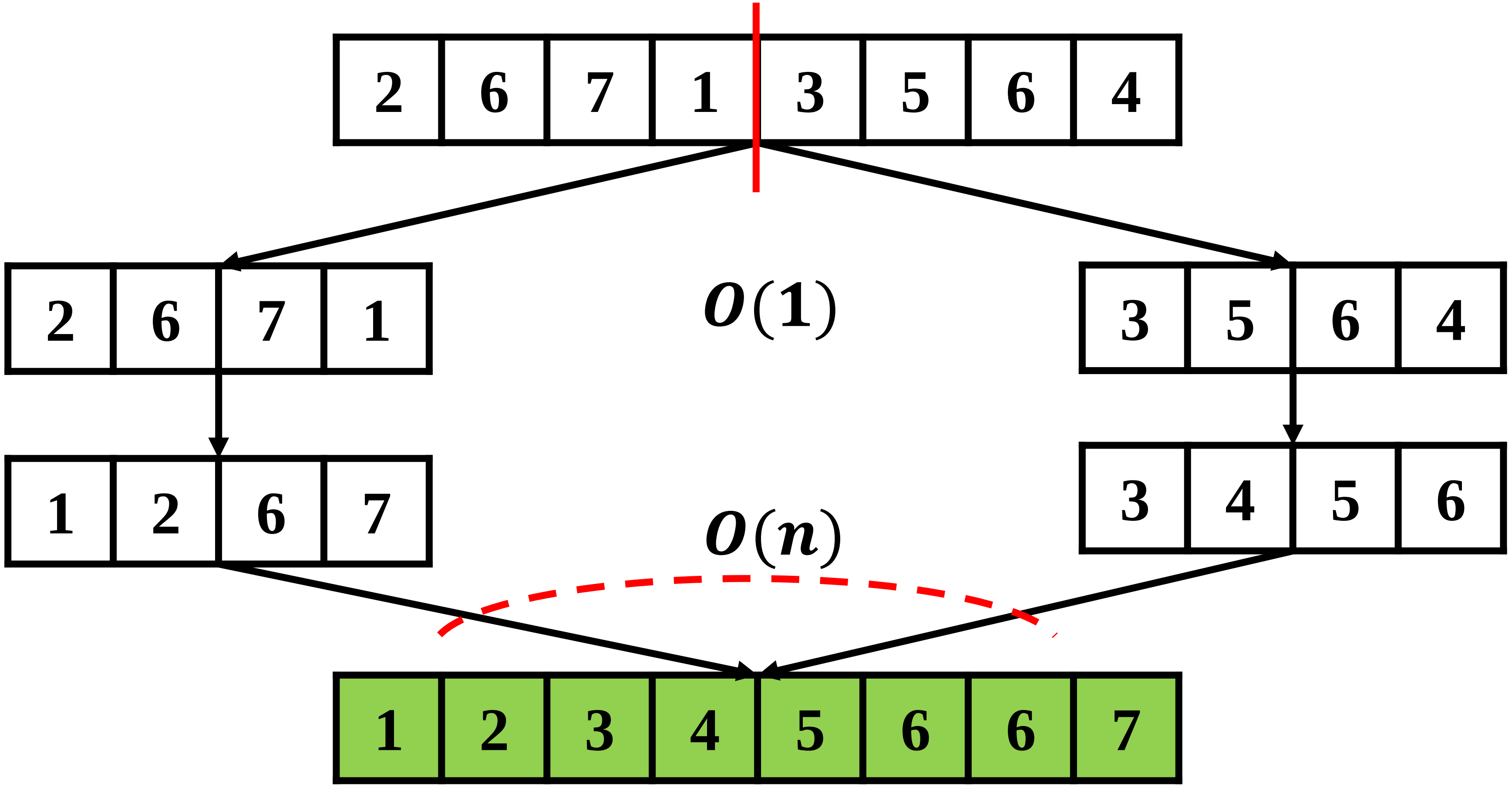
# Design and Analysis of Algorithms

# Lecture 7: Quicksort

童咏昕

北京航空航天大学
计算机学院

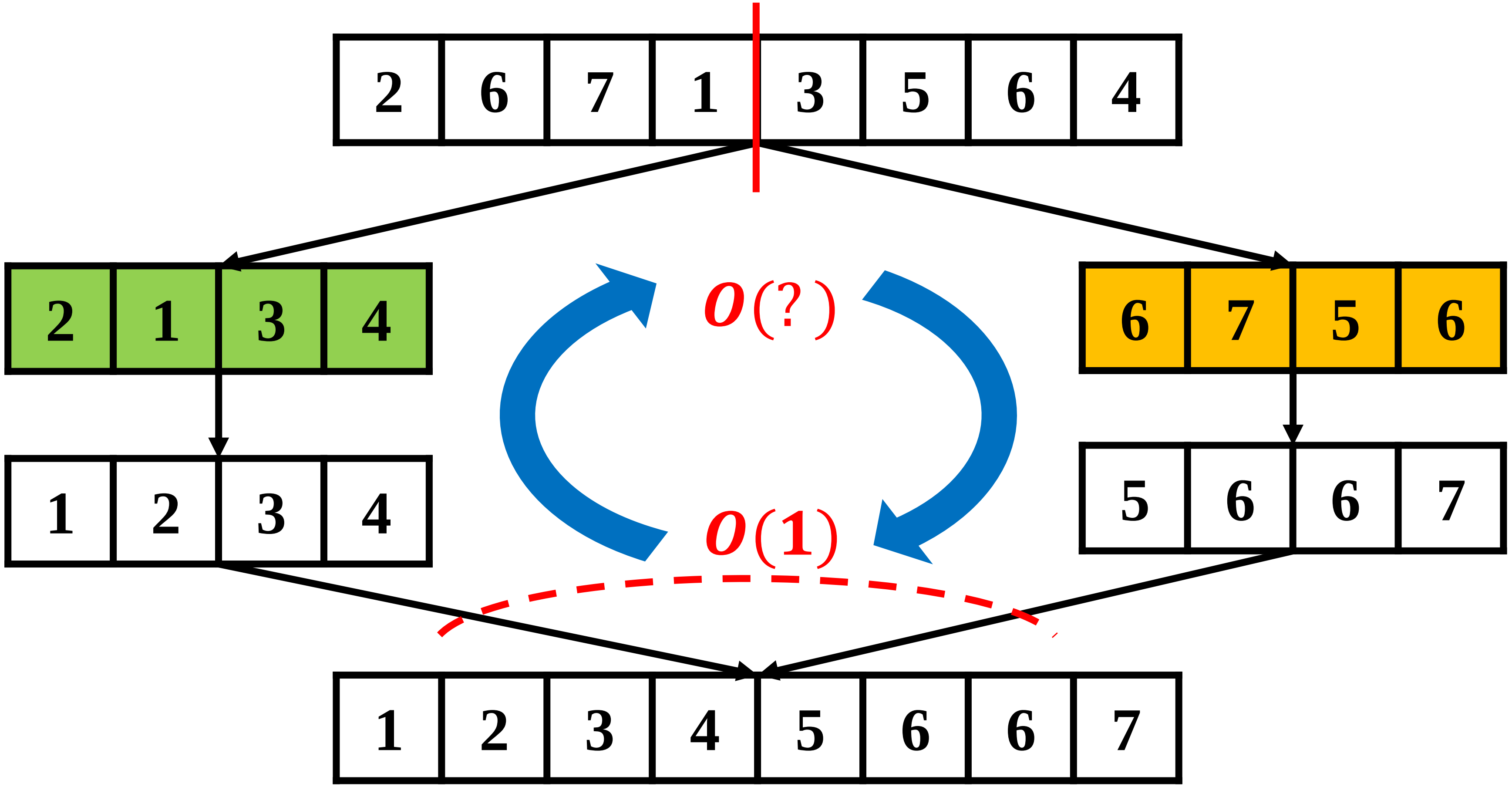
# 算法回顾：归并排序




分而治之框架


归并排序：简化分解，侧重合并

# 从归并排序到快速排序




分而治之框架

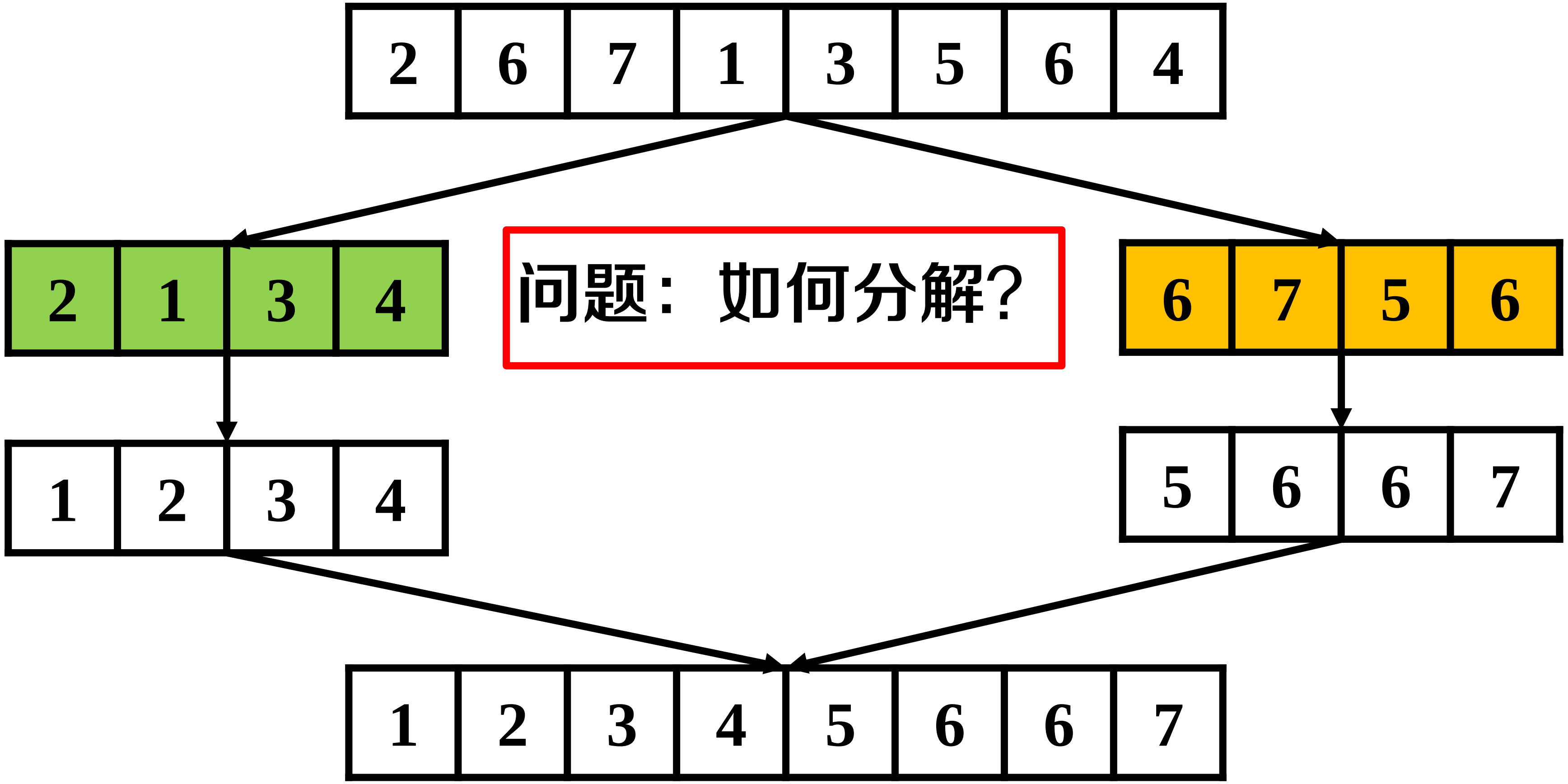
分解原问题

解决子问题

合并问题解

**快速排序：侧重分解，简化合并**

# 从归并排序到快速排序




分而治之框架

分解原问题

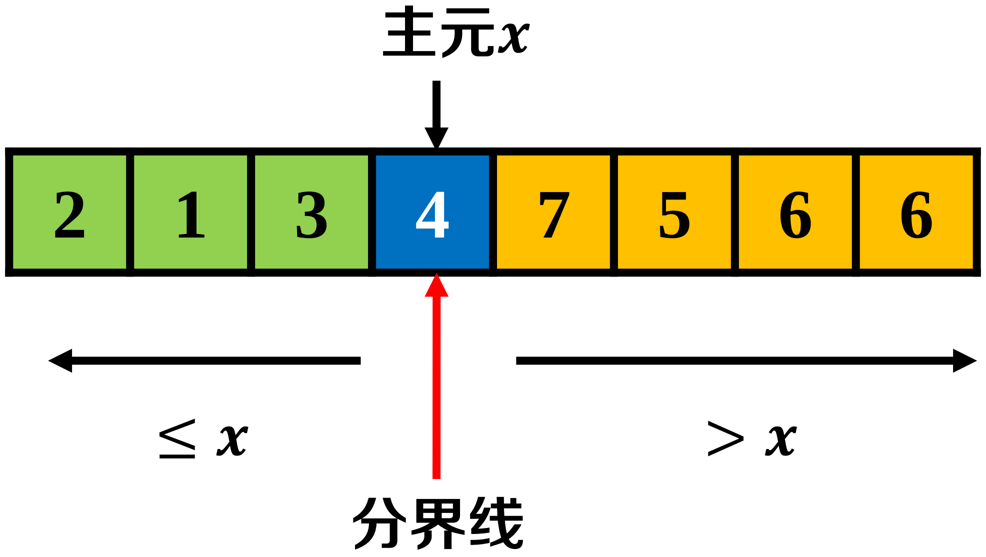
解决子问题

合并问题解

**快速排序：侧重分解，简化合并**

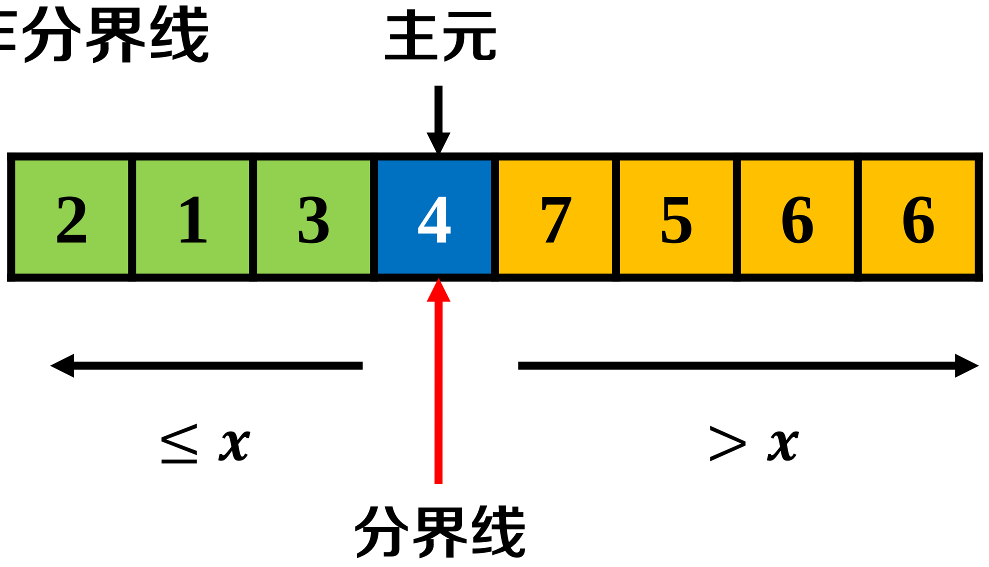
# 数组划分

- 基本思想
  - 任选元素$x$作为分界线，称为主元(pivot)
  - 交换重排，满足$x$左侧元素小于右侧



# 数组划分

- 实现方法
  - 选取固定位置主元$x$（如尾元素）
  - 维护两个部分的右端点变量$i, j$
  - 考察数组元素$A[j]$，只和主元比较
    - 若$A[j] \leq x$，则交换$A[j]$和$A[i+1]$，$i, j$右移
    - 若$A[j] > x$，则$j$右移
  - 把主元放在中间作分界线



# 数组划分：复杂度分析

- Partition$(A, p, r)$

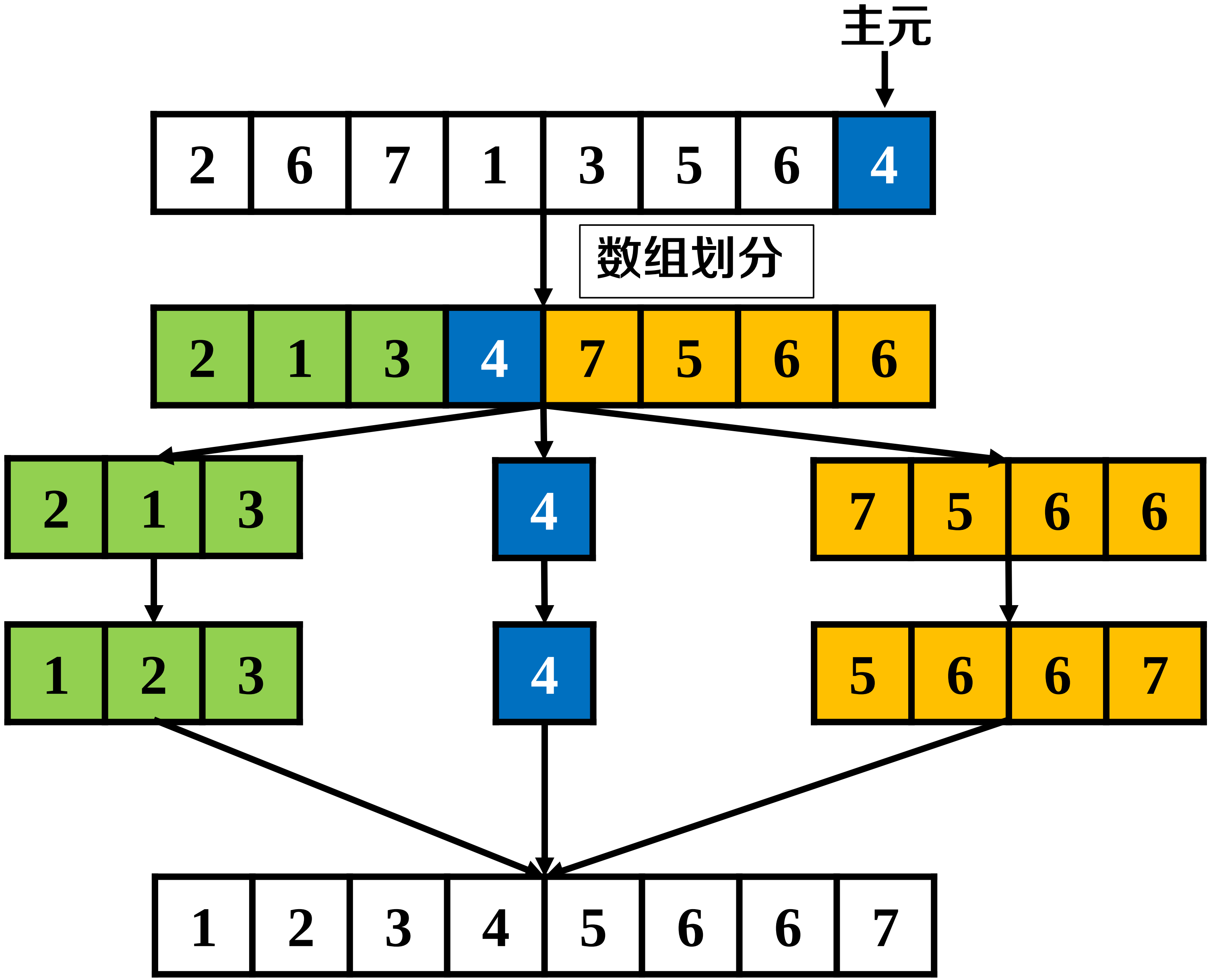
**输入: 数组$A$, 起始位置$p$, 终止位置$r$**
**输出: 划分位置$q$**

$x \leftarrow A[r]$
$i \leftarrow p - 1$
**for** $j \leftarrow p$ *to* $r - 1$ **do**
  **if** $A[j] \le x$ **then**
    exchange $A[i+1]$ with $A[j]$
    $i \leftarrow i + 1$
  **end**
**end**
exchange $A[i+1]$ with $A[r]$
$q \leftarrow i + 1$
**return** $q$

$O(n)$

时间复杂度：$O(n)$

# 快速排序：算法框架

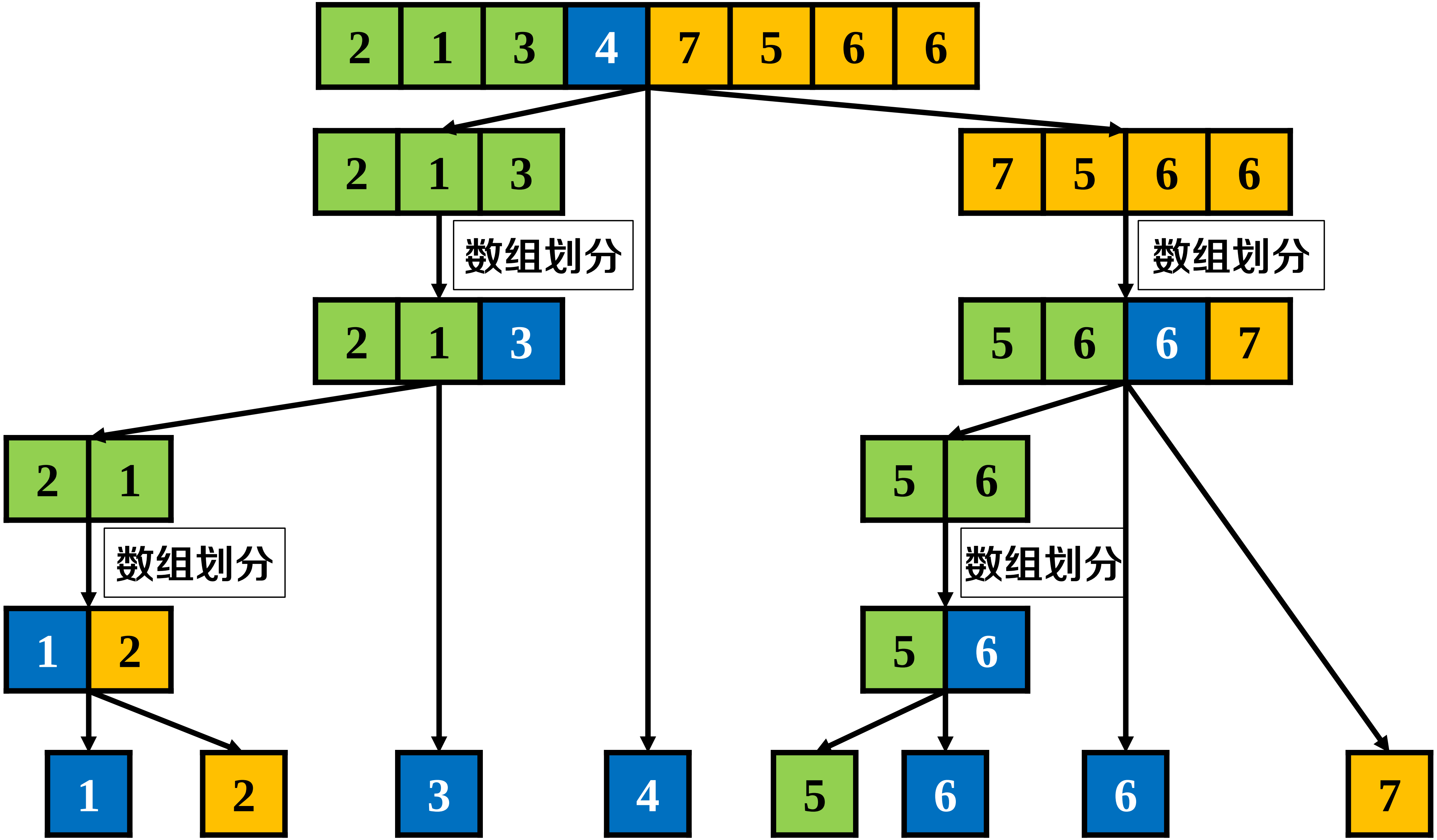



分解原问题

解决子问题

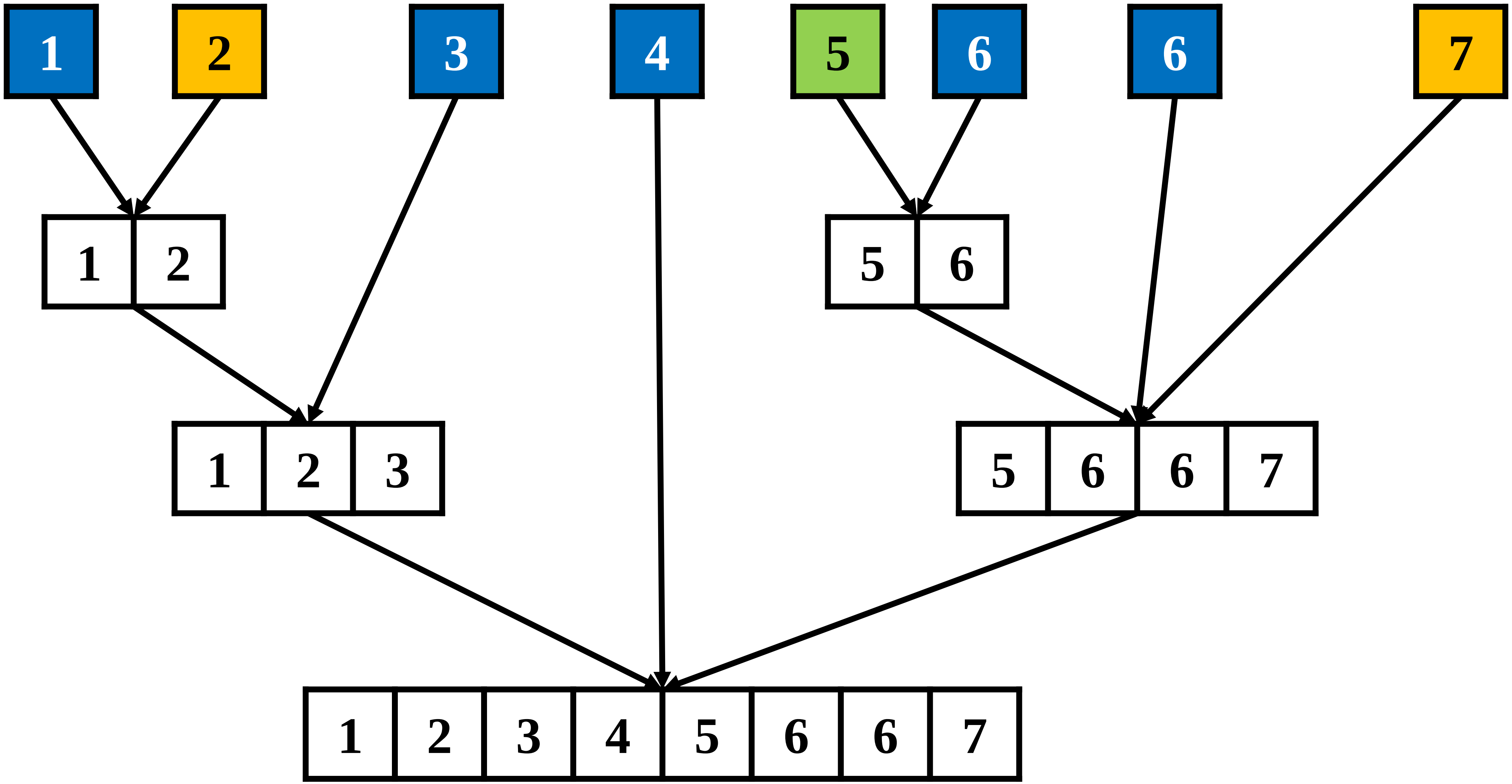
合并问题解

# 算法实例



2 1 3 4 7 5 6 6
2 1 3
7 5 6 6
数组划分
数组划分
2 1 3
5 6 6 7
2 1
5 6
数组划分
数组划分
1 2
5 6
1
2
3
4
5
6
6
7

# 算法实例



# 快速排序：复杂度分析

- **QuickSort($A, p, r$)**

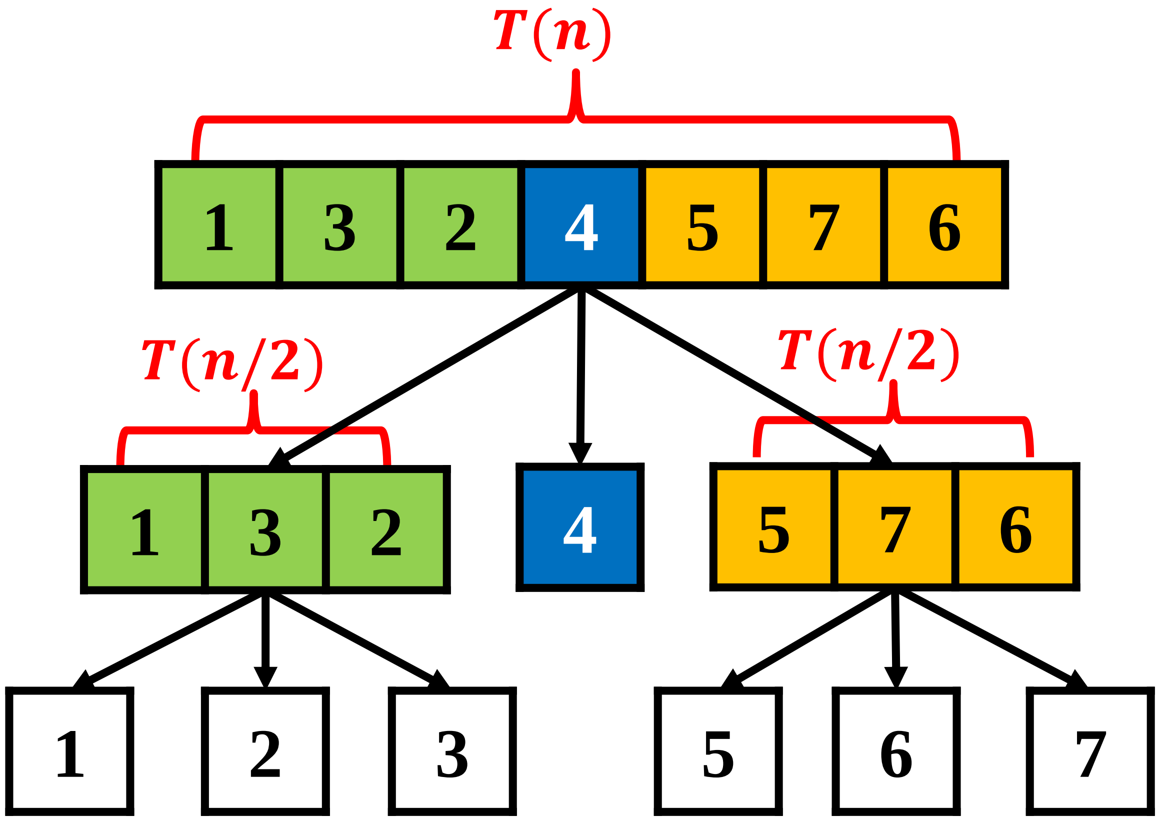
**初始调用： QuickSort($A, 1, n$)**

**输入:** 数组$A$, 起始位置$p$, 终止位置$r$
**输出:** 有序数组$A$
**if** $p < r$ **then**
  $q \leftarrow \text{Partition}(A, p, r)$ — $O(n)$
  $\text{QuickSort}(A, p, q-1)$
  $\text{QuickSort}(A, q+1, r)$ — ?
**end**

问题：子问题规模不确定，如何分析时间复杂度?

# 快速排序：复杂度分析

- 最好情况
  - 数组划分后，每次主元都在中间

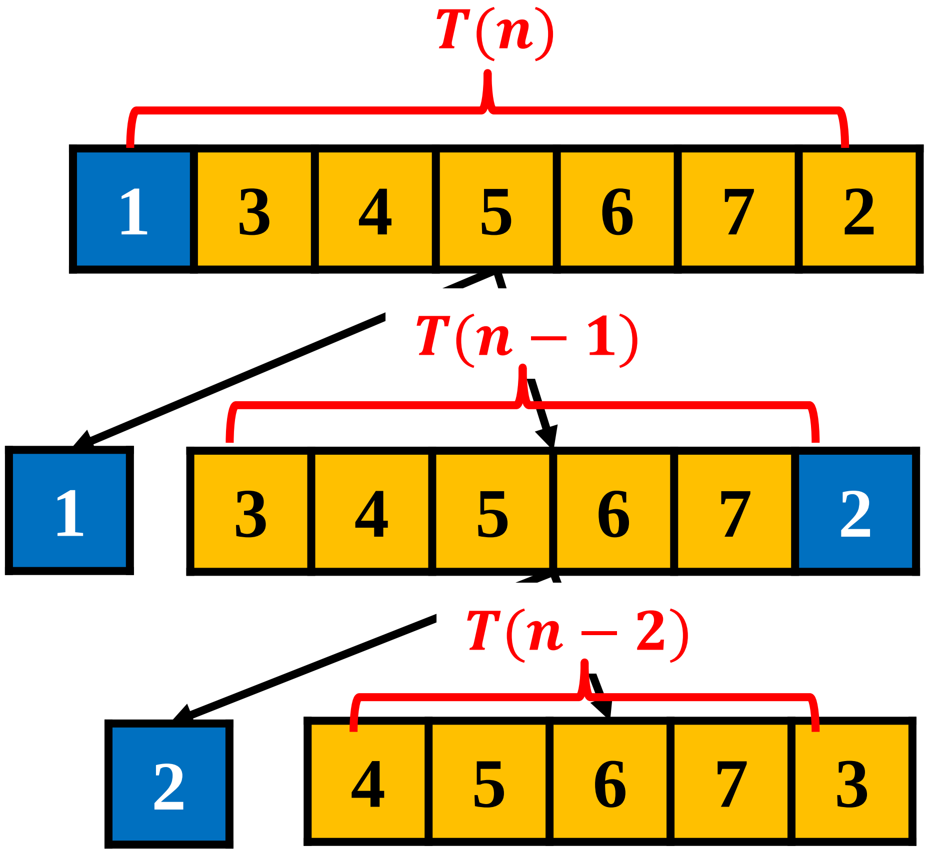



输入: 数组$A$, 起始位置$p$, 终止位置$r$
输出: 有序数组$A$
**if** $p < r$ **then**
  $q \leftarrow \text{Partition}(A, p, r)$ ---- $O(n)$
  $\text{QuickSort}(A, p, q-1)$ ---- $T(n/2)$
  $\text{QuickSort}(A, q+1, r)$ ---- $T(n/2)$
**end**

$$T(n) = 2T(n/2) + O(n) = O(n\log n)$$

# 快速排序：复杂度分析

- 最坏情况
  - 数组划分后，每次主元都在一侧




…

输入: 数组$A$, 起始位置$p$, 终止位置$r$
输出: 有序数组$A$
if $p < r$ then
  $q \leftarrow \text{Partition}(A, p, r)$ - - $O(n)$
  $\text{QuickSort}(A, p, q-1)$ - - $T(0)$
  $\text{QuickSort}(A, q+1, r)$ - - $T(n-1)$
end

$$T(n) = T(n-1) + T(0) + O(n) = O(n^2)$$

# 快速排序：复杂度分析

- $\mathbf{QuickSort}(\boldsymbol{A}, \boldsymbol{p}, \boldsymbol{r})$

初始调用：$\mathbf{QuickSort}(\boldsymbol{A}, \mathbf{1}, \boldsymbol{n})$

**输入:** 数组$A$, 起始位置$p$, 终止位置$r$
**输出:** 有序数组$A$

```
if p < r then
    q ← Partition(A, p, r)
    QuickSort(A, p, q − 1)
    QuickSort(A, q + 1, r)
end
```

- 最好情况：$O(n\log n)$
- 最坏情况：$O(n^2)$

快速排序看似不优于归并排序

# 快速排序：复杂度分析

- $\mathbf{QuickSort}(\boldsymbol{A},\boldsymbol{p},\boldsymbol{r})$

初始调用：$\mathbf{QuickSort}(\boldsymbol{A},\mathbf{1},\boldsymbol{n})$

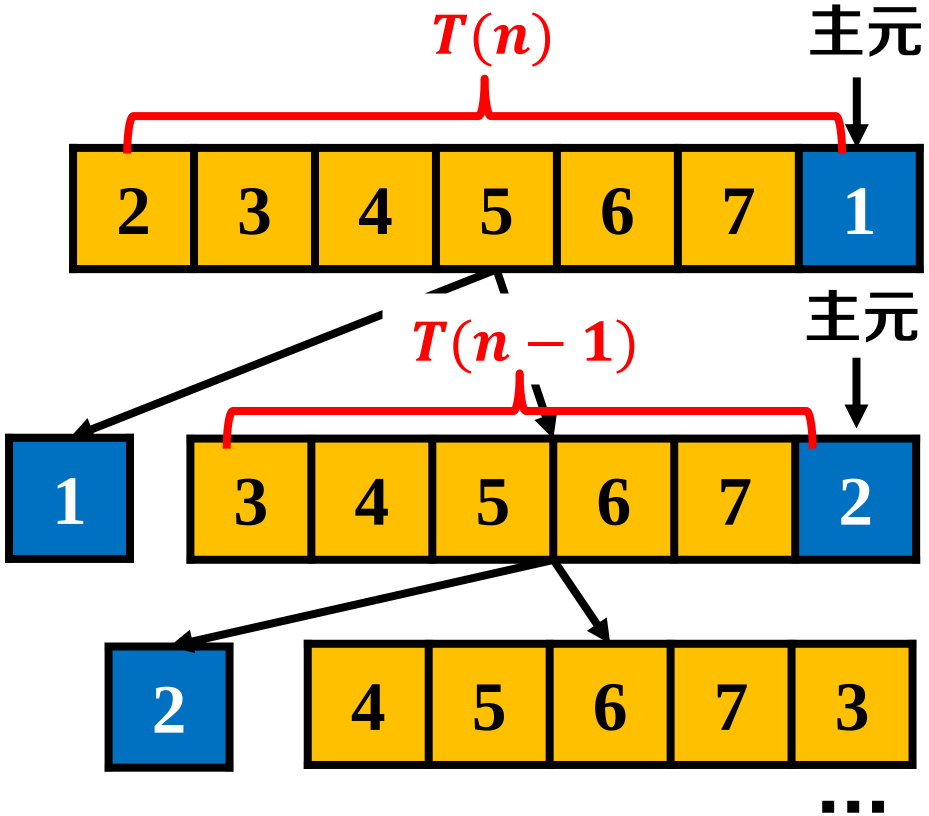
**输入:** 数组$A$, 起始位置$p$, 终止位置$r$
**输出:** 有序数组$A$
**if** $p < r$ **then**
  $q \leftarrow \text{Partition}(A, p, r)$
  $\text{QuickSort}(A, p, q-1)$
  $\text{QuickSort}(A, q+1, r)$
**end**

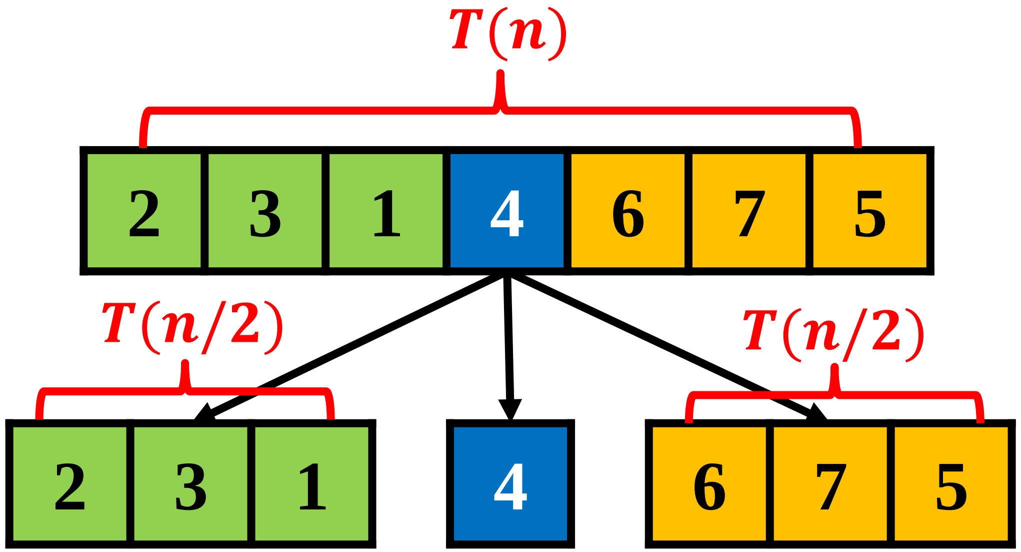
- 最好情况：$\boldsymbol{O}(\boldsymbol{n}\log\boldsymbol{n})$
- 最坏情况：$\boldsymbol{O}(\boldsymbol{n}^2)$

问题：如何摆脱输入导致最坏情况的困境?

# 随机划分

- 反思最差情况
  - 数组划分时选取**固定**位置主元，可以针对性构造最差情况
- 解决方案
  - 数组划分时选取**随机**位置主元，无法针对性构造最差情况



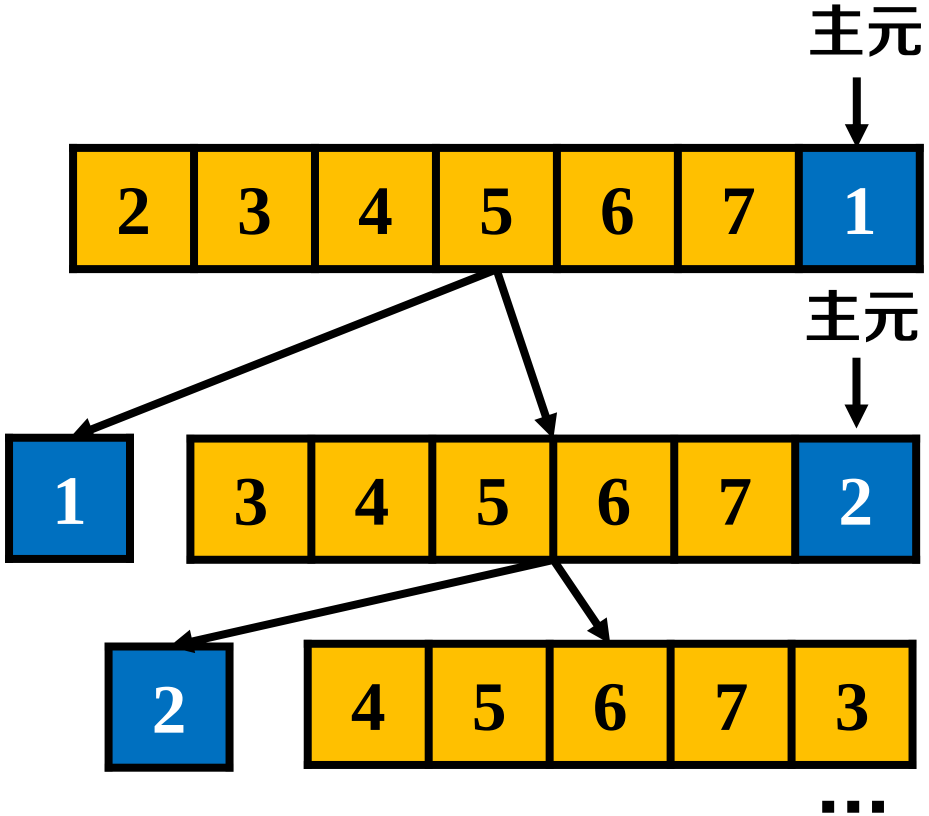
# 随机划分：伪代码

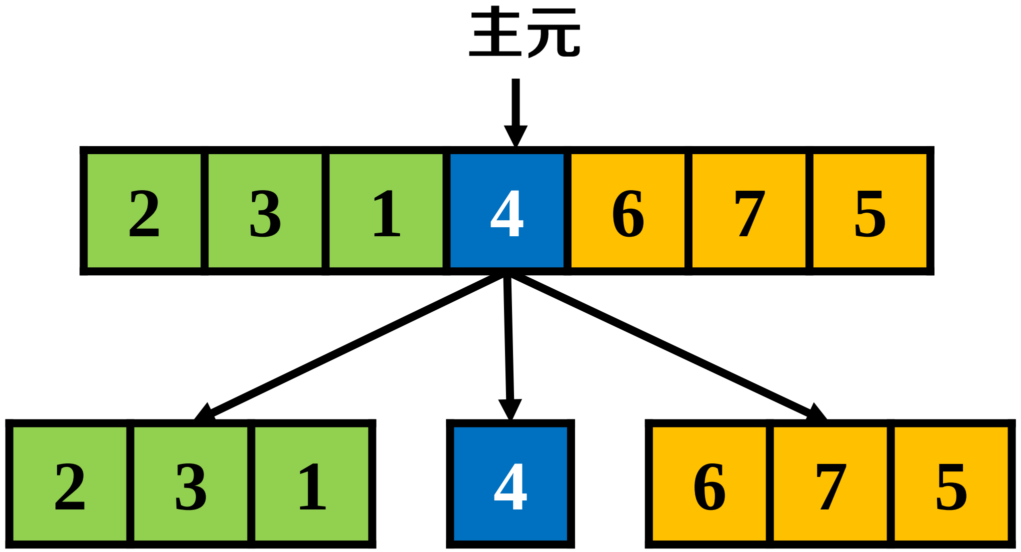
- **Randomized-Partition($A, p, r$)**

**输入:** 数组$A$, 起始位置$p$, 终止位置$r$
**输出:** 划分位置$q$

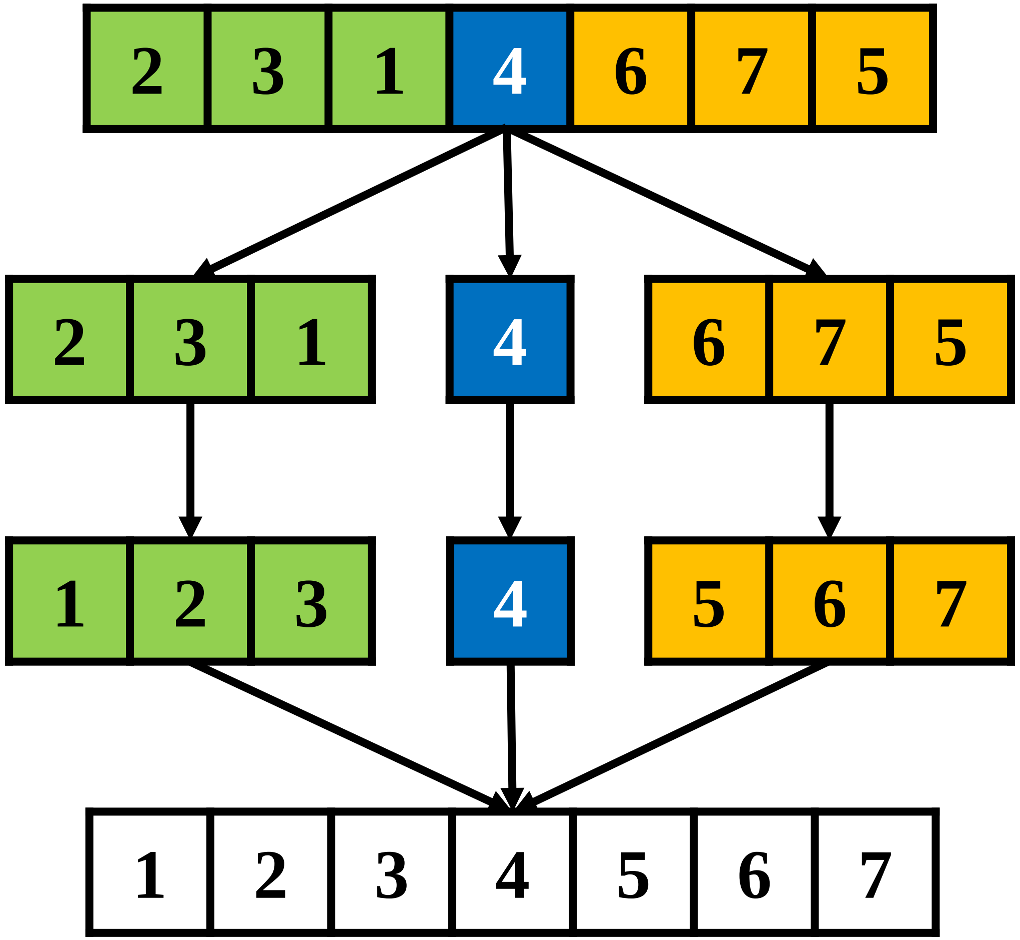
$s \leftarrow \mathrm{Random}(p, r)$
exchange $A[s]$ with $A[r]$
$q \leftarrow \mathrm{Partition}(A, p, r)$
**return** $q$

执行数组划分



# 随机化快速排序：算法框架




# 随机化快速排序：伪代码

- **Randomized-Partition($A, p, r$)**

**输入: 数组$A$, 起始位置$p$, 终止位置$r$**
**输出: 划分位置$q$**

$s \leftarrow \text{Random}(p, r)$
exchange $A[s]$ with $A[r]$
$q \leftarrow \text{Partition}(A, p, r)$
**return** $q$

- **Randomized-QuickSort($A, p, r$)**

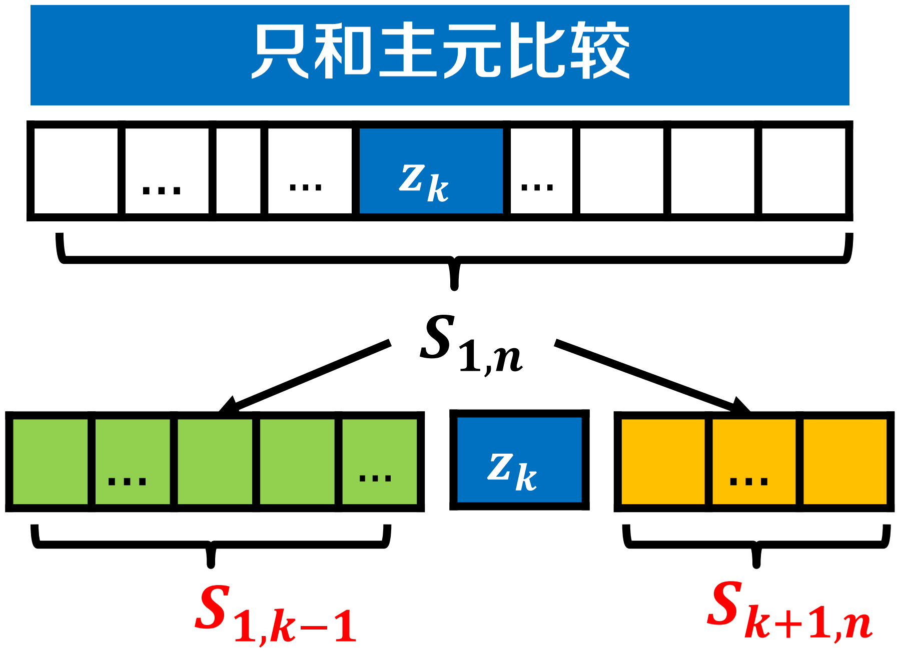
**输入: 数组$A$, 起始位置$p$, 终止位置$r$**
**输出: 有序数组$A$**

**if** $p < r$ **then**
  $q \leftarrow \text{Randomized-Partition}(A, p, r)$
  $\text{Randomized-QuickSort}(A, p, q-1)$
  $\text{Randomized-QuickSort}(A, q+1, r)$
**end**

**问题：如何分析时间复杂度?**

# 随机化快速排序：复杂度分析

- 分析目标：期望复杂度
  - 计算元素期望比较次数$E[X]$
- 符号表示
  - $z_k$：数组$A$中第$k$小的元素（假设元素互不相同）
  - 集合$S_{i,j}$：$\{z_i, \ldots, z_j\}$

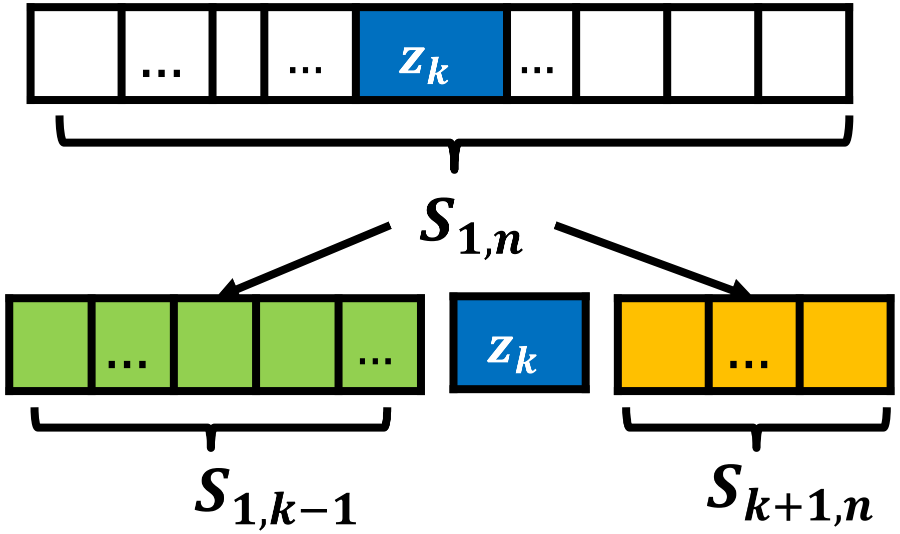


# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$

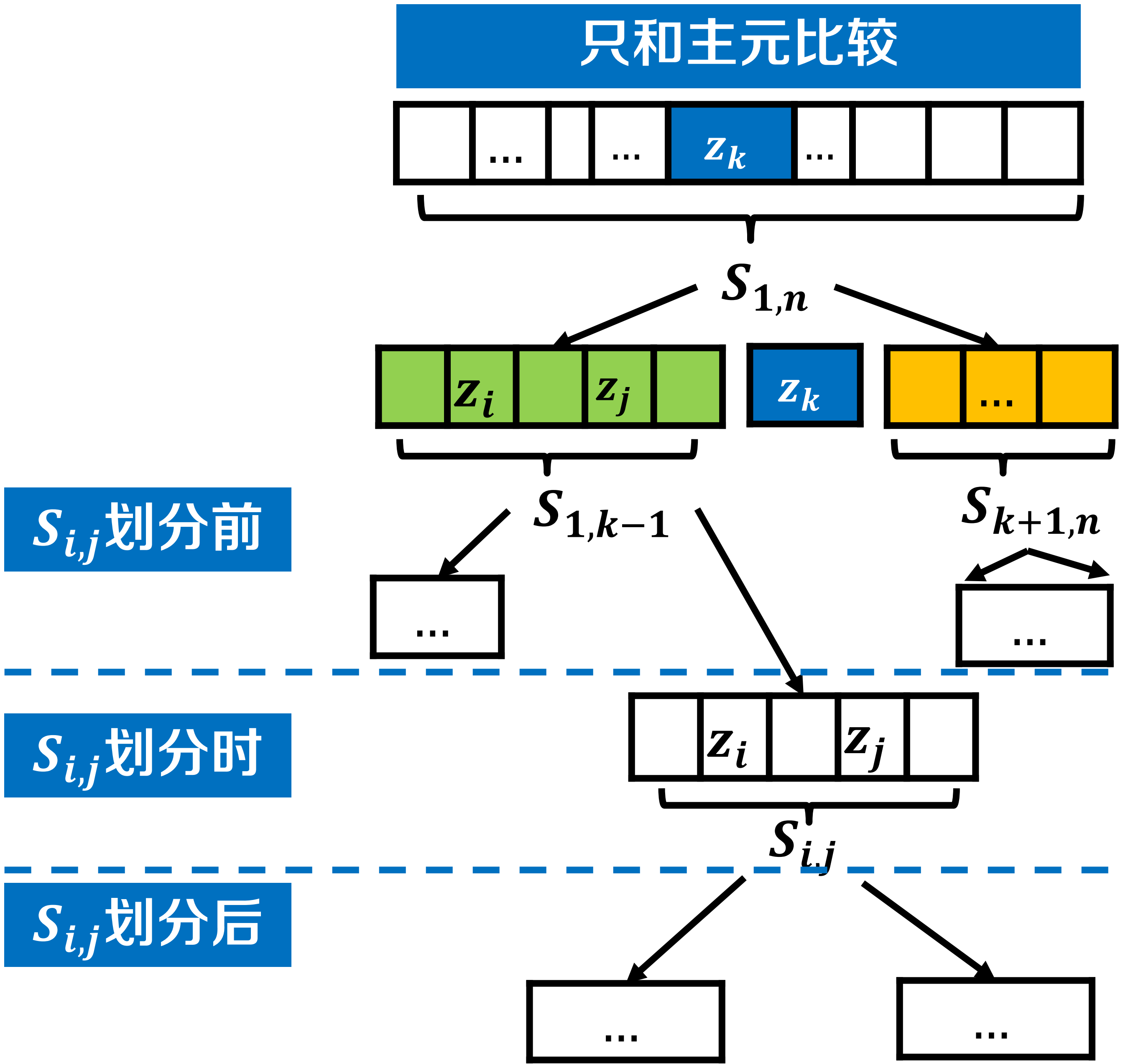
期望的线性特性

只和主元比较



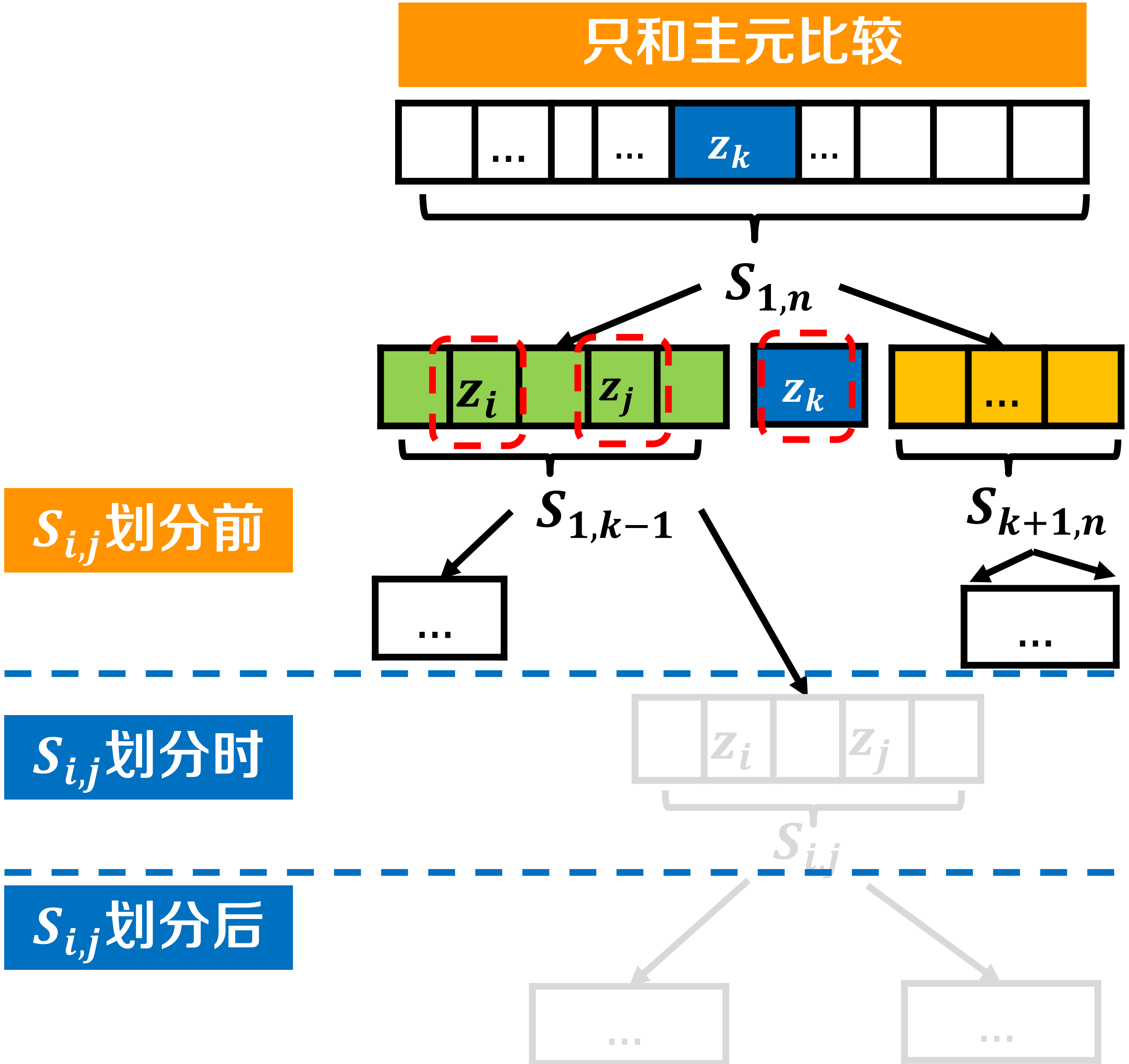
# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}E[X_{ij}]$
  - $E\left[X_{ij}\right]=?$



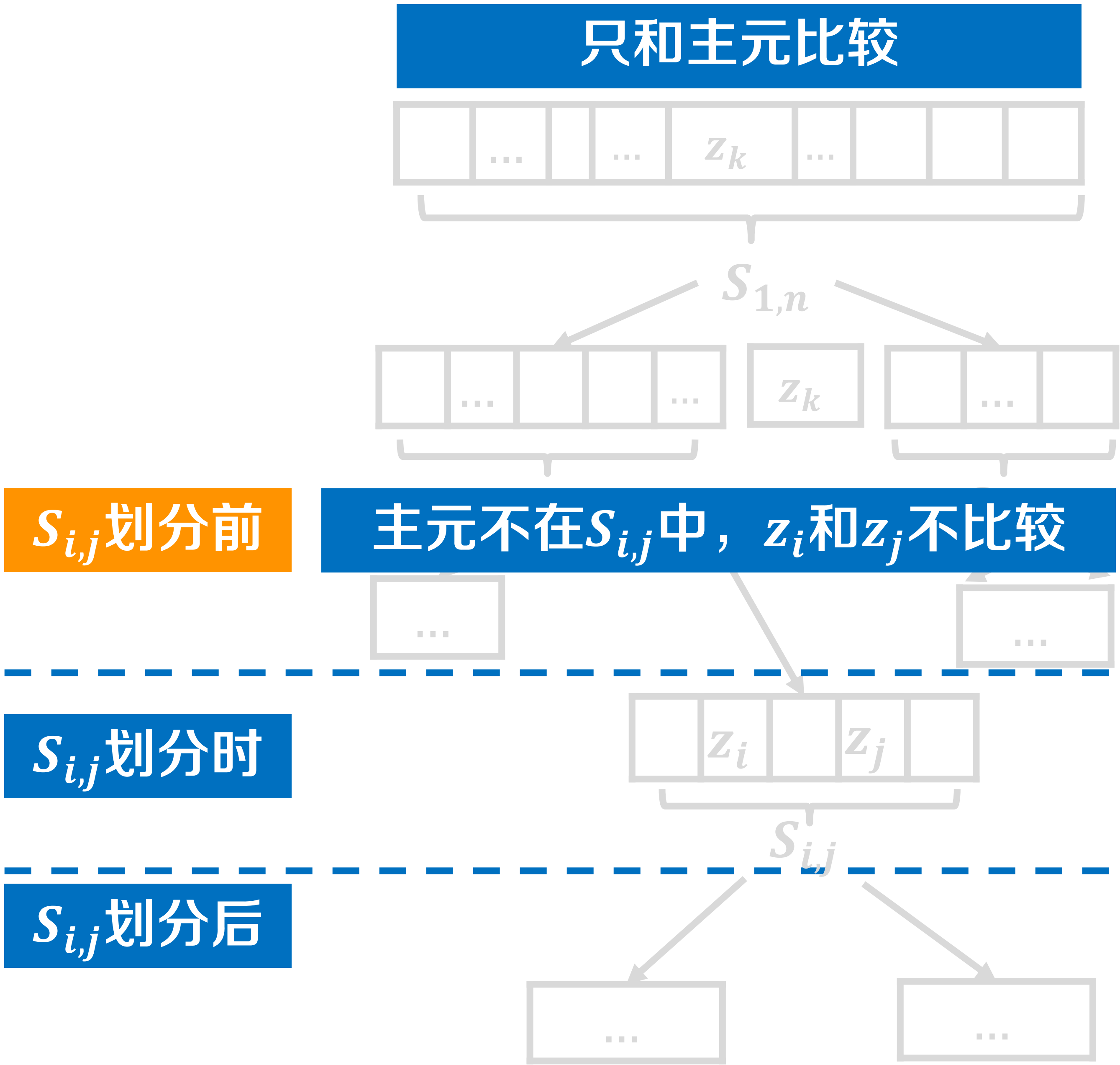
# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$
  - $E\left[X_{ij}\right] = ?$



# 随机化快速排序：复杂度分析

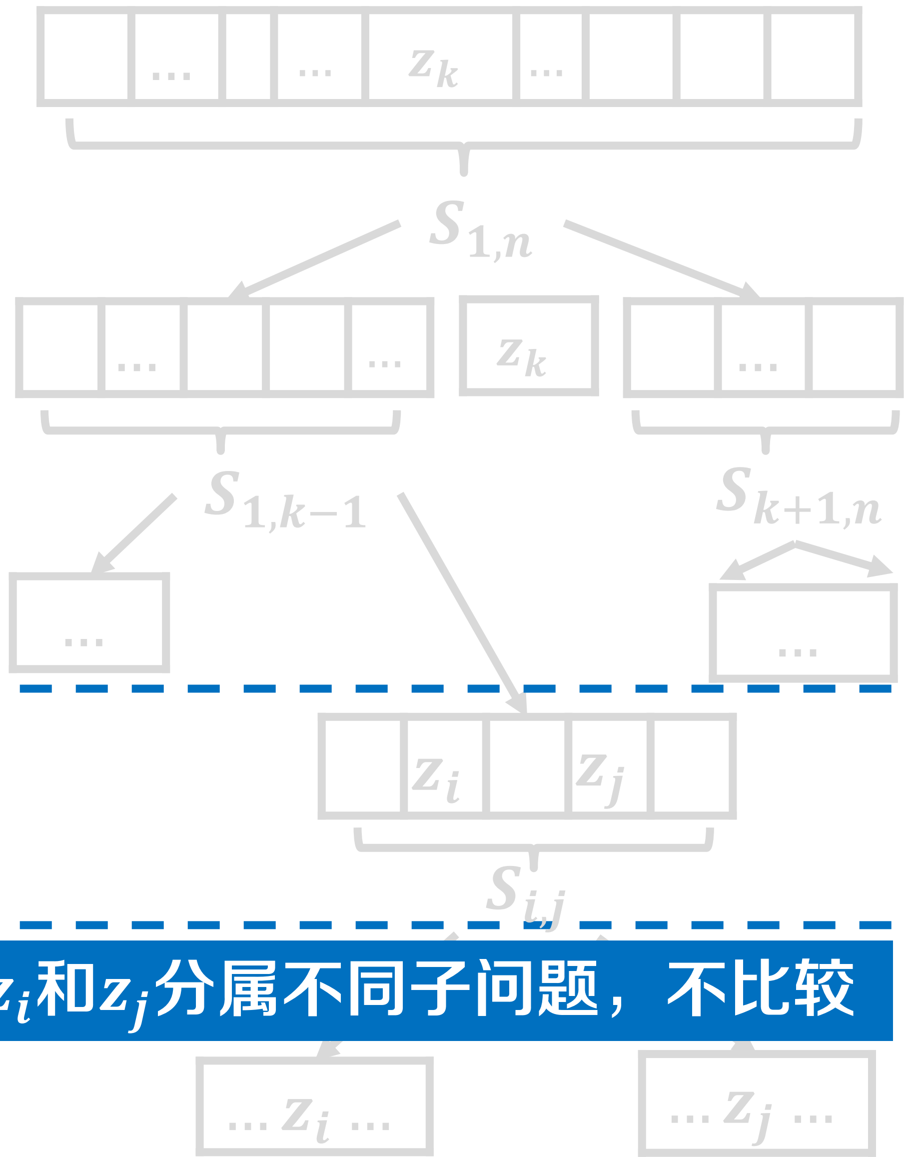
- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}E[X_{ij}]$
  - $E[X_{ij}]=?$



# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$
  - $E[X_{ij}] = ?$

只和主元比较



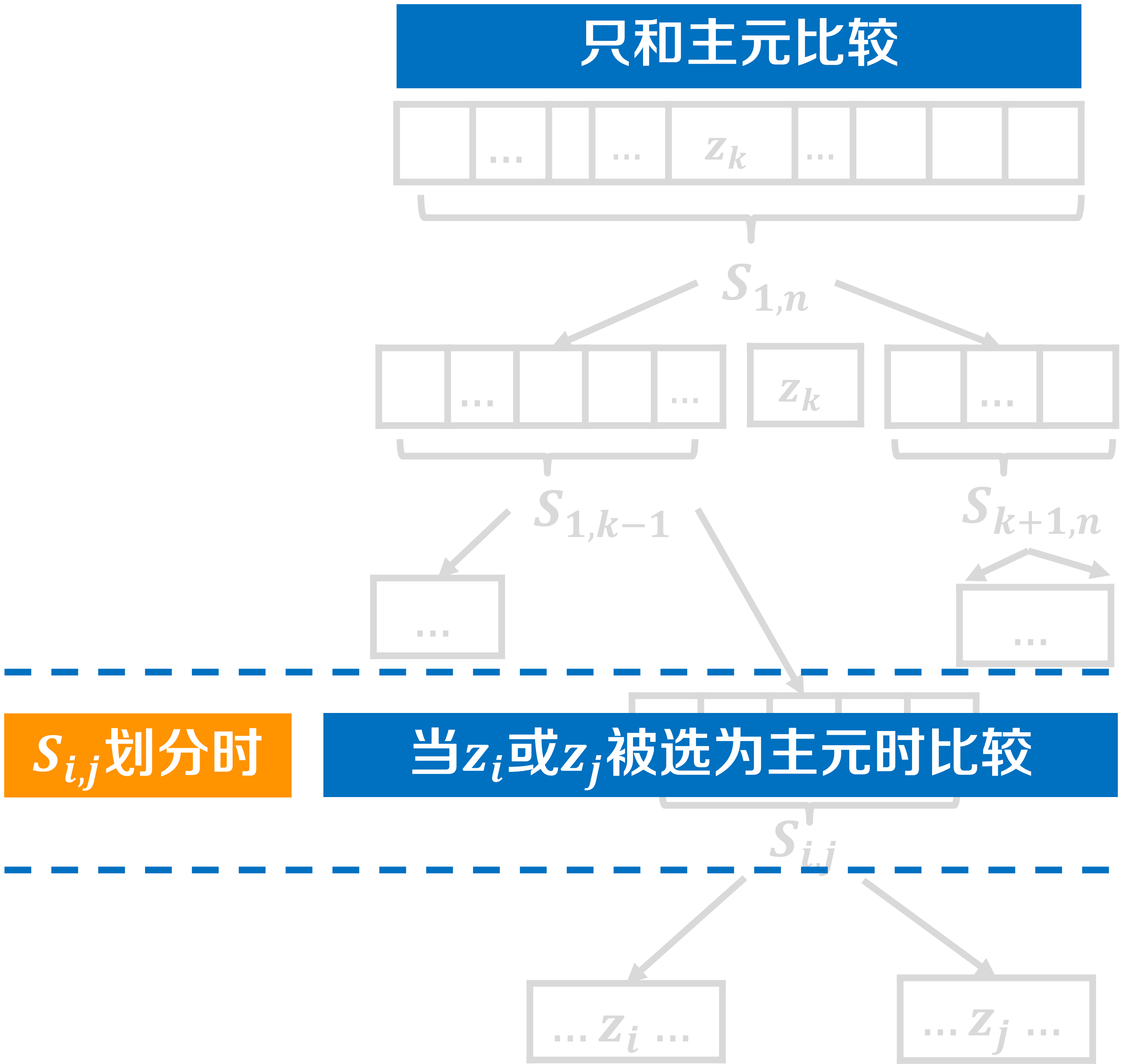

$S_{i,j}$划分时

$S_{i,j}$划分后

$z_i$和$z_j$分属不同子问题，不比较

# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$
  - $E[X_{ij}] = ?$



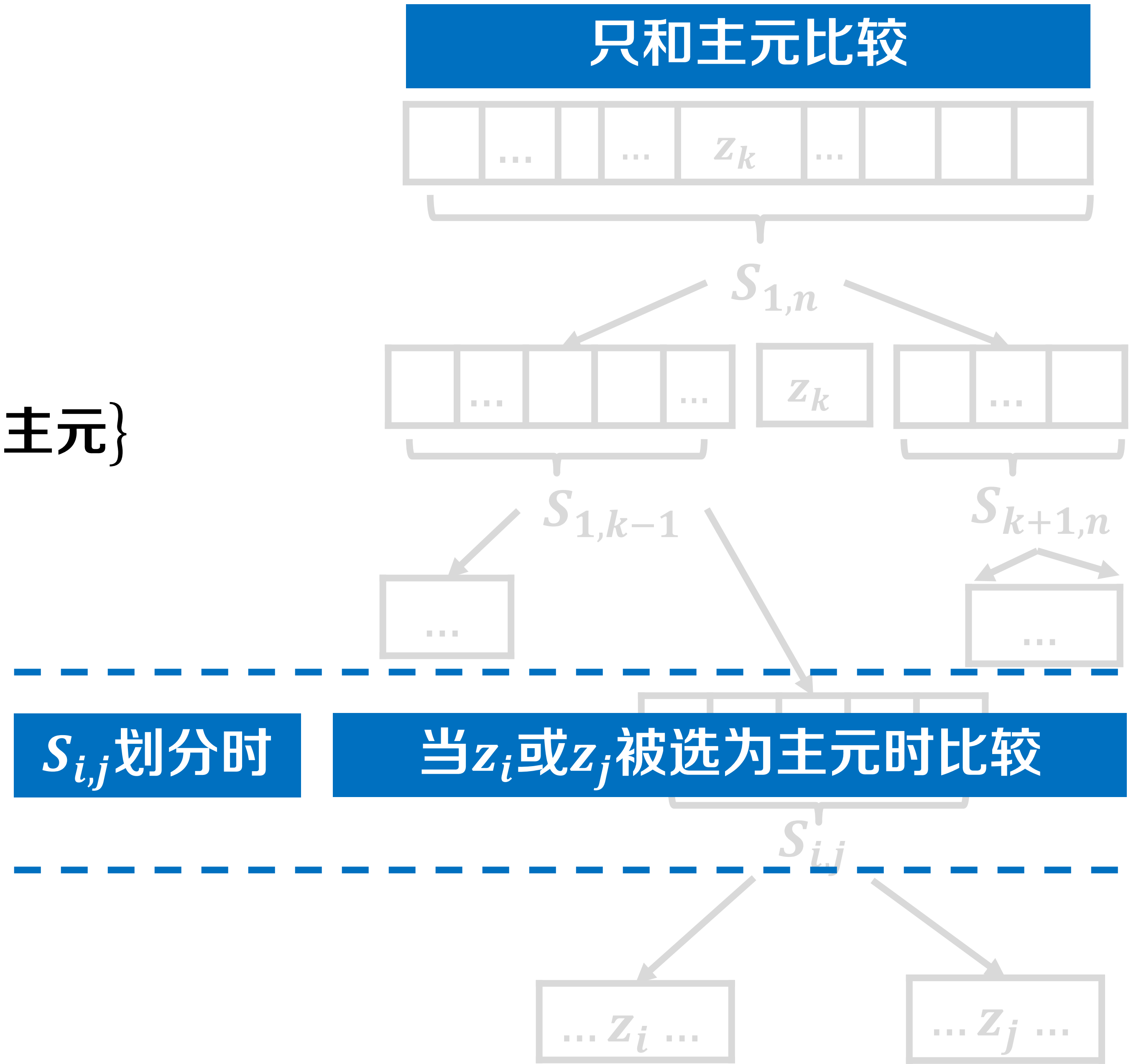
# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}E[X_{ij}]$
  - $E\left[X_{ij}\right]=\Pr\{z_i\text{或}z_j\text{被选为主元}\}$

$$=\Pr\{z_i\text{是主元}\}+\Pr\{z_j\text{是主元}\}$$

$$=\frac{1}{j-i+1}+\frac{1}{j-i+1}$$

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



# 随机化快速排序：复杂度分析

- **推导过程**
  - 随机变量$X_{ij}$：$z_i$和$z_j$比较的次数
  - $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$
  - $E[X_{ij}] = \frac{2}{j-i+1}$
  - $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$

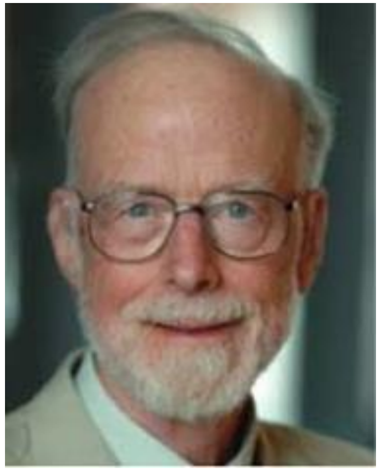
    $= \sum_{i=1}^{n-1} O(\log n) = O(n \log n)$

**期望时间复杂度：$O(n \log n)$**

# 排序算法比较

约翰·冯·诺伊曼
John von Neumann
1945年提出

| 算法名称 | 时间复杂度 |
|---|---|
| 选择排序 | $O(n^2)$ |
| 插入排序 | $O(n^2)$ |
| 归并排序 | $O(n\log n)$ |
| 快速排序 | 最差: $O(n^2)$<br>期望: $O(n\log n)$ |



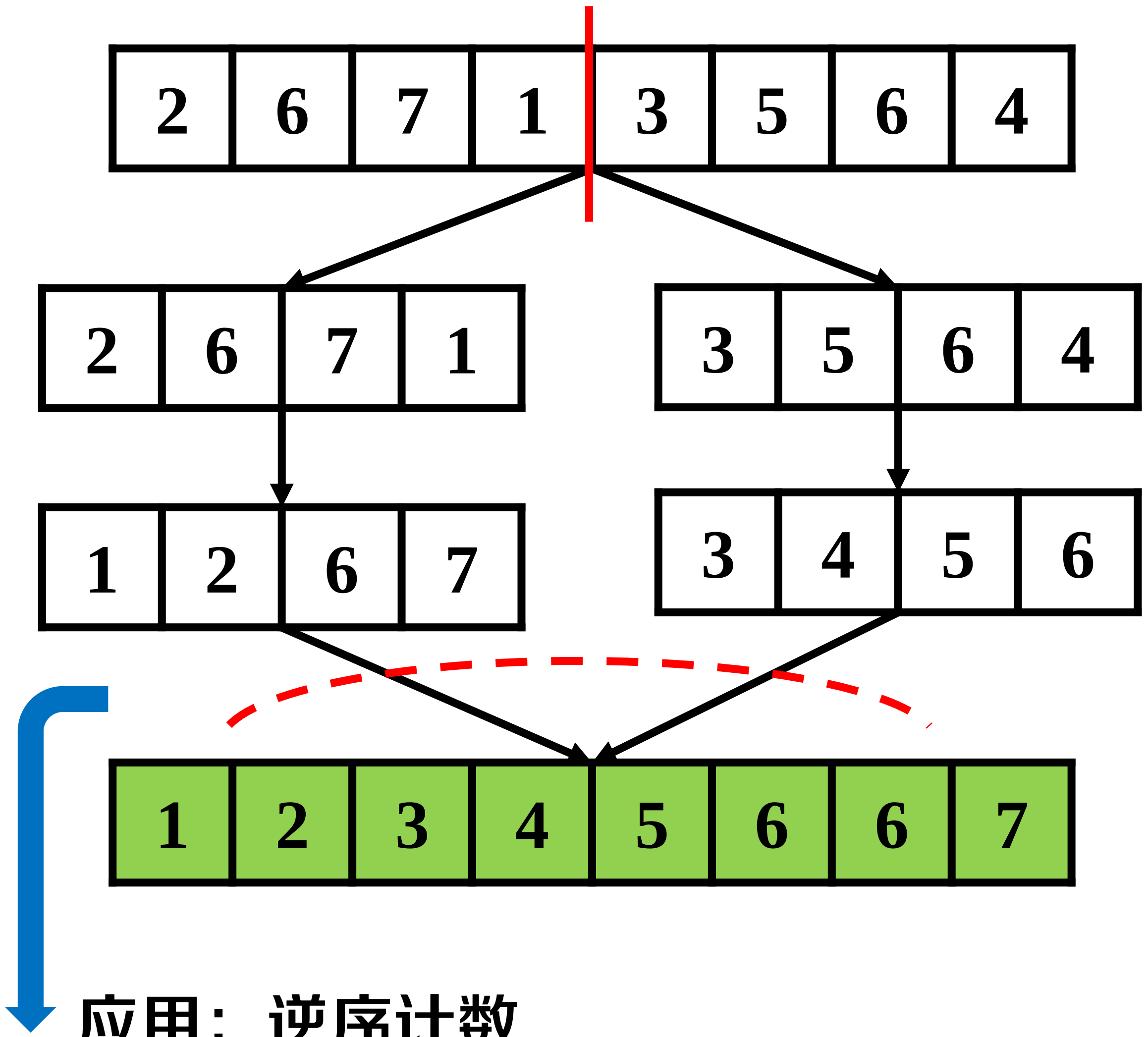
托尼·霍尔
Tony Hoare
1961年提出

问题：能否突破$O(n\log n)$?

基于比较的排序，时间复杂度下界为$\Omega(n\log n)$

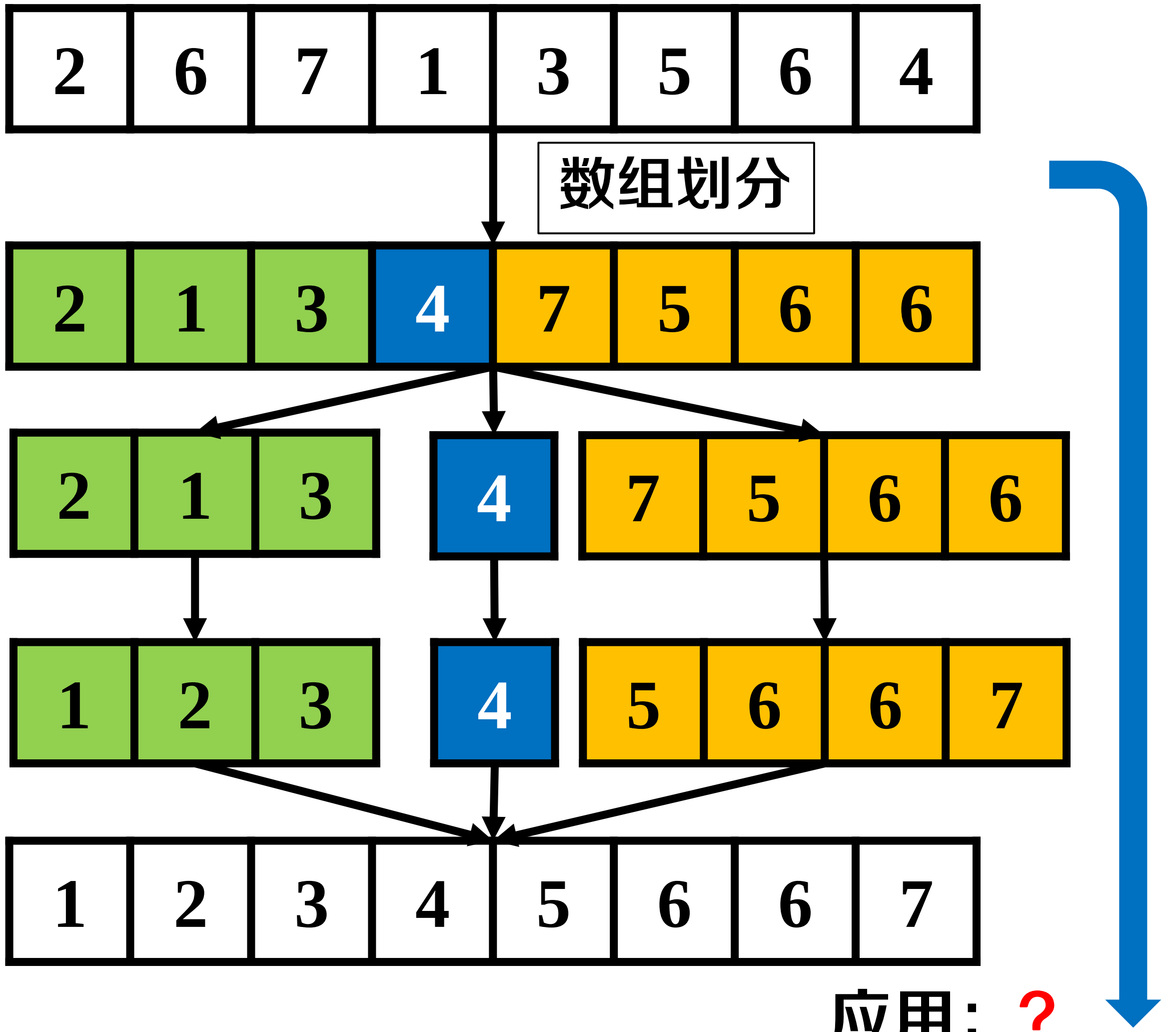
# 小结

- 归并排序




应用：逆序计数

- 快速排序




应用：？

# 谢谢

BDA

Beihang University