12.排序

（a3）快速排序：重复元素

邓俊辉

deng@tsinghua.edu.cn

左见兮鸣鵙，右睹兮呼枭

重复元素

- 大量甚至全部元素重复时

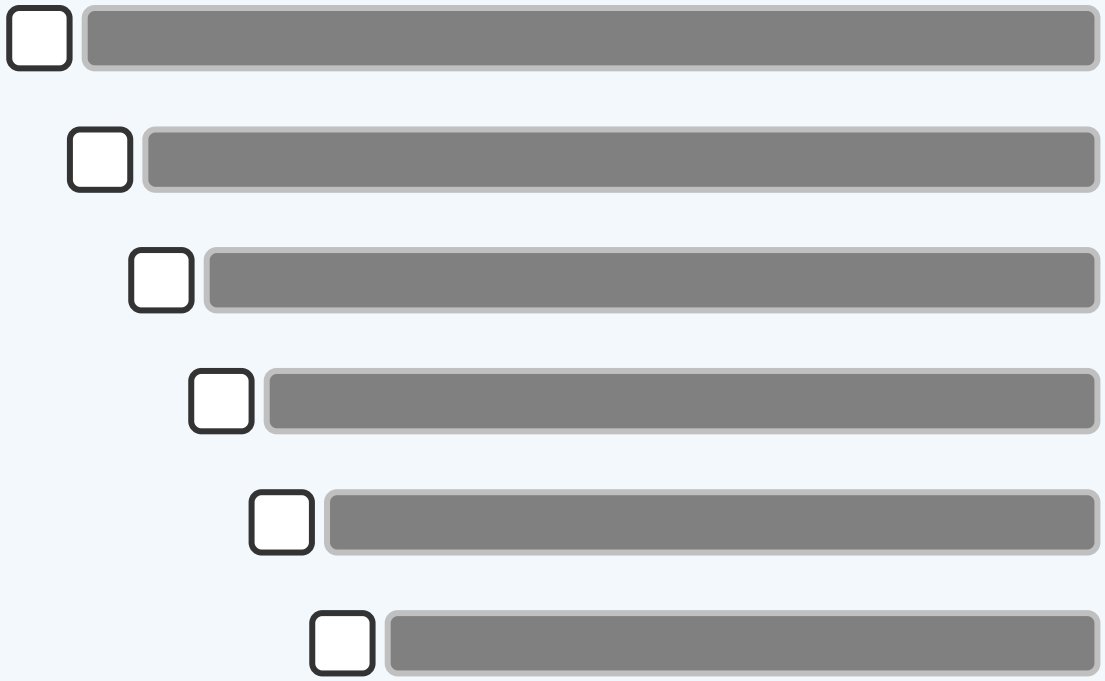
  轴点位置总是接近于lo

  子序列的划分极不均匀

  二分递归退化为线性递归

  递归深度接近于$O(n)$

  运行时间接近于$O(n^2)$



- 移动lo和hi的过程中，同时比较相邻元素：若属于相邻的重复元素，则不再深入递归

  但一般情况下，如此计算量反而增加，得不偿失

- 对算法A略做调整，即可解决问题——为便于对比，先给出等价形式A1...

算法A1

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi]
   swap( _elem[ lo ], _elem[ lo + rand() % ( hi - lo + 1 ) ] ); //随机交换
   T pivot = _elem[ lo ]; //经以上交换，等效于随机选取候选轴点
   while ( lo < hi ) { //从两端交替地向中间扫描，彼此靠拢
      while ( lo < hi && pivot <= _elem[ hi ] ) hi--; //向左拓展G
      if (lo < hi) _elem[ lo++ ] = _elem[ hi ]; //凡小于轴点者，皆归入L
      while ( lo < hi && _elem[ lo ] <= pivot ) lo++; //向右拓展L
      if (lo < hi) _elem[ hi-- ] = _elem[ lo ]; //凡大于轴点者，皆归入G
   } //assert: lo == hi
   _elem[ lo ] = pivot; return lo; //候选轴点归位；返回其秩
}
```

算法B1

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi]
   swap( _elem[ lo ], _elem[ lo + rand() % ( hi - lo + 1 ) ] ); //随机交换
   T pivot = _elem[ lo ]; //经以上交换，等效于随机选取候选轴点
   while ( lo < hi ) { //从两端交替地向中间扫描，彼此靠拢
      while ( lo < hi && pivot < _elem[ hi ] ) hi--; //向左拓展G
      if (lo < hi) _elem[ lo++ ] = _elem[ hi ]; //凡不大于轴点者，皆归入L
      while ( lo < hi && _elem[ lo ] < pivot ) lo++; //向右拓展L
      if (lo < hi) _elem[ hi-- ] = _elem[ lo ]; //凡不小于轴点者，皆归入G
   } //assert: lo == hi
   _elem[ lo ] = pivot; return lo; //候选轴点归位；返回其秩
}
```

算法B

```
template <typename T> Rank Vector<T>::partition( Rank lo, Rank hi ) { //[lo, hi]
   swap( _elem[ lo ], _elem[ lo + rand() % ( hi - lo + 1 ) ] ); //随机交换
   T pivot = _elem[ lo ]; //经以上交换，等效于随机选取候选轴点
   while ( lo < hi ) { //从两端交替地向中间扫描，彼此靠拢
      while ( lo < hi )
         if ( pivot <= _elem[ hi ] ) hi--; //向左拓展G，直至遇到不大于轴点者
         else { _elem[ lo++ ] = _elem[ hi ]; break; } //将其归入L
      while ( lo < hi )
         if ( _elem[ lo ] <= pivot ) lo++; //向右拓展L，直至遇到不小于轴点者
         else { _elem[ hi-- ] = _elem[ lo ]; break; } //将其归入G
   } //assert: lo == hi
   _elem[ lo ] = pivot; return lo; //候选轴点归位；返回其秩
}
```

性能

- 可以正确地处理一般情况，而且复杂度并未实质增高
- 处理重复元素时

  lo和hi会交替移动

  二者移动的距离大致相当

  最终，轴点被安置于 (lo + hi) / 2 处，实现划分均匀

- 相对于算法A的勤于拓展、懒于交换，转为懒于拓展、勤于交换

  因此：1）交换操作有所增多——尤其是雷同元素，在版本A中多不移动

  2）更不稳定