



Ch 4 Sorting

排序(Sorting) 是对一个数据元素集合或序列
重新排列 成一个按数据元素某个项值有序的
序列。作为排序依据的数据项--关键字。



4.1 Intruduction

- 定义:对n个数据元素按照 key 递增/递减排列
- 基于数据元素之间的比较操作的排序算法:插入、冒泡、简单选择、堆、快速、希尔、归并等

此类排序算法需要两种基本操作:

- (1) 比较两个关键字的大小;
- (2) 将数据元素从一个位置移动至另一个位置。

- 分治策略: 归并和快速排序
- 基数排序----不进行数据元素之间的比较

由于通过比较操作的排序算法中, 而数据移动的时间在量级上不会超过比较次数, 所以本书大多数情况下时间复杂度只分析比较次数!



4.1 Intruduction

- 说明:

1. 只讨论内排序
2. 数组存放待排序数据 (许多策略也适合链表)
3. 时间、空间分析

- 重点:

1. 算法时间复杂度的分析----证明
2. 算法改进
3. 算法正确性证明----以冒泡排序为例

- 请务必自己认真补修各种排序算法的过程

4.2 插入排序

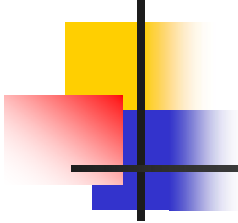


- **基本思想**：将待排序的数据元素按其关键字的大小依次插入到前面已排序的文件的适当位置上。
- **方法**：在插入第 i 个数据元素时， R_0, R_1, \dots, R_{i-1} 已经排序，用顺序查找方法找出应该插入的位置，将 R_i 插入。
- **53 27 36 15 69 42**
- **第一趟排序：27 53 36 15 69 42**
- **第二趟排序：27 36 53 15 69 42**
- **第三趟排序：15 27 36 53 69 42**
- **第四趟排序：15 27 36 53 69 42**
- **第五趟排序：15 27 36 42 53 69**

4.2 插入排序

- 算法：参看数据结构教材
- 一个额外的辅助空间, $O(1)$
- 最好情况：待排序的数据已经有序, $n-1$ 次数据比较, 0 次数据移动-- $O(n)$
- 最坏情况：例如, 待排序的数据逆序
 $W(n) = n(n-1)/2 \in \Theta(n^2)$

$$A(n) = \frac{(n+4)(n-1)}{4} - \sum_{j=2}^n \frac{1}{j} \leq \frac{(n+4)(n-1)}{4} \in \Theta(n^2)$$



Lower bounds on the behavior of certain sorting algorithm

■ Theorem 4.1

任何一个基于关键字的比较且每一次比较最多消除一个逆序的排序算法，**最坏**的情况下**至少**比较 $n(n-1)/2$ 次，**平均**情况**至少**比较 $n(n-1)/4$ 次。

Theorem 4.1 Any algorithm that sorts by comparison of keys and removes at most one inversion after each comparison must do at least $n(n-1)/2$ comparisons in the worst case and at least $n(n-1)/4$ comparisons on the average (for n elements). □

冒泡排序(习题4.2—4.5)

- 将待排序的数据元素的关键字顺次**两两比较**，若为**逆序**则将两个数据元素**交换**。
- 将待排序的数据元素照此方法从头到尾处理一遍称作一趟起泡排序，它将关键字值最大的数据元素交换到排序的最终位置。
- 若某一趟冒泡排序**没发生任何数据元素的交换**，则排序过程结束。
- 对含 n 个记录的文件排序最多需要 **$n-1$ 趟**冒泡排序。

void bubblesort(Element E[],int *n*)

{ int *numpairs*, *didSwitch*, *j*;

***numpairs*=*n*-1; *didSwitch*=1;**

while(*didSwitch*)

{ *didSwitch*=0;

for(*j*=0;*j*<*numpairs*; *j*++)

if(*E*[*j*]>*E*[*j*+1]) {*E*[*j*] \leftrightarrow *E*[*j*+1];

***didSwitch*=1; }**

***numpairs*--;**

}

}



习题4.2 冒泡算法的时间性能

- (a) $W(n)$, what arrangement of keys is a **worst** case?
- (b) How many comparisons does it do in the best case? what arrangement of keys is a **best** case?



习题4.3 冒泡算法的正确性证明

- (a) Prove that, after one pass through the array, the largest entry will be at the end.
- (b) Prove that, if there is no pair of consecutive entries out of order, then the entire array is sorted

证明(a)

```
void bubblesort(Element E[],int n)
{ int numpairs, didSwitch, j;
  numpairs=n-1; didSwitch=1;
  while(didSwitch)
  { didSwitch=0;
    for(j=0;j<numpairs; j++)
      if(E[j]>E[j+1]) {E[j]↔E[j+1];
                      didSwitch=1; }
    numpairs--;
  }
}
```

对j采用数学归纳法

- (1) 当j=0时, E[0]和E[1]比较。若E[0]>E[1], 则交换E[0]和E[1], 从而E[1]为E[0]和E[1]中最大的。若E[0]≤E[1], 则不交换, 从而E[1]为E[0]和E[1]中最大的。
- (2) 当j=1时, E[1]和E[2]比较。若E[1]>E[2], 则交换E[1]和E[2], 从而E[2]为E[1]和E[2]中最大的,也为E[0]、E[1]和E[2]中最大的。若E[1]≤E[2], 则不交换, 从而E[2]为E[1]和E[2]中最大的,也为E[0]、E[1]和E[2]中最大的。
- (3) 当j<k时, E[j+1]为E[0], E[1], ..., E[j+1]中最大的, 即E[k]为E[0], E[1], ..., E[k]中最大的。

当j=k时, E[k]和E[k+1]比较。若E[k]>E[k+1], 则交换E[k]和E[k+1], 从而E[k+1]为E[0], E[1], ..., E[k+1]中最大的。若E[k]≤E[k+1], 则不交换, 从而E[k+1]为E[0], E[1], ..., E[k+1]中最大的。

根据 (1)、(2)、(3)内循环对当前数组扫描一遍, 将最大元放在最后

(b) Prove that, if there is no pair of consecutive entries out of order, then the entire array is sorted

证明(b)

```
void bubblesort(Element E[],int n)
{ int numpairs, didSwitch, j;
  numpairs=n-1; didSwitch=1;
  while(didSwitch)
  { didSwitch=0;
    for(j=0;j<numpairs; j++)
      if(E[j]>E[j+1]) {E[j]↔E[j+1];
                      didSwitch=1; }
    numpairs--;
  }
}
```

- 设最后一次外循环时 $\text{numpairs}=k$, 此时 $\text{didSwitch}=\text{false}$, 即对所有的 $0 \leq j < k$, $E[j] \leq E[j+1]$
- 即: $E[0] \leq E[1] \leq \cdots \leq E[k]$
- 由(a)之前的外循环 $\text{numpairs}=n-1, n-2, \cdots, k+1$:
- 即: $E[n-1] \geq E[n-2] \geq \cdots \geq E[k]$
- 从而: $E[0] \leq E[1] \leq \cdots \leq E[n-1]$ 。



习题4.4 冒泡算法的改进

- We can modify the Bubble Sort to avoid unnecessary comparisons in the tail of the array by keeping track of where the last interchange occurred in the For loop
- Prove that, if the last exchange made in some pass occurs at j th and $(j+1)$ st positions, then all the entries from the $(j+1)$ st through the $(n-1)$ th are in their correct positions.



习题4.5 冒泡算法的改进

- Can something similar to the improvement in the preceding exercise (4.4) be done **to avoid unnecessary comparisons** when the keys at the beginning of the array are already in order? If so, write the modification to the algorithm. If not explain why not.