第 10 章

排序

孙未未
复旦大学

高等教育出版社

提纲

10.1 问题引入

排序是指将数据按照关键字重新排列为升序（从小到大）或降序（从大到小）的处理。

- 升序：关键字从小到大
- 降序：关键字从大到小

示例：

待排序列：	34	12	34'	08	96
升序：	08	12	34'	34	96
降序：	96	34	34'	12	08

排序的稳定性

若序列中关键字值相等的节点经过某种排序方法进行排序之后，仍能保持它们在排序前的相对顺序，则称这种排序方法是稳定的；否则，称这种排序方法是不稳定的。

稳定的排序算法有：

- 插入排序、冒泡排序、归并排序

不稳定的排序算法：

- 选择排序、快速排序、堆排序

示例：

待排序列：	34	12	34'	08	96
稳定：	08	12	34	34'	96
不稳定：	08	12	34'	34	96

排序的分类

- **内部排序、外部排序**（根据内存使用情况）
  - 内部排序：数据存储调整均在内存中进行
  - 外部排序：大部分节点在外存中，借助内存进行调整
- **比较、分配**（根据排序实现手段）
  - 基于“比较”的排序：通过对关键字的比较，交换关键字在序列中的位置
    - 插入排序、冒泡排序、选择排序、快速排序、归并排序、希尔排序、堆排序
  - 基于“分配”的排序：通过将元素进行分配和收集进行排序
    - 基数排序、桶排序
- 根据实现的**难易**程度：
  - 基本排序：插入排序、冒泡排序、选择排序、……
  - 高级排序：快速排序、归并排序、堆排序、基数排序、……

排序的应用例

- 最近数据对(Closest pair)
  - 给定n个数据的序列，求其中差值最小两个数据
  - 如果序列排好序，则差值最小的两个数据一定相邻！
- 数据的唯一性(Element uniqueness)
  - 给定n个数据的序列，所有数据互异还是有重复?
  - 对序列排序，然后从左向右依次检查所有相邻的数据对
- 统计各数据出现的次数(Frequency distribution – Mode)
  - 给定n个数据的序列，查找出现次数最多的数据
  - 序列排序后，所有相同的数据会连续排在一起！
- 中位数与选择(Median and Selection)
  - 求序列中第k小数据
  - 一旦序列按从小到大排好序，第k小的数据排在序列的第k个位置上

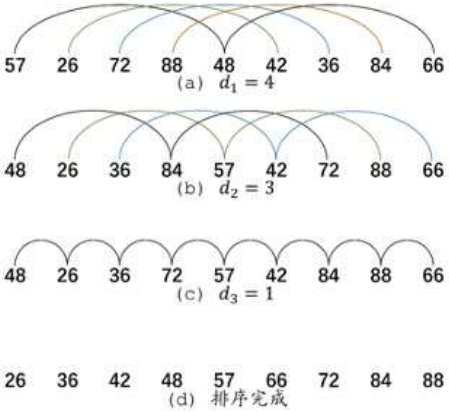
10.2 插入排序

基本思想:将一个记录插入到已排好顺序的序列中，形成一个新的、记录数增1的有序序列

折半插入排序和希尔排序是插入排序的两种改进

68	65	84	65	83	84	82	85	67	84	85	82	69
(65)	68	84	65	83	84	82	85	67	84	85	82	69
65	68	(84)	65	83	84	82	85	67	84	85	82	69
65	(65)	68	84	83	84	82	85	67	84	85	82	69
65	65	68	(83)	84	84	82	85	67	84	85	82	69
65	65	68	83	84	(84)	82	85	67	84	85	82	69
65	65	68	(82)	83	84	84	85	67	84	85	82	69
65	65	68	82	83	84	84	(85)	67	84	85	82	69
65	65	(67)	68	82	83	84	84	85	84	85	82	69
65	65	67	68	82	83	84	84	(84)	85	85	82	69
65	65	67	68	82	83	84	84	84	85	(85)	82	69
65	65	67	68	82	(82)	83	84	84	84	85	85	69
65	65	67	68	(69)	82	82	83	84	84	84	85	85

直接插入排序



希尔排序

10.2.1 直接插入排序

排序步骤：假设 a_0，a_1，…，a_{i-1} 已排序 $(a_0 < a_1 < \cdots < a_{i-1})$，对于$t = a_i$，将他与$a_{i-1}, a_{i-2}, \ldots, a_1$依次进行比较。

若 $a_j > t$，则将 a_j 向后移动一位。直到发现某个$j(1 \le j \le i-1)$；

若 $a_j \le t$，则令 $a_{j+1} = t$，此时完成了将 a_i 插入有序数组的过程；

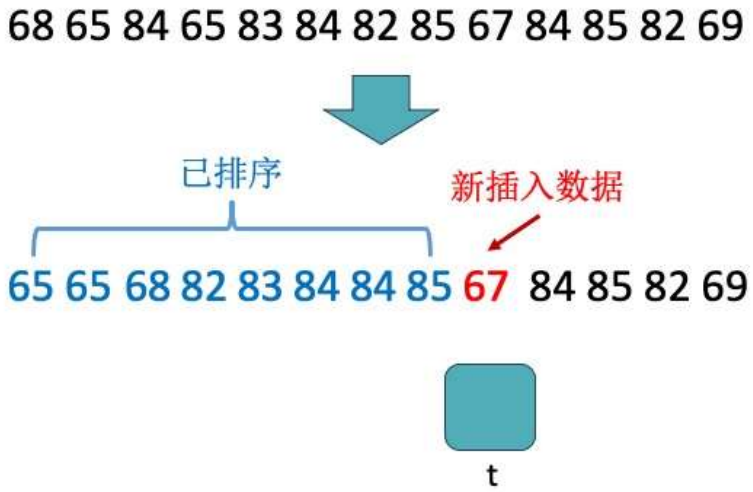
如果这样的a_j不存在，那么在比较过程中，$a_{i-1}, a_{i-2}, \ldots, a_1$都依次后移一个位置，此时令$a_1 = t$。

直接插入排序过程

对关键字序列{68,65,84,65,83,84,82,85,67,84,85,82,69}进行升序排列

68	65	84	65	83	84	82	85	67	84	85	82	69
(65)	68	84	65	83	84	82	85	67	84	85	82	69
65	68	(84)	65	83	84	82	85	67	84	85	82	69
65	(65)	68	84	83	84	82	85	67	84	85	82	69
65	65	68	(83)	84	84	82	85	67	84	85	82	69
65	65	68	83	84	(84)	82	85	67	84	85	82	69
65	65	68	(82)	83	84	84	85	67	84	85	82	69
65	65	68	82	83	84	84	(85)	67	84	85	82	69
65	65	(67)	68	82	83	84	84	85	84	85	82	69
65	65	67	68	82	83	84	84	(84)	85	85	82	69
65	65	67	68	82	83	84	84	84	85	(85)	82	69
65	65	67	68	82	(82)	83	84	84	84	85	85	69
65	65	67	68	(69)	82	82	83	84	84	84	85	85

直接插入排序过程


68 65 84 65 83 84 82 85 67 84 85 82 69
已排序
新插入数据
65 65 68 82 83 84 84 85 67 84 85 82 69
t

直接插入排序伪代码

算法10-1 插入排序 InsertionSort(a, l, r)

输入： 序列a，左端点下标l，右端点下标r

输出： 调整$a_l, a_{l+1}, \ldots, a_r$元素顺序，使元素按照非递减顺序排列

```
1   for i ← l + 1 to r do  //从左边界开始，依次获取每个记录
2   | t ← a_i
3   | for j ← i − 1 downto l do
4   | | if a_j > t then
5   | | | a_{j+1} ← a_j  //若当前记录小，则把前面的记录向后移一个位置
6   | | else
7   | | | a_{j+1} ← t  //将最初获取的记录复制到相应位置
8   | | | break
9   | | end
10  | end
11  | if a_l > t then  //如果a_l > t，将最初获取的记录置于序列开始
12  | | a_l ← t
13  | end
14  end
```

直接插入排序性能分析

时间复杂度：

最佳情况（有序）：$O(n)$

- $n-1$ 次比较，0 次移动.

最坏情况（逆序）：$O(n^2)$

- 插入第 i 个元素时，每次都需要比较前 $i-1$ 个元素才能找到插入位置。
- $\frac{n(n-1)}{2}$ 次比较，$\frac{(n+2)(n-1)}{2}$ 次移动.

平均情况：$O(n^2)$

- 比较次数和记录移动次数约为$\frac{n^2}{4}$
- $O(n^2)$ 次比较，$O(n^2)$ 次移动

直接插入排序性能分析

空间复杂度：$O(1)$

对于每次排序，仅申请一个临时变量用于存储第i个元素，空间复杂度为$O(1)$

稳定性分析：**稳定**

对于值相同的元素，可以选择将后面出现的元素，插入到前面出现元素的后面，这样就可以保持原有的前后顺序不变，所以插入排序是**稳定的**排序算法。

适用场景：节点个数少。

10.2.2 折半插入排序

折半插入排序是对直接插入排序的一种改进。当待排序序列较长时，如果采用折半查找的方法，可以更快地寻找插入位置，减少关键码的比较次数。

折半插入排序虽然可以减少关键码的比较次数，但是并不减少排序元素的移动次数。比较次数为$O(nlogn)$，移动次数为$O(n^2)$

10.2.3 希尔排序

基本思想：相较于插入排序，对位置相隔较大距离的元素进行比较，使得元素在比较后能够一次跨过较大的距离。

排序步骤：

- 先给定一组严格递减的正整数增量$d_0, d_1, \ldots, d_{t-1}$，且取$d_{t-1} = 1$。
- 对于$i=0,1,\ldots,t-1$，进行下面各遍的处理：
  - 将序列分成d_i组，每组中结点的下标相差d_i
  - 对每组节点使用插入排序

希尔排序伪代码

算法10-2 希尔排序 ShellSort(a, l, r)

输入： 序列a，左端点下标l，右端点下标r

输出： 调整$a_l, a_{l+1}, \ldots, a_r$元素顺序，使元素按照非递减顺序排列

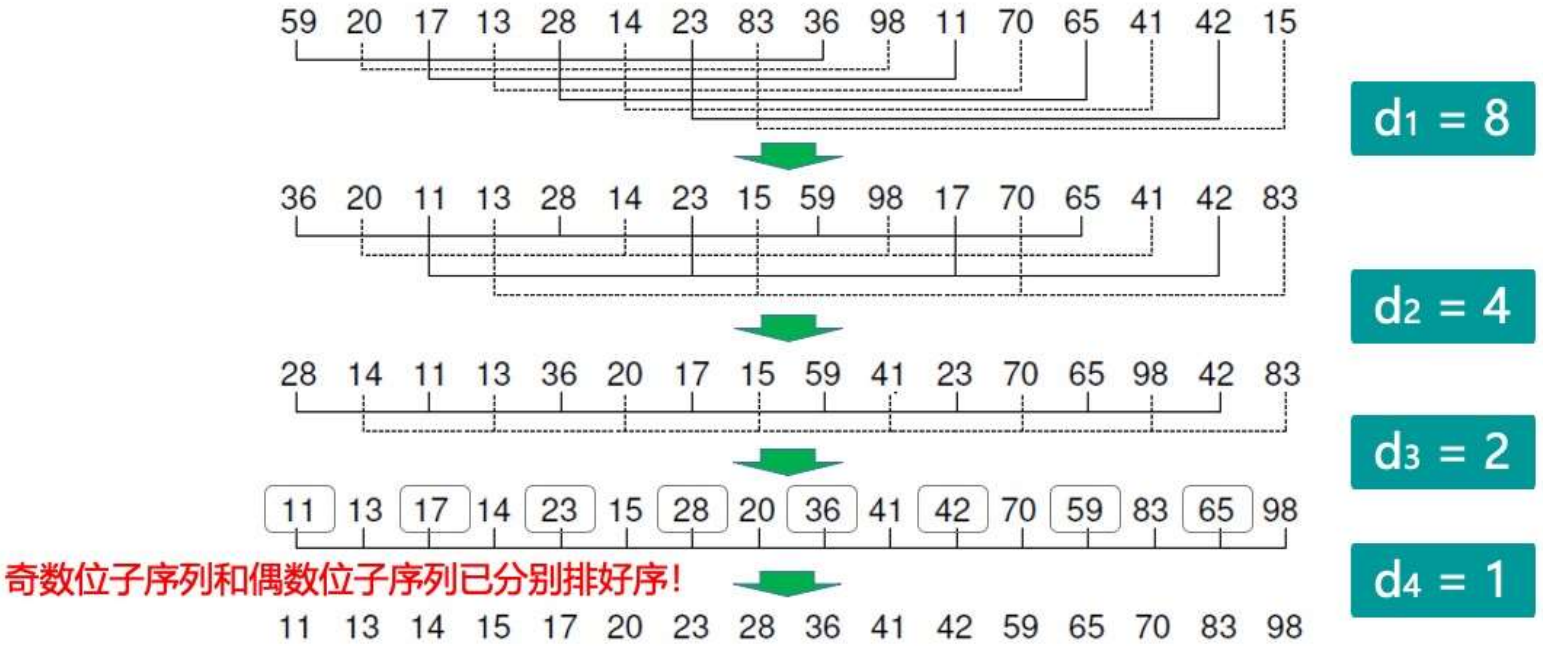
```
1   d ← {d_1, d_2, ..., d_t}  //希尔排序时使用的步长，根据不同步长循环进行插入排序
2   for m ← 1 to |d| do
3   |  for i ← l + d_m to r do
4   |  |  v ← a_i
5   |  |  for j ← i downto l + d_m step d_m do  //对序列a_j, a_{j-d_m}, ...进行插入排序
6   |  |  |  if a_{j-d_m} > v then
7   |  |  |  |  a_j ← a_{j-d_m}
8   |  |  |  else
9   |  |  |  |  a_j ← v
10  |  |  |  |  break
12  |  |  |  end
13  |  |  end
14  |  end
15  end
```

希尔排序的历史

- 希尔排序最早出现在设计者希尔（Donald Shell）在 1959 年所发表的论文“A high-speed sorting procedure”中。
- 1961年，IBM 公司的女程序员 Marlene Metzner Norton（玛琳·梅茨纳·诺顿）首次使用 FORTRAN 语言编程实现了希尔排序算法。
- 在其程序中使用了一种简易有效的方法设置希尔排序所需的增量序列：第一个增量取待排序记录个数的一半，然后逐次减半，最后一个增量为 1。该算法后来被称为 Shell-Metzner 算法。
- 但Metzner 本人在2003年的一封电子邮件中说道：“我没有为这种算法做任何事，我的名字不应该出现在算法的名字中”。

希尔排序过程: Shell-Metzner 算法

- 对长度为16的关键字序列进行升序排列，设置增量序列为$d = \{8,4,2,1\}$



单选题 1分

对序列$a_1, a_2, \ldots, a_n$使用插入排序法按升序（非降序）排序，其中$a_1 \le a_3 \le \cdots \le a_{2k+1} \le \cdots$并且$a_2 \le a_4 \le \cdots \le a_{2k} \le \cdots$成立，则在最坏情况下，排序的时间复杂度是

A O(n)

B $O(n\log(n))$

C $O(n\sqrt{n})$

D $O(n^2)$

希尔排序性能分析

时间复杂度：

依赖于增量序列，没有确切结论

步长序列	最坏情况下复杂度
$n / 2^i$	(n^2)
$2^k - 1$	$(n^{3/2})$
$2^i 3^i$	$(n\log^2 n)$

- Shell-Metzner 算法
- Hibbard间隔

- 作为最早突破 $O(n^2)$复杂度的排序算法之一，希尔排序一直是闪耀着编程艺术之美的存在。

---《计算机程序设计的艺术》The Art of Computer Programming， Donald E. Knuth著

*希尔排序性能分析

- **希尔排序的基本原理：**

对数组$a[1,\dots,n]$希尔排序：
(1) 先用较大的间距(increment) h 插入排序，使
$\forall i \in [1,n-h]: a[i] < a[i+h]$;
(2) 再用较小的间距 l 插入排序 ($l < h$), 同样可得
$\forall j \in [1,n-l]: a[j] < a[j+l]$。

可以证明第（2）次排序后，$\forall i \in [1,n-h]: a[i] < a[i+h]$ 依然成立!

◆ 完整的证明可见《The Art of Computer Programming》， Donald E. Knuth著

- **原理证明所需的基本性质（一道有关序列的趣题！）：**

序列X $=< x_1, x_2, \dots x_i, \dots, x_n >$ —排序→ X′ $=< x'_1, x'_2, \dots x'_i, \dots, x'_n >$

序列 Y $=< y_1, y_2, \dots y_i, \dots, y_n >$ —排序→ Y′ $=< y'_1, y'_2, \dots y'_i, \dots, y'_n >$

希尔排序性能分析

时间复杂度：

依赖于增量序列，没有确切结论

步长序列	最坏情况下复杂度
$n / 2^i$	(n^2)
$2^k - 1$	$(n^{3/2})$
$2^i 3^i$	$(n\log^2 n)$

- Shell-Metzner 算法
- Hibbard间隔

- 作为最早突破 $O(n^2)$复杂度的排序算法之一，希尔排序一直是闪耀着编程艺术之美的存在。

---《计算机程序设计的艺术》The Art of Computer Programming， Donald E. Knuth著

空间复杂度： $O(1)$

稳定性分析： **不稳定 (?)**

10.3 选择排序

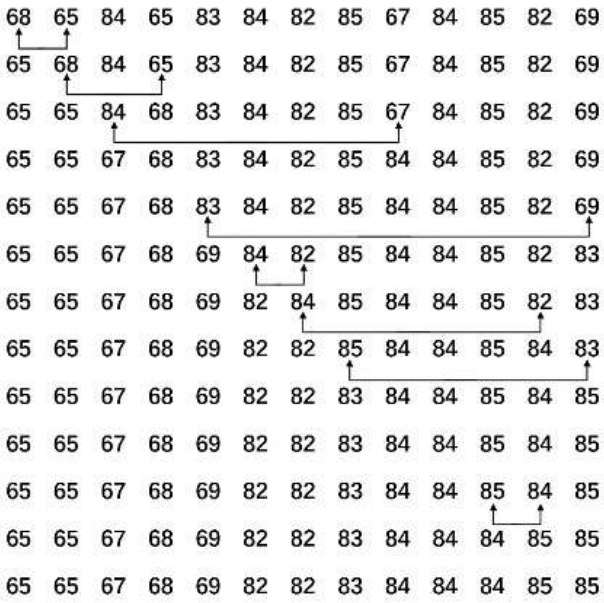
基本思想：首先选出键值**最小**的项，将它与**第一个项**交换位置；然后选出键值**次小**的项，将其与**第二个项**交换位置；...；直到整个序列完成排序。

步骤：

- 假设待排序的序列为$a_0, a_1, a_2, \cdots, a_{n-1}$
- 依次对$i = 0, 1, \cdots, n-2$执行如下步骤：
  - 在$a_i, a_{i+1}, \cdots, a_{n-1}$中选择一个键值最小的项$a_k$
  - 将a_i与a_k交换。

10.3.1 简单选择排序

简单选择排序在从待排序序列中选出键值最小的项时，所用的策略是简单的逐个枚举法。



简单选择排序过程

简单选择排序伪代码

算法10-3 简单选择排序 $SelectionSort(a,l,r)$

输入：序列a，左端点下标l，右端点下标r

输出：调整$a_l, a_{l+1}, \ldots, a_r$元素顺序，使元素按照非递减顺序排列

```
1     for i ← l to r − 1 do //依次从剩余未排序序列中选取一个最小的记录
2     |  min ← i
3     |  for j ← i + 1 to r do
4     |  |  if a_j < a_min then
5     |  |  |  min ← j
6     |  |  end
7     |  end
8     |  t ← a_i //将当前的最小记录放入已排序好的队列的末尾
9     |  a_i ← a_min
10    |  a_min ← t
10    end
```

简单选择排序性能分析

时间复杂度：

最佳情况（有序）：$O(n^2)$

- $\frac{n(n-1)}{2}$ 次比较，0 次移动.

最坏情况（逆序）：$O(n^2)$

- $\frac{n(n-1)}{2}$ 次比较，$n-1$ 次移动.

平均情况：$O(n^2)$

- $O(n^2)$ 次比较，$O(n)$ 次移动

简单选择排序性能分析

空间复杂度：$O(1)$

排序时仅交换未排序序列中最值与未排序序列起始位置，不涉及额外空间的使用。

稳定性分析：不稳定

交换操作时可能会破坏具有相同key值元素的相对位置。因此选择排序为不稳定排序。

运行时间与记录顺序关系很小

交换次数很少