

## 第二部分：排序和顺序统计量

INTRODUCTION TO

# ALGORITHMS

THIRD EDITION

# 快速排序

《算法导论》—— 第6讲

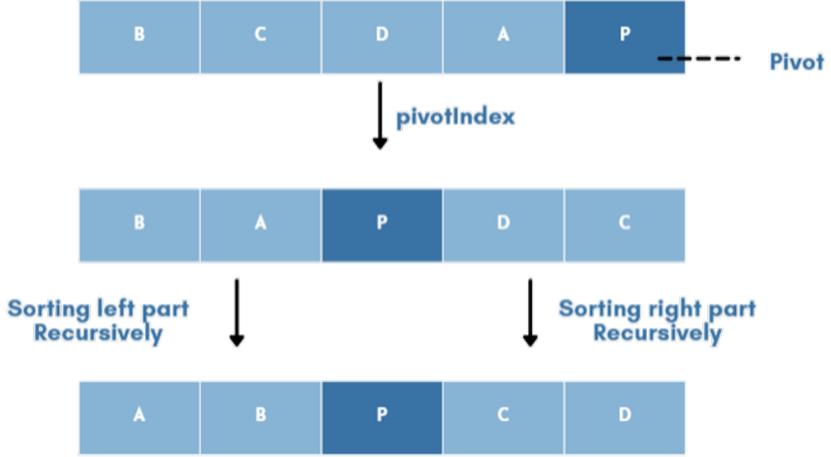
[jiacaicui@163.com](mailto:jiacaicui@163.com)

# 内容提要

- 快速排序的描述
  - 难分易合型的分治算法
  - 重要的划分子过程
- 快速排序的性能
  - 最坏情况运行时间： $\Theta(n^2)$
  - 平均情况运行时间： $\Theta(n \log n)$  且其中的常数系数很小
  - 原址排序
  - 实际应用中最好的选择
- 快速排序的随机化版本
  - 分析使用随机划分的快速排序的期望时间

# 快速排序的描述

最实用的排序算法



# 快速排序的分治策略

```
QUICKSORT( $A, p, r$ )
```

```
1  if  $p < r$ 
```

```
2       $q = \text{PARTITION}(A, p, r)$ 
```

```
3      QUICKSORT( $A, p, q - 1$ )
```

```
4      QUICKSORT( $A, q + 1, r$ )
```

- **分 (Divide)** : 数组  $A[p..r]$  被划分为两个 (可能为空) 子数组  $A[p..q - 1]$  和  $A[q + 1, r]$ , 使得  $A[p..q - 1]$  中的每一个元素都小于等于  $A[q]$ , 而  $A[q]$  也小于等于  $A[q + 1..r]$  中的每个元素。其中, 计算下标  $q$  也是划分过程的一部分。
- **治 (Conquer)** : 通过递归调用快速排序, 对子数组  $A[p..q - 1]$  和  $A[q + 1..r]$  进行排序。
- **合 (Combine)** : 因为子数组都是原址排序的, 所以不需要合并操作: 数组  $A[p..r]$  已经有序。

# 数组的划分

PARTITION( $A, p, r$ )

1  $x = A[r]$  → 主元 (pivot element)

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r - 1$

4     **if**  $A[j] \leq x$

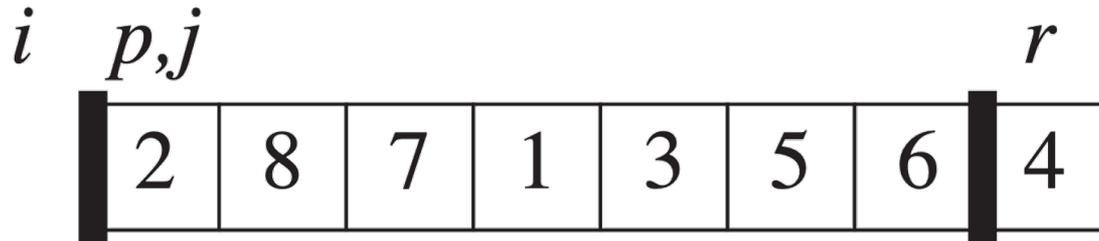
5          $i = i + 1$

6         exchange  $A[i]$  with  $A[j]$

7 exchange  $A[i + 1]$  with  $A[r]$

8 **return**  $i + 1$

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

PARTITION( $A, p, r$ )

1  $x = A[r]$  → 主元 (pivot element)

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r - 1$

4     **if**  $A[j] \leq x$

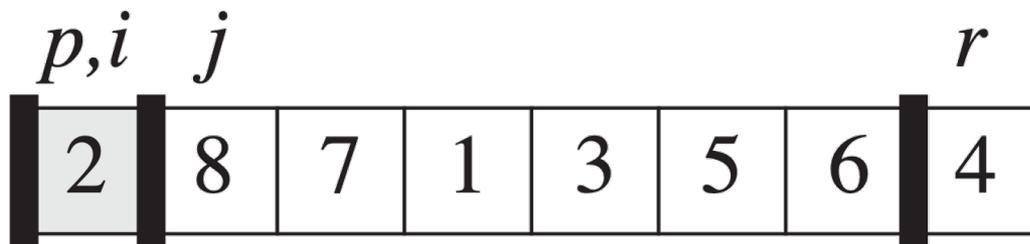
5          $i = i + 1$

6         exchange  $A[i]$  with  $A[j]$

7 exchange  $A[i + 1]$  with  $A[r]$

8 **return**  $i + 1$

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

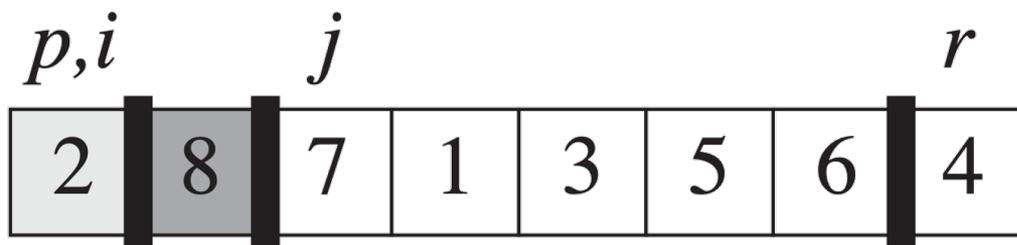
PARTITION( $A, p, r$ )

```

1   $x = A[r]$  → 主元 (pivot element)
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 

```

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

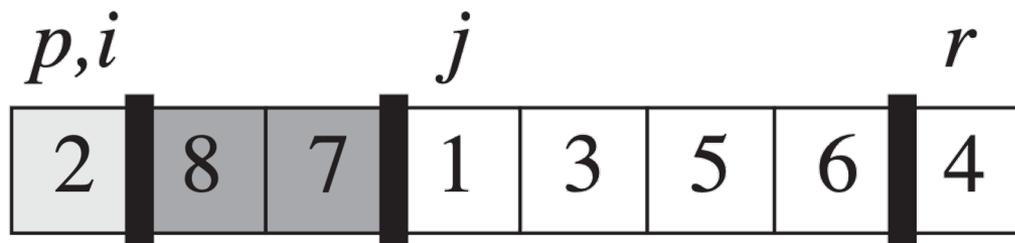
PARTITION( $A, p, r$ )

```

1   $x = A[r]$  → 主元 (pivot element)
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 

```

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

PARTITION( $A, p, r$ )

1  $x = A[r]$  → 主元 (pivot element)

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r - 1$

4     **if**  $A[j] \leq x$

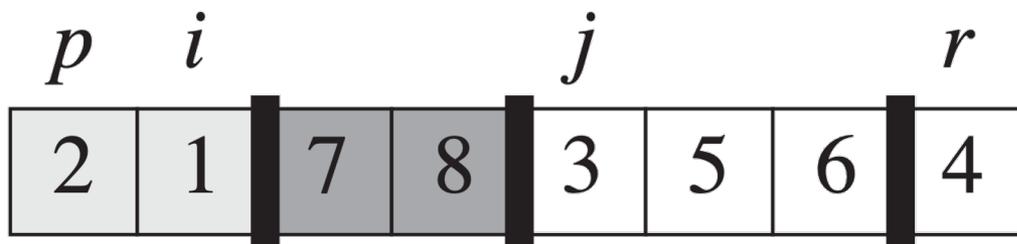
5          $i = i + 1$

6         exchange  $A[i]$  with  $A[j]$

7 exchange  $A[i + 1]$  with  $A[r]$

8 **return**  $i + 1$

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

PARTITION( $A, p, r$ )

1  $x = A[r]$  → 主元 (pivot element)

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r - 1$

4     **if**  $A[j] \leq x$

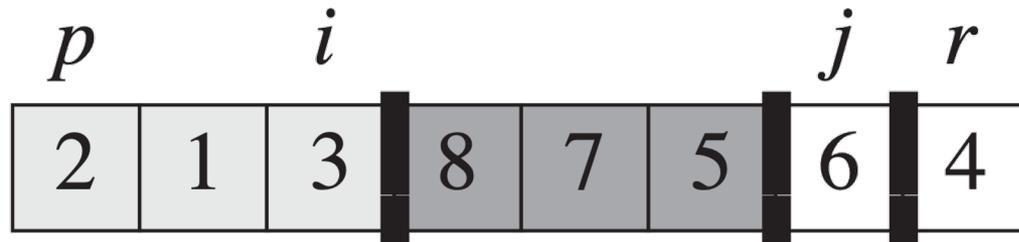
5          $i = i + 1$

6         exchange  $A[i]$  with  $A[j]$

7 exchange  $A[i + 1]$  with  $A[r]$

8 **return**  $i + 1$

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

PARTITION( $A, p, r$ )

1  $x = A[r]$  → 主元 (pivot element)

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r - 1$

4     **if**  $A[j] \leq x$

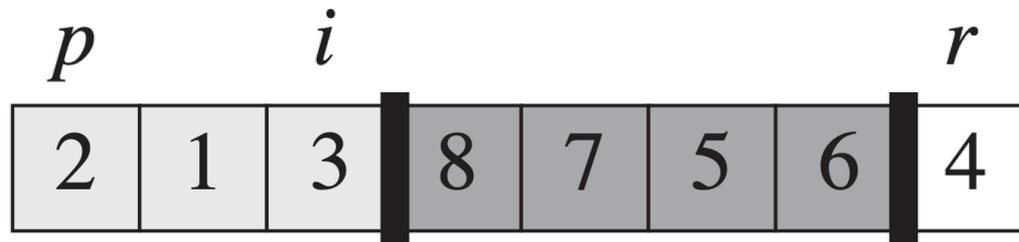
5          $i = i + 1$

6         exchange  $A[i]$  with  $A[j]$

7 exchange  $A[i + 1]$  with  $A[r]$

8 **return**  $i + 1$

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。



# 数组的划分

PARTITION( $A, p, r$ )

1  $x = A[r]$  → 主元 (pivot element)

2  $i = p - 1$

3 **for**  $j = p$  **to**  $r - 1$

4     **if**  $A[j] \leq x$

5          $i = i + 1$

6         exchange  $A[i]$  with  $A[j]$

7 exchange  $A[i + 1]$  with  $A[r]$

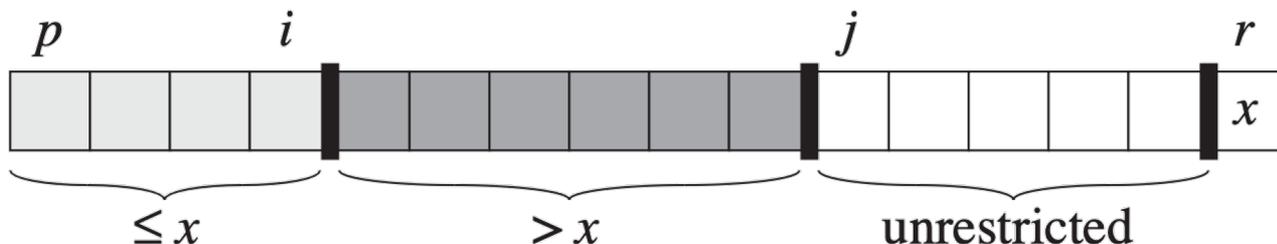
8 **return**  $i + 1$

- 实现了对子数组  $A[p..r]$  的原址 (in place) 重排。

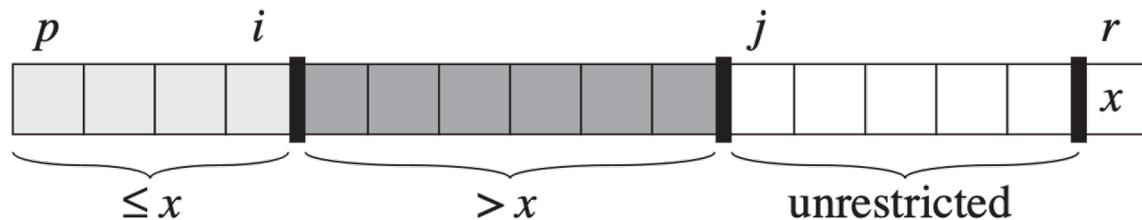


# PARTITION 的正确性

- 目的：
  - $A[p..q-1]$  中的每一个元素都小于等于  $A[q]$ ；
  - $A[q]$  也小于等于  $A[q+1..r]$  中的每个元素。
- 证明：**循环不变式**：在第 3 到 6 行的 **for** 循环每一次迭代开始前，对于任意数组下标  $k$ ，有：
  - 若  $p \leq k \leq i$ ，则  $A[k] \leq x$ 。
  - 若  $i+1 \leq k \leq j-1$ ，则  $A[k] > x$ 。
  - 若  $k = r$ ，则  $A[k] = x$ 。



# PARTITION 的正确性



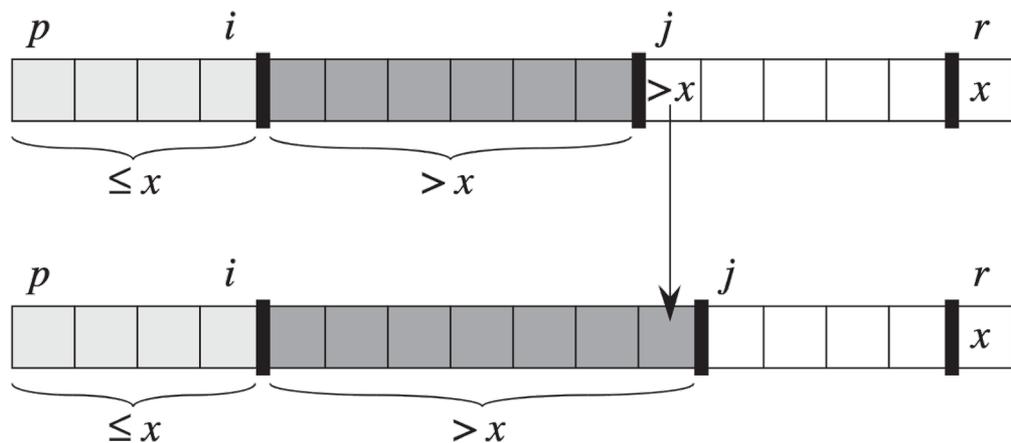
- **初始化**：第一次迭代开始前， $i = p - 1$ ， $j = p$ ，在  $p$  和  $i$  之间、 $i + 1$  和  $j - 1$  之间都不存在值，所以循环不变式的前两条显然满足；算法第 1 行的赋值满足了第三条循环不变式。
- **保持**：（见下一页PPT）
- **终止**：最后一次循环结束时， $j = r$ 。根据循环不变式， $A[p..i]$  中的元素都小于等于  $x$ ， $A[i + 1..r - 1]$  中的元素都大于等于  $x = A[r]$ 。
- 最后，算法第 7 行交换了  $A[i + 1]$  和  $A[r]$ 。此时， $A[p..i]$  中的元素都小于等于  $A[i + 1]$ ， $A[i + 2..r]$  中的元素都大于等于  $A[i + 1]$ 。因此，算法第 8 行返回  $i + 1$  是正确的。

# PARTITION 的正确性

**保持**：不变式第3条显然是保持的，因为循环体并没有改变  $A[r]$ 。

假设上一次迭代保持了不变式，根据第 4 行中条件判断的不同结果，我们需要考虑两种情况：

1.  $A[j] > x$  时，循环体的唯一操作是  $j$  的值加1。在  $j$  值增加后， $A[j-1] > x$ ，不变式第2条满足；其它项都保持不变，没有破坏不变式第1条。循环保持了不变式。

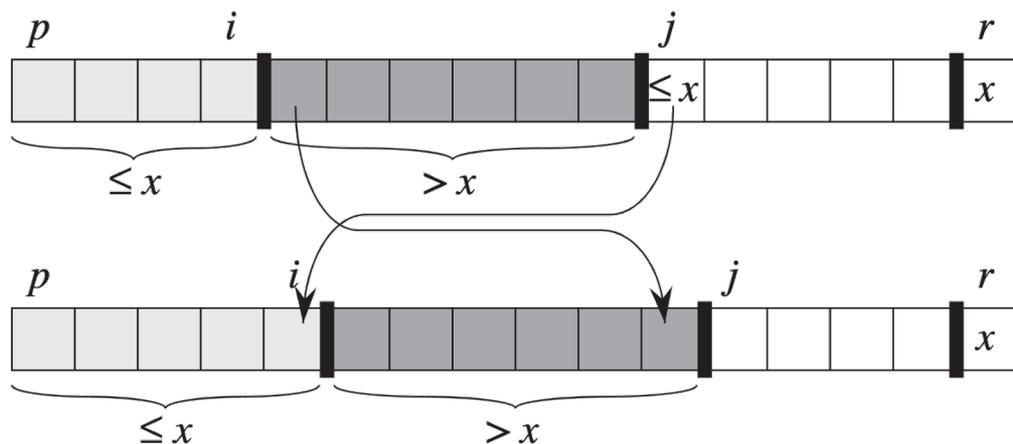


# PARTITION 的正确性

保持：不变式第3条显然是保持的，因为循环体并没有改变  $A[r]$ 。

假设上一次迭代保持了不变式，根据第 4 行中条件判断的不同结果，我们需要考虑两种情况：

2.  $A[j] \leq x$  时，将  $i$  值加 1，交换  $A[i]$  和  $A[j]$ ，再将  $j$  值加 1。因为进行了交换， $A[i] \leq x$ ，不变式第 1 条满足，且  $A[j-1] > x$ ，不变式第 2 条满足。循环保持了不变式。

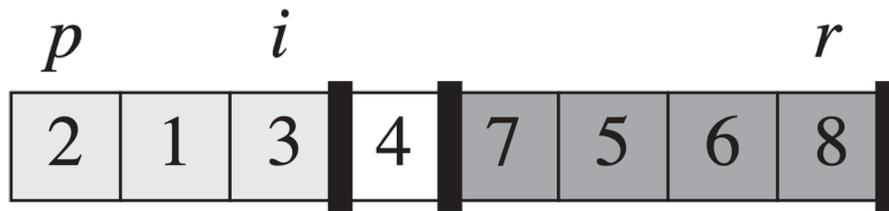


# PARTITION 的运行时间

PARTITION( $A, p, r$ )

```
1  $x = A[r]$ 
2  $i = p - 1$ 
3 for  $j = p$  to  $r - 1$ 
4     if  $A[j] \leq x$ 
5          $i = i + 1$ 
6         exchange  $A[i]$  with  $A[j]$ 
7 exchange  $A[i + 1]$  with  $A[r]$ 
8 return  $i + 1$ 
```

- PARTITION 的运行时间为  $\Theta(n)$ ，其中  $n = r - p + 1$ 。



# 快速排序的性能

快排性能的直观分析



# 最坏情况划分



- 当划分产生的两个子问题分别包含了  $n - 1$  个元素和 0 个元素时，快速排序的**最坏情况**发生了。
- 不妨假设算法中的每一次递归调用都出现了这种不平衡的划分，划分操作的代价为  $\Theta(n)$ 。

$$T(n) = T(n - 1) + T(0) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$$

- 最坏情况下，快速排序并不比插入排序更好。
- **输入数组完全有序时**，快速排序的时间复杂度仍然为  $\Theta(n^2)$ ，但插入排序的时间复杂度为  $O(n)$ 。

# 最好情况与平衡情况划分



- 在可能的最平衡的划分中，PARTITION 得到的两个子问题的规模分别为  $\lfloor \frac{n}{2} \rfloor$  和  $\lfloor \frac{n}{2} \rfloor - 1$ ，忽略细节，递归式为

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

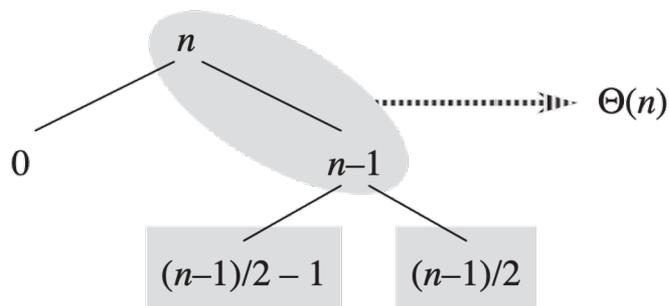
- 更一般地（详见[练习3-2/问题2](#)）：

$$T(n) = T(\alpha n) + T((1 - \alpha)n) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

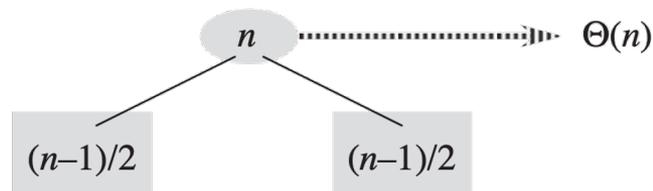
- 只要划分是常数，算法的运行时间总是  $O(n \log n)$ 。
- 快速排序的平均运行时间更接近于其最好情况，而非最坏情况。

# 对于平均情况的直观观察

- 在平均情况下，PARTITION 所产生的划分同时混合有“好”的和“差”的划分，且在执行过程中是随机出现的。
- 基于直觉，假设好的和差的划分交替出现在递归树的各层上，并且好的划分是最好情况划分，而差的划分是最坏情况划分。
- 一次最差划分紧接着一次最好划分与一次最好划分是等价的。



(a)



(b)

- 当好和差的划分交替出现时，快速排序的时间复杂度依旧是  $O(n \log n)$ 。



# 快速排序的随机化版本

主动随机创造平均情形

# 随机划分

- 朴素思路：随机打乱输入序列，然后调用QUICKSORT。
- 随机抽样的思路：在PARTITION操作中随机选取一个元素作为主元（pivot）。
  - 具体操作上，将随机选取的主元和最后一个元素交换，然后调用PARTITION，这样可以复用以前的代码。

**RANDOMIZED-PARTITION**( $A, p, r$ )

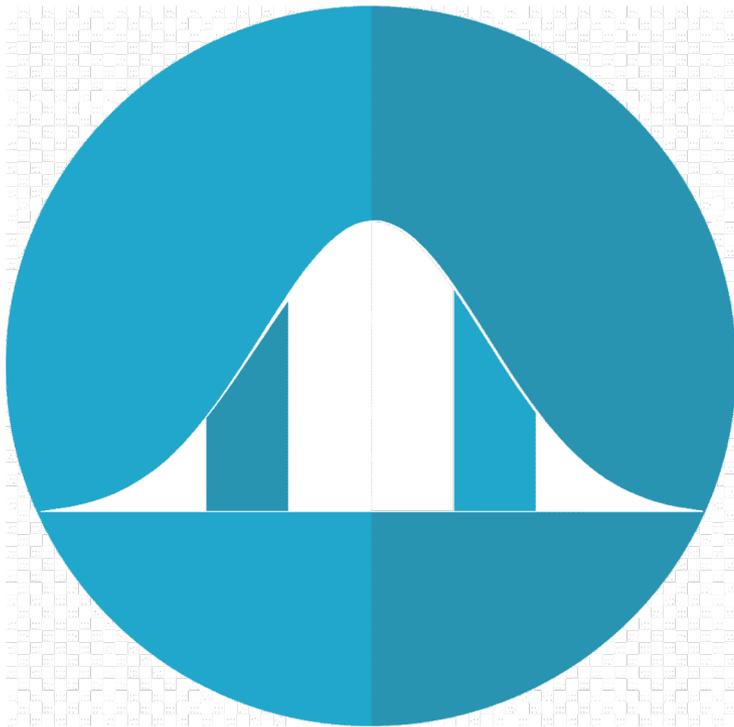
```
1  $i = \text{RANDOM}(p, r)$   
2 exchange  $A[r]$  with  $A[i]$   
3 return PARTITION( $A, p, r$ )
```

**RANDOMIZED-QUICKSORT**( $A, p, r$ )

```
1 if  $p < r$   
2      $q = \text{RANDOMIZED-PARTITION}(A, p, r)$   
3     RANDOMIZED-QUICKSORT( $A, p, q - 1$ )  
4     RANDOMIZED-QUICKSORT( $A, q + 1, r$ )
```

# 快速排序分析

快排性能的严谨分析



# 关键操作：比较

QUICKSORT( $A, p, r$ )

```

1  if  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      QUICKSORT( $A, p, q - 1$ )
4      QUICKSORT( $A, q + 1, r$ )

```

PARTITION( $A, p, r$ )

```

1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 

```

引理 7.1：当在一个包含  $n$  个元素的数组上运行 QUICKSORT 时，假设在 PARTITION 的第 4 行中所做比较的次数为  $X$ ，那么 QUICKSORT 的运行时间为  $O(n + X)$ 。

证明：最多调用  $n$  次 PARTITION，固定工作量  $O(n)$ ；for 循环体执行  $X$  次，总代价为  $O(n + X)$ 。

- 接下来，讨论  $X$  的期望值即可。

# 平均情况下的比较次数

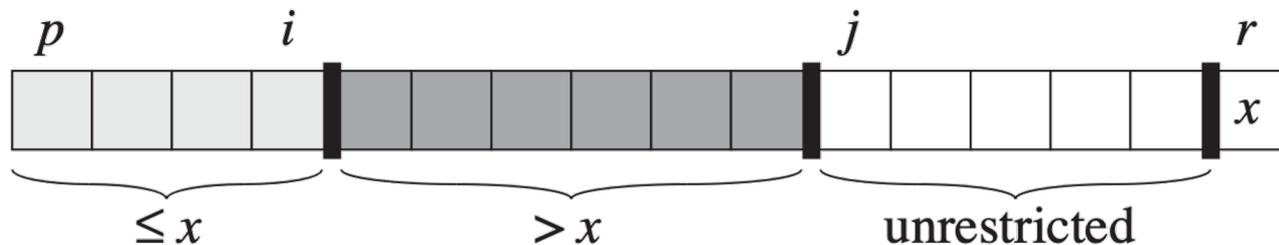
- 我们不关心每一次 PARTITION 中的比较次数，只关心总的比较次数  $X$ 。
- 定义  $z_i$  是数组  $A$  中第  $i$  小的元素， $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ ，记  $z_i$  和  $z_j$  进行比较为事件  $H_{ij}$ ， $X_{ij} = I\{H_{ij}\}$  则有：

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}$$

- 因为每次 PARTITION 元素只和主元比较且主元在本次 PARTITION 结束后再也不会被比较，所以，每对元素最多被比较一次。
- 下面我们只需要求出  $\Pr\{H_{ij}\}$  即可，因为

$$E[X] = E \left[ \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr\{H_{ij}\}$$

# $z_i$ 和 $z_j$ 比较的概率



- 假设每次PARTITION的主元是随机选取的，且每个元素都不相同。考虑两个元素何时不会进行比较。
- 若存在满足  $z_i < x < z_j$  的主元  $x$ ，则  $z_i$  和  $z_j$  永远都不会比较。
- 如果要  $z_i$  和  $z_j$  比较，要么让  $z_i$  在  $Z_{ij}$  中其他元素之前被选为主元，要么让  $z_j$  在  $Z_{ij}$  中其他元素之前被选为主元，两者的概率都是  $\frac{1}{j-i+1}$ 。

• 于是

$$\Pr\{H_{ij}\} = \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

# 平均情况下的比较次数

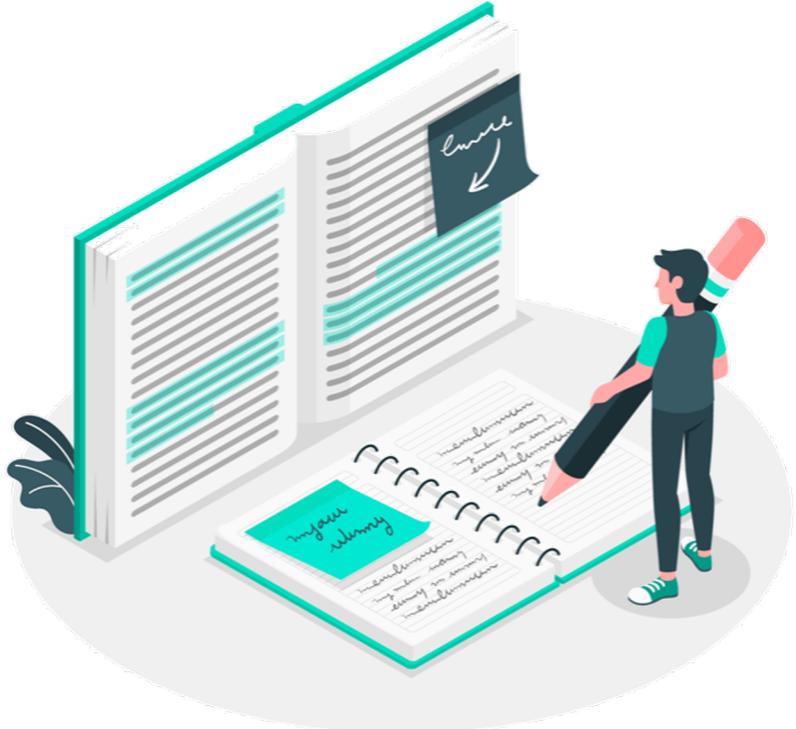
$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr\{H_{ij}\} = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$= 2 \sum_{k=1}^{n-1} \frac{n-k}{k+1} = 2 \sum_{k=1}^{n-1} \frac{n-k-1+1}{k+1} = 2 \sum_{k=1}^{n-1} \left( \frac{n+1}{k+1} - 1 \right)$$

$$= 2(n+1) \left( \sum_{k=1}^{n-1} \frac{1}{k+1} \right) - 2(n-1) = 2(n+1)\Theta(\log n) - 2(n-1) = \Theta(n \log n)$$

- 综上，使用 **RANDOMIZED-PARTITION**，在输入元素互异的情况下，快速排序算法的期望运行时间为  $\Theta(n \log n)$ 。
- 有元素相同的时候，期望运行时间也不会超过  $\Theta(n \log n)$ 。（详见 [作业6-1问题2](#)）。

# 本讲小结



# 内容提要

- 快速排序的描述
  - 难分易合型的分治算法
  - 重要的划分子过程
- 快速排序的性能
  - 最坏情况运行时间： $\Theta(n^2)$
  - 平均情况运行时间： $\Theta(n \log n)$  且其中的常数系数很小
  - 原址排序
  - 实际应用中最好的选择
- 快速排序的随机化版本
  - 分析使用随机划分的快速排序的期望时间

The End!

