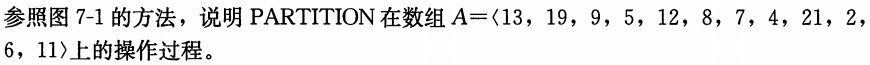
## 第7章 快速排序

#### 练习 7.1-1



**问题分析**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *p*,*j* |  |  |  |  |  |  |  |  |  |  | *r* |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *p* | *j* |  |  |  |  |  |  |  |  |  | *r* |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *p* |  | *j* |  |  |  |  |  |  |  |  | *r* |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p*,*i* |  |  | *j* |  |  |  |  |  |  |  | *r* |
|  | 9 | 19 | 13 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* | *i* |  |  | *j* |  |  |  |  |  |  | *r* |
|  | 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* | *i* |  |  |  | *j* |  |  |  |  |  | *r* |
|  | 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  | *i* |  |  |  | *j* |  |  |  |  | *r* |
|  | 9 | 5 | 8 | 19 | 12 | 13 | 7 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  | *i* |  |  |  | *j* |  |  |  | *r* |
|  | 9 | 5 | 8 | 7 | 12 | 13 | 19 | 4 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  |  | *i* |  |  |  | *j* |  |  | *r* |
|  | 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  |  | *i* |  |  |  |  | *j* |  | *r* |
|  | 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  |  |  | *i* |  |  |  |  | *j* | *r* |
|  | 9 | 5 | 8 | 7 | 4 | 2 | 19 | 12 | 21 | 13 | 6 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  |  |  |  | *i* |  |  |  |  | *r*,*j* |
|  | 9 | 5 | 8 | 7 | 4 | 2 | 6 | 12 | 21 | 13 | 19 | 11 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  |  |  |  | *i* | *q* |  |  |  | *r*,*j* |
|  | 9 | 5 | 8 | 7 | 4 | 2 | 6 | 11 | 21 | 13 | 19 | 12 |

#### 练习 7.1-2



**问题分析**

当数组的元素都相同时，PARTITION返回*q* = *r*。

要使得当数组中所有元素都相同时，q = ，最简单的办法是在代码中检查这种情况，针对这种特殊情况再做处理。检测所有元素是否相同所花的时间为Θ(*n*)，这不会增加PARTITION的渐近时间。

#### 练习 7.1-3



**问题分析**

PARTITON过程需要进行*r*−*p*次迭代，每次迭代花费常数时间，故PARTITION的时间复杂度为Θ(*r*−*p*) = Θ(*n*)。

#### 练习 7.1-4



**问题分析**

PARTITION\_NON\_ASCENDING(*A*, *p*, *r*)

*i* = *p* – 1

**for** *j* = *p* 🡪 *r*-1

**if** *A*[*j*] >= *A*[*r*]

*i*++

exchange *A*[*i*] with *A*[*j*]

exchange *A*[*i*+1] with *A*[*r*]

**return** *i*+1

QUICKSORT\_NON\_ASCENDING(*A*, *p*, *r*)

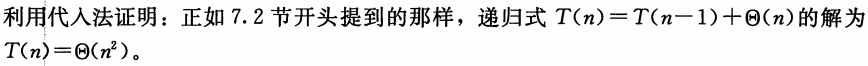
**if** *p* < *r*

*q* = PARTITION\_NON\_ASCENDING(*A*, *p*, *r*)

QUICKSORT\_NON\_ASCENDING(*A*, *p*, *q*-1)

QUICKSORT\_NON\_ASCENDING(*A*, *q*+1, *r*)

#### 练习 7.2-1



**问题分析**

略。

#### 练习 7.2-2



**问题分析**

利用7.1-2的结论，当所有元素都相同时，每次PARTITION都得到一个最不均衡的划分，因此QUICKSORT的时间得杂度为Θ(*n*2)。

#### 练习 7.2-3



**问题分析**

考虑包含*n*个元素的数组*A*，其中元素各不相同，并且按降序排列，即*A*[1] > *A*[2] > … > *A*[*n*]。

第1次划分：左半边为空，右半边包含*n*−1个元素

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *A*[*n*] | *A*[2] | *A*[3] | … … | *A*[1] |

第2次划分：左半边包含*n*−2个元素，右半边为空

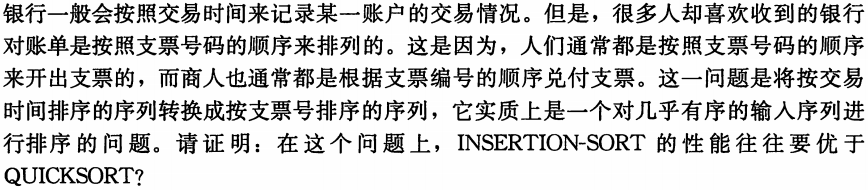
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *A*[*n*] | *A*[2] | *A*[3] | … … | *A*[*n*-1] | *A*[1] |

2次划分后，剩下的子数组*A*[2 .. *n*−1]还是一个降序排序的数组，对这个子数组的划分情况与第1次和第2次相同。因此，每次划分都是一个最不均衡的划分，所以时间复杂度为Θ(*n*2)。

**小结**：对于基本有序的数组，无论升序还是降序，快速排序的效率都很低。

而对于基本按升序排列的数组，插入排序的效率很高；但对于基本按降序排序的数组，插入排序的效率同样很低。

#### 练习 7.2-4



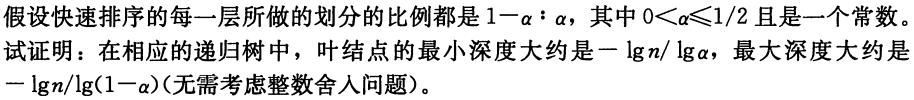
**问题分析**

按照交易时间从早到晚排序的支票序列，它们的支票号基本上也是排好序的。

根据思考题2-4c的结论，插入排序的时间也序列中逆序对的数目成正比。对一个基本排好序的序列，其中的逆序对数目很少，因此插入排序很快，远远小于Θ(*n*2)。

而对一个基本排好序的序列，快速排序几乎每次都做一个不平衡的划分，因此其运行时间接近于Θ(*n*2)。

#### 练习 7.2-5

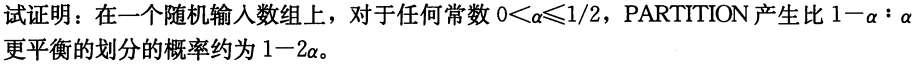


**问题分析**

最小深度的叶结点出现在每次划分比例为*α*的分枝，即子问题规模较小的那个分枝，它的深度为log*αn* = lg*n*/lg*α*。

最大深度的叶结点出现在每次划分比例为1−*α*的分枝，即子问题规模较大的那个分枝，它的深度为log(1−*α*)*n* = lg*n*/lg(1−*α*)。

#### 练习 7.2-6\*



**问题分析**

对数组的划分取决于数组中的最后一个元素。假如数组大小为*n*。如果*A*[*n*]是最大的*αn*个元素中的一个，那么对数组的划分一定不会比1−*α* : *α*更好。如果*A*[*n*]是最小的*αn*个元素中的一个，那么对数组的划分也一定不会比1−*α* : *α*更好。对于剩下的*n*−2*αn*个元素，如果*A*[*n*]是其中一个，那么产生的划分才会比1−*α* : *α*更好。

如果数组的排列是随机的，那意味着每一个元素都有相同的概率出现在数组的最后一个位置上。根据前面的分析，产生比1−*α* : *α*更好划分的概率，也等于数组的最后一个元素为上文提到的*n*−2*αn*个元素之一的概率，这个概率为(*n*−2*αn*)/*n* = 1−2*α*。

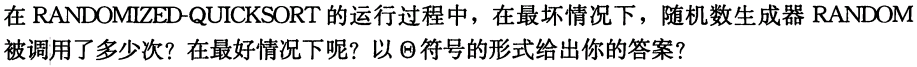
#### 练习 7.3-1



**问题分析**

随机化算法每次运行时间都不一样，即使对同一个输入。其最坏情况也不针对某个特定的输入，而是随机产生的。因此，分析最坏运行时间意义不大。

#### 练习 7.3-2



**问题分析**

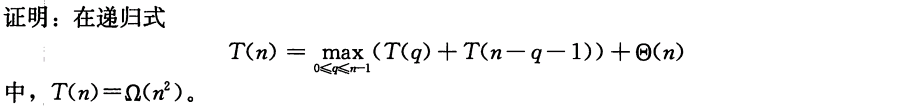
(1) 最坏情况

最坏情况每次都划分出(*n*−1)个元素的子数组，一共进行*n*−1次划分，每次划分调用一次RANDOMIZED-QUICKSORT，故RANDOM调用*n*−1次，即Θ(*n*)。

(2) 最好情况

最好情况每次划分都是平衡划分，即划分为个元素的子数组和个元素的子数组。假设*CR*(*n*)是对*n*个元素的数组进行RANDOMIZED\_QUICKSORT调用RANDOM的次数，可以写出它的递归式*CR*(*n*) = 2*CR*(*n*/2) + 1。求解得到*CR*(*n*) = Θ(*n*)。

#### 练习 7.4-1



**问题分析**

假设*T*(*m*) ≥ *cm*2对所有*m* ≤ *n*−1都成立。现在考*m* = *n*的情形。



在*q* = 0或*q* = *n*−1时取得最大值，=。所以



可以找到一个足够小的*c*，使得(2*cn*−*c*)能小于Θ(*n*)项，此时*T*(*n*) ≥ *cn*2。所以*T*(*n*) = Ω(*n*2)。

#### 练习 7.4-2



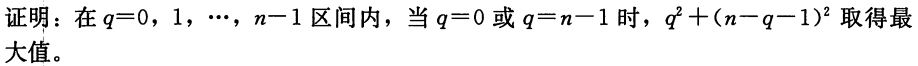
**问题分析**

假设*T*(*n*)是最好情况下快速排序的运行时间，它的递归式为



求解可得到*T*(*n*) = Ω(*n*lg*n*)。

#### 练习 7.4-3



**问题分析**

令*f*(*q*) = *q*2 + (*n*−*q*−1)2，对它求导得到*f’*(*q*) = 4*q* – 2(*n*–1)。*f’*(*q*)在*q* < (*n*–1)/2时小于0，在*q* = (*n*–1)/2时等于0，在*q* > (*n*–1)/2时大于0。这说明当0 ≤ *q* < (*n*–1)/2时，*f*(*q*)是递减的；当(*n*–1)/2 < *q* ≤ *n*–1时，*f*(*q*)是递增的；当*q* = (*n*–1)/2时，*f*(*q*)取最小值。因此，*f*(*q*)的最大值只有可能在*q*的区间的端点上，即*q* = 0或*q* = *n*–1。而无论*q* = 0还是*q* = *n*–1，*f*(*q*)都等于(*n*–1)2。

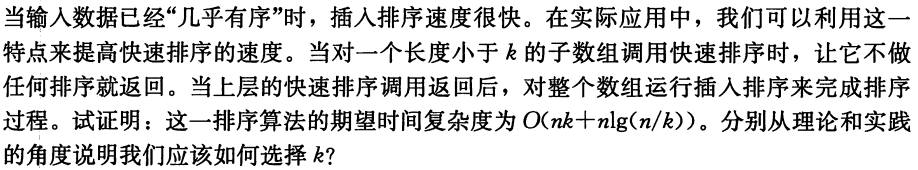
#### 练习 7.4-4



**问题分析**

与QUICKSORT期望运行时间的分析过程一样。

#### 练习 7.4-5



**问题分析**

该算法分为快速排序阶段和插入排序阶段。在快速排序阶段，子数组被分解到规模为*k*就停止了。平均来看，分解的层数为Θ(lg(*n*/*k*))，每一层分解的代价都为Θ(*n*)。因此，快速排序阶段的期望时间复杂度为Θ(*n*lg(*n*/*k*))。

在插入排序阶段，虽然对整个数组进行插入排序，然而实际上相当于对*n*/*k*个规模为*k*的子数组进行插入排序。每个子数组的插入排序的时间复杂度为Θ(*k*2)。所以，插入排序阶段的期望时间复杂度为Θ((*n*/*k*)∙ *k*2) =Θ(*nk*)。

所以，该算法的时间复杂度为两个阶段之和，即Θ(*nk* + *n*lg(*n*/*k*))。

从理论上分析，选择*k*应当使该算法的时间不长于快速排序的时间，可以通过计算两种算法的时间交叉点来选择*k*。

Θ(*nk* + *n*lg(*n*/*k*)) = Θ(*n*lg*n*)

Θ(*nk* + *n*lg*n* - *n*lg*k*) = Θ(*n*lg*n*)

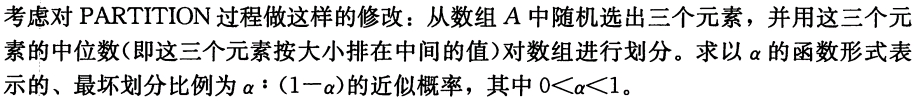
Θ(*nk*) = Θ(*n*lg*k*)

*k* = Θ(lg*k*) = O(lg*n*)

因此，理论上*k*应当比Θ(lg*n*)要小。

在实践中，可以通过实验来选择*k*。

#### 练习 7.4-6\*



**问题分析**

为了简化分析，假设0 < *α* ≤ 1/2。

如果数组已经排好序，可以按照大小将数组分为3个部分，如下图所示。

|  |  |  |
| --- | --- | --- |
| ①最小*αn*个元素 | ②中间(1−2*α*)*n*个元素 | ③最大*αn*个元素 |

如果PARTITION过程的分隔元素位于第②部分，那么产生的划分比*α* : 1−*α*更好。而如果PARTITION过程的分隔元素位于第①部分和第③部分，那么产生的划分比*α* : 1−*α*更坏。因此，我们需要求在数组中任取3个元素，中位数位于第②部分的概率。

而中位数位于第②部分可分为3种情况：

1) 从第②部分中任取3个元素，这种情况可能的种数有



2) 从第②部分中任取2个元素，从第①部分和第③部分中任取1个元素，这种情况可能的种数有



3) 从第①部分、第②部分和第③部分中各任取1个元素



而从数组中任取3种元素的总的可能的种数为



因此，中位数位于第②部分的概率为



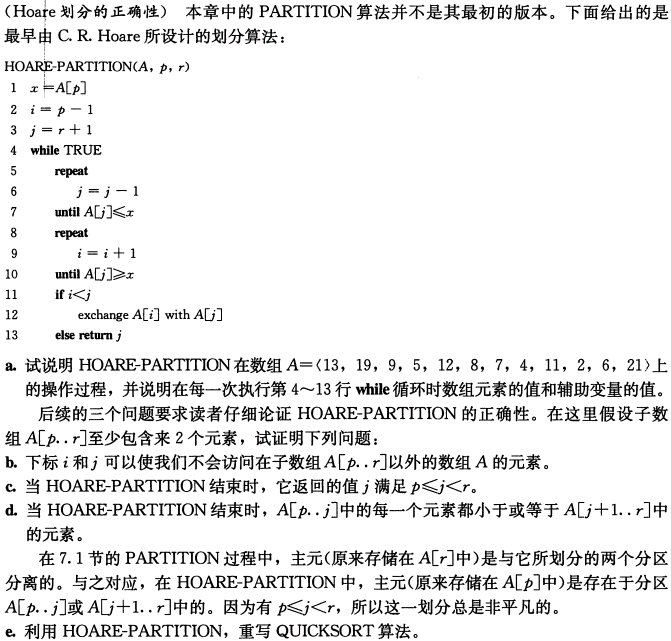
这个概率与数组的规模*n*有关，为了简化分析，可以假设*n*很大，计算*n*🡪∞的概率。



所以，中位数位于第②部分的近似概率为(1–4*α*2+6*α*3)，这也是修改后的PARTITION过程产生的数组划分好于*α* : 1−*α*的概率。

1/2 < *α* < 1的情况是类似的，得到结果为(6*α*2–4*α*3–1)。

#### 思考题 7-1



**a-问题分析**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *p* |  |  |  |  |  |  |  |  |  |  | *r* | *j* |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 2 | 6 | 21 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p*,*i* |  |  |  |  |  |  |  |  |  | *j* | *r* |  |
|  | 6 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 2 | 13 | 21 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* | *i* |  |  |  |  |  |  |  | *j* |  | *r* |  |
|  | 6 | 2 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 19 | 13 | 21 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *p* |  |  |  |  |  |  |  | *j* | *i* |  | *r* |  |
|  | 6 | 2 | 9 | 5 | 12 | 8 | 7 | 4 | 11 | 19 | 13 | 21 |  |

**b-问题分析**

虽然*i*和*j*分别被初始化为*p*−1和*r*+1，但是在第一次**while**迭代中，*i*会被先加1，而*j*会被先减1，这样*i*和*j*分别从*p*和*r*开始迭代，所以在迭代开始，*i*和*j*不会移动到数组之外。

按照题目要求，数组至少包含2个元素。由于以首个元素作为分隔元素，所以在第一轮迭代中，*i*会停在*p*，而*j*会一直向左移动，直到遇到≤ *x*的元素。下面分两种情况说明：

1) 如果*j*在向左移动的过程中一直没有遇到≤ *x*的元素，那么*j*会一直向左移动直到*j* = *p*为止，因为*A*[*p*] = *x*。此时，由于*i*一直停留在*p*处，所以*i* = *j*，故**while**迭代退出。在这个过程中，*i*和*j*没有超出数组的范围。

2) 如果*j*在遇到了一个≤ *x*的元素 (该元素不为*A*[*p*])，那么*A*[*j*]与*A*[*i*]即*A*[*p*]交换。这说明在数组的左侧至少有一个≤ *x*的元素，而在数组的右侧至少有一个≥ *x*的元素。因此，*i*在向右移动的过程中遇到右侧的≥ *x*的元素会停下来，而*j*在向左移动的过程中遇到左侧的≤ *x*的元素也会停下来。由于*i*和*j*相向移动，总有一个时刻，在*i*和*j*都停下来的时候，会有*i* ≥ *j*，此时**while**循环退出。在这个过程中，*i*和*j*也不会超出数组的范围。

**c-问题分析**

根据问题b的分析，在HOARE\_PARTITION结束时，*i*有可能停留在*p*，而*j*一定会至少向左移动一个位置，故*j* < *r*。又因为*j*不会超出数组的范围，故*j* ≥ *p*。所以*p* ≤ *j* < *r*。

**d-问题分析**

可以证明循环不变式：在每轮**while**迭代开始之前，*A*[*p* .. *i*] ≤ *x*，并且*A*[*j* .. *r*] ≥ *x*。具体证明过程省略。

初始化：初始时，*i* = *p*−1并且*j* = *r*+1，*A*[*p* .. *i*]和*A*[*j* .. *r*]为空，循环不变式显然成立。

保持：假设某次迭代之前，循环不变式成立，此时*i*和*j*的值假设分别为*icur*和*jcur*。根据循环不变式，有*A*[*p* .. *icur*] ≤ *x*并且*A*[*jcur* .. *r*] ≥ *x*。进入本次迭代过程，首先*j*会一直向左移动，直到遇到一个元素*A*[*j*] ≤ *x*才停止，假设此时*j*的值为*jnext*，很显然有*A*[*jnext*+1 .. *r*] ≥ *x*。然后，*i*会一直向右移动，直到遇到一个元素*A*[*i*] ≥ *x*才停止，假设此时*i*的值为*inext*，很显然有*A*[*p* .. *inext*−1] ≤ *x*。如果*inext* < *jnext*，则交换*A*[*inext*]和*A*[*jnext*]；否则循环终止。交换后，有*A*[*inext*] ≤ *x*并且*A*[*jnext*] ≥ *x*。于是在进入下一次迭代之前，有*A*[*p* .. *inext*] ≤ *x*并且*A*[*jnext* .. *r*] ≥ *x*。因此，循环不变式在进入下一次迭代之前仍然成立。

终止：假设在进入某次迭代后，循环终止。假设在进入该次迭代之前，*i*和*j*的值分别为*icur*和*jcur*。根据循环不变式，有有*A*[*p* .. *icur*] ≤ *x*并且*A*[*jcur* .. *r*] ≥ *x*。假设执行本次迭代之后，*i*和*j*的值分别停在*inext*和*jnext*，很显然有*A*[*p* .. *inext*−1] ≤ *x*和*A*[*jnext*+1 .. *r*] ≥ *x*。导致循环终止有两种情形：一种是*inext* = *jnext*，一种是*inext* > *jnext*。下面分别对这两种情况进行分析。

1) *inext* = *jnext*。根据算法，*i*停在*inext*处必有*A*[*inext*] ≥ *x*，*j*停在*jnext*处必有*A*[*jnext*] ≤ *x*。由于*inext* = *jnext*，所以必然有*A*[*inext*] = *A*[*jnext*] = *x*。由于*A*[*p* .. *inext*−1] ≤ *x*，所以*A*[*p* .. *inext*] ≤ *x*，也即*A*[*p* .. *jnext*] ≤ *x*。由于*A*[*jnext*+1 .. *r*] ≥ *x*，所以*A*[*p* .. *jnext*] ≤ *A*[*jnext*+1 .. *r*]，即*A*[*p* .. *jnext*]中的每一个元素都小于或等于*A*[*jnext*+1 .. *r*]中的元素。

≤ *x*

≥ *x*

= *x*

*Ap*

*Ap*+1

… …

*Aj*−1

*Ar*−1

*Ar*

… …

*Aj*+1

*Aj*

*r*

*p*

*i*,*j*

≤ *x*

≥ *x*

*Ap*

*Ap*+1

… …

*Aj*−1

*Ar*−1

*Ar*

… …

*Aj*+1

*Aj*

*r*

*p*

*i*,*j*

2) *inext* > *jnext*。根据算法，*i*停在*inext*处必有*A*[*inext*] ≥ *x*，由于*A*[*jnext*+1 .. *r*] ≥ *x*并且*inext* > *jnext*，所以*i*在向右移动到*A*[*jnext*+1 .. *r*]的左边第一个位置就会发现*A*[*i*] ≥ *x*，*i*就停在这个位置，即*inext* = *jnext*+1。同样，也有*jnext* = *inext*−1。由于*A*[*p* .. *inext*−1] ≤ *x*，所以*A*[*p* .. *jnext*] ≤ *x*。由于*A*[*jnext*+1 .. *r*] ≥ *x*，所以*A*[*p* .. *jnext*] ≤ *A*[*jnext*+1 .. *r*]，即*A*[*p* .. *jnext*]中的每一个元素都小于或等于*A*[*jnext*+1 .. *r*]中的元素。

≤ *x*

≥ *x*

*Ap*

*Ap*+1

… …

*Ar*−1

*Ar*

… …

*Aj*+1

*Aj*

*r*

*p*

*j*

*i*

**e-问题分析**

HOARE\_QUICKSORT(*A*, *p*, *r*)

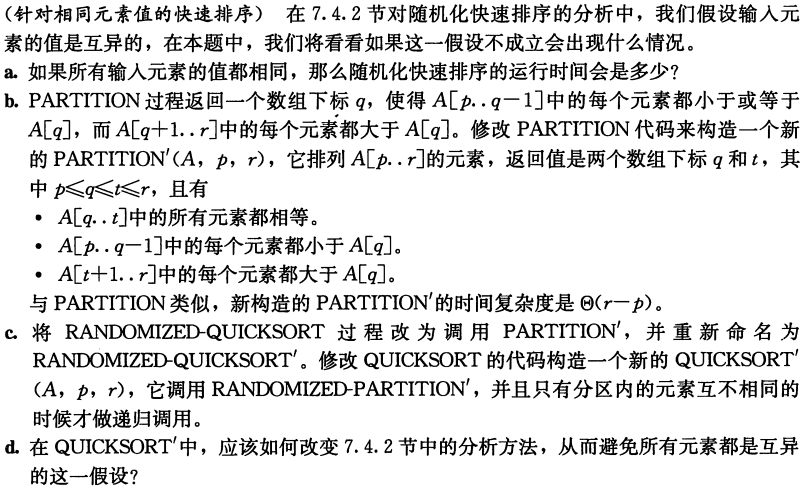
**if** *p* < *r*

q = HOARE\_PARTITION(*A*, *p*, *r*)

HOARE\_QUICKSORT(*A*, *p*, *q*)

HOARE\_QUICKSORT(*A*, *q*+1, *r*)

#### 思考题 7-2



**a-问题分析**

如果数组的所有元素都相同，那么随机化快速排序每次递归都产生一个最不均衡的划分，因此运行时间为Θ(*n*2)。

**b-问题分析**

先给出PARTITION’的伪代码。

PARTITION’(*A*, *p*, *r*)

*x* = *A*[*r*]

*i* = *j* = *p*-1

**for** *k* = *p* 🡪 *r*-1

**if** *A*[*k*] == *x*

*j*++

exchange *A*[*k*] with *A*[*j*]

**elseif** *A*[*k*] < *x*

*i*++

*j*++

exchange *A*[*k*] with *A*[*i*]

exchange *A*[*k*] with *A*[*j*]

exchange *A*[*j*+1] with *A*[*r*]

**return** (*i*+1) and (*j*+1)

用循环不变式来证明PARTITION*’*的正确性。下面给出循环不变式。在每一轮迭代开始之前，对任意数组下标*k*，有：① *A*[*p* .. *i*] < *x*；② *A*[*i*+1 .. *j*] = *x*；③ *A*[*j*+1 .. *k*-1] > *x*。下面证明这个循环不变式。

初始化：在第一轮迭代之前，*i* = *j* = *p*-1并且*k* = *p*。显然，*A*[*p* .. *i*]、*A*[*i*+1 .. *j*]和*A*[*j*+1 .. *k*-1]中都不存在元素，所以循环不变式的条件①②③都满足。条件④显然也满足。故第一轮迭代之前，循环不变式成立。

保持：根据*A*[*k*]的值分为3种情况。

1) *A*[*k*] > *x*：此时唯一的操作是将*k*加1。相当于将*A*[*k*]加入到*A*[*j*+1 .. *k*-1]中，条件③仍然成立。而其他条件没变，所以循环不变式在下一次迭代之前成立。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  | *j* |  |  |  |  | *k* |  |  | *r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj* | *Aj*+1 | … … | | *Ak*−1 | *Ak* | … … | | *Ar* |
| **< *x*** | | | | **= *x*** | | | | **> *x*** | | | | **> *x*** |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  | *j* |  |  |  |  |  | *k* |  | *r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj* | *Aj*+1 | … … | | *Ak*−2 | *Ak*−1 | *Ak* | … | *Ar* |
| **< *x*** | | | | **= *x*** | | | | **> *x*** | | | | |  |  |  |

2) *A*[*k*] = *x*：此时将*j*加1，再交换*A*[*k*]和*A*[*j*]，再将*k*加1。在交换*A*[*k*]和*A*[*j*]后，*A*[*j*] = *x*，条件②仍然成立，并且*A*[*k*] > *x*；将*k*加1后，*A*[*k*-1] > *x*，条件③仍然成立；并且条件①没变。所以循环不变式在下一次迭代之前成立。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  | *j* |  |  |  |  | *k* |  |  | *r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj* | *Aj*+1 | … … | | *Ak*−1 | *Ak* | … … | | *Ar* |
| **< *x*** | | | | **= *x*** | | | | **> *x*** | | | | **= *x*** |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  |  | *j* |  |  |  |  | *k* |  | *r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj*−1 | *Aj* | *Aj*+1 | … | *Ak*−2 | *Ak*−1 | *Ak* | … | *Ar* |
| **< *x*** | | | | **= *x*** | | | | | **> *x*** | | | |  |  |  |

3) *A*[*k*] < *x*：此时循环体将*i*和*j*分别加1，然后交换*A*[*k*]和*A*[*i*]，再交换*A*[*k*]和*A*[*j*]，最后将*k*加1。在交换*A*[*k*]和*A*[*i*]后，*A*[*i*] < *x*，条件①仍然成立，并且有*A*[*k*] = *x*；再交换*A*[*k*]和*A*[*j*]后，*A*[*j*] = *x*，条件②成立，并且*A*[*k*] > *x*；后将*k*加1，*A*[*k*-1] > *x*，条件③仍然成立。所以循环不变式在下一次迭代之前成立。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  | *j* |  |  |  |  | *k* |  |  | *r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj* | *Aj*+1 | … … | | *Ak*−1 | *Ak* | … … | | *Ar* |
| **< *x*** | | | | **= *x*** | | | | **> *x*** | | | | **< *x*** |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  |  | *i* |  |  | *j* |  |  |  |  | *k* |  |  | *r* |
| *Ap* | … … | | *Ai*−1 | *Ai* | *Ai*+1 | … | *Aj* | *Aj*+1 | … … | | *Ak*−1 | *Ak* | … … | | *Ar* |
| **< *x*** | | | | | **= *x*** | | | **> *x*** | | | | **= *x*** |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  |  | *i* |  |  |  | *j* |  |  |  |  | *k* |  | *r* |
| *Ap* | … … | | *Ai*−1 | *Ai* | *Ai*+1 | … | *Aj*−1 | *Aj* | *Aj*+1 | … | *Ak*−2 | *Ak*−1 | *Ak* | … | *Ar* |
| **< *x*** | | | | | **= *x*** | | | | **> *x*** | | | |  |  |  |

终止：当迭代终止时，*k* = *r*。此时数组中的元素被划分为了4个集合，如下图所示。根据循环不变式，有*A*[*p* .. *i*] < *x*，*A*[*i*+1 .. *j*] = *x*并且*A*[*j*+1 .. *r*-1] > *x*。由于分隔元素*A*[*r*]一直没有动，故*A*[*r*] = *x*。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  | *j* |  |  |  |  | *k,r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj* | *Aj*+1 | … … | | *Ar*−1 | *Ar* |
| **< *x*** | | | | **= *x*** | | | | **> *x*** | | | | **= *x*** |

在PARTITION*’*的倒数第二行，将*A*[*r*]与*A*[*j*+1]交换。*A*[*j*+1]是> *x*的集合中最左的元素，而*A*[*j*+1]的左边正是= *x*的集合中最右的元素，这一步将*A*[*r*]移动到正确的位置上，使*A*[*r*]加入到= *x*的集合中。经过这一步操作后，数组中的元素被划分为了3部分：1) 左边部分*A*[*p* .. *i*] < *x*；2) 中间部分*A*[*i*+1 .. *j*+1] = *x*；3) 右边部分*A*[*j*+2 .. *r*] > *x*。PARTITION*’*的最后一行返回正确的*q*和*t*。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *p* |  |  | *i* |  |  |  | *j* |  |  |  |  | *k,r* |
| *Ap* | … … | | *Ai* | *Ai*+1 | … … | | *Aj* | *Aj*+1 | *Aj*+2 | … | *Ar*−1 | *Ar* |
| **< *x*** | | | | **= *x*** | | | | | **> *x*** | | | |

PARTITION*’*需要进行*r*−*p*次迭代，每次迭代花费常数时间。所以PARTITION*’*的运行时间为Θ(*n*)。

针对数组所有元素都相同的情况，PARTITION*’*对数组的划分会产生2个空的子问题。由于子问题为空，所以QUICKSORT在第一次PARTITION*’*后就结束。所以针对数组所有元素都相同的情况，QUICKSORT的时间复杂度为Θ(*n*)。

PARTITION*’*对元素互异的情况没有改进，所以QUICKSORT的期望时间复杂度仍然为Θ(*n*lg*n*)。

**c-问题分析**

RANDOMIZED\_PARTITION’(*A*, *p*, *r*)

*i* = RANDOM(*p*, *r*)

exchange *A*[*r*] with *A*[*i*]

PARTITION’(*A*, *p*, *r*)

RANDOMIZED\_QUICKSORT’(*A*, *p*, *r*)

**if** *p* < *r*

(*q*, *t*) = RANDOMIZED\_PARTITION’(*A*, *p*, *r*)

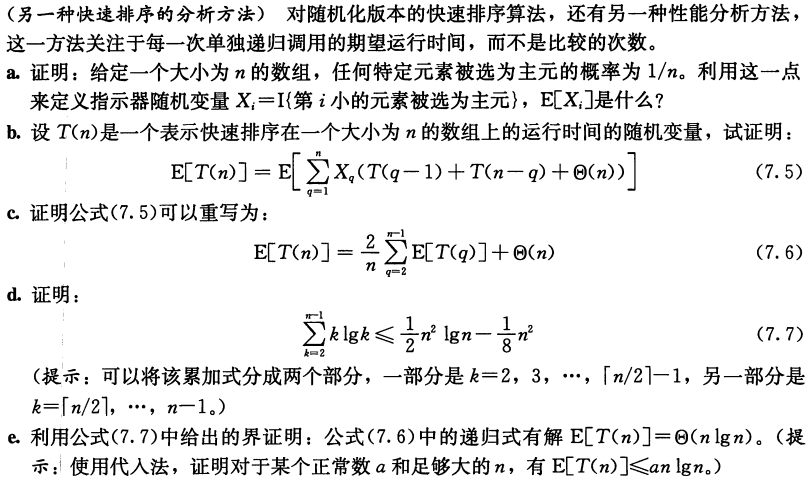
RANDOMIZED\_QUICKSORT’(*A*, *p*, *q*-1)

RANDOMIZED\_QUICKSORT’(*A*, *t*+1, *r*)

**d-问题分析**

简略分析一下。在PARTITION’中，数值相同的一组元素中只要有一个被选中作为主元，这一组元素就不会再与其他元素进行比较了。而在PARTITION中，只有被选中作为主元的那一个元素，才不会再与其他元素进行比较。相比较之下，如果存在元素相同的情况，QUICKSORT’的总的比较次数还要少于QUICKSORT。因此QUICKSORT’的期望运行时间仍然为O(*n*lg*n*)。

#### 思考题 7-3



**a-问题分析**

*E*[*Xi*] = Pr{第*i*小的元素被选为主元} = 1/*n*。

**b-问题分析**

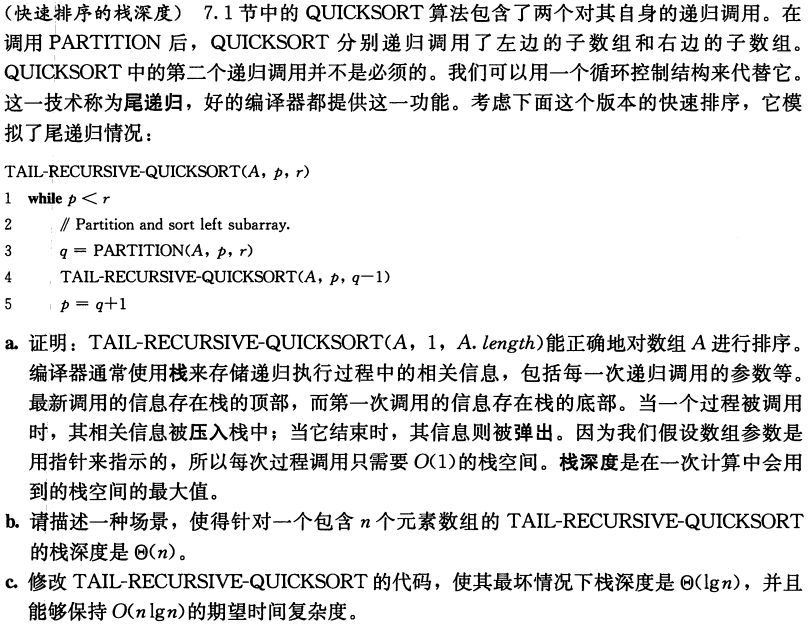
假设在对一个*n*个元素的数组进行快速排序时，第*q*小的元素被选为主元，于是可以得到*T*(*n*)的递归式*T*(*n*) = *T*(*q*−1) + *T*(*n*−*q*) + Θ(*n*)。



**c-问题分析**



#### 思考题 7-4



**a-问题分析**

TAIL-RECURSIVE-QUICKSORT所做的事情与QUICKSORT完全一致。在QUICKSORT(*A*, *p*, *r*)函数中，在递归调用QUICKSORT(*A*, *p*, *q*-1)后，紧接着递归调用QUICKSORT(*A*, *q*+1, *r*)。即在确定划分子数组后，递归排序左边的子数组，再递归排序右边的子数组。而在TAIL-RECURSIVE -QUICKSORT(*A*, *p*, *r*)函数中，在递归调用TAIL-RECURSIVE -QUICKSORT(*A*, *p*, *q*-1)后，即排序左边的子数组，紧接着让*p* = *q*+1，再进入下一轮迭代。在下一轮迭代中，会排序右边的子数组。

**b-问题分析**

考虑一个单调递降排列并且规模为*n*的数组，调用PARTITION后，被分割为2个子数组：左边子数组规模为*n*-1，右半边子数组规模为0。左边的子数组排序通过递归调用QUICKSORT来完成，而右边的子数组为空。因此，规模为*n*的数组的递归深度*D*(*n*) = *D*(*n*-1) + 1，这个递归式的解为Θ(*n*)。

所以，对于一个单调递降排列并且规模为*n*的数组，TAIL-RECURSIVE –QUICKSORT的栈深度为Θ(*n*)。

**c-问题分析**

在TAIL-RECURSIVE –QUICKSORT的实现中，每次递归调用都排序左边的子数组。这样的处理方式遇到某些特殊的输入（参考问题b的分析）会导致递归的栈深度为Θ(*n*)。可以考虑做这样的改进：每次递归调用都排序规模较小的子数组，参考下面的伪代码。

TAIL\_RECURSIVE \_QUICKSORT’(*A*, *p*, *r*)

**while** *p* < *r*

*q* = PARTITION(*A*, *p*, *r*)

**if** *q* – *p* <= *r* – *q*

TAIL\_RECURSIVE \_QUICKSORT’(*A*, *p*, *q*-1)

*p* = *q*+1

**else**

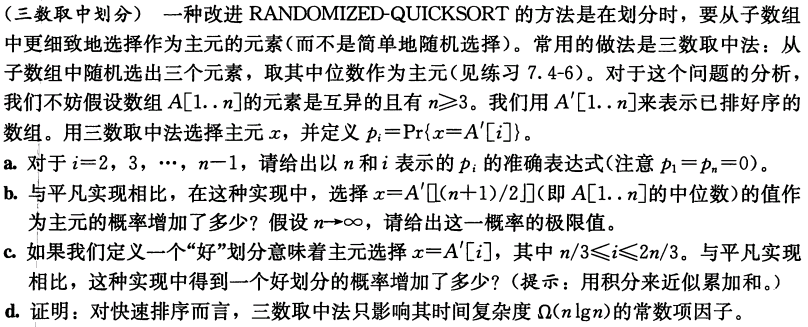
TAIL\_RECURSIVE \_QUICKSORT’(*A*, *q*+1, *r*)

*r* = *q*-1

}

这个算法的最坏情况发生在每次调用PARTITION都产生一个均衡的划分。这种情况下，栈的深度递归式可以表示为*D*(*n*) = *D*(*n*/2) + 1，其解为Θ(lg*n*)。因此最坏情况的时间复杂度为Θ(*n*lg*n*)。所以，针对任意输入，算法能保证O(*n*lg*n*)的时间复杂度。

#### 思考题 7-5



**a-问题分析**

*A’*[1 .. *n*]是一个已排好序的数组，选择其中一个元素*x* = *A’*[*i*]作为主元。该元素将*A’*[1 .. *n*]分为3部分：*A’*[1 .. *i*-1]、*A’*[*i*+1 .. *n*]以及*A’*[*i*]本身。如果采用三数取中法，要选中*A’*[*i*]作为主元，只有一种情况：在*A’*[1 .. *i*-1]中选取一个元素，在*A’*[*i*+1 .. *n*]中选取一个元素，选取*A’*[*i*]本身。这种情况可能有种。而从*A’*[1 .. *n*]中选取3个数的情况总共有种。因此，选中*A’*[*i*]作为主元的概率为



**b-问题分析**

在PARTITION的平凡实现中，任意一个元素都有相等的可能性被选为主元，因此选择中位数作为主元的概率为*qM* = 1/*n*。

如果采用三数取中法，选择中位数作为主元的概率为



下面分两种情况讨论：

1) n为偶数





2) n为奇数





从以上分析可以看出，当*n*趋近于∞时，*pM*/*qM*趋近于1.5。因此，与平凡方法相比，三数取中法选取中位数作为主元的概率在*n*足够大时约等于1.5。

**c-问题分析**

在PARTITION的平凡实现中，任意一个元素都有相等的可能性被选为主元，因此得到一个“好”的划分的概率为



如果采用三数取中法，得到一个“好”的划分的概率为



所以，与PARTITION平凡实现相比，三数取中法得到一个“好”的划分的概率增加的倍数为

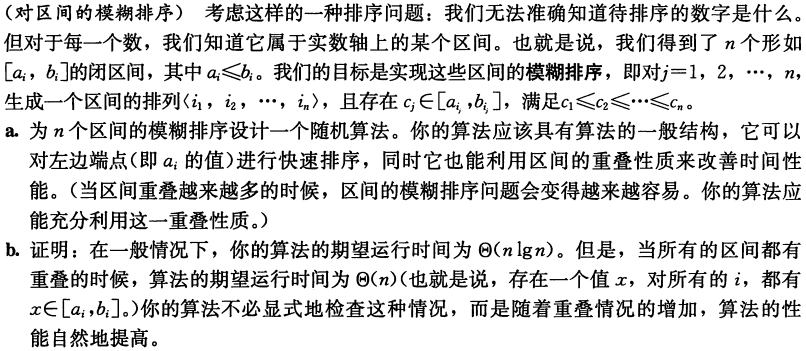


当*n*趋近于∞时，这一比值的极限为。

**d-问题分析**

简略分析。无论是平凡的PARTITON方法，还是三数取中法，快速排序的时间下限都为Ω(*n*lg*n*)，因为快速排序的最好情况的运行时间就是Θ(*n*lg*n*)。

#### 思考题 7-6



**a-问题分析**

考虑2个区间[*ai*, *bi*]和[*aj*, *bj*]之间的关系，分为3种情况：

1) *bi* < *aj*

此时，2个区间不相交，并且[*ai*, *bi*]在[*aj*, *bj*]的左边。对任意的*ci*[*ai*, *bi*]和*cj*[*aj*, *bj*]，必有*ci* < *cj*。所以，在模糊排序中，[*ai*, *bi*]必须要排在[*aj*, *bj*]的左边。因此，可以认为[*ai*, *bi*] < [*aj*, *bj*]。

2) *bj* < *ai*

此时，2个区间不相交，并且[*ai*, *bi*]在[*aj*, *bj*]的右边。对任意的*ci*[*ai*, *bi*]和*cj*[*aj*, *bj*]，必有*ci* > *cj*。所以，在模糊排序中，[*ai*, *bi*]必须要排在[*aj*, *bj*]的右边。因此，可以认为[*ai*, *bi*] > [*aj*, *bj*]。

3) *bi* ≥ *aj*并且*bj* ≥ *ai*

此时，2个区间重叠。这意味着至少存在一个*c*，使得*c*[*ai*, *bi*]并且*c*[*aj*, *bj*]。根据模糊排序的定义，[*ai*, *bi*]和[*aj*, *bj*]的排列顺序可以任意。因此，可以认为[*ai*, *bi*] = [*aj*, *bj*]。

根据以上分析，可以将区间当成可以比较的数字一样，对一组区间调用快速排序。为了改善区间重叠情况的排序性能，可以利用思考题7-2提供的方法，对区间相等的情况作特殊考虑。

INTERVAL\_COMPARE(*a*1, *a*2)

**if** *a*1.*right* < *a*2.*left*

**return** LESS

**elseif** *a*2.*right* < *a*1.*left*

**return** LARGER

**else**

**return** EQULE

INTERVAL\_PARTITION(*A*, *p*, *r*)

*x* = *p*[*r*]

*i* = *j* = *p*-1

**for** *k* = *p* 🡪 *r*-1

**if** INTERVAL\_COMPARE(*A*[*k*], *A*[*r*]) == EQULE

*j*++

exchange *A*[*k*] with *A*[*j*]

**elseif** INTERVAL\_COMPARE(*A*[*k*], *A*[*r*]) == LESS

*i*++

*j*++

exchange *A*[*k*] with *A*[*i*]

exchange *A*[*k*] with *A*[*j*]

exchange *A*[*j*+1] with *A*[*r*]

**return** (*i*+1) and (*j*+1)

INTERVAL\_QUICKSORT(*A*, *p*, *r*)

**if** *p* < *r*

(*q*, *t*) = INTERVAL\_PARTITION(*A*, *p*, *r*)

INTERVAL\_QUICKSORT(*A*, *p*, *q*-1)

INTERVAL\_QUICKSORT(*A*, *t*+1, *r*)

**b-问题分析**

根据思考题7-2的分析，如果所有区间都重叠，这意味着所有区间都相等，这种情况的排序时间复杂度为Θ(*n*)。也如思考题7-2的分析，该算法的期望运行时间为Θ(*n*lg*n*)。