末元素  
        //跳过雷同者；发现不同元素时，向前移至紧邻于前者右侧  
        if ( _elem[ i ] != _elem[ j ] ) _elem[ ++ i ] = _elem[ j ];  
    _size = ++i; shrink(); //直接截除尾部多余元素  
    return j - i; //向量规模变化量，即被删除元素总数  
} //注意：通过remove(lo, hi)批量删除，依然不能达到高效率
```



实例与复杂度

❖ 共计 $n - 1$ 次迭代，每次常数时间，累计 $O(n)$ 时间



课后

❖ 较之无序向量，有序向量的唯一化可以更快地完成

其中的原因，如何理解和解释？