L22：排序 II

吉林大学计算机学院
谷方明
fmgu2002@sina.com

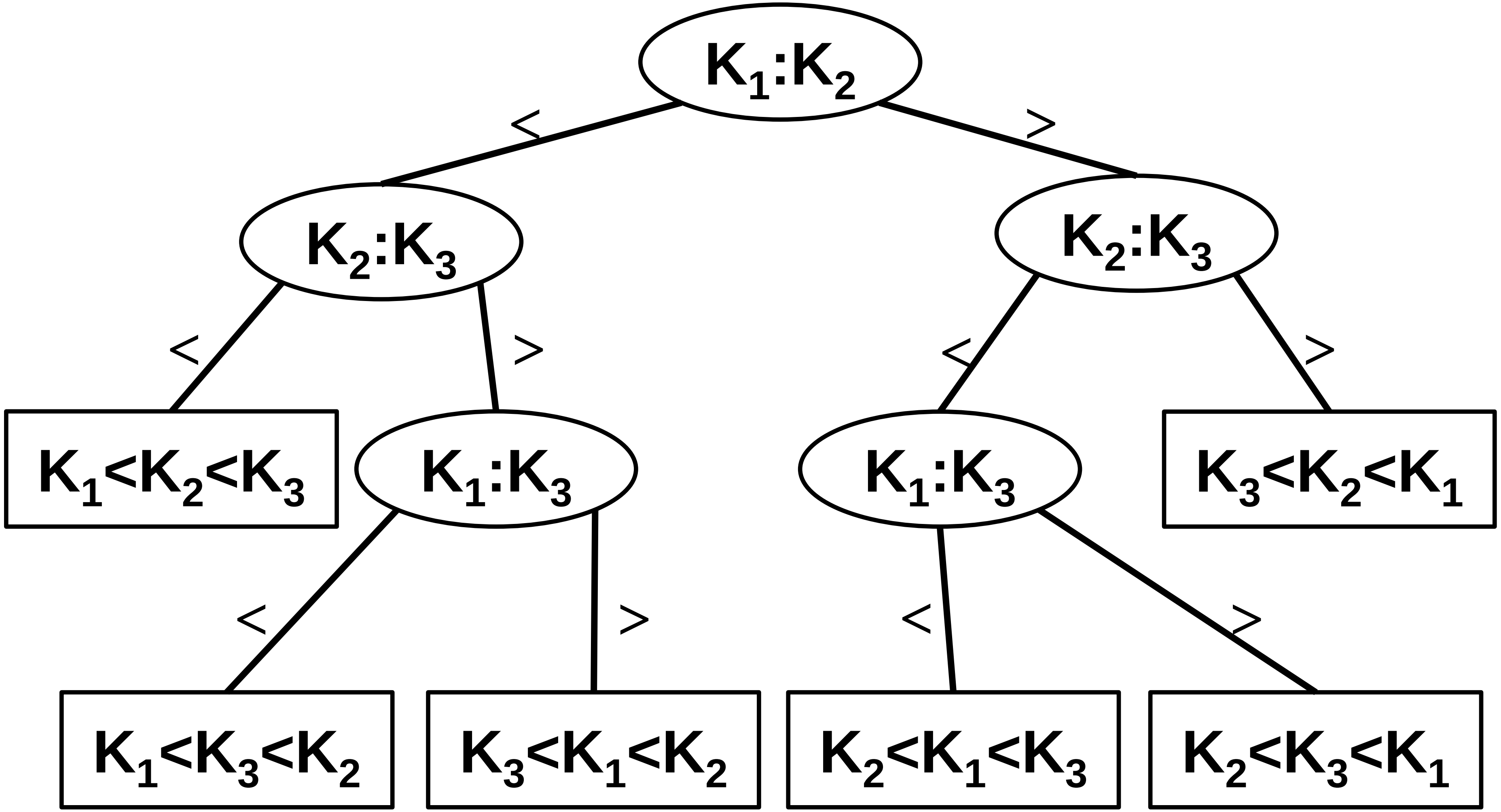
问题下界

- 若一个问题的规模为n，则“该问题之算法的时间复杂度下界为$L(n)$”，其含义是：不存在解该问题的算法，它的时间复杂度小于$L(n)$.

- 基于关键词比较的排序算法时间复杂度下界为 $\Omega(n\log n)$.
  - ✓ 即任何基于关键词比较的排序算法，其关键词比较次数的阶至少为$n\log n$.

排序判定树（基于关键词比较）

- 任意一个排序过程对应着一棵排序判定树
- 分支结点为关键词比较，叶子结点为排序结果



最坏时间复杂度下界

$$s(n) = \min_{\substack{\text{排序}\\\text{算法}}}\{\max_{\text{输入}}\{\text{关键词比较次数}\}\}$$

- 排序判定树的高度**h**是排序算法在最坏情况下的关键词比较次数的最大值。
- 引理：高度为 h 的二叉树最多有**2** ^{h} 个叶结点。
- 证明：

  n! ≤排序判定树的叶结点数 ≤ **2** ^{h}

  h ≥ log(n!)　　//也可用斯特林公式近似

  = logn + log(n-1) + … + 1

  ≥ logn + … + log(n/2)

  ≥ n/2log(n/2)

s(n) = min h ≥ n/2 * log(n/2) = ***Ω(nlogn)***

期望时间复杂度下界

$$\overline{S(n)} = \min_{\substack{排序\\算法}} \left\{ \frac{1}{n!} \sum_{排列} \left(关键词比较次数\right) \right\}$$

- ∑(关键词比较次数) =排序判定树的外通路长度。
- 定理: n个内结点扩充二叉树的内通路长度的最小值为 $(n+1)k - 2^{(k+1)} + 2$. (Huffman拓展)
- 证明:

判定树的叶结点数 $\geq n!$

判定树的内结点数 $N \geq n!-1$, $k = \lfloor \log N \rfloor$

判定树的外通路长度 $\geq (N+1)k - 2^{(k+1)} + 2 + 2N$

$\geq (N+1)k + 2$

$\geq n!\ k$

$S(n) \geq k = \log(n!-1) = \Omega(n\log n)$

课后思考

- 有5个互不相同的元素a、b、c、d、e，至少通过几次比较就能确保将其排好序？并给出比较的方案

更多知识

- 基于关键词比较的排序算法的下界
  - ✓ 最坏时间复杂度Ω(nlogn)；
  - ✓ 期望时间复杂度Ω (nlogn)；

- 当知道待排序关键词的更多知识时，例如分布范围等，就能突破基于关键词比较的排序算法下界，能在最坏情况下达到线性时间。

1. 计数排序(CountingSort)

- 设记录序列 $R_1, R_2, \ldots, R_n$,
- 对应的关键词满足 $u \le K_i \le v$ 且 K_i 为整数 $(1 \le i \le n)$,

- 计数排序思想: 对于每个记录R，确定小于R的记录个数。利用这一信息可确定记录的位置
  - ✓ 用计数数组*COUNT*[ ] 或 *C*[ ]辅助排序。

例子

- 待排序文件为 $R_1, R_2, R_3, R_4, R_5, R_6, R_7$，对应的关键词$K_1=3, K_2=1, K_3=1, K_4=3, K_5=2, K_6=3, K_7=2$
- $u=1$，$v=3$

	C[1]	C[2]	C[3]
D2	2	2	3
D3	2	4	7

	1	2	3	4	5	6	7
R	3	1	1	3	2	3	2
S	1	1	2	2	3	3	3
C	1,0	3,2	6,5,4				

算法D (R , n . S)

```
for( i=u ; i<=v ; i++)  count[i]=0;
for( i=1 ; i<=n ; i++)  count[R[i].K] ++;
//COUNT[Kj ]是关键词等于Kj 的记录的个数
for( i = u+1 ; i<=v ; i++ ) count[i]+=count[i-1];
/* COUNT[Kj ]关键词=Kj 的记录最终排序位置的最大序号 */
for( i = n ; i >=1 ; i-- ) //稳定；
    S[ count[ R[i].K ] -- ] = R[i];
```

计数排序分析

- 时间复杂度： O(|v-u| + n)
- 辅助空间:： O(|v-u| + n)
- 稳定性： 稳定。
- 适用范围：[u,v]的范围不能太大

计数排序扩展

- 如何修改算法D， 使得其最后一步从前向后处理记录并且保证稳定性？

- 计数排序可以只用关键字比较实现。

2. 桶排序(Bucket-Sort)

- 例：英文单词卡片字典序

  首先按首字母分成26堆，第1堆以字母a开头，第2堆以字母b开头，……，然后再对各堆排序；

- 基于关键词的性质分“堆”或“桶”的排序方法，称作桶排序；

桶排序算法思想

算法Bucket (R , n , B)

B1. [分桶]

把n个记录分成B个桶；

B2. [排序]

对每个桶进行桶内排序；

把所有桶按序合并；

- 桶排序需要具体化

桶的选取

- 对整数或实数，可基于关键字的数字性质分桶。如果已知K_1，K_2，…，K_n在区间（K_0，K_{n+1}）上的分布是某种熟悉的分布，则可通过这种分布和区间来选择桶。
- 例如，如果K_1，K_2，…，K_n在（K_0，K_{n+1}）上呈均匀分布，则有b个桶B_1，B_2，…，B_b，且B_j的定义如下（$1 \le j \le b$）：

$$K_0 + \frac{K_{n+1} - K_0}{b}(j-1) < K_i \le K_0 + \frac{K_{n+1} - K_0}{b} j$$

则给定K_i就可确定一个桶j，然后分别独立地排序各桶，最后把所有的桶合并在一起，形成排序文件。

桶内排序算法的选取I

- 普通排序方法，例如插入排序等
- 结论：均匀分布，**n个桶**，使用插入排序进行桶内排序的桶排序算法的期望时间复杂度为O(n)。

证明：$T(n) = \Theta(n) + \sum_{i=1}^{n} O(n_i^2)$

$$\mathrm{E}[T(n)] = \Theta(n) + \sum_{i=1}^{n} O\left(E\left[n_i^2\right]\right)$$

$$X_{ij} = I\{A[j]\text{落入桶}i\} \qquad n_i = \sum_{j=1}^{n} X_{ij}$$

$$E\left[n_i^2\right] = \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1\le j\le n}\sum_{k\ne j} E[X_{ij}X_{ik}]$$

$$E\left[n_i^2\right] = 1 + \frac{n-1}{n} = 2 - \frac{1}{n}$$

$$E[T(n)] = \Theta(n)$$

桶内排序算法的选取II

- 选择桶排序：递归（迭代）
- 结论：如果每个桶内的排序仍递归进行，则“桶”的个数将直接影响排序算法的效率.

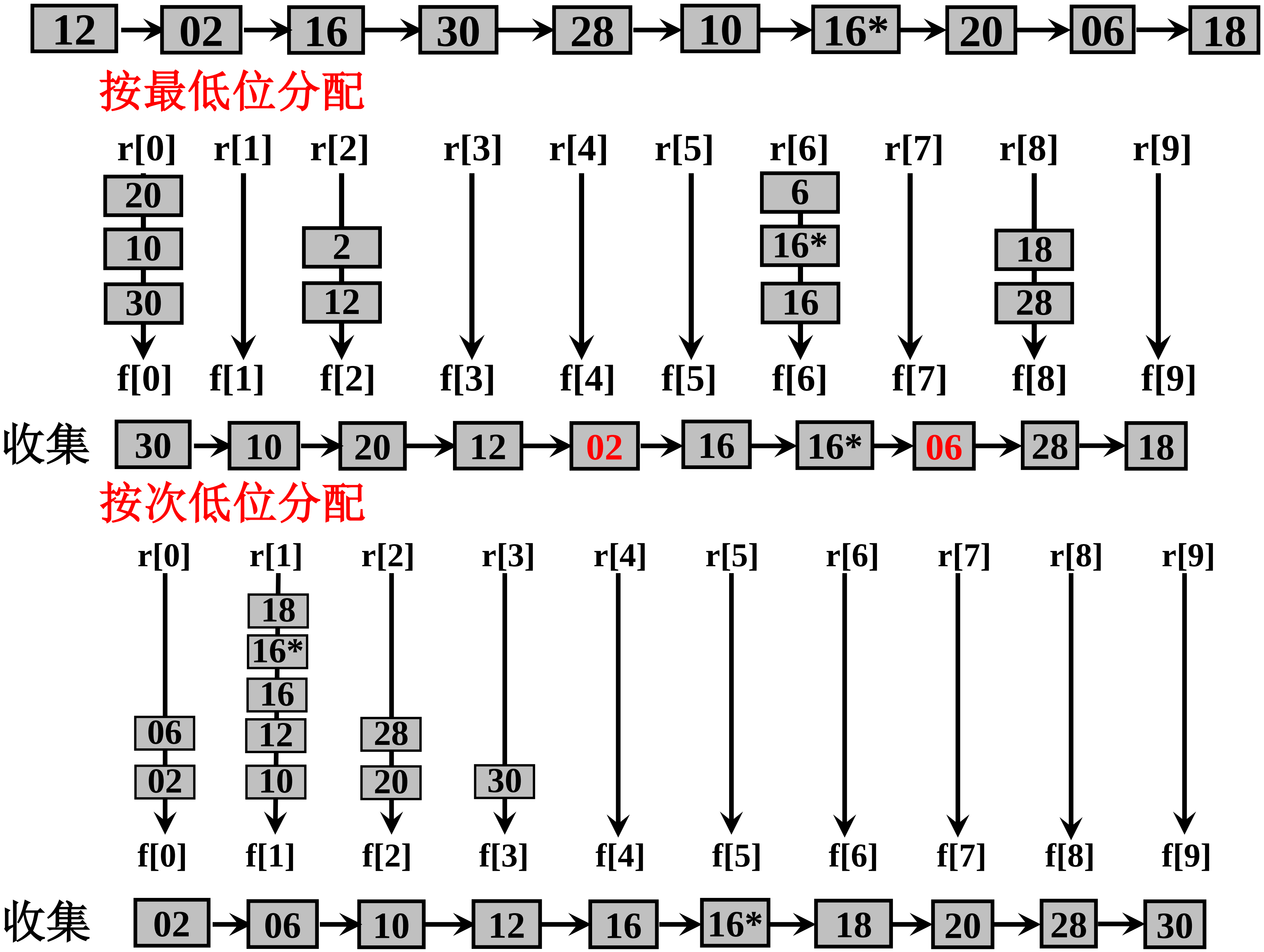
桶数b	桶排序算法的期望时间复杂度
常数	O(n logn)
n	O(n)

3.基数排序(radix sort)

- 卡片排序机（博物馆）

- 将整数看成数位的序列，每次按1位穿孔排序，d位数需排d次，每次关键字范围0-9，基数10.

- 卡片穿孔排序



12 → 02 → 16 → 30 → 28 → 10 → 16* → 20 → 06 → 18
按最低位分配
r[0] r[1] r[2] r[3] r[4] r[5] r[6] r[7] r[8] r[9]
20
10
30
2
12
6
16*
16
18
28
f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7] f[8] f[9]
收集
30 → 10 → 20 → 12 → 02 → 16 → 16* → 06 → 28 → 18
按次低位分配
r[0] r[1] r[2] r[3] r[4] r[5] r[6] r[7] r[8] r[9]
18
16*
16
12
10
06
02
28
20
30
f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7] f[8] f[9]
收集
02 → 06 → 10 → 12 → 16 → 16* → 18 → 20 → 28 → 30

相关定义

- 假定与记录 $R_1, R_2, \ldots, R_n$ 对应的关键词 $K_1, K_2, \ldots, K_n$都可表成 $K_i = (K_{i,p}, K_{i,p-1}, \ldots, K_{i,1})$, 且对每个$t$ $(1 \le t \le p)$都有$0 \le K_{i,t} < r$，r 为基数.
- 基数排序是基于关键词的上述表示, 按字典序由小到大排列。
- 字典序定义:

  $K_i = (K_{i,p}, \ldots, K_{i,1}) < K_j = (K_{j,p}, \ldots, K_{j,1})$

  当且仅当 $K_{i,p} < K_{j,p}$， 或者存在 $1 \le t < p$,

  使得当 $s > t$ 时，有 $K_{i,s} = K_{j,s}$， 而 $K_{i,t} < K_{j,t}$.

基数排序算法思想

- 算法思想

  for i = 1 to p

  use a stable sort to sort A on digit i.

- 第i位排序算法选择
  - ✓ 桶排序：r个桶，分配、收集
  - ✓ 计数排序

算法RadixSort (*Q*, *n*, *p*, *r*)

RS1. 形成初始队列Q.

RS2. [从关键词低位到高位排序]

for(i=1 ; i<=p ;i++) {

将队列 $Q0$, $Q1$, …, $Qr–1$清空 .

while(Q不空) {

$X \Leftarrow Q$.

令X=(K[p], K[p–1], …, K[1]) .

$Q_{K[i]} \Leftarrow X$.

}

合并$Q0$, $Q1$, …, $Qr–1$形成新的Q .

}

算法的正确性

- 定理7.3

  如果算法 RadixSort 第 r 次 FOR 循环时，R_i 的部分关键词为 $X_i=(K_{i,r}, K_{i,j-1}, \ldots, K_{i,1})$, 则第 r 次FOR循环所形成的新序列 Q，是按 X_i 排序的.

- 数学归纳法
  - ✓ r=1时，显然成立。设r<t时成立，当r=t时，如果$R_\alpha<R_\beta$, 两种情况：$K_{\alpha,t}=K_{\beta,t}$ 和$K_{\alpha,t}<K_{\beta,t}$

算法分析

- **辅助空间：r+1个队头指针和队尾指针，n个link域**
- **时间复杂度**
  - ✓ 分配：O(n)
  - ✓ 收集：链队实现收集较快，结点类型(data，link)，O(r)，一般 r<<n ；其它方式能做到O(n)
  - ✓ RadixSort时间复杂度O(n p)，
- **稳定**

排序次序

- 最低位优先法（**LSD：Least Significant Digit First**）：先按最低位进行排序，然后对得到的结果重复处理.
  - ✓ 优先级高的最后排

- 最高位优先法（**MSD：Most Significant Digit First**）：先按最高位进行排序，然后对得到的结果重复处理.
  - ✓ 递归，空间需求大

基数排序应用广泛

- 将单词看成字母的序列

- 多关键字：例：扑克牌排序：
  - 4种花色：Spade,Heart,Club,Diamond
  - 13种数值：1~13

总结 I

- 平方阶的排序算法(直接插入、冒泡排序、选择排序)一般都容易实现，但时间复杂度相对较高。
- Shell排序是第一个次平方的排序算法。
- O(nlogn)类的算法（快速排序、归并排序和堆排序）相对有效，但实现略难。
- 若对数据集有一定先验知识，则可以考虑线性阶的排序算法（基数排序，计数排序等）。

总结II

- 在讨论的内排序方法中，不好说哪一个方法是最好的。一些方法对较小的 n 具有较好的性能，而另一些方法对较大的 n 性能较好。
- 选择排序算法时，先考虑时间限制。在满足时限的情况下，选择最容易实现的。
- 当输入记录是部分有序或n 值较小时，插入排序是较好的排序方法
。
- 快速排序具有最好的平均性能，但其不稳定且容易退化; 归并排序稳定，而且是外部排序的基础

排序问题持续研究

- **BogoSort**
  - ✓ 随机
- **SleepSort**
  - ✓ n个线程

思考题

要对某类商品进行排序，排序规则是：按商品的价格递增排序，且价格相同的商品销量高的排在前面，若不考虑效率，可行的排序方案是：

先按 A/B选一项 进行 C~J选一项或多项 ，再按 A/B选一项 进行 C~J选一项或多项

A. 价格递增　　B. 销量递减

C. 直接插入排序　D. 希尔排序　E. 冒泡排序

F. 直接选择排序　G. 堆排序　H. 快速排序

I.归并排序　J. C~I任一排序