第2章 算法入门

2.1 插入排序

第一个算法是插入排序算法。

输入：n个数$(a_1, a_2, ..., a_n)$。

输出：输入序列的一个排列（即重新排序）$(a'_1, a'_2, ..., a'_n)$，使得$a_1 \leq a'_2 \leq ... \leq a'_n$。

待排序的数也称为关键字(key)。

代码见：Insertion-sort

循环不变式与插入算法的正确性

循环不变式主要用来帮助我们理解算法的正确性。对于循环不变式，必须证明它的三个性质：

初始化：它在循环的第一轮迭代开始之前，应该是正确的。

保持：如果在循环的某一次迭代开始之前它是正确的，那么，下一次迭代开始之前她也应该保持正确。

终止：当循环结束时，不变式给了我们一个有用的性质，它有助于表明算法是正确的。

当头两个性质成立时，就能保证循环不变式在循环的每一轮迭代开始之前，都是正确的。

证明：

初始化：首先，先证明在第一轮迭代开始之前，循环不变式是成立的。此时j=2，而子数组为A[0..j-1]。亦即，它至包含一个元素A[0]，实际上就是最初在A[0]中的那个元素。显然这个子数组是已排序的，这样就证明了循环不变式的第一轮迭代开始之前是成立的。

保持：非形式化证明即，在外层for循环中，要将A[j-1]，A[j-2]，A[j-3]等元素向右移动一个位置，知道找到A[j]的位置为止，此时将A[j]插入。

终止：对于插入排序，当j>=n时，外层for循环技术。在循环不变式中，将j替换为n，就有子数组A[0..n-1]，包含了A[0..N-1]中元素，且已排好序。同时，子数组就是整个数组，因此，整个数组排好序了，这意味着算法是正确的。

练习

2.1-1 以图2-2为模型，说明INSERTION-SORT在数组A=<31,41,59,26,41,58>上的执行过程。

2.1-2 重写过程INSERTION-SORT，使之按非升序（而不是按降序）排列。

Insertion-sort2.cpp

2.1-3 考虑下面的查找问题：

输入：一列数$A = < a_1, a_2, ..., a_n >$和一个值v。

输出：下标i，使得v=A[i]，或者当v不在A中出现时为NIL。

写出针对这个问题的线性查找的伪代码，它顺序地扫描整个序列以查找v。利用循环不变式证明算法的正确性。确保所给出的循环不变式满足三个必要的性质。

Find_value_in_array

2.1-4 有两个各存放在数组A和B中的n位二进制整数，考虑它们的相加问题。两个整数的和以二进制形式存放在具有(n+1)个元素的数组C中。清给出这个问题的形式化描述，并写出伪代码。

BinarySum

2.2 算法分析

算法分析即指对一个算法所需要的资源进行预测。通常我们希望预测的是计算时间。书中采用一种通用的但处理器、随机存取机（RAM）计算模型，在RAM模型中，指令一条接一条执行，没有并发操作。

插入排序算法的分析

输入规模的概念与具体问题有关，对许多问题而言，最自然的度量标准是输入中元素个数。
算法的**运行时间**是指在特定输入时，所执行的基本操作数（或步数）。

最坏情况和平均情况分析

一般考察算法的**最坏情况运行时间**，亦即，对于任何规模为n的任何输入，算法的最长运行时间。原因如下：

1. 一个算法的最坏情况时在任何输入下运行时间的上限。
2. 对于某些算法来说（例如：在数据库检索信息而信息并不在数据库中），最坏情况出现得相当频繁。
3. 大致上看，“平均情况”同拆毁嗯与最坏情况一样差。

在某些情况喜爱，我们会对算法的**平均情况**或**期望**的运行时间感兴趣，第5章会介绍**概率分析**技术来确定一个算法期望的运行时间。采用**随机化算法**可以做出随机选择，从而可以对算法进行概率分析。

增长的量级

运行时间的**增长率**（**增长的量级**），只需要考虑攻势中的最高次项且忽略其系数。例如插入排序的最坏情况时间代价为$\Theta(n^2)$

练习

2.2-1 用Θ形式表示函数$n^3/1000 - 100n^2 - 100n + 3$

$\Theta(n)$

2.2-2 考虑对数组A中的n个数进行排序的问题：首先找出A中的最小元素，并将其与A[1]中的元素进行交换。接着，找出A中的次小小于，并将其与A[2]中的元素进行交换。对A中头n-1个元素继续这一过程。写出这个算法的伪代码，该算法称为**选择排序**。对这个算法来说，循环不变式是什么？为什么它仅需要在头n-1个元素上运行，而不是在所有n个元素上运行？以Θ形式写出选择排序的最佳和最坏情况下的运行时间

Select-Sort

循环不变式是选择的for循环部分。因为每次是前后的元素比较，第n-1个元素与第n个比较。

最佳情况：$\Theta(1)$，最坏情况$\Theta(n^2)$

2.2-3 再次考虑线性查找问题（见习题2.1-3）。在平均情况下需要检查输入序列的多少个元素？假定待查找的元素是数组中任何一个元素的可能性是相等的。在最坏情况喜爱又怎样呢？用Θ表示的画吗线性查找的平均情况和最坏情况怎样？对你的答案加以说明。

平均情况下查找$[\frac{n}{2}]$。最坏的情况下查找n个元素。平均与最坏都是$\Theta(n)$

2.2-4 应如何修改一个算法，才能使之具有良好的最佳情况运行时间?

尽量少用循环

2.3 算法设计

算法设计有很多方法。插入排序使用的是**增量**方法：在排好子数组A[1..j-1]后，将元素A[j]插入，形成排好序的子数组A[1..j]。
本节介绍分治法。

2.3.1 分治法

有很多算法在结构上是递归的：为了解决一个给定的问题，算法要一次或多次地递归调用其自身来解决相关的子问题。这些算法常采用分治策略：将原问题划分为n个规模较小而结构与原问题相似的子问题；递归地解决这些子问题，然后再合并其结果，就得到原问题的解。

分治模式再每一层递归上都有三个步骤：

分解：将原问题分解成一系列子问题；

解决：递归地解各子问题。若子问题足够小，则直接求解；

合并：将子问题的结果合并成原问题的解。

合并排序操作模式如下：

分解：将n个元素分成各含n/2个元素的子序列；

解决：用合并排序法对两个子序列递归地排序；

合并：合并两个已排好序的子序列以得到排序结果。

Divide-Sort

2.3.2 分治法分析

递归可以用递归方程来表示。设$T(n)$为一个规模为n的问题的运行时间，如果问题的规模足够地小，如$n \le c$，(c为常量)，则得到它的直接解的时间为常量，写作$\Theta(1)$。假设把原问题分解成a个子问题，每一个的大小是原问题的$1/b$，如果分解改问题的时间分别为$D(n)$和$C(n)$，则得到递归式：

$$T(n) = \begin{cases} \Theta(1), if\, n \le c \\ aT(n/b) + D(n) + C(n) \end{cases}$$

练习

2.3-1 以图2.4为模型，说明合并排序在输入数组$A = < 3, 41, 52, 26, 38, 57, 9, 49 >$上的执行过程

初始序：$3, 41, 52, 26, 38, 57, 9, 49$
合并一次：$< 3, 41 >$, $< 52, 26 >$, $< 38, 57 >$, $< 9, 49 >$
合并两次：$< 3, 26, 41, 52 >$, $< 9, 38, 49, 57 >$
合并三次：$< 3, 9, 26, 38, 41, 49, 52, 57 >$

2.3-2 改写MERGE过程，使之不使用哨兵元素，而是在一旦数组L或R中的所有元素都被复制回数组A后，就立即停止，再将另一个数组中余下的元素复制回数组A中。

2.3-3 利用数学归纳法证明：当n是2的整数次幂时，递归式

$$T(n)=\begin{cases}\Theta(1), if\, n=2\\2T(n/2)+n, if\, n=2^k,\, k>1\end{cases}$$

的解为$T(n)=nlgn$

证明：$n=2$时，$T(1)=c$

$$if\, T(n)=\begin{cases}\Theta(1), if\, n=2\\2T(n/2)+n, if\, n=2^k,\, k>1\end{cases}$$

$n=2^k$

$T(n)=2T(n/2)+cn=4T(n/4)+2cn=\cdots=2^kT(1)+(k-1)cn=2^kc+2^k(k-1)c=2^kkc$

$nlgn=2^kkc$

证毕

2.3-4 插入排序可以如下写成一个递归过程：为排序A[1..n]，先递归地排序A[1..n-1]，然后再将A[n]插入到已排序的数组A[1..n-1]中去。对于插入排序的这一递归版本，为它的运行时间写一个递归式。

$$T(n)=n-1$$

即

$$T(n)=\Theta(n)$$

2.3-5 写出二分查找法的伪代码，可以是迭代的，也可以是递归的。说明二分查找算法的最坏情况为什么是$\Theta(logn)$

BinarySearch

最坏的情况下要进行logn次循环

2.3-6 2.1节中INSERTION-SORT过程，在5～7行的while循环中没采用了一种线性查找策略，在已排序的子数组$A[1\dots j-1]$中（反向）扫描，是否可以改用二分查找策略，来将插入排序的总体最坏情况运行时间改善至$\Theta(nlogn)$

不能

2.3-7 请给出一个运行时间为$\Theta(nlogn)$的算法，使之能在给定一个由n个整数构成的集合S和另一个整数x时，判断出S中是否存在两个其和为x的元素。

Sum