

《计算机算法设计与分析》

第三章 分治法

马丙鹏

2020年09月29日



第三章 分治法

- 3.1 一般方法
- 3.2 二分检索
- 3.3 找最大和最小元素
- 3.4 归并排序
- 3.5 快速排序
- 3.6 选择问题
- 3.7 斯特拉森矩阵乘法



3.5 快速排序

■ 概述

- 任何一个基于比较来确定两个元素相对位置的排序算法需要 $\Omega(n\log n)$ 计算时间。
- 如果我们能设计一个需要 $O(n\log n)$ 时间的排序算法，则在渐近的意义下，这个排序算法就是最优的。
- 许多排序算法都是追求这个目标。

■ 快速排序的定义

- 快速排序是一种基于划分的排序方法；
- 通过反复地对待排序集合进行划分达到排序目的的排序算法。
- 快速排序在平均情况下需要 $O(n\log n)$ 时间。



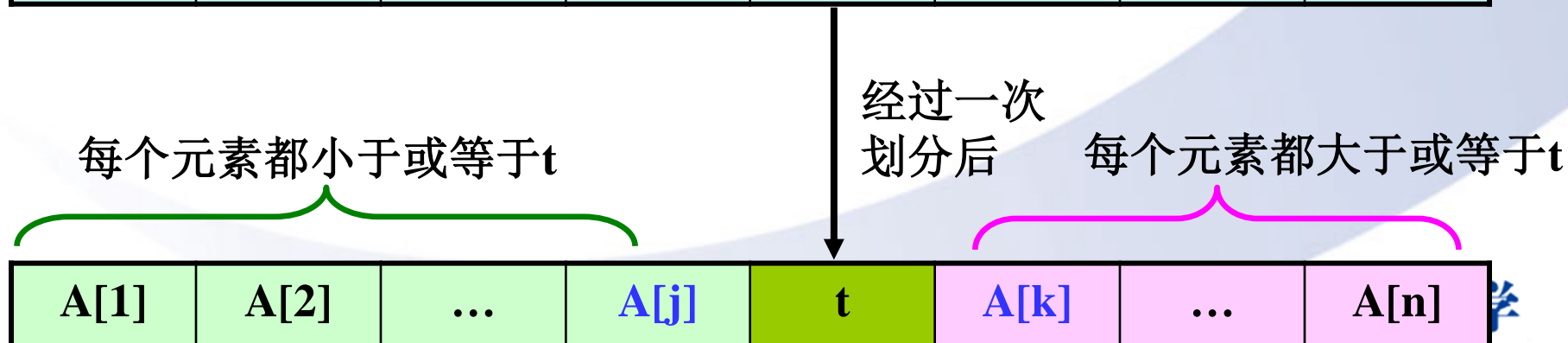
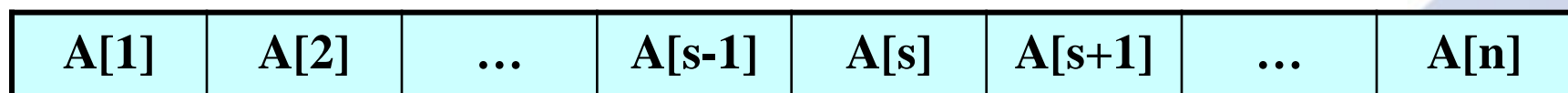
3.5 快速排序

■ 快速排序的定义

□ **划分**：选取待排序集合A中的某个元素t，按照与t的大小关系重新整理A中元素，使得整理后的序列中所有在t以前出现的元素均**小于等于t**，而所有出现在t以后的元素均**大于等于t**。这一元素的整理过程称为划分(Partitioning)。

➤ 元素t称为**划分元素**。

划分元素t



3.5 快速排序

■ 算法描述

- //将 $A(m), A(m+1), \dots, A(p-1)$ 中的元素按如下方式重新排列：如果最初 $t=A(m)$ ，则在重排完成之后，对于 m 和 $p-1$ 之间的某个 q ，有 $A(q)=t$ ，并使得对于 $m \leq k < q$ ，有 $A(k) \leq t$ ，而对于 $q < k < p$ ，有 $A(k) \geq t$ //
- //退出时， p 置为划分元素所在的下标位置 //
- 注：算法对集合 $A(m:p-1)$ 进行划分。并使用待划分区间的第一个元素 $A(m)$ 作为划分元素
- $A(p)$ 不在划分区间内，但被定义，且 $A(p) \geq A(m)$ ，用于限界



算法3.2 用 $A(m)$ 划分集合 $A(m:p-1)$

procedure PARTITION(m, p)

integer m, p, i ; **global** $A(m:p-1)$

$v \leftarrow A(m)$; $i \leftarrow m$ // $A(m)$ 是划分元素//

loop

loop $i \leftarrow i+1$ **until** $A(i) \geq v$ **repeat** // i 由左向右移//

loop $p \leftarrow p-1$ **until** $A(p) \leq v$ **repeat** // p 由右向左移//

if $i < p$ **then**

call INTERCHANGE($A(i), A(p)$)

else exit

endif

repeat

$A(m) \leftarrow A(p)$;

$A(p) \leftarrow v$ //划分元素在位置 p //

end PARTITION



3.5 快速排序

■ 实例

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
A:	65	70	75	80	85	60	55	50	45	$+\infty$	2	9

|.....|

A:	65	45	75	80	85	60	55	50	70	$+\infty$	3	8
----	----	----	----	----	----	----	----	----	----	-----------	---	---

|.....|

A:	65	45	50	80	85	60	55	75	70	$+\infty$	4	7
----	----	----	----	----	----	----	----	----	----	-----------	---	---

|.....|

A:	65	45	50	55	85	60	80	75	70	$+\infty$	5	6
----	----	----	----	----	----	----	----	----	----	-----------	---	---

|.....|

交换A: 65 45 50 55 60 85 80 75 70 $+\infty$ 6 5

划分——>|.....|

元素A: 60 45 50 55 65 85 80 75 70 $+\infty$



中国科学院大学

University of Chinese Academy of Sciences 7

3.5 快速排序

■ 快速排序

- 经过一次“划分”后，实现了对集合元素的调整：其中一个子集合的所有元素均小于等于另外一个子集合的所有元素。
- 按同样的策略对两个子集合进行排序处理。
- 当子集合排序完毕后，整个集合的排序也完成了。
- 这一过程避免了子集合的归并操作。
- 通过反复使用划分过程PARTITION实现对集合元素排序的算法称为快速排序。



3.5 快速排序

■ 3.5 快速排序

算法3.13 快速排序

procedure QUICKSORT(p, q)

//将数组A(1:n)中的元素A(p),...A(q)按递增的方式排序。

A(n+1)有定义, 且假定 $A(n+1) \leftarrow +\infty$ //

integer p, q;

global n, A(1: n)

if p<q **then**

 j←q+1 //进入时, A(j)定义了划分区间[p,q]的**上界**, 初次调用时j=n+1

call PARTITION(p, j) //出口时, j带出此次划分后划分元素所在的**坐标位置** /

call QUICKSORT(p, j-1) //前一子集合上递归调用

call QUICKSORT(j+1, q) //后一子集合上递归调用

endif

end QUICKSORT



3.5 快速排序

	1	2	3	4	5	6	7	8	9
1	[65	70	75	80	85	60	55	50	45]
2	[60	45	50	55]	65	[85	80	75	70]
3	[55	45	50]	60	65	[85	80	75	70]
4	[50	45]	55	60	65	[85	80	75	70]
5	[45]	50	55	60	65	[85	80	75	70]
6	45	50	55	60	65	[85	80	75	70]
7	45	50	55	60	65	[70	80	75]	85
8	45	50	55	60	65	70	[80	75]	85
9	45	50	55	60	65	70	[75]	80	85
10	45	50	55	60	65	70	75	80	85



3.5 快速排序

■ 快速排序分析

□统计的对象：元素的比较次数，记为： $C(n)$

□两点假设

➤参加排序的 n 个元素各不相同

➤PARTITION中的划分元素 v 是随机选取的(针对平均情况的分析)

□随机选取划分元素：

➤在划分区间 $[m, p]$ 随机生成某一坐标：

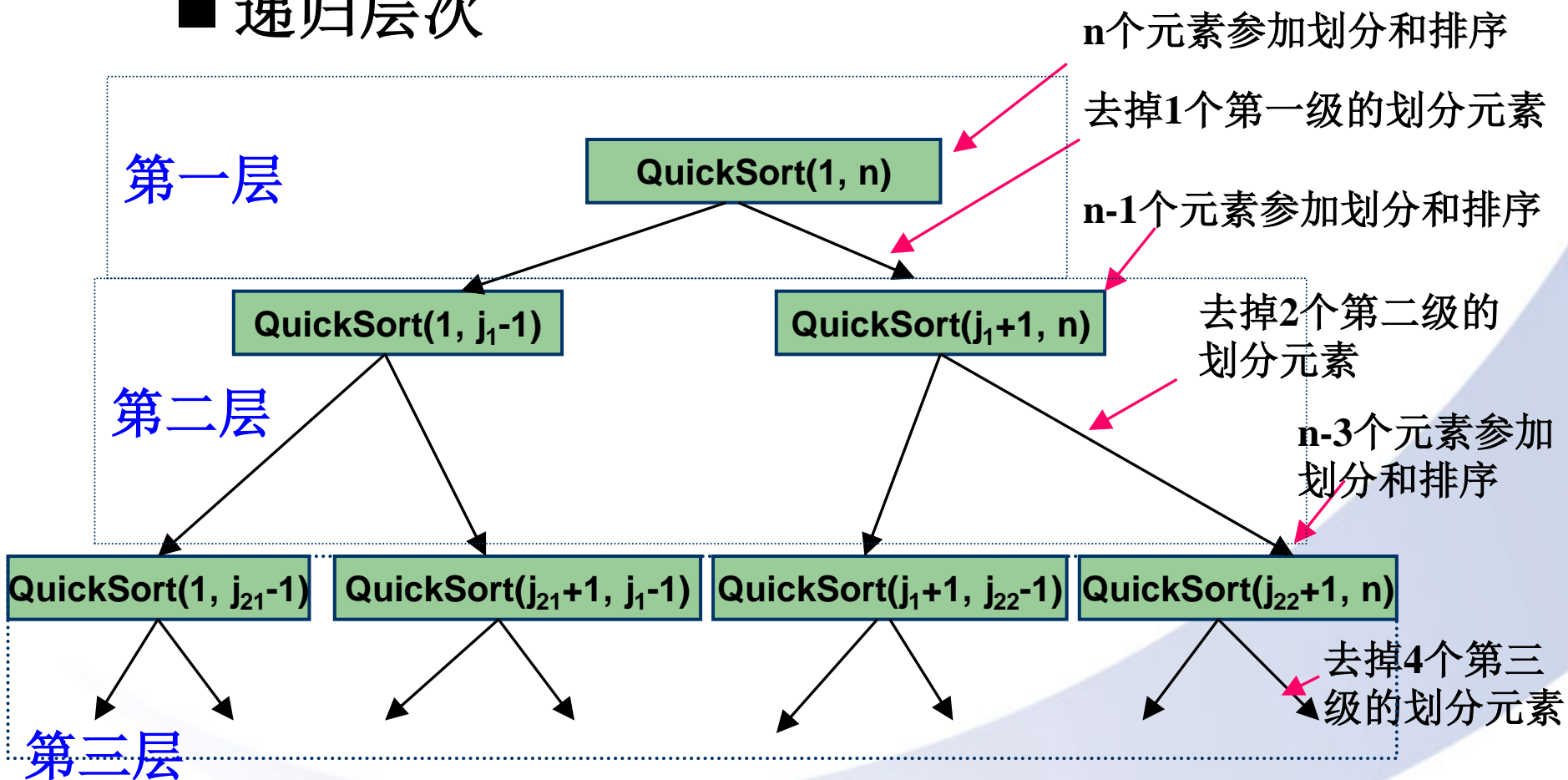
$i \leftarrow \text{RANDOM}(m, p-1)$; 调换 $A(m)$ 与 $A(i)$: $v \leftarrow A(i)$;
 $A(i) \leftarrow A(m)$; $i \leftarrow m$

➤作用：将随机指定的划分元素的值依旧调换到 $A(m)$ 位置。之后，算法主体不变，仍从 $A(m)$ 开始执行划分操作



3.5 快速排序

■ 递归层次



3.5 快速排序

■ 递归层次

□ 设在任一级递归调用上，调用PARTITION处理的所有元素总数为 r ，则初始时 $r=n$ ，以后的每级递归上，由于删去了上一级的划分元素，故 r 比上一级至少少1：

□ 理想情况，

➤ 第一级少1，第二级少2，第三级少4，...；

□ 最坏情况

➤ 每次仅减少1(如集合元素已经按照递增或递减顺序排列)

k	0	1	2	3	4	...	n-1	n	n+1
a[k]	--	n	1	2	3	...	n-2	n-1	∞



3.5 快速排序

■ 最坏情况分析

- 记最坏情况下的元素比较次数是 $C_w(n)$;
- PARTITION一次调用中的元素比较数是 $p-m+1$
- 最坏情况下(第 i 次调用Partition所得的划分元素恰好是第 i 小元素), 每级递归调用的元素总数仅比上一级少1, 故 $C_w(n)$ 是 r 由 n 到2的累加和。

$$\text{即: } C_w(n) = \sum_{2 \leq r \leq n} r = O(n^2)$$



3.5 快速排序

■ 最坏情况分析

k	0	1	2	3	4	...	n-1	n	n+1
a[k]	--	n	1	2	3	...	n-2	n-1	∞

$$C_W(n) = \begin{cases} 3 & n = 2 \\ n + 1 + C_W(n-1) & n > 2 \end{cases}$$



渐近时间复杂度

$$C_W(n) = O(n^2)$$



中国科学院大学

University of Chinese Academy of Sciences 15

3.5 快速排序

■ 最好情况分析

- 在最好情况下，每次划分所取的划分元素恰好为中值，即每次划分都产生两个大小为 $n/2$ 的区域

$$C_B(n) = \begin{cases} 3 & n = 2 \\ 2C_B((n-1)/2) + n + 1 & n > 2 \end{cases}$$



渐近时间复杂度

$$C_B(n) = O(n \log n)$$



中国科学院大学

University of Chinese Academy of Sciences 16

3.5 快速排序

■ 平均情况分析

- **平均情况**是指集合中的元素以任意一种顺序排列，且任选所有可能的元素作为划分元素进行划分和排序，在这些所有可能的情况下，算法执行性能的平均值。



3.5 快速排序

■ 平均情况分析

□ 设调用PARTITION(m, p)时, 所选取划分元素v恰好是A(m:p-1)中的第i小元素($1 \leq i \leq p-m$)的概率相等。则经过一次划分, 所留下的待排序的两个子文件恰好是A(m:j-1)和A(j+1:p-1)的概率是: $1/(p-m)$, $m \leq j < p$ 。记平均情况下的元素比较次数是 $C_A(n)$; 则有,

$$C_A(n) = n + 1 + \frac{1}{n} \sum_{1 \leq k \leq n} (C_A(k-1) + C_A(n-k))$$

□ 其中

➤ $n+1$ 是PARTITION第一次调用时所需的元素比较次数。

➤ $C_A(0) = C_A(1) = 0$



3.5 快速排序

■ 平均情况分析

化简上式可得：

$$\begin{aligned}C_A(n)/(n+1) &= C_A(n-1)/n + 2/(n+1) \\&= C_A(n-2)/(n-1) + 2/n + 2/(n+1) \\&= C_A(n-3)/(n-2) + 2/(n-1) + 2/n + 2/(n+1) \\&\dots \\&= C_A(1)/2 + 2 \sum_{3 \leq k \leq n+1} 1/k\end{aligned}$$

由于

$$\sum_{3 \leq k \leq n+1} 1/k \leq \int_2^{n+1} \frac{dx}{x} < \log_e(n+1)$$

所以得， $C_A(n) < 2(n+1)\log_e(n+1) = O(n \log n)$



3.5 快速排序

■ 空间分析

□ 最坏情况下，

➤ 递归的最大深度为 $n-1$

➤ 需要栈空间： $O(n)$

□ 使用一个迭代模型可以将栈空间总量减至 $O(\log n)$



3.5 快速排序

■ 快速排序算法的迭代模型

□ 处理策略：

➤ 每次在Partition将文件A(p:q)分成两个文件A(p:j-1)和A(j+1, q)后，先对其中较小的子文件进行排序。当小的子文件排序完成后，再对较大的子文件进行排序。

□ 栈：需要一个栈空间保存目前暂不排序的较大子文件。并在较小子文件排序完成后，从栈中退出最新的较大子文件进行下一步排序。

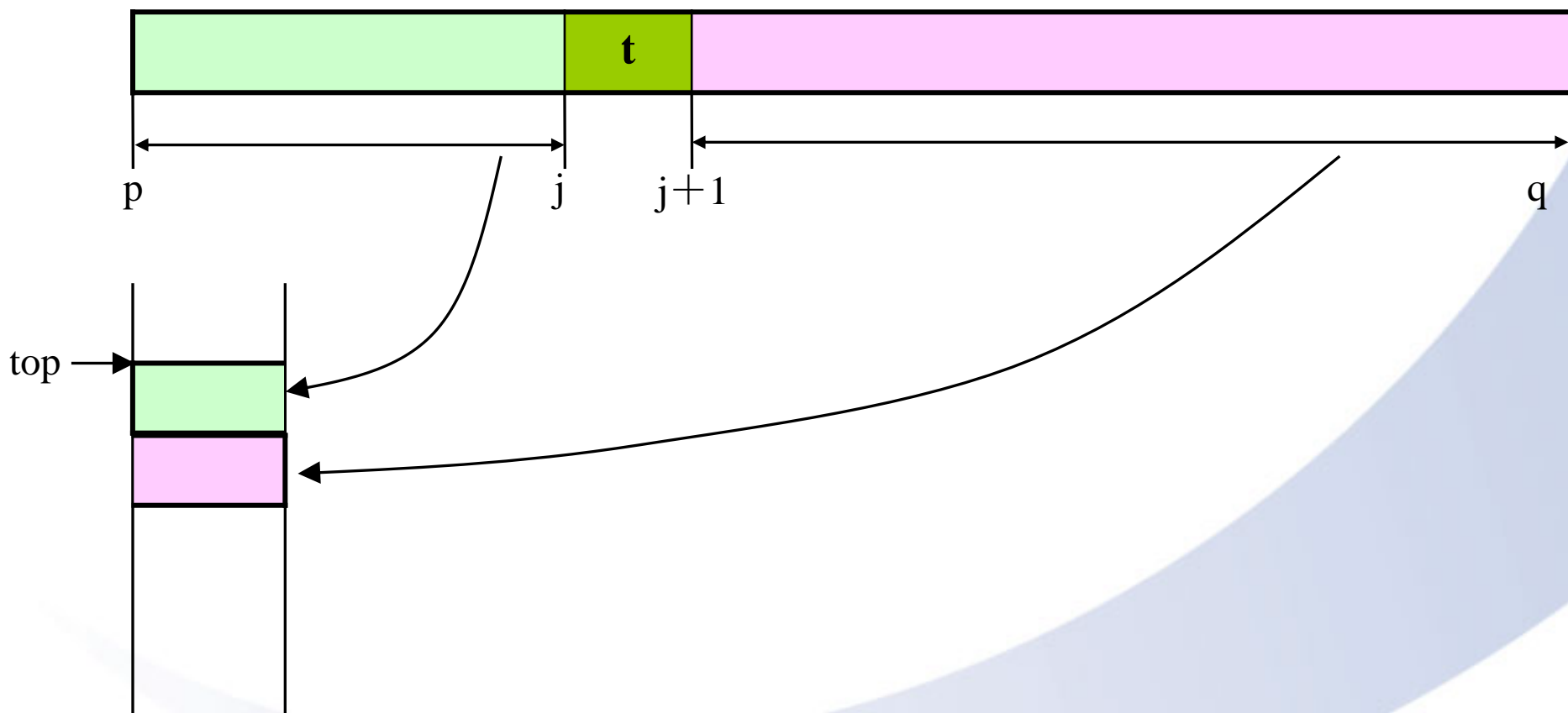
□ 栈空间需要量：O(logn)

□ 算法的终止：当栈空时，整个排序过程结束。



3.5 快速排序

■ 快速排序算法的迭代模型



算法3.14 QuickSort2(p, q)

procedure QUICKSORT2(p, q)

integer STACK(1:max), top //max=2

global A(1:n); **local integer** j

top←0

loop

while p<q **do**

j←q+1

call PARTITION(p, j);

if j - p < q - j //先对较小的子文件排序，并将规模较大的子文件入栈

then STACK(top+1)←j+1; STACK(top+2)←q; q←j-1

else STACK(top+1)←p; STACK(top+2)←j-1; p←j+1

endif

top←top+2 //调整栈顶指针

repeat

if top=0 **then return endif** //如果栈为空，算法结束

q←STACK(top); p←STACK(top-1); top←top-2 //从栈中退出先前保存的较大的子文件

repeat

end QUICKSORT2



中国科学院大学

University of Chinese Academy of Sciences 23

3.5 快速排序

■ 快速排序算法的迭代模型

□ 空间分析

➤ 算法3.14的最大空间是 $O(\log n)$

➤ 推导：设算法所需的最大栈空间是 $S(n)$ ，则有

$$S(n) = \begin{cases} 2 + S(\lfloor (n-1)/2 \rfloor) & n > 1 \\ 0 & n \leq 1 \end{cases}$$



3.5 快速排序

■ 快速排序与归并排序比较

□ 对排序算法应该从以下几个方面综合考虑：

- ① 时间复杂性；
- ② 空间复杂性；
- ③ 稳定性；
- ④ 算法简单性；
- ⑤ 待排序记录个数 n 的大小；
- ⑥ 记录本身信息量的大小；
- ⑦ 关键码的分布情况



3.5 快速排序

■ 快速排序与归并排序比较

□ 时间复杂度:

排序方法	平均情况	最好情况	最坏情况
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$

□ 两者平均情况时间都是 $O(n\log n)$ ，平均情况下哪种快一些？

□ 实验证明快速排序要快一些。



3.5 快速排序

■ 快速排序与归并排序比较

□ 空间复杂度:

排序方法	辅助空间
插入排序	$O(1)$
冒泡排序	$O(1)$
快速排序	$O(\log_2 n) \sim O(n)$
归并排序	$O(n)$



3.5 快速排序

■ 快速排序与归并排序比较

□ 简单性:

- 一类是简单算法，包括直插入排序和冒泡排序，
- 另一类是改进后的算法，包括快速排序和归并排序，这些算法较复杂。

□ 待排序记录个数比较:

- n 越小，采用简单排序方法越合适，
- n 越大，采用改进的排序方法越合适，
- 因为 n 越小， $O(n^2)$ 同 $O(n \log n)$ 的差距越小，并且输入和调试简单算法比高效算法要容易。



3.5 快速排序

■ 快速排序与归并排序比较

□ 简单性数据的分布情况比较：

- 当待排序数据初始有序时，插入排序和冒泡排序能达到 $O(n)$ 的时间复杂度；
- 对于快速排序而言，这是最坏的情况，此时性能蜕化为 $O(n^2)$ ；
- 归并排序的性能不受影响。



作业-课后练习6

- 对下面的三组数据进行快速排序
 - (45, 36, 18, 53, 72, 30, 48, 93, 15, 36)
 - (1, 1, 1, 1, 1)
 - (5, 5, 8, 3, 4, 3, 2)
 - 给出每一次划分后的结果。



作业-课后练习7

■ 问题描述

- 在PARTITION(m, p)中, 将语句if $i < p$ 改为if $i \leq p$, 有何优缺点?
- 在数据集(5, 4, 3, 2, 5, 8, 9)上执行这两个算法, 看看他们在执行时有何不同。



小结

