02-E1 起泡排序

#数据结构邓神

向量元素如果有序排序，计算效率将大大提升

例如：去重复，查找。。。

如何实现向量的有序化?

起泡排序：

排序器： 统一入口

```
template <typename T> void Vector::sort(Rank lo,Rank hi){
    switch(rand() % 5){ // 现在随机选择，但是其实也可以通过某种方法得出要使用的算法
        case 1: bubbleSort(lo,hi);break; // 起泡排序
        case 2: selectionSort(lo,hi);break; // 选择排序
        case 3: mergeSort(lo,hi);break; // 归并排序
        case 4: heapSort(lo,hi);break; // 堆排序
        case 5: quickSort(lo,hi);break; // 快速排序
    }
}
```

起泡排序

```
template <typename T> void Vector<T>::bubbleSort(Rank lo,Rank hi){
    while (!bubble(lo,ho--)); // 一次扫描，直到全部有序
}
```

这个算法的正确性在前面已经被验证了

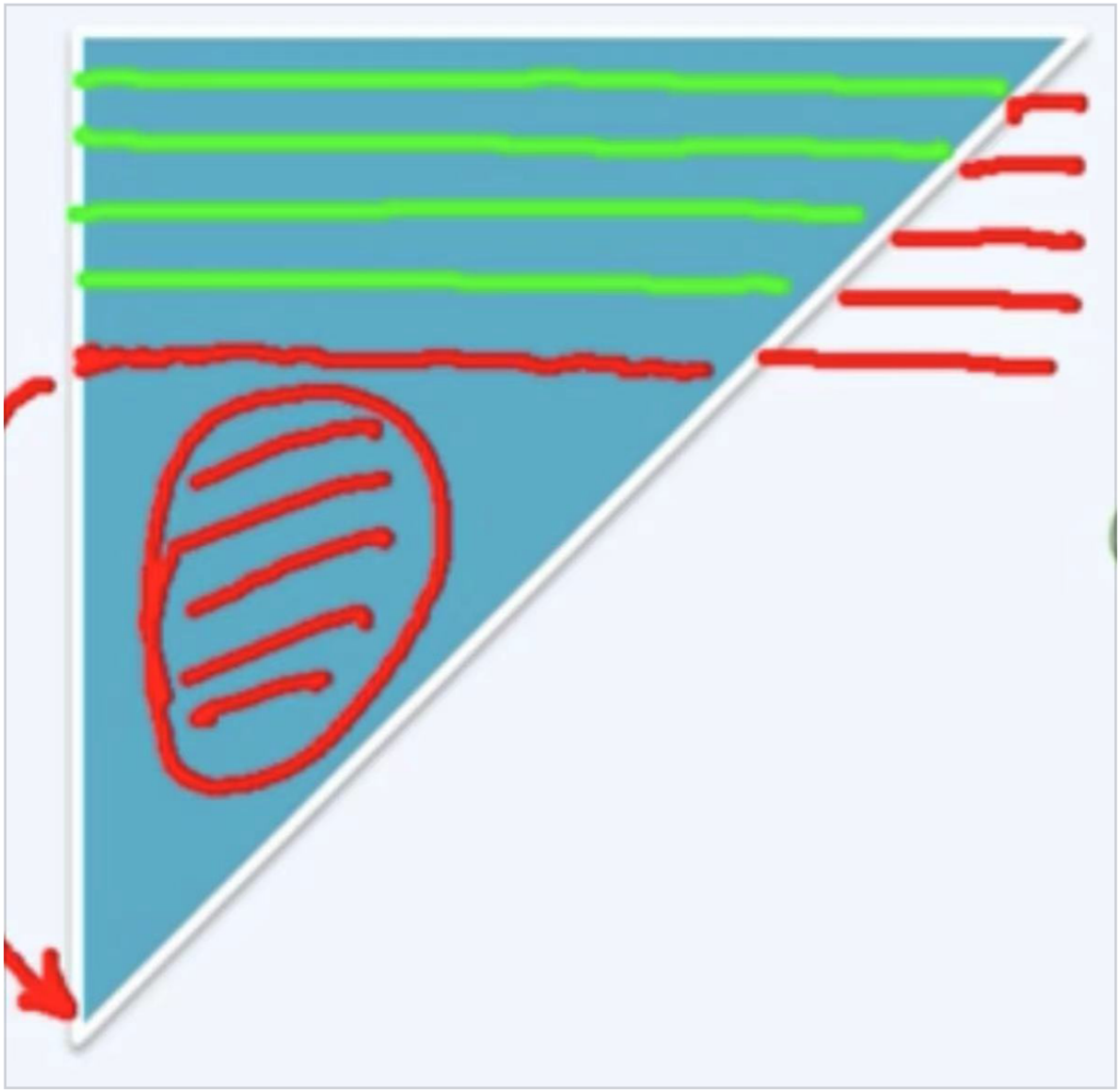
渐进时间复杂度下为 ： O(n^2)

改进方法： 每趟扫描交换都记录下是否存在逆序元素

如果存在 等价于 有过一次交换

```
template <typename T> bool Vector<T>::bubble(Rank lo,Rank hi){
```

```
    bool sorted = true; // 全局有序标识
    while (++lo < hi){
        if (_elem[lo - 1] > _elem[lo]){
            sorted = false; // 如果发生交换就是无序，取消全局标识
            swap(_elem[lo - 1],_elem[lo]);
        }
    }
    return sorted; // 返回标识 作为前面的父程序的while条件
}
```

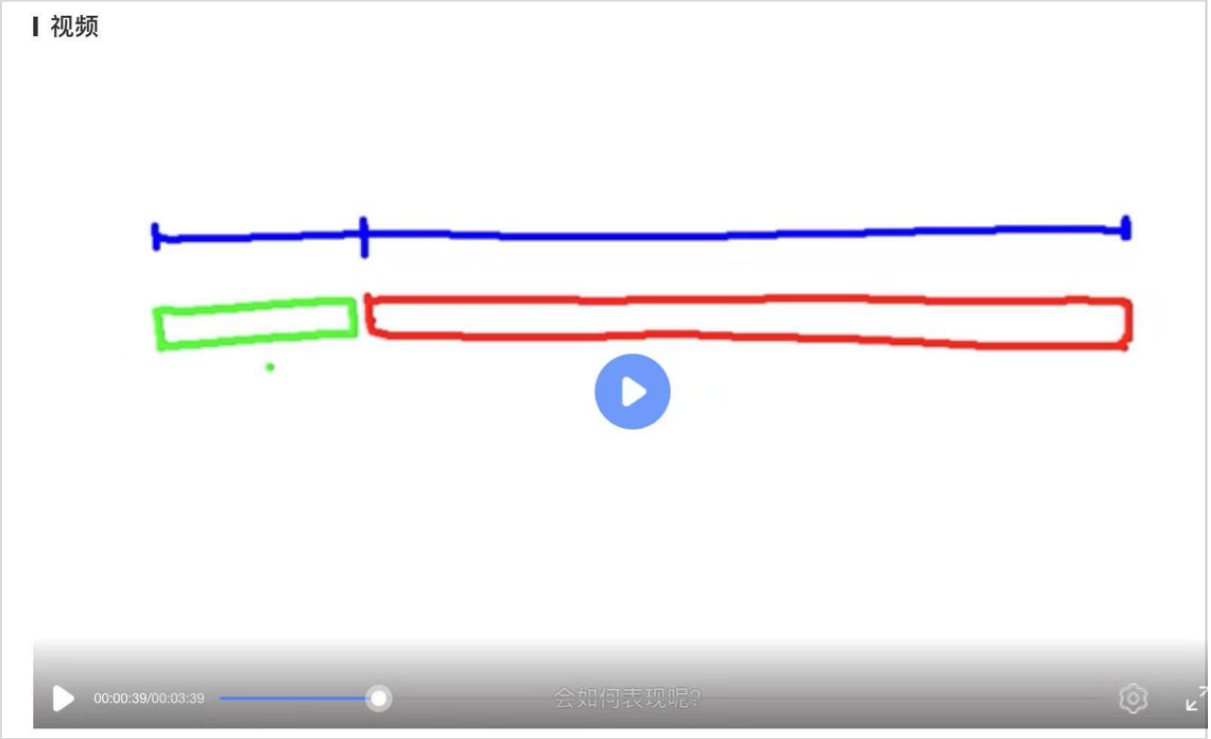


如果原来的版本的时间复杂度固定为一个大三角形那么现在的这个版本就可以在某一时刻绕过后面不必要的扫描

而且当且仅当整个向量反向有序的时候才需要所有的扫描，所以全部扫描的概率是比较低的

所以从时间复杂度上来说减少的时间还是比较客观的

再次改进



绿色表示无序

红色表示有序

总体而言需要绿色部分的长度

我们发现我们做了很多次无意义的扫描就是红色的部分

那么能否让我们的扫描只能省去那些不需要的元素呢?

```
template <typename T> Rank /* 这边与上边不同，这边函数类型为Rank 而不是 bool*/
Vector<T>::bubble(Rank lo,Rank hi){
    Rank last = lo;  // 初始值取为lo
    while (++lo < hi){
        if(_elem[lo-1] > _elem[lo]){
            last = lo;  // 更新最右侧交换逆序对位置，lo是持续递增的
            swap(_elem[lo-1],_elem[lo]);
        }
    }
    return last; // 返回最右侧逆序对的位置
}
```

我们在看那个实例，我们会发现，我们新的算法会直接跳过红色的一段，直接开始处理绿色的

也就说每次扫描的过程中，都会自动跳过末尾连续的一段。

综合评价

效率而言 最好 O(n) 最坏 O(n^2)

稳定性： 也就是重复元素在输出的相对次序，是否与输入的相对次序相对不变

如果不变 ➜ Stable 稳定的

如果不能保证 ➜ unstable 不稳定的

起泡算法不会交换两个相同的数字，所以次序会保持不变

如果把 if语句的 > 改为 >= 会有什么变化： stable ➜ unstable