快排精讲

Quick Sort:

```
template <typename T> //向量快速排序
void Vector<T>::quickSort ( Rank lo, Rank hi ) { //0 <= lo < hi <= size
   /*DSA*/printf ( "\tQUICKsort [%3d, %3d)\n", lo, hi );
   if ( hi - lo < 2 ) return; //单元素区间自然有序，否则...
   Rank mi = partition ( lo, hi - 1 ); //在[lo, hi - 1]内构造轴点
   quickSort ( lo, mi ); //对前缀递归排序
   quickSort ( mi + 1, hi ); //对后缀递归排序
}
```

Partition_a:

```
template <typename T> //轴点构造算法：通过调整元素位置构造区间[lo, hi]的轴点，并返回其秩
Rank Vector<T>::partition ( Rank lo, Rank hi ) { //版本A：基本形式
   swap ( _elem[lo], _elem[lo + rand() % ( hi - lo + 1 ) ] ); //任选一个元素与首元素交换
   T pivot = _elem[lo]; //以首元素为候选轴点——经以上交换，等效于随机选取
   while ( lo < hi ) { //从向量的两端交替地向中间扫描
      while ( ( lo < hi ) && ( pivot <= _elem[hi] ) ) //在不小于pivot的前提下
         hi--; //向左拓展右端子向量
      _elem[lo] = _elem[hi]; //小于pivot者归入左侧子序列
      while ( ( lo < hi ) && ( _elem[lo] <= pivot ) ) //在不大于pivot的前提下
         lo++; //向右拓展左端子向量
      _elem[hi] = _elem[lo]; //大于pivot者归入右侧子序列
   } //assert: lo == hi
   _elem[lo] = pivot; //将备份的轴点记录置于前、后子向量之间
   return lo; //返回轴点的秩
}
```

Partition_a1:

```
template <typename T> //轴点构造算法：通过调整元素位置构造区间[lo, hi]的轴点，并返回其秩
Rank Vector<T>::partition ( Rank lo, Rank hi ) { //版本A1：与版本A等价，可直接转至与版本B等价的版本B1
   swap ( _elem[lo], _elem[lo + rand() % ( hi-lo+1 ) ] ); //任选一个元素与首元素交换
   T pivot = _elem[lo]; //以首元素为候选轴点——经以上交换，等效于随机选取
   while ( lo < hi ) { //从向量的两端交替地向中间扫描
      while ( ( lo < hi ) && ( pivot <= _elem[hi] ) ) //在不小于pivot的前提下
         hi--; //向左拓展右端子向量
      if ( lo < hi ) _elem[lo++] = _elem[hi]; //小于pivot者归入左侧子向量
      while ( ( lo < hi ) && ( _elem[lo] <= pivot ) ) //在保证不大于pivot的前提下
         lo++; //向右拓展左端子向量
      if ( lo < hi ) _elem[hi--] = _elem[lo]; //大于pivot者归入右侧子向量
   } //assert: lo == hi
   _elem[lo] = pivot; //将备份的轴点记录置于前、后子向量之间
   return lo; //返回轴点的秩
}
```

Partition_b:

```
template <typename T> //轴点构造算法：通过调整元素位置构造区间[lo, hi]的轴点，并返回其秩
Rank Vector<T>::partition ( Rank lo, Rank hi ) { //版本B：可优化处理多个关键码雷同的退化情况
   swap ( _elem[lo], _elem[lo + rand() % ( hi - lo + 1 ) ] ); //任选一个元素与首元素交换
   T pivot = _elem[lo]; //以首元素为候选轴点——经以上交换，等效于随机选取
   while ( lo < hi ) { //从向量的两端交替地向中间扫描
      while ( lo < hi )
         if ( pivot < _elem[hi] ) //在大于pivot的前提下
            hi--; //向左拓展右端子向量
         else //直至遇到不大于pivot者
            { _elem[lo++] = _elem[hi]; break; } //将其归入左端子向量
      while ( lo < hi )
         if ( _elem[lo] < pivot ) //在小于pivot的前提下
            lo++; //向右拓展左端子向量
         else //直至遇到不小于pivot者
            { _elem[hi--] = _elem[lo]; break; } //将其归入右端子向量
   } //assert: lo == hi
   _elem[lo] = pivot; //将备份的轴点记录置于前、后子向量之间
   return lo; //返回轴点的秩
}
```

Partition_b1:

```
template <typename T> //轴点构造算法：通过调整元素位置构造区间[lo, hi]的轴点，并返回其秩
Rank Vector<T>::partition ( Rank lo, Rank hi ) { //版本B1：版本B的等价形式，可直接转至与版本A等价的版本A1
   swap ( _elem[lo], _elem[lo + rand() % ( hi-lo+1 ) ] ); //任选一个元素与首元素交换
   T pivot = _elem[lo]; //以首元素为候选轴点——经以上交换，等效于随机选取
   while ( lo < hi ) { //从向量的两端交替地向中间扫描
      while ( ( lo < hi ) && ( pivot < _elem[hi] ) ) //在大于pivot的前提下
         hi--; //向左拓展右端子向量
      if ( lo < hi ) _elem[lo++] = _elem[hi]; //不大于pivot者归入左端子向量
      while ( ( lo < hi ) && ( _elem[lo] < pivot ) ) //在小于pivot的前提下
         lo++; //向右拓展左端子向量
      if ( lo < hi ) _elem[hi--] = _elem[lo]; //不小于pivot者归入右端子向量
   } //assert: lo == hi
   _elem[lo] = pivot; //将备份的轴点记录置于前、后子向量之间
   return lo; //返回轴点的秩
}
```

Partition_c:

```
template <typename T> //轴点构造算法：通过调整元素位置构造区间[lo, hi]的轴点，并返回其秩
Rank Vector<T>::partition ( Rank lo, Rank hi ) { //版本C
   swap ( _elem[lo], _elem[lo + rand() % ( hi - lo + 1 ) ] ); //任选一个元素与首元素交换
   T pivot = _elem[lo]; //以首元素为候选轴点——经以上交换，等效于随机选取
   int mi = lo;
   //-------------------------------------------------------------------------
   //   [ ---- < [lo] ----- ] [ ----- [lo] <= --- ] [ ----- unknown ----- ]
   // X x . . . . . . . . . x . . . . . . . . . x . . . . . . . . . x
   // |                     |                     |                   |
   // lo (pivot)            mi                    k                  hi
   //-------------------------------------------------------------------------
   for ( int k = lo + 1; k <= hi; k++ ) //自左向右扫描
      if ( _elem[k] < pivot ) //若当前元素_elem[k]小于pivot，则
         swap ( _elem[++mi], _elem[k] ); //将_elem[k]交换至原mi之后，使L子序列向右扩展
   //-------------------------------------------------------------------------
   //   [ --------- < [lo] ---------- ] [ ----------- [lo] <= ----------- ]
   // X x . . . . . . . . . . . . . . x . . . . . . . . . . . . . . . . x
   // |                               |                                 |
   // lo                              mi                                hi
   //-------------------------------------------------------------------------
   swap ( _elem[lo], _elem[mi] ); //候选轴点归位
   return mi; //返回轴点的秩
}
```