算法设计与分析—进阶篇

第一讲 从排序看算法设计与分析

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn
http://homepage.hit.edu.cn/pages/wang/

本讲内容

1.1 从排序看算法分析
1.2 快速排序深入剖析
1.3 问题复杂度下界
1.4 基于比较的排序算法的时间复杂度下界

排序

排序的概念

排序是计算机内经常进行的一种操作，其目的是将一组“无序”的记录序列调整为“有序”的记录序列

例如，将下列关键字序列

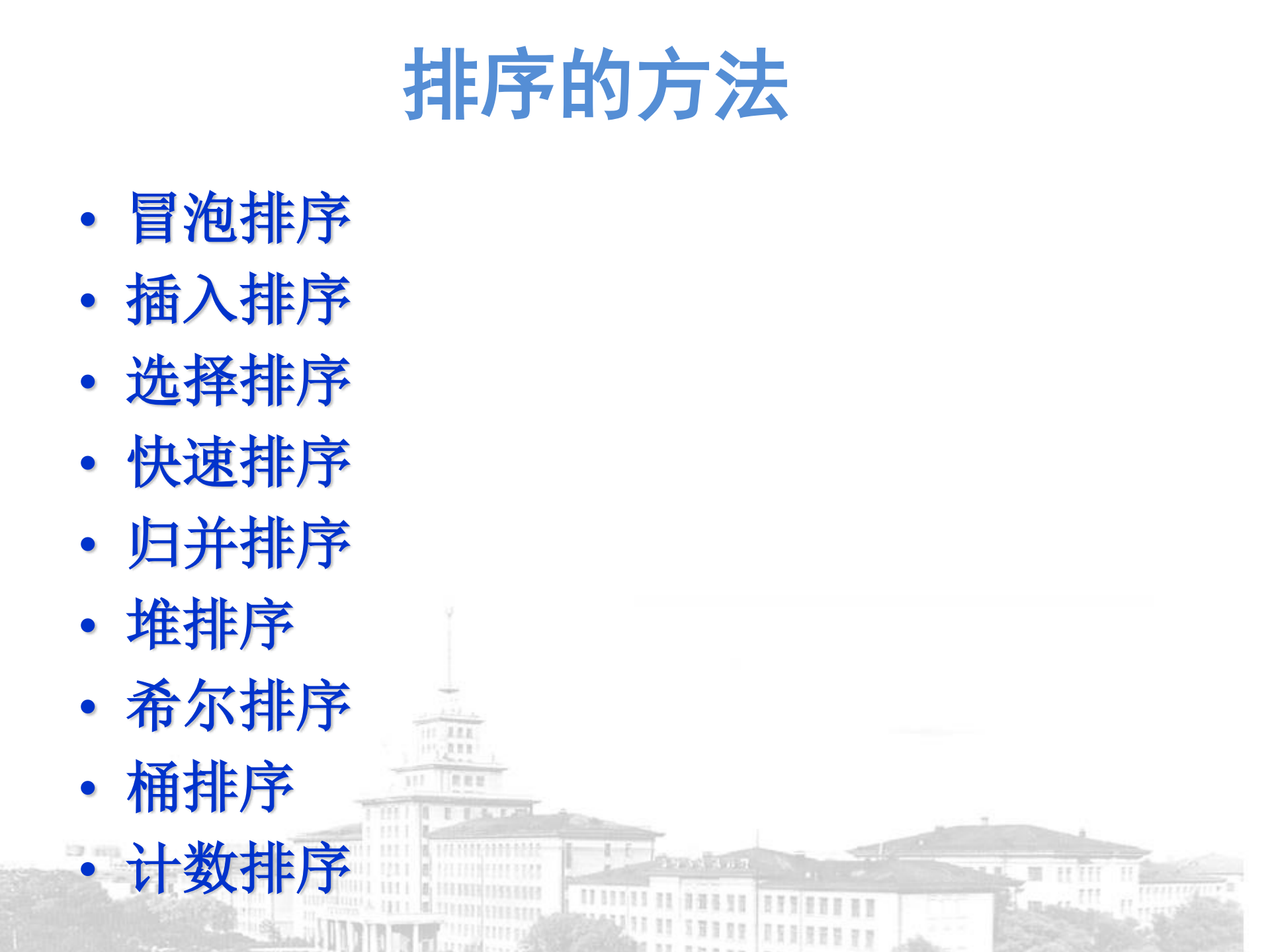
52，49，80，36，14，58，61，23，97，75

调整为

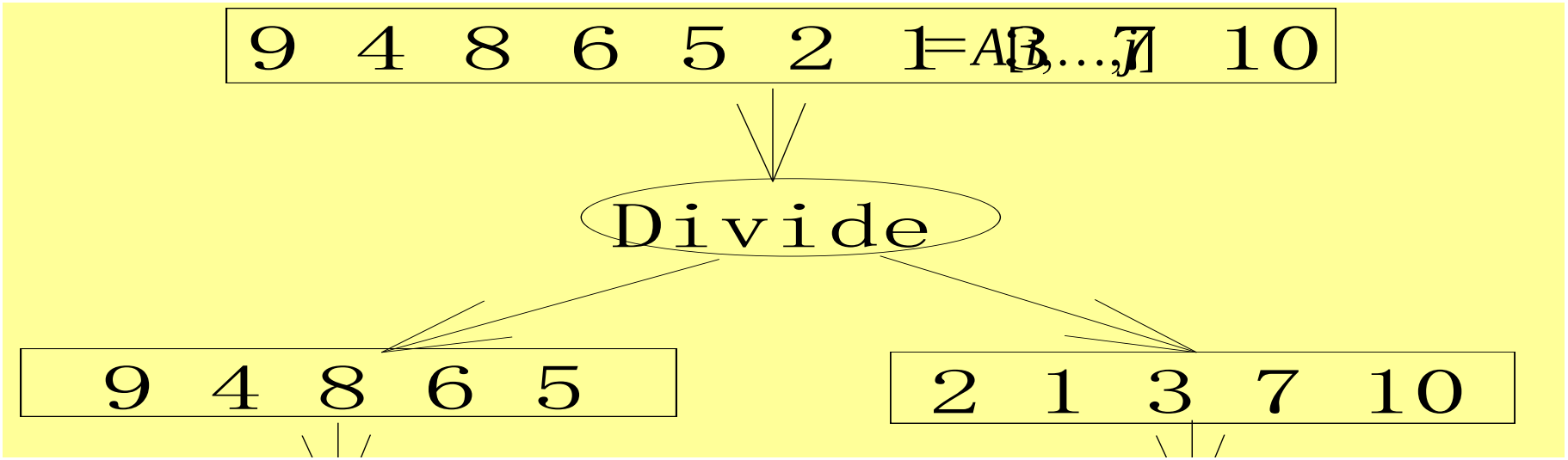
14，23，36，49，52，58，61，75，80，97

排序的方法

- 冒泡排序
- 插入排序
- 选择排序
- 快速排序
- 归并排序
- 堆排序
- 希尔排序
- 桶排序
- 计数排序



归并排序算法的复杂性分析



Divide：

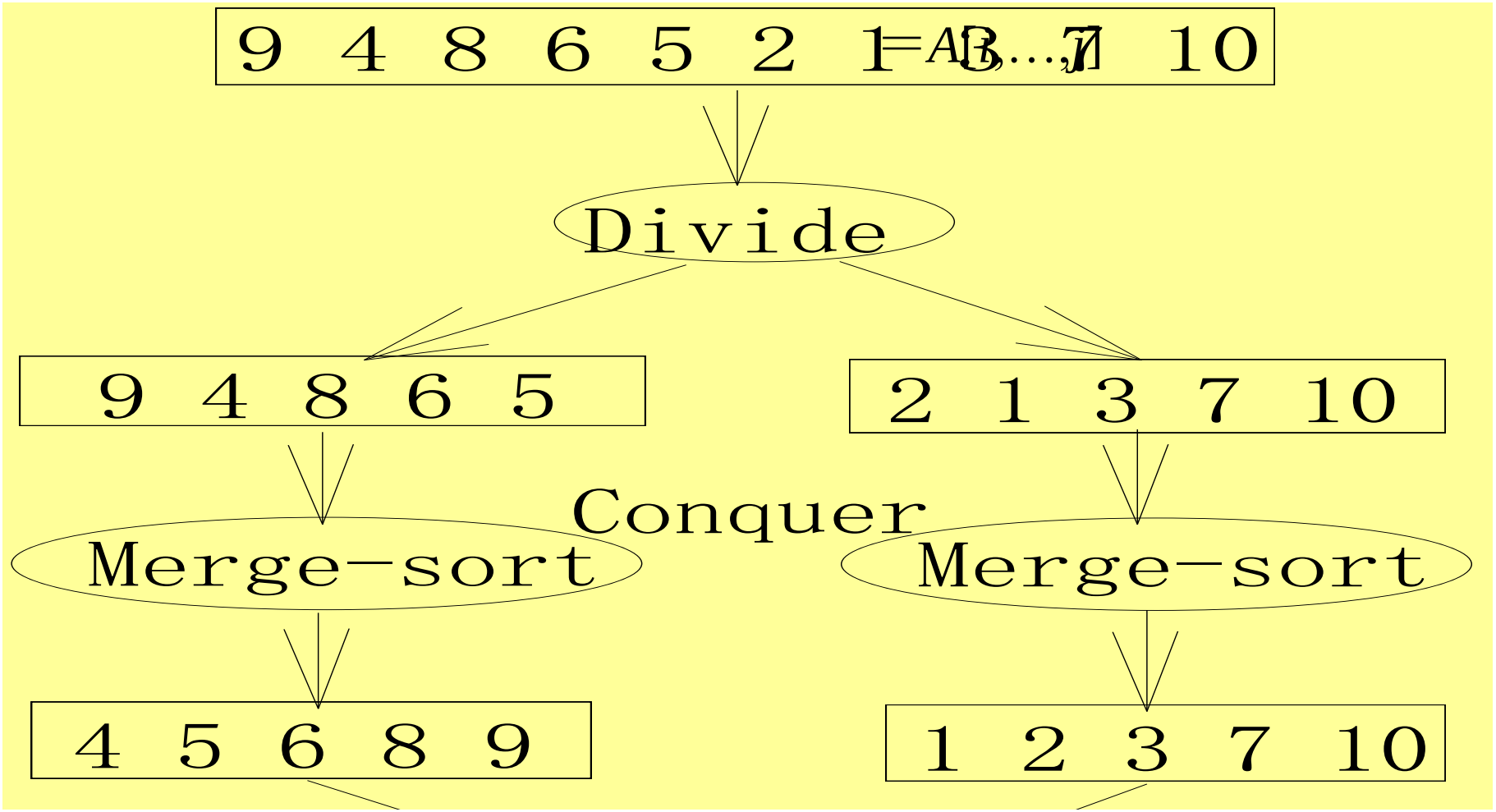
本质上仅需产生划分位置k

第一个子问题是$A[i,\ldots,k]$

第二个子问题是$A[k+1,\ldots,j]$

为使得两个问题的大小大致相当，k可以如下产生

$k=(i+j)/2$



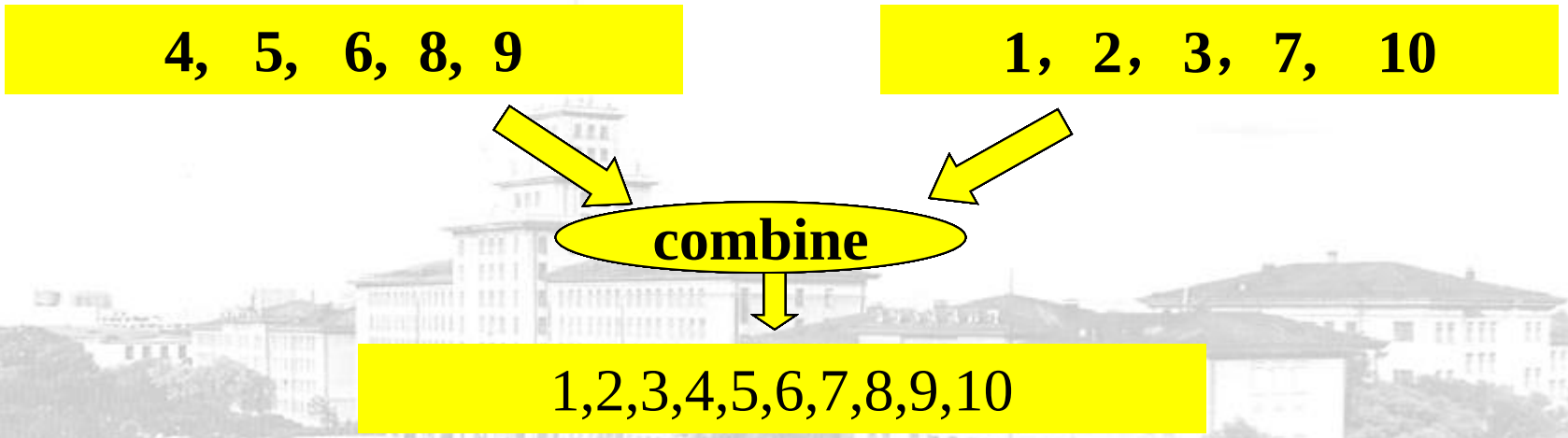
Conquer: 递归求解就是算法的递归调用

Mergesort(*A*,*i*,*k*); //求解第一个子问题

Mergesort(*A*,*k*+1,*j*); //求解第二个子问题

combine: 将两个有序序列合并成一个有序序列

1. $l \leftarrow i$; $h \leftarrow k+1$; $t=i$ //设置指针
2. While $l \le k$ & $h < j$ Do
   IF $A[l] < A[h]$ THEN $B[t] \leftarrow A[l]; l \leftarrow l+1; t \leftarrow t+1;$
   ELSE $B[t] \leftarrow A[h]; h \leftarrow h+1; t \leftarrow t+1;$
3. IF $l<k$ THEH //第一个子问题有剩余元素
   For $v \leftarrow l$ To k Do
   $B[t] \leftarrow A[v]; t \leftarrow t+1;$
4. IF $h<j$ THEN //第二个子问题有剩余元素
   For $v \leftarrow h$ To j Do
   $B[t] \leftarrow A[v]; t \leftarrow t+1;$



MergeSort(*A*,*i*,*j*)
Input: *A*[*i*,…,*j*]
Output:排序后的*A*[*i*,…,*j*]

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

```
1.  k ←(i+j)/2;
2.   MergeSort(A,i,k);
3.  MergeSort(A,k+1,j);
4.  l←i;     h ←k+1;   t=i                    //设置指针
5.  While l≤k &  h< j  Do
6.     IF  A[l] < A[h]  THEN  B[t] ←A[l]; l ←l+1; t ←t+1;
7.     ELSE B[t] ←A[h]; h ←h+1; t ←t+1;
8.   IF  l<k  THEH                            //第一个子问题有剩余元素
9.        For  v ←l  To  k  Do
10.            B[t] ←A[v]; t ←t+1;
11.  IF  h<j  THEN                            //第二个子问题有剩余元素
12.       For  v ←h  To  j  Do
13.            B[t] ←A[v]; t ←t+1;
14.  For  v ←i  To  j  Do                     //将归并后的数据复制到A中
15.       A[v] ←B[v];
```

插入排序的复杂性分析

```
Insertion-sort(A)
Input:  A[1,.....,n]=n个数
output: A[1,.....,n]=n个sorted数
  1.FOR i=2  To n  Do
  2.  key←A[i];
  3.  j←i-1
  4.  WHILE j>0 AND A[j]>key  Do
  5.    A[j+1]←A[j];
  6.    j←j-1;
  7.  A[j+1]←key;
```

复杂性分析——比较操作的次数

最好?

最坏?

平均?

插入排序算法的复杂性分析

插入排序的分析

- 最好情况: 已经排序

  $d_i = 1$ for $2 \le i \le n$

  $\Rightarrow M = n - 1 = \Theta(n)$

- 最坏情况: 逆序

  $d_1 = 0$

  $d_2 = 1$

  :

  $d_i = i-1$

  $d_n = n-1$

$$M = \sum_{i=1}^{n} d_i = \frac{(n-2)(n-1)}{2} = \Theta(n^2)$$

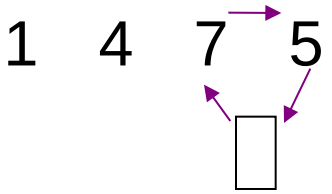
- 平均情况:

  x_j 插入已排序的序列 x_1 x_2 … x_{j-1}时

- x_j 最大的概率: 1/j
  - 1次比较
- x_j 次大的概率 : 1/j
  - 2次比较

  :

- 插入 x_j时的比较次数:



$$\frac{1}{j}+\frac{2}{j}+\cdots+\frac{j-1}{j}+\frac{j-1}{j}=\frac{(j-1)j+2(j-1)}{2j}=\frac{j}{2}+\frac{1}{2}-\frac{1}{j}$$

$$M=\sum_{j=2}^{n}(\frac{j}{2}+\frac{1}{2}-\frac{1}{j})=\Theta(n^2)$$

快速排序算法的复杂性分析

快速排序的复杂性分析

PartitionSort(*A*,*i*,*j*)
Input: $A[i,\ldots,j]$, x
Output: 排序后的$A[i,\ldots,j]$

1. $x \leftarrow A[i]$;
2. k=partition(A,i,j,x);
3. partitionSort(A,i,k);
4. partitionSort(A,k+1,j);

Partition(*A*,*i*,*j*,*x*)

1. *low*←*i* ; *high* ←*j*;
2. While(*low*< *high*) Do
3. swap(*A*[*low*], *A*[*high*]);
4. While(*A*[*low*] < *x*) Do
5. *low*←*low*+1;
6. While(*A*[*low*] < *x*) Do
7. *high*←*high*-1;
8. return(*high*)

复杂性分析——比较操作的次数

最好?

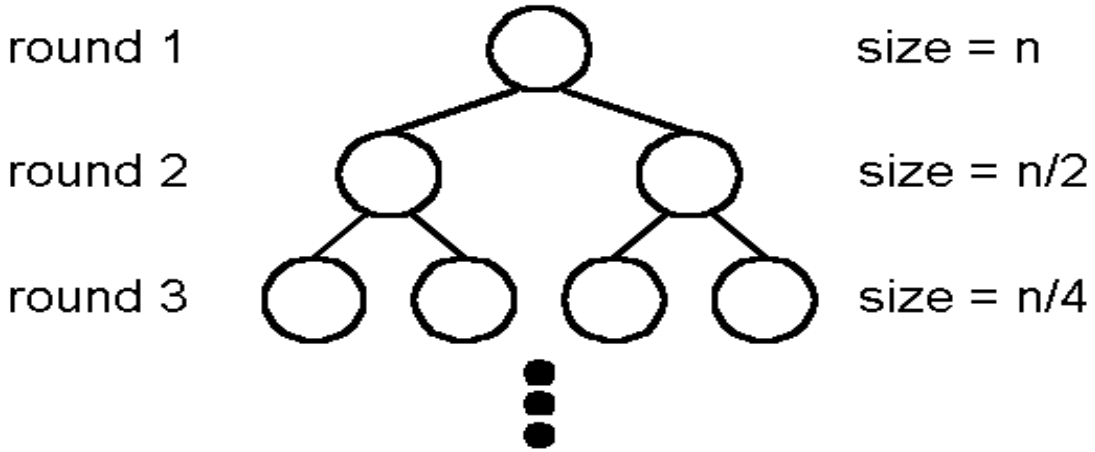
最坏?

平均?

快速排序的复杂性分析

最好情况 ： Θ(n log n)

数组被分为大致相等的两个部分.



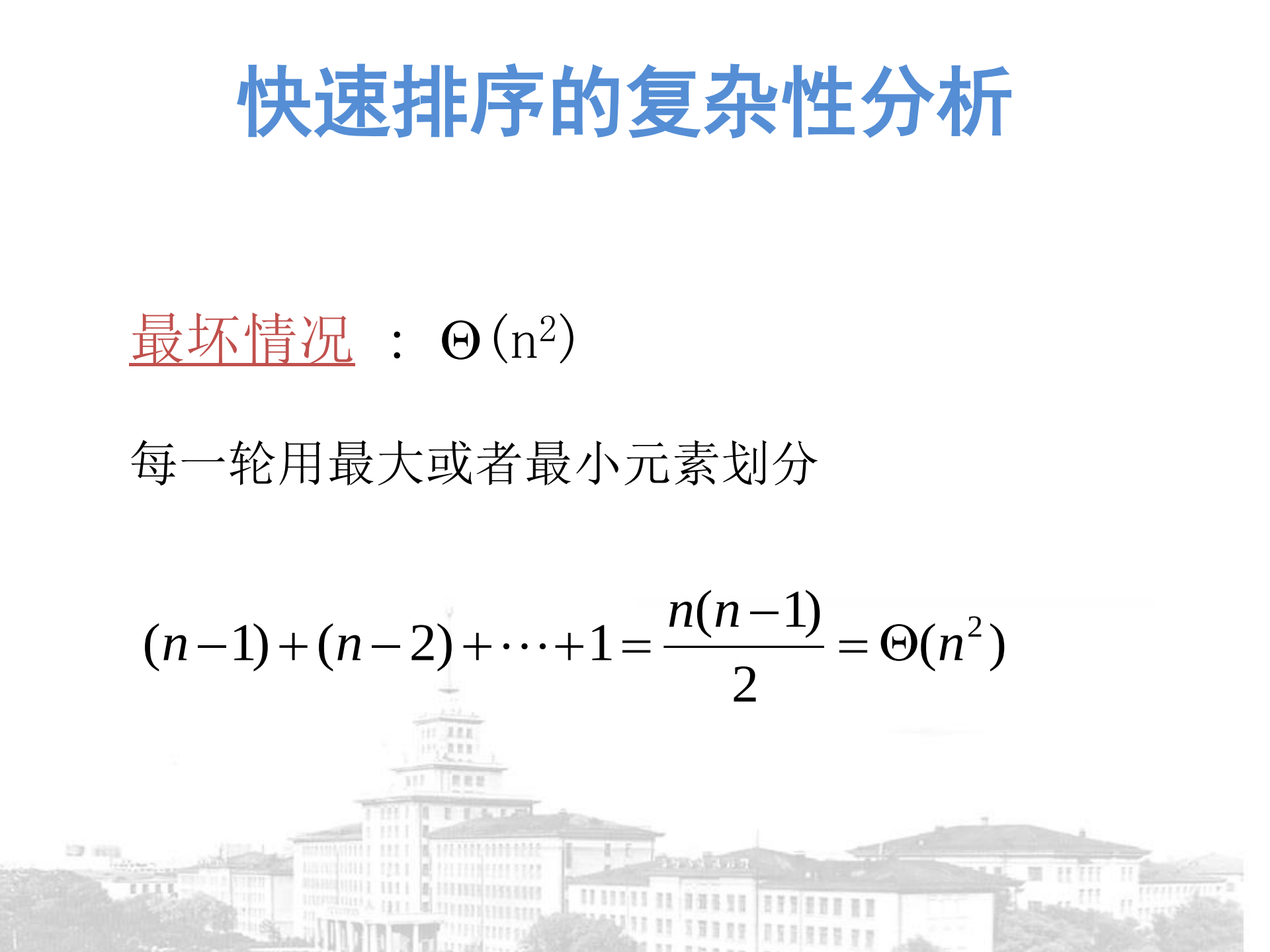
- *需要log n* 轮.
- 每轮需要O(n)次比较

快速排序的复杂性分析

最坏情况 ： $\Theta(n^2)$

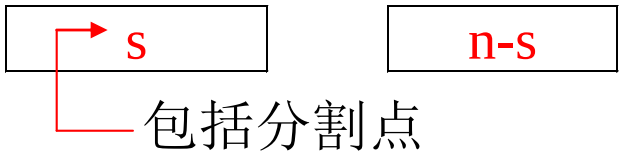
每一轮用最大或者最小元素划分

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=\Theta(n^2)$$



快速排序的复杂性分析

平均情况：$\Theta(n \log n)$



$$T(n) = \underset{1 \le s \le n}{Avg}(T(s) + T(n-s)) + cn$$

$$= \frac{1}{n}\sum_{s=1}^{n}(T(s) + T(n-s)) + cn$$

$$= \frac{1}{n}(T(1)+T(n-1)+T(2)+T(n-2)+\cdots+T(n)+T(0))+cn,\ T(0)=0$$

$$= \frac{1}{n}(2T(1)+2T(2)+\cdots+2T(n-1)+T(n))+cn$$

快速排序的复杂性分析

$$(n-1)T(n) = 2T(1)+2T(2)+\cdots+2T(n-1) + cn^2\cdots\cdots(1)$$
$$(n-2)T(n-1)=2T(1)+2T(2)+\cdots+2T(n-2)+c(n-1)^2\cdots(2)$$

(1) – (2)

$$(n-1)T(n) -(n-2)T(n-1) = 2T(n-1)+c(2n-1)$$
$$(n-1)T(n) -nT(n-1) = c(2n-1)$$

$$\frac{T(n)}{n} = \frac{T(n-1)}{n-1} + c(\frac{1}{n} + \frac{1}{n-1})$$

$$= c(\frac{1}{n} + \frac{1}{n-1}) + c(\frac{1}{n-1} + \frac{1}{n-2}) + \cdots + c(\frac{1}{2} + 1) + T(1),\ T(1) = 0$$

$$= c(\frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{2}) + c(\frac{1}{n-1} + \frac{1}{n-2} + \ldots + 1)$$

快速排序的复杂性分析

调和级数**[Knuth 1986]**

$$H_n = 1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$$

$$=\ln n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \varepsilon, \text{ where } 0<\varepsilon<\frac{1}{252n^6}$$

$$\gamma = 0.5772156649\cdots.$$

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

$$\frac{T(n)}{n} = c(H_n - 1) + cH_{n-1}$$

$$= c(2H_n - \frac{1}{n} - 1)$$

$$\Rightarrow T(n) = 2cnH_n - c(n+1)$$

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

本讲内容

1.1 从排序看算法分析

1.2 快速排序深入剖析

1.3 问题复杂度下界

1.4 基于比较的排序算法的时间复杂度下界

随机快速排序算法

```
QuickSort(A,i,j)
Input:   A[i,…,j], x
Output: 排序后的A[i,…,j]
1.  temp ←rand(i,j);                    //产生i，j之间的随机数
2.  x←A[temp];                          //以确定的策略选择x
3.  k=partition(A,i,j,x);               //用x完成划分
4.  partitionSort(A,i,k);               //递归求解子问题
5.  partitionSort(A,k+1,j);
Partition(A,i,j,x)
1.  low←i ;  high ←j;
2.  While( low< high ) Do
3.          swap(A[low], A[high]);
4.          While( A[low] < x )  Do
5.                low←low+1;
6.          While( A[high] >= x )  Do
7.                high←high-1;
8.   return(high)
```

算法性能的分析

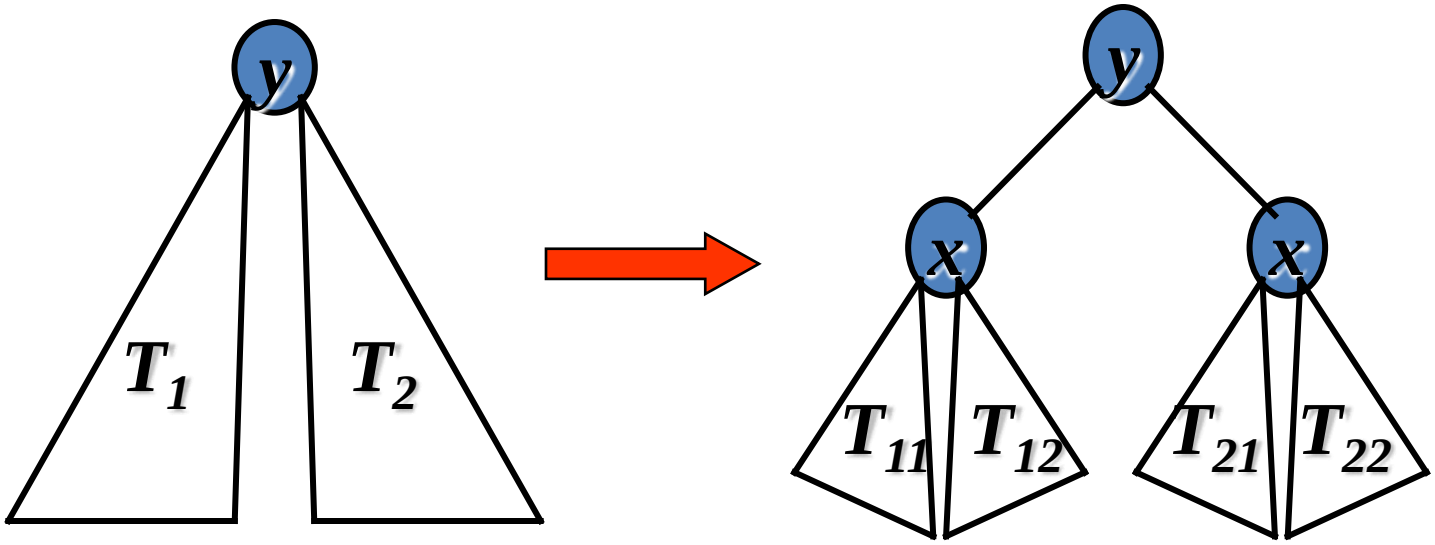
- 基本概念
  - $S_{(i)}$表示S中阶为i的元素
    例如，$S_{(1)}$和$S_{(n)}$分别是最小和最大元素
  - 随机变量X_{ij} 定义如下：
    $X_{ij}=1$如果$S_{(i)}$和$S_{(j)}$在运行中被比较，否则为0
  - X_{ij}是$S_{(i)}$和$S_{(j)}$的比较次数
  - 算法的比较次数为 $\sum_{i=1}^{n}\sum_{j>i} X_{ij}$
  - 算法的平均复杂性为 $E[\sum_{i=1}^{n}\sum_{j>i} X_{ij}]=\sum_{i=1}^{n}\sum_{j>i} E[X_{ij}]$

- 计算$E[X_{ij}]$
  - 设p_{ij}为$S_{(i)}$和$S_{(j)}$在运行中比较的概率，则

$$E[X_{ij}]=p_{ij}\times 1+(1-p_{ij})\times 0=p_{ij}$$

关键问题成为求解p_{ij}

- 求解P_{ij}
  - 我们可以用树表示算法的计算过程



- 我们可以观察到如下事实:
  - 一个子树的根必须与其子树的所有节点比较
  - 不同子树中的节点不可能比较
  - 任意两个节点至多比较一次

• 当$S_{(i)}, S_{(i+1)}, \ldots, S_{(j)}$在同一子树时, $S_{(i)}$和$S_{(j)}$才可能比较

•只有$S_{(i)}$或$S_{(j)}$先于$S(i+1), \ldots, S(j-1)$被选为划分点时, $S_{(i)}$和$S_{(j)}$才可能比较

• $S_{(i)}, S_{(i+1)}, \ldots, S_{(j)}$等可能地被选为划分点, 所以$S_{(i)}$和$S_{(j)}$

进行比较的概率是: $2/(j-i+1)$, 即

$$p_{ij}=2/(j-i+1)$$

- 现在我们有

$$\sum_{i=1}^{n}\sum_{j>i}E[X_{ij}]=\sum_{i=1}^{n}\sum_{j>i}p_{ij}=\sum_{i=1}^{n}\sum_{j>i}\frac{2}{j-i+1}$$

$$\leq\sum_{i=1}^{n}\sum_{k=1}^{n-i+1}\frac{2}{k}\leq 2\sum_{i=1}^{n}\sum_{k=1}^{n}\frac{1}{k}=2nH_n=O(n\log n)$$

定理. 随机排序算法的期望时间复杂性为O(*n*log*n*)

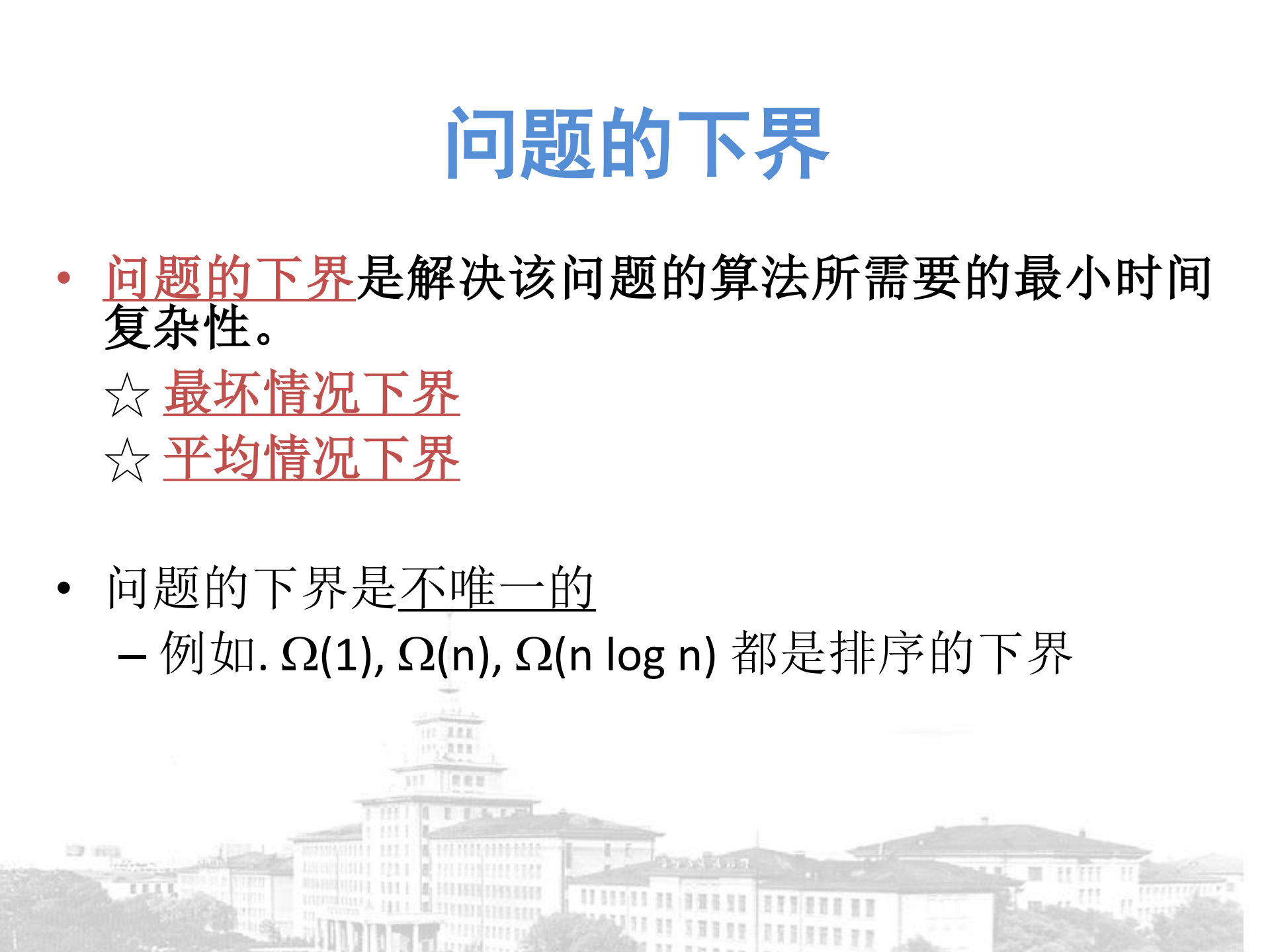
本讲内容

1.1 从排序看算法分析
1.2 快速排序深入剖析
1.3 问题复杂度下界
1.4 基于比较的排序算法的时间复杂度下界

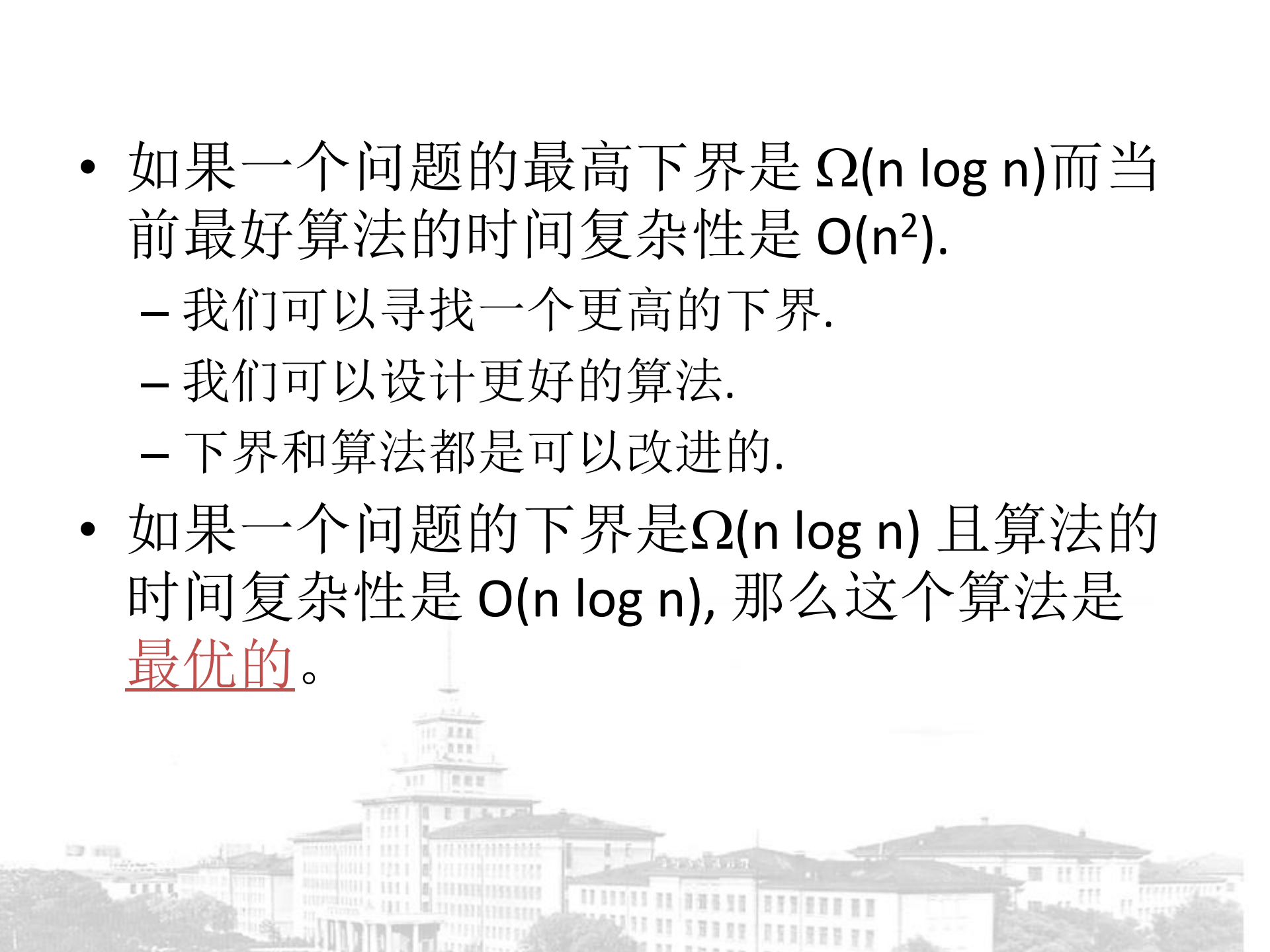
问题的下界

- 问题的下界是解决该问题的算法所需要的最小时间复杂性。
  - ☆ 最坏情况下界
  - ☆ 平均情况下界

- 问题的下界是不唯一的
  - 例如. $\Omega(1)$, $\Omega(n)$, $\Omega(n \log n)$ 都是排序的下界



- 如果一个问题的最高下界是 $\Omega(n \log n)$而当前最好算法的时间复杂性是 $O(n^2)$.
  - 我们可以寻找一个更高的下界.
  - 我们可以设计更好的算法.
  - 下界和算法都是可以改进的.
- 如果一个问题的下界是$\Omega(n \log n)$ 且算法的时间复杂性是 $O(n \log n)$, 那么这个算法是最优的。



利用神谕改进下界

问题 P: 合并两个长度为m和n的序列A和B.

- 神谕尽力使算法在最坏情况下运行(给出一个难解的输入)
- 已排序序列:
  - A: $a_1 < a_2 < \ldots < a_m$
  - B: $b_1 < b_2 < \ldots < b_m$
- 最难的情况:
  - $a_1 < b_1 < a_2 < b_2 < \ldots < a_m < b_m$

- 必须比较:

$a_1 : b_1$
$b_1 : a_2$
$a_2 : b_2$
:
$b_{m-1} : a_{m-1}$
$a_m : b_m$

- 否则对某些输入就会出错.

e.g. 如果不比较 b_1 和 a_2, 我们就不能区分

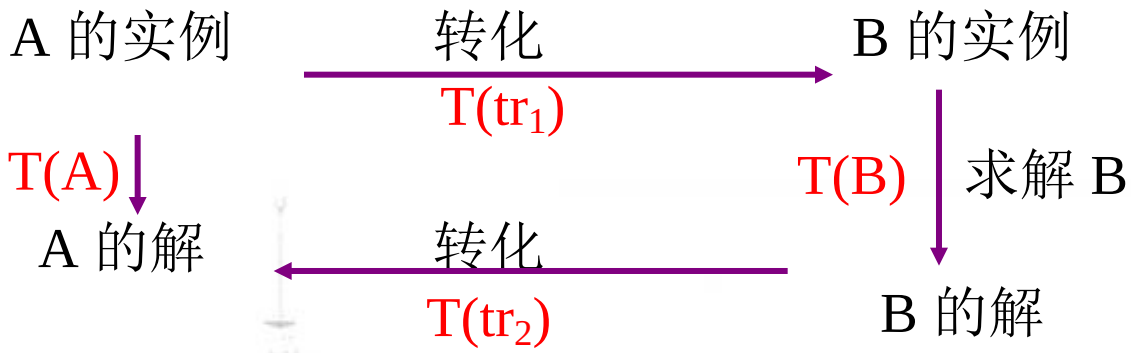
$a_1 < b_1 < a_2 < b_2 < \ldots < a_m < b_m$ 和

$a_1 < a_2 < b_1 < b_2 < \ldots < a_m < b_m$

- 因此, 至少需要$2m-1$次比较。

通过问题转换求下界

- 问题 A 规约到问题 B ($A \propto B$)
  - 当且仅当A可以利用任何解决B的算法求解.
  - 如果$A \propto B$, B 更难.



- 注意：$T(tr_1) + T(tr_2) < T(B)$

  $T(A) \leq T(tr_1) + T(tr_2) + T(B) \sim O(T(B))$

凸包问题的下界

- 排序 $\propto$ 凸包

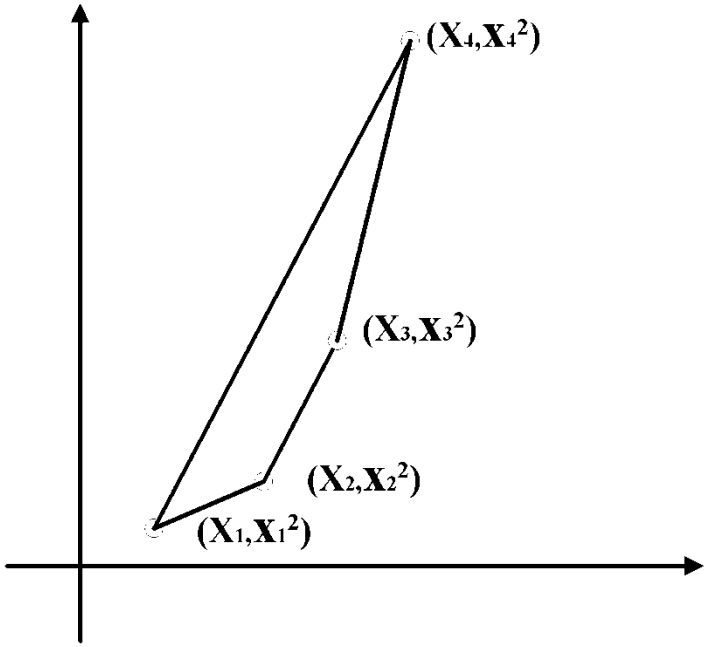
  A　　B

- A的一个实例: $(x_1, x_2, \ldots, x_n)$

  ↓转化

  B的一个实例: $\{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$

  假设: $x_1 < x_2 < \ldots < x_n$



- 如果凸包问题可解，我们可以解决排序问题
  - 排序的下界是: $\Omega(n \log n)$
- 凸包问题的下界是: $\Omega(n \log n)$

本讲内容

1.1 从排序看算法分析
1.2 快速排序深入剖析
1.3 问题复杂度下界
1.4 基于比较的排序算法的时间复杂度下界

最坏情况下排序的下界

3个元素有6种排列

a_1	a_2	a_3
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

直接插入排序

- 输入: (2, 3, 1)

  (1) a_1:a_2

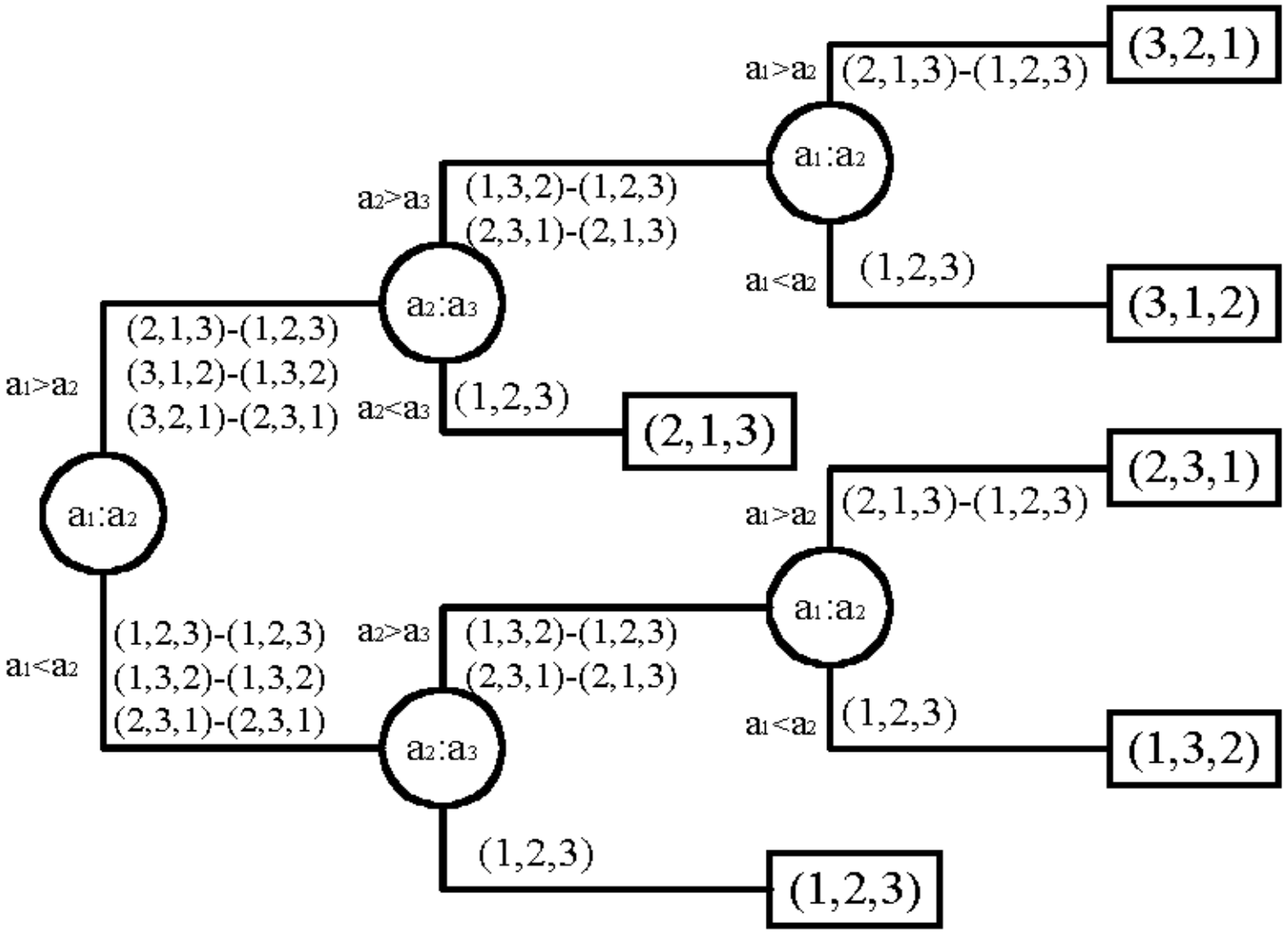
  (2) a_2:a_3, $a_2 \leftrightarrow a_3$

  (3) a_1:a_2, $a_1 \leftrightarrow a_2$
- 输入: (2, 1, 3)

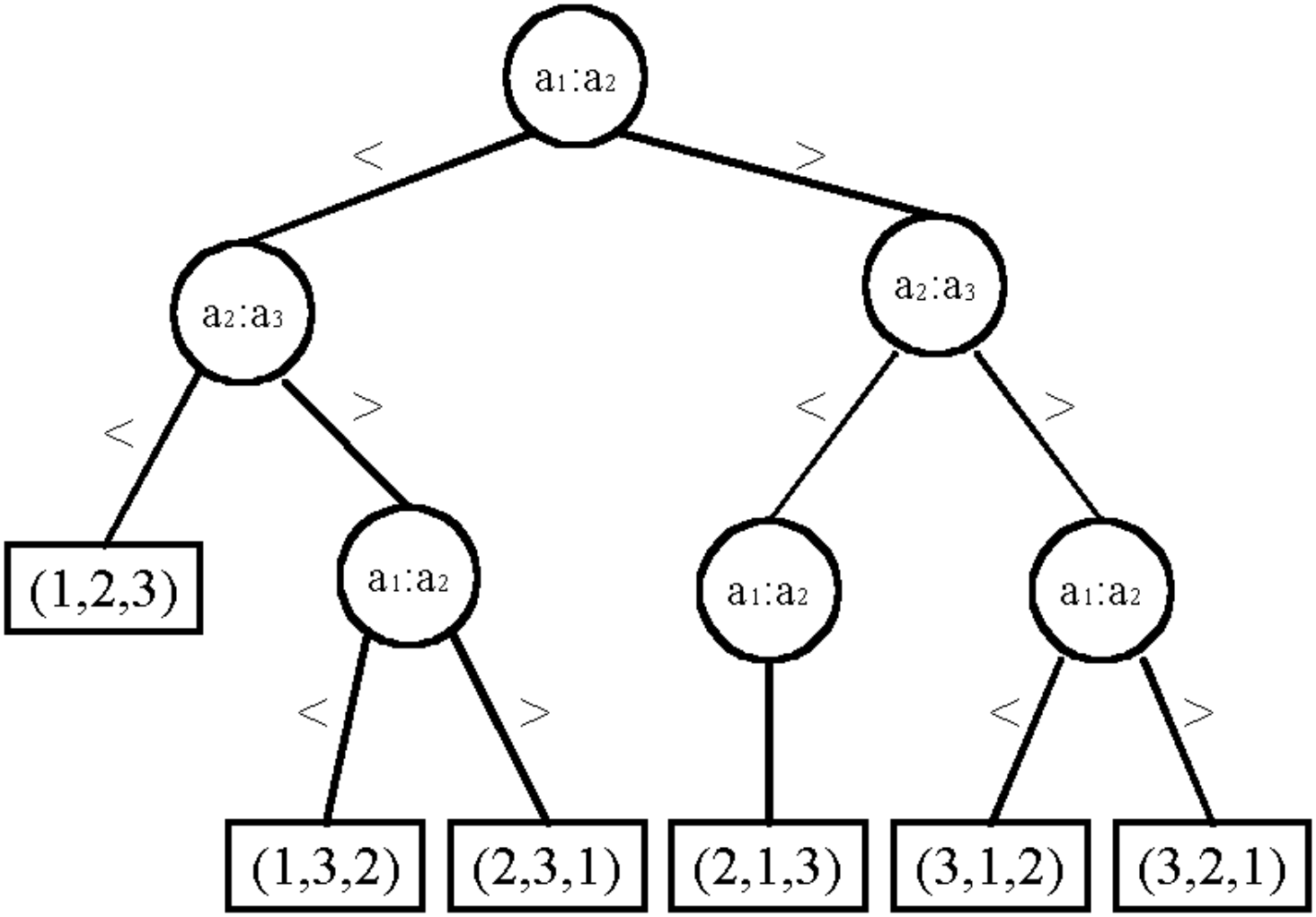
  (1)a_1:a_2, $a_1 \leftrightarrow a_2$

  (2)a_2:a_3

直接插入排序的决策树


a1:a2
a1>a2
(2,1,3)-(1,2,3)
(3,1,2)-(1,3,2)
(3,2,1)-(2,3,1)
a2:a3
a2>a3
(1,3,2)-(1,2,3)
(2,3,1)-(2,1,3)
a2<a3
(1,2,3)
(2,1,3)
a1<a2
(1,2,3)-(1,2,3)
(1,3,2)-(1,3,2)
(2,3,1)-(2,3,1)
a1:a2
a1>a2
(2,1,3)-(1,2,3)
(3,2,1)
a1<a2
(1,2,3)
(3,1,2)
a2:a3
a2>a3
(1,3,2)-(1,2,3)
(2,3,1)-(2,1,3)
a2<a3
(1,2,3)
(1,2,3)
a1:a2
a1>a2
(2,1,3)-(1,2,3)
(2,3,1)
a1<a2
(1,2,3)
(1,3,2)

冒泡排序的决策树



排序的下界

- 为了找到排序的下界，我们需要找到二叉树的最小高度.
- 有n!种不同排列

  二叉决策树有n!个叶子结点.
- 平衡树高度最小

  $\lceil \log(n!) \rceil = \Omega(n \log n)$

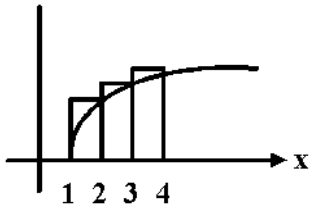
  排序的下界是: $\Omega(n \log n)$

方法 1:

$$\log(n!) = \log(n(n-1)\cdots 1)$$

$$= \log 2 + \log 3 + \cdots + \log n$$

$$> \int_1^n \log x\,dx$$



$$= \log e \int_1^n \ln x\,dx$$

$$= \log e[x \ln x - x]_1^n$$

$$= \log e(n \ln n - n + 1)$$

$$= n \log n - n \log e + 1.44$$

$$\geq n \log n - 1.44n$$

$$= \Omega(n \log n)$$

方法2:

- Stirling近似:
  - $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$
  - $\log n! \approx \log \sqrt{2\pi} + \frac{1}{2}\log n + n\log\frac{n}{e} \approx n \log n \approx \Omega(n \log n)$

n	n!	S_n
1	1	0.922
2	2	1.919
3	6	5.825
4	24	23.447
5	120	118.02
6	720	707.39
10	3,628,800	3,598,600
20	2.433×10^{18}	2.423×10^{18}
100	9.333×10^{157}	9.328×10^{157}

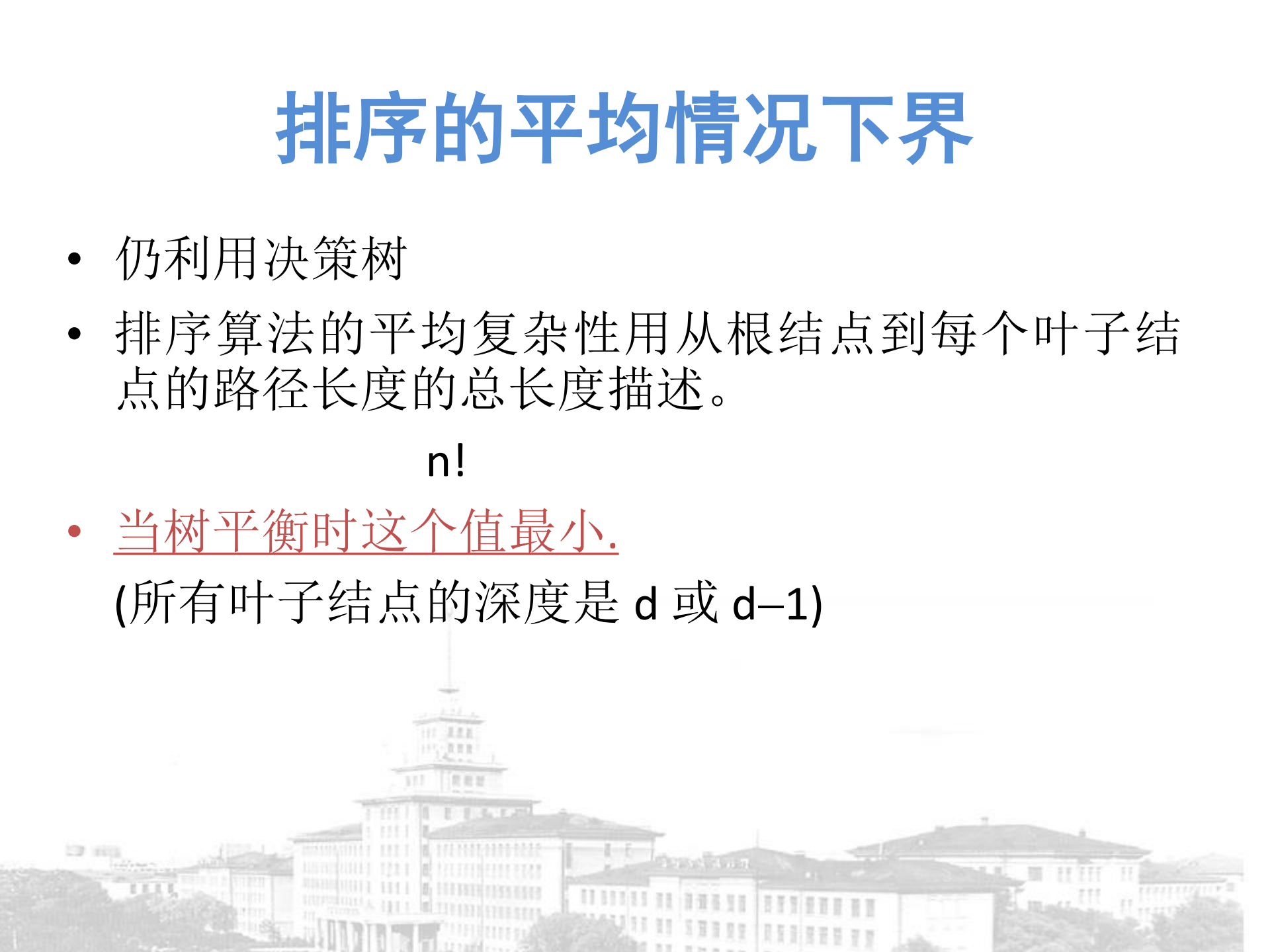
排序的平均情况下界

- 仍利用决策树
- 排序算法的平均复杂性用从根结点到每个叶子结点的路径长度的总长度描述。

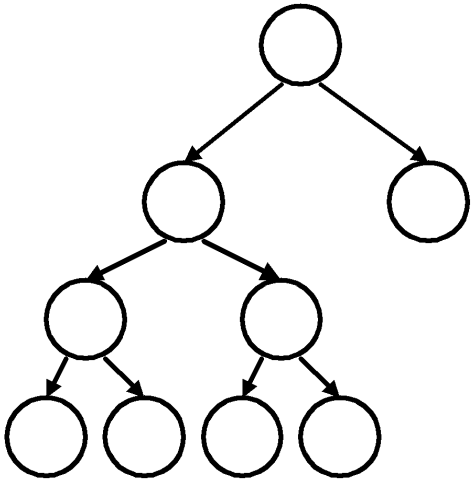
$$n!$$

- 当树平衡时这个值最小.

(所有叶子结点的深度是 d 或 d–1)



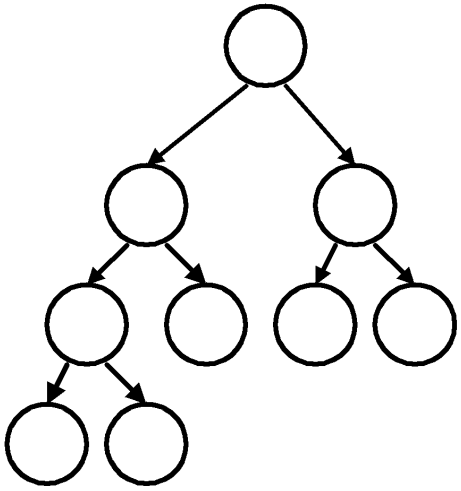
平均情况下排序的下界



不平衡的情况下

总长度

$= 4 \cdot 3 + 1 = 13$



平衡的情况下

总长度

$= 2 \cdot 3 + 3 \cdot 2 = 12$

计算最小路径长度和

1. 有c个叶子结点的二叉树的深度$d = \lceil \log c \rceil$
   叶子结点仅出现在d或d–1层.
2. x_1 个叶子在d–1层
   x_2个叶子在d层
   - $x_1 + x_2 = c$
   - $x_1 + \frac{x_2}{2} = 2^{d-1}$

   $\Rightarrow$ $x_1 = 2^d - c$
   $x_2 = 2(c - 2^{d-1})$

3. 总长度

$$M = x_1(d-1) + x_2 d$$
$$= (2^d - 1)(d-1) + 2(c - 2^{d-1})d$$
$$= c(d-1) + 2(c - 2^{d-1}),\ d - 1 = \lfloor \log c \rfloor$$
$$= c\lfloor \log c \rfloor + 2(c - 2^{\lfloor \log c \rfloor})$$

4. $c = n!$

$$M = n!\lfloor \log n! \rfloor + 2(n! - 2^{\lfloor \log n! \rfloor})$$
$$M/n! = \lfloor \log n! \rfloor + 2$$
$$= \lfloor \log n! \rfloor + c,\ 0 \le c \le 1$$
$$= \Omega(n \log n)$$

平均情况下排序的下界是: $\Omega(n \log n)$